GLOBAL OPTIMIZATION OF COMPUTATIONALLY EXPENSIVE BLACKBOX PROBLEMS USING RADIAL BASIS FUNCTIONS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Tipaluck Krityakierne

August 2014

© 2014 Tipaluck Krityakierne
ALL RIGHTS RESERVED

GLOBAL OPTIMIZATION OF COMPUTATIONALLY EXPENSIVE BLACKBOX PROBLEMS USING RADIAL BASIS FUNCTIONS

Tipaluck Krityakierne, Ph.D.

Cornell University 2014

Three derivative-free global optimization methods are developed based on radial basis functions (RBFs) for computationally expensive blackbox simulation models. First, we develop a multistart global optimization method, called SOMS (SurrOgate MultiStart). SOMS uses an RBF surrogate model to approximate the objective function in order to reduce the number of function evaluations necessary to identify the most promising points from which each nonlinear programming local search is started. We show that SOMS detects any local minimum within a finite number of iterations almost surely. The numerical results show that SOMS performs favorably in comparison to alternative methods and that the surrogate approach saves a significant number of computationally expensive function evaluations.

In the second part of this work, we introduce PADS (PArallel Dynamic coordinate search with Surrogates), which is a surrogate-based global optimization framework for high-dimensional expensive blackbox functions. In each parallel iteration of PADS, multiple points are selected from a large set of candidate points that are generated by perturbing only a subset of the coordinates of the current best solution. The selected points are then evaluated in parallel with up to 16 parallel processors. We show that PADS converges to the global optimum with probability 1. We develop two versions, PADS1 and PADS2, which use different underlying distributions to generate candidate points. We show that PADS1 and PADS2 are able to find better solutions more efficiently compared to alternative methods, with PADS1 performing even better than PADS2 in problems up to 200 dimensions.

In the final part of this dissertation, we develop an effective new parallel surrogate global optimization method called SOP (Surrogate Optimization with Pareto center selection). The

search mechanism of SOP incorporates bi-objective optimization, tabu search, and surrogate assisted local search, which exploits the information from the already evaluated points, for selecting a large number of new evaluation points. The newly selected points are evaluated in parallel, and hence a significant reduction in wall-clock time can be achieved. We give sufficient conditions for almost sure convergence of SOP. The results of our numerical experiments show that SOP performs very well compared to alternative parallel surrogate model algorithms with 8 and 32 processors obtaining superlinear speedup on some test problems.

BIOGRAPHICAL SKETCH

Tipaluck Krityakierne was born as a second daughter to a Thai father and a Singaporean mother. Tipaluck spent several years of her childhood in Tokyo before moving back to Bangkok, Thailand, when she was about five years old. She grew up in a multicultural and multilingual environment.

From pre-school to high school, she attended Satit Prasarnmit Demonstration School. Funded by the Office of the Higher Education Commission of Thailand, she majored in Mathematics at Chulalongkorn University. After graduating as the top ranked student in her program, she received two prestigious fellowships: the Anandamahidol Foundation Scholarship under the patronage of H.M. the King of Thailand and the Fulbright fellowship. These two fellowships gave her an opportunity to come to the US in 2007 to pursue her Ph.D. in Applied Mathematics at Cornell University.

Tipaluck received her Master of Science degree in Applied Mathematics in 2011 and received her Ph.D. degree in August 2014. She will be joining the Institute of Mathematical Statistics and Actuarial Science at University of Bern, Switzerland in September 2014 as a Swiss Government Excellence Postdoctoral Scholar.

To my mom, dad, sister, and brother

ACKNOWLEDGEMENTS

This dissertation is the end of a long journey that began 8544.73 miles away from home. It was made possible through the support of many people, to whom I now have the pleasure of expressing my appreciation and gratitude.

First and foremost, I would like to express my deepest gratitude to my advisor Prof. Christine Shoemaker. Her support and encouraging guidance have been instrumental in shaping this work and helping me reach my destination. I would also like to thank Prof. Peter Frazier and Prof. Michael Nussbaum, who served on my dissertation committee, for their help over the years. I offer my thanks, as well, to Prof. Tapan Mitra for introducing me to the world of mathematical economics. Although this part of my work was not incorporated into this dissertation, it has been a tremendous learning experience for me.

I am grateful to my fellow students at Center for Applied Mathematics for their good humor and good conversation, all of which have made our office a pleasant place to work. I especially thank Adam Chacon for helping with parallel programming, as well as for sharing the difficult and the good moments during this journey with me. To Huimei Delgado, I thank her for all the support and friendship she has given me. I will always remember the endless hours we spent doing homework assignments and debugging code together. I thank Mathav Murugan for his useful suggestions and for checking many of my mathematical proofs. I am also grateful to Yilun Wang, who gave me programming advice during the initial stage of this work. I owe special thanks to Juliane Müller for her valuable suggestions and comments, and her encouragement and willingness to proofread countless pages of this dissertation.

I thank Prapanpong Pongsriiam and Santi Tasena, my academic brothers, who took their time and patience to help me understand many mathematics materials. Their explanations in Thai made it much easier to understand difficult mathematical concepts. I am grateful, as well, to my former college professors: Prof. Patanee Udomkavanich, Prof. Wicharn Lewkeeratiyutkul, and Prof. Kittipat Wong for their helpful advice and being supportive for the past ten years.

For all their love and support, I offer special thanks to my extended family and friends in Thailand, Singapore, Japan, and here in the US. While there are too many to name, I would especially like to thank all my Thai friends at Cornell who have shared part of this journey with me. I have also enjoyed the warm friendship of my Japanese teacher, Naomi Larson, and her family, Kent and Komi. Thank you for always providing the support that I needed.

I would like to thank my dance instructors, Byron Suber and Jumay Chu, for providing enjoyable and challenging classes and unforgettable performance opportunities. Whenever I walk into the ballet studio, I feel the homey atmosphere that releases me from the pressures of research work. All my dance friends at the Department of Performing & Media Arts also play important part of my life at Cornell. This journey would have been all the more difficult were it not for their warm and loving friendship over the years.

Sahoko Ichikawa, thank you for being the best roommate anyone could ask for. I am very grateful for your helping hand, for the healthy and delicious food you cooked me, and for all you have done to support me during all these years. My sincerest thanks to otōsan and okāsan, Tadao and Yoriko Ichikawa, who have been very supportive and caring. Thank you for being my surrogate family in Japan, and for walking through this journey with me. Itu mo arigatō gozaimasu.

Most importantly, I would like to express my deepest appreciation to my Krityakierne family in Thailand: my mother, Nancy, my father, Varaj, my sister, Tirolarn, and my beloved brother, Kousak, for their unconditional love and care, as well as their boundless support in every respect. I appreciate more than I can say all that they have done to support me throughout the course of this long journey.

Finally, I am exceptionally grateful for support from Anandamahidol Foundation Scholarship under the patronage of H.M. the King of Thailand and the Fulbright Fellowship. I am thankful for various teaching assistantships with Cornell's Center for Applied Mathematics and research assistantships with the School of Civil and Environmental Engineering through

grants from NSF to C. Shoemaker CISE-1116298 and from DOE SciDAC to N. Mahowald and C. Shoemaker DE-SC0006791.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

In this dissertation, we consider a real-valued global optimization problem of the form:

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

where $\mathcal{D} = \{\text{lb} \leq x \leq \text{ub}\} \subset \mathbb{R}^d$. Here, lb and ub are the lower and upper variable bounds, respectively, and $f(x)$ is a continuous objective function with the following characteristics:

1. blackbox;
2. computationally expensive;
3. non-differentiable or its gradient is computationally intractable;
4. multimodal.

While these three properties are very common in real-world engineering design problems, conventional optimization methods such as SQP [79], for example, are often not applicable because of the following reasons. For complex simulation models, obtaining accurate derivatives can be infeasible or computationally very expensive. Due to this lack of gradient information, gradient-based optimization algorithms such as the steepest descent method or the conjugate gradient method [11, 23] cannot be used. In addition, heuristic methods

[52, 82] such as simulated annealing, genetic algorithms or tabu search generally require too many expensive function evaluations to converge and are therefore not efficient.

As a result, in recent years, an increasing number of algorithms that incorporate surrogate models (also known as response surfaces or metamodels) have been proposed to efficiently solve the class of optimization problems that have objective functions with the characteristics described above. In surrogate model based optimization, the computationally expensive objective function is approximated with an inexpensive surface. An auxiliary optimization problem on this surrogate surface is solved in each iteration to determine the next point at which the true objective function is evaluated. The new data is used to update the surrogate surface, and thus it is iteratively refined. Several popular response surface models such as radial basis functions (RBFs), kriging, polynomials, support vector regression, and multivariate adaptive regression splines (MARS) have been used in optimization [25, 40, 41, 42, 56, 57, 61, 68, 78, 109, 122]. A detailed review of surrogates used in engineering design optimization can be found in [122]. In this dissertation we use RBFs because these models have been shown to generally perform better than the alternative models [77].

An example of an RBF surrogate is given in Figure 1.1. Five points at which we know the objective function value (marked by $\circ$) are used to fit the surrogate. The dotted line is the constructed RBF surrogate. The solid line is the true objective function, which is in practice unknown.

Most surrogate-based global optimization algorithms follow the framework given in Figure 1.2. The algorithm starts by creating an initial experimental design at which the costly objective function is evaluated. In each iteration of the algorithm the surrogate model is fit to the data. The surrogate is then used (within the algorithm's specific optimization strategy) to select x_k for the next expensive function evaluation, and if the stopping criteria are not met, the algorithm starts the next iteration by updating the surrogate.

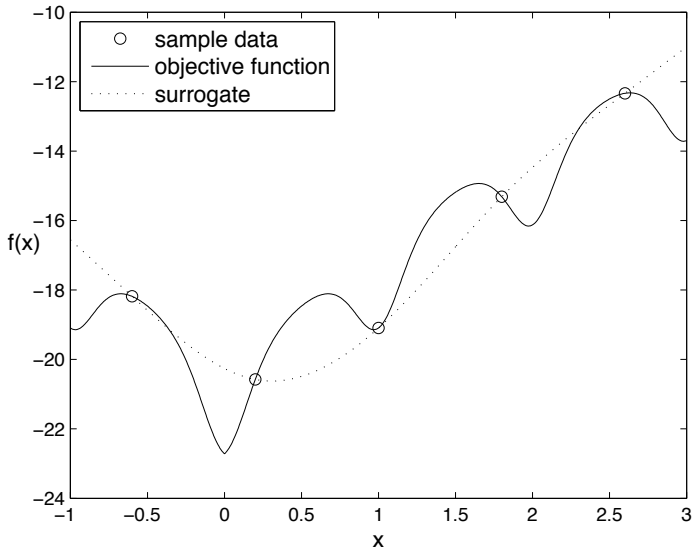


Figure 1.1: Example of an RBF surrogate model fit to sample data

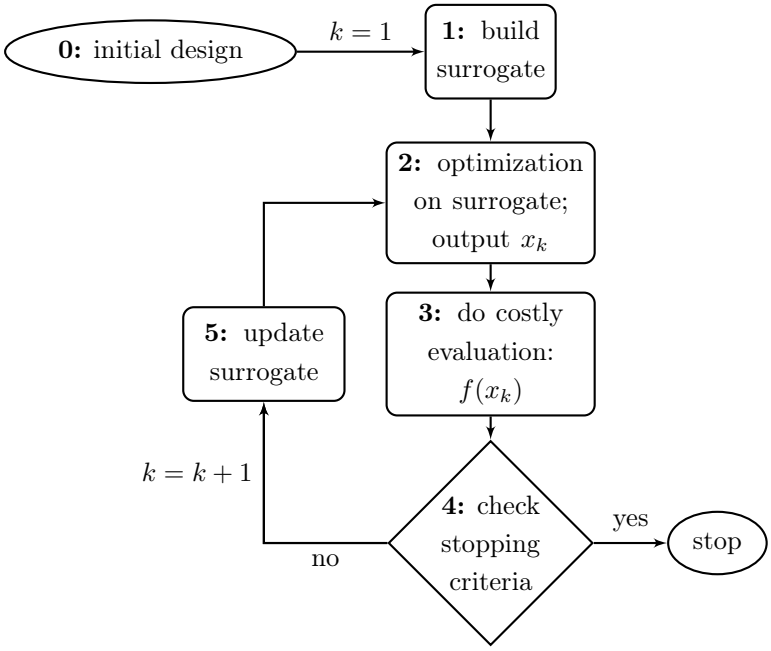


Figure 1.2: Surrogate-based global optimization framework

The aims of this dissertation are to design and implement RBF based global optimization algorithms that are able to achieve a fast decrease in function value with respect to the number of function evaluations (for serial algorithms) or the wall-clock time (for parallel algorithms). The questions investigated and the contributions of this dissertation can be summarized as follows.

1. We investigate the use of RBFs within a multistart local search optimization framework.

2. We investigate the performance of different surrogate-based (both serial and parallel) global optimization algorithms for high dimensional problems (up to 200 dimensions).

3. We investigate the performance of different RBF based algorithms in a parallel optimization framework when a large number of function evaluations are done simultaneously.

To address the first aspect, in Chapter 2, we propose SOMS: **S**urr**O**gate **M**ulti**S**tart algorithm for global optimization. SOMS is an efficient surrogate-based multistart algorithm, which is used in combination with a nonlinear local optimization routine to find the global minimum of the computationally expensive multimodal objective function. In practice, local optimization methods are often applied to complex nonlinear optimization problems. However, local optimization algorithms generally stop at a local minimum, and hence the global minimum may be missed. While methods based on multistart can also be helpful for computationally expensive functions when identifying other good local minima (besides the global minimum) is useful, existing multistart algorithms are inefficient because a large number of sample points have to be evaluated on the expensive function before a starting point for a local search can be located. This motivates the investigation of the use of response surface models within the multistart framework to help reduce the number of function evaluations necessary to identify the most promising points from which to start each nonlinear local solver. SOMS's numerical results are compared with four well-known methods, namely, Multi Level Single Linkage (MLSL) [59, 60], MATLAB's MultiStart, MATLAB's

GlobalSearch [119], and GLOBAL [30]. The numerical results indicate that SOMS performs favorably in comparison to alternative methods. Theoretical properties of SOMS similar to those possessed by MLSL are also verified.

Because searching in high-dimensional spaces (with several hundred dimensions) unavoidably requires a large number of function evaluations ("curse of dimensionality"), solving a class of HEB (High-dimensional, Expensive, and Blackbox) [106] problem with a serial algorithm can be extremely time-consuming (due to both the computational expense of a single function evaluation and the optimization algorithm's own computational overhead). In Chapter 3, we therefore develop and implement PADS (**PA**rallel **D**ynamic coordinate search with **S**urrogates), which is a surrogate-based global optimization framework for high-dimensional expensive blackbox functions. PADS selects the next evaluation point from a set of candidate points that are obtained by perturbing only a subset of the coordinates of the current best solution, which is the key idea proposed in DDS [115] and was recently empirically proven to be very effective especially when the problem dimension is very high (200 variables) [97]. In PADS, multiple expensive function evaluation points are evaluated in each iteration, i.e. the output x_k in the framework shown in Figure 1.2 is replaced by $\left\{x_1^{(k)}, ..., x_J^{(k)}\right\}$, a set of J selected points for simultaneously evaluating the objective function.

In addition, different underlying distributions to generate candidate points for next function evaluation points within Step 2 of Figure 1.2 will also be investigated. A practical implementation that follows the PADS framework is described and several numerical results are illustrated and compared to alternative methods including DYCORS [97], MADS [1, 2, 9, 65, 67], Differential Evolution [113], ParESGRBF [91, 107], ParLMSRBF [95, 96], as well as TOMLAB's rbfSolve [15, 50]. The results demonstrate that PADS makes fast progress towards the best possible solution given a limited computational budget and is well suited for very high-dimensional problem. Finally, we also provide convergence conditions and show that PADS converges to the global optimum with probability 1.

When simulations are computationally expensive, one needs to terminate the algorithm after a certain amount of wall-clock time, e.g. 100 hours. Therefore, for serial algorithms (one function evaluation is done in each iteration), the algorithm might be able to do only few hundred evaluations before termination. This inevitably affects the quality of the final solution. Parallel computing technologies make it possible to solve computationally expensive global optimization problems that cannot be solved otherwise by a serial algorithm. With more points simulated per iteration, one would expect that the algorithm converges within fewer iterations. This is, however, not always the case. The challenge, therefore, becomes how to effectively select many points for simultaneous evaluations.

In Chapter 4, we propose a new method, SOP (**S**urrogate **O**ptimization **P**areto center selection). The search mechanism of the algorithm incorporates bi-objective search, tabu search, and surrogate assisted local search. SOP evaluates multiple points and uses the data from all of these evaluations to update the surrogate model in each iteration. Thus, the number of iterations, and hence the wall-clock time, is reduced as opposed to the total number of simulations. SOP differs from the previous method of Regis and Shoemaker [95, 96, 97] in that the search center used to generate candidate points for the next function evaluation points is not always the best point. This contrasts also with the approach used in PADS. To manage the trade-off between exploration and exploitation, bi-objective optimization over a finite set of already evaluated points is used to select the search centers, where one objective is the function value, and the other objective is the minimum distance from all other evaluated points.

There are no existing surrogate global optimization algorithms that select a large number of evaluation points in each iteration. For example, the maximum number of points used in [14], [96], and [120] were 5, 8, and 10 points, respectively. We have tested SOP with up to 64 simulated points per iteration. SOP can do many expensive objective function evaluations simultaneously which greatly reduces wall-clock time. We compare SOP with two other RBF methods, namely, Parallel Stochastic RBF [96] and ESGRBF [91, 107] on the

Table 1.1: Key features of algorithms developed in this thesis

Algorithm	Key Features
SOMS	- multistart framework - finds multiple local minima - handles expensive functions - high accuracy potentially
PADS	- very high dimensional problems - not too many number of processors - handles expensive functions
SOP	- not too high dimensional problems - many number of processors - handles very expensive functions

Real-Parameter Black-Box Optimization Benchmarking (BBOB) problems [47]. In addition to the BBOB testbed, the algorithms are also compared on an application problem that deals with groundwater bioremediation where the goal is to minimize the cost of the cleanup of contaminated groundwater subject to a contamination constraint. Sufficient conditions for the convergence of SOP will also be discussed.

In summary, this dissertation presents novel algorithms that can quickly identify near optimal solutions of computationally expensive blackbox functions when a relatively small number of function evaluations or a relatively short wall-clock time is allowable. The key features of each of the three algorithms are summarized in Table 1.1. Our algorithms are empirically validated on a broad set of benchmark problems. Finally, under some conditions, almost sure convergence for algorithms that follow the SOMS, PADS and SOP frameworks is proved.

Chapter 2

Surrogate MultiStart Algorithm for Nonlinear Programming Problems

2.1 Introduction

Several stochastic methods have been developed in the past to solve the problem in Eq. (1.1) for blackbox functions. When the objective function is in addition expensive, methods that use surrogate models in place of the objective function were shown to be the most successful [45, 56, 57, 95, 97, 98]. A surrogate-based optimization approach is designed to approximate the computationally expensive objective function. During the optimization search process, the surrogate is updated each time the exact objective function is evaluated, while the search is moving towards the global optima.

While all these methods are developed as stand-alone global optimization algorithms, in this work we are adopting the method based on a multistart framework (see Algorithm 2.1). Here, the term "multistart" will refer to a procedure that (in each iteration) selects a new variable vector which is used as starting a guess for a local optimization algorithm, for example, Sequential Quadratic Programming. Local optimization algorithms generally stop at local optima and may thus not be able to detect the global optimum. The multistart

Algorithm 2.1 Multistart Procedure

1. Set $j = 0$
2. While (Stopping condition is not satisfied)
   (a) $j = j + 1$
   (b) (Global Phase) Select a new decision variable x_j based on some scheme.
   (c) (Local Phase) Apply a local optimization algorithm $\mathcal{A}$ to improve x_j. Let x'_j be the solution obtained.
   (d) Update the best local minimum found so far.

   End

Output: All the local minima found

approach allows to continue the search globally, and therefore it is possible to escape from local optima.

The starting points for the local optimizations in the multistart algorithm have to be chosen carefully in order to avoid repeated convergence to the same local minimum. Different multistart techniques based on clustering were therefore proposed in order to avoid this inefficiency [10, 20, 58, 59, 114, 116, 117, 118].

This multistart approach is very useful since there are numerous applications of local optimization methods to complex nonlinear simulations for which there is no guarantee that there is only a single local minimum. This includes PDE constrained optimization where the simulation is solving a system of partial differential equations (e.g. [13]). However, in some cases, years have gone into setting up the interface between the local optimizer and the complex simulation model. Interfacing a multistart method with the existing pairing of a local optimizer and a complex simulation model can be much easier than it would be to build a new interface between a stand-alone global optimization method and the complex simulation model. Moreover, methods based on multistart can also be helpful for computationally expensive functions when identifying other good local minima besides the global minimum is of benefit, for example, when the practical implementation of the

globally optimal solution is much more difficult or time consuming than the implementation of a suboptimal solution.

Multi Level Single Linkage (MLSL) [59, 60] developed by Rinnooy Kan and Timmer in 1987 is one of the most classical and widely used linkage methods. Some other methods based on linkage methods are, for example, Topographical Multilevel Single Linkage [6] and Random Linkage [70]. See also [19, 103] for a brief summary of other linkage methods. In MLSL, the local optimization algorithm starts from selected sample points for which no other sample points with better objective function values are located within a critical distance. See Algorithm 2.3 in Section 2.4 for a brief review of MLSL. In 1988, Csendes [29] introduced a clustering method called GLOBAL based on Boender's algorithm [20]. In 2008, Csendes et al. modified the previous GLOBAL in some places to achieve higher efficiency and reliability [30]. A brief description of this algorithm is presented in Algorithm 2.6, Section 2.4. Another recent well-known multistart method that is based on scatter search is OptQuest/NLP multistart method [119]. The starting points are generated by a scatter-search algorithm and non-promising starting points (derived from the objective function value) are deleted. The method is available in MATLAB's Optimization Toolbox under the name GlobalSearch solver (Algorithm 2.5, Section 2.4). MATLAB's Optimization Toolbox also offers another multistart method, called MultiStart. MATLAB's MultiStart (Algorithm 2.4, Section 2.4) is simply a multistart algorithm that starts a local solver from every point (either uniformly generated or user-supplied) without estimating whether the point is likely to lead to a new and improved solution. All these methods follow the multistart framework given in Algorithm 2.1.

Due to highly nonlinear and multimodal characteristics of the likelihood function, uncertainty quantification for computationally expensive models can be very difficult to handle. Espinet [37] uses MLSL based global solver (coupled with ORBIT [124, 125]) to accurately define the high posterior density region of SOARS. SOARS [16, 17, 18] is a recently developed method that can produce a sample of the posterior distributions of selected input

parameters in a computationally efficient way for given monitoring data. Unlike traditional MCMC, SOARS uses a surrogate to approximate the likelihood. Espinet [37] reports that using MLSL coupled with the local optimizer ORBIT is more efficient than alternative multistart methods for the examined carbon sequestration application. He was able to decrease the necessary number of evaluations by a factor of twenty while obtaining accurate estimates of the posterior densities (i.e. they were very close to the posterior densities computed with MCMC analysis of the computationally expensive likelihood function without surrogates or optimization). This result is significant because many objective functions/likelihood functions are expensive to evaluate at each possible parameter value so it is not feasible to do the tens of thousands of function evaluations typically required in traditional MCMC. Hence, efficient multistart methods are applicable to uncertainty quantification problems in parameter calibration.

For complex simulation models, getting accurate derivatives can be infeasible or computationally very expensive. For commercial software, source code is in general not available, and thus automatic differentiation cannot be used. In practice, none of these multistart methods require derivatives, and therefore if coupled with derivative-free local search methods such as NEWUOA [88], DFO [27, 28] and ORBIT [124, 125], they can be considered as derivative-free global optimization methods.

We focus on the analysis of a scheme used in Global Phase (Step 2b of Algorithm 2.1) for selecting starting points for the local optimization algorithm. And thus, in the experimental sections, both the local optimization method $\mathcal{A}$ (Step 2c, Algorithm 2.1) and the stopping criterion for a local optimization $\mathcal{A}$ will be fixed across all compared multistart methods. This contrasts with [131] where a method called Dynamic Multistart Sequential Search (DMSS) is developed to identify the length of a single run of an algorithm. Once a decision to restart has been made, the algorithm is restarted with a different starting point which was drawn randomly without a rigorous scheme. In addition, the algorithm they consider is a stochastic global optimization algorithms such as Simulated Annealing (SA) or a simple elitist random

walk, not an actual local optimization algorithm in Step 2c, Algorithm 2.1. While in practice a stochastic global optimization could be restarted when the algorithm seems to get trapped in the local minimum, we do not consider here.

Although sharing some features with other multistart methods, **S**urr**O**gate **M**ulti**S**tart method (SOMS) differs in multiple aspects. While multistart algorithms have been used for solving global optimization problems, none of the available multistart methods are applicable to optimization problems with computationally expensive objective functions for which the number of allowable function evaluations is very limited.

The goals of this chapter are therefore the following.

1. We develop a surrogate model based multistart method, SOMS, which can efficiently locate the global minimum when used in conjunction with a local optimization solver.

2. We compare through empirical studies our method with a variety of alternative multistart methods including MLSL, MATLAB's MultiStart, MATLAB's GlobalSearch, and GLOBAL when the number of allowable function evaluations is limited. This contrasts with some earlier results including those reported in [83], in which the number of function evaluations was set to as high as $2 \times 10^4 \times d$.

3. In Section 2.7, we present a generic approach that allows us to create a test function that mimics the multimodal nature of objective functions arising in many blackbox simulations (e.g. parameter calibration in simulation model [107]). We compare the various multistart methods on our multimodal test functions.

The remainder of this chapter is organized as follows: The new algorithm SOMS is presented in Section 2.2. Theoretical properties of SOMS are discussed in Section 2.3. In Section 2.4, we review alternative multistart methods that are used in the numerical comparison. Section 2.5 gives an overview of the experimental setup followed by computational experiments on standard test functions in Section 2.6. In Section 2.7, we introduce a class of synthetic wavy function which is used to test the performance of multistart methods. The efficiency of each

multistart method is investigated and compared on on these functions in Section 2.8. We conclude with a summary in Section 2.9.

2.2 SOMS: SurrOgate MultiStart

A local optimization algorithm can be used for global optimization problems by combining it with a multistart method which starts the local solver from multiple selected starting points. In each iteration, methods based on multistart usually require a large number of objective function evaluations to ensure a thorough local search and to ideally explore all valleys of the objective function landscape. For problems with computationally expensive objective functions, only a very limited number of function evaluations can be done. Thus, doing many function evaluations in order to select the most promising starting points for the local search is not feasible. In this section, we propose the use of surrogates within the multistart to reduce the number of function evaluations required in the global search phase of multistart methods. As in any multistart method, we will assume that a local optimization algorithm $\mathcal{A}$ is available and that it is able to converge to a local minimum from a given starting point. Although the theorems presented in Section 2.3 require that f has a continuous second derivative, we assume that all derivatives of f are either unavailable or computationally intractable. The SOMS framework is outlined in Algorithm 2.2. The inputs are given below:

- A continuous real-valued function f defined on a compact hyperrectangle $\mathcal{D}$
- A particular response surface model (e.g. radial basis function introduced in Section 2.2.2)
- A local optimization method $\mathcal{A}$ that converges to a local minimum
- A critical distance r_k with the property $r_k \to 0$ as $k \to \infty$ (e.g. see Eq. (2.1))
- The maximum number of function evaluations allowed, denoted by MAXFE

Algorithm 2.2 SurrOgate MultiStart (SOMS).

Parameters:

- $N > 0$ the number of new random sample points generated in each iteration
- $\gamma \in (0, 1]$ the fraction of the total sample points to be selected in each iteration

---

1. Initialize $k = 0$. $C_0 = C_0^{unif} = \emptyset$. Build an initial surrogate $s_0(x)$ with the initial data set $\{(w_0, f(w_0)), ..., (w_{n_0}, f(w_{n_0}))\}$.
2. [Optional] Improve the initial surrogate using any surrogate model based global optimization method. In this step, n_1 additional points will be generated and evaluated with the expensive function f. The data set of points evaluated in this step is $\{(w_{n_0+1}, f(w_{n_0+1})), ..., (w_{n_0+n_1}, f(w_{n_0+n_1}))\}$.
3. Set $k = k + 1$
   (a) Generate N uniform sample points distributed over the variable domain $\mathcal{D}$, $x_{(k-1)N+1}, ..., x_{kN}$ and add them to the cumulative sample set $C_k = C_{k-1} \cup \{x_{(k-1)N+1}, ..., x_{kN}\}$.
   (b) Use the surrogate model $s_{k-1}(x)$ to predict the objective function values of the points in C_k.
   (c) Sort the whole sample C_k such that $s_{k-1}(x_1) \leq ... \leq s_{k-1}(x_{kN})$.
   (d) Reduce the sample set by choosing γ percent of the lowest value of points based on the sorted sample in Step 3c. Call this reduced sample set C_k^s.
   (e) Do the expensive objective function evaluation for every point in C_k^s.
   (f) Generate and evaluate a uniform sample point u_k and add this to a uniform sample set, $C_k^{unif} = C_{k-1}^{unif} \cup \{u_k\}$.
   (g) Combine the two sets $C_k^{combine} = C_k^s \cup C_k^{unif}$. Sort points in $C_k^{combine}$ based on the objective function value: $f(x_i) \leq f(x_j)$ if $x_i, x_j \in C_k^{combine}$ and $i \leq j$. Denote the order set by C_k^{order}.
   (h) Eliminate points form the set C_k^{order} that are within a distance r_k of other points in C_k^{order} that have a lower objective function value. Also delete points from C_k^{order} that have already been used as starting points for the local search.
   (i) Sequentially start a local search from the remaining points in C_k^{order}.
4. If the stopping criterion is not satisfied, update the surrogate and go to Step 3. Otherwise, stop and return the point with the lowest objective function value as the approximate global minimum.

In Step 1 of Algorithm 2.2, the initial response surface is built. The refinement of the surrogate model by selecting additional sample points in Step 2 is optional. Step 3 is the iterative steps. In Step 3a, the algorithm generates N uniform random points and uses the surrogate model to predict their objective function values. Based on the surrogate model predictions, in Steps 3d and 3e, the algorithm selects a fraction γ of the best points and evaluates the true objective function at these points. In Step 3f, a uniform random point is generated and added to the set of points selected in Step 3d. This combined set of points is then sorted based on their objective function values in Step 3g. In Step 3h, the radius rule is applied in order to eliminate some of the points that are too close to any points with a lower objective function value. As in MLSL [59, 60], the critical distance r_k in Eq. (2.1) is used:

$$r_k = \pi^{-\frac{1}{2}} \left(\Gamma(1 + \frac{d}{2})\, m(\mathcal{D})\, \frac{\sigma \log kN}{kN} \right)^{1/d}, \tag{2.1}$$

where Γ denotes the gamma function, m the lebesgue measure, and $\sigma > 0$ is a parameter. In Step 3i, a local optimization search is started from each of the points that passes the radius rule from Step 3h. Finally, in Step 4, the algorithm checks if the stopping condition is satisfied. If not, the algorithm updates the response surface model and continues with Step 3. Otherwise the best solution found as well as all the local minima are returned by the algorithm. While other stopping criteria may be used, we stop the algorithm after the maximum number of allowable computationally expensive function evaluations (MAXFE) has been reached.

2.2.1 Improving the Accuracy of the Initial Response Surface

After building the initial surrogate in Step 1, we allocate some (small) number of function evaluations to refine the initial response surface in Step 2 by using a surrogate model based global optimization routine. In the numerical experiments, we use the Metric Stochastic

Radial Basis Function (MSRBF) method by Regis and Shoemaker [95]. This method has been shown to work very efficiently in achieving decreases in the objective function value on multimodal surfaces given only a limited number of function evaluations. In each iteration, the algorithm builds the surrogate model to approximate the expensive objective function and then selects the most promising point for function evaluation. The point is selected from a set of random candidate points based on two criteria, namely the estimated objective function value based on surrogate model and the minimum distance from the set of previously evaluated points. Our numerical experiments showed that the addition of MSRBF in Step 2 of SOMS results in a more reliable initial response surface and speeds up the convergence in many cases.

2.2.2 A Cubic Radial Basis Function Model

In practice, users can freely choose surrogate models to use in SOMS, e.g. polynomial regression models, RBF models, kriging, support vector regression. An extensive review of these models can be found in [40, 41]. We use a cubic RBF model with a linear polynomial tail as a response surface in this work. For a cubic RBF model with linear polynomial tail, the initial experimental design must contain at least $d+1$ points in order to uniquely compute the surrogate model parameters [89].

In Step 4, at the end of the multistart iteration k, assume that n distinct points, $x_1, ..., x_n \in \mathbb{R}^d$, whose function values are known, are used for building the RBF interpolant which is defined as follows:

$$s(x) = \sum_{i=1}^{n} \lambda_i \phi(\|x - x_i\|) + p(x), \qquad x \in \mathbb{R}^d, \tag{2.2}$$

where $\|\cdot\|$ is the Euclidean norm, $\lambda_i \in \mathbb{R}$ for $i = 1, ..., n$, $\phi : \mathbb{R}_+ \to \mathbb{R}$ is a univariate function, and $p(x)$ is a polynomial tail. The order of the polynomial tail depends on the chosen RBF type. The polynomial is usually added in order to improve the stability.

For a cubic RBF $\phi(r) = r^3$ with a linear polynomial $p(x) = \mathbf{a}^T x + a_0$, where $\mathbf{a} = (a_1, ..., a_d)^T \in \mathbb{R}^d$, Eq. (2.2) can be simplified to the following form:

$$s(x) = \sum_{i=1}^{n} \lambda_i \|x - x_i\|^3 + \mathbf{a}^T x + a_0, \qquad x \in \mathbb{R}^d. \tag{2.3}$$

In order to compute the parameters (λ_i and a_j) of the RBF interpolant we have to solve a system of linear equations

$$\begin{pmatrix} \Phi & P \\ P^T & \mathbf{0}^{(d+1)\times(d+1)} \end{pmatrix} \begin{pmatrix} \lambda \\ \tilde{\mathbf{a}} \end{pmatrix} = \begin{pmatrix} \mathbf{F} \\ \mathbf{0} \end{pmatrix}, \tag{2.4}$$

where

$$P = \begin{bmatrix} x_1^{(1)} & \cdots & x_1^{(d)} & 1 \\ \vdots & \ddots & \vdots & \vdots \\ x_n^{(1)} & \cdots & x_n^{(d)} & 1 \end{bmatrix} \tag{2.5}$$

and where $x = (x^{(1)}, ..., x^{(d)})^T$, $\Phi \in \mathbb{R}^{n\times n}$ with $\Phi_{ij} = \phi(\|x_i - x_j\|) = \|x_i - x_j\|^3$, $i, j = 1, ..., n$, and where $\mathbf{F} = (f(x_1), ..., f(x_n))^T$, $\lambda = (\lambda_1, ..., \lambda_n)^T \in \mathbb{R}^n$, and $\tilde{\mathbf{a}} = \begin{bmatrix} \mathbf{a} \\ a_0 \end{bmatrix} \in \mathbb{R}^{d+1}$.

Powell [89] showed that the coefficient matrix in Eq. (2.4) is invertible if and only if $\text{rank}(P) = d + 1$ for a cubic RBF model. See [89] for other types of RBF models as well as more theoretical details.

If the basis centers are too close to each other, it can lead to numerical ill-conditioning. Therefore, in Step 4 of SOMS, before the surrogate is being updated, points that are too close (within distance τ) to other previously added basis centers are excluded and not used as centers $x_j \in D_k$ in the surrogate model. A tolerance $\tau = 10^{-3} \times \min(\text{ub} - \text{lb})\sqrt{d}$ is used in this work.

Although using RBF response surfaces for global optimization (e.g. [15, 45, 95, 97, 98]) and local optimization (e.g. [124, 125]) is not a new idea, none of these earlier methods employ the surrogate in the global phase of a multistart procedure (as in Algorithm 2.1).

2.2.3 Addition of Uniformly Selected Points

Adding a uniform random point in Step 3f as a sample point improves the performance of the algorithm on many multimodal test cases by adding diversity. Without this step, the response surface $s_{k-1}(x)$ is updated in each iteration based on the data from the initial experimental design and the data obtained from the local searches.

Then, in Step 3d of the next iteration, the best γ percent of the sample points are selected based on this response surface, which may predict better objective function values in the vicinity of already explored local minima. Hence, the uniformly selected point may add knowledge of the objective function in rather unexplored regions of the variable domain, and thus the global fit of the response surface can be improved. Therefore, it is likely that a better starting point for local optimization search is obtained, and the overall performance of the algorithm is improved. Numerical evidence to support this can be found in Section 2.6.3 where we compare the performance of a few selected test functions with and without a uniform point added.

2.3 Theoretical Properties of SOMS

In this section, we show that SOMS shares the theoretical properties of MLSL. The proofs of the following theorems follow similar arguments as those used by Rinnooy Kan and Timmer [59, 60]. All lemmas and theorems for MLSL also hold for SOMS with some modifications. For reasons of completeness, we present the proofs here. Note also that all the proofs in this section do not depend on the accuracy of the surrogate. Of course, in practice we would expect a more accurate surrogate to help speed up the convergence, but the proofs do not address the speed of convergence.

Rinnooy Kan and Timmer showed that in MLSL, two possible failures will not occur after a sufficiently large number of iterations:

1. (Type I error) Local optimization search is repeated in the region of attraction and

results in a local minimum which was already known.

2. (Type II error) Local optimization search will never start in the region of attraction (for which the local minimum has not yet been found) even if at least one sample point has been located in that region of attraction.

Definition 2.1. If x^* is a local optimum, the subset of points $\mathcal{R}(x^*)$ characterized by the property that the local optimization search, when started from any point in $\mathcal{R}(x^*)$ will lead to the local optimum x^*, is called the *region of attraction* of x^*. The region of attraction of the global optima is assumed to have non-null Lebesgue measure.

We will use the following notation:

Y_ν is the set of elements in $\mathcal{D}$ that are within distance ν of a stationary point of f, i.e. $Y_\nu := \{x \in \mathcal{D} : \|x - \bar{a}\| < \nu \text{ for any } \bar{a} \in \Lambda\}$ where Λ is the set of stationary points in $\mathcal{D}$.

Q_τ is the set of elements in $\mathcal{D}$ that are within distance τ of a point on the boundary $\partial\mathcal{D}$ of $\mathcal{D}$, i.e. $Q_\tau := \{x \in \mathcal{D} : \text{dist}(x, \partial\mathcal{D}) < \tau\}$ where $\text{dist}(x, C) := \inf_{x_1 \in C} \|x - x_1\|$, for any set C.

$M_{\tau,\nu}$ consists of the elements in $\mathcal{D}$ that do not belong to Y_ν or Q_τ, so that $\{Y_\nu, Q_\tau, M_{\tau,\nu}\}$ is a partition of $\mathcal{D}$. Note that since Y_ν and Q_τ are defined as open sets and $\mathcal{D}$ is bounded, $M_{\tau,\nu}$ is closed and therefore compact.

$\mathcal{D}_\tau$ is the complement of Q_τ, i.e. $\mathcal{D}_\tau = \mathcal{D} \setminus Q_\tau$. So $\mathcal{D}_\tau$ is the set of elements that are at least distance τ away from the boundary of $\mathcal{D}$.

$X^* \in \Lambda$ is the set of stationary points already *detected* in previous iterations of SOMS.

X^*_ν is the set of elements in $\mathcal{D}$ that are within distance ν of a stationary point that are already *detected*, i.e. $X^*_\nu := \{x \in \mathcal{D} : \|x - a^*\| < \nu, \text{ for any } a^* \in X^*\}$. Note that $X^*_\nu \subseteq Y_\nu$.

As in [60], in order to ensure that the two types of errors will not occur in SOMS, we have to require additional smoothness conditions of f. The following are assumptions (P0 to P5) necessary for the analysis of this algorithm:

(P0) $f \in C^2$.

(P1) A positive constant ν exists such that the distance between any two stationary points exceeds 2ν.

(P2) A positive constant τ exists such that all local minima of f occur in the interior of $\mathcal{D}_\tau$.

(P3) A local optimization method $\mathcal{A}$ is never started in an element of Q_τ or X^*_ν.

(P4) If a local optimization method $\mathcal{A}$ is applied to a point that is within distance ν of a stationary point $\bar{a}$, then local optimizer $\mathcal{A}$ will detect $\bar{a}$ and add it to X^*.

(P5) The number of stationary points is finite.

Lemma 2.2. *For any $\tau > 0$ and $\nu > 0$, let a be an element of $M_{\tau,\nu}$, let $B_{a,r} = \{x \in \mathcal{D} : \|x - a\| \leq r\}$, and let $A_{a,r} = \{x \in \mathcal{D} : \|x - a\| \leq r \text{ and } f(x) < f(a)\}$. Then (uniformly across $a \in M_{\tau,\nu}$), $lim_{r\to 0} \frac{m(A_{a,r})}{m(B_{a,r})} \geq \frac{1}{2}$, where $m(\cdot)$ is the Lebesgue measure.*

Proof. See Rinnooy Kan and Timmer [59]. □

We will now consider the probability that a local optimization method $\mathcal{A}$ is started incorrectly (type I error). The following theorem indicates that if you continue the algorithm indefinitely, there is a finite number K such that no additional local optimization search will be started after iteration K although the sampling in the global phase will continue. Theorem 2.3 is adopted from Rinnoy Kan and Timmer [60].

Theorem 2.3. *If the critical distance r_k is determined by*

$$r_k = \pi^{-\frac{1}{2}} \left(\Gamma(1 + \frac{d}{2})\, m(\mathcal{D})\, \frac{\sigma \log kN}{kN} \right)^{1/d}$$

then with $\sigma > 2$, the probability that a local optimization search is applied by SOMS in iteration k tends to 0 with increasing k. If $\sigma > 4$, then even if the sampling continues forever, the total number of local optimization searches started by SOMS is finite with probability 1.

Proof. Recall C_k^{unif} and $C_k^{combine}$ are defined in Algorithm 2.2 Step 3f and Step 3g, respectively. Let x be a sample point in $C_k^{combine}$. By Assumption (P3) that no local optimization search $\mathcal{A}$ will ever be started in $Q_\tau \cup X_{\nu^*}$, we can only start $\mathcal{A}$ at a point x if $x \in Y_\nu \setminus X_\nu^*$ or if $x \in M_{\tau,\nu}$ and there is no other point $x_j \in C_k^{combine}$ with $f(x_j) < f(x)$ within distance r_k of x. First we start with the latter case when $x \in M_{\tau,\nu}$.

Fix $x \in C_k^{combine} \cap M_{\tau,\nu}$. Denote $C_k^{combine,\,x} := C_k^{combine} \setminus \{x\}$ and $C_k^{unif,\,x} := C_k^{unif} \setminus \{x\}$. Since $C_k^{unif,\,x} \subseteq C_k^{combine,\,x}$,

$$\Pr\left(C_k^{combine,\,x} \cap A_{x,\,r_k} = \emptyset\right) \leq \Pr\left(C_k^{unif,\,x} \cap A_{x,\,r_k} = \emptyset\right). \tag{2.3.6}$$

Since the points in $C_k^{unif,\,x}$ are uniform distributed over $\mathcal{D}$,

$$\Pr\left(C_k^{unif,\,x} \cap A_{x,\,r_k} = \emptyset\right) \leq \left(1 - m(A_{a,\,r_k})/m(\mathcal{D})\right)^{\left|C_k^{unif}\right|-1}. \tag{2.3.7}$$

Note that the exponent, $\left|C_k^{unif}\right| - 1$, on the right side of the inequality (2.3.7) accounts for the case $x \in C_k^{unif}$. If, on the other hand, $x \in C_k^{combine} \setminus C_k^{unif}$, then this exponent will simply be $\left|C_k^{unif}\right|$. In either case, the upper bound, $\left(1 - m(A_{x,\,r_k})/m(\mathcal{D})\right)^{\left|C_k^{unif}\right|-1} = \left(1 - m(A_{x,\,r_k})/m(\mathcal{D})\right)^{k-1}$, is accurate. The radius

$$r_k = \pi^{-\frac{1}{2}} \left(\Gamma(1 + \frac{d}{2})\, m(\mathcal{D})\, \frac{\sigma \log kN}{kN}\right)^{1/d}$$

is chosen such that the final bound is decreasing with iteration k. More precisely, since $x \in M_{\tau,\nu}$, from Lemma 2.2, for any $0 < \beta < \frac{1}{2}$, $\frac{m(A_{x,\,r_k})}{m(B_{x,\,r_k})} \geq \beta$ for sufficiently large k. Since

$$m(B_{x,\,r_k}) = \frac{\pi^{\frac{d}{2}}}{\Gamma(\frac{d}{2}+1)} r_k^d = \sigma \log kN\, m(\mathcal{D})/kN,$$

it follows that

$$m(A_{x,\,r_k}) \geq \beta\sigma \log kN\, m(\mathcal{D})/kN.$$

Ignoring N in all terms in $O(\cdot)$ (since it is constant), we obtain, for sufficiently large k,

$$(1 - m(A_{x,\,r_k})/m(\mathcal{D}))^{k-1} \leq (1 - \beta\sigma \log kN/kN)^{k-1} \sim O(k^{-\beta\sigma}).$$

Thus, we have shown that for a fixed point $x \in C_k^{combine} \cap M_{\tau,\,\nu}$,

$$\Pr\left(C_k^{combine,\,x} \cap A_{x,\,r_k} = \emptyset\right) \sim O(k^{-\beta\sigma}). \tag{2.3.8}$$

Eq. (2.3.8) indicates that for any point $x \in C_k^{combine} \cap M_{\tau,\,\nu}$, the probability that no other point in $C_k^{combine}$ within distance r_k with smaller function value is $O(k^{-\beta\sigma})$.

Since the number of points in $C_k^{combine}$ is $\lceil \gamma kN \rceil + k$, we can conclude that the probability that there exists a sample point in $C_k^{combine} \cap M_{\tau,\,\nu}$ which has no other point in $C_k^{combine}$ within distance r_k with smaller function value is

$$(\lceil \gamma kN \rceil + k)\, O(k^{-\beta\sigma}) \sim O(k^{1-\beta\sigma}). \tag{2.3.9}$$

Hence, for any $\beta < 1/2$, the probability that a local optimization search is started from any element of $M_{\tau,\,\nu}$ is $O(k^{1-\beta\sigma})$. Obviously, if $\sigma > 2$, then we can choose $\frac{1}{\sigma} < \beta < \frac{1}{2}$, so that the probability that a local optimization search is started from any element of $M_{\tau,\,\nu}$ in iteration k tends to 0 with increasing k. Moreover, if we let ξ_k be the number of local optimization searches started from points in $M_{\tau,\,\nu}$ in iteration k, and if we choose $\sigma > 4$, then it is easy to show that $\sum_{k=1}^{\infty} \Pr[\xi_k > 0] < \infty$. The Borel-Cantelli Lemma tells us that even if the sampling continues forever, $\xi_k = 0$ for all but finitely many k's almost surely. That is, the total number of local optimization searches started from points in $M_{\tau,\,\nu}$ is finite with probability 1.

Now, we consider the other case when $x \in C_k^{combine} \cap (Y_\nu \setminus X_\nu^*)$. By Assumption (P4), the local optimization method $\mathcal{A}$ will detect a stationary point and add it to X^*. Since the

number of stationary points is finite, we conclude that the probability that $\mathcal{A}$ is applied to a point in $Y_\nu \setminus X_\nu^*$ tends to 0 with increasing k. Moreover, the total number of local optimization searches started from points in $Y_\nu \setminus X_\nu^*$ is also finite almost surely. □

Lemma 2.4. *For any set $A \subseteq \mathcal{D}$, if $m(A) > 0$, $Pr(C_k^{combine} \cap A \neq \emptyset)$ tends to 1 with increasing in k.*

Proof. Recall that in each iteration k, a uniform sample point is generated and added to the set C_k^{unif} and $C_k^{combine} := C_k^s \cup C_k^{unif}$. Moreover, the probability that a uniform sample of size n contains at least one point in a subset $A \subseteq \mathcal{D}$ is equal to $1-(1-\frac{m(A)}{m(\mathcal{D})})^n$ (Brooks [24]).

Since for any k, $C_k^{unif} \cap A \subseteq C_k^{combine} \cap A$, and $\Pr(C_k^{unif} \cap A \neq \emptyset) = 1-(1-\frac{m(A)}{m(\mathcal{D})})^k \to 1$ as $k \to \infty$, the result is now immediate. □

For a local minimum x^*, let $y_1 \in \mathbb{R}$ be the smallest y for which $L_{x^*}(y)$ contains a stationary point other than x^*. If there is no such y, then y_1 is the maximum of f over $\mathcal{D}$. Clearly, $f(x^*) < y_1$. Lastly, we define the *Basin* B_{x^*} of x^* and Lemma 2.6 as follows.

Definition 2.5. The Basin B_{x^*} of x^* is a subset of points $\{x \in \mathcal{D} : f(x) < y_1\}$ that contains x^* as its only stationary point.

Lemma 2.6. *There exists a $\bar{y}$ with $f(x^*) < \bar{y} \leq y_1$ such that the set $E = \{x \in B_{x^*} : f(x) < \bar{y}\}$ has positive measure and has an empty intersection with Q_τ or $Y_\nu^{x^*}$ where $Y_\nu^{x^*} := Y_\nu \setminus \{x \in \mathcal{D} : \|x - x^*\| < \nu\}$.*

Proof. Assumptions (P1) and (P2) were used to prove this lemma. See Rinnooy Kan and Timmer [60]. □

Theorem 2.7. *If r_k tends to 0 with increasing k, then the probability that SOMS detects any local minimum x^* within a finite number of iterations is equal to one.*

Proof. For any $\delta > 0$, let E_δ be the set $\{x \in B_{x^*} : f(x) < y_\delta\}$, where y_δ is the infimum of f over points in B_{x^*} that are within distance δ of a point outside $E = \{x \in B_{x^*} : f(x) < \bar{y}\}$,

where E is defined in Lemma 2.6. Since $\bar{y} > f(x^*)$, it follows from the continuity of f that there exists a $\delta_0 > 0$ such that E_{δ_0} has positive measure.

Since $r_k \to 0$, there exists a $K > 0$ such that $r_k < \delta_0$ whenever $k > K$. Let ζ be the index of the iteration in which a local minimum x^* is found. We claim that $\forall k > K \left(\{C_k^{combine} \cap E_{\delta_0} \neq \emptyset\} \Rightarrow \{\zeta \leq k\}\right)$.

To prove the claim, we let $k > K$ and suppose that $C_k^{combine} \cap E_{\delta_0} \neq \emptyset$. Let $x_j = \arg\min_{x \in C_k^{combine} \cap E_{\delta_0}} f(x)$.

It follows that either x^* has been discovered previously or (by the definition of E_{δ_0}) a local optimization search $\mathcal{A}$ will be started at x_j to find x^*. In either case, $\zeta \leq k$, and the claim is now verified.

The claim implies that $\Pr(C_k^{combine} \cap E_\delta \neq \emptyset) \leq \Pr(\zeta \leq k)$ for all $k > K$. Since $\Pr(C_k^{combine} \cap E_{\delta_0} \neq \emptyset) \to 1$ (by Lemma 2.4), so does $\Pr(\zeta \leq k)$.

It then follows that $\Pr(\zeta < \infty) = \Pr\left(\cup_{k>K}\{\zeta \leq k\}\right) = \lim_{k\to\infty} \Pr(\zeta \leq k) = 1$. In other words, ζ is finite with probability 1. □

2.4 Multistart Methods in the Numerical Comparison

We will compare SOMS to earlier multistart methods, namely MLSL, MATLAB's MultiStart, MATLAB's GlobalSearch, and GLOBAL. MATLAB's MultiStart and MATLAB's GlobalSearch introduced in the MATLAB R2010a version are contained in MATLAB's global optimization toolbox. Note that while there are several options of local solvers for MATLAB's MultiStart (e.g. fmincon, fminunc, lsqnonlin etc.), fmincon is the only local optimizer used in MATLAB's GlobalSearch. We chose the methods MLSL and GLOBAL because they have been shown to be both theoretically and empirically well-suited multistart methods for blackbox functions. A brief description of each of these algorithms is presented in Algorithms 2.3 through 2.6.

All methods except MATLAB's MultiStart are based on the clustering method that

Algorithm 2.3 Multi Level Single Linkage (MLSL) [59, 60]

Parameters:

- $N > 0$ the number of new random sample points generated in each iteration
- $\gamma \in (0, 1]$ the fraction of the total sample points to be selected in each iteration

1. Generate N uniform points, do expensive function evaluation f at these points, and add the points to the current cumulative sample set $C_k = C_{k-1} \cup \{x_{(k-1)N+1}, ..., x_{kN}\}$
2. Let T_k be the reduced set of sample points constructed by taking the γ percent points in C_k with the lowest function value.
3. Examine the points in T_k one at a time, starting from the point with the lowest function value. A local optimization is initiated at a point in T_k if there is no other point in T_k within some critical distance r_k which has a lower objective function value. Repeat this procedure until every point in T_k has been examined.
4. If the stopping criterion is not met, go to Step 1. Otherwise, return the point with the lowest value found, and stop.

Algorithm 2.4 MATLAB's MultiStart (MS)

1. Generate a starting point from uniform distribution.
2. Start the local solver from the starting point in Step 1.
3. If the stopping criterion is not met, go to Step 1. Otherwise, return the point with the lowest value found, and stop.

Algorithm 2.5 MATLAB's GlobalSearch (GS) [119]

Parameters:

- x_0 the fmincon starting point used in Step 1
- N_1, $N_2 > 0$ the number of points used in Steps 2 and 3

1. Run fmincon from a specified x_0.
2. Generate N_1 trial points using scatter search.
3. (Stage 1) Start a local optimization from the point with the lowest function value among the first $N_2 < N_1$ points.
4. Initialize the basin of attraction, counters and threshold based on points from Step 1 and 2 (see [119] for more details).
5. (Stage 2) Examine all the remaining points, one at a time: Run a local search optimizer from a point if it is not in a basin of any existing local minima and $f(x) <$ threshold. Update the basins of attractions, counters and threshold. Continue this step until all the trial points have been examined or until the stopping criterion is met.
6. Return the point with the lowest value found, and stop.

Algorithm 2.6 GLOBAL (GLOB) [30]

Parameters:

- $N > 0$ the number of new random sample points generated in each iteration
- $\gamma \in (0, 1]$ the fraction of the total sample points to be selected in each iteration

---

1. Generate N uniform points, do expensive function evaluation f at these points, and add these points to the current cumulative sample $C_k = C_{k-1} \cup \{x_{(k-1)N+1}, ..., x_{kN}\}$.
2. Let T_k be the reduced sample constructed by taking the γ percent points in C_k with the lowest function value.
3. Cluster the points in T_k one at a time.
4. Start the local optimization from those points in T_k not yet clustered. Repeat Step 3 until every point has been assigned to a cluster.
5. If a new local minimizer has been found, go to Step 1.
6. Return the point with the lowest value found, and stop.

---

excludes non-promising starting points, decreases the number of local searches, and converges to the global minimum faster. Note that while a uniform random distribution is used to generate new candidate points in SOMS, MLSL and GLOBAL, in practice, a sample obtained by Deterministic Low-Discrepancy Sequences (LDS) (or others) can also be used. See for example [64] where LDS has been used in MLSL.

In all these multistart methods including SOMS, the local phase evaluates the exact objective function f. However, Algorithms 2.3 through 2.6 (i.e. all algorithms except SOMS) also evaluate the exact function in the global phase of the search where SOMS first uses the response surface to exclude some non-promising starting points, which enables more global searching.

In each iteration of MLSL and GLOBAL, the algorithm is based on a sampling strategy where a set of sample points is selected at which the expensive objective function is evaluated. Several of these points are used as a starting point for the local search. Due to the strict limitation of allowable function evaluations, only a small number of total function evaluations

(m_f) can be done in each iteration. The default parameters $N = 10d$, and $\gamma = \min(0.2,\ 2/d)$ are taken for GLOBAL*. For MLSL, $N = 10$ and $N = 20$ are taken for the first three and the last four problems of the Dixon-Szegö test bench [34], respectively.

For MATLAB's GlobalSearch, the sample points that are evaluated before the local search is started are generated only once (whereas in SOMS, MLSL, and GLOBAL a new set of sample points is generated in each iteration). The two main algorithm parameters of GlobalSearch are the number of trial points (N_1) and the number of points used in Stage 1 of the algorithm (N_2). The default values for these two parameters are $N_1 = 1000$ and $N_2 = 200$, respectively. However, in our experiment, the maximum number of allowable function evaluations m_f for each test functions are quite small. Thus, we used lower values for N_1 and N_2 in order to enable several local searches before the budget of function evaluations is exhausted. We experimented with different algorithm parameters for several combinations of ($N_1,\ N_2$) each with 30 trials to pick the best pair for each problem. The resulting parameters are shown in Table 2.2.

MATLAB's MultiStart starts the local solver from a uniformly generated point (Algorithm 2.4), and thus there are no parameters to adjust.

For SOMS, the set of generated sample points of size N are not evaluated with the expensive objective function. We use the computationally cheap surrogate model to predict the objective function values of these points. Thus, N can be chosen rather large. However, with an increasing number of dimensions and increasing N, SOMS' own computational complexity increases, and thus a reasonable trade-off between algorithm efficiency and computational effort must be made by choosing N reasonably.

Denote by $N^{(s)}$, $N^{(m)}$, ($N_1^{(GS)}$, $N_2^{(GS)}$), and $N^{(G)}$ the parameters used in Algorithms 2.2, 2.3, 2.5, and 2.6, respectively. For example, for $d = 20$, $m_f = 2000$, and $N^{(s)} = N^{(m)} = 100d$, SOMS does 2000 computationally cheap function evaluations using the response surface. However, MLSL does these 2000 evaluation with the expensive objective function. Thus,

*The software for GLOBAL is obtained from: http://www.inf.u-szeged.hu/∼csendes/Reg/regform.php.

MLSL uses up the budget of allowable function evaluations before a local search could be started. Therefore, $N^{(m)} = 10d$ is a more appropriate choice. The same argument was used when we select the parameters N for the other multistart methods. The values of the parameters that are used for MLSL, SOMS and GlobalSearch are shown in Tables 2.1 and 2.2.

2.5 Numerical Experiments

In this section, we show the numerical results obtained for SOMS and the alternative algorithms. We did 30 trials with each algorithm for each test problem. For SOMS, the initial evaluation points for fitting the RBF model are generated by a symmetric Latin hypercube design (SLHD) [129]. This initial set of points is used only for fitting the RBF model and the points in this design are not used for starting the local search in Step 3a of SOMS.

2.5.1 Experimental Setup

The experiments are all run using MATLAB 7.14 (R2012a) on Intel(R) Core(TM) i7 CPU @3.40GHz 3.40 GHz. MATLAB's fmincon is used as a local solver in all algorithms. The algorithm starts a new local optimization from the next selected point whenever the local solver has converged to a local minimum until the maximum number of function evaluations (m_f) is reached. The termination tolerance of the fmincon was set to 10^{-8}.

We chose fmincon as local solver since it is the only available local optimization method for GlobalSearch. We use the same local solver for all algorithms to facilitate a fair comparison. We did not supply the gradient of the test function to fmincon because we treat the test functions as blackbox problems in order to determine the applicability of the various methods for true blackbox problems. Thus, in each iteration of fmincon, the gradient

Table 2.1: Algorithm parameters used for SOMS (Algorithm 2.2) and MLSL (Algorithm 2.3) for different sets of test problems. Note that σ in Eq. (2.1) was set to be 4 for both SOMS and MLSL.

(a) Dixon-Szegö

	MLSL		**SOMS**			
Function	$N^{(m)}$	$\gamma^{(m)}$	$N^{(s)}$	$\gamma^{(s)}$	n_0	n_0+n_1
GP	10	0.2	$500d$	0.005	$2(d+1)$	$2(d+1)$
Branin	10	0.2	$200d$	0.005	$2(d+1)$	$2(d+1)$
HA3	10	0.2	$500d$	0.001	$2(d+1)$	$2(d+1)$
HA6	20	0.2	$200d$	0.001	$2(d+1)$	$2(d+2)$
SH5	20	0.2	$200d$	0.005	$2(d+1)$	$2(d+2)$
SH7	20	0.2	$200d$	0.005	$2(d+1)$	$2(d+2)$
SH10	30	0.2	$200d$	0.005	$2(d+1)$	$2(d+2)$

(b) Low-dimensional wavy test Functions

	MLSL		**SOMS**			
Function	$N^{(m)}$	$\gamma^{(m)}$	$N^{(s)}$	$\gamma^{(s)}$	n_0	n_0+n_1
easySquareWavy	$10d$	0.2	$1000d$	0.002	$2(d+1)$	$2(d+1)$
Wavy-1D			$1000d$	0.002	$2(d+1)$	$2(d+1)$
LgWavy1			$1000d$	0.002	$2(d+1)$	$2(d+4)$
LgWavy2			$1000d$	0.003	$2(d+1)$	$2(d+5)$
SchWavy-2D			$500d$	0.006	$2(d+1)$	$2(d+4)$
SchWavy-5D			$150d$	0.01	$2(d+1)$	$2(d+5)$

(c) High-dimensional wavy test functions

	MLSL		**SOMS**			
Function	$N^{(m)}$	$\gamma^{(m)}$	$N^{(s)}$	$\gamma^{(s)}$	n_0	n_0+n_1
Sch-10D	$10d$	0.2	$100d$	0.008	$2(d+3)$	$2(d+7)$
Sch-15D			$100d$	0.006	$2(d+1)$	$2(d+7)$
Sch-20D			$100d$	0.008	$2(d+3)$	$2(d+7)$
SchWav-10D			$100d$	0.008	$2(d+3)$	$2(d+7)$
SchWav-15D			$100d$	0.006	$2(d+1)$	$2(d+7)$
SchWav-20D			$100d$	0.008	$2(d+3)$	$2(d+7)$

Table 2.2: Algorithm parameters for GlobalSearch (Algorithm 2.5) for different sets of test problems

(a) Dixon-Szegö

Function	GP	BR	HA3	HA6	SH5	SH7	SH10
(N_1, N_2)	(120,12)	(80,5)		(180,18)	(350,35)		

(b) Low-dimensional wavy test functions

Function	easySquareWavy	Wavy-1D	LgWavy1
(N_1, N_2)	(50,5)		

Function	LgWavy2	SchWavy-2D	SchWavy-5D
(N_1, N_2)	(80,8)		(120,12)

(c) High-dimensional wavy test functions

Function	All test functions in Table 2.9
(N_1, N_2)	(200,50)

is estimated by finite differences, which requires an additional d function evaluations for a d-dimensional problem. Thus, a fairly large number of function evaluations was spent in the local optimization phase for all algorithms (including SOMS).

2.5.2 Performance Measure

An algorithm is said to *identify* or *locate* the global minimum if the algorithm can get within an absolute tolerance $d \times 10^{-4}$ of the global minimum point x^*. To measure the effectiveness, we run each algorithm for 30 trials and keep track of the number of trials that are able to identify the global minimum within a pre-defined number of function evaluations (m_f). To measure relative efficiency, we compute the average and the standard deviation of the number of function evaluations an algorithm actually required to identify the global minimum.

Table 2.3: Dixon-Szegö testbed. Test functions for numerical experiments.

Function	d	**Domain**	**No. of local min**
Goldstein & Price	2	$[-2, 2]^2$	4
Branin	2	$[-5, 10] \times [0, 15]$	3
Hartmann3	3	$[0, 1]^3$	4
Hartmann6	6	$[0, 1]^6$	6
Shekel5	4	$[0, 10]^4$	5
Shekel7	4	$[0, 10]^4$	7
Shekel10	4	$[0, 10]^4$	10

2.6 Numerical Experiments for Dixon-Szegö Test Functions

The purpose of this section is to evaluate the performance of five algorithms for multistart methods on the well-known Dixon-Szegö test bench [34] when the number of function evaluations allowed is limited. This testbed consists of the seven test problems shown in Table 2.3. The abbreviations MS for MATLAB's MultiStart, GS for MATLAB's GlobalSearch, and GLOB for GLOBAL are used throughout this section.

The performance of an algorithm is measured by computing the number of function evaluations it takes until the algorithm locates the global minimum. For each of the Dixon-Szegö test function, we do 30 trials with each algorithm (7 test functions x 30 trials x 5 algorithms = 1,050 total runs). The corresponding average and the standard deviation of the objective function value are calculated and reported in Table 2.4. The number of trials for which each algorithm failed to locate the global minimum within m_f function evaluations are reported in Table 2.4c. Figure 2.1 shows box plots of the statistics. Note that for the trials an algorithm failed to locate the global minimum within m_f function evaluations, the number of function evaluations for that particular trial was set to m_f, in order to calculate the average and standard deviation in Table 2.4 and Figure 2.1.

Table 2.4: Results for Dixon-Szegö. Shown is the (a) average number and (b) standard deviation (over 30 trials) of function evaluations until the algorithm locates the global minimum. (c) shows the number of trials for which an algorithm failed to find the global minimum within m_f function evaluations.

(a) average number

Method	Goldstein&Price	Branin	Hartmann3	Hartmann6	Shekel5	Shekel7	Shekel10	**Total**
	$m_f = 300$	$m_f = 100$	$m_f = 200$	$m_f = 600$	$m_f = 1000$	$m_f = 1000$	$m_f = 1000$	
MLSL	57.10	**23.57**	**55.10**	149.37	310.10	402.10	316.80	1314.13
SOMS	**56.97**	23.83	56.10	**139.17**	325.60	298.43	**275.67**	**1175.77**
MS	69.37	26.50	68.33	172.80	326.97	**242.53**	288.63	1195.13
GS	125.20	26.50	89.30	251.60	404.27	495.20	622.00	2014.07
GLOBAL	77.80	35.03	60.30	222.87	**271.87**	276.20	349.50	1293.57

(b) standard deviation

Method	Goldstein&Price	Branin	Hartmann3	Hartmann6	Shekel5	Shekel7	Shekel10
MLSL	35.10	1.91	31.14	50.46	199.02	283.30	273.76
SOMS	36.56	2.70	31.51	88.95	198.88	230.56	173.80
MS	41.39	3.94	44.64	63.22	224.97	173.61	183.92
GS	114.82	3.93	71.32	177.24	279.73	431.23	411.33
GLOBAL	46.54	1.65	7.72	89.45	146.10	176.02	253.37

(c) number of failed trials (out of 30 total trials)

Method	Goldstein&Price	Branin	Hartmann3	Hartmann6	Shekel5	Shekel7	Shekel10	**Total No. of Failed Trials**
MLSL	-	-	-	-	-	2	2	4/210
SOMS	-	-	-	-	-	-	-	0/210
MS	-	-	-	-	-	-	-	0/210
GS	7	-	5	3	-	7	10	32/210
GLOBAL	-	-	-	-	-	-	-	0/210

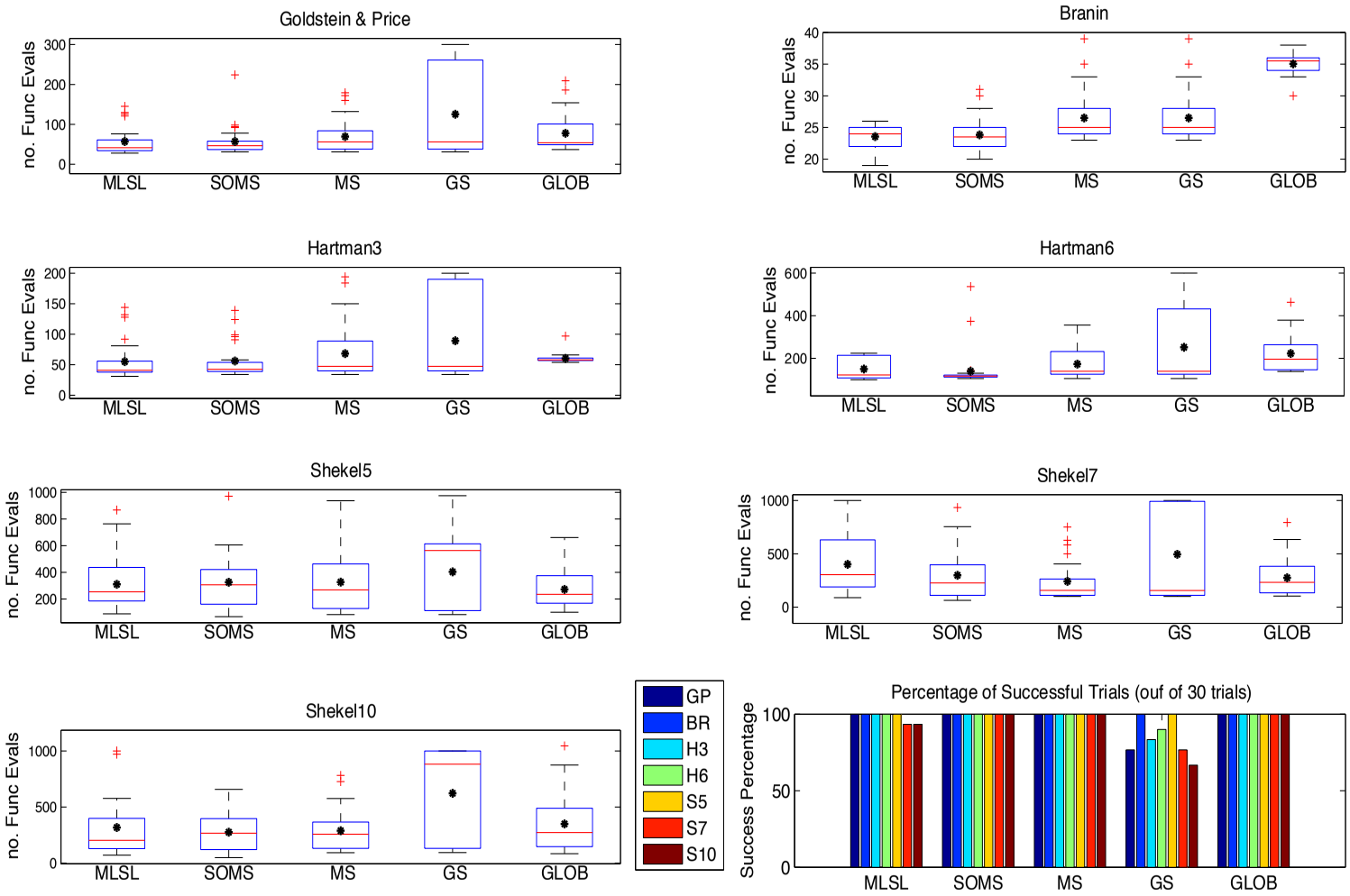


Figure 2.1: Box plots for Dixon-Szegö test functions. Box plots of statistics for the number of function evaluations needed by each algorithm to find the global minimum over 30 trials. The asterisk (*) denotes the mean of the data. Box plots with low values for mean and median are better. The bottom right bar chart is the percentage of trials that are close enough to the global minimum (high is best).

2.6.1 Discussion on the Dixon-Szegö Test Results

From Table 2.4a, we can see that SOMS requires the lowest average number of function evaluations to locate the global minimum for three of the seven test functions. The total number of function evaluations required to locate the global minimum over all seven test functions (reported in the last column of Table 2.4a) is lowest for SOMS. These results can also be seen in Figure 2.1.

Also, the efficacy of SOMS can be seen by the fact that SOMS is able to locate the global minimum in all 30 trials for all test functions (as reported in Table 2.4c). MATLAB's MultiStart method is also able to locate the global minimum for all test problems in all trials. It performs best for the Shekel7 test function and requires the second lowest total number of function evaluations over all problems to find the global minimum.

MLSL performed well for the first five test functions, but it failed to find the global minimum for two of the 30 trials for Shekel7 and Shekel10. GLOBAL performs well for Shekel5 and Shekel7.

MATLAB's GlobalSearch failed to locate the global minimum within m_f evaluations for five of the seven test problems. Thus, it requires a comparatively very large number of function evaluations and has the largest standard deviation for almost all test functions.

These results can also be visualized in Figure 2.1, where the mean of the data is displayed by the asterisk (*), median by the red line in the middle, high and low quartiles by the outer blue box, 1.5-IQR range by the whiskers, and outliers by +.

MATLAB's MultiStart does not perform as well as SOMS, but the results in this section indicate that it is more efficient and effective than its counterpart GlobalSearch, which relies on a complex clustering procedure. MATLAB's MultiStart also performs better than MLSL and GLOBAL for many problems in Dixon-Szegö testbed. The reason for this performance difference may be related to the generation of sample points from which the local search is started. MultiStart generates only a single point while the other methods use $N^{(s)}$, $N^{(m)}$, $N_1^{(GS)}$, and $N^{(G)}$ points, respectively.

Table 2.5: One-tailed two sample t-test results for Dixon-Szegö. A value of 1 indicates that SOMS is better at 5% significance level, and a value of 0 indicates that we do not have evidence to suggest that the performance of SOMS and the compared algorithm is different.

Function	SOMS vs. MLSL	SOMS vs. MS	SOMS vs. GS	SOMS vs. GLOB
Goldstein & Price	0	0	**1**	**1**
Branin	0	**1**	**1**	**1**
Hartmann3	0	0	**1**	0
Hartmann6	0	**1**	**1**	**1**
Shekel5	0	0	0	0
Shekel7	0	0	**1**	0
Shekel10	0	0	**1**	0
Total	0	2	6	3

To justify that the results are statistically significant, a one-tailed two sample t-test has been conducted at the 5% significance level. The test decision for the alternative hypothesis is that the μ_{SOMS} is less than $\mu_{\mathcal{A}}$, where μ_{SOMS} is the average number of function evaluations needed by SOMS to find the global minimum and $\mu_{\mathcal{A}}$ is defined similarly for any compared algorithm $\mathcal{A}$. Test results are reported in Table 2.5. The result is 1 if the test rejects the null hypothesis (i.e. SOMS is superior) at the 5% significance level, and 0 otherwise (i.e. we fail to reject the null hypothesis). The results indicate that SOMS and MLSL are not significantly different for any test functions. SOMS is better than MultiStart, GlobalSearch and GLOBAL for two, six, and three out of seven test functions, respectively.

2.6.2 Parameters of MATLAB's GlobalSearch solver

We compared the performance of GlobalSearch with the default parameter settings ($N_1 = 1000$, $N_2 = 200$) to our alternative parameter settings (Table 2.2). Using the default parameters, we also use the default stopping criterion of GlobalSearch. The results are shown in Table 2.6. We set the total number of function evaluations of GlobalSearch with default values for the trials where the global minimum was not detected to the actual total number of evaluations done in that trial. The results show that even with default parameter values

and a larger budget of function evaluations, GlobalSearch is not able to perform nearly as well as the alternative methods.

2.6.3 Influence of the Uniformly Selected Point in SOMS

As mentioned in Section 2.2.3, adding a uniform point in Step 3f can help improve the performance of SOMS. We illustrate this on the last four test functions (which are higher dimensional test problems) of Dixon-Szegö. Table 2.7 gives the average and the standard deviation of the number of function evaluations over 30 trials of SOMS with and without adding the uniformly selected point until the algorithm locates the global minimum or the total number of function evaluations exceeds m_f. The results show that the performance of SOMS improves for three of the four test problems when adding the uniformly selected point, i.e. SOMS requires fewer function evaluations to detect the global minimum. Moreover, if we do not add the uniformly selected point, SOMS fails to detect the global minimum within m_f function evaluations for Hartmann6 and Shekel5 for one and four trials, respectively.

2.7 Wavy Test Functions

The function given in Figure 2.2 is called the wavy function in [71], where it was used as an example and is defined by

$$f(x) = |2(x-24) + (x-24)\sin(x-24)|\,,\ x \in [-20,\, 60]. \tag{2.7.1}$$

The global minimum point is at $x = 24$ with the function value 0. As illustrated in Figure 2.2, the function has several local minima and is thus a challenging optimization problem [71]. We refer to this function as Wavy-1D.

Table 2.6: GlobalSearch solver with different algorithm parameters. Comparison of the number of function evaluations required by GlobalSearch to locate the global minimum using our modified parameters and the default parameters. The number of trials for which GlobalSearch failed to find the global minimum within m_f function evaluations is shown in the last table.

(a) average number

Param	Goldstein&Price	Branin	Hartmann3	Hartmann6	Shekel5	Shekel7	Shekel10	**Total**
Modified	125.20	26.50	89.30	251.60	404.27	495.20	622.00	2014.07
Default	496.50	26.50	372.53	584.67	1404.53	1067.33	1580.47	5532.53

(b) standard deviation

Param	Goldstein&Price	Branin	Hartmann3	Hartmann6	Shekel5	Shekel7	Shekel10
Modified	114.82	3.93	71.32	177.24	279.73	431.23	411.33
Default	709.86	3.94	521.44	653.023	1281.23	1130.91	1369.95

(c) number of failed trials (out of 30 total trials)

Param	Goldstein&Price	Branin	Hartmann3	Hartmann6	Shekel5	Shekel7	Shekel10	**Total Failed Trials**
Modified	7	-	5	3	-	7	10	32/210
Default	1	-	-	-	4	4	3	12/210

Table 2.7: Results of SOMS with and without adding a uniform point in Step 3f of Algorithm 2.2. Shown is the average and the standard deviation of the number of function evaluations over 30 trials until the algorithm locates the global minimum or the total number of function evaluations exceeds m_f. The number of failed trials is given in the last column.

Test Function	Average		Std Dev		No. of failed Trials	
	with Unif	w.o. Unif	with Unif	w.o. Unif	with Unif	w.o. Unif
Hartmann6 ($m_f = 600$)	**139.17**	199.37	88.95	160.36	-	1
Shekel5 ($m_f = 1000$)	**325.60**	391.27	198.88	306.73	-	4
Shekel7 ($m_f = 1000$)	298.43	**253.07**	230.56	178.25	-	-
Shekel10 ($m_f = 1000$)	**275.67**	333.97	173.80	242.51	-	-

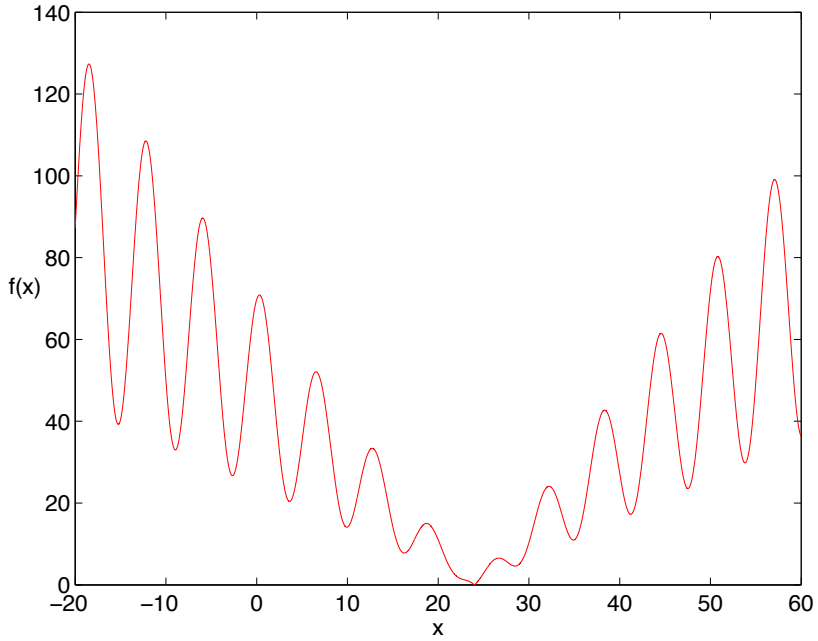


Figure 2.2: Wavy-1D [71]

2.7.1 Creating Wavy Test Functions for Optimization

In this section we discuss how a more complex and multivariate version of the Wavy 1-D test function can be generated. Given the underlying function f_u, a wavy test function of f_u can be constructed by adding a so-called "wavy part". Mathematically, if f_u is a real-valued function on a domain $\mathcal{D} \in \mathbb{R}^d$ with the global minimum at $x^* = (m_1, ..., ., m_d) \in \mathcal{D}$, then the wavy function of f_u, denoted by $f_{\text{wavy}} : \mathcal{D} \to \mathbb{R}$ is a function defined at a given point $x = (x_1, ..., x_d) \in \mathcal{D}$ by

$$f_{\text{wavy}}(x) = f_u(x) + Wavy(x) \tag{2.7.2}$$

where

$$Wavy(x) = A(x) \times (W_1(x) \prod_{i=2}^{d} W_i(x) + 1), \tag{2.7.3}$$

$$W_j(x) = \begin{cases} \sin(\omega_j(x_j - m_j) - \pi/2); & j = 1 \\ \sin(\omega_j(x_j - m_j) + \pi/2); & j > 1 \end{cases}, \tag{2.7.4}$$

$A : \mathcal{D} \to (0, \infty)$ and $\Omega = (\omega_1, ..., \omega_d) \in \mathbb{R}^d_+$.

A is the amplitude function of $Wavy(x)$ controlling the height of the wave at each point x in the domain. $A_1(x) = c$, where $c > 0$, $A_2(x) = c \times \|x - x^*\|$, or $A_3(x) = c/(1 + \|x - x^*\|)$ are examples of the amplitude function. While A_1 is just a constant amplitude, with A_2, points near the global minimum have a small amplitude while points further away have a larger amplitude. The effect of A_3 on the wavy function is the opposite of A_2, i.e. points near the global minimum have a larger amplitude than points further away. For each variable j, ω_j is the angular frequency. The argument x_j in each of the $W_j(\cdot)$ is shifted to $(x_j - m_j)$, rescaled by ω_j, and finally $\pi/2$ is added or subtracted. We define the function in this way so that the new function f_{wavy} in Eq. (2.7.2) has the same global minimum point as f_u and the same global minimum value. Thus, if the global optimum of the underlying function f_u is known, the global optimum of f_{wavy} is known as well. The following example is a simple example illustrating the method.

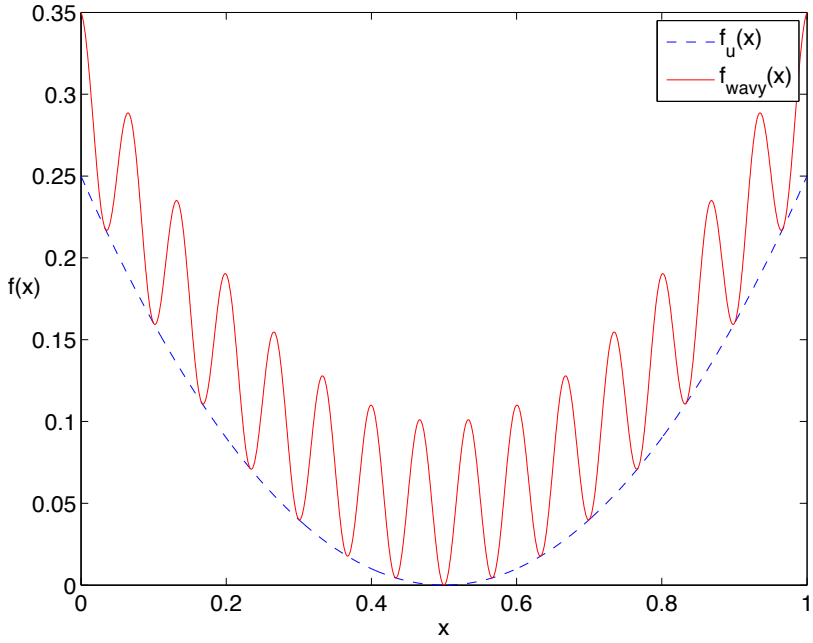


Figure 2.3: easySquareWavy

In this example, the underlying function is

$$f_u = (x - 0.5)^2.$$

The minimum value is 0 at $x^* = 0.5$. Let $A(x) = A_1(x) = 0.05$ and $\omega_1 = 30\pi$. Then,

$$f_{\text{wavy}}(x) = (x - 0.5)^2 + 0.05(\sin(30\pi(x - 0.5) - \pi/2) + 1)$$

has the same global minimum as f_u. f_u and f_{wavy} are illustrated in Figure 2.3. We will refer to this function as easySquareWavy.

2.7.2 Underlying Function

To obtain a multimodal underlying function f_u (before adding the wavy part), we use (a) the Lagrange interpolating polynomial [105] and (b) the Schoen test function [102] as f_u. We now give a brief review on these two functions.

2.7.2.1 Lagrange interpolating polynomial

For a given set of $n+1$ distinct points $\{(x_0, y_0), ..., (x_n, y_n)\}$, there exists an nth order polynomial which passes through all of these points. In particular, for $i = 1, .., n$, consider the basis polynomials $L_i(x)$ of the following form:

$$L_i(x) = \prod_{j=0, j\neq i}^{n} \frac{(x - x_j)}{(x_i - x_j)}.$$

The basis polynomial $L_i(x)$ has the property that

$$L_i(x_j) = \begin{cases} 0 & j \neq i \\ 1 & j = i \end{cases}.$$

Define a polynomial $L(x)$ by

$$L(x) = \sum_{i=0}^{n} y_i L_i(x).$$

Then, $L(x)$ has the property that $L(x_i) = y_i, i = 1, ..., n$ i.e. L interpolates the function through these points. The polynomial L is called the Lagrange interpolating polynomial.

Therefore, by specifying appropriate pairs of (x_j, y_j), we can use the Lagrange polynomial to define an underlying function for the wavy function which passes through the specified points. Moreover, since the optimum value of a polynomial function can be determined easily, it is an advantage for using it as an underlying function to create a wavy test function for optimization. An example of a one-dimensional Lagrange function along with its wavy function is shown in Figure 2.4. The function $L(x)$ in the figure (the blue dashed line) is used as the underlying function. The function $L_{\text{wavy}}(x)$ (red solid line) is the function LagrangeWavy2 listed in Table 2.8.

2.7.2.2 Schoen Function

The Schoen functions are defined as follows [102]:

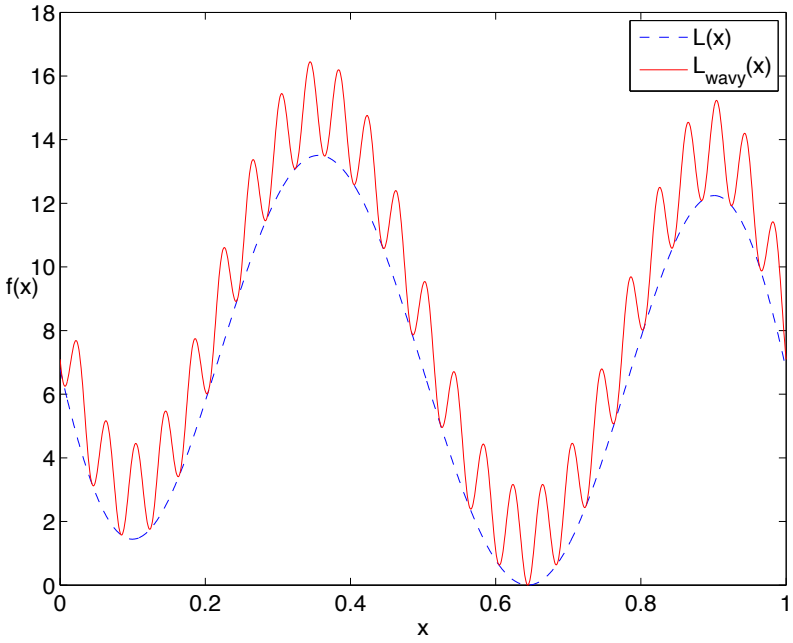


Figure 2.4: LagrangeWavy2

$$f(x) = \frac{\sum_{i=1}^{k} f_i \prod_{1 \le j \le k, j \neq i} \|x - z_j\|^{\alpha_j}}{\sum_{i=1}^{k} \prod_{1 \le j \le k, j \neq i} \|x - z_j\|^{\alpha_j}}, \; x \in [0,\, 1]^d, \tag{2.7.5}$$

where $k \in \mathbb{N}$, $z_i \in [0,1]^d$, $f_i \in \mathbb{R}$, and $\alpha_i \in \mathbb{R}^+ \, \forall i = 1, ..., k$. Given the constants z_i, f_i, α_i , this class of functions has the following properties:

1. $f(z_i) = f_i \, \forall i = 1, ..., k$,

2. $\min_{1 \le i \le k} f_i \le f(x) \le \max_{1 \le i \le k} f_i \, \forall x \in [0,\, 1]^d$, and

3. if $\alpha_i > 1$, $\lim_{x \to z_i} \nabla f(x) = 0 \, \forall i = 1, ..., k$.

Note that (1) and (2) imply that the global minimum value is $\min_{1 \le i \le k} f(z_i)$. Moreover, (3) implies that $z_1, ..., z_k$ are stationary points.

Figure 2.5 shows a 3D-contour plot of a two-dimensional Schoen function (left) and its corresponding SchoenWavy function (right). We can see that adding the wavy part to the underlying Schoen function makes the problem more difficult to solve because many local

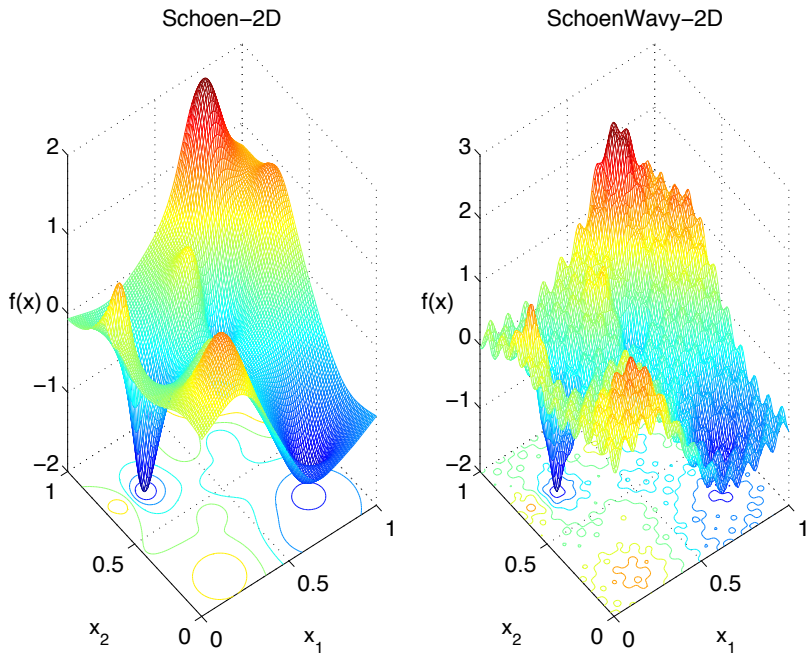


Figure 2.5: SchoenWavy-2D

minima have been added and they are multidimensional. The description of SchoenWavy-2D can be found in Table 2.8.

2.8 Numerical Experiments of Wavy Test Functions

Multistart methods spend a relatively large number of function evaluations in each local search phase. With a limited number of function evaluations, SOMS (as well as any alternative methods) is not expected to work well on general synthetic test problems that have a large number of local minima without a structure of the underlying function f_u. On the other hand, SOMS is expected to perform especially well compared to the alternatives on the class of wavy test functions when the underlying f_u has a moderate number of local minima. Nevertheless, with a moderate number of local minima in the underlying function f_u, we see, for example in Figure 2.5, that adding a wavy part results in a highly multimodal objective function.

Table 2.8: Low-dimensional wavy test functions. A_1 and w_j are defined following Eq. (2.7.2)–Eq. (2.7.4).

Wavy Test Function	f_u	d	A_1	ω_j	**Domain**
easySquareWavy	convex	1	0.05	30π	[0, 1]
Wavy-1D	-*	1	-	-	$[-20\,, 60]$
LagrangeWavy1	multi**	1	1.8	30π	[0, 1]
LagrangeWavy2	multi	1	1.5	50π	[0, 1]
SchoenWavy-2D	multi	2	0.05	20π	$[0, 1]^2$
SchoenWavy-5D	multi	5	0.05	12π	$[0, 1]^5$

* Wavy-1D has no f_u; ** multi = multimodal

2.8.1 Wavy Test Functions

Table 2.8 gives details of six low-dimensional wavy test functions we will use to test the performance of the multistart methods. The characteristics of the underlying function f_u are shown in the column f_u (convex or multimodal). d denotes the problem dimension. The parameters A and ω_j for the wavy part of each test function (defined following Eq. (2.7.2)–Eq. (2.7.4)) are also shown.

The first example in the table, easySquareWavy, is the example given at the end of Section 2.7.1. Wavy-1D was introduced at the beginning of Section 2.7. Since the function Wavy-1D is by itself wavy (i.e. was not defined via our scheme Eq. (2.7.2)–Eq. (2.7.4) of adding the wavy part to an underlying function), the columns A and ω_j for this test function are left blank. The next two functions are based on the same underlying Lagrange polynomial (Section 2.7.2.1). The last two wavy functions are based on the underlying Schoen function (Section 2.7.2.2) with dimensions 2 and 5, respectively.

We also create 10-, 15- and 20-dimensional problems based on the Schoen underlying function. Table 2.9 gives a summary of the high-dimensional Schoen functions (Table 2.9a) along with their wavy functions, called SchoenWavy (Table 2.9b), that will be used to test the effectiveness and efficiency of the multistart methods.

Table 2.9: High-dimensional Wavy Test Functions. (a) Schoen and (b) SchoenWavy.

(a) Schoen function

Function f_u	d	**Domain**
Schoen-10D	10	$[0, 1]^{10}$
Schoen-15D	15	$[0, 1]^{15}$
Schoen-20D	20	$[0, 1]^{20}$

(b) SchoenWavy function. A_1 and w_j are defined following Eq. (2.7.2)–Eq. (2.7.4).

Function	**Description**
SchoenWavy-10D	$A_1 = 0.05$, $\omega_j = c_j\pi$ $c = [13, 12, 11, 13, 14, 17, 16, 17, 11, 20]$
SchoenWavy-15D	$A_1 = 0.1$, $\omega_j = c_j\pi$ $c = [20, 15, 20, 15, 20, 20, 15, 20, 15, 20, 20, 15, 20, 15, 20]$
SchoenWavy-20D	$A_1 = 0.05$, $\omega_j = c_j\pi$ $c = [13, 12, 11, 13, 14, 17, 16, 17, 11, 20, 13, 11, 19, 14, 12, 13, 12, 11, 13, 14]$

Note that all the underlying functions f_u in Table 2.9a and hence the corresponding wavy functions in Table 2.9b are multimodal.

2.8.2 Results for Low-dimensional Wavy Test Functions

We do 30 trials with each algorithm for the six low-dimensional wavy test function in Table 2.8. We impose the maximum number of function evaluations m_f on each of these test functions, and again this number is used to calculate the statistics for those trials which failed to determine the global minimum within m_f evaluations.

The results for the low-dimensional wavy test functions are reported in Table 2.10, where the average and the standard deviation of the number of function evaluations required to detect the global minimum are summarized. The number of failed trials is given in Table 2.10c. A box plot summarizing the statistics is shown in Figure 2.6.

From Figure 2.6, we see that SOMS performs significantly better than the four alternative methods for all but one of the six test functions. It required a whole lot fewer number of function evaluations to locate the location of the global minimum (efficiency) with SOMS versus the other methods. Also the number of trials that SOMS failed to locate the global minimum within the budget m_f is also smallest, which measures effectiveness (see bottom Figure 2.6) among all the algorithms. The one case that SOMS did not perform best is

LgWavy2 test function where 4 trials failed to locate the global minimum within 200 function evaluations.

SOMS performs better than the alternative methods in particular for the 2- and 5-dimensional SchoenWavy test problems, for which the alternative methods failed to locate the global minimum for more than half of the trials (SOMS only failed for one trial of the 2-dimensional problem and for no trials of the 5-dimensional problem).

Many trials of MATLAB's MultiStart solver and the GLOBAL performed poorly for the wavy test functions. This contrasts the results for the Dixon-Szegö testbed for which MultiStart and GLOBAL were able to detect the global minimum for all trials within m_f evaluations. For the wavy test functions, MultiStart and GLOBAL failed to find the global minimum for many trials.

The box plots in Figure 2.6 summarizes these results. It can be clearly seen that SOMS requires on average fewer function evaluations than the alternative methods to detect the global minimum (asterisks). The barplot at the bottom of the same figure also indicates that the SOMS trials succeeded in getting within the radius of $d \times 10^{-4}$ of the global optimum much more frequently than any other methods.

Similar to Dixon-Szegö, a one-tailed two sample t-test for low-dimensional wavy test functions has been conducted at the 5% significance level and reported in Table 2.11 to determine whether SOMS is better than the compared multistart method in terms of the number of function evaluations required to identify the global minimum. The result is 1 if the test rejects the null hypothesis (i.e. SOMS is superior) at the 5% significance level, and 0 otherwise. SOMS is found to be superior to MLSL, MultiStart, and GLOBAL for all but one test function. SOMS is significantly better than GlobalSearch for all low-dimensional wavy test functions.

Table 2.10: Results for low-dimensional wavy test functions. Shown is the (a) average number and (b) standard deviation of function evaluations over 30 trials until the algorithm locates the global minimum or the total number of function evaluations exceeds m_f. (c) shows the number of trials for which each algorithm failed to find the global minimum within m_f function evaluations.

(a) average number

Method	easySquareWavy	Wavy-1D	LgWavy1	LgWavy2	SchWavy-2D	SchWavy-5D	**Total**
	$m_f = 100$	$m_f = 100$	$m_f = 100$	$m_f = 200$	$m_f = 250$	$m_f = 800$	
MLSL	56.23	55.97	60.57	**91.67**	167.47	574.60	1006.5
SOMS	**16.03**	**35.63**	**33**	109.60	**95.57**	**311.07**	**600.9**
MS	58.73	58.10	77.73	106.63	203.37	641.47	1146.03
GS	21.50	72.30	89.23	181.17	223.30	722.80	1310.3
GLOBAL	60.10	66.30	62.13	99.67	156.20	597.20	1041.6

(b) standard deviation

Method	easySquareWavy	Wavy-1D	LgWavy1	LgWavy2	SchWavy-2D	SchWavy-5D
MLSL	33.75	22.32	26.61	56.95	89.28	265.64
SOMS	5.64	6.79	19.97	62.90	63.92	222.03
MS	33.54	27.10	28.77	65.86	71.46	254.97
GS	3.04	32.65	27.25	47.27	56.09	200.37
GLOBAL	29.83	19.99	27.68	58.45	96.56	257.70

(c) number of failed trials (out of 30 total trials)

Method	easySquareWavy	Wavy-1D	LgWavy1	LgWavy2	SchWavy-2D	SchWavy-5D	**Total No. of Failed Trials**
MLSL	9	1	3	2	14	16	45/180
SOMS	-	-	-	4	1	0/30	5/180
MS	5	5	14	8	18	20	70/180
GS	-	17	25	24	22	24	112/180
GLOBAL	6	4	6	2	12	16	46/180

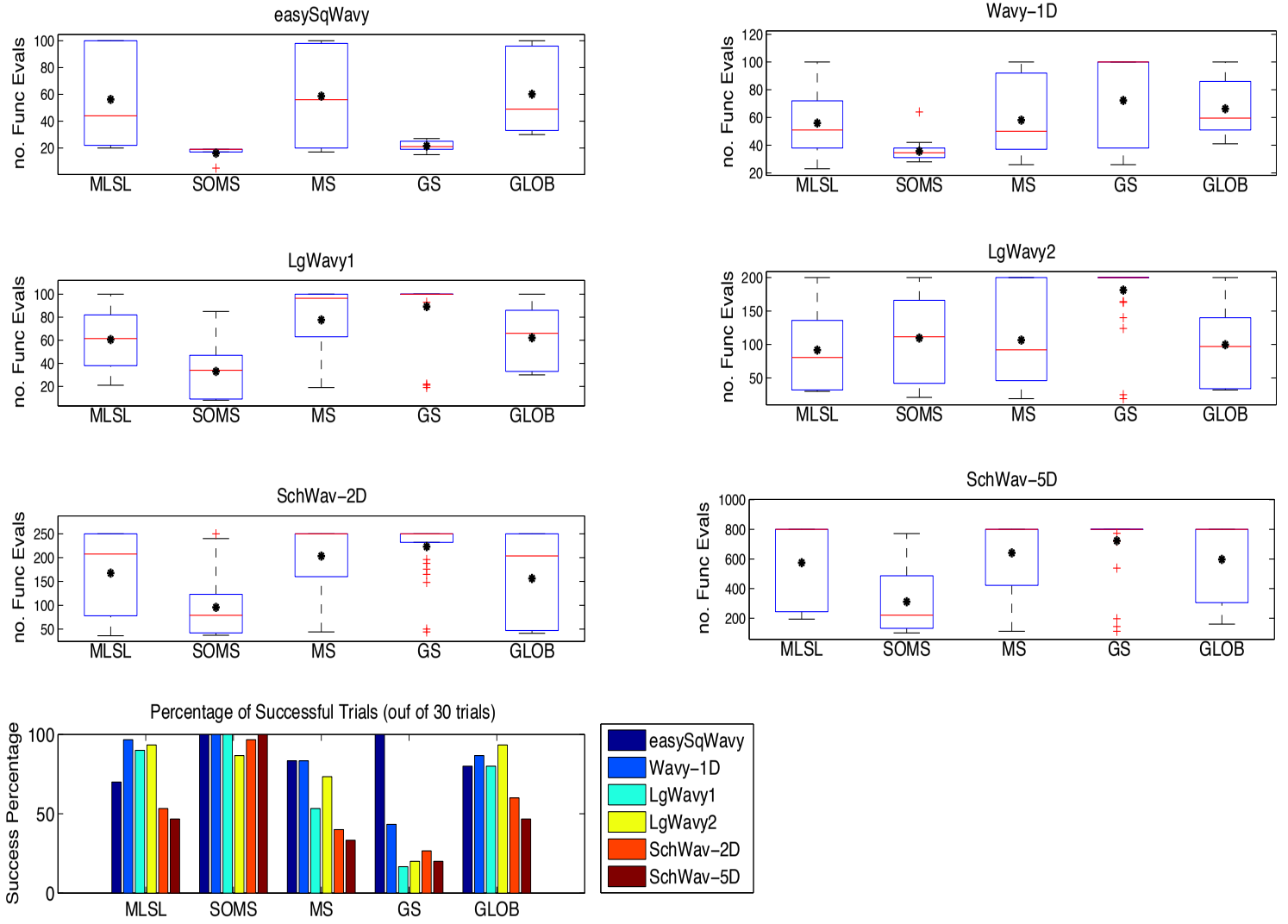


Figure 2.6: Box plots for low-dimensional wavy test functions. Low box plot is best and high bar chart is best.

Table 2.11: One-tailed two sample t-test results for low-dimensional wavy test functions. A value of 1 indicates that SOMS is better at 5% significance level, and a value of 0 indicates that we do not have evidence to suggest that the performance of SOMS and the compared algorithm is different.

Function	SOMS-MLSL	SOMS-MS	SOMS-GS	SOMS-GLOB
easySquareWavy	**1**	**1**	**1**	**1**
Wavy-1D	**1**	**1**	**1**	**1**
LagrangeWavy1	**1**	**1**	**1**	**1**
LagrangeWavy2	0	0	**1**	0
SchoenWavy-2D	**1**	**1**	**1**	**1**
SchoenWavy-5D	**1**	**1**	**1**	**1**
Total	5	5	6	5

2.8.3 Results for High-dimensional Schoen and SchoenWavy Test Functions

In Table 2.12 and Figure 2.7, the results for the high-dimensional Schoen underlying test functions and their corresponding wavy functions are shown. The description of the tables and figures is the same as for the low-dimensional wavy test functions in the previous section.

The left side of Tables 2.12a and b shows the results for the non-wavy Schoen test functions. While all methods are able to locate the global minimum for all trials, we see the superior efficiency and effectiveness of SOMS as it achieved the smallest values for both the average and the standard deviation of the number of function evaluations needed to detect the global minimum. The performance of MLSL and GLOBAL is very similar and both methods outperform MultiStart and GlobalSearch. The statistics are visualized in the leftmost plots of Figure 2.7.

Since all algorithms are able to locate the global minimum of all three non-wavy Schoen test functions within m_f function evaluations, we do not report the number of failed trials in Table 2.12c.

Next, we discuss the performance of algorithms on SchoenWavy test functions. Finding

the global optimum of the wavy Schoen test functions is considerably more difficult. As may be expected, the performance of all algorithms decreased. Table 2.12c shows that SOMS performs best with respect to the number of trials in which the global minimum was detected. SOMS fails only for one trial of SchoenWavy-10D and for none of the other two problems. On the other hand, The alternative algorithms failed for 20–35 out of 90 trials in total. Similar to the results of low-dimensional wavy test functions in the previous section, the performance of MLSL and GLOBAL is very similar and both algorithms outperform MultiStart and GlobalSearch.

Therefore, the numerical results suggest that SOMS performs in general better at detecting the global minima of the wavy functions.

The results of the one-tailed two sample t-test for high-dimensional wavy test functions at the 5% significance level are reported in Table 2.13. SOMS is significantly better than MLSL in determining the global minimum with fewer number of function evaluations for all but one test function. SOMS is found to outperform MultiStart, GlobalSearch, and GLOBAL for all test functions.

Note that SOMS is designed for computationally expensive optimization problems and the performance is assessed based on the objective function value found after a maximal number of allowed function evaluations. In order to get a better idea of the performance for computationally expensive functions, assume that each function evaluation takes one hour. For example, for the three high-dimensional SchoenWavy functions in Table 2.12a, the last column represents the approximate number of hours required on average to detect the global minimum (not counting the algorithms' own computational overheads). We see that the difference in the total wall-clock time of SOMS and the other methods is between 300–1300

Table 2.12: Results for high-dimensional wavy test functions. Shown is the (a) average number and (b) standard deviation of function evaluations over 30 trials until the algorithm locates the global minimum or the total number of function evaluations exceeds m_f whichever comes first. (c) shows the number of trials for which each algorithm failed to find the global minimum within m_f function evaluations.

(a) average number (Left = Schoen, Right = SchoenWavy)

MultiStart Method	Sch-10D $m_f = 1000$	Sch-15D $m_f = 1000$	Sch-20D $m_f = 1000$	**Total Sch**	SchWav-10D $m_f = 1300$	SchWav-15D $m_f = 2700$	SchWav-20D $m_f = 2100$	**Total ShcWav**
MLSL	253.067	427.467	494.7	1175.23	914.467	1367.3	1298.37	3580.13
SOMS	**156.467**	**321.067**	**320.867**	**798.4**	**643.567**	**1305.5**	**933.633**	**2882.7**
MS	280.6	506.467	536.767	1323.83	1120.47	1608.53	1343.03	4072.03
GS	285.8	528.3	536.767	1350.87	1135.53	1694.8	1344.93	4175.27
GLOBAL	271.567	422.767	501.767	1196.1	841.533	1199.57	1176.93	3218.03

(b) standard deviation (Left = Schoen, Right = SchoenWavy)

MultiStart	**Schoen (Sch)**			**SchoenWavy (SchWav)**		
Method	**10D**	**15D**	**20D**	**10D**	**15D**	**20D**
MLSL	35.3387	81.1787	73.0206	372.308	780.107	493.384
SOMS	41.2417	63.0681	58.1839	303.43	625.173	391.052
MS	63.0296	146.384	85.2278	309.756	767.163	484.242
GS	82.0704	181.509	85.2278	294.16	775.605	487.242
GLOBAL	57.5468	100.622	80.5818	407.801	635.891	433.453

(c) number of failed trials (out of 30 total trials)*

MultiStart	**SchoenWavy (SchWav)**			**Total No. of**
Method	**10D**	**15D**	**20D**	**Failed Trials**
MLSL	11	5	4	20/90
SOMS	1	-	-	1/90
MS	21	8	6	35/90
GS	18	7	8	33/90
GLOBAL	12	3	5	20/90

*Note that since all algorithms are able to locate the global minimum of all the three Schoen (non-wavy) functions, we do not include the results of Schoen test function in Table 2.12c.

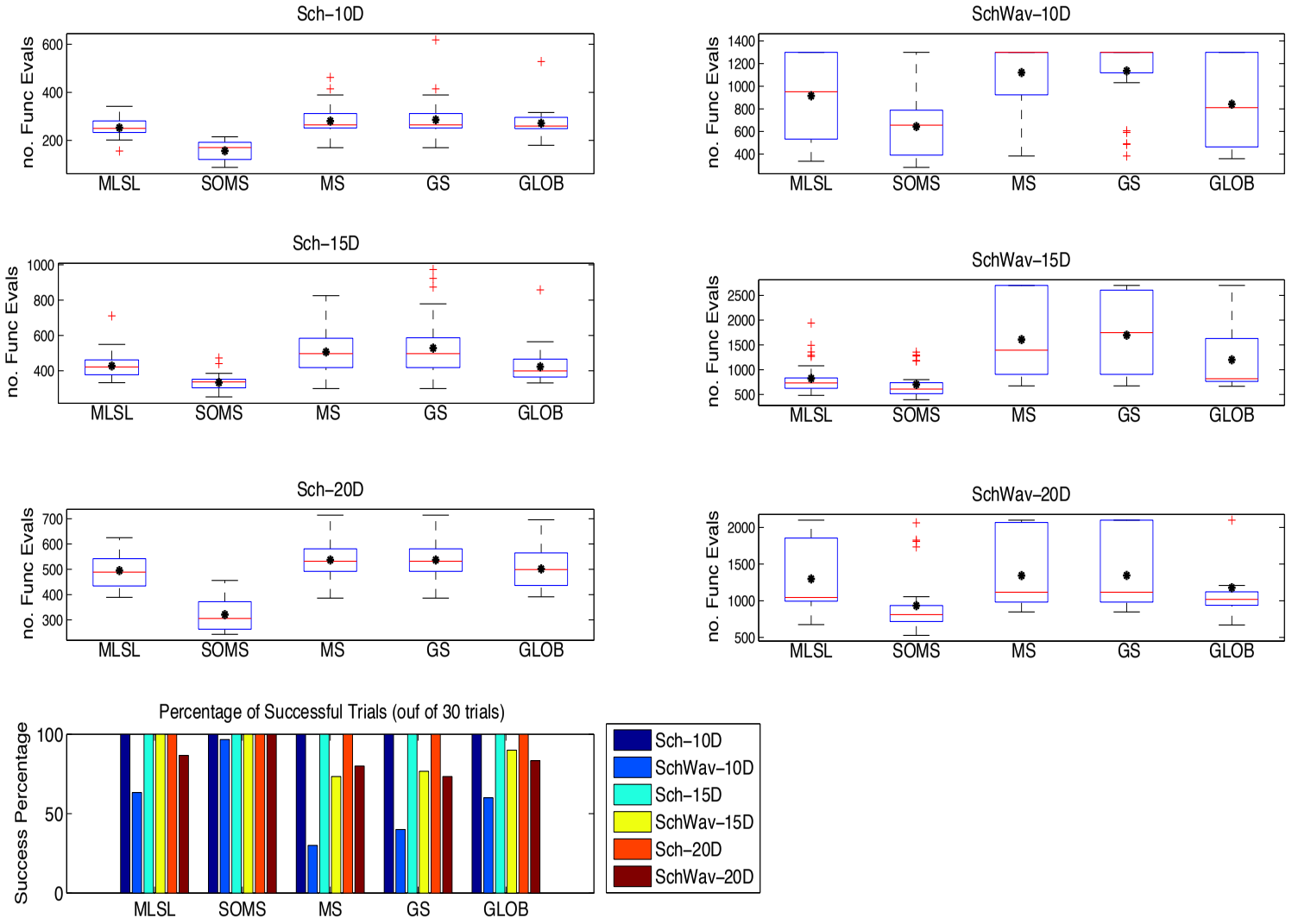


Figure 2.7: Box plots for high-dimensional wavy test functions. Low box plot is best and high bar chart is best.

Table 2.13: One-tailed two sample t-test results for high-dimensional wavy test functions. A value of 1 indicates that SOMS is better at 5% significance level, and a value of 0 indicates that we do not have evidence to suggest that the performance of SOMS and the compared algorithm is different.

Function	SOMS-MLSL	SOMS-MS	SOMS-GS	SOMS-GLOB
Sch-10D	**1**	**1**	**1**	**1**
SchWav-10D	**1**	**1**	**1**	**1**
Sch-15D	**1**	**1**	**1**	**1**
SchWav-15D	0	**1**	**1**	**1**
Sch-20D	**1**	**1**	**1**	**1**
SchWav-20D	**1**	**1**	**1**	**1**
Total	5	6	6	6

hours. Therefore, besides being more effective (fewest total number of failed trials), we see that SOMS is much more efficient than the other multistart methods and a considerable amount of computation time can be saved by using SOMS.

2.8.4 Solution Accuracy of SOMS

In this section we briefly justify the performance measures we used for reporting the computational results. An alternative way to report the results of global optimization algorithms is to plot the best average objective function value against the number of objective function evaluations performed (e.g. [95]). Such "*progress plots*" fail however to show the accuracy of the detected global minimum point.

For example in Figure 2.8, we show the solution found by Stochastic RBF [95] for the LagrangeWavy2. Stochastic RBF is designed to efficiently find decreases of the objective function value within a very limited number of function evaluations. Stochastic RBF is a surrogate-based global optimization algorithm that does not use the multistart procedure shown in Algorithm 2.1. Unlike multistart methods, Stochastic RBF was not designed to find the exact location of the global minimum, but instead, it was designed to quickly decrease the objective function value within a limited number of function evaluations. Once the algorithm terminates, the lowest objective function value will be reported as the approximate global

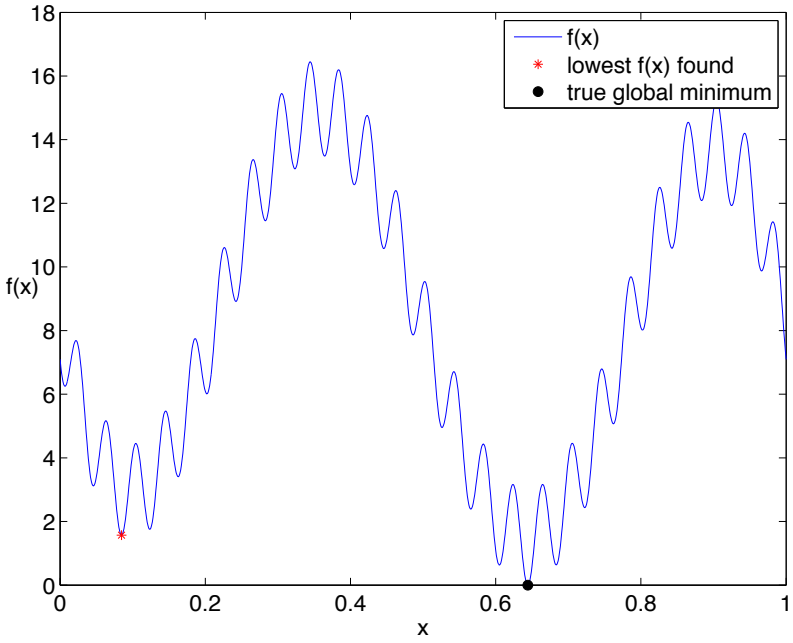


Figure 2.8: The approximate global minimum found by Stochastic RBF algorithm for LagrangeWavy2

minimum value. This same performance measure has been used widely in surrogate global optimization methods.

We did 30 trials with Stochastic RBF for the LagrangeWavy2 problem with $m_f = 200$. Stochastic RBF located the local optimum marked by the red asterisk in Figure 2.8 in all 30 trials (at $(x,\ f(x)) \approx (0.0846,\ 1.5712)$). The true global minimum, which is at $x^* = 0.6442$ with $f(x^*) = 0$, is marked by a black circle. The solution identified by Stochastic RBF does have an objective function value somewhat close to the true global minimum value, but the location of the solution is rather far away from the true global minimum point. The commonly used progress plots do not give any information about the difference between the best point found during the optimization and the location of the true global minimum.

2.9 Conclusions

In this chapter we showed how the combination of a multistart method with a local optimization algorithm can give high accuracy solutions of global optimization problems. Another

advantage of using multistart methods over other types of global optimization algorithms is that it enables users who are currently applying a local optimizer on a complex simulation model to search for a globally optimal solution to relatively easily interface the existing local optimization-simulation code with a multistart method.

We have presented SurrOgate Multistart (SOMS), which is a multistart method for optimization of blackbox functions. SOMS uses the RBF surrogate model to select promising points from which the local search is started. SOMS provides a more efficient multistart method that can reduce the number of local searches required to find the global minimum.

We also proposed a framework for constructing wavy test functions for global optimization algorithms from any underlying function f_u. The constructed wavy test function retains the same global minimum as the underlying function f_u which enables the assessment of the solution quality of the optimization algorithms.

The practical applicability of SOMS is demonstrated by comparing its overall performance with four other multistart methods of similar nature. The results of extensive numerical experiments show that SOMS is an effective and efficient multistart strategy that can escape from local minima and that converges to the global minimum faster than the alternative methods. Numerical experiments show that SOMS statistically outperforms (at 5% significance level) MLSL, MATLAB's MultiStart, MATLAB's GlobalSearch, and GLOBAL for 10, 13, 18, and 14 out of 19 test problems. Hence, using the surrogate model in the multistart framework is a very efficient approach for detecting accurate global optima of multimodal objective functions whose computational expense limits the number of function evaluations that can be done.

Chapter 3

Parallel Dynamic Coordinate Search with Surrogates

3.1 Introduction

Global optimization of computationally blackbox expensive function are important in many application areas. In this chapter, we focus on those problems which also have a high-dimension. Shan and Wang [106] used the acronym HEB (High dimensional, Expensive, and Blackbox) to refer to this type of optimization problems. The combination of these three aspects makes the problem very difficult to solve. There are, however, many engineering application problems that have these characteristics, e.g. the design of the High Speed Civil Transport aircraft [43, 62, 112], optimization of a measure of vibration of a helicopter rotor blade [21]. Better algorithms are important for the high dimensional problems because they arise in many areas of engineering and science. As the computational effort increases exponentially with the problem dimension [106], many available methods for computationally expensive blackbox objective functions are not suitable or even applicable to high-dimensional problems because of the increasing computational overhead of the algorithm itself.

3.1.1 Related Work

When the optimization problem of interest is based on the blackbox simulations, a derivative-free method or a blackbox solver plays an important role in solving problems that cannot be solved otherwise. The increase in the number of blackbox solvers in the literature follows closely the growing number of blackbox optimization problems emerging as applications in engineering. Examples of blackbox solvers are DAKOTA solvers (originally designed as a toolkit of blackbox optimization methods) [35], GLOBAL [30] (a MATLAB implementation of a multistart stochastic method proposed by Boender et al. [20]), HOPSPACK (a library implementing derivative-free direct search algorithms) [85], MADS/NOMAD (Mesh Adaptive Direct Search) [1, 2, 66], and DiceOptim (Kriging-based optimization for computer experiments) [100], as well as many solvers from TOMLAB [49] such as TOMLAB/rbfSolve and TOMLAB/EGO. A good review and comparison of algorithms on blackbox functions are given in [99]. In addition, population-based optimization algorithms such as Genetic Algorithm (GA) [48], Differential Evolution (DE) [113], Evolution Strategies [12] are also often used to solve blackbox optimization problems.

Since simulations in engineering problems can be computationally extremely expensive, several surrogate-based methods have been proposed to solve problems in global optimization. Response surfaces such as polynomial response surface [53], kriging (Gaussian process regression) [54, 57], radial basis functions (RBFs) [15, 89, 95, 96, 107], MARS (Regression Splines), as well as mixture models [76] have been successfully applied to solve many optimization problems arising in engineering from aerospace and aircraft design, hybrid electric car design, groundwater contaminations, watershed calibration, a pharmaceutical formulation system etc. (e.g. [7, 43, 46, 54, 81, 95, 101, 107, 108]). Among many, a few examples of surrogate-based optimization methods are Gutmann's RBF method [45] which involves the minimization of a bumpiness measure to determine the new sample point in each iteration, Jones' EGO method [57], which uses a kriging surface and selects sample points by maximizing the expected improvement, Regis and Shoemaker's LMSRBF method [95] which is also

based on RBF and uses the weighted score of the response surface criterion and the distance criterion, and is used as a framework of DYCORS [97]. The interested reader is referred to [41, 46, 63, 122] for a review and more information as well as applications of existing response surface based methods for global optimization. A recent survey of surrogate-assisted evolutionary optimization techniques can also be found in [55].

While most of the previous surrogate-based optimization methods are suitable and have only been tested on problems with low dimension, DYCORS, on the other hand, was developed especially for problems in a class of HEB problems. DYCORS incorporates an idea from the DDS [115], wherein the next evaluated point is selected from random trial solutions obtained by perturbing only a subset of the coordinates of the current best solution. DYCORS was tested on problems up to 200 dimensions [97].

The progress in parallel computing technologies opened the possibility to develop algorithms for blackbox global optimization problems that exploit the availability of several processors by doing several expensive function evaluations simultaneously. Parallel TRIOPT [127, 128] uses parallel partitioning to divide the problem domain into smaller sub-spaces. Partitions that hold a promise of containing the global optimum are re-partitioned according to different triangular splitting strategies. HOPSPACK is a parallel pattern search package for nonlinear programs [85]. Parallelism in HOPSPACK is done by assigning the individual function evaluations to different processors. There are also several parallel versions of MADS [9]: p-MADS (evaluates the trial points in parallel), COOP-MADS (launches concurrent executions of MADS with different seeds), and PSD-MADS (explores subspaces of variables in parallel). The results in [9] indicate that p-MADS gives in general equivalent results to MADS but is much faster. COOP-MADS is usually more efficient than p-MADS, and PSD-MADS is well suited for larger problems (up to 500 variables) [9]. Regis and Shoemaker [96] also developed a parallel version of LMSRBF (ParLMSRBF) where parallelism is exploited by selecting several new function evaluation points in each iteration and distributing these points for the computationally expensive function evaluation among the available

processors. Viana et al. [120] proposed MSEGO, a method based on EGO [57], which uses multiple surrogates to select several points per iteration for simultaneous evaluation. Finally, in population-based methods, the offspring created in each generation can be distributed to different processors for fitness function evaluations in parallel. Examples of parallel heuristic methods are parallel simulated annealing [80], parallel genetic algorithm [5]. Exploiting parallelism in optimization with computationally expensive function evaluations reduces the total computation time tremendously, and thus in this chapter we develop the new algorithm PADS that exploits multiple processors.

3.1.2 Stochastic Response Surface (SRS) Framework

Regis and Shoemaker [95] introduced a class of stochastic response surface algorithms, called SRS (Stochastic Response Surface) which is a framework for expensive global optimization. Examples of methods based on SRS are LMSRBF [95] and DYCORS [97]. A parallel version of SRS was introduced in [96] where multiple points are selected per iteration and distributed across several processors in order to reduce the overall execution time of the optimization algorithm.

The SRS framework is given in Algorithm 3.1. The iteration of any surrogate-based optimization consists of performing the function evaluations of the points in the initial experimental design and building an RBF model to approximate an expensive objective function. The algorithm then makes use of this computationally inexpensive response surface to determine the location of the next function evaluation point. Finally, the algorithm updates the surrogate after the new evaluated point is added in.

3.1.3 Main Contributions

PADS (**PA**rallel **D**ynamic coordinate search with **S**urrogates) is a global optimization method developed for high-dimensional expensive blackbox functions. As the idea of DDS (to perturb only a subset of the coordinates of the current best solution) has recently been shown

Algorithm 3.1 Stochastic Response Surface (SRS) [95]

Input: initial experimental design

```
1:  build initial response surface
2:  repeat
3:      generate candidate points around x_best;
4:      select a point for function evaluation;
5:      update x_best, the best point found so far;
6:      update response surface;
7:  until termination condition is met
```

Output: the approximate global minimum

to be effective for high-dimensional problems in DYCORS, in this work, we also incorporate the idea of DDS into PADS. However, PADS is more general than DYCORS in that it allows different ways of generating a set of candidate points (from which the next function evaluation points are selected). In addition, instead of simulating one point per iteration as was done in DYCORS, the algorithm is more flexible in that it simulates multiple points per iteration for simultaneous expensive function evaluations.

The purpose of this study is three-fold:

1. In Section 3.2 we develop PADS (**PA**rallel **D**ynamic coordinate search with **S**urrogates), a global optimization framework for HEB problems, which is an extended version of SRS. In each iteration, J new sample points are selected and evaluated by J processors simultaneously. Candidate points of PADS are generated by perturbing only a subset of coordinates of the current best solution as was done in DDE and DYCORS.

2. Two algorithms that follow PADS framework are introduced, namely, PADS1 and PADS2. PADS1 uses a truncated Gaussian distribution while PADS2 uses a standard Gaussian distribution with reflective bound as a perturbation distribution. While a Gaussian distribution has been implemented previously (e.g. [95, 97]), a truncated Gaussian distribution is a new distribution for SRS that has never been implemented in any previous algorithms that follows SRS.

3. In Section 3.3 we prove that PADS converges to the global minimum almost surely.

As PADS can be considered as an extended parallel version (with more flexibility) of DYCORS [97], the convergence of PADS also verifies the convergence of DYCORS, which was not provided in [97]. We compare PADS to other widely used blackbox global optimization algorithms (serial and parallel algorithms) for 30- and 200-dimensional problems in Section 3.5. We compare to the serial methods NOMADm-RBF and NOMADm-DACE (MATLAB implementation of the MADS algorithm coupled with RBF and kriging, respectively), LMSRBF [95], DE [113], ESGRBF [91, 107], and rbfSolve (an RBF based method implemented in Tomlab optimization toolbox for Matlab) [15, 50]. The parallel algorithms included in the comparison are PSD-MADS [9], ParLMSRBF [96], ParDE, and ParESGRBF where at each generation, the expensive objective function evaluations of population are distributed across processors. Good results on seven multimodal test functions, each with 30 and 200 variables illustrate the advantages of our algorithms. The analysis of PADS1 under two measures, namely, the α-Speedup and the α-Work Ratio will also be discussed.

3.2 PADS Framework

Different algorithms use different techniques to select the next evaluation point and update the surrogate. These steps are usually the algorithm's computationally intensive task especially when the problem dimension is high. Most of the surrogate-based algorithms, however, only simulate one expensive function evaluation and update the surrogate immediately after every simulation is done. On the other hand, in each iteration of PADS, the algorithm simulates multiple points for simultaneous function evaluations, and the algorithm updates the surrogate only after function evaluations on all selected points are completed.

3.2.1 PADS(J) Description

The objective function of box-constrained minimization problem is defined in Eq. (1.1). We will use PADS(J) to mean PADS with $J > 0$ function evaluations (simulations) per iteration. PADS(J) requires the input information shown below and the pseudocode is shown in Algorithm 3.2.

Inputs:

- A continuous real-valued function f defined on a hyperrectangle $\mathcal{D} = [\text{lb}, \text{ub}] \subset \mathbb{R}^d$
- The number of expensive function evaluations simulated per iteration J
- A response surface model
- A set of initial evaluation points $\mathcal{S}_0 = \{x_1, ..., x_{n_0}\}$
- The number of candidate points in each iteration, denoted by N_{cand}, where $N_{\text{cand}} \gg J$
- The maximum number of function evaluations allowed, denoted by $N_{\max}$
- The initial step size σ_{init} and the minimum step size $\sigma_{\min}$
- The tolerance for the number of consecutive failed iterations τ_{fail} and the threshold for the number of consecutive successful iterations τ_{success}

Outputs: The best point x_{best} and the corresponding value f_{best}

Below we will give a description of the four functions, namely, $\varphi(n)$ (Step 3b), **Perturb_x** (Step 3(c)ii), **Select_J_Evaluation_Points** (Step 3d), and **Adjust_Step_Size** (Step 3g). Two algorithms PADS1 and PADS2 that follow PADS framework will also be introduced along the way. Note the difference of these two algorithms in **Perturb_x** and **Adjust_Step_Size**. We use the term PADS whenever the description is applied to both algorithms of PADS. We will specifically mention PADS1 or PADS2 if the description only applies to one but not the other.

Algorithm 3.2 PArallel Dynamic coordinate search with Surrogates (PADS)

1. **Initialization.** Set $\sigma_0 \leftarrow \sigma_{\text{int}}$, $C_{\text{fail}} \leftarrow 0$, $C_{\text{success}} \leftarrow 0$.

2. **Evaluation of initial design points.** Do expensive function evaluations on $\mathcal{S}_0 = \{x_1, ..., x_{n_0}\}$, the initial points. Set $n = n_0$. Denote the best point found so far by x_{best}, i.e. $x_{\text{best}} = \text{argmin}_{x \in S_0} f(x)$, and $f_{\text{best}} = f(x_{\text{best}})$.

3. **While stopping criterion not met (while $n < N_{\max}$ is true)**

   (a) Build or update the surrogate model, $s_n(x)$, using all evaluated points in $\mathcal{S}_n$ and their corresponding function values.

   (b) Compute the probability of perturbing a coordinate: $p_{\text{select}} = \varphi(n)$.

   (c) Generate N_{cand} candidate points around x_{best} : $\Omega_n = \{y_{n,1}, ..., y_{n,N_{\text{cand}}}\}$. For $j = 1, ..., N_{\text{cand}}$, each candidate point $y_{n,j}$ is generated by

      i. Select perturbation coordinates: For each coordinate i, generate d uniform random numbers $u_1, ..., u_d \in [0, 1]$. Let $I_{\text{perturb}} = \{i : u_i < p_{\text{select}}\}$ be the set of indices of coordinates of x_{best} to be perturbed. If $I_{\text{perturb}} = \emptyset$, randomly select the coordinate i from $\{1, ..., d\}$ and set $I_{\text{perturb}} = \{i\}$.

      ii. Generate candidate point: $y_{n,j} = \textbf{Perturb\_x}(x_{\text{best}}, I_{\text{perturb}}, \sigma_n, \text{lb}, \text{ub})$

   (d) Select the next J points to evaluate: We select the set of points $\{x_{n+1}, ..., x_{n+J}\}$ from the set of candidate points Ω_n based on the information from the surrogate $s_n(x)$ and from the location of previously evaluated points through the function **Select_J_Evaluation_Points**$(\Omega_n, s_n(x), \mathcal{S}_n)$.

   (e) Perform expensive function evaluations: For each $l = 1, ..., J$, computes $f(x_{n+l})$.

   (f) Update counters: If $\min_{1 \le l \le J} f(x_{n+l}) < f_{\text{best}}$, reset $C_{\text{success}} = C_{\text{success}} + 1$ and $C_{\text{fail}} = 0$; otherwise reset $C_{\text{fail}} = C_{\text{fail}} + 1$ and $C_{\text{success}} = 0$.

   (g) Adjust step size σ_{n+J}: $[\sigma_n, C_{\text{success}}, C_{\text{fail}}] = \ldots$
   Adjust_Step_Size$(\sigma_n, C_{\text{success}}, \tau_{\text{success}}, C_{\text{fail}}, \tau_{\text{fail}})$.

   (h) Update the set of evaluated points: $\mathcal{S}_{n+J} = \mathcal{S}_n \cup \{x_{n+1}, ..., x_{n+J}\}$.

   (i) Reset $n = n + J$, and update the best point x_{best} found and $f(x_{\text{best}}) = \min_{x \in \mathcal{S}_n} f(x)$.

4. Return the best solution found: Return x_{best} and f_{best}.

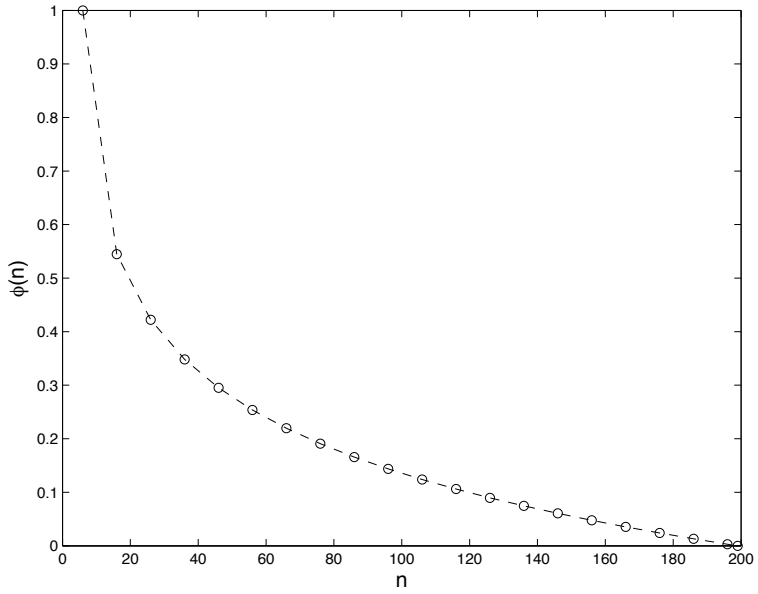


Figure 3.1: Probability $\varphi(n)$ of selecting a variable of x_{best} for perturbation versus the number of function evaluations when $n_0 = 6$ and $N_{\max} = 200$

3.2.1.1 Function $p_{\textbf{select}} = \varphi(n)$

In Step 3b, the probability $\varphi(n)$ of perturbing a coordinate of x_{best} when generating candidate points is defined by

$$\varphi(n) = \varphi_0 \times [1 - \ln(n - n_0 + 1)/\ln(N_{\max} - n_0)], \tag{3.2.1}$$

for all $n_0 \leq n \leq N_{\max} - 1$, where $\varphi_0 = \min(20/d,\, 1)$ as was done in [97]. Thus, if the problems dimension d is larger than 20, the average number of variables being initially perturbed is $d \times (20/d) = 20$. With this definition of $\varphi(n)$, the algorithm searches initially globally because all coordinates of x_{best} are perturbed. As the number of total function evaluations approaches the computational budget, the probability $\varphi(n)$ decreases and fewer variables of x_{best} are perturbed. Thus, towards the end, the algorithm searches locally for improvements. This approach has been shown to improve the solution quickly for HEB problems [97]. Figure 3.1 shows an example of the probability $\varphi(n)$ of perturbing a variable of x_{best} as a function of total function evaluations when $n_0 = 6$ and $N_{\max} = 200$.

3.2.1.2 Function $y_{n,j}$ = Perturb_x(x_{best}, I_{perturb}, σ_n, lb, ub) and two versions of PADS: PADS1 and PADS2

Given x_{best}, I_{perturb}, σ_n, and the domain $\mathcal{D}$ = [lb, ub] where lb = $(l^{(1)}, ..., l^{(d)})$, ub = $(u^{(1)}, ..., u^{(d)}) \in \mathbb{R}^d$, two types of candidate points are introduced in Figure 3.2.

Definition 3.1. We will call the two versions of PADS algorithm based on the type of candidate points generated in **Perturb_x**:

(1) PADS1: the PADS algorithm with truncated Gaussian candidate points

(2) PADS2: the PADS algorithm with Gaussian with reflective bound candidate points.

Function $y_{n,j}$ = **Perturb_x**(x_{best}, I_{perturb}, σ_n, lb, ub)

- generate $y_{n,j} = x_{\text{best}} + z$, where $z^{(i)}$ is a random variable such that:
- **PADS1:** Truncated Gaussian Perturbation
  - $z^{(i)} = \begin{cases} N_{\text{truncated}}(0, \sigma_n^2; a^{(i)}, b^{(i)}) & \text{for } i \in I_{\text{perturb}} \\ 0 & \text{for } i \notin I_{\text{perturb}} \end{cases}$, where
    $a^{(i)} = l^{(i)} - x_{\text{best}}^{(i)}$; $b^{(i)} = u^{(i)} - x_{\text{best}}^{(i)}$.

  (Refer to section 3.2.2 for details about the truncated normal distribution.)
- **PADS2:** Gaussian Perturbation with Reflective Bound
  - $z^{(i)} = \begin{cases} N(0, \sigma_n^2) & \text{for } i \in I_{\text{perturb}} \\ 0 & \text{for } i \notin I_{\text{perturb}} \end{cases}$.
  - If $y_{n,j} \notin \mathcal{D}$, then replace it by a point in $\mathcal{D}$ obtained by performing successive reflection of $y_{n,j}$ about the closest point on the boundary of $\mathcal{D}$.

Figure 3.2: Pseudocode **Perturb_x**

While the Gaussian distribution has been used to generate candidate points in previous algorithms that follow SRS framework [90, 95, 96, 97], the truncated Gaussian distribution is new for SRS framework. In previous versions of SRS, if a Gaussian candidate point $y_{n,j} \notin \mathcal{D}$, the point is either replaced by the nearest point in $\mathcal{D}$ [95, 96] or successive reflected about

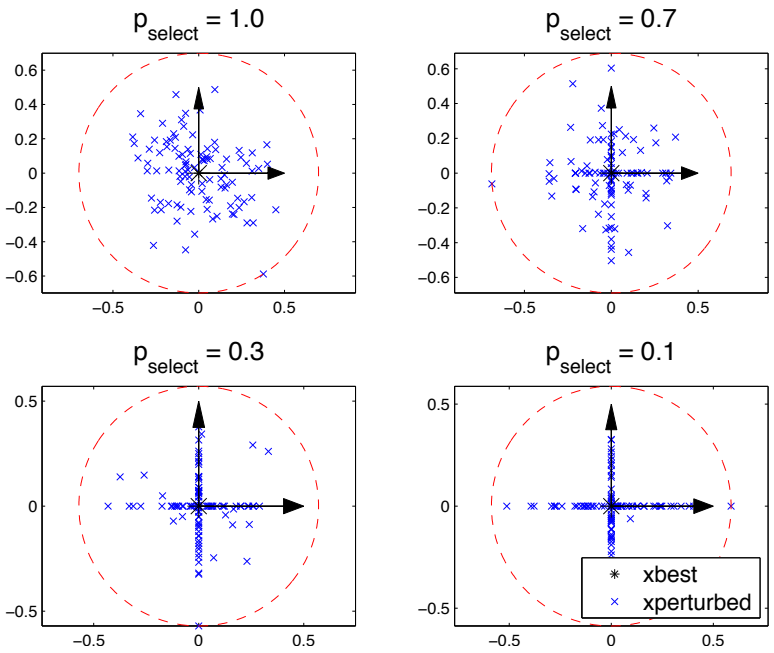


Figure 3.3: Candidate points (×) generated around x_{best} (*) for various values of p_{select}

the closest point on the boundary of $\mathcal{D}$ [97]. Unlike any previous SRS methods, PADS1 generates truncated normal candidate points within the domain, and thus the reflection is not necessary.

Figure 3.3 illustrates the perturbation of x_{best} for various values of p_{select} in a two-dimensional example on the domain $[-1,\ 1]\times[-1,\ 1]$. We see that as the algorithm progresses and p_{select} decreases, fewer coordinates are selected for perturbation. Hence, the search becomes more local.

3.2.1.3 Function $[x_{n+1}, ..., x_{n+J}]$ = Select_J_Evaluation_Points$(\Omega_n,\ s_n(x),\ \mathcal{S}_n)$

The function **Select_J_Evaluation_Points** in Step 3d is based on a weighted score which is used to select the next J evaluation points from the set of candidate points Ω_n. The J points with the best weighted scores of the RBF and the distance criterion are selected for function evaluations. A balance between exploration and exploitation is achieved by cycling through a set of weights for these two criteria. This weight cycling method is a key component

of the SRS method [95] and was used in subsequent papers by Regis and Shoemaker [90, 96] including DYCORS-LMSRBF [97]. The weight pattern for the RBF criterion is denoted by $\Upsilon =< v_1, ..., v_\kappa >$, where $0 \leq v_1 \leq \cdots \leq v_\kappa \leq 1$. If n is the number of previously evaluated points, the weights w_n^R and w_n^D for the RBF and the distance criterion, respectively, are defined for $n \geq n_0$ by

$$w_n^R = \begin{cases} v_{\mathrm{mod}(n-n_0+1,\,\kappa)} & \text{if} \quad \mathrm{mod}(n-n_0+1,\,\kappa) \neq 0 \\ v_\kappa & \text{otherwise} \end{cases}, \text{ and } w_n^D = 1 - w_n^R.$$

As in [96], given n previously evaluated points $S_n = \{x_1, ..., x_n\}$ and the weights w_n^R and w_n^D, the function **Select_J_Evaluation_Points** described in Figure 3.4 selects $\{x_{n+1}, ..., x_{n+J}\}$ for evaluation in parallel. When $J > 1$, each of the J points $\{x_{n+j} : j = 1, .. J\}$ within the current iteration is selected sequentially so that each point x_{n+j} is far away from both previous evaluated points $S_n = \{x_1, ..., x_n\}$, and the $j-1$ already selected points in the current iteration, $\{x_{n+1}, ..., x_{n+j-1}\}$. When $J = 1$, the scheme **Select_J_Evaluation_Points** selects one point for function evaluation, which coincides with the function **Select_Evaluation_Point** of DYCORS [97].

Function $[x_{n+1}, ..., x_{n+J}]$ = **Select_J_Evaluation_Points**(Ω_n, $s_n(x)$, $\mathcal{S}_n$) [95, 96]

1. Estimate function value of candidate points: For any point $y \in \Omega_n$, compute the surrogate value $s_n(y)$, as well as $s_n^{\max} = \max\{s_n(y) : y \in \Omega_n\}$ and $s_n^{\min} = \min\{s_n(y) : y \in \Omega_n\}$.

2. Compute score based on RBF surrogate value: For any point $y \in \Omega_n$, compute $V_n^R(y) = (s_n(y) - s_n^{\min})/(s_n^{\max} - s_n^{\min})$ if $s_n^{\max} \neq s_n^{\min}$ and $V_n^R(y) = 1$ otherwise.

3. Select the J evaluation points sequentially: For $j = 1, ... J$, do

   (a) Calculate minimum distance from previously evaluated points and previously selected points: For any point $y \in \Omega_n$, compute $D_{n+j-1}(y) = \min\{\|x - y\| : x \in \mathcal{S}_n \cup B_j\}$, where $\|\cdot\|$ is the Euclidean norm and $B_1 = \emptyset$, $B_j = \{x_{n+1}, ..., x_{n+j-1}\}$ for $j = 2, ..., J$. Note that B_j contains previously selected points within the current iteration. Also, compute $D_{n+j-1}^{\max} = \max\{D_{n+j-1}(y) : y \in \Omega_n\}$ and $D_{n+j-1}^{\min} = \min\{D_{n+j-1}(y) : y \in \Omega_n\}$.

   (b) Compute score based on minimum distance: For any point $y \in \Omega_n$, compute $V_{n+j-1}^D(y) = (D_{n+j-1}^{\max} - D_{n+j-1}(y))/(D_{n+j-1}^{\max} - D_{n+j-1}^{\min})$ if $D_{n+j-1}^{\max} \neq D_{n+j-1}^{\min}$ and $V_{n+j-1}^D(y) = 1$ otherwise.

   (c) Compute weighted score: For any point $y \in \Omega_n$, compute $W_{n+j-1}(y) = w_{n+j-1}^R V_n^R(y) + w_{n+j-1}^D V_{n+j-1}^D(y)$.

   (d) Select the next evaluation point: $x_{n+j} = \arg\min_{y \in \Omega_n} W_{n+j-1}$.

Figure 3.4: Pseudocode **Select_J_Evaluation_Points**

3.2.1.4 Function $[\sigma_n, C_{\textbf{success}}, C_{\textbf{fail}}]$ = Adjust_Step_Size(σ_n, $C_{\textbf{success}}$, $\tau_{\textbf{success}}$, $C_{\textbf{fail}}$, $\tau_{\textbf{fail}}$)

We now define the function **Adjust_Step_Size** (Step 3g). When $C_{\text{success}} \geq \tau_{\text{success}}$, different schemes are used to update σ_n in PADS1 and PADS2.

Function $[\sigma_n, C_{\text{success}}, C_{\text{fail}}] =$ **Adjust_Step_Size**$(\sigma_n, C_{\text{success}}, \tau_{\text{success}}, C_{\text{fail}}, \tau_{\text{fail}})$

- If $C_{\text{fail}} \geq \tau_{\text{fail}}$, then
  - $\sigma_{n+1} = \max(\sigma_n/2, \sigma_{\min})$, and
  - $C_{\text{fail}} = 0$.

  End
- If $C_{\text{success}} \geq \tau_{\text{success}}$, then
  - $\sigma_{n+1} = \begin{cases} \min(2\sigma_n, \sigma_{\text{init}}) & \text{for PADS1} \\ 2\sigma_n & \text{for PADS2} \end{cases}$, and
  - $C_{\text{success}} = 0$.

  End

Figure 3.5: Pseudocode **Adjust_Step_Size**

The current step size σ_n for both PADS1 and PADS2 is reduced by half and C_{fail} is reset to 0 whenever C_{fail} reaches τ_{fail}. This step size is decreased to facilitate convergence and the minimum step size $\sigma_{\min}$ prevents the candidate points from getting too close to x_{best}. On the other hand, the current step size σ_n is doubled and C_{success} is reset to 0 whenever C_{success} reaches τ_{success}. The step size is doubled to accelerate the search in the domain, which was proven to be effective [90, 95, 96, 97].

Note that the bound for PADS2 is defined to be the same as that of DYCORS. However, for PADS1, we also impose an upper bound σ_{init} so that the step size used to perturb x_{best} does not become unnecessarily large.

Remark 3.2. When $J = 1$, PADS2(1) coincides with DYCORS-LMSRBF in [97].

3.2.2 Truncated Gaussian Distribution

The truncated Gaussian distribution (or the truncated normal distribution) is the probability distribution of a normally distributed random variable whose value is bounded in the domain $[a, b]$, where $-\infty < a < b < \infty$.

Suppose $X \sim N(\mu,\, \sigma^2)$ has a normal distribution. Then, X conditional on $a \leq X \leq b$ has a truncated normal distribution with the probability density function given by [44]

$$f(x;\, \mu,\, \sigma,\, a,\, b) = \begin{cases} \frac{\frac{1}{\sigma}\phi(\frac{x-\mu}{\sigma})}{\Phi(\frac{b-\mu}{\sigma})-\Phi(\frac{a-\mu}{\sigma})} & ;\, a \leq x \leq b \\ 0 & ;\, \text{otherwise} \end{cases} \tag{3.2.2}$$

where $\phi(z) = \frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}z^2)$ is the probability density function of the standard normal distribution $N(0,\, 1)$ and Φ is its corresponding cumulative distribution function. We denote this distribution by $N_{\text{truncated}}(\mu,\, \sigma^2;\, a,\, b)$.

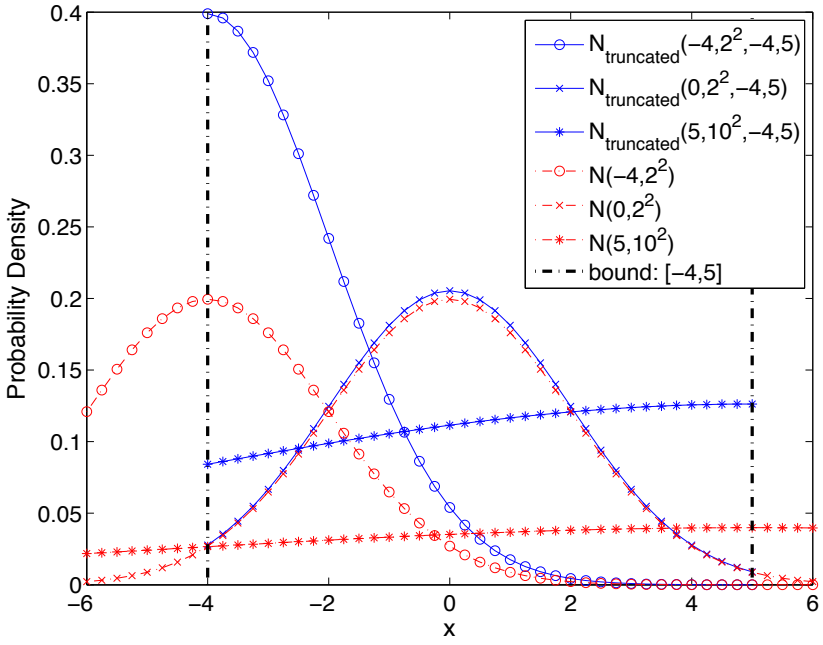


Figure 3.6: Comparison of Gaussian and truncated Gaussian distributions

Figure 3.6 gives various examples of the probability density function of the truncated Gaussian distribution and the Gaussian distribution. With a fixed $(\mu,\, \sigma^2)$ values, we see that the two distributions are very different especially when σ^2 is large relative to the bound ($\sigma^2 = 10^2$) or when μ does not fall towards the middle of the bound ($\mu = -4$). In PADS2 and DYCORS-LMSRBF, if a generated point falls into the area outside the bound $[-4,\, 5]$, the

algorithm will successively reflect the point about the closest boundary. On the other hand, PADS1 generates a point according to a truncated Gaussian distribution, the generated point will always fall into the bound $[-4, 5]$.

3.3 Convergence of PADS Methods

With some technical conditions for the algorithm parameters, it has been shown that LM-SRBF converges to the global minimum almost surely [95]. In this section we will prove the global convergence of the PADS(1) algorithm. The method can be extended to prove the convergence of PADS(J) when $J > 1$.

To prove the convergence of serial PADS(1), we will apply the theorem that was proved in [95]. Using the notation introduced in [95], we will show that PADS(1) satisfies all conditions required for the theorem.

Definition 3.3. For $1 \leq j \leq N_{\text{cand}}$ and $n \geq n_0$,

$Y_{n,j}$ is the random vector representing the random candidate point $y_{n,j}$ before it was forced to be in the domain $\mathcal{D}$ (See function **Perturb_x** in Figure 3.2).

$Y_{\mathcal{D}}$ is a transformation of a random vector Y (whose realization is in $\mathbb{R}^d$) so that $Y_{\mathcal{D}}$ is always in $\mathcal{D}$. i.e. $\mathrm{T} : \mathbb{R}^d \to \mathcal{D}$ is a deterministic function and $Y_{\mathcal{D}} = \mathrm{T}(Y) \in \mathcal{D}$.

X_{n+1} is the random vector representing the $(n+1)$th function evaluation point x_{n+1}.

Note that the generated candidate points of PADS1 are always in the domain $\mathcal{D}$, and thus $Y_{n,j} = (Y_{n,j})_{\mathcal{D}}$ in this case. Therefore, the transformation $Y_{\mathcal{D}}$ defined above in fact occurs only in PADS2.

Fix $n \geq n_0$, and let x_n^* be the best point found so far (the point with the lowest objective function value). Recall that in PADS, each coordinate of x_n^* has a probability $\varphi(n) \in [0, 1]$ to be perturbed and any candidate point y is generated by perturbing a subset of the variables of

x_n^*. In case no variable of x_n^* is selected for perturbation with $\varphi(n)$, one variable is randomly chosen, and thus at least one variable of x_n^* will always be perturbed to obtain y.

Let Q be a random vector in $\{0,\, 1\}^d \setminus \{(0, ..., 0)\}$ that determines which coordinates of x_n^* will be perturbed to obtain y, i.e. $Q(j) = 1$ if the jth coordinate of x_n^* is selected to perturb, and $Q(j) = 0$ otherwise. Then, Q can be modeled as follows through the random vectors R and E:

Define $R = (R^{(1)}, ..., R^{(d)})$ to be a random vector in $\{0,\, 1\}^d$ that follows the multivariate Bernoulli distribution with parameter p, i.e. each coordinate i of R is such that $P(R^{(i)} = r) = p^r(1-p)^{1-r}$, $r \in \{0,\, 1\}$. If at least one of the coordinates is selected for perturbations, then the distribution of Q will simply follow R. However, in the case that none of the coordinates are being selected, then the algorithm will uniformly pick exactly one of the coordinates for a perturbation. We shall introduce another random vector, called $E = (E^{(1)}, ..., E^{(d)})$, to capture the latter scenario.

Let $e_i = (0,\, 0, ..., 0,\, 1,\, 0, ..., 0)$ be the ith row of the $d \times d$ identity matrix. Then, E is a random vector such that $P(E \in \cup_{i=1}^{d}\{e_i\}) = 1$ and $P(E = e_i) = 1/d$ for $i = 1, .. d$.

Combining the two random vectors R and E gives us a correct representation of Q :

$$Q = R1_{\left\{R \neq \vec{0}\right\}} + E1_{\left\{R = \vec{0}\right\}}, \tag{3.3.1}$$

where for any set A,

$$1_A(x) = \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{if } x \notin A \end{cases} \tag{3.3.2}$$

is an indicator function and $\vec{0}^{(i)} = 0$ for all $i = 1, ... d$.

Definition 3.4. For $1 \leq j \leq N_{\text{cand}}$ and $n \geq n_0$, let random vectors $Q_{n,\,j}$, $R_{n,\,j}$, $E_{n,\,j}$ have the same distributions corresponding to Q, R and E defined above where the parameter $p = \varphi(n) > 0$. Then, $Q_{n,\,j}$ is the random vector that determines which coordinates of X_n^* are chosen for a perturbation to obtain $Y_{n,\,j}$, where X_n^* is the random vector representing x_n^*.

For each $n \geq n_0$, let

- $\mathcal{F}_n := \{X_1, .., X_{n_0}, Y_{n_0,1}, ..., Y_{n_0,N_{\text{cand}}}, ..., Y_{n,1}, ..., Y_{n,N_{\text{cand}}}\}$,
- $\mathcal{Q}_n := \{Q_{n_0,1}, ..., Q_{n_0,N_{\text{cand}}}, ..., Q_{n,1}, ..., Q_{n,N_{\text{cand}}}\}$.

Then, define

- $\mathcal{E}_n = \mathcal{F}_n \cup \mathcal{Q}_n$ for $n \geq n_0$, and $\mathcal{E}_{n_0-1} := \{X_1, ..., X_{n_0}\}$.

So $\mathcal{F}_n$ is the set of points that were used to build the initial surrogate model for PADS(1) (in Step 2) and all candidate points generated (in Step 3c of Algorithm 3.2) in all iterations up to n. $\mathcal{Q}_n$ is the set of vectors describing which coordinates of X_n^* are perturbed to obtain each of the candidate point $Y_{n,j}$ in all iterations up to n.

Remark 3.5. In PADS(1), for each $n > n_0$, the value of X_n is selected deterministically from the values of the random vectors $(Y_{n-1,1})_{\mathcal{D}}, ..., (Y_{n-1,N_{\text{cand}}})_{\mathcal{D}}$ (see Step 3d of Algorithm 3.2). Therefore, after the nth function evaluation, the entire path of the algorithm is completely determined by $\sigma(\mathcal{E}_{n-1})$, the $\sigma - algebra$ generated by the random vectors in $\mathcal{E}_{n-1}$.

To show that PADS(1) converges to the global minimum almost surely, we will apply the following theorem which was presented in [95].

Theorem 3.6. *Let f be a function defined on $\mathcal{D} \subseteq \mathbb{R}^d$ and suppose that $x^* = \min_{x \in \mathcal{D}} f(x) > -\infty$ is the unique global minimizer of f in $\mathcal{D}$ such that $\min_{x \in \mathcal{D}, \|x - x^*\| \geq \eta} f(x) > f(x^*)$ for all $\eta > 0$. Suppose further that the SRS method generates the random vectors $\{X_n\}_{n \geq 1}$ and $\{Y_{n,1}, ..., Y_{n,N_{cand}}\}_{n \geq n_0}$ satisfying the following two conditions:*

[C1] For each $n \geq n_0$, $Y_{n,1}, ..., Y_{n,N_{\text{cand}}}$ are conditionally independent given the random vectors in $\mathcal{E}_{n-1}$.

[C2] For any $j = 1, ..., N_{\text{cand}}$, $x \in \mathcal{D}$ and $\delta > 0$, there exists $\nu_j(x, \delta) > 0$ such that

$$P[Y_{n,j} \in B(x, \delta) \cap \mathcal{D} | \sigma(\mathcal{E}_{n-1})] \geq \nu_j(x, \delta)$$

for all $n \geq n_0$, where $B(x, \delta)$ is the open ball of radius δ centered at x.

If the sequence of random vectors $\{X_n^*\}_{n\geq 1}$ is defined by $X_1^* = X_1$ and

$$X_n^* = \begin{cases} X_n & \text{if } f(X_n) < f(X_{n-1}^*) \\ X_{n-1}^* & \text{otherwise} \end{cases},$$

then $X_n^* \xrightarrow{a.s.} x^*$.

Proof. Replacing $\mathcal{E}_n$ defined in [95] with our (larger) $\mathcal{E}_n$ defined in Definition 3.4 and by a straightforward replication of the proof in [95], this theorem also holds for the PADS(1) framework. □

To apply this theorem, one needs to show that PADS(1) satisfies conditions **[C1]** and **[C2]** of Theorem 3.6. The condition **[C1]** is trivial. The condition **[C2]** says that the algorithm is able to sample points in any region of $\mathcal{D}$. The following two lemmas will show that PADS(1) indeed satisfies **[C2]**.

Lemma 3.7. *For a fixed $j \in \{1, \ldots N_{cand}\}$, let H be the event that all coordinates of x_n^* are selected for perturbation (to generate $y_{n,j}$), i.e. $H = \{R_{n,j}(i) = 1 \text{ for all } i = 1, ..., d\}$. Let $g_{n,j}$ be the conditional density of $Y_{n,j}$ given $\sigma(\mathcal{E}_{n-1})$ and H. Then, there is a constant $C > 0$ such that $g_{n,j}(y) \geq C$ for all $y \in \mathcal{D} = [lb, ub] \subset \mathbb{R}^d$, and $n \geq n_0$.*

Proof. First note that $Y_{n,j}$ is a random vector before a candidate point is forced into the domain $\mathcal{D}$ (see Definition 3.3). Under the assumption that all coordinates of x_n^* are selected for perturbation and that all the information up to function evaluation $n-1$ are known, the conditional density $g_{n,j}$ of each version of PADS(1) can be written in one of the following forms:

PADS1(1): $g_{n,j}(y) = A_1 \exp\left\{\frac{-\|y-x_n^*\|^2}{2\sigma_n^2}\right\}$ for $y \in \mathcal{D}$ and 0 otherwise (truncated normal density);

PADS2(1): $g_{n,j}(y) = A_2 \exp\left\{\frac{-\|y-x_n^*\|^2}{2\sigma_n^2}\right\}$ for all $y \in \mathbb{R}^d$ (multivariate normal density).

In either case, A_1, $A_2 > 0$ are normalizing constants and it is easy to see that $C := A_i \exp\left\{\frac{-\|\text{ub}-\text{lb}\|^2}{2(\inf_{n\geq n_0}\sigma_n)^2}\right\} > 0$ will be a desired constant such that $g_{n,\,j}(y) \geq C$ for all $y \in \mathcal{D} =$ [lb, ub]. □

Lemma 3.8. *If* $\inf_{n\geq n_0}\varphi(n) > 0$, *the Condition [C2] holds for PADS(1).*

Proof. Assume that $\inf_{n\geq n_0}\varphi(n) > 0$. Let $j \in \{1, \ldots N_{\text{cand}}\}$, $x \in \mathcal{D}$ and $\delta > 0$ be given. Continuing with the notation used in Lemma 3.7, in particular recall that $H = \{R_{n,\,j}(i) = 1 \text{ for all } i = 1, ..., d\}$ and $g_{n,\,j}$ the conditional density of $Y_{n,\,j}$ given $\sigma(\mathcal{E}_{n-1})$ and H.

$$P[Y_{n,\,j} \in B(x,\,\delta) \cap \mathcal{D} | \sigma(\mathcal{E}_{n-1})]$$

$$\geq P[(Y_{n,\,j} \in B(x,\,\delta) \cap \mathcal{D}) \cap H | \sigma(\mathcal{E}_{n-1})]$$

$$= P[Y_{n,\,j} \in B(x,\,\delta) \cap \mathcal{D} | \sigma(\mathcal{E}_{n-1}),\, H] \times P(H)$$

$$= \left(\int_{B(x,\,\delta)\cap\mathcal{D}} g_{n,\,j}(y)dy \right) \times \varphi(n)^d$$

$$\geq C\mu\left(B(x,\,\delta) \cap \mathcal{D}\right) \times \left(\inf_{n\geq n_0}\varphi(n)\right)^d := \nu_j(x,\,\delta)$$

for any $n \geq n_0$, where $C > 0$ is a constant existing in Lemma 3.7. Then, by Lemma 3.7 and the fact that $\mathcal{D}$ is a compact hyperrectangle and our assumption on φ that $\inf_{n\geq n_0}\varphi(n) > 0$, one can easily see that $\nu_j(x,\,\delta) > 0$. Note also that $\nu_j(x,\,\delta)$ is independent of n. Thus, the condition **[C2]** is now verified. □

Since the two conditions **[C1]** and **[C2]** hold for PADS(1), we can apply Theorem 3.6 and conclude that PADS(1) converges to the global minimum with probability 1.

Remark 3.9. Analogous to the arguments we made above for serial PADS(1), in general parallel PADS(J) where $J > 1$ can also be shown to converge to the global minimum in a probabilistic sense in a similar way.

Example 3.10. One example of φ that satisfies the sufficient condition given in Lemma 3.8 is:

$$\varphi(n) = \begin{cases} \varphi_0 \times [1 - \ln(n - n_0 + 1)/\ln(M - n_0)] & \text{if } n_0 \leq n \leq M - 2 \\ \varphi(M-2) & n > M - 2 \end{cases},$$

where $\varphi_0 > 0$ and $M >> 0$. This is an extension of $\varphi(n)$, which was defined in [97] up to $n = N_{\max}$, to the space of natural numbers larger than n_0.

Remark 3.11. Since DYCORS framework is equivalent to PADS2(1), the convergence of DYCORS is proven which was not done in [97].

3.4 Computational Experiments

3.4.1 Test Problems

Our algorithm is tested on the set of 7 well-known benchmark functions each with 30 and 200 dimensions, making a total of 14 functions. The benchmark functions are given in Table 3.1. The domain, the global minimum value, and its characteristic (unimodal or multimodal) are also given in the table.

3.4.2 Alternative Global Optimization Methods

There are several freely available serial surrogate model algorithms. However, only very few algorithms exploit the possibilities of parallel computing. For example, the serial code for NOMADm (implemented in MATLAB [2]) has a build-in response surface (e.g. kriging, RBF) options. However, PSD-MADS (a parallel version of MADS available in C++ [9]) does not have a response surface implemented. Moreover, the serial rbfSolve from TOMLAB [49, 50, 51] also does not have a parallel implementation. In addition to PSD-MADS

Table 3.1: Test problems used in the experiments

Test function	**Domain**	$f(x^*)$	**Uni/Multi-modal (U/M)**
Ackley30	$[-15, 20]^{30}$	-22.7183	M
Griewank30	$[-400, 600]^{30}$	1	M
Levy30	$[-10, 10]^{30}$	1	M
Michalewicz30	$[0, \pi]^{30}$	n/a	M
Rastrigin30	$[-4, 5]^{30}$	-30	M
Rosenbrock30	$[-5, 10]^{30}$	1	U
Schwefel30	$[-500, 500]^{30}$	1.0004	M
Ackley200	$[-15, 20]^{200}$	-22.7183	M
Griewank200	$[-400, 600]^{200}$	1	M
Levy200	$[-10, 10]^{200}$	0	M
Michalewicz200	$[0, \pi]^{200}$	n/a	M
Rastrigin200	$[-4, 5]^{200}$	-200	M
Rosenbrock200	$[-5, 10]^{200}$	0	U
Schwefel200	$[-500, 500]^{200}$	0.0025	M

which is a parallel method designed specifically for large-scale problems, the alternative parallel surrogate-based methods we use in this study are ParLMSRBF [96]. Moreover, since population-based algorithms are also well suited for blackbox optimization [87] and it also offers an opportunity for parallelization, the parallel version of Differential Evolution, ParDE, as well as ParESGRBF, a parallel version of Evolution Strategy with surrogate [91, 107], are used in the comparison. A description of alternative algorithms for our comparison is discussed below.

3.4.2.1 rbfSolve

rbfSolve implemented in the TOMLAB Optimization Environment [50] is based on Gutmann's method [45]. The method uses an RBF model and seeks to balance between using the minimum of the response surface and minimizing a bumpiness measure [15, 45]. The solver does a warm start and resumes optimization from where the last run ended. We set the experimental design as ExD5, where SLHDs is used as the the initial points supplied to the solver. Note that this is the same as those used in PADS as well as other RBF based methods for the numerical comparison. The default values were assigned for other

parameters.

3.4.2.2 LMSRBF

The major difference between LMSRBF [95] and PADS is that a trial point in LMSRBF is generated by applying normal distributed random perturbations on all coordinates of the current x_{best} as opposed to just some of the coordinates as done in PADS. See Table 3.2 for the parameter settings.

3.4.2.3 Evolutionary Algorithms

The use of evolutionary algorithms to solve blackbox optimization problems has gained popularity in recent years. Among many evolutionary algorithms, Differential Evolution (DE) [113] which is based on weighted vector differences is considered one of the most powerful stochastic optimization algorithms for blackbox functions. While traditional evolutionary strategies use predetermined probability distributions to perturb population vectors, DE uses vector differences for perturbing the vector population with self adaptation. The Matlab implementation of DE that is available from: http://www.icsi.berkeley.edu/∼storn/code.html is used for our comparison.

The main parameters of DE are: NP (the population size ≥ 4), CR (the crossover constant $\in [0, 1]$), and F (the weight applied to random differential, called a scaling factor $\in [0, 2]$). Since there is no single strategy that works out to be the best for all given problems, following Storn and Price [113], we set the parameters F = 0.5 for the scaling factor and CR = 0.9 for the probability of crossover. The recommended value of NP is between $5 \times d$ and $10 \times d$ (i.e. $150 \leq \text{NP} \leq 300$ for $d = 30$ and $1000 \leq \text{NP} \leq 2000$ for $d = 200$). However, with the limitation of the number of function evaluations allowed, this choice is no longer appropriate when $d = 200$. Therefore, after some preliminary tests, NP = 304 ($\approx 10 \times d$) and 512 ($\approx 2.5 \times d$) are taken for problems with 30 and 200 dimensions, respectively.

There are many variants of DE proposed in [113]. In addition to DE/rand/1, which is

the classical version of DE, we also use DE/best/1 strategy, with the same set of parameters ("rand" refers to a randomly chosen population vector, and "best" refers to the vector of lowest cost from the current populations). We found that the DE/best/1 strategy works much better on all the test functions we consider. With this strategy, we also tried a different set of parameters: (F = 0.3, CR = 1) where NP are kept the same as previous. In addition, for $d = 30$ problems, we also reduced the number of NP to 64 ($\approx 2 \times d$).

Preliminary results of DE for various test problems and different parameter settings are summarized in Figures 3.7 and 3.8 for 30- and 200-dimensional problems, respectively. We see that for the 30-dimensional problems, DE/best/1 performs better with the parameters (F = 0.5, CR = 0.9). However, for the 200-dimensional problems, (F = 0.3, CR = 1) works significantly better for all problems. In addition, for the 30-dimensional problems with smaller NP, the algorithm is able to find improved objective function values for most problems only within the first few hundred evaluations and then flattens out, which indicates that the algorithm may be in a local minimum.

In the following we will use, for example, DE/rand/1-(0.5,0.9,304) to refer to the DE/rand/1 strategy with parameters (F = 0.5, CR = 0.9, NP = 304). For a clearer presentation, for a 30-dimensional problem, we will include only the best two algorithms: DE/best/1-(0.5,0.9,304) and DE/best/1-(0.3,1,64), while for a 200-dimensional problem, we include DE/best/1-(0.3,1,512) in our numerical comparison with other alternative methods.

In addition to DE, we also compare an evolutionary algorithm that uses RBF approximation (ESGRBF) [91, 107] with our method. ESGRBF incorporates the use of RBFs into a standard Evolution Strategy (ES). Based on self-adaptive algorithm parameters, in each generation, $\lambda \geq \nu$ offspring are generated and their objective function value is predicted by the RBF surrogate. The true objective function is evaluated for the ν individuals with the best predicted objective function values. Finally, the μ parents for the next generation are then selected from these ν offspring. In the numerical experiments, the parameters are taken

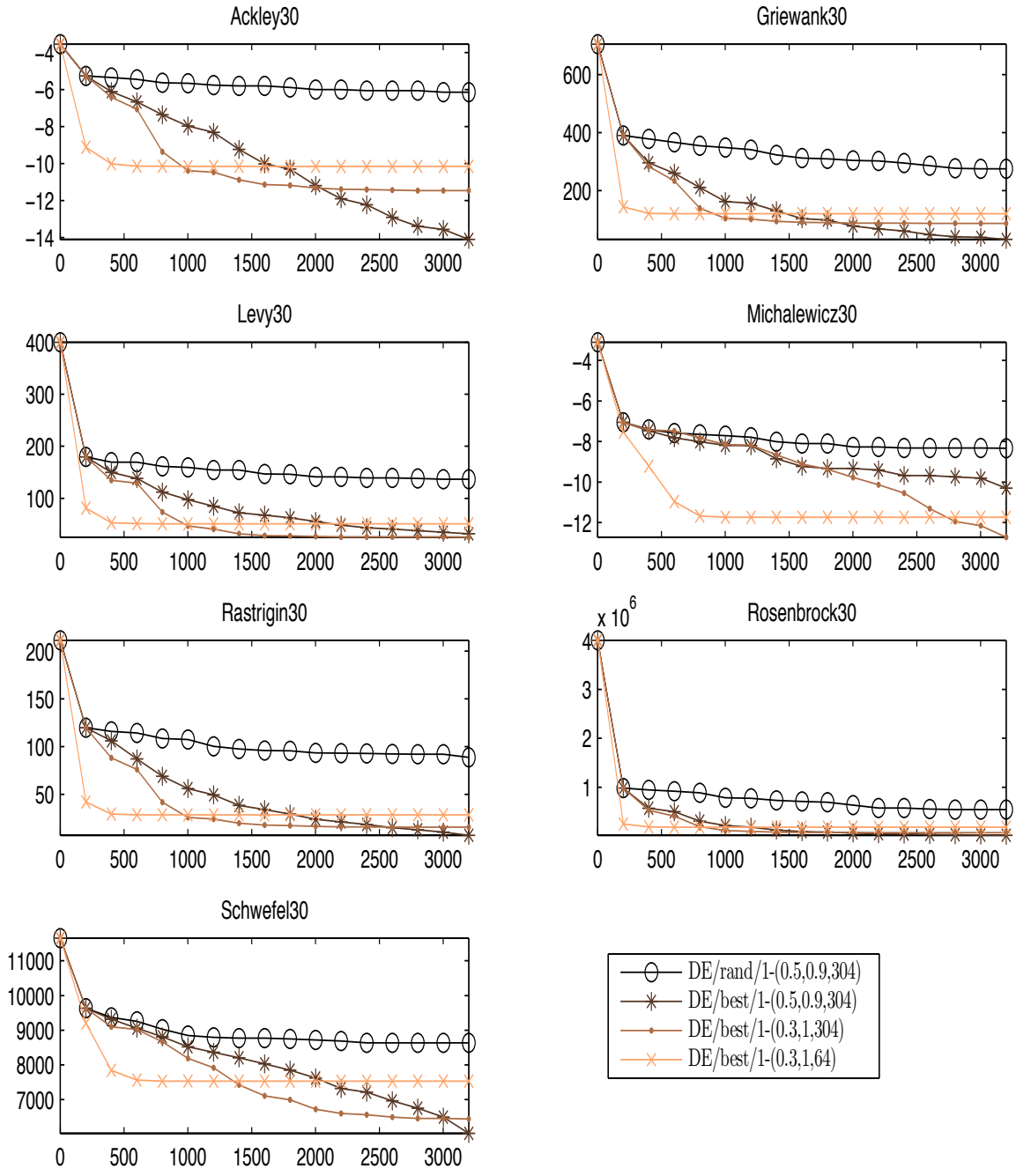


Figure 3.7: Results of DE for 30-dimensional problems. Plots show the mean of the best objective function value (y) vs. the number of function evaluations (x).

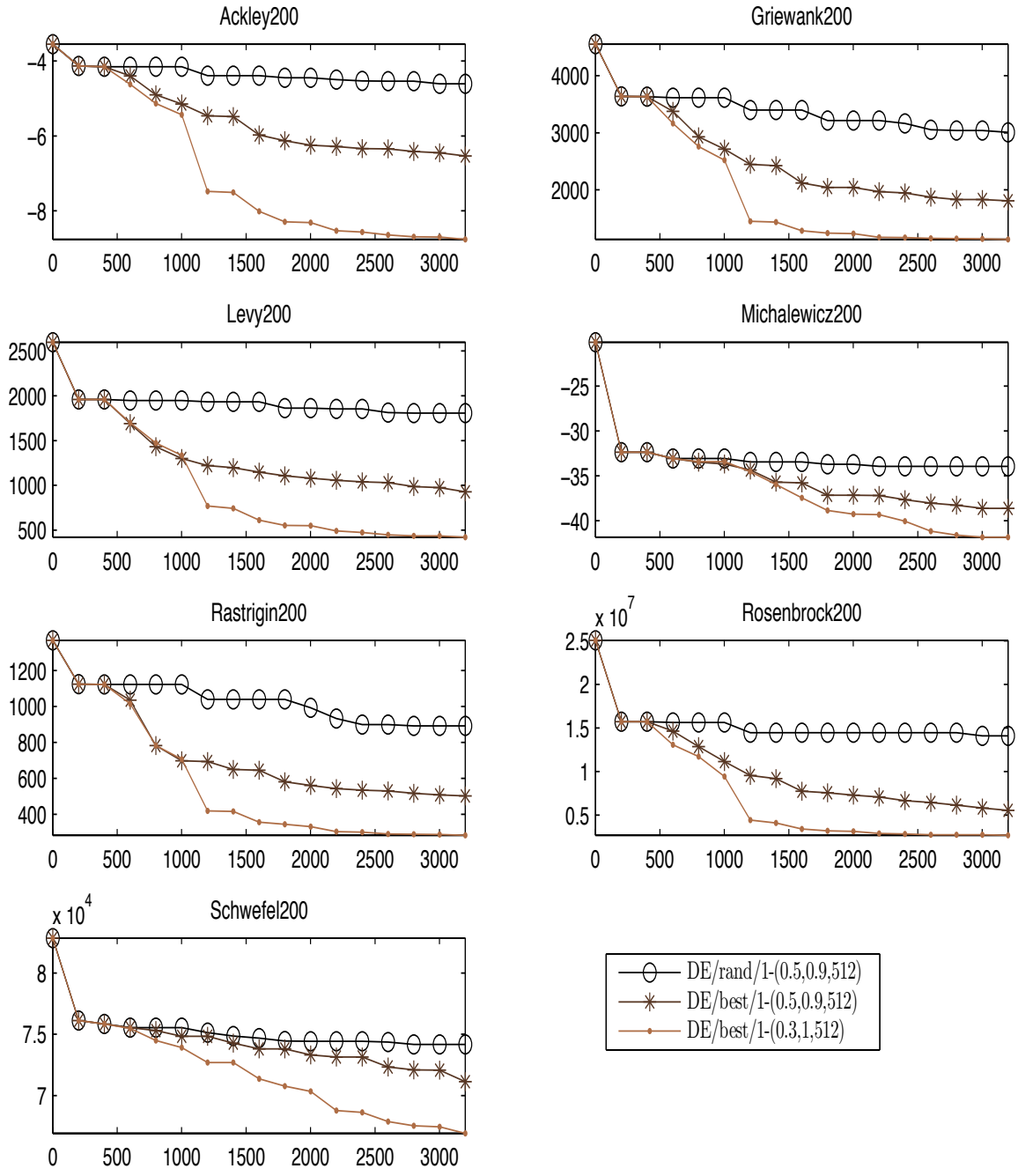


Figure 3.8: Results of DE for 200-dimensional problems. Plots show the mean of the best objective function value (y) vs. the number of function evaluations (x).

as: $\mu = 8$, $\lambda = 50$, and $\nu = 16$. Note that the parameter ν was set to 20 in [107]; however, after a preliminary test, we found that the results are not significantly different from $\nu = 16$. Thus, to make it easy to parallelize (with $P = 4, 8, 16$ processors), we set $\nu = 16$.

3.4.2.4 MADS (Mesh Adaptive Direct Search methods)

NOMAD [1, 9, 65, 67] is a C++ implementation of MADS methods [4, 8] and is designed to solve nonlinear, nonsmooth optimization problems. Besides a serial version, several parallel versions of MADS have also been proposed. P-MADS is the most straightforward implementation of MADS, where several points are evaluated in parallel by different processors. Recently, Audet et al. [9] proposed PSD-MADS which uses Parallel Space Decomposition within the MADS framework. PSD-MADS splits the variables of the original problem into a fixed number of subproblem, where each subproblem has only a small number of variables. The algorithm was used to solve problems with up to 500 variables [9].

For the comparison of serial algorithms, we use NOMADm, which is a MATLAB implementation of the MADS algorithm [2]. NOMADm can be used with an RBF or a kriging (DACE) surrogate model. We refer to these algorithms as NOMADm-RBF and NOMADm-DACE, respectively. For the comparison of the parallel algorithms, we will use the C++ implementation of PSD-MADS [9, 66].

Good results for a high-dimensional problem were obtained by setting $bbe_{\max} = 10$ and $ns = 2$ in [9, 66]. We tested PSD-MADS both with $ns = 2$ and $ns = 20$, and we also found that having the processors work on small-dimensional subspaces with $ns = 2$ leads to better results. Therefore, we keep $bbe_{\max} = 10$ and $ns = 2$ fixed for all test problems.

3.4.3 Experimental Setup

All numerical experiments except for PSD-MADS were carried out on MATLAB 7.14 (R2012a) on Intel(R) Core(TM) i7 CPU @3.40GHz 3.40 GHz. Since PSD-MADS is implemented in C++, the parallel experiments were performed on the NSF and NCAR Yellowstone com-

Table 3.2: Parameter values for PADS and LMSRBF

Parameter	Value
$N_{\text{cand}} = \lvert\Omega_n\rvert$ (number of trial points for each iteration)	$\min(500d, 5000)$
Υ (weight pattern)	$< 0.3, 0.5, 0.8, 0.95 >$
κ (number of weights in Υ)	4
σ_{int} (initial step size)	$0.2l(\mathcal{D})$
$\sigma_{\min}$ (minimum step size)	$(0.2)(1/2)^6 l(\mathcal{D})$
τ_{success}	3
τ_{fail}	$\max(d, 5)$

puting environment [26] and solutions were computed and stored using double precision. Ten trials of all algorithms are performed for all 30-dimensional problems and five trials are performed for the 200-dimensional problems.

All algorithms based on RBF used the cubic RBF model with a linear polynomial tail. The RBF model is initialized using symmetric Latin hypercube designs (SLHDs) [129] for the 30-dimensional problems, while non-symmetric Latin hypercube designs (LHDs) are used for the 200-dimensional problems as was done in [97]. The initial sample size n_0 for an algorithm running with P processors is set to the smallest non-negative integer in the set $\mathcal{M}_P$, where $\mathcal{M}_P = \{m \in \mathbb{N} : \, m \geq 2(d+1) \,\text{and}\, P \mid m\}$, and $\mathcal{M}_P = \{m \in \mathbb{N} : \, m \geq d+1 \,\text{and}\, P \mid m\}$ are used for problems with 30 and 200 dimensions, respectively.

The parameter values for PADS and LMSRBF are set as in [97] and are for convenience shown in Table 3.2, where $l(\mathcal{D})$ denotes the length of the shortest side of the hyperrectangle $\mathcal{D}$.

3.5 Numerical Results and Discussion

In this section, the experimental results are presented for the benchmark test functions described in Section 3.4.1.

3.5.1 Performance Measurement Setup

Assume that the computational time for each function evaluation is approximately equal. Let A_P be an algorithm we consider that uses P processors for doing P function evaluations in parallel. Then, for any A_P, P function evaluations can be done simultaneously in one parallel iteration, and the total wall-clock time can be broken down as:

$$W(P) = \underbrace{(n_f(P)/P)}_{=:\ I_P} t + T_{-f}(P) \tag{3.5.1}$$

where

- P is the number of processors
- t is the CPU time for one function evaluation
- $n_f(P)$ is the total number of function evaluations performed by any of the P processors
- $I_P := n_f(P)/P$ is the number of parallel iterations (where $n_f(P)$ is divisible by P)
- $T_{-f}(P)$ is the wall-clock time the algorithm spends on everything else except the function evaluations. In particular, this includes fitting and updating the response surface, selecting the next evaluation points as well as the communication time between the master and the workers, and contention for shared data structures.

Note that the number of iterations (I_P) should not be confused with the number of function evaluations ($n_f(P)$). Since we assume that each function evaluation takes approximately the same amount of time to compute, P function evaluations can be done simultaneously for algorithm A_P in one iteration. For example, PADS(16), PADS(8), and PADS(4) perform 16, 8 and 4 function evaluations, respectively, in one parallel iteration while the serial algorithm PADS(1) only does a single function evaluation in each iteration. Hence, if $I_P = 10$, then $n_f(P) = 160$ for PADS(16), $n_f(P) = 80$ for PADS(8), $n_f(P)= 40$ for PADS(4), and $n_f(P) = 10$ for the serial algorithm.

As discussed in [96], when the objective function is truly expensive (i.e. t is very large), then $T_{-f}(P)$ is generally negligible compared to $tI_P = t(n_f(P)/P)$ which is the total time spent on function evaluations. In addition, since the actual computation times of our test problems are low (milliseconds), we compare the algorithms based on the number of parallel iterations I_P.

3.5.2 Comparison with Alternative Methods

In this section, PADS1 and PADS2 are compared with the methods described in Section 3.4.1. We assume that the number of processors P = the number of evaluated points per iteration J.

3.5.2.1 Serial Algorithms

Figures 3.9 and 3.10 show the progress curve of the serial algorithms for 30- and 200-dimensional problems, respectively. PADS1 performs similarly to PADS2 for the 30-dimensional problems. PADS1 outperformed or was as least as good as PADS2 for most 200-dimensional problems. PADS2 performs better than LMSRBF for all but one test problem (Ackley200). Recall that for the serial algorithm, PADS2(1) is essentially DYCORS-LMSRBF [97]. Thus, the new truncated Gaussian distribution implemented in PADS1(1) is proven more effective on this test suite than the Gaussian distribution with the reflective bound used in DYCORS.

Since DE does not use surrogate models, one would not expect it to work as well compared to the other methods. We can see that DE with both strategies performs worst for most 30-dimensional problems (Figure 3.9). For the 200-dimensional problems (Figure 3.10), DE outperforms NOMADm-DACE for three problems, and performs equally for the remaining problems. Recall that the latter is based on the surrogate.

NOMADm-RBF performs in general better than NOMADm-DACE. Both NOMADm versions are outperformed by the PADS and LMSRBF algorithms. ESGRBF was shown to be very promising algorithm on watershed calibration problems [107], but it is outperformed

for the test problems by several alternative algorithms including NOMADm-RBF.

We also investigated the performance of rbfSolve for the test problems. However, since the algorithm was not designed for high-dimensional problems, the algorithm's own computational time becomes unacceptable (e.g. about 5 days for 1 trial of 800 function evaluations on Levy200 function versus less than 1 hour (using the same machine) when running PADS1(1)). Thus, we used rbfSolve only for the 30-dimensional problems. While rbfSolve did very well for three of the 30-dimensional problems (Griewank30, Levy30, Rastrigin30), rbfSolve was outperformed for the other test problems. Thus, overall the results of the serial algorithms suggest that the PADS algorithms are promising and more robust over a much wider set of problems than the alternative methods.

3.5.2.2 Parallel Algorithms

Figures 3.11 and 3.12 show that overall PADS1(4) and PADS2(4) (with 4 function evaluations per iteration) perform best for all problems in comparison to the alternative methods with 4 processors. The performance of these two algorithms is very similar for the 30-dimensional problems. For the 200-dimensional problems, PADS1(4) performs in general better than PADS2(4). The overall performance of PADS(4) becomes more outstanding than that of others on 200-dimensional problems.

Figures 3.13, 3.14 and Figures 3.15, 3.16 show the progress curves of the algorithms when using 8 and 16 processors, respectively. The results are very similar to those with four processors, i.e. PADS1 and PADS2 are always the best two algorithms. PSD-MADS,

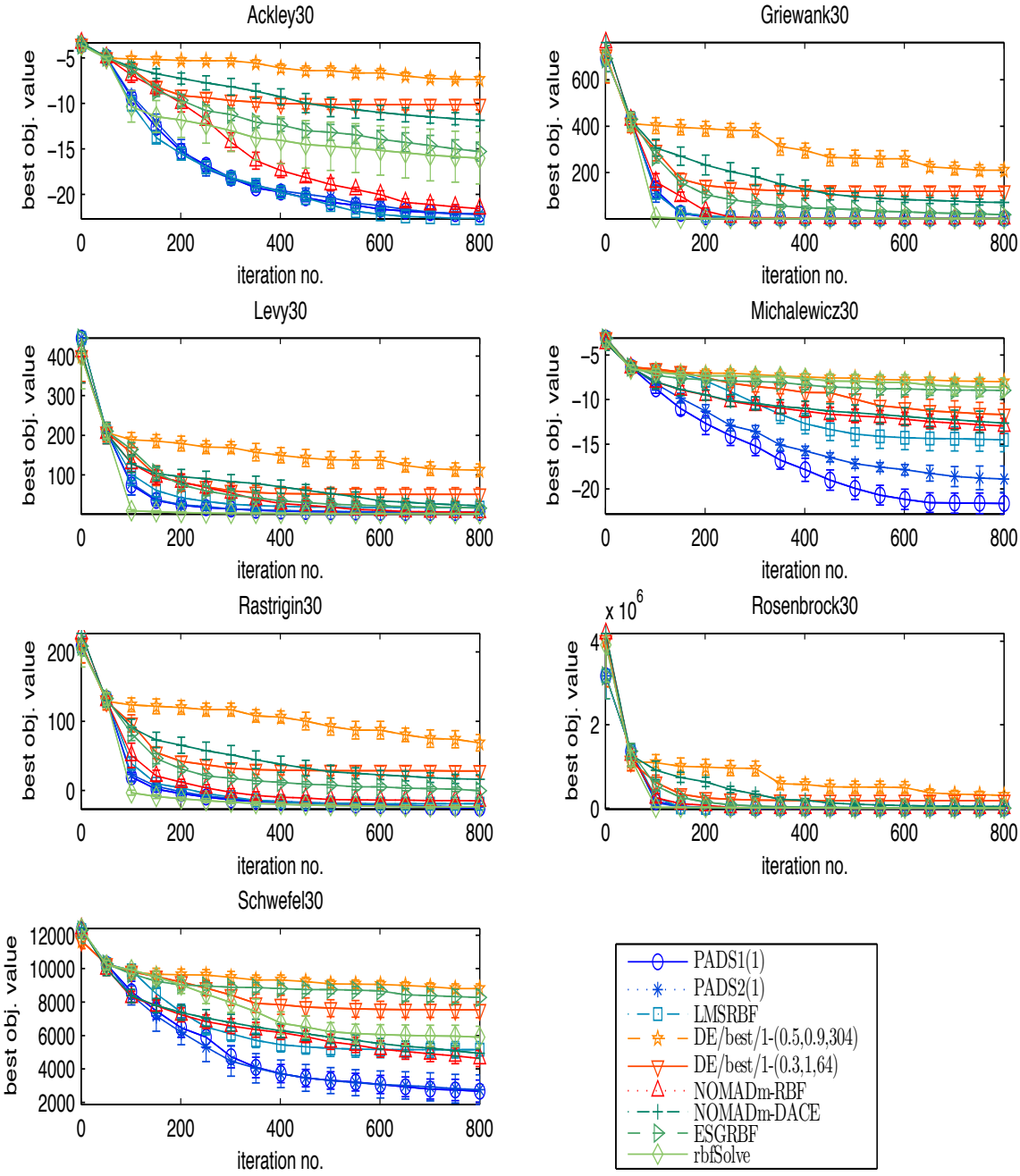


Figure 3.9: SERIAL, 30D: Performance comparison of serial algorithms for 30-dimensional problems. Shown are the progress curves (the mean of the best function values vs. the number of iterations) of the compared algorithms. For serial algorithms, the number of iterations equals the number of function evaluations. Error bars represent 95% confidence intervals about the mean.

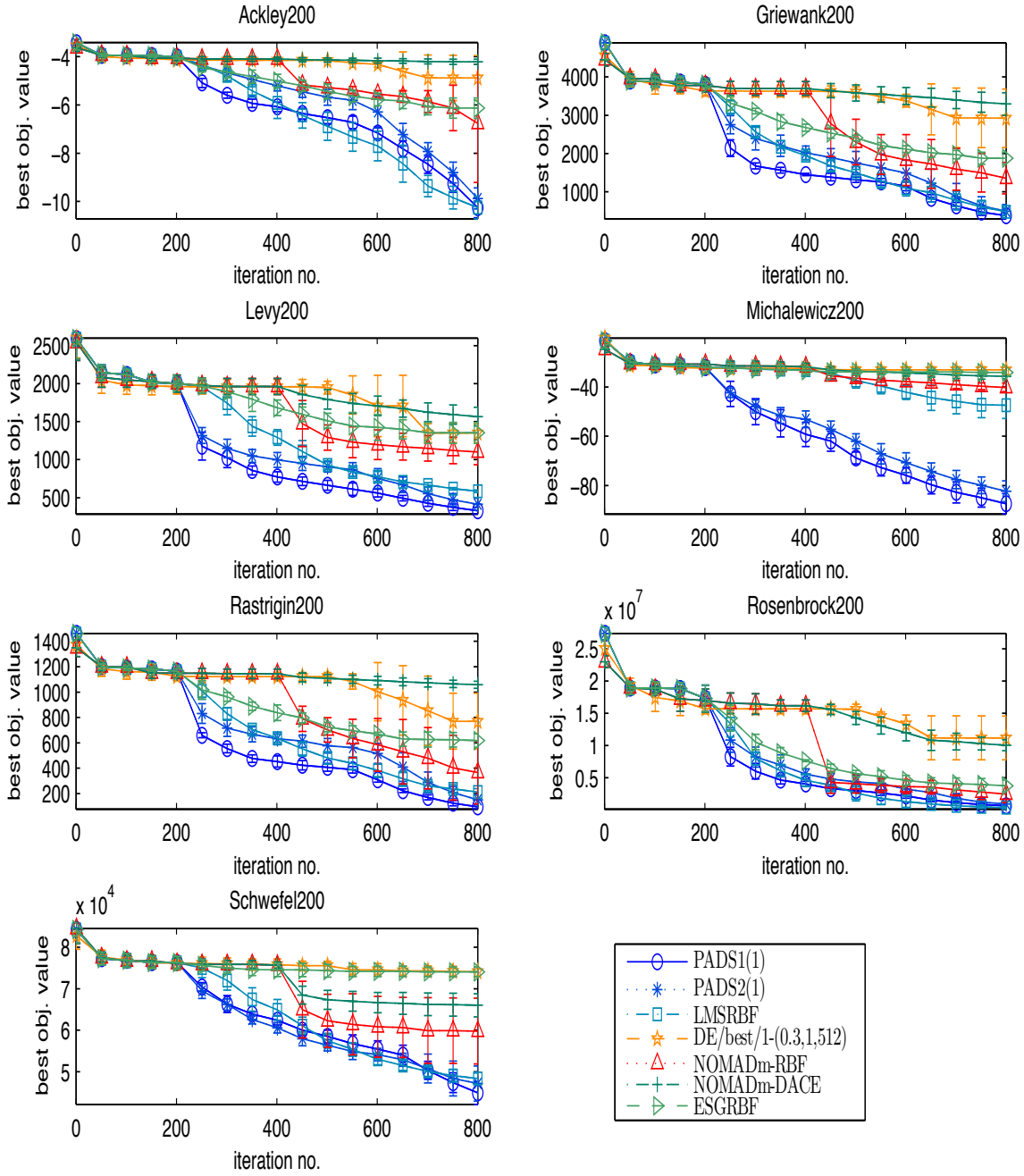


Figure 3.10: SERIAL, 200D: Performance comparison of serial algorithms for 200-dimensional problems. Shown are the progress curves (the mean of the best function values vs. the number of iterations) of the compared algorithms. For serial algorithms, the number of iterations equals the number of function evaluations. Error bars represent 95% confidence intervals about the mean.

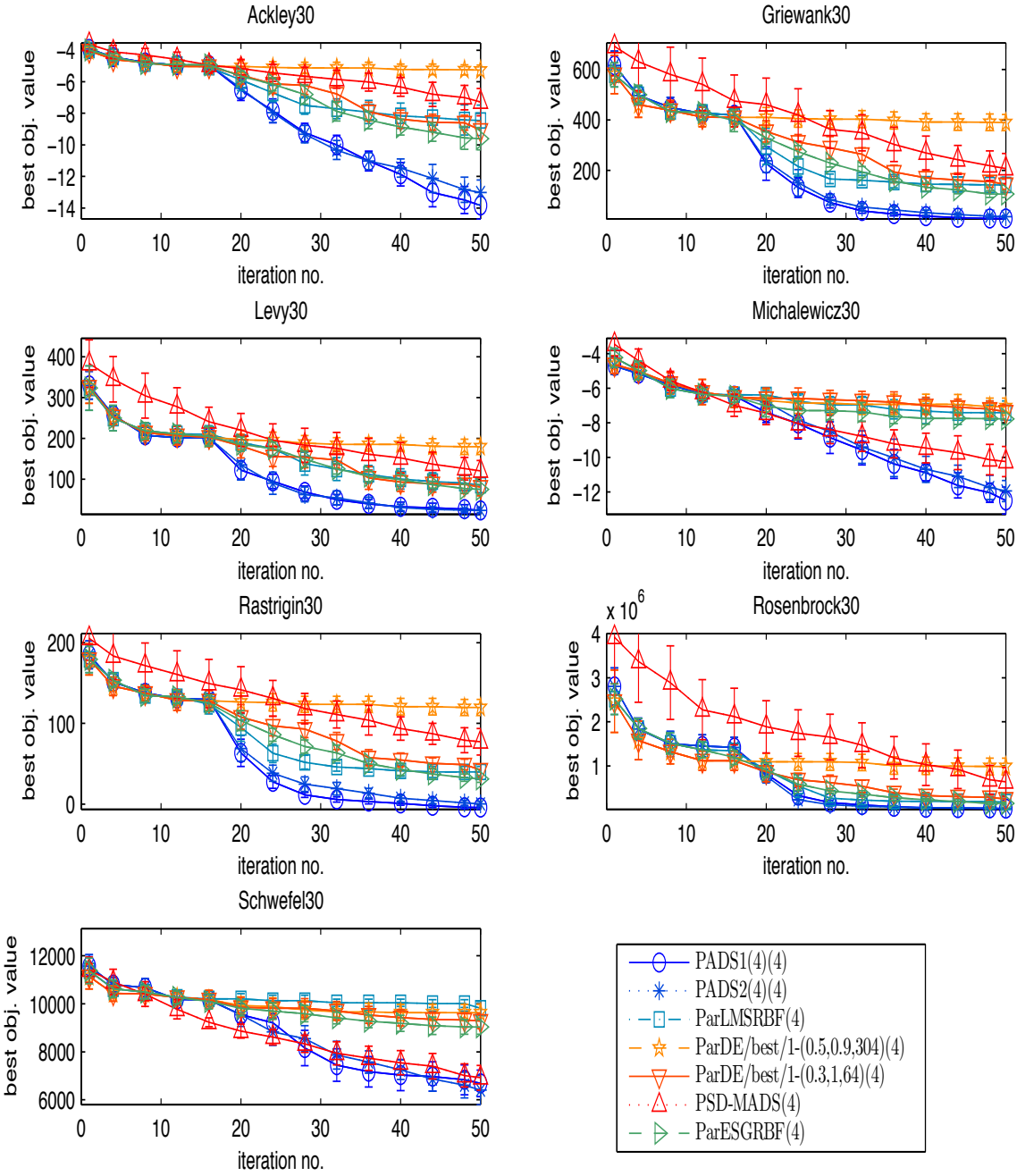


Figure 3.11: 4PARALLEL, 30D: Comparison when using 4 processors for 30-dimensional problems. Shown are the progress curves (the mean of the best function values vs. the number of iterations) of the compared algorithms. Error bars represent 95% confidence intervals about the mean.

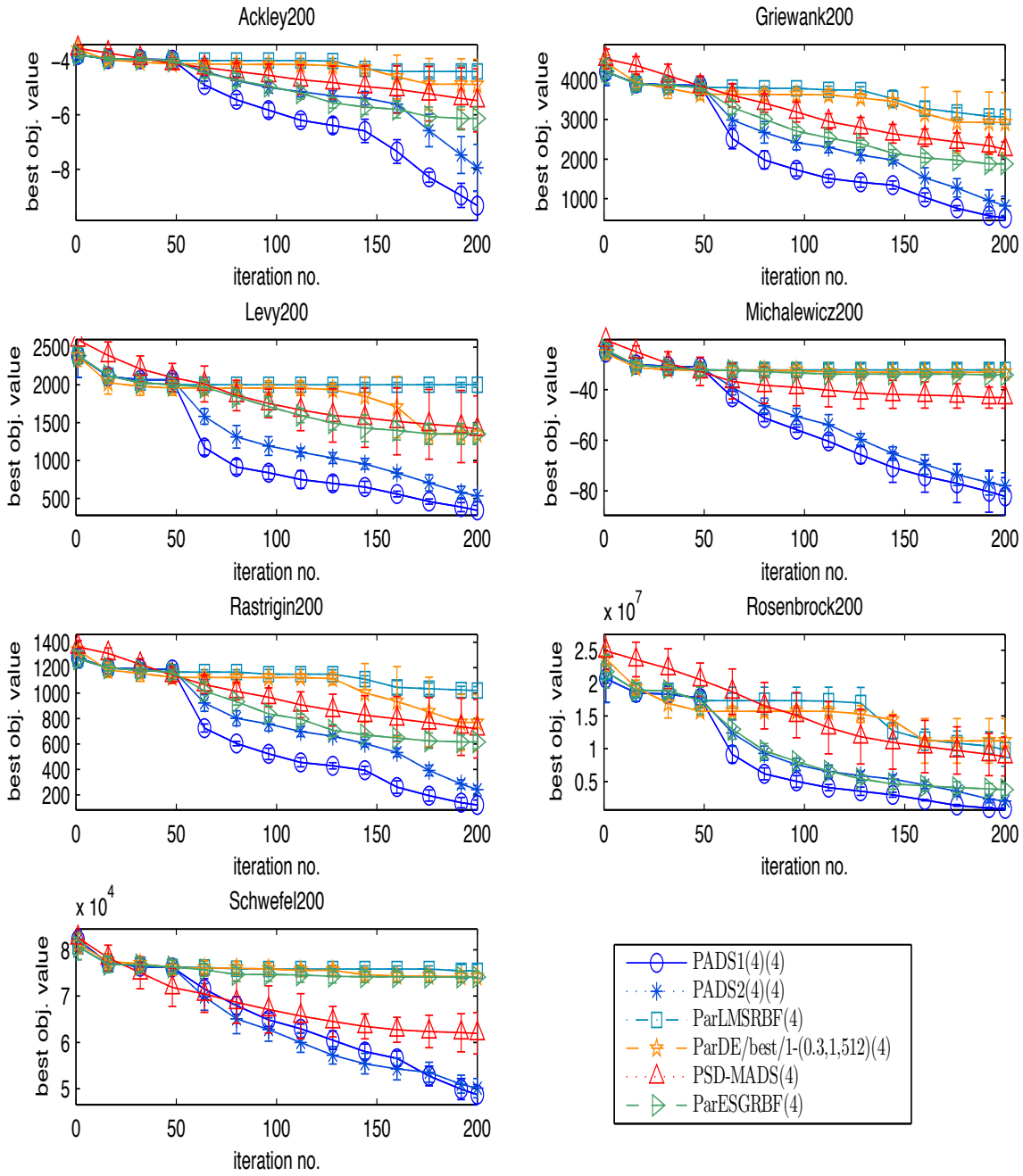


Figure 3.12: 4PARALLEL, 200D: Comparison when using 4 processors for 200-dimensional problems. Shown are the progress curves (the mean of the best function values vs. the number of iterations) of the compared algorithms. Error bars represent 95% confidence intervals about the mean.

ParESGRBF, and ParLMSRBF perform quite well for the 30-dimensional problems, but their overall performance becomes worse for many of 200-dimensional problems.

Overall, the results indicate that both PADS1(J) and PADS2(J) are much more efficient algorithms for HEB problems compared to the alternative methods. The alternative methods ParLMSRBF and ParESGRBF perform comparatively well for the 30-dimensional problems, but their performance for the 200-dimensional problems degrades significantly as the number of processors increase. Hence, not all parallel surrogate-based optimization methods are suitable for HEB problems. Lastly, although PSD-MADS was reported to work well for constrained problems with up to 500 dimensions [9], PSD-MADS did not perform well for any of the test problems used in this chapter in comparison to the algorithms used here.

3.5.2.3 Cumulative normalized results index

We compiled the results of all test problems with the same dimension into one cumulative index, namely $\mathfrak{I}(\mathcal{A}, n)$, to reflect the performance of algorithm $\mathcal{A}$ for all test functions after n iterations.

Let $V_f(\mathcal{A}, n)$ be the average best objective value of algorithm $\mathcal{A}$ for problem f found within n iterations. Let $\mathfrak{N}$ denote the number of algorithms.

Fix a test function f, let $m_f = \min_{1 \leq i \leq \mathfrak{N}} V_f(\mathcal{A}_i, n)$ and $M_f = \max_{1 \leq i \leq \mathfrak{N}} V_f(\mathcal{A}_i, n)$. So then m_f and M_f are the minimum and maximum, respectively, of the best average value

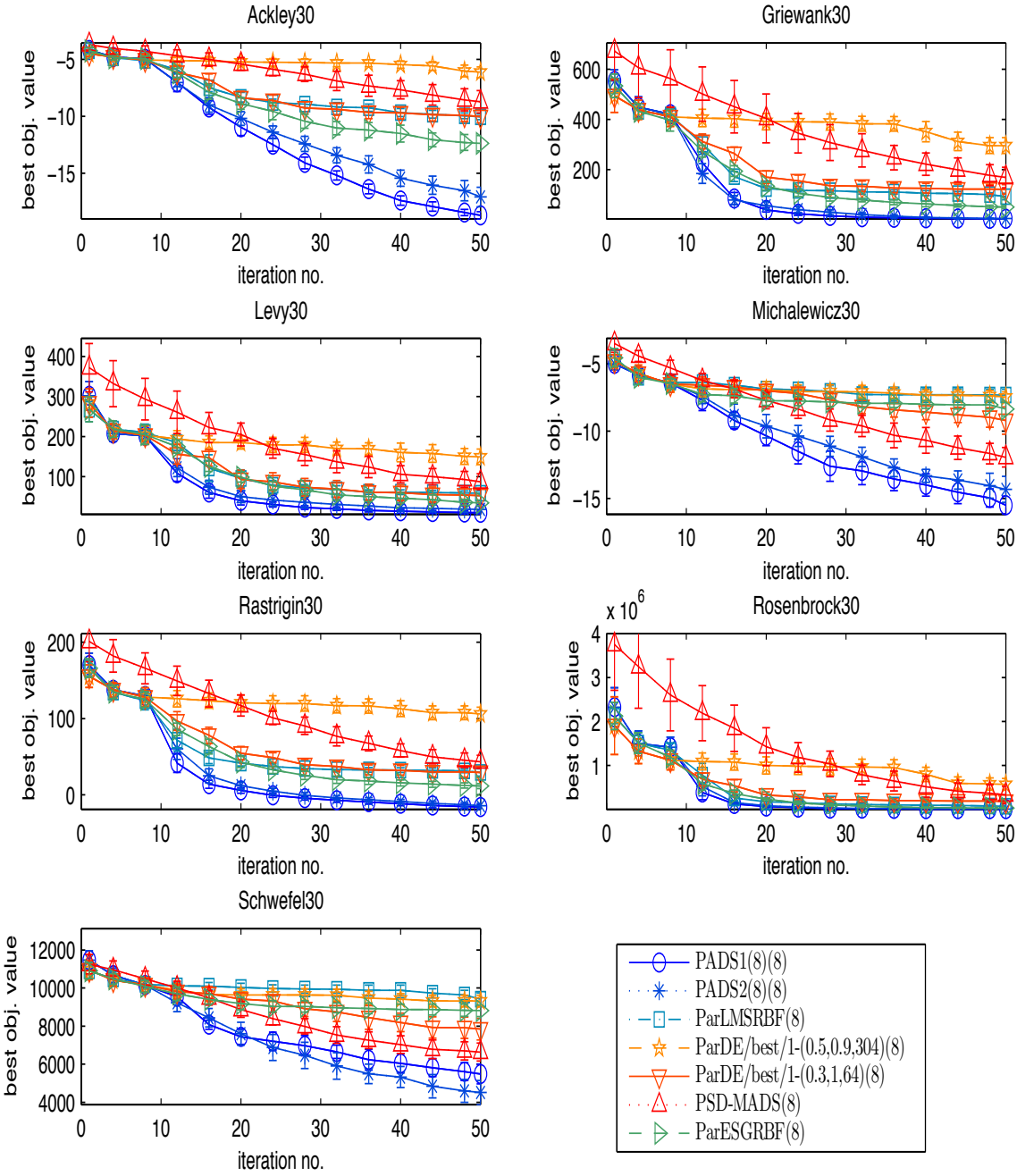


Figure 3.13: 8PARALLEL, 30D: Comparison when using 8 processors for 30-dimensional problems. Shown are the progress curves (the mean of the best function values vs. the number of iterations) of the compared algorithms. Error bars represent 95% confidence intervals about the mean.

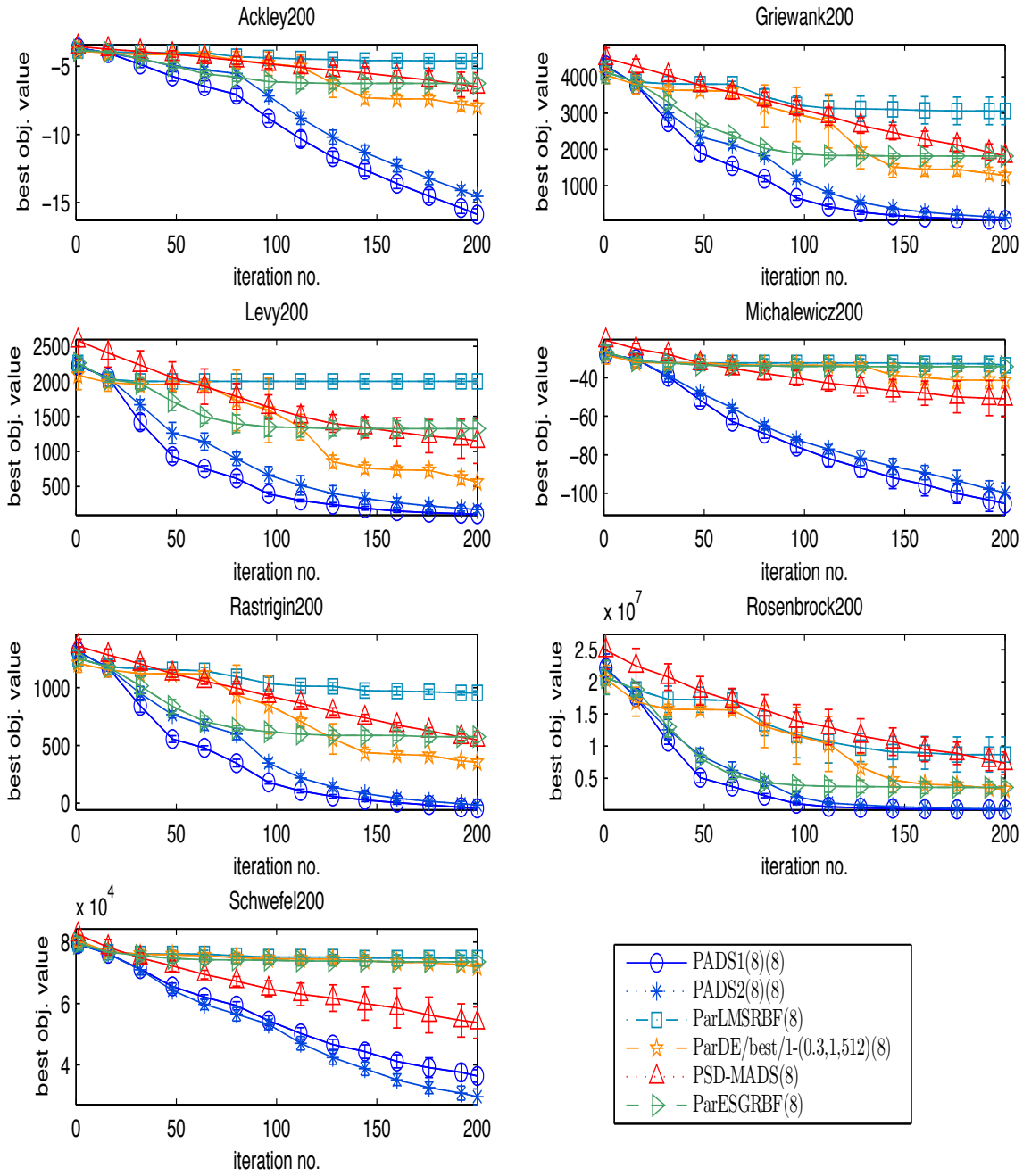


Figure 3.14: 8PARALLEL, 200D: Comparison when using 8 processors for 200-dimensional problems. Shown are the progress curves (the mean of the best function values vs. the number of iterations) of the compared algorithms. Error bars represent 95% confidence intervals about the mean.

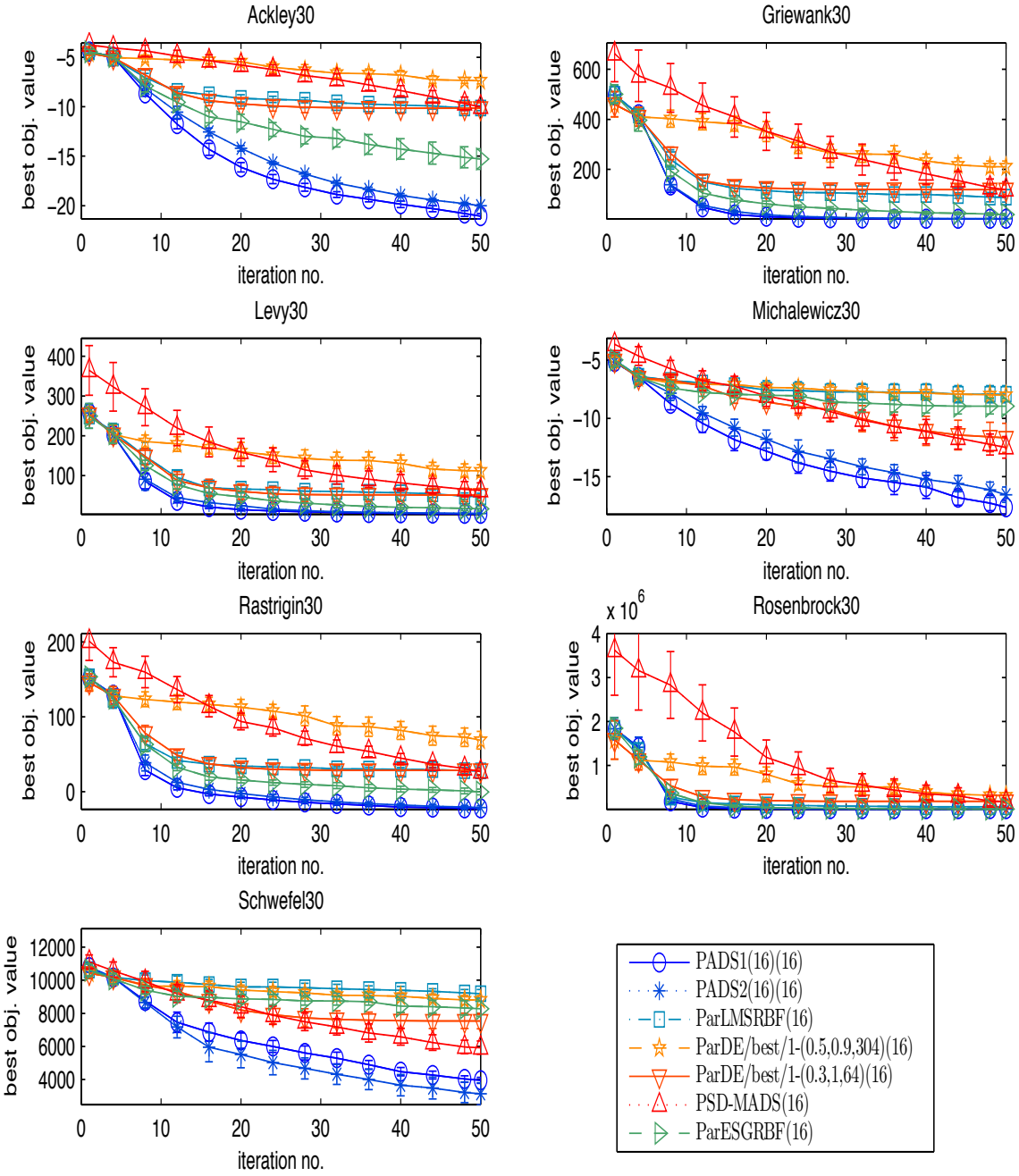


Figure 3.15: 16PARALLEL, 30D: Comparison when using 16 processors for 30-dimensional problems. Shown are the progress curves (the mean of the best function values vs. the number of iterations) of the compared algorithms. Error bars represent 95% confidence intervals about the mean.

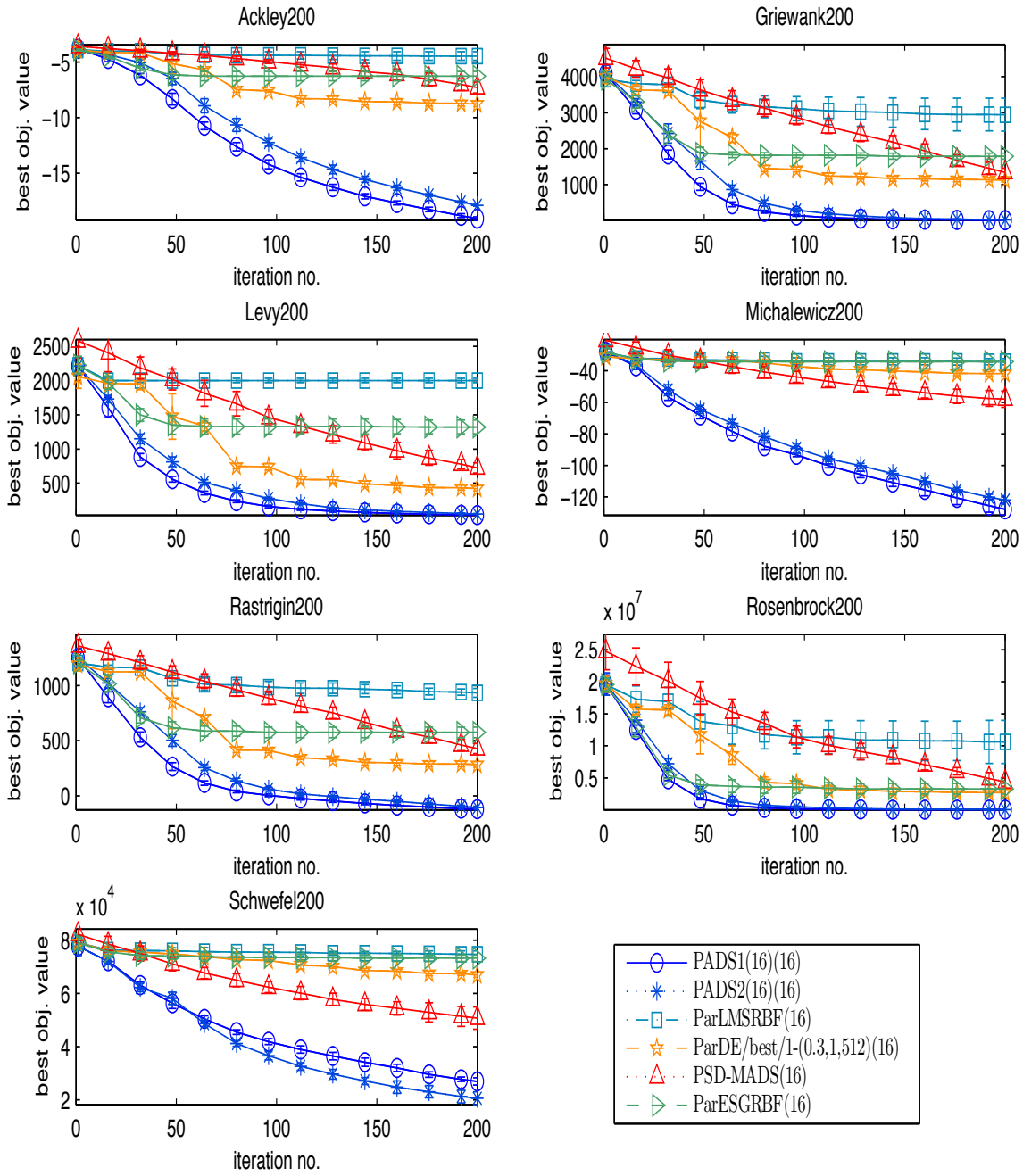


Figure 3.16: 16PARALLEL, 200D: Comparison when using 16 processors for 200-dimensional problems. Shown are the progress curves (the mean of the best function values vs. the number of iterations) of the compared algorithms. Error bars represent 95% confidence intervals about the mean.

found among all algorithms. We first calculate the normalized value $R_f(\mathcal{A}_i, n)$ for any algorithm $\mathcal{A}_i$ as follow. For $i = 1, ..., \mathfrak{N}$,

$$R_f(\mathcal{A}_i, n) = \frac{V_f(\mathcal{A}_i, n) - m_f}{M_f - m_f} \times 100. \tag{3.5.2}$$

$R_f(\mathcal{A}_i, n)$ is defined in such a way that, for a fixed test function f and iteration number n, 0 is assigned to the best algorithm, 100 to the worst algorithm, and a value between [0, 100] to other algorithms.

Let $\mathfrak{M}$ denote the number of test functions. Then, the cumulative index for algorithm $\mathcal{A}$ after n iterations is defined by

$$\mathfrak{I}(\mathcal{A}, n) = \sum_{i=1}^{\mathfrak{M}} R_{f_i}(\mathcal{A}, n)/\mathfrak{M}. \tag{3.5.3}$$

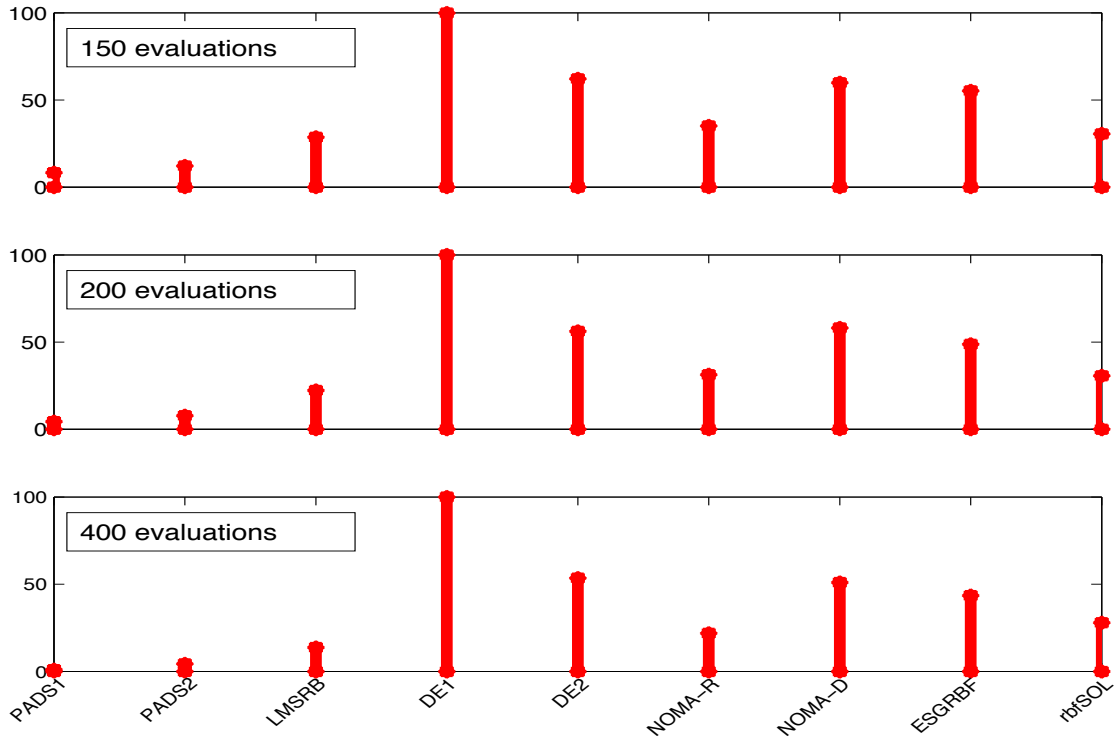
So $\mathfrak{I}(\mathcal{A}, n)$ is the average value over all test functions, rescaled to be in [0, 100].

The cumulative indices for results that were presented in Sections 3.5.2.1 and 3.5.2.2 are shown in Figures 3.17 through 3.20. The abbreviations LMSRB for LMSRBF, DE1 for DE/best/1-(0.5,0.9,304), DE2 for DE/best/1-(0.3,1,64), DE3 for DE/best/1-(0.3,1,512), NOMA-R for NOMADm-RBF, NOMA-D for NOMADm-DACE, rbfSOL for rbfSolve, and PSD-MA for PSD-MADS are used throughout this section.

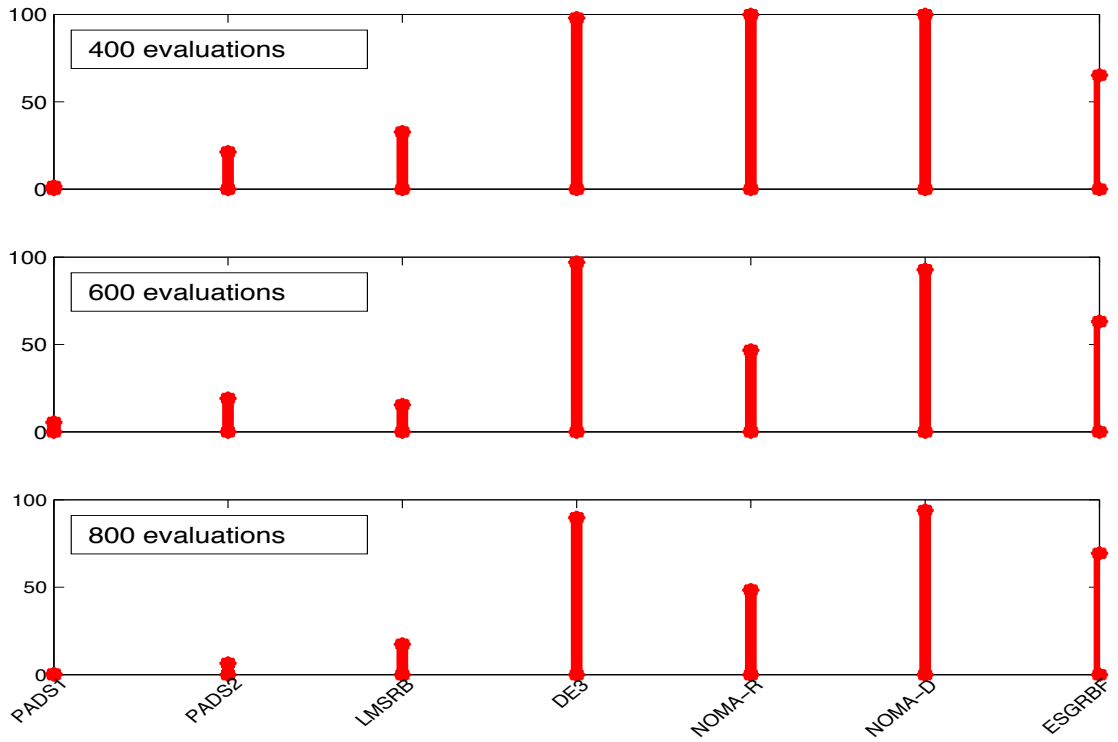
We see that the cumulative index of PADS1 in all cases are lowest for 30- and 200-dimensional problems. Hence, it is the best algorithm based on this measure. PADS2 also does very well according to this measure.

3.5.3 Analysis of PADS1(J)

In this section, we will compare the performance of PADS1(J) for J being the number of simulated points per iteration. The analysis for PADS2 can be done in the same manner.

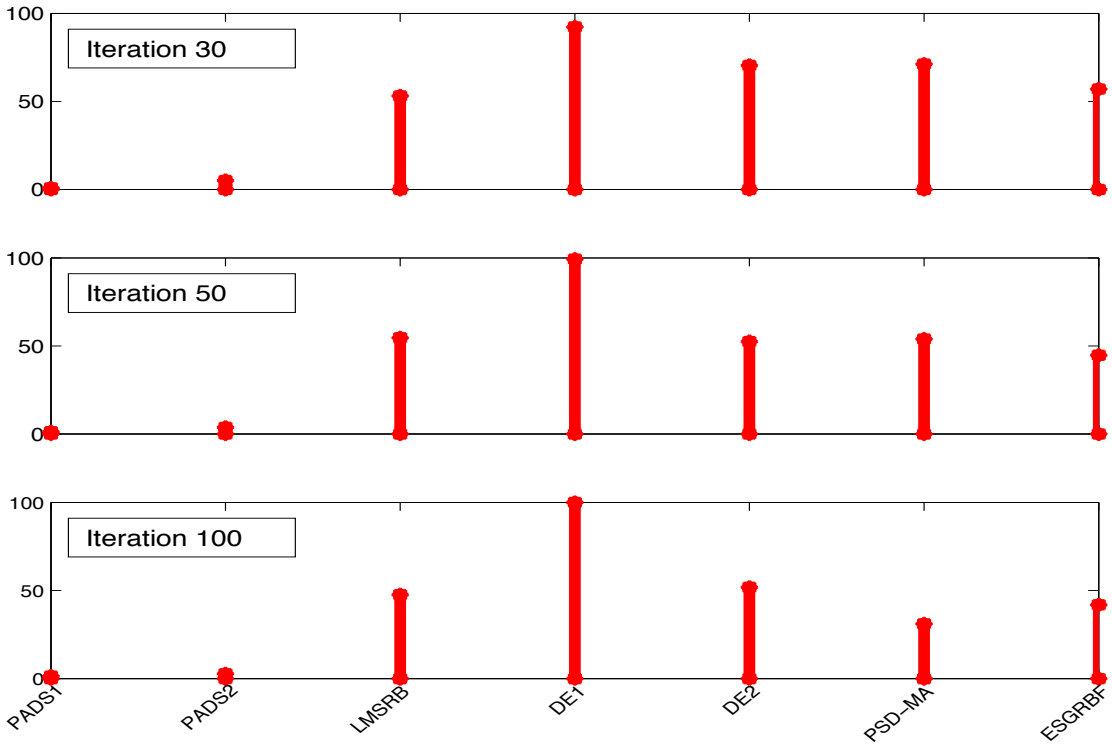


(a) 30-dimensional problems

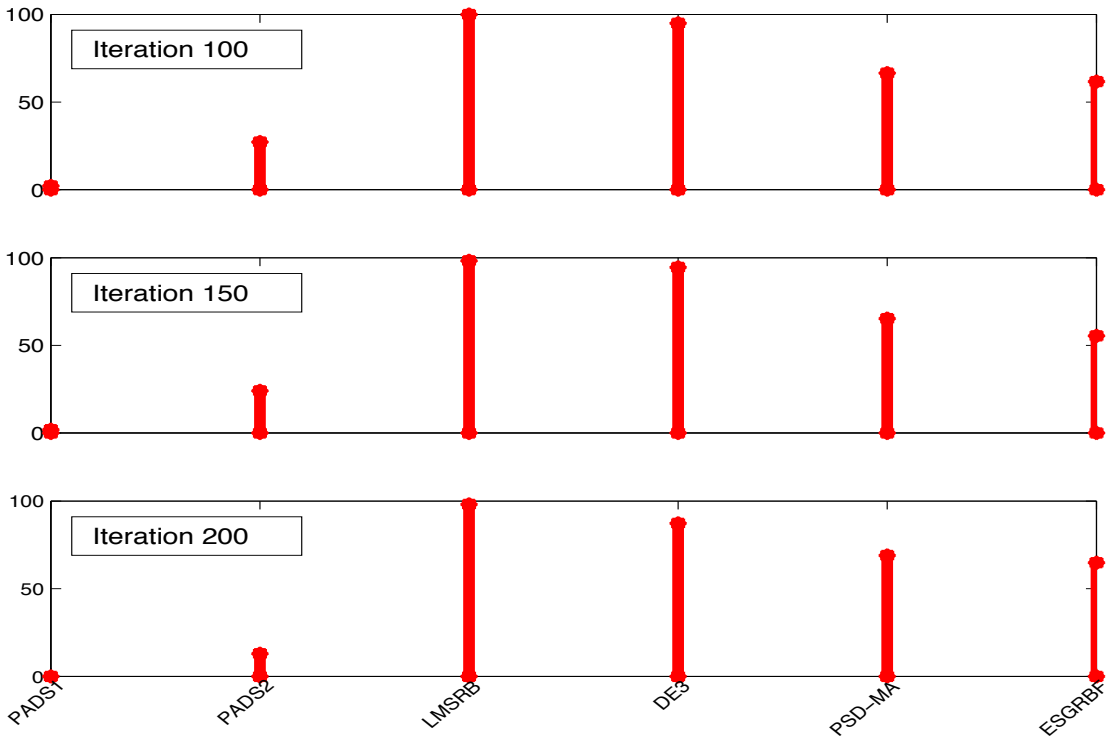


(b) 200-dimensional problems

Figure 3.17: Cumulative indices when using 1 processor

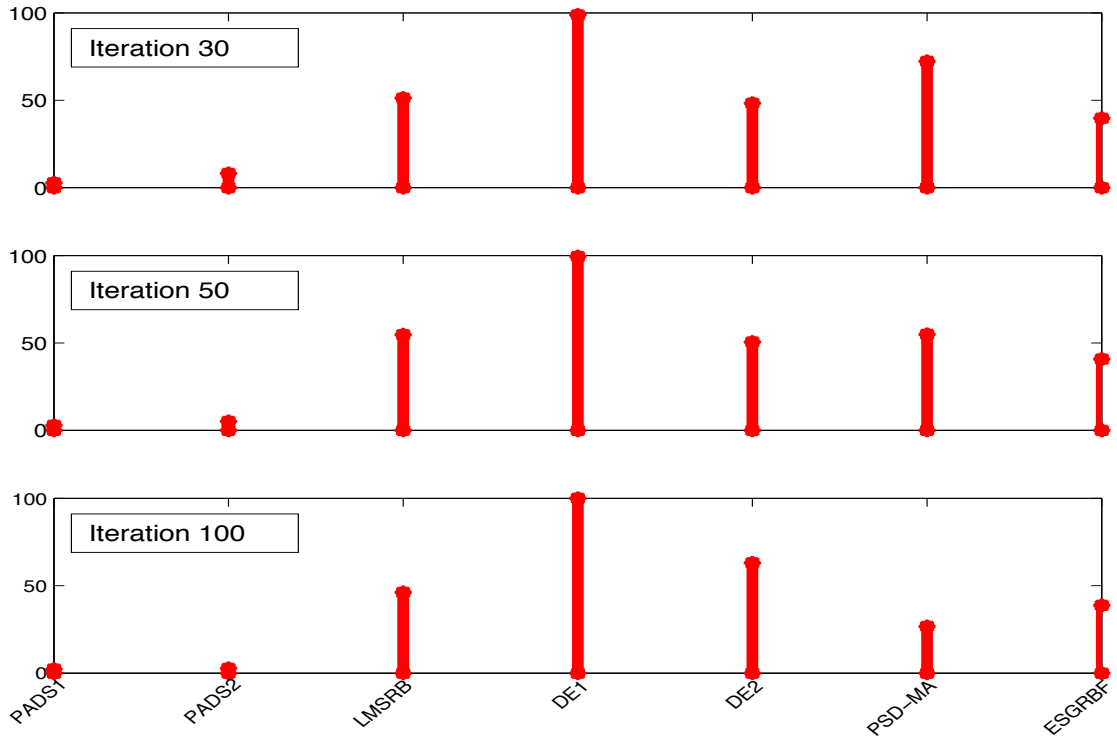


(a) 30-dimensional problems

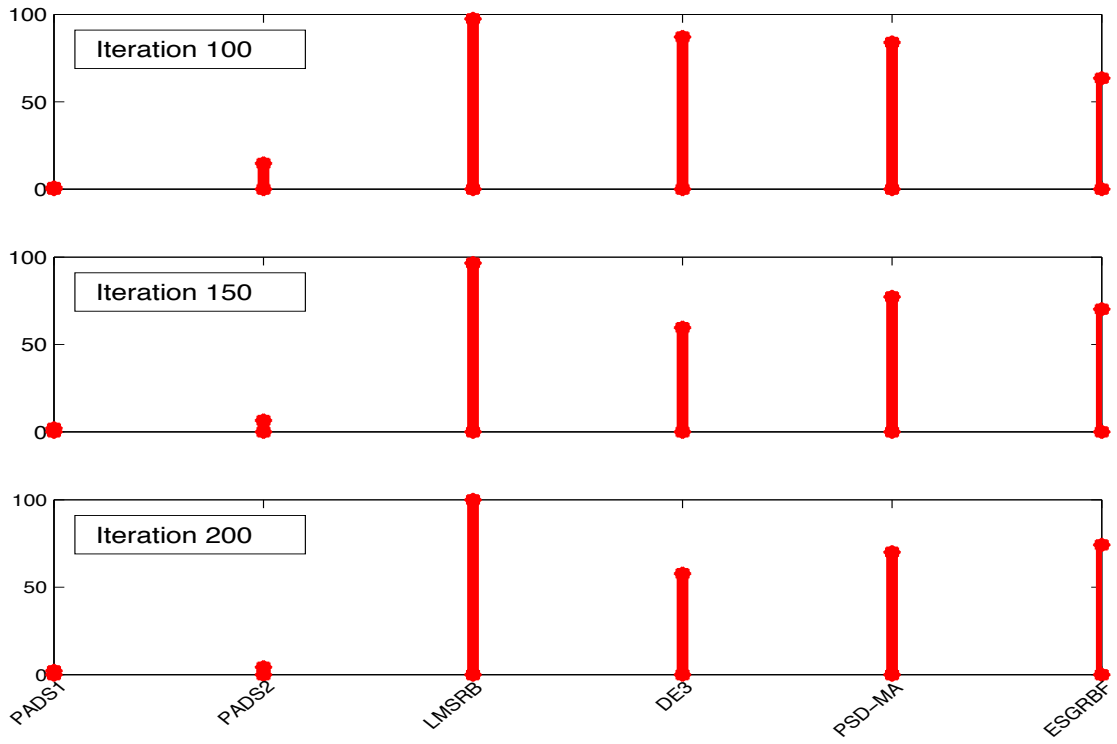


(b) 200-dimensional problems

Figure 3.18: Cumulative indices when using 4 processors

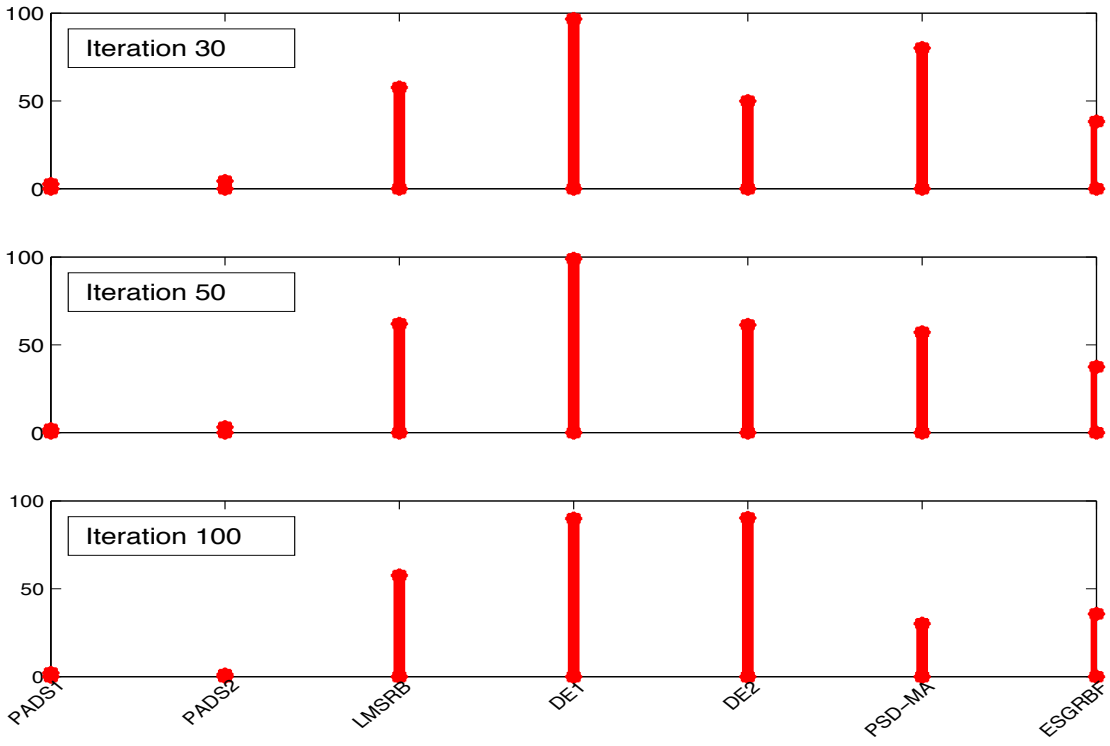


(a) 30-dimensional problems

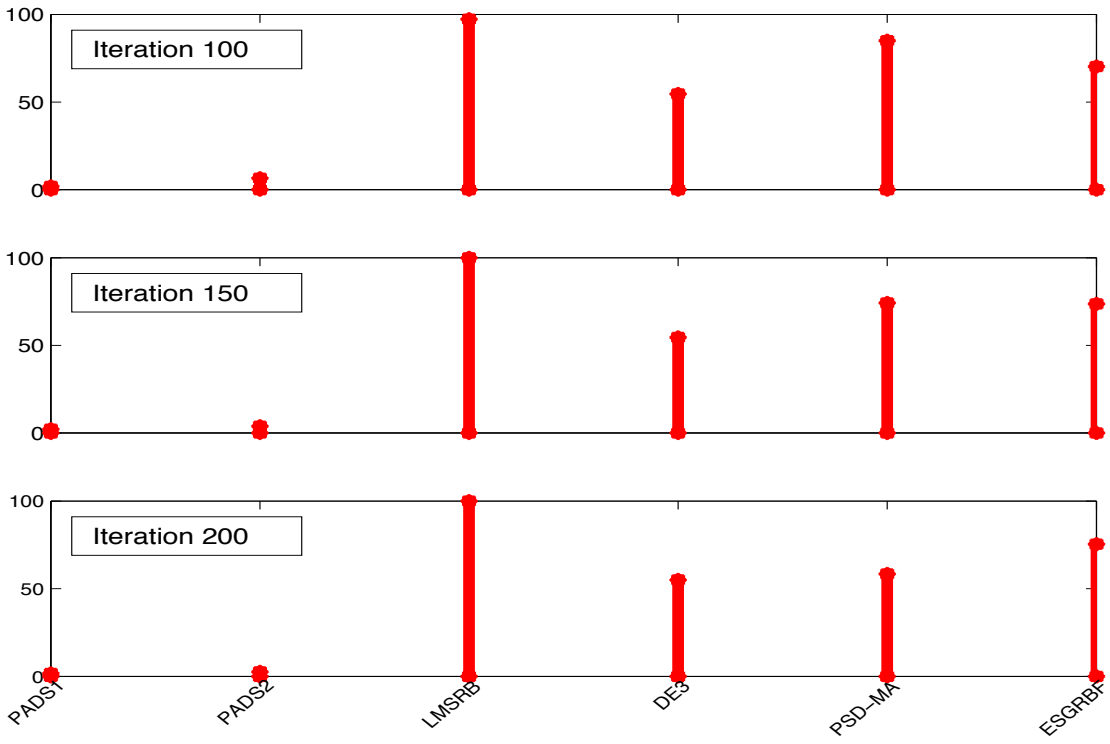


(b) 200-dimensional problems

Figure 3.19: Cumulative indices when using 8 processors



(a) 30-dimensional problems



(b) 200-dimensional problems

Figure 3.20: Cumulative indices when using 16 processors

3.5.3.1 Progress Plot

In Figures 3.21 and 3.22, we present the progress plots for PADS1(J) for $J = 1$, 4, 8, and 16 processors. Each subplot shows the mean of the best objective function value versus the number of iterations of PADS1(J). The results show that for a fixed number of iterations, PADS1(J) performs significantly better as J increases. The effect of a larger number of processors J becomes more prominent in 200-dimensional test functions where the plot of PADS1(16) has clearly separated from the rest. One thing we should point out is that for a serial PADS1(1), the first two hundred function evaluations of a 200-dimensional problem are spent merely on building the response surface, this explains why the plot is almost flat in this case in Figure 3.22.

3.5.3.2 The α-Speedup

Assume that the number of processors (P) is equal to the number of evaluated points per iteration (J). Let

$$\{\underbrace{x_1, ..., x_J,}_{\text{Iter.1}} \cdots, \quad \underbrace{x_{(k-1)J+1}, ..., x_{kJ},}_{\text{Iter.}k} \cdots, \quad \underbrace{x_{(N_{\max}-J)+1}, .., x_{N_{\max}}}_{\text{Iter.}(N_{\max}/J)}\}$$

be the sequence of evaluated points for PADS1(J), where in every iteration, J function evaluations are done simultaneously. Let $n^{(\alpha)}(J)$ and $\mathcal{I}^{(\alpha)}(J)$ be the number of function evaluations and the number of iterations required by PADS1(J) to reach a specified value, called an $\alpha-$level, i.e.

$$n^{(\alpha)}(J) = \text{argmin}_i\{f(x_i) \leq \alpha \,\text{given}\, J \,\text{processors}\} \tag{3.5.4}$$

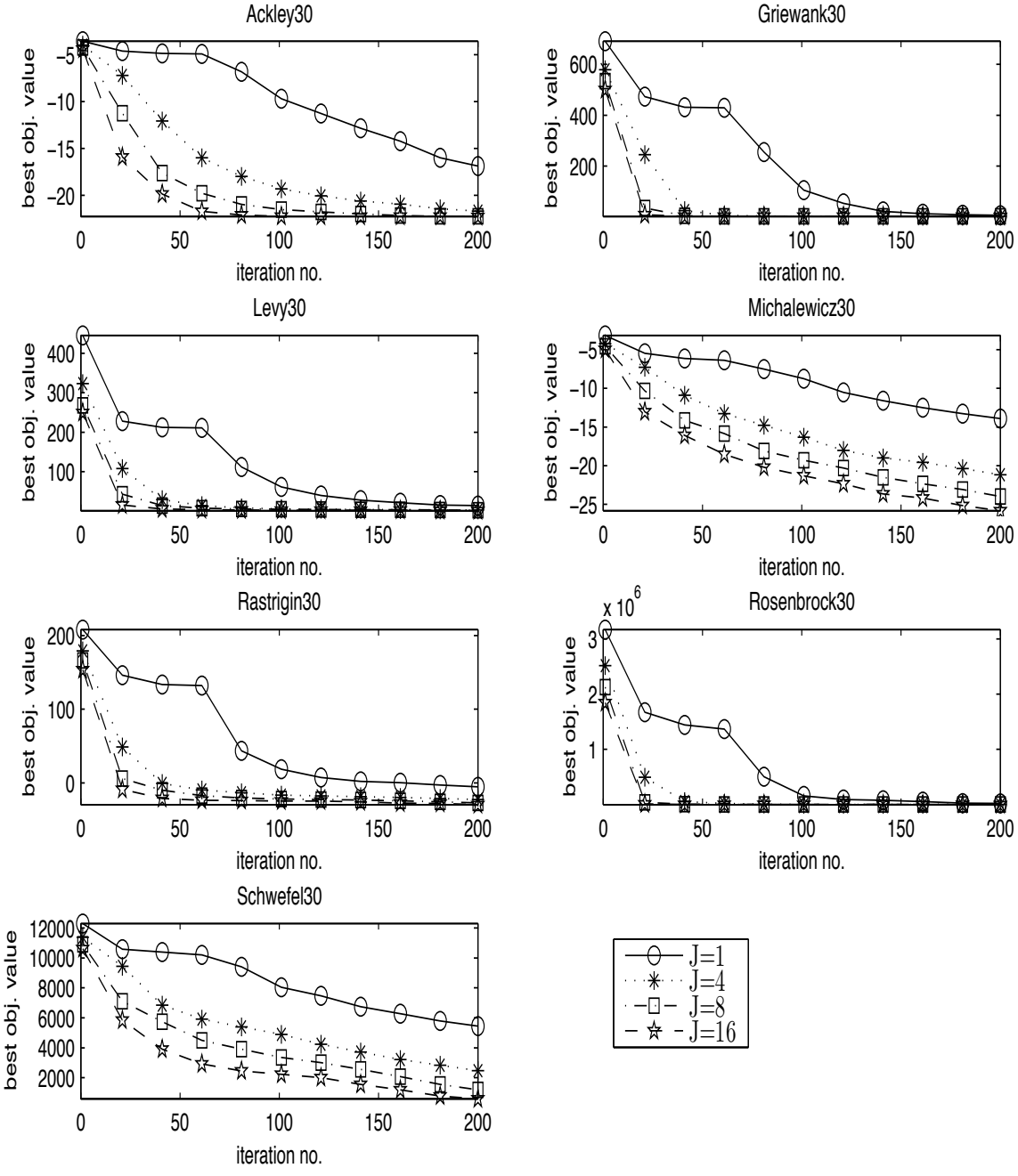


Figure 3.21: Progress plots of PADS1 for 30-dimensional problems. Shown is the average best function value vs. the number of iterations for various values of J. Iteration number is proportional to wall-clock time for parallel computation.

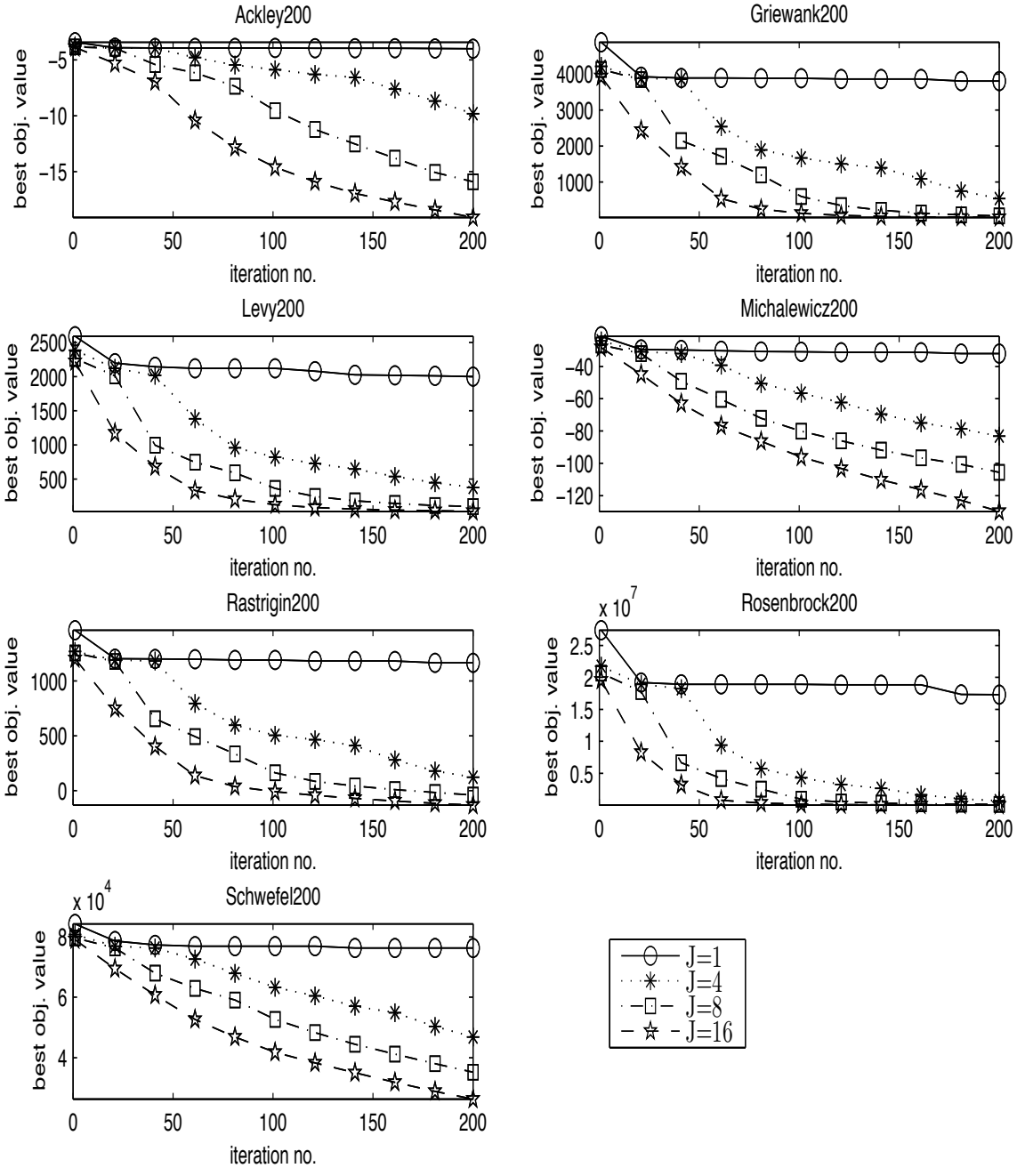


Figure 3.22: Progress plots of PADS1 for 200-dimensional problems. Shown is the average best function value vs. the number of iterations for various values of J.

and

$$\mathcal{I}^{(\alpha)}(J) = \left\lceil n^{(\alpha)}(J)/J \right\rceil . \tag{3.5.5}$$

Then, the $\alpha-$level speedup for PADS1(J) is defined by:

$$\alpha\text{-Speedup(J)} := \mathcal{I}^{(\alpha)}(1)/\mathcal{I}^{(\alpha)}(J) = n^{(\alpha)}(1)/\left\lceil n^{(\alpha)}(J)/J \right\rceil . \tag{3.5.6}$$

For each test function, the α-Speedup(J) is calculated for many different α-levels. The α-levels for each test problem are shown in Table 3.3.

For each test function, six α-levels presented in Table 3.3 are calculated using the following procedure. We ran PADS1(16) for 3200 function evaluations, and PADS1(1), PADS1(4), and PADS1(8) for 1600 function evaluations. Let y_1^*, y_4^*, y_8^* and y_{16}^* be the average best objective function values obtained from PADS1(1), PADS1(4), PADS1(8) and PADS1(16), respectively. First, we set $\alpha_6 = \max\{y_1^*, y_4^*, y_8^*, y_{16}^*\}$ so that PADS1(J) with any number of processors can reach α_6, which corresponds to the lowest level. α_1 is defined to be the average best objective function value obtained by PADS1(1) within 100 function evaluations for 30-dimensional problems and within 300 function evaluations for 200-dimensional problems. Finally, we divide the $[\alpha_6, \alpha_1]$ range into five equal subranges with $\alpha_6 < \alpha_5 < ... < \alpha_1$.

The relative distance (%) between α_6 and y_1^* is defined by

$$(\alpha_6 - y_1^*) \,/\, (y_1^w - y_1^*) \times 100 \tag{3.5.7}$$

where y_1^* and y_1^w are the average best and worst objective function values obtained from PADS1(1), respectively. This value describes relatively how close the α_6 is to the best objective value y_1^* obtained from PADS1(1) after 1600 function evaluations (scaled by the difference of y_1^* and y_1^w). Table 3.4 shows the relative distance from α_6 to y_1^* for each test function.

Figures 3.23 and 3.24 show the α-Speedup(J) for PADS1(J) versus the number of pro-

cessors ($J = 1, 4, 8, 16$). Each graph corresponds to a different α-level. When the objective function is computationally expensive, one would need to terminate the algorithm before it reaches the global minimum. Thus, the analysis of the algorithm through α-Speedup(J) is useful because it shows how much the parallel PADS1(J) when $J > 1$ is faster than a corresponding sequential algorithm PADS1(1) for different levels of α.

While the ideal speedup (α-Speedup(J) = J) is not obtained in most of the cases, we can see a relatively good scalability with PADS1. In some cases, superlinear speedup (a speedup of more than J when using J processors) occurs here. For example, at α_6-level for Schwefel-30D.

For 30-dimensional problems, Griewank-30D and Rosenbrock-30D are two cases for which the α_6-Speedup (corresponding to smaller function values) is worse than the rest of α_i-Speedup ($i < 6$). For the Griewank function, this might be due to the fact that it becomes unimodal (hence simpler) as the dimension increases [69]. For the Rosenbrock function, although it is unimodal, since the global minimum is located inside a long, narrow, parabolic flat valley, the minimizer is very difficult to find [74, 84]. Therefore, for these two test functions, once the function value gets small enough (α_6-level), it may not be necessary for PADS1 to generate many points that are far away from one another (hence far from the best point found so far) as done in the parallel algorithm, but instead focusing around the best point found so far as done in the serial algorithm.

On the other hand, the α_6-Speedup of both Michalewicz-30D and Schwefel-30D is better than the rest of α_i-Speedup ($i < 6$). Michalewicz-30D has $d!$ local optima, and the function values for points in the space outside the narrow peaks give very little information on the location of the global optimum [73, 86]. As for Schwefel-30D, since the global minimum is geometrically distant over the parameter space from the next best local minima, the algorithm is likely to converge in the wrong direction [86, 104]. These characteristics of the functions make it very hard to get a solution that is close to the global minimum within a limited number of function evaluations. In addition, as can be seen from Table 3.4, the α_6-levels of these two test functions are relatively high. With only a fixed number of evaluations

allowed, evaluating at J spread-out points in each iteration may be more useful than doing one function evaluation per iteration throughout the function evaluation budget.

As for 200-dimensional problems, the algorithm achieves very similar values of α_i-Speedup at all α_i-levels. Due to the high-dimensionality of the problems, with a limited number of iterations, the α_6-level are still relatively high compared to those for 30-dimensional problems. Therefore, in this case, it may be useful to do many function evaluations per iteration at any α_i-levels we used.

3.5.3.3 The α-Work Ratio

To analyze the performance of PADS1(J), in addition to the α-Speedup(J), we introduce the concept of the α-Work Ratio(J):

$$\alpha\text{-Work Ratio(J)} := n^{(\alpha)}(J)/n^{(\alpha)}(1) \tag{3.5.8}$$

where $n^{(\alpha)}(J)$ is defined earlier in Eq. (3.5.4) as the number of function evaluations required for algorithm PADS1(J) to reach a given α-level. This is a measure of how much the change in the algorithm to enable J simultaneous evaluations per iteration affects the total number of evaluations required to reach a given α-level. α-Work Ratio(J) $\times$ 100 is approximately (parallel efficiency)$^{-1}$, since communication times are insignificant in comparison to function evaluation times. Therefore, α-Work Ratio(J) < 1 implies the parallel efficiency is greater than 100%.

Since PADS1(J) selects J points at once for function evaluations and updates the surrogate every J function evaluations, typically the inequality $n^{(\alpha)}(J) > n^{(\alpha)}(1)$ holds, i.e. the

Table 3.3: α-levels

Test Functions	α_1	α_2	α_3	α_4	α_5	α_6
Ackley-30D	-9.162189e+00	-1.177654e+01	-1.439089e+01	-1.700524e+01	-1.961959e+01	-2.223394e+01
Griewank-30D	1.267379e+02	1.018004e+02	7.686281e+01	5.192524e+01	2.698768e+01	2.050109e+00
Levy-30D	7.775693e+01	6.264772e+01	4.753850e+01	3.242929e+01	1.732008e+01	2.210867e+00
Michalewicz-30D	-8.029906e+00	-1.068863e+01	-1.334735e+01	-1.600607e+01	-1.866479e+01	-2.132352e+01
Rastrigin-30D	2.353054e+01	1.387214e+01	4.213748e+00	-5.444647e+00	-1.510304e+01	-2.476144e+01
Rosenbrock-30D	1.616795e+05	1.293796e+05	9.707968e+04	6.477976e+04	3.247983e+04	1.799128e+02
Schwefel-30D	8.485614e+03	7.403291e+03	6.320968e+03	5.238646e+03	4.156323e+03	3.074001e+03
Ackley-200D	-5.441320e+00	-7.533715e+00	-9.626109e+00	-1.171850e+01	-1.381090e+01	-1.590329e+01
Griewank-200D	1.831223e+03	1.476137e+03	1.121050e+03	7.659644e+02	4.108783e+02	5.579217e+01
Levy-200D	9.417298e+02	7.736041e+02	6.054784e+02	4.373527e+02	2.692270e+02	1.011013e+02
Michalewicz-200D	-4.943077e+01	-6.066993e+01	-7.190908e+01	-8.314823e+01	-9.438739e+01	-1.056265e+02
Rastrigin-200D	5.282097e+02	4.146363e+02	3.010628e+02	1.874894e+02	7.391591e+01	-3.965754e+01
Rosenbrock-200D	4.867998e+06	3.918320e+06	2.968641e+06	2.018962e+06	1.069284e+06	1.196049e+05
Schwefel-200D	6.777597e+04	6.125220e+04	5.472844e+04	4.820468e+04	4.168091e+04	3.515715e+04

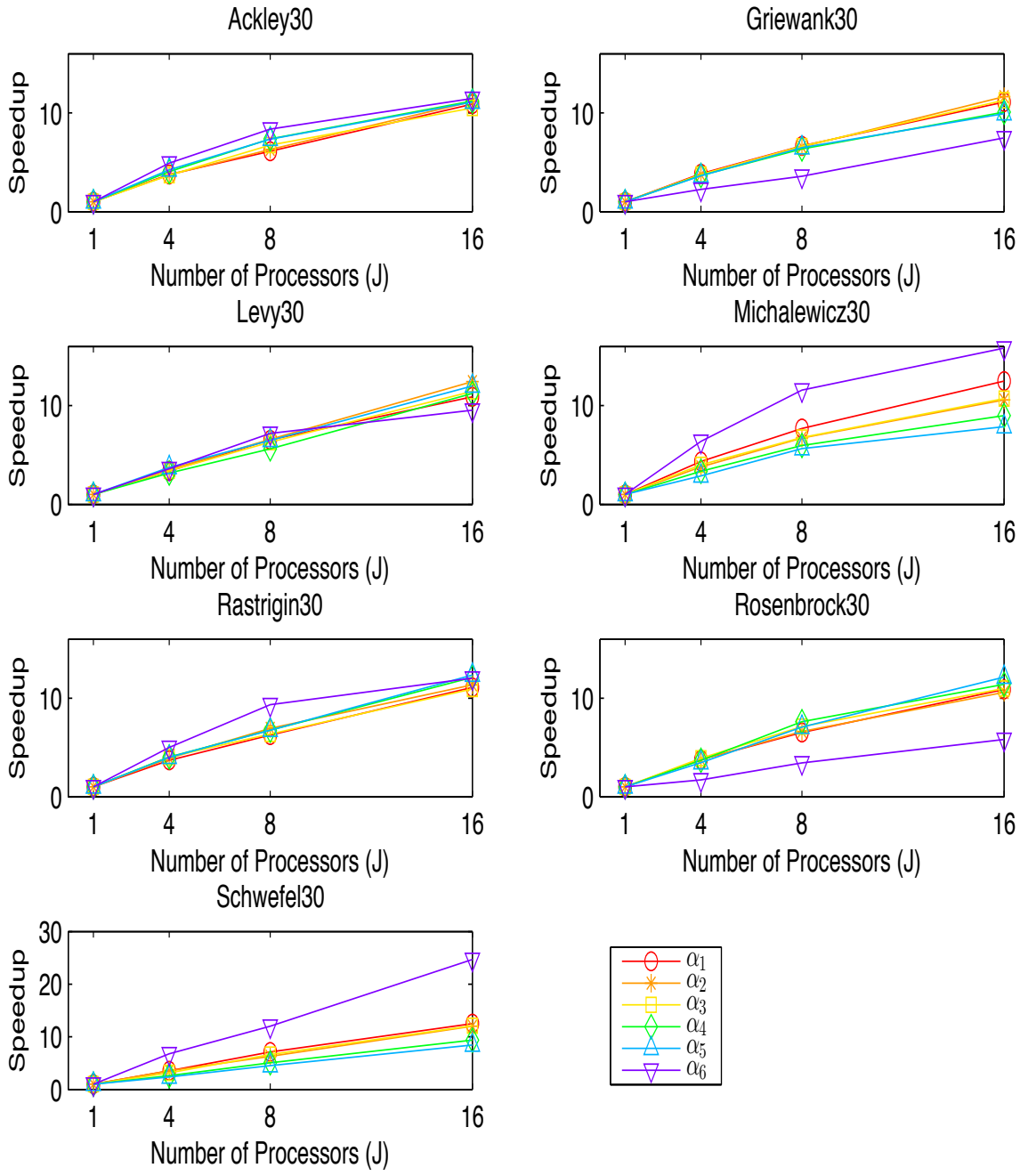


Figure 3.23: α-Speedup(J) of PADS1(J), $J = 1$, 4, 8, 16, for 30-dimensional problems

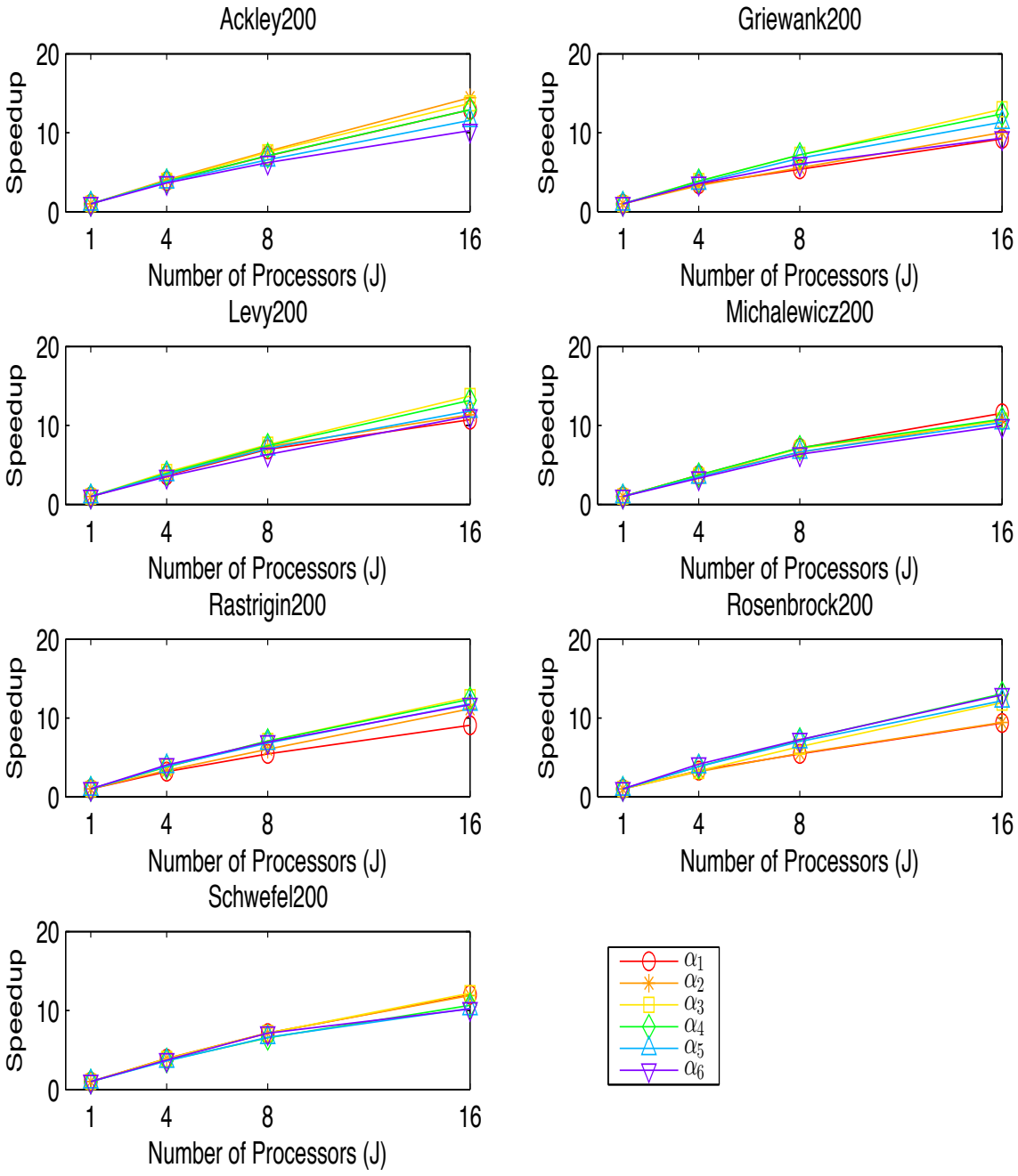


Figure 3.24: α-Speedup(J) of PADS1(J), $J = 1$, 4, 8, 16, for 200-dimensional problems

Table 3.4: The relative distance (%) from α_6 to y_1^*

Test Functions	30-D	200-D
Ackley	0.01	17.02
Griewank	0.01	0.84
Levy	0.06	1.41
Michalewicz	0.00	10.75
Rastrigin	0.00	1.74
Rosenbrock	0.00	0.11
Schwefel	0.00	5.00

total work (e.g. number of expensive evaluations) done by J processors to reach the given α-level for PADS1(J) is more than the work done by a single processor for serial PADS1(1).

Figures 3.25 and 3.26 show a plot of the α-Work Ratio(J) versus the number of processors. As in α-Speedup(J), each graph corresponds to a different α-level (Table 3.3). We see that for any fixed number of processors and for all α-levels, the α-Work Ratio(J) is always less than J. The α-Work Ratio(J)'s for all the problems are in fact bounded above by 3. Moreover, in some cases (e.g. Schwefel30 with $\alpha_6 \approx 300$), the α-Work Ratio(J) is below 1 for any $J > 1$. That is to say, the algorithm can reach that particular α-level with less work when J is increasing.

3.6 Conclusions

In this chapter we have developed PADS, a new parallel global optimization framework, which is empirically proven to be a promising algorithm for HEB problems. In the numerical experiments, we introduced two algorithms PADS1(J) and PADS2(J). Parallelism is exploited when doing the expensive objective function evaluations at $J > 1$ selected points simultaneously in every iteration.

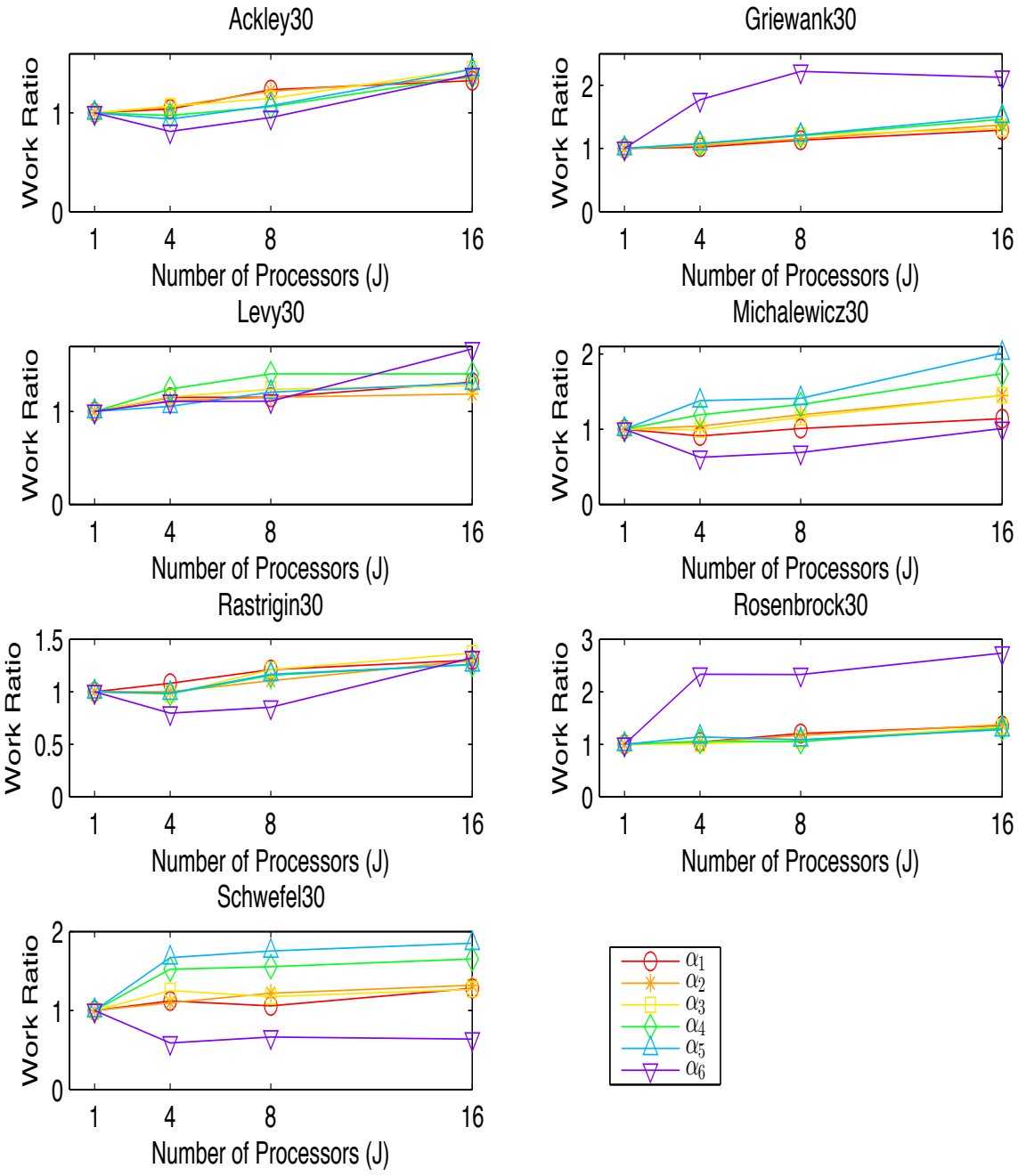


Figure 3.25: α-Work Ratio(J) of PADS1 for 30-dimensional problems

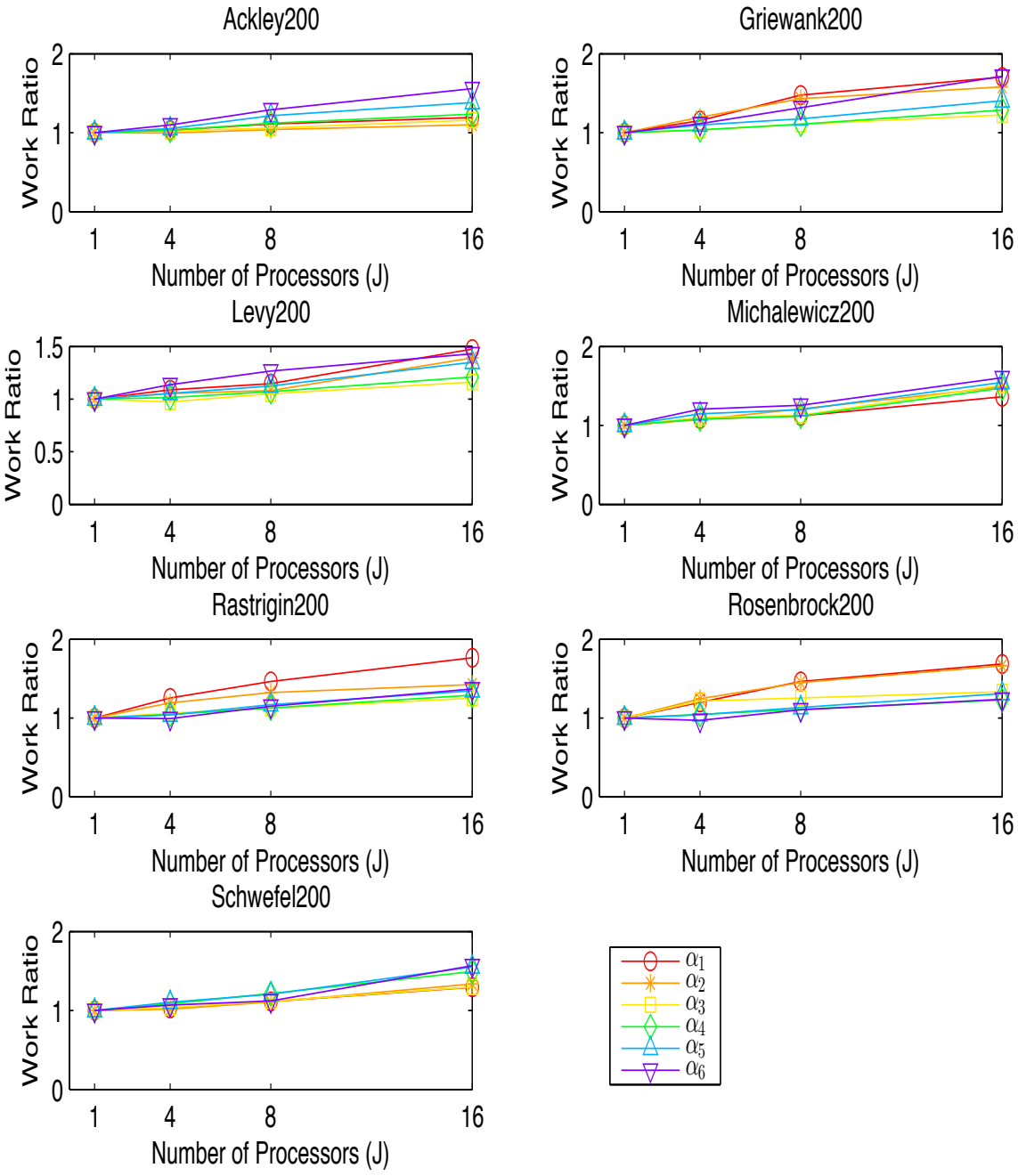


Figure 3.26: α-Work Ratio(J) of PADS1 for 200-dimensional problems

We compared the serial versions of PADS to two evolutionary algorithms, namely Differential Evolution and ESGRBF, as well as to DYCORS, LMSRBF, NOMADm-DACE and NOMADm-RBF. We compared the parallel versions of PADS to ParDE, ParESGRBF, ParLMSRBF, and PSD-MADS. While there are a few parallel algorithms available for MADS methods, we used PSD-MADS as it was shown to be well suited for high dimensional problems. These serial and parallel methods (with $J = 4$, 8, and 16 processors) are compared on 7 test problems, each with 30 and 200 dimensions, respectively. In addition to these mentioned methods, TOMLAB/rbfSolve was also used to solve the 30-dimensional problems.

The algorithms are compared in terms of the average (over multiple trials) best objective function value after an equal number of iterations, where the number of iterations corresponds to the number of function evaluations divided by the number of parallel processors. The results indicate that both PADS1 and PADS2 are more efficient for HEB problems than the alternative algorithms. The computation results show that PADS1 and PADS2 perform very similar for the 30-dimensional problems and that PADS1, which is based on the truncated Gaussian candidate points, performs better than PADS2 for the 200-dimensional problems. In addition, the analysis through α-Speedup and α-Work Ratio encourages the use of PADS1 for selecting multiple points per iteration for simultaneous function evaluation.

We provided a theorem that shows that the PADS algorithms converge to the global optimum with probability one under some mild conditions. This also proves the conjecture about the convergence of DYCORS which was raised but was not yet verified in [97].

Chapter 4

Surrogate Optimization with Pareto Center Selection for Single Objective Problems*

4.1 Introduction

Real-world applications in various fields, such as science, physics, engineering, or economics often have a simulation model which is multimodal, computationally expensive, and black-box. "Blackbox" implies many mathematical characteristics are not known including derivatives or number of local minima. Many existing optimization methods for blackbox functions such as genetic algorithm, simulated annealing, or pattern search are not suitable for this type of problem due to a large number of objective function evaluations that these methods generally require.

One approach for dealing with this type of problem is to use a surrogate model (alternatively called metamodel or response surface) to approximate the objective function. Response surface based optimization methods start by building a (computationally inexpen-

*This chapter will be published with authors T. Krityakierne, T. Akhtar, C.A. Shoemaker.

sive) surrogate surface, which is then used to iteratively select new points for the expensive function evaluation. The surrogate surface is updated in each iteration.

The purpose of this research is to present a new way to solve surrogate optimization problems for computationally expensive functions in parallel. Previous efforts (mostly serial) have usually involved generating candidate points by normal perturbations around one center point (usually the best solution found so far), by uniform sampling in the domain, or by using an optimization method on the surrogate to find the point that satisfies some criterion. In this work, we use a different approach involving non-dominated sorting on previously evaluated points to select multiple centers, which are points on the Pareto fronts. Hence, we are using concepts from multi-objective optimization for single objective optimization of computationally expensive functions.

4.1.1 Literature Review

Jones et al. [57] used kriging as a basis function to develop a global optimization method, Efficient Global Optimization (EGO), where the next point to evaluate is obtained by maximizing an expected improvement, balancing the response surface value with the uncertainty of the surface. Huang et al. [54] extended Jones' EGO algorithm and developed a global optimization method for noisy cost functions. Booker et al. [22] also used kriging surface to speed up the pattern search algorithms. Gutmann [45] built the response surface with radial basis functions where the next point to evaluate is obtained by minimizing the bumpiness of the interpolation surface. Sóbester et al. [111] used Gaussian radial basis functions and proposed weighted expected improvement to control the balance of exploitation and exploration to select the next evaluation point.

In [92], Regis and Shoemaker developed the CORS-RBF method based on radial basis functions. The next point chosen for the expensive function evaluation is the point $x \in \mathcal{D}$ that minimizes the CORS-RBF auxiliary optimization subproblem (an RBF model subject to the constraints that x is at least of distance τ_n away from each of the previously evaluated points,

where the threshold τ_n depends on all previously evaluated points). Regis and Shoemaker [95] also used radial basis functions in the Metric Stochastic Response Surface algorithm (also known as the Stochastic RBF algorithm), which is a global optimization algorithm wherein the next point to evaluate is chosen by randomly generating a large number of candidate points and selecting the point that minimizes a weighted score of response surface predictions and a distance metric. Wild et al. [124, 125] developed ORBIT (Optimization by Radial Basis Interpolation in Trust-regions), which is a method for unconstrained local optimization, using radial basis function models and utilizing a trust-region framework. The multistart global optimization algorithm GORBIT that runs ORBIT from multiple start points was introduced in [126].

Due to the advanced technology of parallel computing, there has recently been an interest in developing surrogate algorithms that select and evaluate multiple (computationally expensive) points in each iteration to reduce the wall-clock time ($\propto$ the number of optimization iterations). For example, Sóbester et al. [110] developed a parallel version of EGO. Several local maximum points of the expected improvement function are selected for the parallel computationally expensive evaluations. In [94], Regis and Shoemaker developed parallel versions of two global optimization methods by radial basis functions: (1) Parallel Gutmann-RBF method [93] and (2) Parallel CORS-RBF method [92]. In the first approach, P distinct target values are chosen and the next P function evaluation points are obtained by solving a corresponding minimization problem for the bumpiness of the interpolation surface. In the second approach, the next P function evaluation points are chosen by solving the CORS-RBF auxiliary optimization subproblem sequentially. In 2009, Regis and Shoemaker [96] proposed a parallel version of the stochastic RBF algorithm [95]. In each iteration, a fixed number of points are selected for doing the expensive function evaluations. The selection is done sequentially and based on the weighted score of (1) the surrogate value, and (2) the minimum distance from previously evaluated points and previously selected points within that parallel iteration, until the desired number of points are selected. The expensive

function evaluations at the selected points are done simultaneously. The experimental results showed that the algorithm is very efficient compared to their previous methods.

Viana et al. [120] proposed MSEGO that is based on the idea of maximizing the expected improvement (EI) functions, but instead of using just one surrogate as in [57], multiple surrogates are used. Since different EI functions are obtained for different surrogates, multiple points can be selected per iteration. Although MSEGO was shown to reduce the number of iterations, the numerical convergence rate does not scale up with the number of points selected in each iteration for evaluation, i.e. the results when using five and ten surrogates per optimization iteration were very similar. Recently, Bischl et al. [14] proposed MOI-MBO which is a parallel kriging based optimization algorithm that uses a multiobjective infill criterion that rewards the diversity and the expected improvement for selecting the next expensive evaluation points. Many versions of MOI-MBO were proposed based on various multiobjective infill criteria: mean of the surrogate model, model uncertainty, expected improvement, distance to the nearest neighbor, and the distance to the nearest better neighbor. Evolutionary optimization was used to handle the embedded multiobjective optimization problem. The overall test results suggested that the version that used a combination of mean, model uncertainty, and nearest neighbor worked best. More reviews on techniques for selecting multiple points as well as parallel surrogate-based methods can be found in [14].

4.1.2 Differences between SOP and Previous Algorithms

SOP (**S**urrogate **O**ptimization with **P**areto center selection) is a parallel surrogate assisted algorithm, where many points can be chosen simultaneously for computationally expensive evaluations. Furthermore, the search mechanism of the algorithm incorporates bi-objective search, tabu search, and surrogate assisted local search.

The major difference between SOP and other existing parallel optimization algorithms is the use of a Pareto non-dominated sorting technique to select P distinct evaluated points

(whose function values are small and are far away from other evaluated points) which will then serve as centers. The selected centers are then used for generating a set of candidate points from which the next function evaluation points are chosen. The latter step can be done in parallel. This contrasts with the approach Regis and Shoemaker [96] used to generate the P points for expensive function evaluations. In [96], all the P points are obtained from the same center that is the best point found so far (x_{best}). Selecting the P points from different centers (as was done in this work) allows the algorithm to search more globally simultaneously in each iteration. The method proposed in [14] is also based on the multiobjective optimization. However, their embedded multiobjective optimization problem was solved on the surrogate while in our approach, a bi-objective optimization problem is solved on previously evaluated expensive function evaluation points.

We found no journal papers on surrogate global optimization that select a large number of evaluation points in each iteration. For example, the maximum number of points used in [14], [96], and [120] were 5, 8, and 10 points, respectively. On the other hand, SOP can do many expensive objective function evaluations simultaneously. (SOP was tested on as many as 64 points per parallel iteration.) SOP thus greatly reduces wall-clock time.

The structure of this chapter is as follows: Section 4.2 gives a general framework for surrogate model based optimization and summarizes the necessary basics of bi-objective optimization. In Section 4.3, we describe SOP in detail. We give sufficient conditions for almost sure convergence of SOP in Section 4.4. In Section 4.5, we illustrate the performance of SOP and compare algorithms on a number of test functions as well as a groundwater bioremediation problem. Lastly, we give our concluding remarks in Section 4.6.

Algorithm 4.1 General framework: P surrogate-based optimization

Input: initial set of n_0 evaluated points and their objective function values:
$S^{(0)} = \left\{x_1^{(0)}, ..., x_{n_0}^{(0)}\right\}$, $T^{(0)} = \left\{f(x_1^{(0)}), ..., f(x_{n_0}^{(0)})\right\}$

1: initialize $n = 0$
2: **while** $n <$ MAXIT
3: fit or update surrogates $g_n(\cdot)$ to the data $(S^{(n)}, T^{(n)})$;
4: select new P points $\left\{x_1^{(n+1)}, \ldots, x_P^{(n+1)}\right\}$ using $g_n(\cdot)$ and other criteria;
5: evaluate $y_i^{(n+1)} = f\left(x_i^{(n+1)}\right)$ for $i = 1, ..., P$;
6: update: $S^{(n+1)} = S^{(n)} \cup \left\{x_1^{(n+1)}, ..., x_P^{(n+1)}\right\}$;
7: $T^{(n+1)} = T^{(n)} \cup \left\{y_1^{(n+1)}, ..., y_P^{(n+1)}\right\}$;
8: $n = n + 1$;
9: **end** while

Output: $y_{\text{best}} = \min(T^{(n)})$, $x_{\texttt{best}} = \left\{x \in S^{(n)} : f(x) = y_{\texttt{best}}\right\}$

4.2 Background

4.2.1 General Framework for Surrogate Model Based Optimization

The general framework of the surrogate model based optimization method is presented in Algorithm 4.1. First, the algorithm constructs a surrogate based on n_0 initial data points. The algorithm selects the next P points for function evaluations using algorithm-specific selection criteria, e.g. model value, model uncertainty. After the P promising points are selected, they will be evaluated and if the budget of allowable expensive function evaluations is not exhausted, the algorithm then uses all (or some) of the evaluated points to update the surrogate. One of the most important aspects of the surrogate-based optimization algorithm is how to effectively select the next P points for expensive evaluations for simultaneous evaluation on multiple processors.

4.2.2 Bi-objective Optimization

We now give a brief review of some concepts related to bi-objective optimization that will be used in Section 4.3 for the selection of already evaluated points in the main structure of the algorithm. Here we simultaneously consider two conflicting objective functions of the form:

$$\begin{matrix} \min\left[F_1(x),\, F_2(x)\right] \\ \text{subject to}\, x \in \mathcal{S} \end{matrix} . \tag{4.2.1}$$

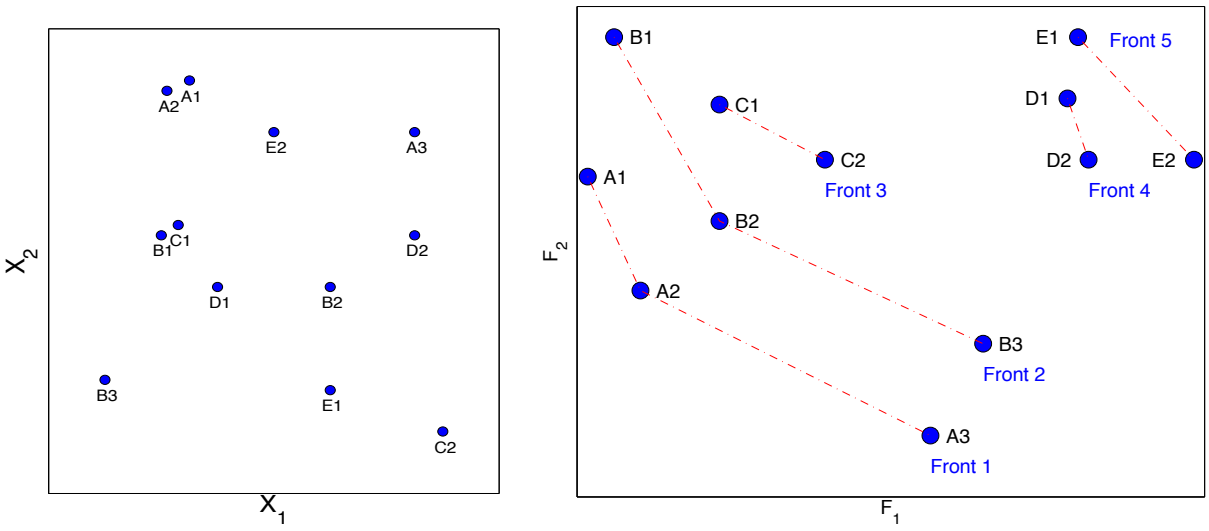
The goal of bi-objective optimization problems is to find the best compromise between these two conflicting objectives. Because we cannot usually find a solution which is best with respect to both objectives, we instead seek for a set of solutions such that for any points in this set, there does not exist another point that will improve one attribute without degrading another attribute. That is, we face with a set of non-dominated solutions, which is known as Pareto optimal or non-dominated solutions. For more details see [32, 121].

4.2.2.1 Non-dominated solutions and Pareto front

Definition 4.1. A point $x_1 \in \mathcal{S}$ is said to **dominate** another point $x_2 \in \mathcal{S}$ if (1) the point x_1 is no worse than x_2 in both objectives, and (2) the point x_1 is strictly better than x_2 in at least one objective. We denote this by "$x_1 \preceq x_2$". Mathematically, for $x_1,\, x_2 \in \mathcal{S}$,

$$x_1 \preceq x_2 \Leftrightarrow \begin{cases} F_i(x_1) \leq F_i(x_2) & \forall i \in \{1,\, 2\} \\ \\ \exists j \in \{1,\, 2\} & F_j(x_1) < F_j(x_2) \end{cases} . \tag{4.2.2}$$

The set of all non-dominated solutions in the decision space is called **Pareto set** ($\mathcal{X}^*$). The image of the Pareto set is called **Pareto front** ($\mathcal{P}^*$). That is, $\mathcal{X}^* = \{x \in \mathcal{S} :\, \nexists x' \in \mathcal{S},\, x' \preceq x\}$ and $\mathcal{P}^* = F\left[\mathcal{X}^*\right]$. Note that $F(x_1) \preceq F(x_2)$ is used to indicate $x_1 \preceq x_2$ for $F(x)$.



(a) Decision space

(b) Objective space

Figure 4.1: A two-objective space and the corresponding non-dominated fronts

4.2.2.2 Non-dominated sorting

The procedure for sorting points into levels of fronts is referred as non-dominated sorting, which was introduced by Deb et. al [31, 33]. To sort points into levels of non-dominated fronts, first, the Pareto front of non-dominated points is determined based on all available points (Front 1 in Figure 4.1b). Next, points in Front 1 (Pareto front) are removed and the non-dominated front is recalculated for the remaining points to determine the points of Front 2. Continue the process to get Front k by removing all points on Front 1 to Front k-1 and calculating the non-dominated front from the remaining points. Figure 4.1 shows an example of a two-objective space and the corresponding non-dominated fronts.

4.2.2.3 The Hypervolume

The hypervolume (S-metric or Lebesgue measure) is a measure used to indicate the quality of the Pareto front. It was initially proposed by Zitzler and Thiele [134], in which it was called the size of dominated space. The hypervolume is widely used and many methods have

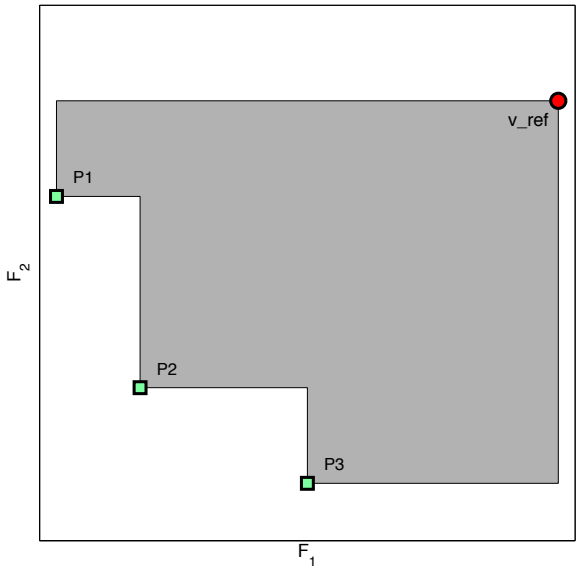


Figure 4.2: Hypervolume

been developed to efficiently calculate the hypervolume (e.g. [36, 123]). Figure 4.2 shows an example of the hypervolume of a non-dominated set $\mathcal{P} = \{\text{P1}, \text{P2}, \text{P3}\}$ with respect to the reference point v_{ref}.

Definition 4.2. Given a set $\mathcal{P} \subset \mathbb{R}^2$ of non-dominated solutions in the objective space, the hypervolume $\mathbf{HV}(\mathcal{P}, v_{\text{ref}})$ of $\mathcal{P}$ is a measure relative to a reference point $v_{\text{ref}} \in \mathbb{R}^2$. It is an area that is dominated by the points in $\mathcal{P}$. Mathematically,

$$\mathbf{HV}(\mathcal{P}, v_{\mathbf{ref}}) = \mathbf{Vol}\left(\cup_{p \in \mathcal{P}} [p(1), v_{\mathbf{ref}}(1)] \times [p(2), v_{\mathbf{ref}}(2)]\right) \tag{4.2.3}$$

where **Vol** denotes the Lebesgue measure, and for sets A and B, $A \times B$ denotes the Cartesian product, i.e. $A \times B = \{(a, b) : a \in A \wedge b \in B\}$.

The hypervolume indicator has been the only measure that is known to have the property that [39, 132, 135]

$$A \preceq B \Rightarrow \mathbf{HV}(A, v_{\text{ref}}) > \mathbf{HV}(B, v_{\text{ref}}), \tag{4.2.4}$$

where $A \preceq B$ if and only if $\forall y \in B \, \exists x \in A : \, x \preceq y$. That is the dominance relation is extended to a relation on sets A, $B \subset \mathcal{S}$.

If a solution set $\mathcal{P}$ has a larger hypervolume than $\mathcal{P}'$, then $\mathcal{P}$ is said to have a better set of solutions than $\mathcal{P}'$ as it is likely to give a better trade-offs. Maximizing the hypervolume is equivalent to finding the Pareto set [38]. See e.g. [133] for more details of hypervolumes as well as other measures used for Pareto set comparison.

4.3 SOP: Surrogate Optimization with Pareto center selection

The objective function we consider is a box-constrained real-valued function as defined in Eq. (1.1). Algorithm 4.2 shows the SOP algorithm, which is based on the framework presented in Algorithm 4.1. Below are inputs and output of the algorithm.

Input:

- N_{cand} - the number of candidate points generated in Local Candidate Search
- N_{fail} - the maximum number of non-registered improvement before a particular center will be declared as tabu
- m - the tenure length
- r_{int} - the initial search radius
- τ - the tolerance for local improvement
- P - the number of centers per iteration (= the number of function evaluation points per iteration)
- MAXIT - the maximum number of optimization iterations

- f - a continuous real-valued objective function to minimize
- $\mathcal{D} = \{\text{lb} \leq x \leq \text{ub}\} \subset \mathbb{R}^d$ - the hyperrectangle domain of f

Output: The best point found

List of notation used in Algorithm 4.2:

- $x_i^{(n)}$ the ith evaluation point in nth optimization iteration ($1 \leq i \leq P$)
- $y_i^{(n)}$ the function value corresponding to $x_i^{(n)}$
- $S^{(n)}$ the list (ordered set) containing all evaluated points up to nth iteration, i.e. $S^{(n)} = \langle s_1, ..., s_{n_0+nP} \rangle = \left\langle \underbrace{x_1^{(0)}, ..., x_{n_0}^{(0)}}_{\text{initial pts}}, \underbrace{x_1^{(1)}, ..., x_P^{(1)}}_{\text{iteration 1}}, \cdots, \underbrace{x_1^{(n)}, ..., x_P^{(n)}}_{\text{iteration } n} \right\rangle$. Note $\left|S^{(n)}\right| = n_0 + nP$.
- $\lambda_i^{(n)}$ the minimum distance from $x_i^{(n)}$ to the set of already evaluated points $S^{(n)} \setminus \left\{x_i^{(n)}\right\}$
- $T^{(n)}$, $D^{(n)}$, $R^{(n)}$ a similar list to $S^{(n)}$ containing all function values, minimum distances, and search radii.
- $\langle \cdot \rangle$ is used to distinguish a list from a regular set.

We treat $S^{(n)}$, $T^{(n)}$, $D^{(n)}$, $R^{(n)}$ as a list. That is, the order of the members in these lists is important. Whenever we write $s_i \in S^{(n)}$ for example, we do not take into account in which iteration the point s_i is simulated. On the other hand, $x_i^{(n)}$ tells us that this point is generated as the ith point in the nth iteration. Note the relationship: $s_{n_0+((n-1)P+i)} = x_i^{(n)}$ for $n \geq 1$ and $s_i = x_i^{(0)}$ holds. For example, $s_1 = x_1^{(0)}, ..., s_{n_0} = x_{n_0}^{(0)}$, $s_{n_0+1} = x_1^{(1)}, ..., s_{n_0+P} = x_P^{(1)}$, etc. Similar argument applies to members in other lists. In addition, tabu_struct consists of two lists tabu_struct(1) and tabu_struct(2), corresponding to s_i. We now explain in detail each of the steps in Algorithm 4.2:

Algorithm 4.2 Surrogate Optimization with Pareto center selection (SOP)

1. Set counter $n = 0$. Generate initial points $S^{(0)} = \left\langle x_1^{(0)}, ..., x_{n_0}^{(0)} \right\rangle$ and do function evaluations $y_i^{(0)} = f\left(x_i^{(0)}\right)$ for $i = 1, ..., n_0$. Calculate $d_i = \min\left\{\|s_i - s\| : s \in S^{(0)} \setminus \{s_i\}\right\}$, for $s_i \in S^{(0)}$. Let $T^{(0)} = \left\langle y_1^{(0)}, ..., y_{n_0}^{(0)} \right\rangle$, $D^{(0)} = \langle d_1, ..., d_{n_0} \rangle$ and $R^{(n_0)} = \langle \sigma_1, ..., \sigma_{n_0} \rangle$, where $\sigma_i = r_{\text{int}}$. In addition, let tabu_struct = [tabu_struct(1), tabu_struct(2)], where tabu_struct(1) = tabu_struct(2) $= \langle z_1, ... z_{n_0} \rangle$, where $z_i = 0$ for all $i = 1, ..., n_0$.

2. Define $F = (F_1, F_2) : S^{(0)} \to T^{(0)} \times -D^{(0)}$ by $F(s_i) = (t_i, -d_i)$ for $s_i \in S^{(0)}$, $t_i \in T^{(0)}$ and $d_i \in D^{(0)}$.

3. Iterative Loop (**while** $n <$ MAXIT)

   (a) Fit or update the response surface model $g^{(n)}(\cdot)$.

   (b) Find $[\mathcal{X}_{\text{int}}, \mathcal{P}_{\text{int}}] =$ **pareto_front**$\left(S^{(n)}, F_1, F_2\right)$.

   (c) Select P center points:

      i. $\mathcal{J} = [\mathcal{J}_1, ..., \mathcal{J}_P] =$ **select_P_indices**$(S^{(n)}, T^{(n)}, D^{(n)}, R^{(n)}$, tabu_struct, $P)$.

      ii. For $i = 1, ..., P$, $c_i = s_{(\mathcal{J}(i))} \in S^{(n)}$ is the selected center point and $r_i = \sigma_{(\mathcal{J}(i))} \in R^{(n)}$ the search radius of center i.

   (d) $p_{\textbf{select}} = \varphi(n)$. **for** $i = 1 : P$ **do**

   Generate a new point $x_i^{(n+1)} = ...$

   local_candidate_search$\left(c_i, r_i, \mathcal{D}, N_{\text{cand}}, g^{(n)}(\cdot), p_{\text{select}}\right)$.

   Evaluate $y_i^{(n+1)} = f(x_i^{(n+1)})$.

   end for

   Update $S^{(n+1)} = S^{(n)} \cup \left\langle x_1^{(n+1)}, ..., x_P^{(n+1)} \right\rangle$ and $T^{(n+1)} = T^{(n)} \cup \left\langle y_1^{(n+1)}, ..., y_P^{(n+1)} \right\rangle$.

   Calculate $d_i = \min\left\{\|s_i - s\| : s \in S^{(n+1)} \setminus \{s_i\}\right\}$ for all $s_i \in S^{(n+1)}$.

   Redefine $D^{(n+1)} = \left\langle d_1, ..., d_{n_0+(n+1)P} \right\rangle$.

   Update $F = (F_1, F_2) : S^{(n+1)} \to T^{(n+1)} \times -D^{(n+1)}$ by $F(s_i) = (t_i, -d_i)$ for $s_i \in S^{(n+1)}$, $t_i \in T^{(n+1)}$, and $d_i \in D^{(n+1)}$.

   (e) **for** $i = 1 : P$ **do**

   $\lambda_i^{(n+1)} = d_{n_0+(nP+i)} \in D^{(n+1)}$.

   $I_i =$ **hypervol_improv_indc**$\left(\mathcal{X}_{\text{int}}, \mathcal{P}_{\text{int}}, F_1, F_2, x_i^{(n+1)}, \left[y_i^{(n+1)}, \lambda_i^{(n+1)}\right], \tau\right)$.

   end for

   (f) Update $R^{(n+1)} = R^{(n)} \cup \langle \sigma_1, ..., \sigma_P \rangle$, where $\sigma_i = r_{\text{int}}$, and tabu_struct(1) = tabu_struct(1)$\cup\langle z_1, ... z_P \rangle$, tabu_struct(2) = tabu_struct(2)$\cup\langle z_1, ... z_P \rangle$, where $z_i = 0$. Set $n = n + 1$.

   (g) Update $\left[\text{tabu\_struct}, R^{(n)}\right] =$ **update**$\left(\text{tabu\_struct}, I, \mathcal{J}, N_{\text{fail}}, m, r_{\text{int}}, R^{(n)}, P\right)$.

4. Terminate and return the best point found.

Step 1: n_0 initial points are generated and expensive function evaluations are done at these points. Let $S^{(n)}$ be the list of evaluated points so far and $T^{(n)}$ the list of corresponding expensive function values. d_i is the minimum distance of each $s_i \in S^{(n)}$ from other points in $S^{(n)}$, where $\|\cdot\|$ is the Euclidean norm on $\mathbb{R}^d$. Then, $D^{(n)}$ is defined to be the list of such distances. $R^{(n)}$ is the list of corresponding search radii (initially set to r_{int}). Finally, tabu_struct is a pair of lists of tabu structure. Initially, all these lists consist of n_0 elements.

Step 2: defines a bi-objective function over a finite set, $F = (F_1, F_2) : S^{(0)} \to T^{(0)} \times -D^{(0)}$, where F_1, F_2 map each member (in order) of the list $S^{(0)}$ onto the lists $T^{(0)}$, $-D^{(0)}$, respectively. Here, instead of maximizing the minimum distance from points already evaluated, we minimize its negative. Thus, the second objective is defined on the set $-D^{(0)}$. See Section 4.3.1 for a detailed discussion of this bi-objective function.

Note since the values $t_i = f(s_i)$ and d_i for each point $s_i \in S^{(0)}$ are pre-computed and stored in $T^{(0)}$ and $D^{(0)}$, respectively, no additional expensive evaluation of f or distance calculation is required in this step.

Step 3 is the iterative step, starting from:

- Step 3a: the algorithm fits or updates the surrogate $g^{(n)}(\cdot)$ based on previously evaluated points in $S^{(n)}$ and their corresponding function values in $T^{(n)}$.

- Step 3b: finds the Pareto front of $S^{(n)}$ based on two objectives F_1 and F_2. The function **pareto_front** is defined as in Definition 4.1.

- Step 3c: selects the P center points $c_1, ..., c_P$ from the list $S^{(n)}$. The algorithm keeps track of the original index of each of the point $s_{(\cdot)}$ in the list $S^{(n)} = \langle s_1, \ldots, s_{n_0+nP} \rangle$ that are selected as centers. That is, $\mathcal{J} = [\mathcal{J}_1, ..., \mathcal{J}_P] \subset \{1, ..., n_0 + P\}^P$ is a vector of indices such that $c_i = s_{(\mathcal{J}(i))} \in S^{(n)}$ for $i = 1, ..., P$. Also, the search radius of center i, $r_i = \sigma_{(\mathcal{J}(i))} \in R^{(n)}$, is defined accordingly. The pseudocode for function **select_P_indices** is shown in Figure 4.3. The **select_P_indices** is based on the sub-functions **non_dom_sorting** (Figure 4.7) and **P_centers_sel** (Figure 4.8).

Since the selection of P center indices is the key step and is based on the concept of bi-objective optimization over a finite set $S^{(n)}$, we postpone the details of these two sub-functions until Section 4.3.1 so as not to interrupt the flow of the main algorithm.

Function $\mathcal{J} = [\mathcal{J}_1, ..., \mathcal{J}_P] =$ **select_P_indices**$(S^{(n)}, T^{(n)}, D^{(n)}, R^{(n)}$, tabu_struct, $P)$

1. $\left[S^{\text{order}}, R^{\text{order}}, \text{tabu\_struct}^{\text{order}}\right] = ...$
   non_dom_sorting$\left(S^{(n)}, T^{(n)}, D^{(n)}, R^{(n)}, \text{tabu\_struct}\right)$
2. $[c_1, ..., c_P] =$ **P_centers_sel**$\left(S^{\text{order}}, R^{\text{order}}, \text{tabu\_struct}^{\text{order}}, P\right)$
3. For $j = 1, ..., P$, let $\mathcal{J}(j)$ be such that $c_j = s_{\mathcal{J}(i)} \in S^{(n)}$. That is, $\mathcal{J}$ is the P-vector containing the indices of the selected centers corresponding to the original list $S^{(n)}$.

Figure 4.3: Pseudocode **select_P_indices** used in Step 3(c)i

- Step 3d: the algorithm updates $\varphi(n)$ by

$$\varphi(n) = \varphi_0 \times [1 - \ln(nP + 1)/\ln(\text{MAXIT} \times P)], \tag{4.3.1}$$

for all $0 \leq n \leq \text{MAXIT} - 1$, where $\varphi_0 = \min(20/d, 1)$.

This formula is adopted from [97] with n denoting here the number of optimization iterations rather than the number of function evaluations. The probability $p_{\text{select}} = \varphi(n)$ is an input of the function **local_candidate_search** (Figure 4.10). Each coordinate of the center has a probability of $\varphi(n)$ to be perturbed and any candidate point is generated by perturbing a subset of the variables of the center. This is defined precisely in Figure 4.10 (see Section 4.3.3).

After the probability of selection is updated, **local_candidate_search** is applied around each assigned center c_i to determine the next function evaluation points $x_i^{(n+1)}$. Two types of local candidate search are introduced in Section 4.3.3. The expensive function evaluations $y_i^{(n+1)} = f(x_i^{(n+1)})$ are done at each of the P selected points.

The algorithm updates $S^{(n+1)}$ and $T^{(n+1)}$. The distance metric d_i for all members $s_i \in S^{(n+1)}$ are calculated. Note that the algorithm has to recalculate this value also for those

previously evaluated points (in previous iterations) since the distance of all points to all other evaluated points is of interest. Update $D^{(n+1)}$. Finally, the bi-objective function F is redefined accordingly.

- Step 3e: Hypervolume Improvement Indicator (**hypervol_improv_indc**) is calculated at each of the P newly evaluated points and the P-vector $I = [I_1, ..., I_P]$ of hypervolume improvement indicator is defined.

The pseudocode for function **hypervol_improv_indc** is shown in Figure 4.4. First, it checks if the new point x_{new} is dominated by any points on the previous Pareto front $\mathcal{X}_{\text{int}}$. If this is the case, HI is set to zero. Otherwise, it computes the relative hypervolume improvement (μ) as follows. $v_{\text{int\_best}}$ is a vector of the best point in the objective space with respect to $\mathcal{P}_{\text{int}} = F[\mathcal{X}_{\text{int}}]$. $\mathcal{P}_{\text{new}}$ is the new Pareto front of $\mathcal{X}_{\text{int}} \cup \{x_{\text{new}}\}$ (the previous Pareto set plus the new point). v_{ref} is a vector of the worst point in the objective space with respect to $\mathcal{P}_{\text{new}}$. Then, the relative hypervolume improvement μ is calculated. See Eq. (4.2.3) for a definition of **HV**. Finally, if $\mu > \tau$, HI = 1, otherwise HI = 0. Recall that all objective values $F_1(s_i) = t_i$ and $F_2(s_i) = -d_i$ are pre-computed and stored in the list $T^{(n)}$ and $-D^{(n)}$, respectively. Therefore, no additional expensive function evaluations or any distance calculations are required.

Function HI = **hypervol_improv_indc**($\mathcal{X}_{\text{int}}$, $\mathcal{P}_{\text{int}}$, F_1, F_2, x_{new}, $[y_{\text{new}}, \lambda_{\text{new}}]$, τ)
  if for some $x \in \mathcal{X}_{\text{int}}$, $x \preceq x_{\text{new}}$
    HI= 0
  else
    $v_{\text{int\_best}} = (\min F_1[\mathcal{X}_{\text{int}}], \min F_2[\mathcal{X}_{\text{int}}])$
    $v_{\text{ref}} = (\max F_1[\mathcal{X}_{\text{int}} \cup \{x_{\text{new}}\}], \max F_2[\mathcal{X}_{\text{int}} \cup \{x_{\text{new}}\}])$
    $[\mathcal{X}_{\text{new}}, \mathcal{P}_{\text{new}}]$ =**pareto_front**($\mathcal{X}_{\text{int}} \cup \{x_{\text{new}}\}$, F_1, F_2)
    $\mu = (\mathbf{HV}(\mathcal{P}_{\text{new}}, v_{\text{ref}}) - \mathbf{HV}(\mathcal{P}_{\text{int}}, v_{\text{ref}})) / \prod_{i=1}^{2} \left(v_{\text{ref}}(i) - v_{\text{int\_best}}(i)\right)$
    HI = $1_{\mu>\tau}$
  end if

Figure 4.4: Pseudocode **hypervol_improv_indc** used in Step **3e**

Figure 4.5 shows an example of the hypervolume improvement. First, the Pareto front is $\mathcal{P}_{\text{int}} = \{\text{P1, P2, P3}\}$. The hypervolume corresponding to $\mathcal{P}_{\text{int}}$ is the gray shade. Then, after

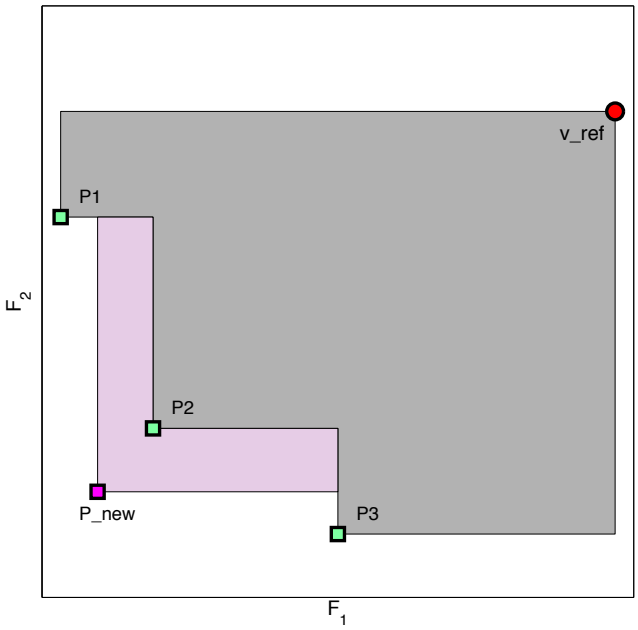


Figure 4.5: Hypervolume improvement

adding the new point P_new, the new Pareto front becomes $\mathcal{P}_{\text{new}} = \{\text{P1, P\_new, P3}\}$. We see that the hypervolume is bigger. The light magenta area is the improvement made after the new point is added.

Definition 4.3. We say that the **local improvement is registered** (and write HI = 1) if the relative hypervolume improvement (the difference of the hypervolume calculated after and before adding a newly evaluated point x_{new}) is larger than some tolerance τ. Otherwise, we say that the local improvement is not registered (and write HI = 0).

- Step 3f: the algorithm updates the list $R^{(n+1)}$. Also, tabu_struct(1) and tabu_struct(2) attach P copies of zeros to the end of the list (these zeros correspond to the tabu_struct of P newly evaluated points). Finally, the optimization iteration counter n is updated.

Definition 4.4. We define as **tabu** any center whose local improvement is not registered for N_{fail} search efforts around that center. We will call such center a **tabu center**. The tabu center will not be used as a center for m iterations, i.e. the **tenure length** is m.

- Step 3g: the algorithm updates the tabu structure and search radii of centers by checking whether the local improvement is registered. The input of the function **update** are tabu_struct (tabu structure), I (a P-vector of hypervolume improvement indicator), $\mathcal{J}$ (a P-vector of selected center indices), N_{fail} (the maximum number of non-registered improvements allowed), m (tenure length), r_{int} (the initial search radius), and $R^{(n)}$ (the list of search radii). tabu_struct consists of two lists tabu_struct(1) and tabu_struct(2):

  - the i-th element in tabu_struct(1) keeps track of the number of times local improvement is not registered when using $s_i \in S^{(n)}$ as a center.
  - the i-th element in tabu_struct(2) keeps track of the number of wait time ($0 \leq$ wait_time $\leq m$) until a particular tabu point $s_i \in S^{(n)}$ can be used as a center again.

The pseudocode for function **update** is shown in Figure 4.6. In the first for-loop in the function **update**, the algorithm updates the number of non-registered improvements for the centers that failed to find a local improvement ($I_j = 0$). Also, the search radii of these centers are reduced to half.

The second for-loop in the function **update** only applies to the first $n_0+(n-1)P$ points in $S^{(n)}$ (which are the points that were evaluated in previous iterations). It reduces the counter of wait time of the tabu centers ($t_2(i) > 0$). For any center whose local improvement is not registered for N_{fail} search efforts ($t_1(i) > N_{\text{fail}}$), the corresponding center will be marked as tabu ($t_2(i) = m$) and will be skipped for the next m iterations (wait_time is m). Also, the counter of the non-registered local improvement is reset to zero ($t_1(i) = 0$). Finally, the search radii of the newly marked tabu points are reset to r_{int}.

Function $\left[\text{tabu\_struct},\, R^{(n)}\right] = \textbf{update}\left(\text{tabu\_struct},\, I,\, \mathcal{J},\, N_{\text{fail}},\, m,\, r_{\text{int}},\, R^{(n)},\, P\right)$

```
    t_1 = tabu_struct(1)
    t_2 = tabu_struct(2)
    for j = 1 : P
        if I_j = 0
            t_1(J(j)) = t_1(J(j)) + 1
            R^(n)(J(j)) = R^(n)(J(j))/2
        end if
    end for
    for i = 1 : n_0 + (n-1)P
        if t_2(i) > 0
            t_2(i) = t_2(i) - 1
        else if t_1(i) > N_fail
            t_2(i) = m
            t_1(i) = 0
            R^(n)(i) = r_int
        end if
    end for
```

Figure 4.6: Pseudocode **update** used in Step **3g**

4.3.1 Non-dominated Sorting (NDS) and P Center Selection

This section gives a description of sub-functions **non_dom_sorting** and **P_centers_sel** that are called within the function **select_P_indices** in Step 3c (Figure 4.3) used in Algorithm 4.2. Unlike [96] where only the best point found so far is used as a center of perturbation, SOP selects P points from the set of all evaluated points to use as centers by using these two sub-functions.

To manage the trade-off between exploration and exploitation, in addition to the underlying objective function value (F_1), an additional second objective, namely, the minimum distance to the set of already evaluated points, is considered. High distance indicates regions where the accuracy of the current solution can possibly be improved with additional points in that region. Thus, to explore previously unexplored regions, the algorithm maximizes this distance. Maximizing the minimum distance from points already evaluated is the same as

minimizing its negative. Thus, F_2 is defined as the negative of the minimum distance to the set of already evaluated points.

Mathematically, the following bi-objective optimization over a finite set $S^{(n)}$ is considered for center selection:

$$\min_{x \in S^{(n)}} [F_1(x),\, F_2(x)], \tag{4.3.2}$$

where $F(x) = [F_1(x),\, F_2(x)] \in \mathbb{R}^2$ is a vector of the conflicting objective functions that we want to minimize simultaneously. These conflicting objective in the nth iteration are to find x that 1) minimizes the expensive function value $f(x)$, and 2) minimizes the negative of the minimum distance between x and other previously evaluated points in $S^{(n)}$. For simplicity, let $N = n_0 + nP$ be the number of points in $S^{(n)}$. $F : S^{(n)} \to T^{(n)} \times \left(-D^{(n)}\right)$, where $S^{(n)} = \{s_1, ..., s_N\}$ is the set of all evaluated points so far, $T^{(n)} = \{t_1, ..., t_N\}$, where $t_i = f(s_i)$, and $-D^{(n)} = -\{d_1, ..., d_N\}$ is the image of the negative of the minimum distance of point $s_i \in S^{(n)}$ from all other evaluated points in $S^{(n)}$, i.e. $d_i = \min\left\{\|s_i - s\| : s \in S^{(n)} \setminus \{s_i\}\right\}$.

The pseudocode of two sub-functions **non_dom_sorting** and **P_centers_sel** is represented in Figures 4.7 and 4.8, respectively.

Function $\left[S^{\textbf{order}}, R^{\textbf{order}}, \textbf{tabu\_struct}^{\textbf{order}}\right] = ...$
non_dom_sorting$(S^{(n)}, T^{(n)}, D^{(n)}, R^{(n)}, \text{tabu\_struct})$

1. Sort all the evaluated points in the set $S^{(n)}$ into levels of non-dominated fronts using the procedure in Section 4.2.2.2 and Eq. (4.2.2) according to the two following (pre-computed) objectives:
   (a) the objective function value (F_1) in $T^{(n)}$;
   (b) the negative of the minimum distance from all other evaluated points (F_2) in $-D^{(n)}$.
2. Let $x^{\text{Front}(i,j)}$ be the jth point of the ith front that is ordered such that for a fixed front i, $x^{\text{Front}(i,j)}$ is in an increasing order of objective function values, i.e. $f\left(x^{\text{Front}(i,1)}\right) \leq f\left(x^{\text{Front}(i,2)}\right) \leq \cdots$.
3. Denote by $S^{\text{order}} = \left\{x^{\text{Front}(1,1)}, x^{\text{Front}(1,2)}, \cdots x^{\text{Front}(2,1)}, x^{\text{Front}(2,2)}, ...\right\}$ the list of points obtained from this procedure starting from Front 1. Also, let R^{order} and tabu_struct$^{\text{order}}$ be the sorted $R^{(n)}$ and sorted tabu_struct based on this order.

Figure 4.7: Pseudocode **non_dom_sorting** used in **select_P_indices** (Figure 4.3)

The purpose of Non-dominated Sorting (NDS) is to maintain the balance between exploitation (local search) and exploration (global search). By NDS, those points whose function values are small (exploitation) and/or are far away from other evaluated points (exploration) will be more likely to be selected as a center.

After the function **non_dom_sorting** sorts points in $S^{(n)}$ into levels of non-dominated fronts, the function **P_centers_sel** (Figure 4.8) iteratively selects a center in order (based on Radius Rule and Tabu Rule) from the list S^{order} starting from the first point in the list.

Assume that $c_1, ..., c_{i-1}$ $(i-1 < P)$ have been selected as centers. In **P_centers_sel**, the next point s_{o_k} on the front will be selected as the ith center if it is (1) at least ϵ_j distance away from all the previous selected center c_j, $j = 1, ..., i-1$ (Radius Rule) and also (2) not marked as tabu, i.e. $t_{o_k} = 0$ (Tabu Rule). These two conditions appear in **Do [Until...]** loop of the function **P_centers_sel** in Figure 4.8. Note that only the second list of tabu_struct$^{\text{order}}$ (i.e. tabu_struct$^{\text{order}}(2)$), which stores the wait time for tabu points, is actually used in **P_centers_sel**.

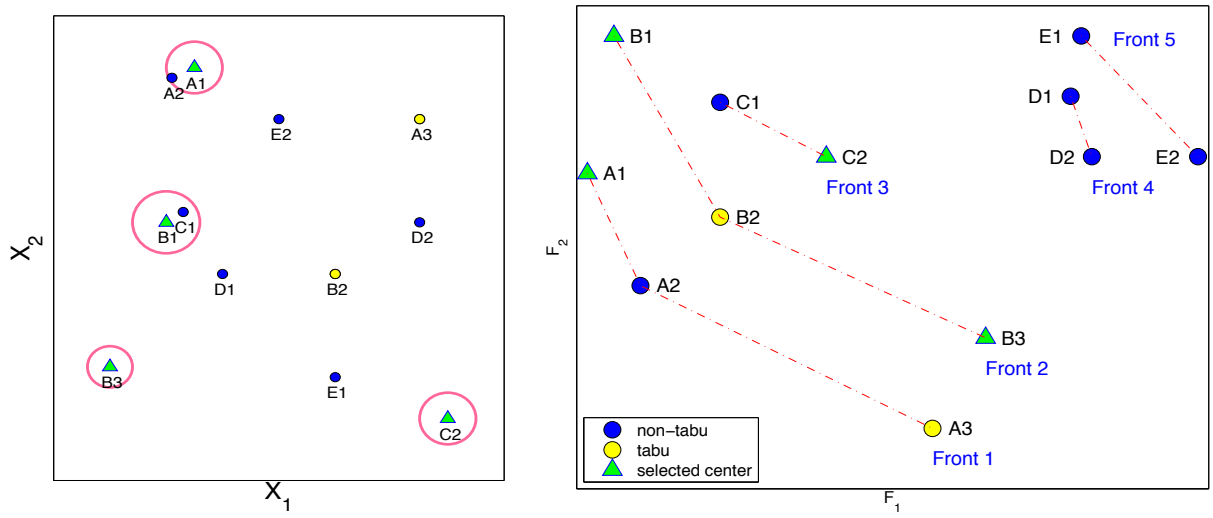
Function $[c_1, ..., c_P] =$ **P_centers_sel**$\left(S^{\text{order}}, R^{\text{order}}, \text{tabu\_struct}^{\text{order}}, P\right)$
Let N be the number of points in S^{order}.
Let $S^{\text{order}} = \{s_{o_1}, s_{o_2}, ..., s_{o_N}\}$, $R^{\text{order}} = \{r_{o_1}, ... r_{o_N}\}$, and
tabu_struct$^{\text{order}}(2) = \{t_{o_1}, ..., t_{o_N}\}$.
Set $c_1 = s_{o_1}$, $\epsilon_1 = r_{o_1}$, and $k = 1$.
for $i = 2 : P$
$k = k + 1$
Do [Until $(\|c_j - s_{o_k}\| > \epsilon_j \, \forall j = 1, \ldots, i-1)$ and $(t_{o_k} = 0)$**]**
$k = k + 1$
[Exit Do]
Set $c_i = s_{o_k}$, $\epsilon_i = r_{o_k}$
end for

Figure 4.8: Pseudocode **P_centers_sel** used in **select_P_indices** (Figure 4.3)

The selection of the P center points is done sequentially to avoid selecting points that are close to other center points already selected in the same iteration. Note that the best point with the lowest function value (corresponding to s_{o_1}) is always chosen as a center regardless of its tabu structure. That is, the algorithm makes a locally optimal choice (greedy) as one of the P choices in the hope that it will lead to a globally optimal solution. We found, in practice, that selecting the best point as one of the centers in every iteration accelerates the convergence of the algorithm.

In case that all the points on the sorted fronts S^{order} have already been examined but the number of selected centers (i) is less than P (the number of centers needed), we reexamine the tabu points on the fronts (i.e. those with $t_{o_k} > 0$) starting from Front 1. This time with only one condition imposed: $(\|c_j - s_{o_k}\| > \epsilon_j \, \forall j = 1, \ldots, i-1)$. Finally, if the number of centers selected is still less than P, the next $P - i$ centers, c_{i+h}, $h = 1, .., P - i$, will be selected iteratively by cycling through the set of already selected centers $\{c_1, ..., c_i\}$, i.e. $c_{qi+l} = c_l$ for $q \geq 1$ and $1 \leq l \leq i$, until all the P centers are selected. This is because good solutions often lie in a very small neighborhood. Therefore, it is also necessary to focus the local candidate search around those promising centers.

Figure 4.9 illustrates an example of the center selection via **P_centers_sel**. After the points in $S^{(n)}$ are sorted into the levels of non-dominated fronts (Figure 4.9b), in this



(a) Decision space (b) Objective space

Figure 4.9: Example of center selection in (a) decision space and (b) objective space

example, four center points ($P = 4$) are selected sequentially along the sorted fronts. First, the point A1 on the first front is selected. Since A2 is within a radius (defined in Figure 4.8) of A1, and A3 is marked as tabu (defined in Definition 4.4), neither of these two points are selected. We move on to the second front and the next point that will be selected is therefore B1. Continue the process, the four points that pass both the Radius Rule and the Tabu Rule are A1, B1, B3, and C2.

4.3.2 Example of Tabu Structure

In this section we provide a simple example to illustrate how tabu_struct is updated using functions **P_centers_sel** (Figure 4.8) and **update** (Figure 4.6). Let $x \in S^{\text{order}}$ be the point on the front that we consider. If x is selected as a center, the new point suggested by the center x (for a function evaluation) is denoted by x_{select}.

Iterations 1 through 11 are shown in Table 4.1. The first column is the number of iterations, the second column will be marked as "Y" if the point x passes the Radius Rule,

and "N" otherwise. The third column indicates whether the point x that has passed the Radius Rule will be used as a center by checking whether or not that point is marked as tabu. This can be done by checking the last column wait_time of the previous row. The wait_time column of the previous row records the number of iterations that the tabu point has to wait before it can be used as a center again. 0 means the center is currently not marked as tabu, and so it can be used as a center if it is selected. Otherwise if this number is greater than 0, then the point will not be used as a center (even though the point passed the Radius Rule). The fourth column records the number of failed improvement trials so far. Once this number exceeds N_{fail}, the maximum number of non-registered local improvements allowed, that center will be marked as tabu, resetting this number to 0 and marking the column wait_time as m. Once the center has been marked as tabu, that center has to wait for m more iterations (the number in this column for a tabu point decreases by 1 each iteration) before it can be used as a center again. Note that the column #failed improvement trials corresponds to tabu_struct(1) while the column wait_time corresponds to tabu_struct(2) in Algorithm 4.2.

In this particular example, let $N_{\text{fail}} = 3$ be the maximum number of non-registered local improvement allowed before that particular point is marked as tabu, and $m = 5$ the tenure length. In iteration 1, the point x passed the Radius Rule ("Y" in the second column). To check whether this point can be used as a center or not, we look at the column wait_time of the previous row (i.e. row iteration 0). Since the wait_time in the previous row is 0, the point x is selected. After a new point x_{select}, suggested by this center, has been evaluated, the column #failed improvement trials will be increased by 1 if there is no improvement. In iteration 3, the point x did not pass the Radius Rule; hence, it cannot be used as a center and the third column is marked as N (without having to check whether it is tabu or not). In both iterations 4 and 5, the point x does pass the Radius Rule and is selected as a center. In iteration 5, since there is again no improvement, #failed improvement trials is now increased to 4. Since it exceeds the maximum number of non-registered local improvements allowed

Table 4.1: Example of tabu structure at a particular center $x \in S^{\text{order}}$. $N_{\text{fail}} = 3$ and $m = 5$ are used in this example.

Iter#	Pass Radius Rule?	Selected (Y/N)		#failed improvement trials	wait_time
0	-	-		-	0
1	Y	Y	$\rightarrow$	1	0
2	Y	Y		2	0
3	N	N	after	2	0
4	Y	Y	evaluating	3	0
5	Y	Y	x_{select}	4→0	5
6	Y	N	suggested	0	4
7	N	N	by	0	3
8	Y	N	center x	0	2
9	N	N		0	1
10	Y	N		0	0
11	Y	Y		0	0

($N_{\text{fail}} = 3$), that center is now marked as tabu. The #failed improvement trials is reset to 0, while the column wait_time is updated to 5 (which means the point x has been marked as tabu for the next $m = 5$ iterations). In iteration 6, the point x passes the Radius Rule, but since column wait_time in iteration 5 is marked as 5 (x is tabu for another 5 iterations), this point will not be selected as a center. As the algorithm proceeds, the number in column wait_time decreases by 1 in each iteration. In iteration 11, the point x passed the Radius Rule and also the last column of the previous row (row iteration 10) is 0. Therefore, the point x can now be used as a center point again. This time, there is an improvement after a point x_{select} suggested by this center has been evaluated. Hence, the #failed improvement trials in the last row is still 0.

4.3.3 Local Candidate Search

The pseudocode of the function **local_candidate_search** in Step 3d is given in Figure 4.10. For convenience, we will suppress the subscript i (of c_i, r_i and $x_i^{(n+1)}$). In **local_candidate_search**, a center point c is perturbed to generate N_{cand} candidate points $\{v_1, ..., v_{N_{\text{cand}}}\}$. As in [97], we randomly select the variables of c that will be perturbed and

generate the candidate points by adding a random perturbation. Finally, the candidate point whose predicted objective function value is smallest is selected for doing the next computationally expensive function evaluation. Here, rand(1) denotes a uniformly selected real value in [0, 1] and randi(d) a random integer between 1 and d. Recall that $\mathcal{D} = \{x : \text{lb} \leq x \leq \text{ub}\}$.

Function $x =$ **local_candidate_search**(c, r, $\mathcal{D}$, N_{cand}, $g(\cdot)$, p_{select})
  for $j = 1 : N_{\text{cand}}$
    for $i = 1 : d$, $u_i = \text{rand}(1)$, **end for**
    let $I_{\text{perturb}} = \{i : u_i < p_{\text{select}}\}$
    if $I_{\text{perturb}} = \emptyset$, $I_{\text{perturb}} = \text{randi}(d)$, **end if**
    generate $v_j = \mathcal{L}(c, r, \mathcal{D}, I_{\text{perturb}})$
  end for
  $x = \arg\min_{1 \leq j \leq N_{\text{cand}}} g(v_j)$

Figure 4.10: Pseudocode **local_candidate_search** used in Step 3d

The local search mechanisms similar to **local_candidate_search** have been shown to be very effective, e.g. [95, 96, 97]. The Gaussian distribution has been used to generate candidate points in these previous algorithms. If a generated candidate point does not fall inside the domain $\mathcal{D}$, the point will be either replaced by the nearest point in $\mathcal{D}$ [95, 96] or successive reflected about the closest point on the boundary of $\mathcal{D}$. In this work, we investigate two other generator types $\mathcal{L}$ within SOP as described below.

4.3.3.1 Type 1 ($\mathcal{L} = N_{\text{truncated}}$): truncated normal candidate points (nSOP)

For nSOP, we generate the jth candidate point v_j as follows:

- $v_j = c + Z$, where $Z(i) \sim N_{\text{truncated}}(0, \sigma^2; a(i), b(i))$, $\sigma = r$, $a(i) = \text{lb}(i) - c(i)$ and $b(i) = \text{ub}(i) - c(i)$ for $i \in I_{\text{perturb}}$, and $Z(i)$ is zero for $i \notin I_{\text{perturb}}$. See Section 3.2.2 for a description of the truncated normal distribution, $N_{\text{truncated}}$.

4.3.3.2 Type 2 ($\mathcal{L} = \mathcal{U}$): uniform candidate points (uSOP)

Let $\mathcal{U}[u_1, u_2]$ denote a uniform random variable on $[u_1, u_2]$. For uSOP, we uniformly select candidate points from the interval $[a(i), b(i)]$ defined as follows:

- $v_j(i) = \mathcal{U}[a(i), b(i)]$ where $a(i) = \max(\text{lb}(i), c(i) - r)$, $b(i) = \min(c(i) + r, \text{ub}(i))$ for $i \in I_{\text{purturb}}$, and $v_j(i) = c(i)$ for $i \notin I_{\text{purturb}}$.

Because both the truncated normal candidate points and the uniform candidate points are generated within the domain, neither the reflection nor the replacement of the candidate point is necessary in our case.

4.4 Convergence of nSOP

In this section, we will give some conditions which ensure the global convergence in a probabilistic sense (with probability 1) of nSOP. First, note the main differences between SOP, StochRBF [95] and ParStochRBF [96].

	StochRBF	ParStochRBF	SOP
# of simulated points per iteration	1	P	P
perturbation center(s)	x_{best}	x_{best}	$c_1, ..., c_P$
# of candidate point batches per iteration	1	1	P

Since ParStochRBF is essentially the same as StochRBF except for the number of points selected for evaluation in each iteration, one can easily apply Theorem 1 in [95] and prove the convergence for ParStochRBF. On the other hand, each center of SOP does its own local candidate search (Step 3d of Algorithm 4.2), which results in P batches of candidate points per iteration. Moreover, only some of the coordinates of the centers in SOP are selected for perturbation, whereas all coordinates of the center x_{best} in [95] are perturbed for generating candidate points. In order to prove the convergence of SOP, we need the following definitions.

Definition 4.5. For $1 \le j \le N_{\text{cand}}$, $n \ge 1$, and $1 \le i \le P$,

$V_{i,j}^{(n-1)}$ is the random vector representing the jth random candidate point $v_{i,j}^{(n-1)}$ generated around center c_i in function **local_candidate_search** of Step 3d (Figure 4.10).

$X_i^{(n)}$ is the random vector representing $x_i^{(n)}$, the ith function evaluation point of iteration n (chosen from $\left\{V_{i,j}^{(n-1)} : j = 1, ..., N_{\text{cand}}\right\}$)

Fix $n \geq 0$ (optimization iteration counter) and $1 \leq i \leq P$. Let $c_i^{(n)} \in S^{(n)}$ be the ith selected center for perturbation in iteration n (Algorithm 4.2, Step 3c). Recall that in function **local_candidate_search** of Step 3d (Figure 4.10), N_{cand} candidate points $\left\{v_{i,j}^{(n)} : j = 1, ..., N_{\text{cand}}\right\}$ are generated by perturbing randomly selected coordinates of $c_i^{(n)}$. Each coordinate of $c_i^{(n)}$ has a probability $p_{\text{select}} = \varphi(n) \in [0, 1]$ to be perturbed. If no variable of $c_i^{(n)}$ is selected for perturbation, one variable is chosen at random.

We next define the σ-algebra generated by relevant information available up to iteration n.

Definition 4.6. Let $n \geq 1$ and $1 \leq i \leq P$. For $1 \leq j \leq N_{\text{cand}}$, let $Q_{i,j}^{(n-1)}$ be the random vector that determines which coordinates of $c_i^{(n-1)} \in S^{(n-1)}$ are chosen for a perturbation to obtain $V_{i,j}^{(n-1)}$.

For each $n \geq 1$, define $\mathcal{F}_{n-1}$, $\mathcal{Q}_{n-1} \subset \mathbb{R}^d$ by

- $\mathcal{F}_{n-1} := \left\{ \begin{array}{cccc} & V_{1,1}^{(0)}, ..., V_{1,N_{\text{cand}}}^{(0)}, & & V_{1,1}^{(n-1)}, ..., V_{1,N_{\text{cand}}}^{(n-1)}, \\ \underbrace{X_1^{(0)}, .., X_{n_0}^{(0)}}_{\text{initial pts}}, & \vdots & \cdots & \vdots \\ & \underbrace{V_{P,1}^{(0)}, ..., V_{P,N_{\text{cand}}}^{(0)},}_{\text{cand pts iter 0}} & & \underbrace{V_{P,1}^{(n-1)}, ..., V_{P,N_{\text{cand}}}^{(n-1)}}_{\text{cand pts iter } n-1} \end{array} \right\} \begin{array}{c} \leftarrow \text{center1} \\ \vdots \\ \leftarrow \text{center}P \end{array}$

- $\mathcal{Q}_{n-1} := \left\{ \begin{array}{cccc} Q_{1,1}^{(0)}, ..., Q_{1,N_{\text{cand}}}^{(0)}, & & Q_{1,1}^{(n-1)}, ..., Q_{1,N_{\text{cand}}}^{(n-1)}, \\ \vdots & & \vdots \\ \underbrace{Q_{P,1}^{(0)}, ..., Q_{P,N_{\text{cand}}}^{(0)},}_{\text{iteration 0}} & \cdots & \underbrace{Q_{P,1}^{(n-1)}, ..., Q_{P,N_{\text{cand}}}^{(n-1)}}_{\text{iteration } n-1} \end{array} \right\} \begin{array}{c} \leftarrow \text{center1} \\ \vdots \\ \leftarrow \text{center}P \end{array}.$

Then, define

- $\mathcal{E}_{n-1} = \mathcal{F}_{n-1} \cup \mathcal{Q}_{n-1}$ for $n \geq 1$, and $\mathcal{E}_{-1} = \left\{ X_1^{(0)}, .., X_{n_0}^{(0)} \right\}$.

Then, $\mathcal{F}_{n-1}$ contains the initial data used to fit the initial surrogate and all the candidate points $V_{i,j}^{(n-1)}$ generated within the function **local_candidate_search** of Step 3d up to iteration $n-1$. $\mathcal{Q}_{n-1}$ contains all $Q_{i,j}^{(n-1)}$, the random vector that determines which coordinates of selected centers are chosen for a perturbation, up to iteration $n-1$.

Remark 4.7. For $n \geq 1$, since the new ith evaluation point $X_i^{(n)}$, $1 \leq i \leq P$, is selected deterministically from the values of the random vectors $\left\{ \left(V_{i,1}^{(n-1)} \right), ..., \left(V_{i,N_{\text{cand}}}^{(n-1)} \right) \right\}$ through the **local_candidate_search**, the entire path of algorithm up to optimization iteration n is completely determined by $\sigma(\mathcal{E}_{n-1})$, where $\sigma(\mathcal{E}_{n-1})$ is the $\sigma - algebra$ generated by the random vectors in $\mathcal{E}_{n-1}$.

Theorem 4.8 is an extension theorem of Theorem 1 in [95] (see Theorem 3.6, Chapter 3 for the statement of this theorem). Unlike [95] whose sequence $\{X_n^*\}_{n\geq 1}$ is defined for every function evaluation point (since the number of function evaluations = number of optimization iteration in [95]), $\left\{ X^{*(n)} \right\}_{n\geq 0}$ in our case is defined for n that is the number of optimization iterations.

Theorem 4.8. *Let f be a function defined on $\mathcal{D} \subseteq \mathbb{R}^d$ and suppose that $x^* = \min_{x\in\mathcal{D}} f(x) > -\infty$ is the unique global minimizer of f in $\mathcal{D}$ such that $\min_{x\in\mathcal{D}, \|x-x^*\|\geq\eta} f(x) > f(x^*)$ for all $\eta > 0$. Suppose further that the algorithm generates the random vectors $\left\{ X_i^{(0)} \right\}_{1\leq i\leq n_0}$, $\left\{ X_i^{(n)} : n \geq 1, 1 \leq i \leq P \right\}$ and $\left\{ V_{i,1}^{(n)}, ..., V_{i,N_{cand}}^{(n)} : n \geq 0, 1 \leq i \leq P \right\}$ satisfying the following two conditions:*

[E1] For each $n \geq 0$, the random vectors $\left\{ V_{i,j}^{(n)} : 1 \leq i \leq P, 1 \leq j \leq N_{\text{cand}} \right\}$ are jointly independent conditional on $\mathcal{E}_{n-1}$.

[E2] For any $j = 1, ..., N_{\text{cand}}$, $x \in \mathcal{D}$ and $\delta > 0$, there exists $\nu_j(x, \delta) > 0$ such that

$$P[V_{i,j}^{(n)} \in B(x, \delta) \cap \mathcal{D} | \sigma(\mathcal{E}_{n-1})] \geq \nu_j(x, \delta)$$

for all $n \geq 0$, $1 \leq i \leq P$, where $B(x, \delta)$ is the open ball of radius δ centered at x.

If the sequence of random vectors $\{X^{*(n)}\}_{n\geq 0}$ is defined by $X^{*(0)} = \text{argmin}_{x\in\{X_1^{(0)},\ldots,X_{n_0}^{(0)}\}} f(x)$ and

$$X^{*(n)} = \begin{cases} \text{argmin}_{x\in\{X_1^{(n)},\ldots,X_P^{(n)}\}} f(x) & \text{if } \min\left\{f(X_1^{(n)}),\ldots,f(X_P^{(n)})\right\} < f(X^{*(n-1)}) \\ X^{*(n-1)} & \text{otherwise} \end{cases}$$

for $n \geq 1$, then $X^{*(n)} \xrightarrow{a.s.} x^*$

Proof. In order to avoid the tedious arguments, we will use Theorem 1 in [95] to prove our theorem. First, replace $\mathcal{E}_n$ defined in [95] with the one defined in Definition 4.6 above. We define random vectors $\{X_k\}_{k\geq 1}$ and $\{Y_{k,1},\ldots,Y_{k,t}\}_{k\geq n_0}$ that satisfy the two conditions **[C1]**–**[C2]** of Theorem 1 in [95] as follow:

$$X_k = \begin{cases} X_k^{(0)} & \text{for } 1 \leq k \leq n_0 \\ \text{argmin}_{x\in\{X_1^{(k-n_0)},\ldots,X_P^{(k-n_0)}\}} f(x) & \text{for } k \geq n_0+1 \end{cases}$$

Let $t = N_{\text{cand}}P$. Then, $\{Y_{k,1},\ldots,Y_{k,t}\}_{k\geq n_0}$ is defined as $\left\{V_{i,j}^{(k-n_0)} : i = 1,\ldots,P,\ j = 1,\ldots,N_{\text{cand}}\right\}_{k\geq n_0}$. That is, all the candidate points generated from each of the P centers in the same optimization iteration are combined into one set. Since the two conditions **[E1]**–**[E2]** in this theorem are assumed, it follows immediately that the two conditions **[C1]**–**[C2]** in [95] also hold. Finally, we define $X_1^* = X_1$ and $X_k^* = X_k$ if $f(X_k) < f(X_{k-1}^*)$ and $X_k^* = X_{k-1}^*$ otherwise. Then, by Theorem 1 of [95], we can conclude that $X_k^* \xrightarrow{a.s.} x^*$. Since $\left(X^{*(k)}\right)$, (X_k^*) have the same tail, they converge to the same limit. Therefore, $X^{*(n)} \xrightarrow{a.s.} x^*$. □

We now show that conditions **[E1]** and **[E2]** of Theorem 4.8 hold for nSOP. The condition **[E1]** is trivial. Condition **[E2]** requires that the algorithm is able to sample at any point of the variable domain.

Lemma 4.9. *[nSOP] For a fixed $j \in \{1, \ldots N_{cand}\}$, let H be the event that all coordinates of the center $c_i^{(n)}$ are selected for perturbation with search radius $r_i^{(n)} > 0$ (and $v_{i,j}^{(n)}$ is generated). Let $h_{i,j}^{(n)}$ be the conditional density of $V_{i,j}^{(n)}$ in nSOP given $\sigma(\mathcal{E}_{n-1})$ and H. Then, there is a constant $C > 0$ such that $h_{i,j}^{(n)}(u) \geq C$ for all $u \in \mathcal{D} = \{x : lb \leq x \leq ub\}$, $1 \leq i \leq P$ and $n \geq 0$.*

Proof. Assume that all coordinates of $c_i^{(n)}$ are selected for perturbation and that all the information $\sigma(\mathcal{E}_{n-1})$ is known. Since candidate points of nSOP follow multivariate truncated normal distributions, given that all coordinates of $c_i^{(n)}$ are selected for perturbation, the conditional density $h_{i,j}^{(n)}$ can be written as: $h_{i,j}^{(n)}(u) = A \exp\left\{\frac{-\left\|u - c_i^{(n)}\right\|^2}{2\sigma^2}\right\}$ for $u \in \mathcal{D}$ and 0 otherwise, where $\sigma = r_i^{(n)}$ and $A > 0$ is a normalizing constant for truncated multivariate normal density. Recall that in function **update** of Step 3g, each time when the local improvement of a particular center is not registered, the search radius of that center is shrunk by half. Moreover, the center whose local improvement is not registered for N_{fail} search efforts will be marked as tabu and the search radius of that center is reset back to r_{int}. Therefore, $r_i^{(n)} \geq r_{\text{int}}/2^{N_{\text{fail}}} := B > 0$. Taking $C := A \exp\left\{\frac{-\|\text{ub}-\text{lb}\|^2}{2B^2}\right\}$. It follows that $h_{i,j}^{(n)}(u) \geq C > 0$ for all $u \in \mathcal{D}$, $1 \leq i \leq P$ and $n \geq 0$. □

Lemma 4.10. *If $\inf_{n \geq 0} \varphi(n) > 0$, the condition [E2] holds for nSOP.*

Proof. Recall that $\varphi(n)$ is used to compute p_{select}, the probability for selecting a variable of the current center $c_i^{(n)}$ for perturbation. Assume that $\inf_{n \geq 0} \varphi(n) > 0$. Let $j \in \{1, \ldots N_{\text{cand}}\}$, $x \in \mathcal{D}$ and $\delta > 0$ be given. Continuing with the notation used in Lemma 4.9, in particular recall the definition of H,

$$P[V_{i,j}^{(n)} \in B(x, \delta) \cap \mathcal{D} | \sigma(\mathcal{E}_{n-1})]$$

$$\geq P[\left(V_{i,j}^{(n)} \in B(x, \delta) \cap \mathcal{D}\right) \cap H | \sigma(\mathcal{E}_{n-1})]$$

$$= P[V_{i,j}^{(n)} \in B(x, \delta) \cap \mathcal{D} | \sigma(\mathcal{E}_{n-1}), H] \times P(H)$$

$$= \left(\int_{B(x,\,\delta)\cap\mathcal{D}} h_{i,j}^{(n)}(u)du \right) \times \varphi(n)^d$$

where $h_{i,j}^{(n)}$ is defined in Lemma 4.9.

$$\geq C\mu\left(B(x,\,\delta)\cap\mathcal{D}\right) \times \left(\inf_{n\geq 0} \varphi(n) \right)^d := \nu_j(x,\,\delta)$$

for any $n \geq 0$, where $C > 0$ is a non-negative constant existing in Lemma 4.9. Due to the compactness of hypercube $\mathcal{D}$ and our assumption on φ that $\inf_{n\geq 0} \varphi(n) > 0$, it follows that $\nu_j(x,\,\delta) > 0$. Note also that $\nu_j(x,\,\delta)$ is independent of n and i. Thus, the condition **[E2]** is now verified. □

Since the conditions of Theorem 4.8 hold for nSOP, we can conclude that nSOP converges to the global minimum with probability 1. Note, however, that since uSOP does not satisfy the condition **[E2]**, we cannot apply Theorem 4.8 directly. Therefore, another approach is necessary but outside our scope of this work.

4.5 Numerical Experiments

4.5.1 Alternative Optimization Algorithms

We compare our algorithm to Parallel Stochastic RBF [96] and an evolutionary algorithm that uses radial basis function approximation (ESGRBF) [107]. Both of these algorithms were shown to be very efficient for computationally expensive functions. In terms of applications, Parallel Stochastic RBF was used to solve an optimization problem arising in groundwater bioremediation introduced in [75] and ESGRBF was used to calibrate computationally expensive watershed models.

4.5.1.1 Parallel StochRBF

Parallel Stochastic RBF [96] is a parallel version of Stochastic RBF by Regis and Shoemaker [95]. In each iteration, the algorithm generates candidate points around x_{best} and the best P points are selected. The selection is done sequentially, based on the weighted score of (1) the surrogate value, and (2) the minimum distance from previously evaluated points and previously selected points within that parallel iteration. The weights for these two criteria are adjusted in a cycling manner according to a predefined weight pattern.

4.5.1.2 ESGRBF

An evolutionary algorithm that uses radial basis function approximations ESRBF within a standard evolutionary strategy (ES) was proposed in [91]. ESRBF uses multiple local RBF models in each generation of an evolutionary algorithm, one RBF model for each offspring. Based on self-adaptive algorithm parameters, in each generation, $\lambda \geq \nu$ offspring are generated and their objective function values are estimated using the RBF surrogate model. The true objective function is evaluated at the ν best individuals. The μ parents for the next generation are selected from these ν best offspring. In ESRBF, to estimate the objective function value of any offspring x, first the k nearest neighbors of x among all previously evaluated parameter vectors are located. Then these nearest neighbors are used to build a local RBF approximation that is then used to estimate $f(x)$. In contrast, ESGRBF [107] uses a single RBF model to estimate the fitness values of the offspring in each generation. Shoemaker et al. [107] found that the performances of ESRBF and ESGRBF are almost identical because the global RBF approximation in ESGRBF typically agrees with the individual local RBF approximations (for each offspring). However, as mentioned in [107] that ESGRBF is much easier to implement and it involves less overhead than ESRBF, therefore we also use ESGRBF in this study.

Table 4.2: Parameter values for SOP

Parameter	Value
N_{cand}	$\min(500d,\ 5000)$
N_{fail}	3
m	5
r_{int} (nSOP)	$0.2 \times l\left(\mathcal{D}\right)$
r_{int} (uSOP)	$0.1 \times l\left(\mathcal{D}\right)$
τ	10^{-5}

4.5.2 Experimental Setup

All algorithms are run with $P = 8$ and $P = 32$ function evaluations per iteration. We use the notation $\mathcal{A}$-8P and $\mathcal{A}$-32P to distinguish the algorithm $\mathcal{A}$ that does 8 function evaluations from the one that does 32 function evaluations per iteration. For example, Parallel StochRBF algorithm simulating 8 function evaluations per iteration is denoted by StochRBF-8P.

We use a cubic RBF interpolation model [89] for all three examined algorithms. We use Latin hypercube sampling [129] for generating the initial evaluation points. The size of the initial experimental designs was set to $n_0 = \min\left\{n :\ n \geq 2(d+1) \wedge P|n\right\}$, which is the smallest integer larger than $2(d+1)$ and divisible by P. The number $2(d+1)$ has previously been used and shown to be an efficient size for initial experimental data set (see e.g. [95, 96]).

The definition of n_0 is based on the assumption that (1) each function evaluation takes approximately the same time and (2) P parallel processors are available and the P expensive evaluations can be distributed to these P processors such as to use the available computing power most efficiently.

We do ten trials of each algorithm for each test problem. All algorithms use the same initial experimental design in order to facilitate a fair comparison. Table 4.2 summarizes the values of the algorithm parameters for SOP. $l\left(\mathcal{D}\right)$ denotes the length of the shortest side of the hyperrectangle $\mathcal{D}$. For Parallel StochRBF, all algorithm parameter values are set as recommended in [96]. For ESGRBF, the parameters are set to $\mu = 4$, $\lambda = 20$, and $\nu = 8$ for $P = 8$ and $\mu = 14$, $\lambda = 80$, and $\nu = 32$ for $P = 32$.

Table 4.3: Benchmark functions for SOP

No.	Test Function
F15	Rastrigin Function
F16	Weierstrass Function
F17	Schaffers Function
F18	Schaffers Function, moderately ill-conditioned
F19	Composite Griewank-Rosenbrock Function F8F2
F20	Schwefel Function
F21	Gallagher's Gaussian 101-me Peaks Function
F22	Gallagher's Gaussian 21-hi Peaks Function
F23	Katsuura Function
F24	Lunacek bi-Rastrigin Function

4.5.3 Test Functions

Ten benchmark functions F15–F24 were selected from the BBOB test suite [47]. All of them are 10 dimensional problems. These functions are multimodal, where F15–F19 are of adequate global structure, and F20–F24 of weak global structure. The functions are listed in Table 4.3. For definition and properties of these functions, refer to [47].

4.5.4 Progress Curve in Wall-clock Time

Although the test functions are computationally inexpensive, we assume that each function evaluation takes one hour computation time and that other computational overheads arising, for example, from updating the response surface are negligible. This approach enables us to efficiently compare the performance of the individual algorithms for computationally expensive application problems. Under the assumption that P function evaluations are simulated simultaneously in each unit of wall-clock time, we plot progress curve as a function of wall-clock time as was done in [96]. When the objective functions are computationally

expensive and function evaluations are done in parallel, the stopping criterion for the optimization is a given limit on the allowable wall-clock time rather than a certain number of function evaluations. The progress curve enables us to compare the performance of the different algorithms over a range of the allowable computation time.

We set the maximum wall-clock time to be 60 hours. The total number of function evaluations ($= 60P$) for $P = 8$ and $P = 32$ will then be 480 and 1920, respectively.

4.5.5 Experimental Results and Discussion

Figures 4.11, 4.12 and 4.13 show the progress curves of selected test functions. The mean of the best objective function value is plotted on the vertical axis and the wall-clock time is plotted on the horizontal axis. Figure 4.11 shows the case for doing eight expensive evaluations simultaneously and Figure 4.12 shows the results when doing 32 parallel evaluations in each iteration. For each test function, in addition to the main plot, the last 15 iterations are plotted separately in the small subfigure so that the tail of the plot before the algorithm terminates can be seen clearly.

Overall, SOP together with either normal or uniform strategies (nSOP or uSOP) leads to a very good performance, clearly outperforming the other two alternative methods both when $P = 8$ and $P = 32$. We also find that for $P = 8$, StochRBF-8P performs better than ESGRBF-8P. However, when $P = 32$, ESGRBF-32P performs better than StochRBF-32P.

Figure 4.13 shows the results of all algorithms using $P = 8$ and $P = 32$ parallel evaluations for a clearer comparison.

When doing more function evaluations per iteration, it should be expected that the algorithm improves the objective function value in less wall-clock time since more information is obtained for refining the response surface.

However, in some cases our methods with 8 points outperformed the alternative methods with 32 points. For example, in Figure 4.13, for function F15, nSOP-8P and uSOP-8P with 8 processors got better results in a shorter wall-clock time than StochRBF-32P and ESGRBF-

32P. For function F16, both ESGRBF-8P and ESGRBF-32P are worse than nSOP-8P, uSOP-8P, as well as StochRBF-8P. In addition, ESGRBF-8P even outperformed ESGRBF-32P after around 15 hours.

As for functions F21 and F22, nSOP and uSOP converge fastest and to a better final solution. Moreover, nSOP-8P and uSOP-8P again outperformed both StochRBF-32P and ESGRBF-32P by getting a lower answer in less wall-clock time for SOP with 8 processors versus the alternative methods with 32 processors. StochRBF-32P did very poorly on these two test functions and StochRBF-8P even surpassed StochRBF-32P in both functions.

Although Parallel StochRBF was shown to work well in [96], we find that our method SOP outperformed it here. While SOP sophisticatedly selects various centers for generating candidate points using the Pareto trade-off strategy, Parallel StochRBF uses x_{best} as the only center and generates only one batch of candidate points from which the next P function evaluation points are selected. We assume this is why Parallel StochRBF is not as effective when P is large.

The mean and standard deviation of the best function value obtained after the algorithms terminate (60 hours) when using 8 and 32 processors are reported in Tables 4.4 and 4.6, respectively. In addition to ParStochRBF and ESGRBF, we also report the results of two other serial surrogate-based algorithms, namely Stochastic RBF [95] and NOMADm-DACE (NOMADm with kriging surrogate) [2, 3, 8, 72] after 480 function evaluations in Table 4.4.

We also conduct a two-sample t-test to compare the results of the final solution from Table 4.4. The one-tailed test is performed at the 5% level of significance. Tables 4.5a and b show the number of test functions (out of ten) in which nSOP or uSOP is significantly better or worse than the compared algorithm (in terms of the final solution).

For example, to read the number of test functions in which nSOP-8P significantly outperforms ESGRBF-8P, one would look at the row 'better' in Table 4.5a across the columns

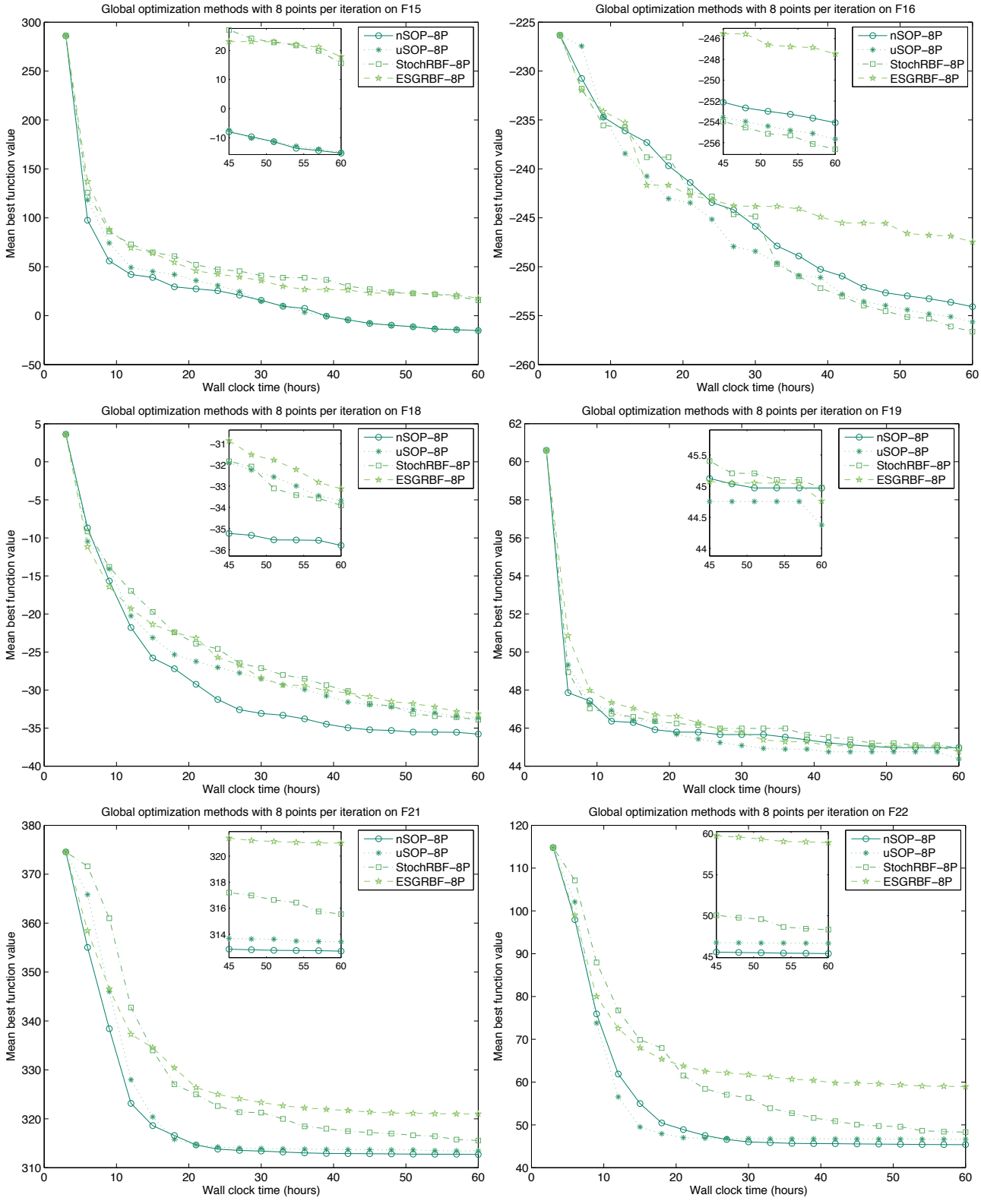


Figure 4.11: Best objective function value found averaged over ten trials versus wall-clock time. Eight points are simulated per iteration. Assume that each function evaluation takes 1 hour and 8 processors are available.

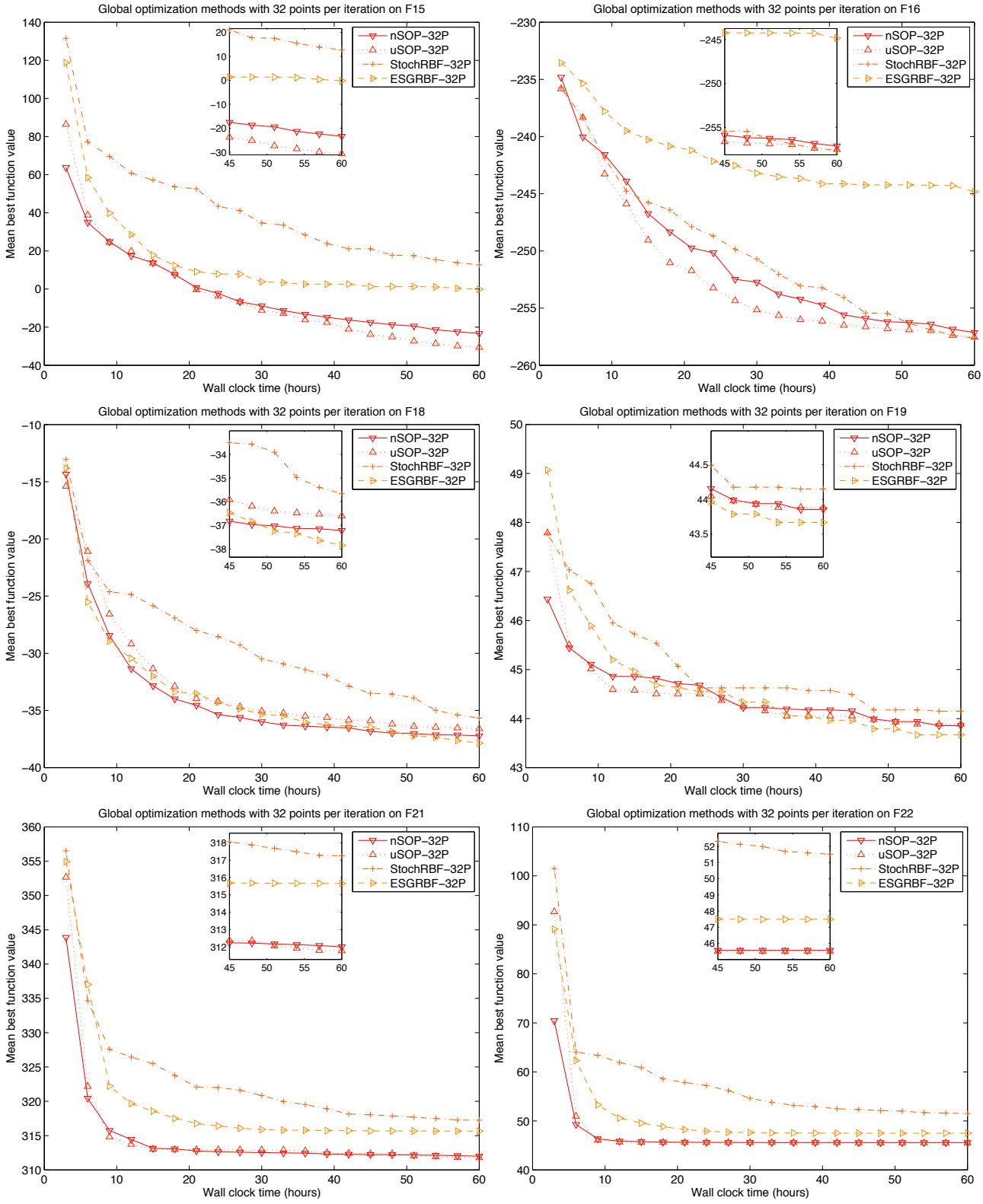


Figure 4.12: Best objective function value found averaged over ten trials versus wall-clock time. Thirty-two points are simulated per iteration. Assume that each function evaluation takes 1 hour and 32 processors are available.

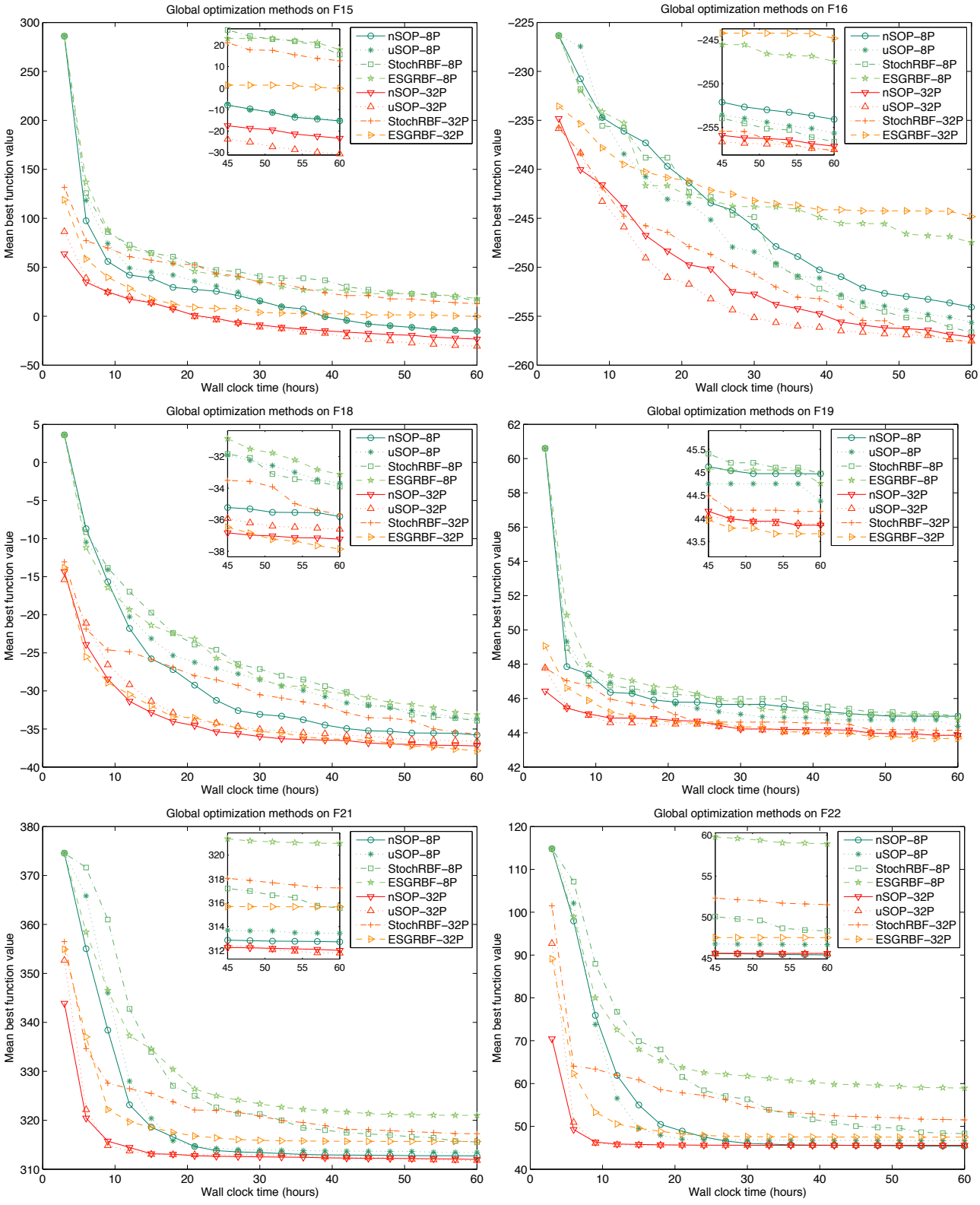


Figure 4.13: Combined results of Figures 4.11 and 4.12

Table 4.4: The results for F15–F24 from BBOB testbed using 8 processors. Mean and standard deviation of the best function value after 480 function evaluations

Alg	nSOP-8P		uSOP-8P		StochRBF-8P		ESGRBF-8P		StochRBF*		NOMADm-DACE*	
Test Func	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std
F15	**-15.276**	12.608	-14.993	13.575	15.616	10.299	17.834	7.193	-2.421	22.896	27.586	23.731
F16	-254.080	3.144	-255.652	1.768	**-256.633**	1.591	-247.473	4.957	-256.387	1.366	-253.097	3.530
F17	-37.648	0.307	-37.460	0.508	-37.616	0.608	**-37.667**	0.612	-35.371	1.936	-34.576	2.821
F18	**-35.795**	1.091	-33.681	4.181	-33.904	1.706	-33.130	2.228	-28.136	5.279	-21.989	11.048
F19	44.967	0.767	**44.375**	0.999	44.974	0.962	44.759	0.896	44.549	1.121	44.787	1.773
F20	185.634	0.248	185.447	0.279	185.828	0.407	185.637	0.516	**184.658**	0.416	185.102	0.543
F21	**312.711**	1.397	313.432	2.806	315.551	3.058	320.990	11.478	313.676	2.958	317.403	8.411
F22	45.380	0.528	46.634	3.901	48.310	7.498	58.931	19.076	**45.318**	1.917	60.297	18.733
F23	212.922	0.671	212.868	0.582	213.057	0.449	212.888	0.699	**211.733**	0.438	212.072	1.062
F24	**110.676**	6.868	113.276	13.650	116.597	9.293	121.462	10.161	111.856	14.105	150.201	37.613

*StochRBF and NOMADm-DACE are serial algorithms that simulate 1 function evaluation per iteration.

to find 'ESGRBF-8P', which yields a value of 6. On the other hand, the value of 0 in the row 'worse' in the same column indicates that nSOP-8P is not outperformed by ESGRBF-8P on any of the 10 test functions.

From Table 4.5, the values of zero in the first column indicate that nSOP-8P and uSOP-8P are not significantly different. However, nSOP-8P outperforms the compared algorithms on more test functions than uSOP-8P does (total 17 versus 14 test functions).

While nSOP-8P, uSOP-8P, StochRBF-8P, and ESGRBF-8P simulate eight function evaluations per iteration, the serial algorithms Stochastic RBF and NOMADm-DACE simulate one point per iteration and are therefore able to exploit more information during the optimization, and the surrogate is updated after each function evaluation. Hence, given the same number of function evaluations (480), one would expect these serial algorithms to outperform the algorithms that simulate $P = 8$ function evaluations per iteration.

We see that both nSOP-8P and uSOP-8P could attain a better final solution than StochRBF for 2 test problems and are significantly better than NOMADm-DACE for 6 out of 10 test problems.

Table 4.5: The total number of test functions (out of ten) that SOP-8P can attain a better or worse final solution than the compared algorithm at 5% significance level. The data is collected after 480 function evaluations.

(a) nSOP-8P

nSOP-8P vs.	uSOP-8P	StochRBF-8P	ESGRBF-8P	StochRBF*	NOMADm-DACE*	**Total**
better	0	3	6	2	6	17
worse	0	1	0	3	2	6

(b) uSOP-8P

uSOP-8P vs.	nSOP-8P	StochRBF-8P	ESGRBF-8P	StochRBF*	NOMADm-DACE*	**Total**
better	0	2	4	2	6	14
worse	0	0	0	2	2	4

*StochRBF and NOMADm-DACE are serial algorithms that simulate 1 function evaluation per iteration.

Table 4.6: The results for F15–F24 from BBOB testbed using 32 processors. Mean and standard deviation of the best function value after 1920 function evaluations

Alg	**nSOP-32P**		**uSOP-32P**		**StochRBF-32P**		**ESGRBF-32P**	
Test Func	**mean**	**std**	**mean**	**std**	**mean**	**std**	**mean**	**std**
F15	-23.385	5.541	**-30.612**	3.304	12.689	6.762	-0.159	6.700
F16	-257.141	0.707	-257.538	1.052	**-257.637**	1.686	-244.834	1.868
F17	-38.168	0.325	-38.075	0.379	-38.144	0.275	**-38.609**	0.038
F18	-37.224	0.352	-36.604	0.877	-35.666	0.979	**-37.852**	0.604
F19	43.855	0.498	43.885	0.462	44.151	0.683	**43.668**	0.896
F20	185.237	0.241	185.117	0.336	185.480	0.292	**185.081**	0.253
F21	312.001	0.414	**311.774**	0.901	317.249	2.963	315.660	5.573
F22	45.572	0.001	**45.503**	0.216	51.512	10.272	47.490	6.068
F23	**212.361**	0.551	212.584	0.442	212.683	0.524	212.813	0.608
F24	**96.234**	11.835	96.954	10.063	105.678	7.581	96.665	8.889

Table 4.7: The total number of test functions (out of ten) that SOP-32P can attain a better or worse final solution than the compared algorithm at 5% significance level. The data is collected after 1920 function evaluations.

(a) nSOP-32P

nSOP-32P vs.	uSOP-32P	StochRBF-32P	ESGRBF-32P	**Total**
better	1	6	4	11
worse	1	0	2	3

(b) uSOP-32P

uSOP-32P vs.	nSOP-32P	StochRBF-32P	ESGRBF-32P	**Total**
better	1	6	3	10
worse	1	0	2	3

Similarly, we conduct a two-sample t-test to compare the final results obtained in Table 4.6 at the 5% level of significance. Tables 4.7a and b show the number of test functions in which nSOP-32P or uSOP-32P is significantly better or worse than the compared algorithm.

From Table 4.7, we see that both nSOP-32P and uSOP-32 outperform each other on one test function. Both algorithms are better than StochRBF-32P on 6 test functions. nSOP-32P is better than ESGRBF-32P on 4 test functions while uSOP-32P is better than ESGRBF-32P on 3 test functions.

4.5.6 Groundwater Bioremediation Problem

In addition to the BBOB testbed, we also tested our algorithms on a 12-dimensional problem arising in the detoxification of contaminated groundwater using aerobic bioremediation [130]. The constrained global optimization problem aims at finding the cheapest cleanup strategy by determining the pumping rates of oxygenated water into the ground to promote the degradation of the contaminants by microbes. The locations of the injection wells are known. Monitoring wells are also set up in order to measure the concentration of the contaminant

and to ensure that it will be below some threshold level at specified time periods. Because the pumping of oxygenated water is expensive, the goal is to determine the pumping rates for each injection well at the beginning of each management period in order to minimize the total pumping cost required to reduce the contaminant concentration at the monitoring wells to the maximum allowed by EPA regulations. [130] introduced a penalty term to incorporate a contamination constraint into the total pumping cost (objective function). This implementation resulted in a box-constrained global optimization problem. The decision variables are the pumping rates at each of the 3 injection wells in each of the management periods. The four time period problem (resulting in 12 decision variables) is considered. We shall refer to this groundwater problem as GWB12D.

In addition to $P = 8$ and $P = 32$, the results using $P = 64$ are also given in Figure 4.14. The parameters $\mu = 26$, $\lambda = 160$, and $\nu = 64$ are set for ESGRBF-64P. All other algorithm parameters are the same as those used in Section 4.5.5. We did ten trials with each algorithm.

From Figure 4.14, both nSOP and uSOP can make a faster reduction in function values than the alternative methods at each wall-clock time unit. The mean and the standard deviation of the best function value after 60 iterations (hence, 480, 1920, 3840 function evaluations for $P = 8$, 32, 64) are also reported in Table 4.8. We see that SOP can reach the lowest average final value with the smallest standard deviation in all cases (uSOP is best on $P = 8$ and $P = 32$ while nSOP is best on $P = 64$). StochRBF with $P = 8$ was shown in [96] to work well on another groundwater bioremediation model of chlorinated ethenes introduced in [75]. However, here we find that the method did not perform nearly as well as SOP on GWB12D with any of the processor numbers evaluated. This significantly supports the effectiveness of our method.

Similar to that of BBOB, we also carry out a two-sample t-test to compare the final results of GWB12D obtained in Table 4.8a. For any two algorithms $\mathcal{A}_1$, $\mathcal{A}_2$, we write $\mathcal{A}_1 \approx \mathcal{A}_2$ if $\mathcal{A}_1$ and $\mathcal{A}_2$ are not significantly different and write $\mathcal{A}_1 \prec \mathcal{A}_2$ if $\mathcal{A}_1$ is significantly better than

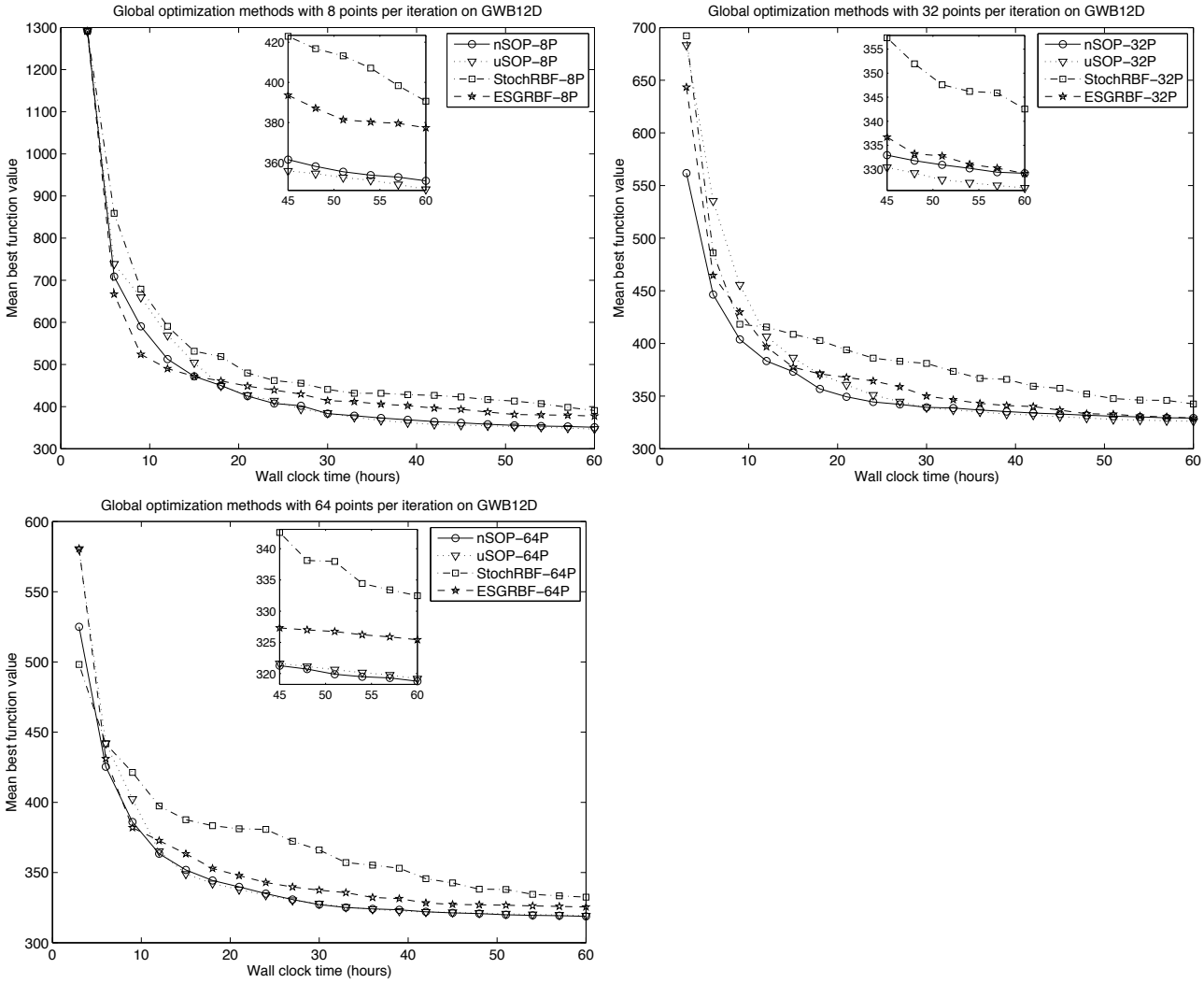


Figure 4.14: Best objective function value on groundwater problem GWB12D averaged over ten trials versus wall-clock time. Eight (top left), thirty-two (top right) and sixty-four (lower left) expensive evaluations are done simultaneously per iteration.

Table 4.8: Results for GWB12D after 60 hours

(a) Mean and standard deviation of the best function value after 60 hours for P=8, 32 and 64 processors on GWB12D. The best value for each P is shown in bold font.

Alg	**nSOP**		**uSOP**		**StochRBF**		**ESGRBF**	
P	**mean**	**std**	**mean**	**std**	**mean**	**std**	**mean**	**std**
8	350.964	17.940	**346.727**	**15.606**	390.614	31.145	377.373	27.752
32	329.166	8.034	**326.112**	**5.016**	342.550	13.997	329.128	9.875
64	**318.774**	**2.465**	319.182	3.658	332.473	4.390	325.420	8.837

(b) Two-sample t-test results for GWB12D at 5% significance level

P	Two-sample t-test results
8	nSOP $\approx$ uSOP $\prec$ ESGRBF $\approx$ StochRBF
32	nSOP $\approx$ uSOP $\approx$ ESGRBF $\prec$ StochRBF
64	nSOP $\approx$ uSOP $\prec$ ESGRBF $\prec$ StochRBF

$\mathcal{A}_2$ at the 5% level of significance. The statistical results are summarized in Table 4.8b.

In short, overall SOP can reduce the function values faster and achieve a better final solution than the alternative methods on more test functions in BBOB and the groundwater bioremediation problem.

4.5.7 Relative Speedup

To compute speedups for computationally expensive functions, we need to be able to compare the number of function evaluations an algorithm required to reach a solution of a certain accuracy. The results can vary according to the accuracy level, which we call $\alpha-$level. Hence, in Chapter 3.5.3.2, we have defined

$$\alpha\text{-Speedup(P)} := \mathcal{I}^{(\alpha)}(1)/\mathcal{I}^{(\alpha)}(P) = n^{(\alpha)}(1)/\left\lceil n^{(\alpha)}(P)/P \right\rceil,$$

where

$$n^{(\alpha)}(P) = \text{argmin}_i\{f(x_i) \leq \alpha \,\text{given}\, P \,\text{processors}\}$$

and

$$\mathcal{I}^{(\alpha)}(P) = \left\lceil n^{(\alpha)}(P)/P \right\rceil$$

are the number of function evaluations and the number of iterations that an algorithm required to reach a specified $\alpha-$level. Note that SOP is not designed to run in serial. If SOP selects only one center per iteration, the point with the lowest objective function value will be selected as the center in each iteration regardless of the results obtained from the Pareto non-dominated sorting (Figures 4.7 and 4.8). We thus use Stochastic RBF, which was proven to be very efficient [95] as a serial algorithm to compute $\mathcal{I}^{(\alpha)}(1)$ and $n^{(\alpha)}(1)$.

The *relative* α-Speedup(P) for each test function is calculated for different α-levels, $\alpha_1 > \alpha_2 > \alpha_3$. To get the three α-levels of nSOP for test functions in BBOB testbed, nSOP(8) and Stochastic RBF are run for 496 function evaluations and nSOP(32) for 1984 function evaluations. Let y_1^*, y_8^*, and y_{32}^* be the average best objective function values obtained from Stochastic RBF, nSOP(8) and nSOP(32), respectively. We set $\alpha_3 = \max\{y_1^*, y_8^*, y_{32}^*\}$, $\alpha_2 = \alpha_3 + |\alpha_3| \times 0.01$ and $\alpha_1 = \alpha_3 + |\alpha_3| \times 0.05$, i.e. α_3 is the smallest value that can be reached by all the three algorithms, and α_2, α_1 are α-values corresponds to relative errors of α_3 at 1% and 5%, respectively. For GWB12D test function, we ran in addition nSOP(64) for 3968 function evaluations. Let y_{64}^* be the average best objective function values obtained from nSOP(64), and define $\alpha_3 = \max\{y_1^*, y_8^*, y_{32}^*, y_{64}^*\}$, $\alpha_2 = \alpha_3 + |\alpha_3| \times 0.01$ and $\alpha_1 = \alpha_3 + |\alpha_3| \times 0.05$.

We calculate also three α-levels for uSOP using the same approach as in nSOP. The α-levels for nSOP and uSOP are shown in Table 4.9.

Table 4.10 shows the α-Speedup(P) for nSOP and uSOP using 8, 32 processors for test functions in BBOB testbed and 8, 32, 64 processors for GWB12D test function. We can see that α-Speedup for each test function varies based on the number of processors and the α-level.

Table 4.9: α-levels

Test Function	nSOP			uSOP		
	α_1	α_2	α_3	α_1	α_2	α_3
F15	-2.876	-2.997	-3.027	-2.876	-2.997	-3.027
F16	-241.649	-251.824	-254.367	-243.050	-253.284	-255.842
F17	-33.626	-35.041	-35.395	-33.626	-35.041	-35.395
F18	-26.742	-27.868	-28.149	-26.742	-27.868	-28.149
F19	47.176	45.378	44.929	46.686	44.908	44.463
F20	194.821	187.399	185.543	194.693	187.276	185.422
F21	329.358	316.811	313.675	329.358	316.811	313.675
F22	47.851	46.028	45.572	48.958	47.093	46.627
F23	223.550	215.034	212.905	223.511	214.996	212.868
F24	115.991	111.572	110.467	118.940	114.409	113.276
GWB12D	367.716	353.708	350.206	362.463	348.655	345.203

Table 4.10: α-Speedup of nSOP and uSOP

Test Function	P	nSOP			uSOP		
		α_1	α_2	α_3	α_1	α_2	α_3
F15	8	11.805	11.976	12.195	11.524	11.690	11.905
	32	19.360	19.640	20.000	20.167	20.458	20.833
F16	8	2.318	2.545	3.159	3.105	3.182	4.889
	32	5.100	4.308	5.378	6.556	5.600	9.059
F17	8	10.333	23.706	27.500	8.857	21.211	24.750
	32	24.800	57.571	61.875	20.667	44.778	49.500
F18	8	17.529	23.000	24.800	12.417	15.607	17.103
	32	37.250	48.556	55.111	29.800	39.727	45.091
F19	8	8.800	6.500	5.689	7.600	9.639	8.267
	32	29.333	37.143	34.700	22.800	31.545	19.840
F20	8	5.636	5.333	2.032	6.200	6.400	2.143
	32	20.667	16.000	3.048	15.500	12.800	3.649
F21	8	3.818	14.056	19.192	3.500	14.056	10.848
	32	8.400	31.625	38.385	7.000	31.625	38.385
F22	8	12.391	10.258	10.578	17.375	13.714	4.698
	32	40.714	31.800	8.207	39.714	36.000	32.889
F23	8	2.000	8.375	1.746	2.000	13.800	2.651
	32	2.000	33.500	5.000	2.000	23.000	4.071
F24	8	7.275	9.226	8.197	6.577	7.220	7.267
	32	13.741	14.382	14.706	24.429	20.286	18.957
GWB12D	8	3.850	3.255	3.000	4.289	3.345	3.397
	32	9.059	8.950	9.000	8.150	7.462	7.926
	64	12.833	11.933	11.812	12.538	12.125	12.588

The algorithm is changed by using P centers to generate candidate points and updating the response surface only once every P evaluations. On some problems, the change is quite helpful and actually reduces the number of evaluations required to reach an α-level. In this case, the speedup is "superlinear", i.e. α-Speedup(P) is greater than P. The superlinear speedup holds for F17, F18, F21, and F22. F17 is Schaffers function, which is highly multimodal– both the frequency and amplitude of the modulation vary. F18 is a moderately ill-conditioned counterpart to F17, with conditioning of about 1000. Both F21 and F22 are Gallagher's Gaussian function, where F21 consists of 101 optima, and the conditioning around the global optimum is about 30. F22 consists of 21 optima with the conditioning around the global optimum is about 1000 [47].

The three functions that have poor scalability (i.e. speedup low compared to P) are F16, F20, and F23. These functions are highly rugged and moderately to highly repetitive.

For F15, the superlinear speedup holds for 8 processors but not for 32 processors. As for F19, and F24, although they do not achieve a superlinear speedup, we can see that the speedup seems to improve in the number of processors. These three functions, despite being highly multimodal, are not as rugged as those mentioned in the previous paragraph. The global amplitude of F15 is large compared to local amplitudes. As for F24, the function was constructed to be deceptive for some evolutionary algorithms with large population size. This might be the reason why the scalability of SOP on this test function deteriorates with 32 processors.

Finally, we do not get good scalability for our groundwater bioremediation application, GWB12D. The speedup is around 3, 9, and 12 for 8, 32, and 64 processors. Recall however none of the other algorithms did as well as SOP (see Figure 4.14) on this blackbox problem.

4.6 Conclusions

Parallel computation has the potential to greatly reduce the wall clock time to solve a global optimization problem for a computationally expensive objective function, but that potential can only be realized if the parallel algorithm is able to effectively select the work to be computed in parallel. Because of the expense of the objective function, the parallel surrogate algorithms (Algorithm 4.1) are designed so that the objective function (simulation) evaluations for P values of the decision vector are done simultaneously (in parallel) and the remaining calculations for optimization (used to determine which new decision variable values should be next evaluated) can be done in serial since they take an insignificant amount of time in comparison to the objective function evaluations. As a result the surrogate is updated only once for each P expensive evaluations done in parallel.

The efficiency of the parallel computation depends on selecting the values of the P decision vectors to be evaluated in each iteration to provide the most useful information over the course of many iterations, which are used to construct the surrogate surface used to help guide the selection of evaluation points. From the results of the n iterations, we obtain the values $\{\left(x_1^{(0)},\, f(x_1^{(0)})\right), ..., \left(x_{n_0}^{(0)},\, f(x_{n_0}^{(0)})\right), \left(x_1^{(1)},\, f(x_1^{(1)})\right), ..., \left(x_P^{(1)},\, f(x_P^{(1)})\right), ..., \left(x_1^{(n)},\, f(x_1^{(n)})\right),$ $..., \left(x_P^{(n)},\, f(x_P^{(n)})\right)\}$. We want these values to both help improve the surrogate surface accuracy over the domain and to help identify the neighborhood of the global minimum, which are two different objectives. To increase efficiency with a large numbers of processors, we improve the diversity of the points selected for the next expensive evaluation by utilizing non-dominated sorting on these two objectives. We also incorporate a tabu structure to further diversify the search. Hence, we have used multi-objective and tabu search approaches to improve parallel computation of single-objective multimodal problems.

In this chapter we introduced the algorithm SOP, which is based on the selection of center points through Pareto non-dominated sorting. Promising points on the Pareto fronts whose function values are small and are far away from other evaluated points are selected as a center. The selected centers are then used for generating a set of candidate points

from which the next function evaluation points are chosen. Multiple centers generate a more diverse set of candidate points.

Two types of random candidate point generations are introduced, namely candidate point generation by uniform perturbations (uSOP) and truncated normal perturbations (nSOP). A more general convergence theorem of Theorem 1 in [95] was proven that allows the possibility that candidate points are generated from many centers as opposed to just one center as was done in [95]. Following this general convergence theorem, the nSOP method is shown to converge to the global optimum almost surely.

In the numerical experiments, we compared SOP to the two available and feasible parallel surrogate methods, ParStochRBF and ESGRBF. For all compared algorithms, evaluations with eight and thirty-two parallel processors are done on 10-dimensional BBOB testbed and a 12-dimensional groundwater bioremediation problem. In addition, 64 parallel evaluations are also done on the latter groundwater problem. In some cases SOP with 8 processors obtained a better result in less wall-clock time than the other algorithms (ParStochRBF and ESGRBF) with 32 processors. The results of these algorithms show that SOP is more efficient on average in making a fast reduction in function values compared to the alternative algorithms, and SOP finds better solutions in many more problems including the groundwater problem.

Chapter 5

Conclusions

In this dissertation, we investigated the performance of surrogate model based optimization algorithms for computationally expensive blackbox functions. (a) We developed a multistart surrogate model framework. (b) We developed a parallel method and investigated the applicability of surrogate models to high-dimensional problems (tested on up to 200 variables). (c) We developed a new parallel algorithm that uses a bi-objective optimization approach for effectively selecting several new evaluation points in each iteration.

5.1 A Surrogate Multistart Framework

In Chapter 2, we propose **SOMS**, which is a multistart method that uses radial basis function models as a surrogate. We use the surrogate model approach to select promising starting points for local searches. We carried out numerical experiments to examine the ability of SOMS to find the global optimum of multimodal functions and we compared SOMS with four other multistart methods on a set of benchmark problems. The numerical experiments showed that SOMS outperforms the other methods and finds a significantly better solution within an equal number of function evaluations. Furthermore, SOMS can be easily interfaced with existing local optimization codes for complex simulation models.

5.2 Surrogate Models for HEB Problems

In Chapter 3, we propose **PADS**, which is a surrogate model based global optimization algorithm for high-dimensional, expensive, blackbox (HEB) problems. PADS uses a dynamic coordinate search strategy in order to deal with the large problem dimensions. We implemented PADS such that several function evaluations can be done in parallel. We compared PADS in numerical experiments with different surrogate-based global optimization algorithms with up to 200 dimensions (which is considered large-scale for the problems usually addressed with surrogate models). The results show that PADS is a very effective method compared to the alternative methods because it finds better solutions within the same unit of wall-clock time.

5.3 Surrogate Model Algorithm with Bi-objective Point Selection Optimization

In Chapter 4, we propose **SOP**, which is a new parallel surrogate model algorithm that selects several evaluation points by solving a computationally cheap bi-objective optimization problem. We investigated the performance of SOP when the number of computationally expensive evaluation points per iteration increases. We simulated up to 64 points in parallel in each iteration in the numerical experiments. This helps push the frontier in surrogate-based optimization since previous methods have done only up to 10 function evaluations per iteration. We compared the performance of SOP with two other RBF based algorithms with various numbers of evaluation points per iteration and found that SOP is efficient and reduces the objective function values faster than the alternatives.

5.4 Future Work

The SOMS algorithm introduced in Chapter 2 can be extended by coupling it with a surrogate-based local optimization algorithm such as ORBIT [124, 125]. This has the advantage that the local optimization will require fewer expensive function evaluations and we expect the performance to be significantly improved. By sharing the points generated during the global phase (from uniform sampling) and the local phase (from the local optimization), a new efficient surrogate multistart algorithm for global optimization can be developed.

PADS and SOP, which are parallel algorithms, can be extended to solve optimization problems that involve also computationally expensive blackbox inequality constraints. This can be done by building several surrogate models, one for the objective function and the others for the constraint functions. These surrogate models can then be used to select the points where the objective and constraint functions will be evaluated next. The objective and constraint function evaluations of all selected points can be done simultaneously. Hence, a significant reduction in wall-clock time can be achieved.

Bibliography

[1] M Abramson, Charles Audet, G Couture, J Dennis Jr, S Le Digabel, and C Tribes. The nomad project. Technical report, Technical report, 2011.

[2] M.A. Abramson. *NOMADm Version 4.5 User's Guide.*, 2007.

[3] Mark A Abramson and Charles Audet. Convergence of mesh adaptive direct search to second-order stationary points. *SIAM Journal on Optimization*, 17(2):606–619, 2006.

[4] Mark A Abramson, Charles Audet, John E Dennis Jr, and Sébastien Le Digabel. Orthomads: A deterministic mads instance with orthogonal directions. *SIAM Journal on Optimization*, 20(2):948–966, 2009.

[5] Enrique Alba, Francisco Luna, and Antonio J Nebro. Advances in parallel heterogeneous genetic algorithms for continuous optimization. *International Journal of Applied Mathematics and Computer Science*, 14(3):317–334, 2004.

[6] M Masoom Ali and C Storey. Topographical multilevel single linkage. *Journal of Global Optimization*, 5(4):349–358, 1994.

[7] P Anand, BVN Siva Prasad, and Ch Venkateswarlu. Modeling and optimization of a pharmaceutical formulation system using radial basis function network. *International journal of neural systems*, 19(02):127–136, 2009.

[8] Charles Audet and John E Dennis Jr. Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on optimization*, 17(1):188–217, 2006.

[9] Charles Audet, John E Dennis Jr, and Sébastien Le Digabel. Parallel space decomposition of the mesh adaptive direct search algorithm. *SIAM Journal on Optimization*, 19(3):1150–1170, 2008.

[10] Ronald W Becker and GV Lago. A global optimization algorithm. In *Proceedings of the 8th Allerton Conference on Circuits and Systems Theory*, pages 3–12, 1970.

[11] Ashok D Belegundu and Tirupathi R Chandrupatla. *Optimization concepts and applications in engineering*. Cambridge University Press, 2011.

[12] Hans-Georg Beyer and Hans-Paul Schwefel. Evolution strategies–a comprehensive introduction. *Natural computing*, 1(1):3–52, 2002.

[13] George Biros and Omar Ghattas. Parallel lagrange–newton–krylov–schur methods for pde-constrained optimization. part i: The krylov–schur solver. *SIAM Journal on Scientific Computing*, 27(2):687–713, 2005.

[14] Bernd Bischl, Simon Wessing, Nadja Bauer, Klaus Friedrichs, and Claus Weihs. Moimbo: Multiobjective infill for parallel model-based optimization.

[15] Mattias Björkman and Kenneth Holmström. Global optimization of costly nonconvex functions using radial basis functions. *Optimization and Engineering*, 1(4):373–397, 2000.

[16] Nikolay Bliznyuk. *Posterior Approximation by Interpolation for Bayesian Inference in Computationally Expensive Statistical Models*. PhD thesis, Cornell University, 2008.

[17] Nikolay Bliznyuk, David Ruppert, Christine Shoemaker, Rommel Regis, Stefan Wild, and Pradeep Mugunthan. Bayesian calibration and uncertainty analysis for computationally expensive models using optimization and radial basis function approximation. *Journal of Computational and Graphical Statistics*, 17(2), 2008.

[18] Nikolay Bliznyuk, David Ruppert, and Christine A Shoemaker. Local derivative-free approximation of computationally expensive posterior densities. *Journal of Computational and Graphical Statistics*, 21(2):476–495, 2012.

[19] C Guus E Boender and H Edwin Romeijn. Stochastic methods. In *Handbook of global optimization*, pages 829–869. Springer, 1995.

[20] CGE Boender, AHG Rinnooy Kan, GT Timmer, and L Stougie. A stochastic method for global optimization. *Mathematical programming*, 22(1):125–140, 1982.

[21] Andrew J Booker, JE Dennis Jr, Paul D Frank, David B Serafini, and Virginia Torczon. Optimization using surrogate objectives on a helicopter test example. In *Computational Methods for Optimal Design and Control*, pages 49–58. Springer, 1998.

[22] Andrew J Booker, JE Dennis Jr, Paul D Frank, David B Serafini, Virginia Torczon, and Michael W Trosset. A rigorous framework for optimization of expensive functions by surrogates. *Structural optimization*, 17(1):1–13, 1999.

[23] Stephen P Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[24] Samuel H Brooks. A discussion of random methods for seeking maxima. *Operations research*, 6(2):244–251, 1958.

[25] Stella M Clarke, Jan H Griebsch, and Timothy W Simpson. Analysis of support vector regression for approximation of complex engineering analyses. *Journal of mechanical design*, 127(6):1077–1087, 2005.

[26] Computational and Information Systems Laboratory. Yellowstone: Ibm idataplex system (university community computing). boulder, co: National center for atmospheric research. http://n2t.net/ark:/85065/d7wd3xhc, 2012.

[27] Andrew R Conn, Katya Scheinberg, and Ph L Toint. Recent progress in unconstrained nonlinear optimization without derivatives. *Mathematical programming*, 79(1-3):397–414, 1997.

[28] AR Conn, K Scheinberg, and Ph L Toint. A derivative free optimization algorithm in practice. In *Proceedings of 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, St. Louis, MO*, volume 48, 1998.

[29] Tibor Csendes. Nonlinear parameter estimation by global opitmization - efficiency and reliability. *Acta Cybern.*, 8:361–372, 1988.

[30] Tibor Csendes, László Pál, J Oscar H Sendín, and Julio R Banga. The global optimization method revisited. *Optimization Letters*, 2(4):445–454, 2008.

[31] Kalyanmoy Deb. *Multi-objective optimization using evolutionary algorithms*, volume 16. John Wiley & Sons, 2001.

[32] Kalyanmoy Deb. Introduction to evolutionary multiobjective optimization. In *Multiobjective Optimization*, pages 59–96. Springer, 2008.

[33] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and Tanaka Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. *Lecture notes in computer science*, 1917:849–858, 2000.

[34] LCW Dixon and GP Szegö. The global optimization problem: an introduction. *Towards global optimization*, 2:1–15, 1978.

[35] Michael S Eldred, Anthony A Giunta, Bart G van Bloemen Waanders, Steven F Wojtkiewicz, William E Hart, and Mario P Alleva. *DAKOTA, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis: Version 4.1 reference manual*. Sandia National Laboratories Albuquerque, NM, 2007.

[36] Michael TM Emmerich and Carlos M Fonseca. Computing hypervolume contributions in low dimensions: asymptotically optimal algorithm and complexity results. In *Evolutionary Multi-Criterion Optimization*, pages 121–135. Springer, 2011.

[37] Antoine Jean Espinet. *Optimal parameter and CO2 plume estimation and uncertainty quantification using process-based multiphase models and monitoring data for geological carbon sequestration*. Cornell University, 2012.

[38] Mark Fleischer. The measure of pareto optima applications to multi-objective metaheuristics. In *Evolutionary multi-criterion optimization*, pages 519–533. Springer, 2003.

[39] Carlos M Fonseca, Joshua D Knowles, Lothar Thiele, and Eckart Zitzler. A tutorial on the performance assessment of stochastic multiobjective optimizers. In *Third International Conference on Evolutionary Multi-Criterion Optimization (EMO 2005)*, volume 216, page 13, 2005.

[40] Alexander Forrester, Andras Sobester, and Andy Keane. *Engineering design via surrogate modelling: a practical guide*. John Wiley & Sons, 2008.

[41] Alexander IJ Forrester and Andy J Keane. Recent advances in surrogate-based optimization. *Progress in Aerospace Sciences*, 45(1):50–79, 2009.

[42] Jerome H Friedman. Multivariate adaptive regression splines. *The annals of statistics*, pages 1–67, 1991.

[43] Anthony A Giunta. *Aircraft multidisciplinary design optimization using design of experiments theory and response surface modeling methods*. PhD thesis, virginia polytechnic institute and state university, 1997.

[44] William Greene. H.(2003): Econometric analysis. *New Jersey, ua: Prentice Hall*, 2003.

[45] H.-M. Gutmann. A radial basis function method for global optimization. *Journal of Global Optimization*, 19:201–227, 2001.

[46] Zhong-Hua Han and Ke-Shi Zhang. Surrogate-based optimization. *Intec Book, Real-World Application of Genetic Algorithm. InTech*, pages 343–362, 2012.

[47] Nikolaus Hansen, Steffen Finck, Raymond Ros, Anne Auger, et al. Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions. 2009.

[48] Francisco Herrera, Manuel Lozano, and Jose L. Verdegay. Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis. *Artificial intelligence review*, 12(4):265–319, 1998.

[49] K Holmström, AO Göran, and MM Edvall. User's guide for tomlab 7. tomlab optimization, 2006.

[50] K Holmström, AO Göran, and MM Edvall. *User's Guide for TOMLAB/CGO. Tomlab Optimization*. http://tomopt.com, 2007.

[51] Kenneth Holmström and Marcus M Edvall. The tomlab optimization environment. In *Modeling Languages in Mathematical Optimization*, pages 369–376. Springer, 2004.

[52] Reiner Horst, Panos M Pardalos, and H Edwin Romeijn. *Handbook of global optimization*, volume 2. Springer, 2002.

[53] Serhat Hosder, Layne T Watson, Bernard Grossman, William H Mason, Hongman Kim, Raphael T Haftka, and Steven E Cox. Polynomial response surface approximations for the multidisciplinary design optimization of a high speed civil transport. *Optimization and Engineering*, 2(4):431–452, 2001.

[54] Deng Huang, Theodore T Allen, William I Notz, and N Zeng. Global optimization of stochastic black-box systems via sequential kriging meta-models. *Journal of global optimization*, 34(3):441–466, 2006.

[55] Yaochu Jin. Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation*, 1(2):61–70, 2011.

[56] Donald R Jones. A taxonomy of global optimization methods based on response surfaces. *Journal of global optimization*, 21(4):345–383, 2001.

[57] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

[58] AHG Rinnooy Kan and GT Timmer. Stochastic methods for global optimization. *American Journal of Mathematical and Management Sciences*, 4(1-2):7–40, 1984.

[59] AHG Rinnooy Kan and GT Timmer. Stochastic global optimization methods part i: Clustering methods. *Mathematical programming*, 39(1):27–56, 1987.

[60] AHG Rinnooy Kan and GT Timmer. Stochastic global optimization methods part ii: Multi level methods. *Mathematical Programming*, 39(1):57–78, 1987.

[61] AJ Keane. Design search and optimisation using radial basis functions with regression capabilities. In *Adaptive Computing in Design and Manufacture VI*, pages 39–49. Springer, 2004.

[62] Patrick N Koch, Timothy W Simpson, Janet K Allen, and Farrokh Mistree. Statistical approximations for multidisciplinary design optimization: the problem of size. *Journal of Aircraft*, 36(1):275–286, 1999.

[63] Slawomir Koziel and Leifur Leifsson. *Surrogate-Based Modeling and Optimization: Applications in Engineering*. Springer Science & Business, 2013.

[64] Sergei Kucherenko and Yury Sytsko. Application of deterministic low-discrepancy sequences in global optimization. *Computational Optimization and Applications*, 30(3):297–318, 2005.

[65] Sébastien Le Digabel. Algorithm 909: Nomad: Nonlinear optimization with the mads algorithm. *ACM Transactions on Mathematical Software (TOMS)*, 37(4):44, 2011.

[66] Sébastien Le Digabel, Mark A Abramson, Charles Audet, and JE Dennis Jr. Parallel versions of the mads algorithm for black-box optimization. *Optimization days, Montreal, May*, 2010.

[67] Sébastien Le Digabel, Christophe Tribes, and Charles Audet. Nomad user guide version 3.6. 2.

[68] Stephen J Leary, Atul Bhaskar, and Andy J Keane. A knowledge-based approach to response surface modelling in multifidelity optimization. *Journal of Global Optimization*, 26(3):297–319, 2003.

[69] Marco Locatelli. A note on the griewank test function. *Journal of global optimization*, 25(2):169–174, 2003.

[70] Marco Locatelli and Fabio Schoen. Random linkage: a family of acceptance/rejection algorithms for global optimisation. *Mathematical Programming*, 85(2):379–396, 1999.

[71] Craig Loehle. Global optimization using mathematica: A test of software tools. *Mathematica in Education and Research*, 11(2), 2006.

[72] Søren Nymand Lophaven, Hans Bruun Nielsen, and Jacob Søndergaard. Dace-a matlab kriging toolbox, version 2.0. Technical report, 2002.

[73] Zbigniew Michalewicz. *Genetic algorithms+ data structures= evolution programs*. springer, 1996.

[74] Jorge J Moré, Burton S Garbow, and Kenneth E Hillstrom. Testing unconstrained optimization software. *ACM Transactions on Mathematical Software (TOMS)*, 7(1):17–41, 1981.

[75] Pradeep Mugunthan, Christine A Shoemaker, and Rommel G Regis. Comparison of function approximation, heuristic, and derivative-based methods for automatic calibration of computationally expensive groundwater bioremediation models. *Water Resources Research*, 41(11), 2005.

[76] Juliane Müller and Robert Piché. Mixture surrogate models based on dempster-shafer theory for global optimization problems. *Journal of Global Optimization*, 51(1):79–104, 2011.

[77] Juliane Müller and Christine A Shoemaker. Influence of ensemble surrogate models and sampling strategy on the solution quality of algorithms for computationally expensive black-box global optimization problems. *Journal of Global Optimization*, pages 1–22, 2014.

[78] Raymond H Myers and Christine M Anderson-Cook. *Response surface methodology: process and product optimization using designed experiments*, volume 705. John Wiley & Sons, 2009.

[79] Jorge Nocedal and SJ Wright. Numerical optimization: Springer series in operations research and financial engineering. *Springer-Verlag*, 2006.

[80] Esin Onbaşoğlu and Linet Özdamar. Parallel simulated annealing algorithms in global optimization. *Journal of Global Optimization*, 19(1):27–50, 2001.

[81] Yew S Ong, Prasanth B Nair, and Andrew J Keane. Evolutionary optimization of computationally expensive problems via surrogate modeling. *AIAA journal*, 41(4):687–696, 2003.

[82] Ibrahim H Osman and James P Kelly. *Meta-heuristics: theory and applications*. Springer, 1996.

[83] László Pál, Tibor Csendes, Mihály Csaba Markót, and Arnold Neumaier. Black box optimization benchmarking of the global method. *Evolutionary computation*, 20(4):609–639, 2012.

[84] Victor Picheny, Tobias Wagner, and David Ginsbourger. A benchmark of kriging-based infill criteria for noisy optimization. *Structural and Multidisciplinary Optimization*, 48(3):607–626, 2013.

[85] Todd D Plantenga. Hopspack 2.0 user manual. *Sandia National Laboratories Technical Report Sandia National Laboratories Technical Report SAND2009-6265*, 2009.

[86] Hartmut Pohlheim. Examples of objective functions. *Retrieved*, 4(10):2012, 2007.

[87] Petr Pošík and Jiří Kubalík. Experimental comparison of six population-based algorithms for continuous black box optimization. *Evolutionary computation*, 20(4):483–508, 2012.

[88] Michael JD Powell. The newuoa software for unconstrained optimization without derivatives. In *Large-scale nonlinear optimization*, pages 255–297. Springer, 2006.

[89] M.J.D. Powell. *The Theory of Radial Basis Function Approximation in 1990.* Advances in Numerical Analysis, vol. 2: wavelets, subdivision algorithms and radial basis functions. Oxford University Press, Oxford, pp. 105-210, 1992.

[90] Rommel G Regis. Stochastic radial basis function algorithms for large-scale optimization involving expensive black-box objective and constraint functions. *Computers & Operations Research*, 38(5):837–853, 2011.

[91] Rommel G Regis and Christine A Shoemaker. Local function approximation in evolutionary algorithms for the optimization of costly functions. *Evolutionary Computation, IEEE Transactions on*, 8(5):490–505, 2004.

[92] Rommel G Regis and Christine A Shoemaker. Constrained global optimization of expensive black box functions using radial basis functions. *Journal of Global Optimization*, 31(1):153–171, 2005.

[93] Rommel G Regis and Christine A Shoemaker. Improved strategies for radial basis function methods for global optimization. *Journal of Global Optimization*, 37(1):113–135, 2007.

[94] Rommel G Regis and Christine A Shoemaker. Parallel radial basis function methods for the global optimization of expensive functions. *European journal of operational research*, 182(2):514–535, 2007.

[95] Rommel G Regis and Christine A Shoemaker. A stochastic radial basis function method for the global optimization of expensive functions. *INFORMS Journal on Computing*, 19(4):497–509, 2007.

[96] Rommel G Regis and Christine A Shoemaker. Parallel stochastic global optimization using radial basis functions. *INFORMS Journal on Computing*, 21(3):411–426, 2009.

[97] Rommel G Regis and Christine A Shoemaker. Combining radial basis function surrogates and dynamic coordinate search in high-dimensional expensive black-box optimization. *Engineering Optimization*, 45(5):529–555, 2013.

[98] Rommel G Regis and Christine A Shoemaker. A quasi-multistart framework for global optimization of expensive functions using response surface models. *Journal of Global Optimization*, 56(4):1719–1753, 2013.

[99] Luis Miguel Rios and Nikolaos V Sahinidis. Derivative-free optimization: A review of algorithms and comparison of software implementations. *Journal of Global Optimization*, 56(3):1247–1293, 2013.

[100] Olivier Roustant, David Ginsbourger, Yves Deville, et al. Dicekriging, diceoptim: Two r packages for the analysis of computer experiments by kriging-based metamodeling and optimization. 2012.

[101] Michael J Sasena, Panos Papalambros, and Pierre Goovaerts. Exploration of metamodeling sampling criteria for constrained global optimization. *Engineering optimization*, 34(3):263–278, 2002.

[102] Fabio Schoen. A wide class of test functions for global optimization. *Journal of Global Optimization*, 3(2):133–137, 1993.

[103] Fabio Schoen. Two-phase methods for global optimization. In *Handbook of global optimization*, pages 151–177. Springer, 2002.

[104] Hans-Paul Schwefel. *Numerical optimization of computer models*. John Wiley & Sons, Inc., 1981.

[105] Raymond Séroul. *Programming for mathematicians*. Springer, 2000.

[106] Songqing Shan and G Gary Wang. Survey of modeling and optimization strategies to solve high-dimensional design problems with computationally-expensive black-box functions. *Structural and Multidisciplinary Optimization*, 41(2):219–241, 2010.

[107] Christine A Shoemaker, Rommel G Regis, and Ryan C Fleming. Watershed calibration using multistart local optimization and evolutionary optimization with radial basis function approximation. *Hydrological sciences journal*, 52(3):450–465, 2007.

[108] Timothy W Simpson, Timothy M Mauery, John J Korte, and Farrokh Mistree. Kriging models for global approximation in simulation-based multidisciplinary design optimization. *AIAA journal*, 39(12):2233–2241, 2001.

[109] Alex J Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and computing*, 14(3):199–222, 2004.

[110] András Sobester, SJ Leary, and Andy J Keane. A parallel updating scheme for approximating and optimizing high fidelity computer simulations. *Structural and multidisciplinary optimization*, 27(5):371–383, 2004.

[111] András Sóbester, Stephen J Leary, and Andy J Keane. On the design of optimization strategies based on global response surface approximation models. *Journal of Global Optimization*, 33(1):31–59, 2005.

[112] A Srivastava, K Hacker, K Lewis, and TW Simpson. A method for using legacy data for metamodel-based design of large-scale systems. *Structural and Multidisciplinary Optimization*, 28(2-3):146–155, 2004.

[113] Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

[114] Gerrit Theodoor Timmer. *Global optimization: A stochastic approach*. PhD thesis, Centrum voor Wiskunde en Informatica, 1984.

[115] Bryan A Tolson and Christine A Shoemaker. Dynamically dimensioned search algorithm for computationally efficient watershed model calibration. *Water Resources Research*, 43(1), 2007.

[116] A Törn. Global optimization as a combination of global and local search, gothenburg business adm. *Studies*, 17:191–206, 1973.

[117] Aimo Törn and Antanas Zilinskas. *Global optimization*. Springer-Verlag New York, Inc., 1989.

[118] Aimo A Törn. *A search-clustering approach to global optimization*. Åbo Swedish University School of Economics, 1977.

[119] Zsolt Ugray, Leon Lasdon, John Plummer, Fred Glover, James Kelly, and Rafael Martí. Scatter search and local nlp solvers: A multistart framework for global optimization. *INFORMS Journal on Computing*, 19(3):328–340, 2007.

[120] Felipe AC Viana, Raphael T Haftka, and Layne T Watson. Efficient global optimization algorithm assisted by multiple surrogate techniques. *Journal of Global Optimization*, 56(2):669–689, 2013.

[121] Chankong Vira and Yacov Y Haimes. *Multiobjective decision making: theory and methodology*. Number 8. North-Holland, 1983.

[122] G Gary Wang and S Shan. Review of metamodeling techniques in support of engineering design optimization. *Journal of Mechanical Design*, 129(4):370–380, 2007.

[123] Lyndon While, Philip Hingston, Luigi Barone, and Simon Huband. A faster algorithm for calculating hypervolume. *Evolutionary Computation, IEEE Transactions on*, 10(1):29–38, 2006.

[124] Stefan M Wild, Rommel G Regis, and Christine A Shoemaker. Orbit: Optimization by radial basis function interpolation in trust-regions. *SIAM Journal on Scientific Computing*, 30(6):3197–3219, 2008.

[125] Stefan M Wild and Christine Shoemaker. Global convergence of radial basis function trust-region algorithms for derivative-free optimization. *SIAM Review*, 55(2):349–371, 2013.

[126] Stefan Martin Wild. *Derivative-free optimization algorithms for computationally expensive functions*. PhD thesis, Cornell University, 2009.

[127] Yong Wu, L Ozdamar, and A Kumar. Triopt: a triangulation-based partitioning algorithm for global optimization. *Journal of computational and applied mathematics*, 177(1):35–53, 2005.

[128] Yong Wu, Linet Özdamar, and Arun Kumar. Parallel triangulated partitioning for black box optimization. In *Global Optimization*, pages 487–506. Springer, 2006.

[129] Kenny Q Ye, William Li, and Agus Sudjianto. Algorithmic construction of optimal symmetric latin hypercube designs. *Journal of statistical planning and inference*, 90(1):145–159, 2000.

[130] Jae-Heung Yoon and Christine A Shoemaker. Comparison of optimization methods for ground-water bioremediation. *Journal of Water Resources Planning and Management*, 125(1):54–63, 1999.

[131] Zelda B Zabinsky, David Bulger, and Charoenchai Khompatraporn. Stopping and restarting strategy for stochastic sequential search in global optimization. *Journal of Global Optimization*, 46(2):273–286, 2010.

[132] Eckart Zitzler, Dimo Brockhoff, and Lothar Thiele. The hypervolume indicator revisited: On the design of pareto-compliant indicators via weighted integration. In *Evolutionary Multi-Criterion Optimization*, pages 862–876. Springer, 2007.

[133] Eckart Zitzler, Joshua Knowles, and Lothar Thiele. Quality assessment of pareto set approximations. In *Multiobjective Optimization*, pages 373–404. Springer, 2008.

[134] Eckart Zitzler and Lothar Thiele. Multiobjective optimization using evolutionary algorithmsÑa comparative case study. In *Parallel problem solving from natureÑPPSN V*, pages 292–301. Springer, 1998.

[135] Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M Fonseca, and Viviane Grunert Da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *Evolutionary Computation, IEEE Transactions on*, 7(2):117–132, 2003.