ALGORITHMS FOR DERIVATIVE-FREE OPTIMIZATION

BY

LUIS MIGUEL RIOS

B.S., Pontificia Universidad Catolica del Peru, 2000

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Industrial Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2009

Urbana, Illinois

Doctoral Committee:

Adjunct Professor Nikolaos Sahinidis, Chair, Director of Research
Professor Sheldon Jacobson
Professor Barbara Minsker
Assistant Professor Vinyah Shanbhag

Abstract

Fueled by a growing number of applications in science and engineering, the development of derivative-free optimization algorithms has long been studied and found renewed interest. The problem addressed is the optimization of a deterministic function $f : \mathbb{R}^n \to \mathbb{R}$ over a domain of interest that possibly includes constraints $g(x) \leq 0$, with $g : \mathbb{R}^n \to \mathbb{R}^m$. We assume that the derivatives of f and g are neither symbolically available nor numerically computable, and that bounds, such as Lipschitz constants, for the derivatives of f and g are also unavailable.

In this thesis, we begin by presenting a comprehensive list of available methods and software and performing an extensive computational study that compares the solvers over a publicly available problem set. Then, we develop *Model and Search* (`M&S`), a new local search algorithm for derivative-free optimization. `M&S` performs a local search from a given point. The search is guided by identifying descent directions from a quadratic model fitted around the best known point, while using information from other evaluated points. We prove that `M&S` enjoys global convergence to a stationary point. We also propose a new global search algorithm for derivative-free optimization problems, in particular the *Branch and Model* (`B&M`) algorithm that is based on modeling the function of interest around each evaluated point by using information from other nearby evaluated points. Algorithm `B&M` is shown to perform a dense search and thus converge to a global minimum. While oriented towards a global search, `B&M` relies on the `M&S` algorithm for occasional local searches. Finally, we present an application of derivative-free solvers, including `B&M`, to the protein-ligand docking problem.

Results show that B&M delivers satisfactory ligand conformations, even outperforming the state-of-the-art protein docking software AutoDock.

To my parents, Luis and Lourdes

Acknowledgements

I express my thanks and appreciation to all those who supported and helped me during my graduate studies. I am deeply indebted to my advisor Nick. His tolerance, guidance and support were vital throughout my PhD program, from the moment I was admitted to UIUC and beyond my graduation. Apart from engineering and computational concepts, I have learned from Nick not only engineering and computational concepts but also to have endurance to achieve goals, goals with difficulties that at times seem to be too difficult to overcome. I feel deeply lucky to have such an incredible advisor and supporter.

I would like to thank the Optimization Technology Team from the Upstream Research Company at Exxon-Mobil for having me as an intern twice, and eventually presenting me the research area that become the main topic of this dissertation. I would like to express my gratitude to Federico Carvallo, Amr El-Bakry and Adam Singer for making my internships very rewarding experiences and helping me achieve the goals of the projects.

I want to say thank you to the former and current fellows at the Sigma Optimization Group, including Anastasia Vaia, YoungJung Chang, Wei Xie, Alex Smith, Joseph Elble, and Xiaowei Bao. I enjoyed and learned a lot from the time we spent at the office and at our group meetings. I also want to thank Marco Milla, Pablo Reyes, Roberto Lavarello and Kely Garcia for many interesting conversations and programming tips.

I want to thank Joseph for reading this thesis and making many improvements.

Lastly, I would like to thank my wife, Ana Sofia, for supporting me during all those days and nights I spent studying and working. I could not have accomplished this without her support. To my parents, Luis and Lourdes, I thank you for all your love and being a positive example in my life. Thanks for believing in me.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The problem addressed in this thesis is the optimization of a deterministic function $f : \mathbb{R}^n \to \mathbb{R}$ over a box-bounded domain of interest $\mathcal{C}$. We assume that the derivatives of f are neither symbolically available nor numerically computable, and that bounds, such as Lipschitz constants, for the derivatives of f are also unavailable. The problem is of interest when derivative information is unavailable, unreliable, or impractical to obtain. Furthermore, the function f is assumed expensive to evaluate or somewhat noisy. We refer to this problem as *derivative-free optimization.*

Derivative-free optimization is an area of long history and current rapid growth, fueled by increasing interest in broad applications, ranging from science problems [5, 40, 41, 54, 139] to engineering design and facility location problems [3, 14, 16, 49, 50, 57, 59, 93, 94].

Derivative-free algorithms and their computational implementations are the preferred option for optimization of problems for which function evaluations are expensive and derivative information is unavailable.

The development of derivative-free algorithms dates back to the works of Spendley *et al.* [130] and Nelder and Mead [101] with their simplex-based algorithms, while recent works on the subject have included convergence properties [6, 9, 29, 81, 88, 90] and incorporate the use of simplex gradients [7, 32, 37]. Significant progress on derivative-free optimization has resulted from all this work, especially thanks to the development of strategies to select the most promising points to explore based on the optimization of surrogate models. Concurrent with the development of algorithms,

software implementations for this class of optimization problems have resulted in a wealth of software packages. Since its beginnings in the 1960's, derivative-free optimization has grown from being considered a set of heuristics to locally and globally convergent algorithms [80].

Chapter 1 presents a comprehensive review of state-of-the-art derivative-free optimization methods and software implementations. A set of 20 leading software implementations were tested on a large collection of publicly available problems. Computational results show that attaining the best solutions even for very small problems is a challenge for most of these solvers.

Chapter 2.8 proposes the *Model and Search* (`M&S`) derivative-free optimization algorithm and proves global convergence to a stationary point. The search is oriented to improve the objective of the best known point by using models fitted by information from nearby evaluated points. Designed primarily to find local minima, `M&S`'s ideal role is to refine points identified by other algorithms with broader global capabilities. In particular, `M&S` is docked with the derivative-free global algorithm `B&M` proposed in Chapter 3.5. Algorithm `B&M` prioritizes reusing information from previously evaluated points to guide the search before evaluating new points. The algorithm is based on modeling the function using a collection of surrogate models. Each surrogate model has an area of influence around an evaluated point. The area of influence is determined during each iteration by a partitioning algorithm. `B&M` is shown to be convergent in the limit to a global minimum by guaranteeing that the search is dense.

Both `M&S` and `B&M` were implemented in `MATLAB` and tested over publicly available problems that we collected. Results show that the performance of the proposed algorithms is comparable or better than state-of-the-art derivative-free implementations.

Chapter 4.5 presents an application of DFO solvers, including our `B&M` solver, to the protein-ligand docking problem. Results show that `B&M` delivers satisfactory ligand conformations, even outperforming the state-of-the-art software `AutoDock`.

Finally, Chapter 6 presents conclusions of this thesis, while a unified interface to existing derivative-free solvers is described in the Appendix.

Chapter 2

Derivative-free optimization: A review and comparison

2.1 Introduction

This chapter addresses the solution of optimization problems using algorithms that require only the availability of objective and constraint function values but no derivative information. We refer to these algorithms as derivative-free algorithms. Fueled by a growing number of applications in science and engineering, the development of derivative-free optimization algorithms has long been studied and found renewed interest in recent time. Along with many derivative-free algorithms, many software implementations have also appeared. The chapter presents a brief review of derivative-free algorithms, followed by a systematic comparison of 20 related implementations using a test set of 258 problems. The algorithms are tested under the same conditions and ranked under several criteria, including their ability to find global solutions to nonconvex problems. We find that even obtaining feasible solutions is not always easy with these solvers, and their ability to obtain good solutions diminishes with increasing problem size. While `LGO` and `MCS` are better, on average, than other derivative-free solvers, all solvers tested are useful in the sense that there are at least a few problems for which each solver is best in terms of solution quality.

The problem addressed in this chapter is the optimization of a deterministic function $f : \mathbb{R}^n \to \mathbb{R}$ over a domain of interest that possibly includes constraints $g(x) \leq 0$, with $g : \mathbb{R}^n \to \mathbb{R}^m$. We assume that the derivatives of f and g are neither symbolically available nor numerically computable, and that bounds, such as Lipschitz constants,

for the derivatives of f and g are also unavailable.

The problem is of interest when derivative information is unavailable, unreliable, or impractical to obtain, for instance when f is expensive to evaluate or somewhat noisy. We refer to this problem as *derivative-free optimization*. We further refer to any algorithm applied to this problem as a *derivative-free algorithm*, even if the algorithm involves the computation of derivatives for functions other than f and g.

Derivative-free optimization is an area of long history and current rapid growth, fueled by a growing number of applications that range from science problems [5, 40, 54, 139] to medical problems [92, 106] to engineering design and facility location problems [3, 14, 16, 49, 50, 57, 59, 93, 94].

The development of derivative-free algorithms dates back to the works of Spendley *et al.* [130] and Nelder and Mead [101] with their simplex-based algorithms, while recent works on the subject have included convergence properties [6, 9, 29, 81, 88, 90] and incorporate the use of simplex gradients [7, 32, 37]. Significant progress on derivative-free optimization has resulted from all this work, especially thanks to the development of strategies to select the most promising points to explore based on the optimization of surrogate models. Concurrent with the development of algorithms, software implementations for this class of optimization problems have resulted in a wealth of software packages, including five `COLINY` solvers [58], `DFO` [125], `EGO` [62, pp. 11–24], `GLBsolve` [61, pp. 125–127], `GLCsolve` [61, pp. 142-146], `IMFIL` [76], `LGO` [109], `MCS` [104], `NEWUOA` [115], `NOMAD` [34], `PSWARM` [134], `RBFsolve` [62, pp. 5–10], and `SID-PSM` [38].

Mongeau *et al.* [98] performed a comparison of derivative-free solvers in 1998 by considering six solvers over a set of eleven test problems. Since the time of that comparison, 16 new derivative-free optimization solvers have become available. Other comparisons, such as [11, 31, 36, 49, 55, 66, 68, 99, 135], are restricted to a few solvers related to the algorithms proposed in these papers. In addition, these comparisons

consider small sets of test problems. There is currently no systematic comparison of the current derivative-free optimization algorithm implementations on a large collection of test problems. The primary purpose of the current chapter is to present such a comparison, aiming at addressing the following questions:

- What is the quality of solutions obtained by current solvers for a given limit on the number of allowable function evaluations? Does quality drop significantly as problem size increases?

- Which solver is more likely to obtain global or near-global solutions for nonconvex problems? Does a multi-start strategy change the relative performance of solvers in this regard?

- Is there a subset of existing solvers that would suffice to solve a large fraction of problems when all solvers are independently applied to all problems of interest? Conversely, are there problems which are solved only by one or a few solvers?

Before addressing these questions computationally, the chapter begins by presenting a review of the underlying theory and motivational ideas of the algorithms. We classify algorithms developed for this problem as *direct* and *indirect*. Direct algorithms determine search directions by computing values of the function f directly, whereas indirect algorithms construct and utilize a surrogate model of f or its derivatives to guide the search process. Algorithms are further classified as *stochastic* or *deterministic*, depending upon whether they require random search steps or not.

Section 2.2 presents direct algorithms. These include some of the oldest and best known strategies for the problem, including the Hooke and Jeeves method and the Nelder-Mead simplex algorithm, both of which have been used extensively over the past few decades. Section 2.3 presents algorithms that are based on the optimization of a surrogate model of the real function. First, we describe two basic approaches that make use of quadratic and more complex models, including radial basis functions

and polynomials, to approximate the objective to be optimized. Then, we present algorithms that estimate a Lipschitz constant for f, thus allowing for the construction of an underestimating surrogate function. These methods permit global optimization since the underestimator can be refined by partitioning the search space. Stochastic algorithms are reviewed in Section 2.4. Section 2.5 concludes the review of algorithms by a brief historical overview and overall discussion. Leading software implementations of the algorithms are discussed in Section 2.6. The discussion includes software characteristics, requirements, and types of problems handled. Extensive computational experience with these software is presented in Section 2.7. A total of 258 test problems from the `globallib`, `princetonlib` and LukŠan and VlŠek [91] collections were used to test 20 solvers using using uniform starting points and bounding boxes on all variables. Finally, conclusions are drawn in Section 2.8.

Readers familiar with the subject matter of this chapter may have noticed that there exists literature that reserves the term *derivative-free algorithms* only for what we refer to as *indirect algorithms* in the current chapter. In addition, what we refer to as *derivative-free optimization* is often also referred to as *optimization over black boxes*. Furthermore, what we refer to as *direct algorithms* in the current chapter is often referred to as *direct search*. The literature on these terms is inconsistent and often confusing (cf. [80] and discussion therein).

2.2 Direct search methods

Hooke and Jeeves [64] describe *direct search* as the sequential examination of trial solutions generated by a certain strategy. Classical direct search methods did not come with proofs of termination or convergence to stationary points. However, recent papers starting with [131, 132] have proved convergence to stationary points, among other properties. Despite their long history, these methods remain popular due to

their simplicity, flexibility, and reliability. We next describe specific direct search methods.

2.2.1 Nelder-Mead simplex algorithm

The Nelder-Mead algorithm introduced in [101] starts with a set of points that form a simplex. In each iteration, the algorithm attempts to replace the worst vertex of the simplex with a better point that still forms a simplex with the remaining vertices. Candidate replacement points are obtained by transforming the worst vertex through a number of operations about the centroid of the current simplex: reflection, expansion, inside and outside contractions. If there is no improvement of the current worst vertex, a contraction operation is performed, in which only the best vertex is kept. Computational experiments demonstrated that the Nelder-Mead algorithm can stagnate even in simple smooth convex problems [96]. To prevent stagnation, Kelley [75] proposed to enforce a sufficient decrease condition determined by an approximation of the gradient. If stagnation is observed, the algorithm restarts from a different simplex.

2.2.2 Generating set search (GSS) methods

Torczon [132] introduced generalized pattern search methods (GPS) for unconstrained optimization. GPS generalizes direct search methods including the Hooke and Jeeves [64] algorithm. At the beginning of iteration k, GPS searches by *exploratory moves* around the current iterate x_k. The points to be evaluated form a *pattern* of points around x_k and are determined by a step δ_k and a *generating matrix* C_k that spans $\mathbb{R}^n$.

Further generalizing GPS, Kolda, Lewis and Torczon [80] coined the term GSS in order to describe, unify, and analyze direct search methods that apply to constrained problems. The main idea of GSS methods is to restrict the search to directions that

span the cone defined by the "almost active" constraint normals, *i.e.*, constraints that are binding within a neighborhood of the current iterate.

Each iteration k of GSS methods consists of two basic steps. The *search* step is performed first over a finite set of search directions $\mathcal{H}_k$ generated by some, possibly heuristic, strategy that aims to improve the current iterate but may not guarantee convergence. If the search step fails to produce a better point, GSS methods continue with the *poll* step, which is associated with a generating set $\mathcal{G}_k$ that spans positively $\mathbb{R}^n$. Generating sets are usually positive bases, with a cardinality of $[n+1, 2n]$. A generating set guarantees the existence of a vector $c \in \mathcal{G}_k$ such that $-\nabla f(x)^T c > 0$. Therefore, $\mathcal{G}_k$ contains at least one descent direction of f.

Given $\mathcal{G} = \{d^{(1)}, \ldots, d^{(p)}\}$ with $p \geq n+1$ and $d^{(i)} \in \mathbb{R}^n$, the function f is evaluated at a set of trial points $\mathcal{P}_k = \{x_k + \triangle_k d : d \in \mathcal{G}_k\}$, where $\triangle_k$ is the step length. An iteration is successful if $f(y) < f(x_k) - \rho(\triangle_k)$ where $y = \arg\min_{x \in \mathcal{P}_k} f(x)$ and ρ is a forcing function. Successful iterations update the iterate x_{k+1} to y and possibly increase the step length $\triangle_{k+1}$ to $\phi_k \triangle_k$, with $\phi_k \geq 1$. Unsuccessful iterations maintain the same iterate, *i.e.*, $x_{k+1} = x_k$, and reduce the step length $\triangle_{k+1}$ to $\theta_k \triangle_k$, with $0 < \theta_k < 1$. The forcing function ρ is included in order to impose a sufficient decrease condition and is required to be a continuous decreasing function that converges to 0 when its argument converges to 0.

Lewis and Torczon [86] extended pattern search methods to bound-constrained problems by including axis directions in the set of poll directions. Lewis and Torczon [87] further extended pattern search methods to linearly constrained problems by forcing the generating set to span a tangent cone $\mathcal{T}_\Omega(x_k, \epsilon)$, which restricts the tangent vectors to satisfy all constraints within an ϵ-neighborhood from x_k.

Under mild conditions, [132] showed convergence of GSS methods to stationary points, which, in theory, could be local minima, local maxima, or even saddle points.

Mesh adaptive direct search (MADS) methods

The MADS methods (Audet and Dennis [11]) modified the poll step of GPS algorithms to consider infinitely many different directions. MADS generates the poll points using two parameters: a poll size parameter, which restricts the region from which points can be selected, and a MESH size parameter, which defines a grid inside the region limited by the poll size parameter. MADS incorporates dynamic ordering, giving precedence to previously successful poll directions. Audet and Dennis [11] went further to propose random generation of poll directions for each iteration.

Abramson and Audet [6] showed convergence of the MADS method to second-order stationary points under the assumption that f is continuously differentiable with Lipschitz derivatives near the limit point. Under additional assumptions (f twice strictly differentiable near the limit point), MADS is shown to converge to a local minimizer with probability 1. A number of problems for which GPS stagnates and MADS converges to an optimal solution are presented in [11]. Some examples are presented in [6] that show how GPS methods stall at saddle points, while MADS escapes and arrives at a local minimum with probability 1.

Augmented Lagrangian pattern search methods

Lewis and Torczon [88] proposed to adapt the augmented Lagrangian algorithm of Conn, Gould and Toint [28], which involves solving a sequence of bound-constrained minimization augmented Lagrangian subproblems. The inner iteration in the algorithm of [88] solves the bound-constrained problem via pattern search algorithms [86, 87]. The stopping criterion depends on the size of the pattern step and the penalty parameters. The parameters are updated after each iteration. The Lagrange multipliers are updated using the Hestenes-Powell multiplier update rule.

Filter pattern search methods

Audet and Dennis [10] proposed a pattern search filter method that considers the optimization of a composite function that combines the true objective function and a constraint violation function constructed using the l_2-norm of the violations of the constraints. A point is classified as a filtered point if there exists a previously evaluated point with better composite function value. The algorithm keeps a list of points not filtered (unfiltered). The poll center can be chosen among the best feasible or the least infeasible unfiltered points. Audet and Dennis [10] provide examples for which the algorithm finds an optimal solution, whereas a barrier pattern search method stalls at a non-stationary point. Convergence is proved but, in practical cases, the algorithm may still stall because not all directions may be tried. Abramson [3] and Abramson *et al.* [8] extended the filter algorithm to mixed-variable problems that may include categorical variables.

Pattern search methods using simplex gradients

Custódio and Vicente [37] proposed to enhance the poll step by giving preference to directions that are closest to the simplex gradient. Simplex gradients are an approximation to the real gradient and are calculated out of a simplex composed of previously evaluated points.

2.3 Algorithms based on surrogate models

The availability of a high-fidelity surrogate model permits one to exploit its underlying properties to guide the search in a intelligent way. Properties such as the gradient and higher order derivative information, as well as the probability distribution function of the surrogate model are used. Since a high-fidelity surrogate model is typically unavailable for a given problem, these methods start by sampling the search space

and building an initial surrogate model. The methods then proceed iteratively to optimize the surrogate model, evaluate the solution point, and update the surrogate model.

2.3.1 Quadratic models and trust-region methods

Trust-region methods use a surrogate model that is usually smooth, easy to evaluate, and presumed to be accurate in a neighborhood (trust-region) about the current iterate. Conn *et al.* [29, 30] proposed to use a quadratic model of the form:

$$q_k(x_k + s) = f(x_k) + \langle \hat{g}_k, s \rangle + \frac{1}{2}\langle s, H_k s \rangle$$

where at iteration k, x_k is the current iterate, $\langle \cdot, \cdot \rangle$ is the inner product, $\hat{g}_k \in \mathbb{R}^n$, and H_k is a symmetric matrix of dimension n. Rather than using derivative information, g_k and H_k are estimated by requiring q_k to interpolate a set Y of sample points: $q_k(x^{(i)}) = f(x^{(i)})$ for $i = 1, \ldots, p$. Unless conditions are imposed on the elements of g_k and H_k, at least $(n+1)(n+2)/2$ points are needed to determine g_k and H_k uniquely. Let x^* denote a minimizer of q_k within the trust region and define the ratio $\rho_k = (f(x_k) - f(x^*)) / (q_k(x_k) - q_k(x^*))$. If ρ_k is greater than a user-defined threshold, x^* replaces a point in Y and the trust region is increased. Otherwise, if the geometry of the set Y is adequate, the trust-region radius is reduced. The algorithm terminates when the trust-region radius drops below a given tolerance.

Powell [114] proposed an algorithm that uses a quadratic model relying on fewer interpolation points in way that minimizes the Frobenius norm of the change to the Hessian of the surrogate model.

2.3.2 Response surface methods (RSMs)

These methods approximate an unknown function f by a response surface (or meta-model) $\hat{f}$ [15]. Any mismatch between f and $\hat{f}$ is assumed to be caused by model error and not because of noise in experimental measurements.

Response surfaces may be non-interpolating or interpolating [71]. The former are obtained by minimizing the sum of square deviations between f and $\hat{f}$ at a number of points, where measurements of f have been obtained. The latter produce functions that pass through the sampled responses. A common choice for non-interpolating surfaces are low-order polynomials, the parameters of which are estimated by least squares regression on experimental designs. However, the ability of polynomials to model complex responses has been questioned [65]. Interpolating surfaces are preferred over polynomials [71] and so are splines [70, 84] because they result in higher accuracy. Independent of the functions used, the quality of the predictor depends on selecting an appropriate sampling technique [13].

The interpolating predictor of a new point x^* is of the form:

$$\hat{f}(x^*) = \sum_{i=1}^{m} \alpha_i f_i(x^*) + \sum_{i=1}^{p} \beta_i \varphi\left(x^* - x^{(i)}\right),$$

where f_i are polynomial functions, α_i and β_i are unknown coefficients to be estimated, φ is a basis function, and $x^{(i)} \in \mathbb{R}^n$, $i = 1, \ldots, p$, are sample points. Basis functions include linear, cubic, thin plate splines, multiquadratic, and kriging.

Originally used for mining exploration models, kriging models a deterministic response as the realization of a stochastic process by means of a kriging basis function. Let μ be the mean of a stochastic process and φ_K be a kriging basis function. A random variable Y is assumed to model the realization of the objective function $f(x)$.

The metamodel, $\hat{f}$, and the correlation function, Corr, are defined as follows:

$$\hat{f}(x^*) = \mu + \sum_{i=1}^{p} \varphi_K \left(x^* - x^{(i)}\right), \tag{2.1}$$

$$\text{Corr}\left[Y(x^{(i)}), Y(x^{(j)})\right] = \exp\left[\sum_{h=1}^{n} \theta_h \left|x_h^{(i)} - x_h^{(j)}\right|^{p_h}\right], \tag{2.2}$$

$$\theta_h \geq 0, \quad p_h \in [0,2], \qquad h = 1, \ldots, n. \tag{2.3}$$

The correlation is a function of the distance between points and has the property of assigning high correlation (Corr ≈ 1) for nearby points and low correlation (Corr ≈ 0) for distant points. The weights θ_h and p_h account for the importance and the smoothness of the corresponding variables. Model (2.1)–(2.3) is often referred to as a *Design and Analysis of Computer Experiments* (DACE) stochastic model [122]. Its parameters are estimated to maximize the likelihood of the observed sample points.

Efficient global optimization (EGO)

The EGO algorithm [73,126] starts by performing a space-filling experimental design. Maximum likelihood estimators for the DACE model are calculated and the model is then tested for consistency and accuracy. A branch-and-bound algorithm is used to optimize the expected improvement, $E[I(x)]$, at the point x. This expected improvement is defined as: $\text{E}[I(x)] = \text{E}\left[\max(f_{\min} - Y(x), 0)\right]$, where $f_{\min}$ is the best objective value known and Y is assumed to follow a normal distribution with mean and standard deviation equal to the DACE predicted values. Although the expected improvement function can be reduced to a closed-form expression [73], it can be highly multimodal.

Radial basis functions

Radial basis functions approximate f by considering an interpolating model based on radial functions $\varphi\left(\|x^* - x^{(i)}\|\right)$. Powell [113] extended radial basis functions for

derivative-free optimization.

Given a set of sample points, Gutmann [55] proposed to find a new point x^* such that the updated interpolant predictor $\hat{f}$ satisfies $\hat{f}(x^*) = T$ for a target value T. Assuming that smooth functions are more likely than "bumpy" functions, x^* is chosen to minimize a measure of "bumpiness" of $\hat{f}$. This approach is similar to maximizing the probability of improvement [71], where x^* is chosen to maximize the probability: Prob $= \Phi\left[(T - Y(x^*))/s(x^*)\right]$, where Φ is the normal cumulative distribution function and $s(x^*)$ is the standard deviation predictor.

Various strategies have been proposed and analyzed [63, 106, 119]. Radial basis functions methods have been extended to constrained optimization [118]. Other implementations include the recently proposed solver ORBIT [137].

Sequential design for optimization (SDO)

The SDO algorithm [35] considers the minimization of the statistical lower bound of the function $\widehat{f}(x^*) - \tau s(x^*)$ for some $\tau \geq 0$.

2.3.3 Surrogate management framework (SMF)

Booker *et al.* [24] proposed a pattern search method that utilizes a surrogate model. SMF involves a search step that uses points generated by the surrogate model in order to produce potentially optimal points as well as improve the accuracy of the surrogate model. The search step alternates between evaluating candidate solution points and calibrating the surrogate model until no further improvement occurs, at which point the algorithm switches to the poll step.

2.3.4 Lipschitzian-based methods

Lipschitzian-based methods construct and optimize a function that underestimates the original one. By constructing this underestimator in a piecewise fashion, these

methods provide possibilities for the global, as opposed to only local, optimization of the original problem. Let $L > 0$ denote a Lipschitz constant of f. Then $|f(a)-f(b)| \leq L \parallel a - b \parallel$ for all a, b in the domain of f. Assuming L is known, Shubert [127] proposed an algorithm for bound-constrained problems. This algorithm evaluates the extreme points of the feasible region and constructs linear underestimators by means of the Lipschitz constant. The algorithm then proceeds to evaluate the minimum point of the underestimator and construct a piecewise underestimator by partitioning the search space.

A straightforward implementation of Shubert's algorithm for derivative-free optimization problems has two major drawbacks: the Lipschitz constant is unknown and the number of function evaluations increases exponentially, as the number of extreme points of an n-dimensional hypercube is 2^n. To overcome these challenges, several approaches have been suggested.

The DIRECT algorithm

Jones *et al.* [72] proposed the DIRECT algorithm (DIvide a hyperRECTangle), with two main ideas to extend Shubert's algorithm to derivative-free optimization problems. First, function values are computed only at the center of an interval, instead of all extreme points. By subdividing intervals into thirds, one of the resulting partition elements inherits the center of the initial interval, where the objective function value is already known. The second main idea of the DIRECT algorithm is to select from amongst current hyperrectangles one that (a) has the lowest objective function value for intervals of similar size and (b) for which the rate of change of the objective promises significant decrease in the current objective function value. The amount of potential decrease in the current objective function value represents a setable parameter in this algorithm and can be used to balance local and global search; larger values ensure that the algorithm is not local in its orientation.

In the absence of a Lipschitz constant, the DIRECT algorithm terminates once the number of iterations reaches a predetermined limit. Under mild conditions, Finkel and Kelley [48] proved that the sequence of best points generated by the algorithm converges to a KKT point. Convergence was also established for general constrained problems, using a barrier approach.

Branch-and-bound (BB) search

BB sequentially partitions the search space, and determines lower and upper bounds for the optimum. Partitions that are inferior or infeasible are fathomed in the course of the search. Let $\Omega = [x_l, x_u]$ be the region of interest and let $x^*_\Omega \in \Omega$ be a global minimizer of f in Ω. The availability of a Lipschitz constant L along with a set of sample points $\Lambda = \{x^{(i)}, i = 1, \ldots, p\} \subset \Omega$ provides lower and upper bounds:

$$\max_{i=1,\ldots,p} \{f(x^{(i)}) - L\delta_i\} = \underline{f}_\Omega \leq f(x^*_\Omega) \leq \overline{f}_\Omega = \min_{i=1,\ldots,p} f(x^{(i)}),$$

where δ_i is a function of the distance of $x^{(i)}$ to the vertices of $[x_l, x_u]$. The Lipschitz constant L is unknown but a lower bound $\underline{L}$ can be estimated from the sampled objective function values:

$$\underline{L} = \max_{i,j} \frac{|f(x^{(i)}) - f(x^{(j)})|}{||x^{(i)} - x^{(j)}||} \leq L, \qquad i, j = 1, \ldots, p \quad , i \neq j.$$

Due to the difficulty of obtaining deterministic bounds, statistical bounds relying on extreme order statistics were proposed in [110]. This approach assumes samples are randomly generated from Ω and their corresponding objective values are random variables. Subset-specific estimates of L can be significantly smaller than the global L, leading to faster convergence of BB.

2.3.5 Multilevel coordinate search (MCS)

MCS [66] splits the search space into boxes with an evaluated *base point*. Unlike the DIRECT algorithm, MCS allows base points anywhere in the corresponding boxes. Boxes are divided with respect to a single coordinate. The global-local search is balanced by a multilevel approach, according to which each box is assigned a *level* s proportional to the number of times the box has been processed. Boxes with level $s = s_{\max}$ are considered too small to be further split.

At each iteration, MCS selects boxes with the lowest objective value for each level value and marks them as candidates for splitting. Let n_j the number of splits in coordinate j. If $s > 2n(\min n_j + 1)$, the box is considered for *splitting by rank*, which prevents having unexplored coordinates for boxes with high s values; in this case, the splitting index k is chosen such that $n_k = \min n_j$. Otherwise, the box is considered for *splitting by expected gain*, which selects the splitting index and coordinate value by optimizing a local separable quadratic model using previously evaluated points. MCS with local search performs local searches from boxes with level $s_{\max}$, provided that the corresponding base points are not near previously investigated points. Huyer and Neumaier [66] showed that, as $s_{\max}$ approaches to infinity, the base points of MCS form a dense subset of the search space and MCS converges to a global minimum.

2.3.6 Optimization by branch-and-fit

Huyer and Neumaier [67] proposed a hybrid optimization algorithm that combines ideas from surrogate models and stochastic methods. Candidate points are generated using different criteria. Points of Class 1 are obtained from optimizing a local quadratic model around the best current point. The model considers an error term and is determined by changing linear and quadratic terms that minimize the sum of squares of the errors. Points of Classes 2 and 3 are obtained by minimizing a linear model with an error term. Models are constructed around all the other points, mak-

ing the model over-determined. Points for both models are chosen from the nearest neighbors to each point. Points of Class 2 are obtained from local points, where the objective at the center point is significantly less than that of their neighbors. Points of Class 3 are obtained from non-local points. Points of Class 4 seeks to evaluate unexplored areas by analyzing the number of times a given area has been partitioned. Points of Class 5 are randomly selected points in certain areas that will later be used to determine local models. The algorithm confines the search to multiples of a user-provided resolution vector. The authors proposed a merit function to solve general constrained problems.

2.3.7 Implicit filtering

Implicit filtering [51, 138] uses an approximation of the gradient to guide the search, resembling the steepest descent method when the gradient is known. The approximation of the gradient at an iterate is typically based on forward or centered differences. Forward differences require n function evaluations, whereas centered difference gradients require $2n$ function evaluations over the set $\Gamma = \{x \pm se_i : i = 1, \ldots, n\}$, where x is the current iterate, s is the *scale* of the stencil, and e_i are coordinate unit vectors. As these points are distributed around the iterate, they produce approximations less sensitive to noise than forward differences [51]. A line search is then performed along the direction of the approximate gradient. The candidate point is required to satisfy a minimum decrease condition of the form $f(x - \delta\nabla_s f(x)) - f(x) < -\alpha\delta\|\nabla_s f(x)\|^2$, where δ is the step and α is a parameter. The algorithm continues until no point satisfies the minimum decrease condition, after which the scale s is decreased. The implicit filtering algorithm terminates when the approximated gradient is less than a certain tolerance proportional to s.

2.4 Stochastic search algorithms

This section presents approaches that require non-deterministic algorithmic steps. These algorithms aim to explore the design space and locate a good solution while occasionally allowing intermediate moves to lesser quality points than the solution currently at hand. The literature on these algorithms is very extensive, especially on the applications side, since their implementation is rather straightforward. Many of these methods converge asymptotically to a global solution but their finite-time average performance is largely unknown.

2.4.1 Simulated annealing

Initially proposed to handle combinatorial optimization problems [78], the algorithm was later extended to continuous problems [17]. At iteration k, simulated annealing generates a new trial point $\hat{x}$ that is compared to the incumbent x^k and accepted with a probability function [97]:

$$\mathrm{P}(\hat{x}|x_k) = \begin{cases} \exp[-\frac{f(\hat{x})-f(x_k)}{T_k}] & \text{if } f(\hat{x}) > f(x_k) \\ 1 & \text{if } f(\hat{x}) \leq f(x_k). \end{cases}$$

As a result, simulated annealing allows moves to points with objective function values worse than the incumbent. The probability P depends on the "temperature" parameter T_k; the sequence $\{T_k\}$ is referred as the cooling schedule. Cooling schedules are decreasing sequences that converge to 0 sufficiently slow to permit the algorithm to escape from local optima.

Asymptotic convergence results to a global optimum have been presented [1] but there is no guarantee that a good solution will be obtained in a finite number of iterations [121]. Finite-time performance is of particular interest for problems with expensive function evaluations [25, 108].

2.4.2 Genetic algorithms

Genetic algorithms, introduced by Holand [60], resemble natural selection and reproduction processes governed by rules that assure the survival of the fittest in large populations. Individuals (points) are associated with identity genes that define a fitness measure (objective function value). A set of individuals form a population, which adapts and mutates following probabilistic rules. Given a population P of size M, a *gene pool* GP of size M is formed by probabilistically selecting individuals from P using a function of their fitness, possibly allowing duplication of individuals. The new population NP is obtained by randomly accepting some individuals from GP and combining the others by a crossover operation. Typical crossover operations define a crossover point on the gene structure of the two parents and combine the fragments. Finally, random individuals from NP are selected for a mutation operation that involves modifications to random sections of their genes.

Bethke [19] extended genetic algorithms to continuous problems by representing continuous variables by an approximate binary decomposition. Liepins and Hilliard [89] suggested that population sizes should be between 50 and 100 to prevent failures due to bias by the highest fitness individuals.

Particle swarm algorithms

Particle swarm optimization is a population-based algorithm introduced by Kennedy and Eberhart [42, 77] that maintains at each iteration a swarm of particles (set of points) with a velocity vector associated with each particle. A new set of particles is produced from the previous swarm using rules that take into account particle swarm parameters (inertia, cognition, and social) and randomly generated weights. Particle swarm optimization has enjoyed recent interest resulting in hybrid algorithms that combine the global scope of the particle swarm search with the faster local convergence of the Nelder-Mead simplex algorithm [47] or GSS methods [135].

2.4.3 Hit-and-run algorithms

Proposed independently by Boneh and Golan [23] and Smith [128], each iteration of hit-and-run algorithms compares the current point x with a randomly generated candidate. The generation of candidates is based on two random components. A direction d is generated using a uniform distribution over the unit sphere. For the given d, a step s is generated using a uniform distribution over the set of steps S in a way that $x+ds$ is feasible. Bélisle *et al.* [17] generalized hit-and-run algorithms by allowing arbitrary distributions to generate both the direction d and step s, and proved convergence to a global optimum under mild conditions for continuous optimization problems.

2.5 Historical overview and current status

A timeline in the history of innovation in the context of derivative-free algorithms is provided in Figure 2.1. In addition, the number of citations is provided in Table 2.1 for works that have received over 1000 citations each. The Hooke and Jeeves and Nelder-Mead simplex algorithms were the dominant approaches in the 1960s and 1970s, and continue to be popular. Stochastic algorithms were introduced in the 1980s and have been the most cited. There was relatively little theory behind the deterministic algorithms until the 1990s. Over the last two decades, the emphasis in derivative-free optimization has shifted towards the theoretical understanding of existing algorithms as well as the development of approaches based on surrogate models. The understanding that management of the geometry of surrogate models has a considerable impact on the performance of the underlying algorithms led to the development of several new competing techniques. The algorithmic and theoretical sides of derivative-free optimization have been extensively developed over the past two decades, with the most important results being the recent model-based algorithms

Table 2.1: Citations of most cited works in derivative-free algorithms.

Publication	Year appeared	Citations [1]
Hooke and Jeeves [64]	1961	1243
Nelder and Mead [101]	1965	6557
Holland [60]	1975	2083
Kirkpatrick *et al.* [78]	1983	12495
Eberhart and Kennedy [42, 77]	1995	6560

[1] From Google Scholar on 26 December 2008

as well as proofs of global convergence for direct and model-based approaches. The status of software implementations is investigated in the following two sections.

2.6 Derivative-free optimization software

The software considered here have been under development and/or released since 1998. They are all capable of dealing with black-box functions in a non-intrusive way, *i.e.*, the source code of the optimization problem to be solved is assumed to be unavailable or impractical to modify. Table 2.2 presents a list of the solvers and their main characteristics. Each solver is discussed in detail in the sequel.

2.6.1 ASA

Adaptive Simulated Annealing–ASA (version 26.30) [69] is a C implementation developed for unconstrained optimization problems. ASA departs from traditional simulated annealing by considering a generating probability density function with fatter tails than the typical Boltzmann distribution, allowing ASA to possibly escape local minima by considering farther points. Separate temperature parameters are assigned for each variable and they are updated as the optimization progresses.

The main inputs to ASA are the objective function in the form of a C routine, a

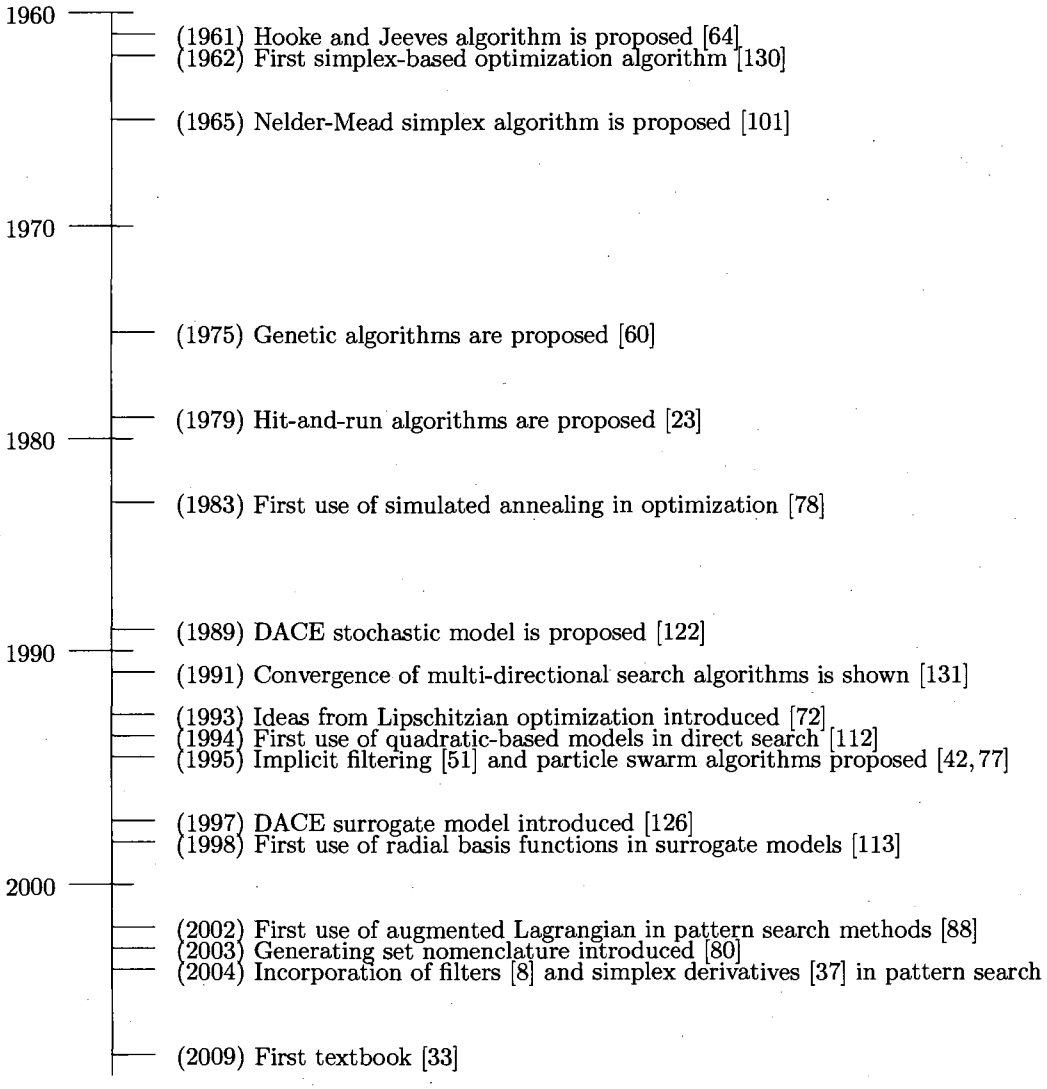

Figure 2.1: Timeline of innovation in derivative-free optimization.

Table 2.2: Available solvers for derivative-free problems.

Solver	URL	Version	Language	Bounds	Linear constraints	Black-box constraints
ASA	www.ingber.com	26.30	C	required	no	no
DAKOTA/APPS	software.sandia.gov/appspack/	4.1	C++	required	yes	yes
DAKOTA/DIRECT	www.cs.sandia.gov/dakota/	4.1	C++	required	yes	yes
DAKOTA/EA	www.cs.sandia.gov/dakota/	4.1	C++	required	yes	yes
DAKOTA/PATTERN	www.cs.sandia.gov/dakota/	4.1	C++	required	yes	yes
DAKOTA/SOLIS-WETS	www.cs.sandia.gov/dakota/	4.1	C++	required	yes	yes
DFO	projects.coin-or.org/Dfo	2.0	Fortran	required	yes	yes
FMINSEARCH	www.mathworks.com	N/A	Matlab	no	no	no
IMFIL	www4.ncsu.edu/~ctk/imfil.html	0.7	Matlab	required	no	no
MCS	www.mat.univie.ac.at/~neum/software/mcs/	2.0	Matlab	required	no	no
NEWUOA	N/A	N/A	Fortran	no	no	no
NOMAD	www.gerad.ca/NOMAD/	2.0	C++	required	no	yes
PSWARM	www.norg.uminho.pt/aivaz/pswarm/	0.1	C	required	no	no
SID-PSM	www.mat.uc.pt/sid-psm/	1.0	Matlab	yes[2]	yes[2]	yes[2]
SNOBFIT	www.mat.univie.ac.at/~neum/software/snobfit/	2.1	Matlab	required	no	no
TOMLAB/EGO	tomopt.com	N/A	Matlab	required	yes	yes
TOMLAB/GLB	tomopt.com	N/A	Matlab	required	no	no
TOMLAB/GLC	tomopt.com	N/A	Matlab	required	yes	yes
TOMLAB/LGO	www.pinterconsulting.com/	N/A	Matlab	required	yes	yes
TOMLAB/RBF	tomopt.com	N/A	Matlab	required	yes	yes

[2] Gradients of constraints at evaluated points are required but trivially reduce to unit vectors for bound-constrained problems.

starting point, the number of variables and bounds, and a limit on the number of function evaluations.

2.6.2 DAKOTA solvers

Design Analysis Kit for Optimization and Terascale Applications (DAKOTA) [43] is a project at Sandia National Laboratories. DAKOTA's initial scope was to create a toolkit of black-box optimization methods. The scope was later expanded to include additional optimization methods and other engineering applications, including design of experiments, parameter studies, and nonlinear least squares.

DAKOTA contains a collection of optimization software packages featuring the COLINY library [58] that includes, among others, the following solvers:

1. DAKOTA/EA: various evolutionary algorithms,
2. DAKOTA/DIRECT: an implementation of the DIRECT algorithm,
3. DAKOTA/PATTERN: various pattern search methods,
4. DAKOTA/SOLIS-WETS: greedy search, comparing the incumbent with points generated from a multivariate normal distribution, and
5. DAKOTA/APPS: asynchronous parallel pattern search.

The main inputs to DAKOTA are the objective and constraints as black-box executable functions, a starting point, the number of variables and bounds, constraint bounds, and a limit on the number of function evaluations.

2.6.3 DFO

Derivative Free Optimization–DFO (version 2.0) [26, 125]–is an open-source Fortran implementation of the trust-region-based algorithm originally developed by Conn *et*

al. [29,30] and expanded by Conn *et al.* [31]. Given a set of points, DFO identifies the point with the best objective found and builds a quadratic model by interpolating a selected subset of points. The resulting model is optimized within a trust-region centered at the best point.

DFO classifies constraints as easy, difficult, and virtual, depending primarily on the complexity of function evaluation and availability of derivative information.

The main inputs to DFO are the objective, constraint functions in the form of Fortran subroutines, starting point(s), the number of variables and bounds, constraint bounds, initial and lower bound for trust-region radius, and limits on the number of function evaluations and iterations. The optional parameters control the maximum trust-region radius, minimum number of points used for the interpolation, and other rules and conditions to accept points and perform the interpolation. Our computational experiments used the open-source Fortran software IPOPT [27] to solve the trust-region subproblems.

2.6.4 FMINSEARCH

FMINSEARCH is an implementation of the Nelder-Mead simplex-based method of Lagarias *et al.* [83]. This code is included as a Matlab built-in function in the Optimization Toolbox and handles unconstrained optimization problems. The main inputs to FMINSEARCH are the objective function in the form of a Matlab function, a starting point, and a limit on the number of function evaluations.

2.6.5 IMFIL

IMFIL (Version 0.7) [76] is a Matlab implementation of the implicit filtering algorithm [51,138]. The main inputs to IMFIL are the objective as a black-box function in the form of a Matlab function, variable bounds, and a limit on the number of function evaluations. The optional parameters control the choice of the difference gradient

stencil (centered, forward, or positive basis), the choice of the Hessian matrix update scheme, and the minimum size of the scale parameter.

2.6.6 MCS

Multilevel coordinate search (MCS) [104] is a Matlab implementation of the algorithm proposed by Huyer and Neumaier [66] for global optimization of bound-constrained problems. The main inputs to MCS are the objective in the form of a Matlab function, variable bounds, and a limit on the number of function evaluations. The optional parameters control the set of initial points evaluated and the local search strategy.

2.6.7 NEWUOA

NEWUOA [115] is a Fortran implementation of Powell's model-based algorithm [114]. The main inputs to NEWUOA are the objective in the form of a Fortran subroutine, a starting point, the number of variables, initial and lower bounds for the trust-region radius, a limit on the number of function evaluations, and the number of function values to be used for interpolating the quadratic model.

2.6.8 NOMAD

NOMAD (Version 2.0) [34] is a C++ implementation of the MADS filter algorithm [3]. It is designed to solve nonlinear, nonsmooth, noisy optimization problems with bounded variables. Optionally, NOMAD can be installed along with the BerkeleyDB database [107] to allow use of a previously evaluated set of points. The main inputs to NOMAD are the objective and constraints executable functions, the number of variables and their bounds, a starting point, and scaling choice and scale parameters. The optional parameters specify the search and poll step characteristics. The termination criterion may involve a combination of the number of points evaluated, minimum mesh size,

and the acceptable number of consecutive trials that fail to improve the incumbent.

A related implementation, NOMADm [4], is a collection of Matlab functions that solve bound-constrained, linear or nonlinear optimization problems, with continuous, discrete, and categorical variables.

2.6.9 PSWARM

PSWARM [134] is a C implementation of the particle swarm pattern search method [135]. Its search step performs global search based on the particle swarm algorithm. Its poll step relies on a coordinate search method. The poll directions coincide with positive and negative unit vectors of all variable axes.

PSWARM allows the user to compile and use the solver as a stand-alone software or as a custom AMPL solver. A related Matlab implementation PSwarmM is also available [134].

2.6.10 SID-PSM

SID-PSM (version 1.0) [38] is a Matlab implementation of a pattern search method with the poll step guided by simplex derivatives [37]. The search step relies on the optimization of linear and quadratic surrogate models. SID-PSM is designed to solve unconstrained and constrained problems. The main inputs to SID-PSM are the objective, constraints, and gradients of constraint functions in the form of Matlab routines, and a feasible starting point. The optional parameters include control over the strategy of poll directions.

2.6.11 SNOBFIT

SNOBFIT (version 2.1) is a Matlab implementation of the branch-and-fit algorithm proposed by Huyer and Neumaier [67]. The main inputs to SNOBFIT are the objective

in the form of a Matlab function, variable bounds, a limit on the number of function evaluations, and the resolution vector described in Section 2.3.6.

2.6.12 TOMLAB solvers

TOMLAB [61] is a Matlab environment that provides access to the following derivative-free optimization solvers:

1. TOMLAB/GLBSOLVE [61, pp.125–127]: an implementation of the DIRECT algorithm [72], specifically designed to handle box-bounded problems;

2. TOMLAB/GLCSOLVE [61, pp.142–146]: an implementation of an extended version of the DIRECT algorithm [72] that can handle integer variables and linear or nonlinear constraints;

3. TOMLAB/EGO [62, pp. 11–24]: an implementation of the EGO algorithm [73,126] modified to handle linear and nonlinear constraints;

4. TOMLAB/LGO [111]: a suite of global and local nonlinear solvers [110] that implements a combination of Lipschitzian-based branch-and-bound with deterministic and stochastic local search; and

5. TOMLAB/RBFSOLVE [62, pp.5–10]: an implementation of the radial basis function [20,55] that can handle box-constrained global optimization problems.

The main inputs to the TOMLAB solvers are the objective in the form of a Matlab function, a starting point, variable bounds, the constraint functions in the form of Matlab functions (supported for some solvers), constraint bounds (supported for some solvers), and limits on the number of function evaluations, iterations, and CPU seconds.

Table 2.3: Characteristics of test problems.

Problem type	Number of variables	Number of problems
Type 1	1 to 2	87
Type 2	3 to 9	97
Type 3	10 to 30	27
Average	5.2	

2.7 Computational experience

2.7.1 Test problems

Test problems were used from the `globallib` [52] and `princetonlib` [116] collections of nonlinear programming problems. As most of the optimization packages tested in this study were designed for low-dimensional unconstrained problems, the problems considered were restricted to a maximum of 30 variables with bound constraints only, resulting in a total of 211 problems. We use the number of variables to classify each problem in one of three problem types as shown in Table 2.3. The test problems are diverse, involving sums of squares problems, quadratic and higher degree polynomials, convex and non-convex functions, continuous and discontinuous functions, 29 problems with trigonometric functions, and 23 problems with exponential or logarithmic functions.

Whereas the problems from `globallib` and `princetonlib` are all smooth, we also used nonsmooth problems for testing purposes. In particular, we used a collection of 47 nonlinear, nonsmooth, unconstrained optimization problems compiled by LukŠan and VlŠek [91].

For all problems, variable bounds are required by many of the solvers but were not available for all test problems. For problems that lacked such bounds in the problem formulation, we restricted all variables to the interval $[-10000, 10000]$ unless these bounds resulted in numerical difficulties due to overflowing. In such cases, tighter bounds were used, provided that they still included the best known solution for each

problem. The same variable bounds were used for all solvers. Whenever starting points were required, they were generated randomly from this box-bounded region. The same randomly generated starting points were used across all solvers.

Despite the capabilities of some solvers to incorporate derivative information to expedite the search, only objective function information was provided to all solvers. The only exception was SID-PSM, which requires the gradient of the constraints. As the problems considered here were bound-constrained with no additional constraints, the gradients provided to SID-PSM were simply a set of unit vectors. Given the trivial nature of these gradients, we decided to include this solver in this study.

2.7.2 Algorithmic settings

Algorithmic parameters for the codes under study should be chosen in a way that is reflective of the relative performance of the software under consideration. Unfortunately, the optimization packages tested have vastly different input parameters that may have a significant impact upon the performance of the algorithm. This presents a major challenge as a computational comparison will have to rely on a few choices of algorithmic parameters for each code. In a recent extensive comparison of global optimization solvers, Neumaier *et al.* [105] faced a similar problem and recognized that the typical user of optimization packages is not an expert on the theory and implementation of the underlying algorithm. Thus, comparisons were carried out in [105] using the default parameter values for each package, along with uniform stopping criteria and starting points across solvers. We adopt the same approach here.

2.7.3 Computational results with smooth problems

The 20 solvers of Table 2.2 were tested on the 211 problems, initially using a limit of 2500 function evaluations. Each solver was run from ten different starting points, with the exception of DAKOTA/DIRECT, MCS, TOMLAB/EGO, TOMLAB/GLB and TOMLAB/GLC

and TOMLAB/RBF. The latter solvers override the selection of a starting point and start from the center of the box-bounded region. This resulted in a total number of 30,806 optimization instances to be solved. All computations were performed on Intel Xeon 1700 Mhz processors running Linux and Matlab 7.3 R2006b.

Most instances were solved within a few minutes. Since the test problems are algebraically and computationally simple and small, the total time required for function evaluations for all runs was negligible. Most of the CPU time was spent by the solver on processing function values and determining the sequence of iterates. A limit of 900 CPU seconds was imposed on each run. In practice, this time limit was enforced only to EGO and RBFSOLVE that demanded significantly more time to run than the other solvers.

In order to assess the quality of the solutions obtained by different solvers, we used the general-purpose global optimization solver BARON [123,124] to solve as many of the test problems as possible. Unlike derivative-free solvers, BARON requires explicit algebraic expressions rather than function values alone. BARON's branch-and-bound strategy was able to guarantee global optimality for most of the 211 problems, although this solver does not accept trigonometric and some other nonlinear functions. For problems for which BARON was unable to provide a global solution, the best known solutions were obtained from Web sites devoted to the test collections [52,102,103,116].

Most test problems are nonconvex and most of the solvers tested are local solvers. For this reason, we ran each solver from ten different starting points. In comparing the quality of solutions returned, we will compare the average as well as best case behavior of each solver. For the average case behavior, we compare solvers using for each solver the median objective function value of the ten different runs. For the best case comparison, we compare the best solution found by each solver after all ten runs.

Figure 2.2 shows how the different solvers progress for camel6, a two-dimensional test problem referred to as the 'six-hump camel back function' that contains six local

minima, two out of which are global minima. Each graph presents the colored surface of the camel6 function, where red represents high values and blue represents low values. Global minima are located at $[-0.08980.7126]$ and $[0.0898 - 0.7126]$ and are marked with magenta circles. Each solver was given a limit of 2500 function evaluations and the points evaluated are marked with red crosses. Solvers that require a starting point were given the same starting point. Starting points are marked with a green circle. The trajectory of the progress of the best point is marked with a cyan line, and the final solution is marked with a yellow circle. Solvers DAKOTA/APPS, DAKOTA/PATTERN, DAKOTA/SOLIS-WETS, FMINSEARCH, and NEWUOA perform a local search, exploring the neighborhood of the starting point and converging to a local minimum far from the global minima. DIRECT-based methods DAKOTA/DIRECT, TOMLAB/GLB, and TOMLAB/GLC perform similar search patterns, concentrating evaluation points around the local minima. Indeed, the two global minima are found by these solvers. Figure 2.3 addresses the average case behavior and presents the fraction of test problems for which the solver median solution was within an absolute tolerance of 10^{-3} from the best known solution. The horizontal axis shows the progress of the algorithm as the number of iterations gradually reached 2500. As shown in this figure, MCS and TOMLAB/LGO attained the highest percentage of global solutions, with MCS leading for up to 500 function evaluations and with TOMLAB/LGO leading when more iterations were allowed. Both solvers found over 55% of the global solutions, followed by a group of solvers that found global solutions in 45 to 48% of the cases. This group included SID-PSM, TOMLAB/GLC, TOMLAB/GLB, and SNOBFIT. PSWARM returned the best results among the stochastic solvers. The solvers ASA and DAKOTA/EA did not provide good solutions, although there was improvement in the solutions they obtained when the limit on the function evaluations was increased. Most solvers find global optima for a relatively small fraction of the problems.

Figure 2.4 addresses the best case behavior and presents the fraction of test prob-

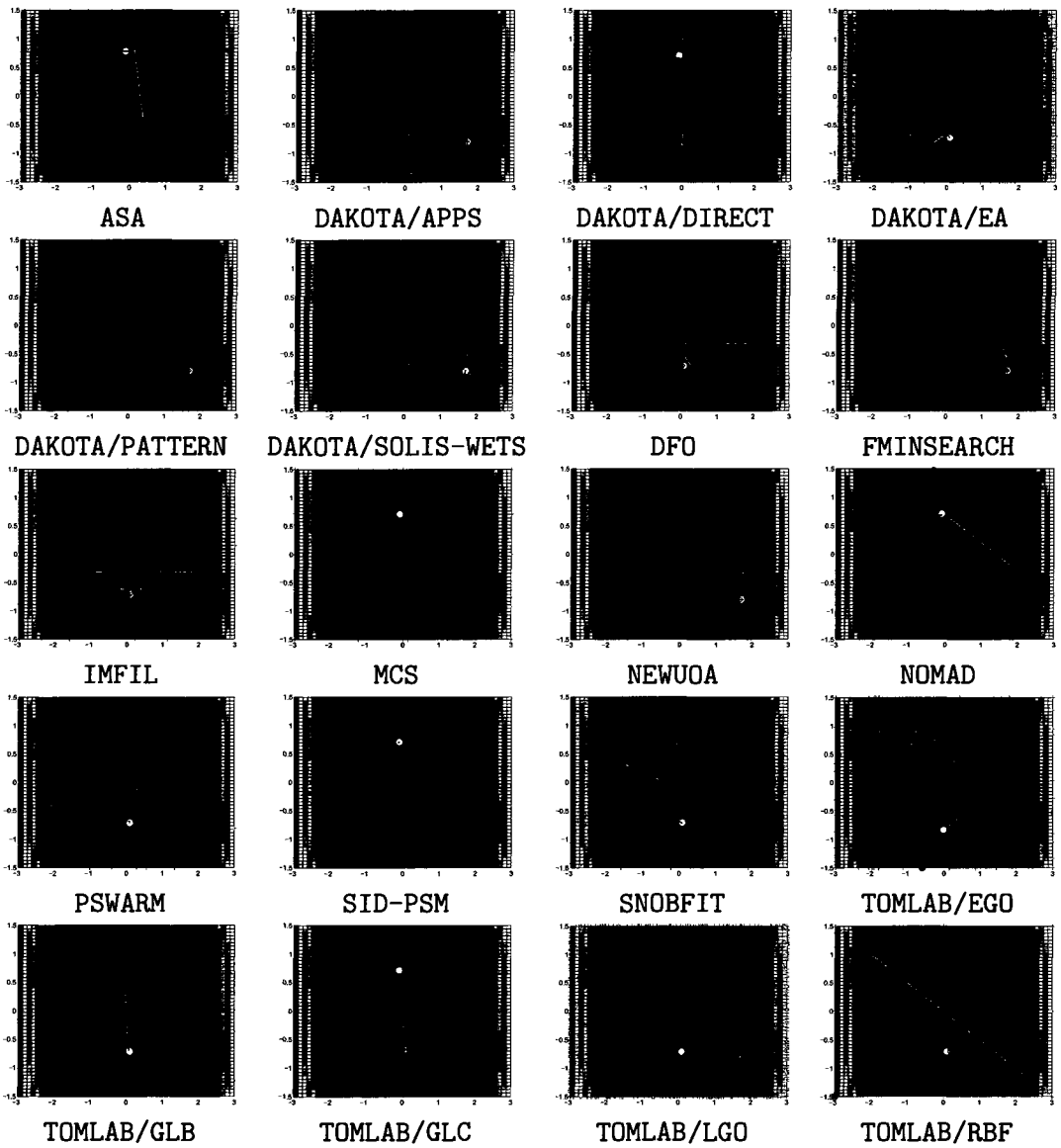

Figure 2.2: Optimization progress for test problem `camel6`.

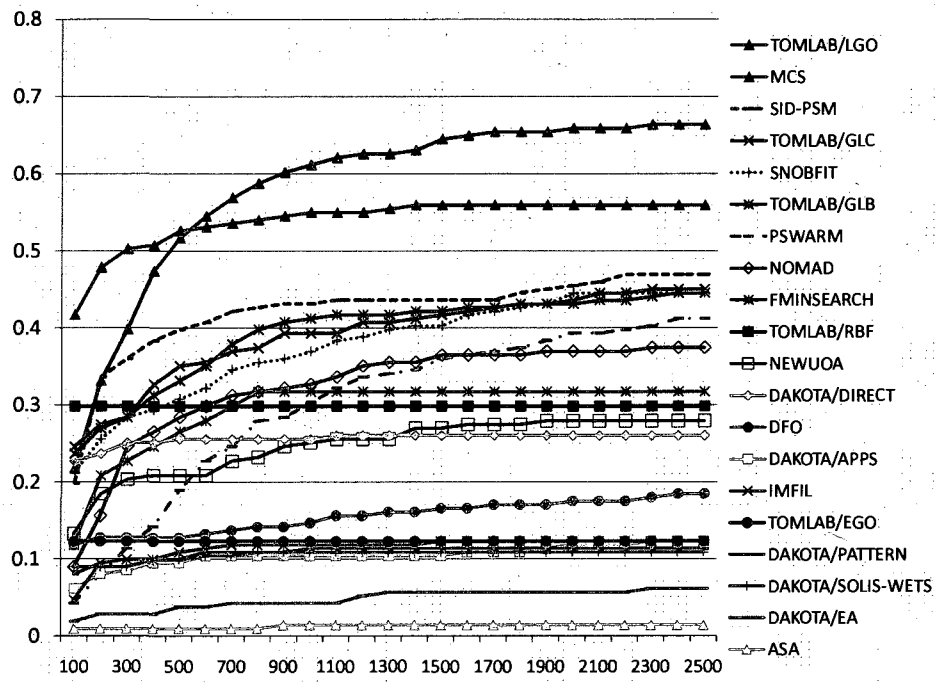

Figure 2.3: Fraction of problems solved to global optimality (average solver performance) as a function of allowable number of function evaluations.

lems for which the best solution returned by a solver was within an absolute tolerance of 10^{-3} from the best known solution. As in the previous figure, the horizontal axis shows the progress of the algorithm as the number of iterations gradually reached 2500. As shown in this figure, `TOMLAB/LGO` achieves the highest number of problems solved, with 71% of the total number of problems, followed by `SID-PSM` and `FMINSEARCH`, both of which surpassed `MCS`. However, note that results for `MCS` correspond to only one run since `MCS` overrides the selection of a starting point. `SNOBFIT` and `NEWUOA` also gained positions and followed `MCS` closely.

A somewhat different point of view is taken in Figure 2.5, where we present the fraction of problems for which each solver achieved a solution as good as the best solution amongst all solvers, without regard to the best known solution for the problems. As before, the horizontal axis denotes the number of allowable function evaluations. With the exception of solvers `ASA`, `DAKOTA/EA`, and `DAKOTA/SOLIS-WETS`,

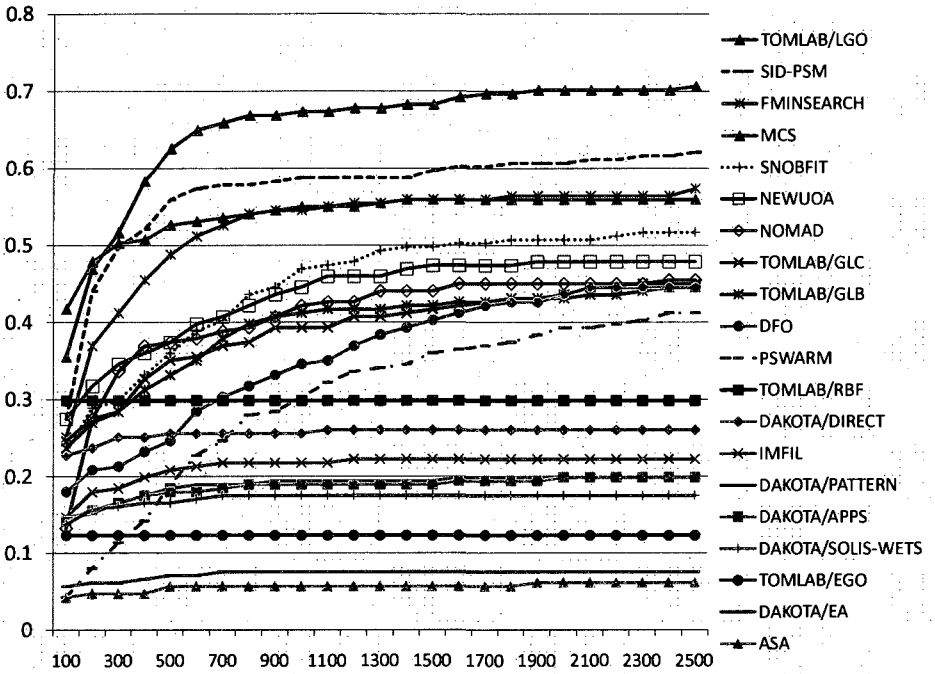

Figure 2.4: Fraction of problems solved to global optimality at least once by each solver, as a function of allowable number of function evaluations.

all solvers are found to be the best at least for one problem, showing that there is no strong dominance of any method over the rest. Strong performance is, once again, observed for `TOMLAB/LGO` and `MCS`, both of which achieved the best results over 55% of the problems considered. As before, `MCS` performs better for fewer than 500 evaluations, while `TOMLAB/LGO` performs better for over 500 function evaluations.

An alternative benchmark, proposed by Moré and Wild [99], measures each algorithm's ability to improve a starting point. For a given $0 \leq \tau \leq 1$, a solver is considered to have successfully improved a starting point if

$$f(x_0) - f_{solver} \geq (1 - \tau)(f(x_0) - f_L),$$

where $f(x_0)$ is the starting objective value, f_{solver} is the solution reported by the solver, and f_L is the best f_{solver} among all solvers. Since the global solution is known,

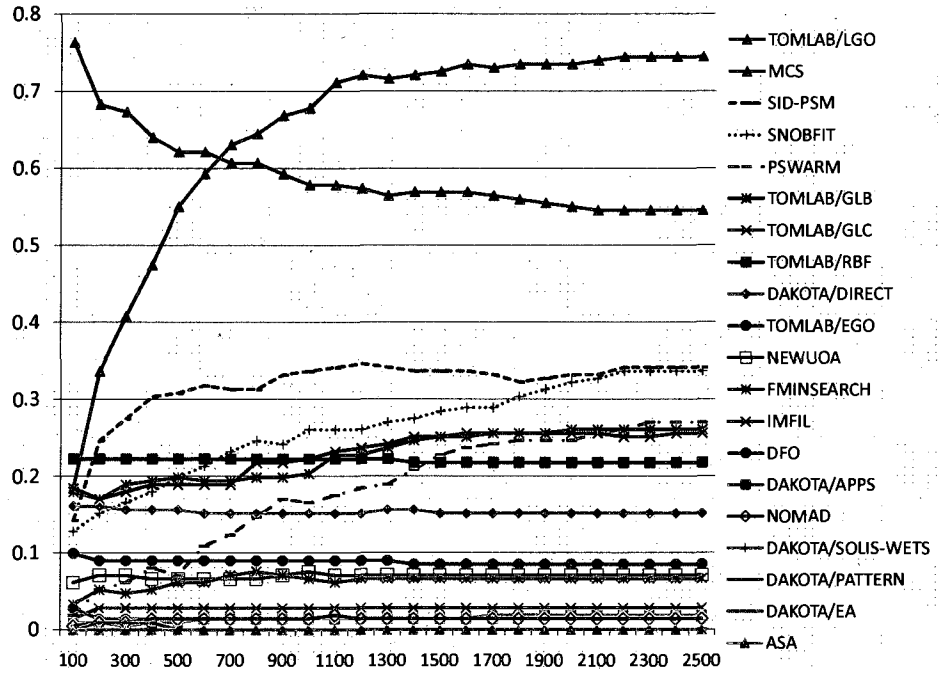

Figure 2.5: Fraction of problems, as a function of allowable number of iterations, for which a solver found the best solution amongst all solvers tested.

we used it in place of f_L in evaluating this measure. We used this measure to evaluate the average case performance of each solver, *i.e.*, a problem was considered 'solved' by a solver if five or more of the ten runs from different starting points improved the starting point by at least a fraction of $(1-\tau)$ of the largest possible reduction. Figure 2.6 presents the fraction of problems for which the starting point was improved by a solver as a function of the τ values. Solvers `MCS` and `TOMLAB/LGO` are found to solve more than 90% of the problems for τ values as low as 10^{-6}. The ranking is similar to that displayed in Figure 2.3. At a first look, it appears pretty remarkable that a large number of solvers can improve the starting point of 90% of the problems by 90%. Looking more closely at the specific problems at hand, reveals that many of them involve polynomials and exponential terms. As a result, with a bad starting point, the objective function value is easy to improve by 90%, even though the final solution is still far from being optimal.

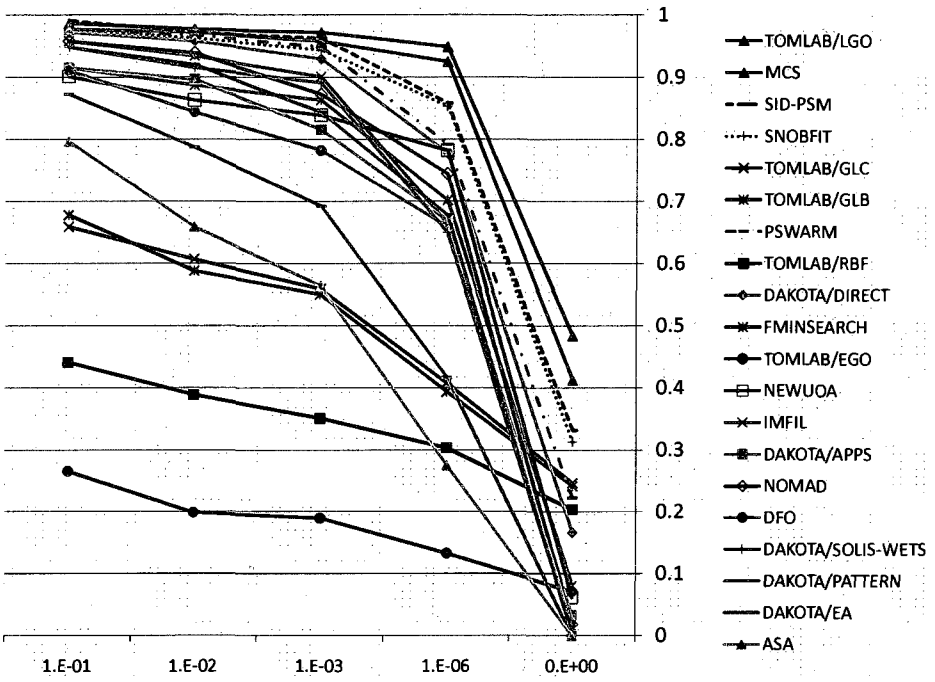

Figure 2.6: Fraction of problems for which starting points were improved vs. τ values for 2500 function evaluations.

Finally, we use the same performance measure to study the effect of the problem size on each solver in Figure 2.7, where the left, middle, and right columns of graphs correspond to problems of Type 1, Type 2, and Type 3, respectively. Many solvers have the same or better performance for Type 2 problems compared to Type 1 problems. In general, as the size of the problem increases, the chances of obtaining better solutions decrease. A notable degrading performance is displayed by `FMINSEARCH`, going from top-ranked to mid-ranked as we move from Type 1 to Type 3. Overall, Solvers `MCS` and `TOMLAB/LGO` are found to be top-ranked among all three types of problems.

Given that no single solver seems to dominate over all others, the last question addressed is whether there exists a minimal cardinality subset of the solvers capable of collectively solving all problems or a certain fraction of all problems. In this case, a problem will be considered solved if any of the chosen solvers succeeds in

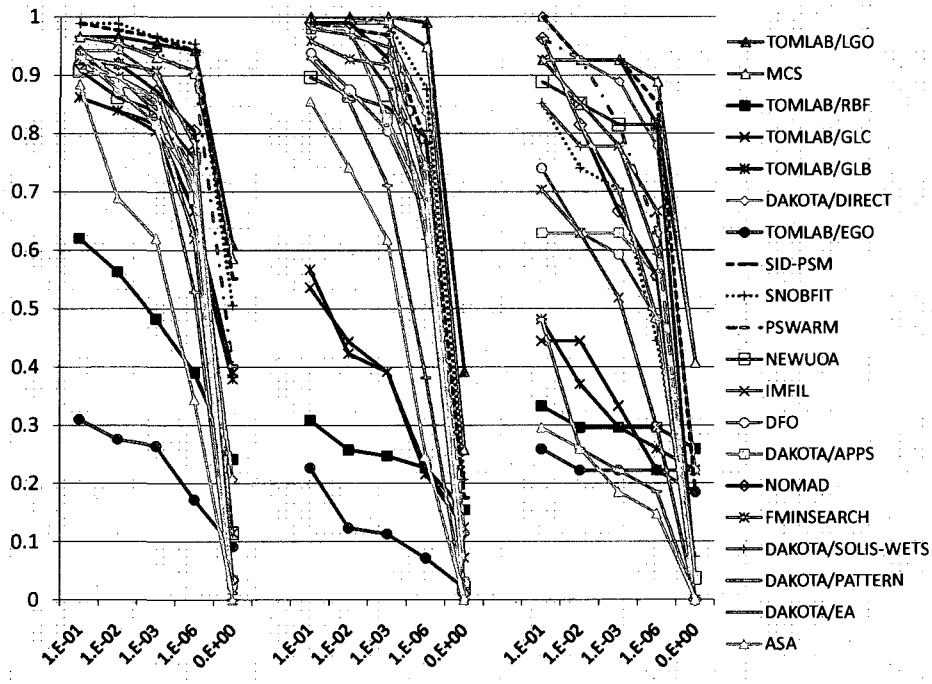

Figure 2.7: Fraction of problems for which starting points were improved vs. τ values for 2500 function evaluations for problems of Type 1 (left), Type 2 (middle), and Type 3 (right).

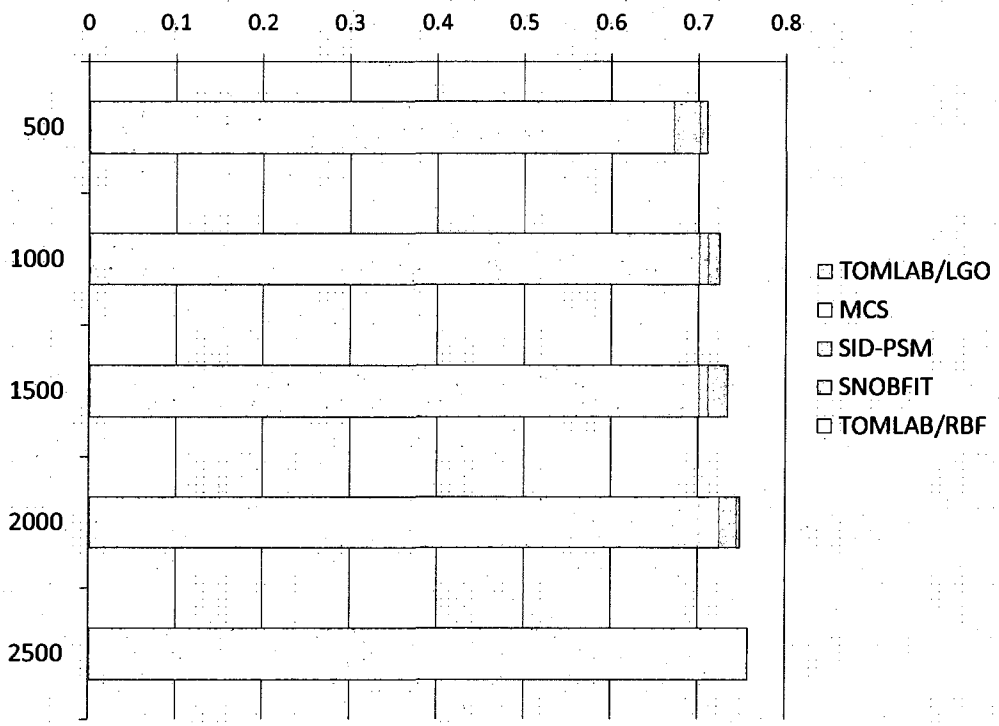

Figure 2.8: Minimum number of solvers required to solve test problems for various limits of function evaluations.

solving the problem with an absolute tolerance of 10^{-3} during any one of the ten runs from randomly generated starting points. LGO is the solver that is able to solve most problems alone. LGO and MCS is the combination of two solvers that solves the most number of problems compared to any other two-solver combination. Similarly, SID-PSM, SNOBFIT and TOMLAB/RBF are the next solvers to be added to a minimal set of solvers with maximum impact in terms of solving ability of the collection of solvers. The results are shown in Figure 2.8. For different numbers of function evaluations (vertical axis), the bars show the fraction of problems (horizontal axis) that are solved by the minimal solver subset. Remarkably, for 2500 function evaluations, LGO alone can solve the same number of problems that the combined 20 solvers are able to solve. There is no problem in the test set for which LGO was not able to find a global solution and other solvers were able.

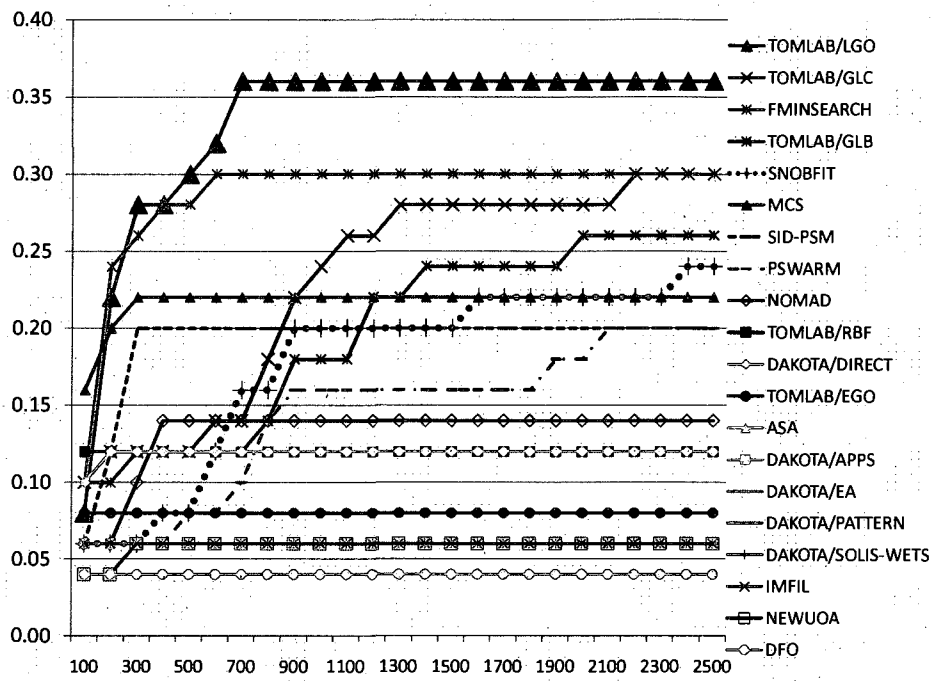

Figure 2.9: Fraction of problems solved to global optimality (average solver performance) as a function of allowable number of function evaluations.

2.7.4 Computational results with nonsmooth problems

Similarly, the 20 solvers of Table 2.2 were tested on the 47 nonsmooth problems. Figure 2.9 presents the fraction of problems solved to global optimality. LGO was able to solve the largest fraction of the problems amongst all solvers, although at 35% LGO succeed in solving a much smaller fraction than the number of problems it can solve for smooth problems. Figure 2.10 presents the fraction of problems for which a solver found the best solution amongst all solvers. For a small budget of function evaluations, MCS clearly dominates all other solvers, whereas for other cases its performance is comparable to LGO.

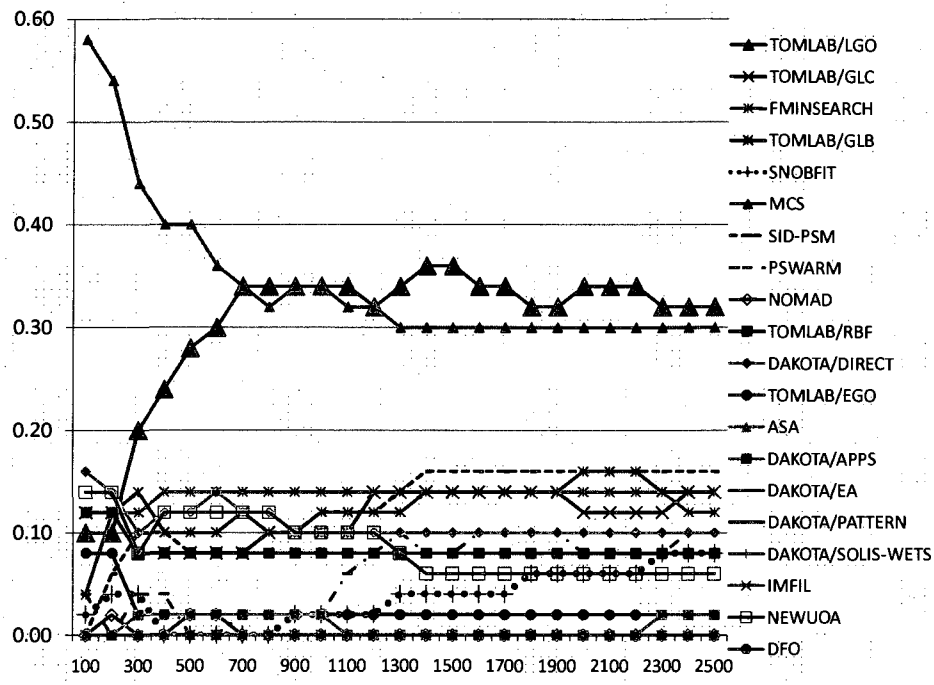

Figure 2.10: Fraction of nonsmooth problems, as a function of allowable number of iterations, for which a solver found the best solution amongst all solvers tested.

2.8 Conclusions

Derivative-free algorithms and their computational implementations are the preferred option for optimization of problems for which function evaluations are expensive and derivative information is unavailable. This is an area of recent growth generating considerable interest in the theory and practice of optimization. For several decades since the 1960s, the Hooke and Jeeves and Nelder-Mead simplex algorithms were the preferred approaches for these problems. Starting in the early 1990s, significant research in this area has produced most of the algorithms currently available for these problems and reviewed and evaluated in this chapter.

A set of 20 leading software implementations of state-of-the-art derivative-free optimization algorithms were tested on a large collection of publicly available problems. Our computational results show that attaining the best solutions even for very small

problems is a challenge for most of these solvers. The solvers MCS and TOMLAB/LGO, on average, provide the best solutions among all the solvers tested. However, computational results show that there is no single solver whose performance dominates that of all others. As many as 16 of the solvers provided the best solution possible for at least some of the test problems. Although no subset of solvers suffices to solve all problems at different absolute tolerance values, our results suggest that the combination of MCS and TOMLAB/LGO is sufficient to provide the best results in most cases. Problem dimensionality was found to rapidly increase the complexity of the search.

Given the rate at which derivative-free algorithms and software have developed over the past decade, as well as the current interest in this field, one can expect to see further progress in the near future. The need is obvious for implementations that can handle larger problems as well as for algorithms that handle the constrained case in more natural and more efficient ways.

Chapter 3

Model and Search: A derivative-free local algorithm

3.1 Introduction

The problem addressed in this chapter is the optimization of a deterministic function $f : \mathbb{R}^n \to \mathbb{R}$ over a box-bounded domain of interest $\mathcal{C}$. We assume that the derivatives of f are neither symbolically available nor numerically computable, and that bounds, such as Lipschitz constants, for the derivatives of f are also unavailable. The problem is of interest when derivative information is unavailable, unreliable, or impractical to obtain. Furthermore, the function f is assumed expensive to evaluate or somewhat noisy.

The derivative-free optimization algorithm *Model and Search* (`M&S`) is proposed and shown to be convergent to a stationary point. The search is oriented to improve the objective of the best known point by using models fitted by information from nearby evaluated points. Designed primarily to find local minima, `M&S`'s ideal role is to refine solutions obtained by algorithms with a broader global search component. In particular, `M&S` will be docked with the derivative-free global algorithm presented in Chapter 3.5. Algorithm `M&S` prioritizes reusing information from previously evaluated points to guide the search before evaluating new points.

Each iteration of the algorithm starts by collecting evaluated points around the best known point, thus allowing use of the complete set of previously evaluated points. The primary objective of an iteration is to find an improving point. The iteration may stop prematurely if an improving point is found. The second objective of the

iteration is to build a model that predicts a potential improving point. Models are built by interpolating points, and the simplest model considered is a fully linear model. Depending on availability of points, complete and incomplete quadratic models are considered. Incomplete quadratic models contains quadratic terms that are supposed to have greater sensibility on the objective function. If the model fails to predict an improving point, the final objective of an iteration is to ensure there is a positive basis around the best known point. Unsuccessful positive basis iterations are pivotal to proof the algorithm converge to a stationary point.

The Model and Search derivative-free optimization algorithm proposed combines several features present in other methods such as use of previously evaluated points, incomplete/complete model building and variable search directions and steps. In contrast to simplex-based algorithms ([101,130]), no set of points is kept through iterations but only a single point. Each iteration starts by collecting already evaluated points in a neighborhood of the best point. If this collected set permits to build a fully linear (or greater) model, then the model is optimized and its predicted improving point is evaluated. Otherwise, new points are evaluated until a fully linear model can be built, where a minimum of n linearly independent vectors are required. When points are not enough to build a complete quadratic model, incomplete quadratic models are built sequentially, giving priority to quadratic terms that have greatest sensibility on the objective function. The sensibility of a quadratic term is approximated by the measure of its respective components of the model's gradient vector. New points are selected such that together with the set evaluated form a set of linearly independent vectors and are spread around the best point. If an improving point is found during this process, new points in the same direction are evaluated until a worse point is found. Since the purpose of building a model is to have the best point in the center to better predict its neighborhood, the finding of an improving point marks the end of a given iteration. This model-building feature is similar to Trust-region model

based algorithms but allowing flexibility on the geometry of the points and the form and length of the model built, in spite of guaranteed accuracy of the model. If the model is unable to predict an improving point, new points are included such that all collected points form a positive basis around the best point. Unsuccessful iterations are equivalent to full iterations of generating set search methods, but considering a variable set of search directions and a variable steps for each direction. Finally, the update rule for the radius is to increase the radius when the best point was updated and even greater radius increases when more than one improving point was found during the iteration.

Section 3.2 presents a description of the algorithm and auxiliary routines. Section 3.3 presents convergence results for the local algorithm. Convergence results are based on two basic components of the algorithm. The first component is that at unsuccessful iterations there is a set of directions forming a positive basis. The second component is that each improving point needs to satisfy a sufficient decrease condition. Under mild assumptions, the algorithm is shown to converge to first-order stationary points. Section 3.4 presents computational experiments including comparisons to four black-box solvers over a set of 51 test problems from publicly available sources. Finally, Section 3.5 presents conclusions and future directions for research.

3.2 Model and Search (M&S): A Model-based local search algorithm

This section presents a detailed description of the proposed algorithm. Sections 3.2.1-3.2.4 present auxiliary routines used in the main algorithm described in Section 3.2.5.

3.2.1 Point projection

The *Project point* algorithm [**Algorithm 1**] receives a point z and returns a point $p(z) \in \mathcal{C}$. If z already belongs to $\mathcal{C}$, then its projection is $p(z) = z$. Otherwise, let $\mathcal{L}$ be the line that passes through the points z and c_k, where c_k is the current best point at iteration k. Let $\mathcal{S}$ be a set such that $\mathcal{S} = \{\mathcal{L} \cap \mathcal{C}\}$. Then, $p(z)$ is the closest point from $\mathcal{S}$ to z.

Algorithm 1 Project a point

1: **if** $z \in \mathcal{C}$ **then**
2: $\quad p(z) = z$
3: **else**
4: $\quad$ Determine the search direction: $v_s = z - c_k$
5: $\quad$ Find the maximum scalar: $\alpha^* = \max\{\alpha \mid c_k + \alpha v_s \in \mathcal{C}\}$
6: $\quad p(z) = c_k + \alpha^* v_s$
7: **end if**

3.2.2 Evaluation of points

Except for the evaluation of the set of starting points, all function evaluations are performed using the algorithm *line search* [**Algorithm 2**]. Candidate points for evaluation z are first projected using **Algorithm 1**, thus only evaluating points $z^* \in \mathcal{C}$. An evaluated point is regarded as an improving point if its objective value satisfies a minimum decrease criteria:

$$f(z^*) \leq f(c_k) - \rho(\|z^* - c_k\|) \tag{3.1}$$

where ρ is a forcing function that satisfies:

$$\rho(x)/x \to 0 \quad \text{as} \quad x \to 0 \tag{3.2}$$

If the point z^* satisfies condition (3.1), then iteration k is considered successful

and c_k is updated to z^*. Upon finding an improving point, additional points in the search direction $v_s = z - c_k$ are evaluated until a point is found not to be an improving point, a point is at the boundary of $\mathcal{C}$, or a maximum of β_e consecutive points are evaluated. All evaluated points are included in the set $\mathcal{H}$.

In addition, if an improving point is found, a signal is returned to the main loop. The iteration is considered successful and is terminated, obviating the evaluation of other possible candidate points and all other remaining steps of the iteration.

Algorithm 2 Line search

Initialize number of consecutive evaluations: $\beta_e = 0$
Initialize step coefficient: $\beta_c = 1$
Determine search direction: $v_s = z - c_k$
while ($count \leq M$) and ($v_e \leq v_e^*$) **do**
    Determine next candidate point: $z = c_k + \beta_c v_s$
    Project point: $z^* = p(z)$
    if $\min_{x \in \mathcal{H}} \|z^* - x\| > \delta_{min}$ **then**
        Evaluate z^*, include z^* in $\mathcal{H}$
        Increase *count* and β_e
        if $f(z^*) \leq f(c_k) - \rho(\|z^* - c_k\|)$ **then**
            Update $c_k = z^*$
            Update $\beta_c = \phi_3 \beta_c$
        else
            Exit
        end if
        if $(z \neq p(z))$ **then**
            Exit
        end if
    else
        Exit
    end if
end while
if $\beta_e > 0$ **then**
    End iteration from main loop
end if

3.2.3 Determine linear independence

Information from points evaluated is used to build models. These models are determined by interpolating a set of linearly independent points.

Given a set of vectors $\Pi_k = \{\pi_i\}$, algorithm *linear independence* [**Algorithm 3**] returns a set $\Pi_k^* \subset \Pi_k$ whose vectors are linearly independent. The set Π_k^* is obtained by an iterative process where at each iteration it is determined if a vector π_i can be expressed as a linear combination of the vectors $\Pi_k \backslash \pi_i$. The set Π_k is updated to $\Pi_k = \Pi_k \backslash \pi_i$ if there exists such linear combination.

Vectors in Π_k are sorted decreasingly with respect to their norms. Thus, vectors with larger norms will be eliminated first and Π_k^* will include the set of linearly independent vectors with smallest norms. Vectors with smaller norms are preferred since it is expected that vectors closer to c_k better describe the behavior of the function around a neighborhood of c_k and thus resulting in more accurate models.

Algorithm 3 Linear independence

1: Sort Π_k in increasing order with respect to the vector's norms
2: Let I be the set of indices $I = \{1, \ldots, |\Pi_k|\}$
3: Initialize $\nu = |I|$ and $i = \nu$
4: **while** $(i > 0)$ and $(\nu > 1)$ **do**
5: Solve the feasibility problem: $L = \min_{\alpha,\varepsilon} 0$ s.t. $\sum_{j \in I \backslash i} \alpha_j \pi_j + \varepsilon = \pi_i$, $-\alpha^* \leq \alpha \leq \alpha^*, -\varepsilon^* \leq \varepsilon \leq \varepsilon^*$
6: **if** L is feasible **then**
7: Update $I = I \backslash i$
8: **end if**
9: Update $i = i - 1$
10: **end while**
11: Let $\Pi_k^* = \{\pi_i \mid i \in I\}$

3.2.4 Model building

Models can be fitted using a given a set of points.

Algorithm *Build linear/quadratic models* [**Algorithm 4**] interpolates models around

the current best point c_k. Depending on the availability of points, models fitted varies from fully linear to fully quadratic models, allowing incomplete quadratic models.

Algorithm 4 starts by interpolating a linear model using n linearly independent vectors. Subsequent steps attempt to build incomplete quadratic models by including one quadratic term to the model at a time. The next quadratic term to be included is the term which is expected to achieve the most improvement in the accuracy of the model. It is assumed that the potential for improvement by including the quadratic term (i, j) is proportional to the combined linear coefficients of elements i and j. Then, the next quadratic term is chosen to have the greatest sum of the absolute values of the i and j gradient components, among the pairs (i, j) not included in the quadratic model yet. The vectors are increased one dimension by including a new component equal to the multiplication of the i and j components. The quadratic term (i, j) is included in the model only if there exists a set of vectors that is able to interpolate the resulting (incomplete) quadratic model. The process is repeated until no more quadratic terms can be included in the model. Finally, the models are optimized. If quadratic terms were added to the model, the resulting quadratic function is minimized over a neighborhood of c_k. Otherwise, since the function is linear, the optimal point is restricted to a distance of r_k from c_k.

3.2.5 A Model-based local search algorithm

The Model and Search algorithm [**Algorithm 5**] starts with the user providing an initial set of points which can include both evaluated and unevaluated points. The initial set of points $\mathcal{I}$ are all evaluated and included in the set of evaluated points $\mathcal{H}$. All points evaluated at later stages of the algorithm will also be included in $\mathcal{H}$, using a variable *count* to control the number of evaluations.

The best point found at the beginning of iteration k is denoted as c_k and is initialized to the best point from $\mathcal{H}$. Finally, d^* is a vector which is expected to be a

Algorithm 4 Build linear/quadratic models

Initialize Q to be a 0 matrix $n \times n$
Let M and M^* be matrices composed of the vectors of Π_k and Π_k^* respectively. $M = [\ (\pi_i)\ ]$ and $M^* = [\ (\pi_i^*)\ ]$
Let m^* be a vector $m^* = [\ (f(\pi_i^*) - f(c_k))\ ]^T$
Let $d^* = -M^{*-1}m^*$
Let *continue* $= 1$
while *continue* $= 1$ **do**
    $\max_{j,k \in [1,n]} |(d^*)_j| + |(d^*)_k|$ s.t. $Q(j,k) = 0$
    Add a new row to M with values $(\pi_i)_j(\pi_i)_k$
    Check linear independence for columns of M and obtain M^*
    Construct the m^* vector using the updated Π_k^*
    if M^* is full rank **then**
        Let $\delta^* = -M^{*-1}m^*$
        Include element (i,j) in the quadratic model. $Q(i,j) = 1$
        Label (i,j) the last index of vector δ^*
    else
        continue $= 0$
    end if
end while
if $Q \neq 0$ **then**
    Let d^* have the first n elements of δ^*
    Let matrix $C_Q(i,j) = \delta^*_{(i,j)}$ if $\delta^*_{(i,j)}$ exists, otherwise $C_Q(i,j) = 0$.
    Let $y = c_k + r_k \left(\text{argmin}_p p d^{*T} + p C_Q p^T\right)$ s.t. $-1 \leq p \leq 1$
else
    Let $y = c_k + r_k d^*$
end if

descent direction and is initialized to $\overrightarrow{0}$.

An iteration is considered *successful* if an improving point is found. The primary objective of an iteration is to find an improving point. To this purpose, the algorithm determines a set of candidate points. However, the iteration is terminated upon finding an improving point, omitting the evaluation of the remaining set of candidate points. In contrast, an *unsuccessful* iteration is not able to find an improving point after evaluating a set of candidate points that form a positive basis. A positive basis is a set of points that positively spans $\mathcal{R}^n$. Minimal positive basis are preferred since they require the minimum number of points to be evaluated [39]. A method to obtain a positive basis is to add to a linear basis the negative of a vector drawn from the

interior of the linear basis's positive cone. The availability of a set of points permits to build an interpolating model of the function. The number and geometry of the points available will limit the complexity of the model that can be obtained. Since an unsuccessful iteration eventually has a set of linearly independent points, at least a fully linear model will be obtained.

An iteration begins if *count* is within the maximum number of evaluations C_{max} and the radius r_k is greater than the minimum radius τ_r. In an effort to recycle information from previously evaluated points, points in $\mathcal{H}$ within a distance from c_k between $\phi_1 r_k$ and $\phi_2 r_k$ are collected in a set Y_k. If the set Y_k is empty, then a point y is evaluated in the line that passes through c_k and x^*, the second best point of $\mathcal{H}$, expecting this line to be a descent direction beyond c_k. The set Y_k is transformed, shifting the origin to c_k and scaling by $1/r_k$, obtaining the set Π_k, where all vectors are in the region between the hyperspheres with radii ϕ_1 and ϕ_2.

The algorithm then aims to build a basis. Algorithm *Linear independence* lets identify a set of independent vectors $\Pi_k^* \subset \Pi_k$. If there are fewer than n number of vectors in Π_k^*, new points will need to be incorporated to obtain a basis.

If c_k is an interior point, Π_k^* is complemented with a new set of vectors orthogonal to themselves and previous vectors in Π_k^*. Such vectors are determined by sequentially solving problem $S(\Pi_k^*)$, updating Π_k^* with the solution vector p.

$$
\begin{aligned}
(\mathrm{S}(\Pi_{\mathrm{k}}^*)) \qquad \max \quad & \sum_{i=1}^{|\Pi_k^*|} \|\pi_i^I - p\| \\
\text{s.t.} \quad & \pi_i^* \cdot p = 0, \quad i = 1, \ldots, |\Pi_k^*| \\
& p_l \leq \|p\| \leq p_u \\
& -1 \leq (p)_j \leq 1, \quad j = 1, \ldots, n
\end{aligned}
$$

Problem $S(\Pi_k^*)$ finds a vector p orthogonal to vectors in Π_k^* and with maximum sum of distance to these vectors. As long as $|\Pi_k^*| < n$ the set of orthogonal vectors to Π_k^* is

not empty and thus the solution to $S(\Pi_k^*)$ is feasible. Parameters $p_l < 1 < p_u$ bound the norm of p to within a neighborhood of 1.

When c_k is a boundary point, extreme directions need to be included in order to span the constrained space. Let L and U be the set of indices whose coordinates are at their bounds, and defined as $L = \{j \mid (c_k)_j = (lb)_j\}$ and $U = \{j \mid (c_k)_j = (ub)_j\}$. Extreme directions are determined sequentially considering the nature of Π_k^*. If $\Pi_k^* < n$, requiring a set of orthogonal vectors may lead to an infeasible problem. Problem $(B(\Pi_k^*, L, U))$ is solved instead and requires p to be composed of two components. The first component is vector r, orthogonal to vectors in Π_k^*, whereas the second component is a linear combination of the vectors in Π_k^*.

$$
\begin{aligned}
(\mathrm{B}(\Pi_k^*, \mathrm{L}, \mathrm{U})) \quad \min \quad & \gamma \\
\text{s.t.} \quad & \pi_i^* \cdot r = 0, \quad i = 1, \ldots, |\Pi_k^*| \\
& r + \sum_{i=1}^{|\Pi_k^*|} \varphi_i \pi_i^* = p \\
& p \cdot \pi_i^* \leq \gamma, \quad i = 1, \ldots, |\Pi_k^*| \\
& r_l \leq \|r\| \\
& -1 \leq (p)_j, (r)_j \leq 1, \quad j = 1, \ldots, n \\
& -1 \leq \gamma \leq 1 \\
& -\varphi^* \leq \varphi \leq \varphi^* \\
& 0 \leq (p)_j, \quad j \in L \\
& (p)_j \leq 0, \quad j \in U
\end{aligned}
$$

If $\Pi_k^* = n$, any additional vector will be linearly dependent. Since no orthogonal vectors nor linearly independent vectors are available, Problem $(PB(\Pi_k, L, U))$ finds instead an extreme vector which forms the maximum minimum angle to all other

vectors in Π_k.

$$
\begin{aligned}
(\mathrm{PB}(\Pi_k, \mathrm{L}, \mathrm{U})) \quad & \min \ \gamma \\
& \text{s.t.} \ \ \pi_i \cdot p \leq \gamma, \quad i = 1, \ldots, \|\Pi_k\| \\
& \qquad p_l \leq \|p\| \leq p_u \\
& \qquad -1 \leq (p)_j \leq 1, \quad j = 1, \ldots, n \\
& \qquad -1 \leq \gamma \leq 1 \\
& \qquad 0 \leq (p)_j, \quad j \in L \\
& \qquad (p)_j \leq 0, \quad j \in U
\end{aligned}
$$

After all extreme directions are included and if Π_k^* is not yet a linear basis, new vectors p are sequentially determined by solving Problem $(B(\Pi_k^*, L, U))$ and added to Π_k^* until forming a basis.

Points determined by solving problems $S(\Pi_k^*)$, $(B(\Pi_k^*, L, U))$ and $(PB(\Pi_k, L, U))$ are collected in a set P of candidate points for evaluation. Points in P are scaled to its original space and evaluated using the *line search* algorithm. If an improving point is found, line search keeps evaluating points as described in the *line search algorithm.* Upon finding an improving point, the iteration is considered successful and is terminated, omitting the evaluation of the remaining points in P. Since all points may not be evaluated, points expected to be improving points are given priority. It is expected that the gradient does not change radically, so the points are sorted increasingly with respect to the angle they form with the previously determined search direction d^*, which is an approximation to the gradient from previous iterations.

The availability of a set of points forming a linear basis, along with the point c_k permits to interpolate a fully linear model from which the search direction d^* is updated.

Information from additional points is exploited by building a quadratic model.

Since the number of additional points may not be enough to interpolate a complete quadratic model, ideally it should be considered the largest incomplete quadratic model the available points can interpolate, including the most significant quadratic terms. Algorithm *Build linear/quadratic models* sequentially includes a quadratic term (i, j) which have the greatest expected potential to improve the accuracy of the quadratic model. Before including the quadratic term (i, j) it is checked if the set of points is able to interpolate the resulting model. The process of expanding the model continues until the set of points is not able to interpolate larger models.

If the points available only permit to interpolate a linear model, an expected improving point is located along the search direction d^* and a candidate point is set to $y = c_k + r_k d^*$. If quadratic terms were successfully included, the candidate point y is the minimizer of the resulting quadratic model around a neighborhood of c_k. In either case, point y is evaluated and included in the set Π_k after its evaluation.

If no improving point was found, the iteration is declared unsuccessful upon checking that the set Π_k forms a positive basis. Problem $(PB(\Pi_k, L, U))$ is solved. A solution $\gamma > 0$ indicates that the set is indeed a positive basis. However, in order to enforce a minimum value of the cosine measure, γ is required to be greater than $\gamma^* > 0$. The cosine measure is defined:

$$\kappa(\Pi_k) = \min_{v \in \mathrm{R}^n} \max_{d \in \Pi_k} \frac{v^T d}{\|v\| \|d\|} \tag{3.3}$$

While $\gamma < \gamma^*$, the solution p determined by solving $(PB(\Pi_k, L, U))$ is evaluated, included in Π_k and the problem with updated Π_k is resolved.

The radius r_k is updated upon the result of the iteration. Unsuccessful iterations $(\beta_e = 0)$ decrease the radius by a factor of ψ_0. Successful iterations $(\beta_e \geq 1)$ increase the radius depending on the number of improving points found during the iteration.

The scheme rewards multiple improving points with higher ψ values.

$$r_{k+1} = r_k\psi, \quad \text{where } \psi = \begin{cases} \psi_0 < 1 & \text{when } \beta_e = 0, \\ \psi_1 > 1 & \text{when } \beta_e = 1, \\ \psi_2 \geq \psi_1 & \text{when } \beta_e = 2, \\ \psi_2^2 & \text{when } \beta_e > 2. \end{cases} \tag{3.4}$$

There is an exception to the updating rule in the case the incumbent and previous c_k point are within a distance equal to a fraction of r_k. This case can happen if the quadratic model predicts an improving point close to the original c_k, in which case it is sensible to conduct a focused analysis in this area. Another case is when the projection of a candidate point lies close to the original c_k. In both cases, despite the finding of an improving point, the radius is decreased.

3.3 Convergence of the algorithm

Under mild conditions, the Model and Search algorithm is shown to be convergent to first-order stationary points. The function f is assumed to be bounded below. By using the forcing function to accept improving points, the radius r_k is shown to converge to 0. Finally, the cosine measure imposed over the set of directions provides an upper bound to the norm of the gradient, which is shown to be bounded above by a function of the radius r_k. The following proof is presented for a more general case in Kolda *et al.* [80].

THEOREM 1. *Let f be bounded below and ρ a forcing function that satisfies condition (3.2). Then:*

$$\lim_{k\to\infty} r_k = 0$$

Algorithm 5 Model and Search algorithm

Evaluate starting point(s) $\mathcal{I}$. Include evaluated points in $\mathcal{H}$
Initialize $count = |\mathcal{I}|$, $k = 1$, $d^* = \overrightarrow{0}$ and $c_1 = \text{argmin}_{x \in \mathcal{H}} f(x)$
while $(count \leq C_{max})$ and $(r_k > \tau_r)$ **do**
    Find set of nearby points $Y_k = \{x \in \mathcal{H} \mid \phi_1 r_k \leq \|c_k - x_k\| \leq \phi_2 r_k\}$
    if $|Y_k| = 0$ **then**
        Set $y = c_k + r_k \frac{x^* - c_k}{\|x^* - c_k\|}$ where $x^* = \text{argmin}_{x \in \mathcal{H} \setminus c_k} f(x)$
        Perform a *line search* from y
    end if
    Shift and scale point set Y_k. $\Pi_k = \{\pi = \frac{x - c_k}{r_k} \mid x \in Y_k\}$
    Check linear independence of Π_k and obtain Π_k^*
    if $|\Pi_k^*| < n$ **then**
        Initialize $P = \varnothing$
        if c_k is an interior point **then**
            for $\lambda = 1$ to $v_{req} : n - |\Pi_k^*|$ **do**
                Solve $S(\Pi_k^*)$; include p in Π_k^*, Π_k and P
            end for
        else
            for $\lambda \in (B = L \cup U)$ **do**
                if $\nexists \bar{\pi} \in \Pi_k$ such that $(\bar{\pi})_\lambda \neq 0$ and $(\bar{\pi})_{b \in B \setminus \lambda} = 0$ **then**
                    if $|\Pi_k^*| < n$ **then**
                        Solve $B(\Pi_k^*, L, U)$; include p in Π_k^*, Π_k and P
                    else
                        Solve $PB(\Pi_k, L, U)$; include p in Π_k and P
                    end if
                end if
            end for
            for $\lambda = 1$ to $v_{req} : n - |\Pi_k^*|$ **do**
                Solve $B(\Pi_k^*, L, U)$ and include p in Π_k^*, Π_k and P
            end for
        end if
    end if
    Sort P increasingly with respect to $p \cdot d^*$
    Perform a *line search* from each unscaled point of P
    Interpolate a linear/quadratic model. Update d^* and obtain y
    Perform a *line search* from y. Scale y and include in Π_k
    repeat
        Solve $PB(\Pi_k, L, U)$
        if $\gamma < \gamma^*$ **then**
            Perform a *line search* from unscaled p and include p in Π_k
        end if
    until $\gamma \geq \gamma^*$
    Update r_k
end while

Proof. The limit above holds if:

$$\lim_{k\to\infty} \inf r_k = \lim_{k\to\infty} \sup r_k = 0$$

The first part is to show that $\lim_{k\to\infty} \inf r_k = 0$. The proof is by contradiction. Assume $\lim_{k\to\infty} \inf r_k = \triangle_* > 0$ and let $\rho(r_k) \geq \rho(\triangle_*) > 0$. Since $r_k > 0$ and considering the update rule (3.4), the successful steps $\mathcal{S}$ are infinite. Then, there exists a direction d_k such that:

$$f(x_k + r_k d_k) - f(x_k) < -\rho(r_k) \leq -\rho(\triangle_*) < 0 \quad \forall k \in \mathcal{S}$$

Since, $x_{k+1} = x_k$ for $k \in \mathcal{U}$, it follows that $f(x_k) \to \infty$ which contradicts the boundedness of f.

The second part is to show that $\lim_{k\to\infty} \sup r_k = 0$ The proof again is by contradiction. From the first part of the proof, it follows that there exist a subsequence $\{\triangle_i\}_{i\in K_1}$ converging to 0. Assume $\lim_{k\to\infty} \sup r_k = \triangle^* > 0$. Then, there exist a subsequence $\{\triangle_j\}_{j\in K_2}$ converging to $\triangle^*$. Assume that the sequences are selected such that K_1 is a nonincreasing sequence, K_2 is a nondecreasing sequence and $\{\triangle_i\}_{i\in K_1} < \{\triangle_j\}_{j\in K_2}$. Let $\{r_k\}_{k\in K_3}$ be a sequence of successful steps such that $\{\triangle_i\}_{i\in K_1} < \{r_k\}_{k\in K_3}$. Note that K_3 is an infinite sequence since there are infinite $i \in K_1$ and for each $i \in K_1$ there exists an $j \in K_2$, $i_* \in K_1$ with $i < j < i_*$ with finitely many elements $k \in K_3$ between (i, j) due to the update rule (3.4). Finally, there exists d_k such that:

$$f(x_k + r_k d_k) - f(x_k) < -\rho(r_k) \leq -\rho(\triangle_{i_1}) < 0 \quad \forall k \in K_3$$

Since, $f(x_{k+1}) \leq f(x_k)$ for $k \notin K_3$, it follows that $f(x_k) \to \infty$ which contradicts the boundedness of f. □

THEOREM 2. *Let* $\lim_{k\to\infty} r_k = 0$ *then:*

$$\lim_{k\to\infty} \|\nabla f(x_k)\| = 0 \tag{3.5}$$

Proof. Let $k \in \mathcal{U}$.

$$f(x_k) - \rho(r_k) \leq f(x_k + r_k d_k^*) \tag{3.6}$$

By the cosine measure (3.3), there exists a direction d_k^* from the set of directions D_k such that:

$$\kappa(D_k)\|\nabla f(x_k)\|\|d_k^*\| \leq -\nabla f(x_k)^T d_k^* \tag{3.7}$$

By the mean value theorem

$$f(x_k + r_k d_k^*) - f(x_k) = r_k \nabla f(x_k + \theta_k r_k d_k^*)^T d_k^* \quad \theta_k \in (0,1) \tag{3.8}$$

Reordering (3.8) and combining with (3.6):

$$0 \leq r_k \nabla f(x_k + \theta_k r_k d_k^*)^T d_k^* + \rho(r_k) \tag{3.9}$$

Dividing (3.9) by r_k and substracting $-\nabla f(x_k)^T d_k^*$ to each side:

$$-\nabla f(x_k)^T d_k^* \leq [\nabla f(x_k + \theta_k r_k d_k^*) - \nabla f(x_k)]^T d_k^* + \frac{\rho(r_k)}{r_k} \tag{3.10}$$

Finally, combining (3.7) and (3.10):

$$\kappa(D_k)\|\nabla f(x_k)\|\|d_k^*\| \leq [\nabla f(x_k + \theta_k r_k d_k^*) - \nabla f(x_k)]^T d_k^* + \frac{\rho(r_k)}{r_k} \tag{3.11}$$

Since $\kappa(D_k)$ and $\|d_k^*\|$ are bounded, then as $r_k \to 0$, $\|\nabla f(x_k)\| \to 0$. □

Table 3.1: Distribution of test problems.

	number of variables	number of problems
type 1	1 to 2	17
type 2	3 to 9	21
type 3	10 to 30	13
average	5.2	

3.4 Computational experiments

3.4.1 Test problems

As the majority of derivative free solvers, including the present algorithm, are relatively recent developments, test problems were restricted to a maximum of 30 variables. A total of 51 test problems were selected from the `globallib` [52] and `princetonlib` [116] collections of nonlinear programming problems. Table 3.1 presents a breakdown of the number of problems as a function of the number of variables. The test problems are diverse, including sum of squares problems, quadratic and higher degree polynomials, convex and non-convex functions, continuous and discontinuous problems, 5 problems with trigonometric functions, and 7 problems with exponential or logarithmic functions.

Test problems with free variables were imposed bounds of $[-10000, 10000]$. Tighter bounds were used if the proposed bounds were found to produce numerical difficulties due to overflowing. Each problem was tested starting from ten randomly generated points inside the box-bounded region. The same randomly generated points were used across other solvers involved in the comparison.

3.4.2 Solvers selected for comparison

The selected solvers share design concepts with the `M&S` algorithm and their main steps are primarily geared towards local search:

- IMFIL (Version 0.6).
- FMINSEARCH
- DAKOTA/PATTERN
- NEWUOA

3.4.3 Computational results

The 51 test problems were tested with a limit on the number of function evaluations set to 100. Since the M&S algorithm can only guarantee a local solution and it has been observed that only after a few function evaluations are required to reach such local solution, the limit of function evaluations is set to 100. The M&S algorithm presented is coded in MATLAB. Subproblems $(B(\Pi_k^*, L, U))$ and $(PB(\Pi_k, L, U))$ were solved using BARON. The computational experiments were performed on a cluster of 22 Intel Xeon 1700 Mhz processors running Linux and MATLAB 7.3 R2006b.

Information from test problem Web sites [52, 102, 103, 116] and results obtained by solving the test problems with the general purpose global optimization solver BARON [123, 124] were used as obtain the global optimal solution for each test problem to be used for comparison of the results obtained by the solvers under study.

Figure 3.1 presents how optimization is carried out over the test problem camel6. Evaluated points are marked with red crosses. The starting point and final solution are marked with a green and yellow circle respectively. The trajectory of the progress of the best point is marked with a cyan line. As shown in Figure 3.1, two consecutive searches through descent directions are sufficient to arrive to a neighborhood of a local minimum, which in this case is also a global minimum. The solution is refined by evaluating points around the global minima.

Table 3.2 presents the test problem name, number of variables, global solution and the solution reported by the solvers. The solvers solutions are the median values of the

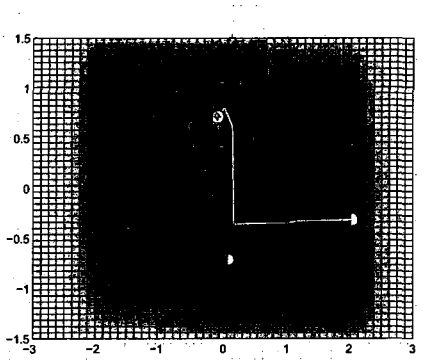

Figure 3.1: Optimization progress of M&S for test problem camel6.

ten instances solved. Median values are presented instead of averages since there are instances with high objective values that distort the average value. Despite the limited number of function evaluations, the median of the solutions for the M&S algorithm is able to reach the global solution (within a tolerance of $1E-5$) for 7 problems, compared to 2, 1, 3 and 1 respectively for IMFIL, FMINSEARCH, DAKOTA/PATTERN and NEWUOA.

Figure 3.2 presents a synthesis of the results for the 10 instances solved for each test problem. Each color bar denotes the comparison of a given solver (*e.g.*, IMFIL, FMINSEARCH, DAKOTA/PATTERN and NEWUOA) with the M&S algorithm. The blank bar case occurs when the solver and the M&S algorithm break even, this is, each solver got better results for 5 instances. A bar drawn in the right side of the graph denotes the number of instances where the M&S algorithm got better solutions from the break even point; a value of +5 happens when the M&S algorithm solved +5 instances in addition to the break even value of 5, this is, the M&S algorithm got better results for 10 out of 10 instances. Similarly, bars drawn in the left side of the graph denotes the number of instances when the solver performed better than the M&S algorithm from the break even point. Figure 3.2 shows a majority of bars in the right side, providing evidence that the M&S algorithm performs better than the solvers compared in most of the instances. From the total of instances solved, the M&S algorithm obtained

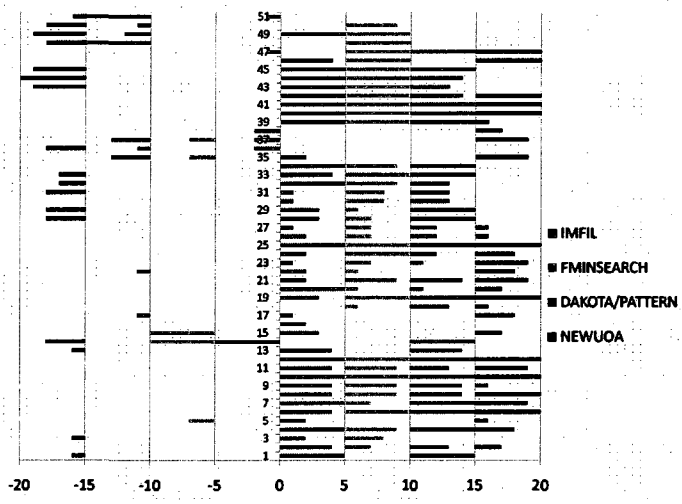

Figure 3.2: Comparison of solvers: 10 instances.

better solutions in 76.8% of the instances compared to IMFIL, 75.7% compared to FMINSEARCH, 73.7% compared to DAKOTA/PATTERN and 61.6% compared to NEWUOA.

Figures 3.3 - 3.5 present a ranking of the median solutions by type of problem. The median solutions are ranked, where rank 1 is reserved for the best solution among the 5 solvers compared. The test problems are presented in an order such that the left part of the figures present problems for which the M&S algorithm achieves a good ranking (rank 1 or 2) and the right part presents the remaining problems. Across these figures, it can be noted that the M&S algorithm achieves a good ranking for the majority of the test problems.

3.5 Conclusions

A local derivative-free algorithm is presented that combines techniques such as positive-basis-set based search, gradient estimation, and quadratic model building and optimization. The solver is shown to be convergent to a first-order stationary point. A MATLAB implementation is described and computational results are performed for 51 test problems from publicly available sources. Results show that performance is

Table 3.2: Median solution (10 instances) for test problems.

problem	vars	global	M&S	IMFIL	FMINSEARCH	DAKOTA/PATTERN	NEWUOA
3pk	30	1.72E+00	9.10E+10	8.92E+10	3.80E+12	4.01E+12	2.96E+12
aircrftb	5	0.00E+00	1.24E+15	1.25E+14	1.19E+15	2.90E+13	4.71E+14
allinit	3	1.67E+01	1.72E+01	1.76E+11	2.26E+15	3.80E+12	6.50E+15
allinitu	4	5.74E+00	7.06E+15	5.97E+10	6.71E+15	6.67E+16	7.59E+15
arglinb	10	4.63E+00	1.06E+05	2.60E+08	8.60E+08	6.40E+08	1.15E+05
arglinc	8	6.14E+00	3.83E+04	5.11E+08	1.36E+08	1.20E+08	1.07E+03
beale	2	0.00E+00	5.78E+00	8.61E+09	4.54E-01	1.48E+00	1.63E+08
biggs3	3	0.00E+00	2.23E+00	2.73E+00	2.23E+00	2.23E+00	3.77E+02
box2	2	0.00E+00	1.47E-01	1.46E+00	6.17E-03	8.12E-02	1.60E+00
box3	3	0.00E+00	2.58E-04	8.29E+05	3.82E+01	1.24E+04	1.36E+08
bqp1var	1	0.00E+00	0.00E+00	0.00E+00	1.17E-04	5.00E-06	2.07E-04
brownbs	2	0.00E+00	9.93E+11	9.92E+11	9.95E+11	9.84E+11	1.00E+12
camel1	2	-1.03E+00	-1.03E+00	-1.03E+00	-1.03E+00	-1.03E+00	9.44E-01
camel6	2	-1.03E+00	-1.03E+00	-1.03E+00	-1.03E+00	-1.03E+00	-4.72E-01
chi	2	-4.33E+01	-4.33E+01	-4.24E+01	-3.13E+01	-4.23E+01	-3.11E+01
cliff	2	2.00E-01	2.00E-01	2.03E-01	2.00E-01	2.00E-01	2.00E-01
cube	2	0.00E+00	4.21E+02	5.84E+07	3.90E+02	1.02E+10	3.23E+02
denschna	2	0.00E+00	5.58E-07	2.25E+03	2.29E-03	1.63E+03	5.60E+14
denschnb	2	0.00E+00	4.32E-01	7.64E+02	6.65E+02	8.64E-01	1.92E+00
denschnc	2	0.00E+00	1.01E-01	1.21E+10	7.17E+06	3.70E+07	1.57E+19
denschnd	3	0.00E+00	6.32E+13	4.40E+18	2.72E+15	2.76E+15	7.29E+19
denschne	3	0.00E+00	1.59E+07	2.04E+08	1.91E+06	4.76E+04	1.83E+25
denschnf	2	0.00E+00	5.28E-05	5.30E+06	6.56E-02	2.19E+01	1.17E+04
dixon3dq	10	0.00E+00	1.61E+07	5.71E+07	1.91E+08	2.64E+07	7.84E+05
expfit	2	2.41E-01	2.40E+01	2.40E+01	2.40E+01	2.40E+01	2.40E+01
genhumps	5	0.00E+00	1.64E+03	3.46E+04	7.52E+05	8.85E+04	2.23E-01
growth	3	1.00E+00	3.54E+03	3.52E+03	3.54E+03	3.54E+03	1.76E+07
hart6	6	-3.32E+00	-3.04E+00	-2.95E+00	-2.42E+00	-2.76E+00	-3.18E+00
hs003	2	0.00E+00	6.31E-16	5.19E-04	2.54E+00	1.14E+00	2.37E+02
hs110	10	-4.58E+01	-3.12E+01	-2.93E+01	-7.37E+00	-4.09E+01	-3.95E+01
osbornea	5	5.46E-05	9.82E+07	5.55E+06	8.40E+06	3.27E+06	3.39E+08
palmer1	4	1.18E+04	4.37E+04	4.41E+04	4.45E+04	4.43E+05	4.39E+04
palmer1c	8	9.76E-02	8.60E+11	5.98E+12	1.79E+12	8.34E+13	2.83E+11
palmer3c	8	1.95E-02	4.93E+10	2.39E+11	4.97E+10	3.21E+11	4.50E+09
palmer3e	8	5.07E-05	3.38E+09	1.72E+10	2.53E+10	2.12E+10	2.16E+10
polygon	19	-1.00E+01	-3.60E+00	-3.88E+00	-3.42E+00	-5.87E+00	-4.40E+00
s272	6	4.84E-18	8.70E+00	8.79E+00	1.52E+03	8.70E+00	4.38E+02
s273	6	8.25E-20	7.67E+12	7.79E+15	2.03E+18	2.78E+16	4.96E+16
s281	10	1.45E-05	1.27E+03	3.33E+03	2.87E+03	2.08E+03	7.95E+02
s282	10	1.16E-20	3.43E+08	2.28E+16	8.23E+16	7.82E+15	4.81E+15
s287	20	2.13E-19	1.80E+15	3.75E+17	8.47E+17	2.26E+17	9.84E+16
s288	20	2.70E-12	9.27E+14	2.76E+16	4.68E+17	1.00E+16	4.85E+16
s289	30	6.66E-16	1.00E+00	1.00E+00	1.00E+00	1.00E+00	1.00E+00
s292	30	0.00E+00	5.15E+09	1.09E+10	1.38E+10	8.34E+09	5.86E+08
s371	9	1.40E-06	6.75E+12	7.71E+13	1.14E+14	2.37E+14	5.00E+12
s379	11	2.03E-01	1.59E+07	8.46E+07	7.26E+27	3.84E+06	1.37E+20
shekel	4	-1.02E+01	-3.89E+00	-3.02E+00	-2.53E+00	-2.67E+00	-3.89E+00
sineali	20	-1.90E+03	-1.09E+03	-1.12E+03	-8.68E+02	-1.21E+03	-1.39E+03
ex8_1_6	2	-1.01E+01	-3.22E-07	-2.58E-02	-5.05E+00	-6.01E-08	-5.05E+00
rbrock	2	0.00E+00	2.63E-02	2.41E+00	4.71E-01	4.71E+00	2.01E-01
st_bsj3	6	-8.68E+04	-8.68E+04	-8.68E+04	-4.61E+04	-8.40E+04	-5.05E+04

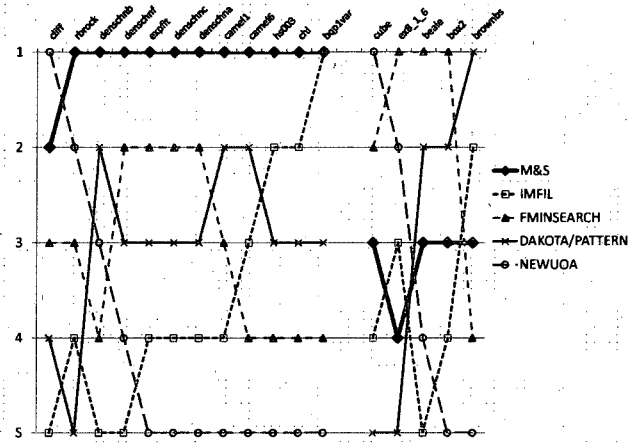

Figure 3.3: Ranking of solvers for problems of type 1.

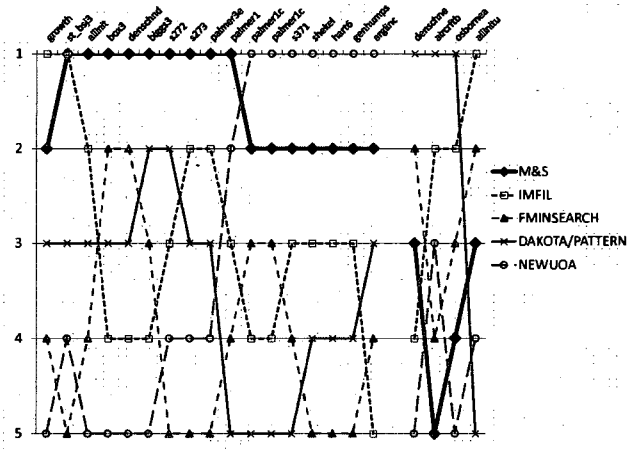

Figure 3.4: Ranking of solvers for problems of type 2.

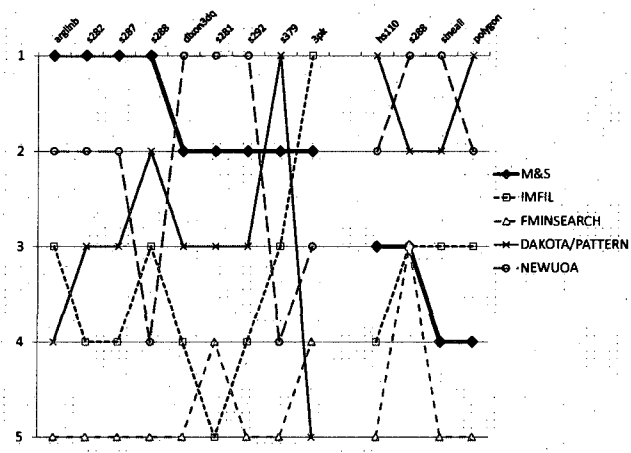

Figure 3.5: Ranking of solvers for problems of type 3.

better in comparison to other derivative-free solvers.

Chapter 4

Branch and Model: A derivative-free global algorithm

4.1 Introduction

The problem addressed in this chapter is the optimization of a deterministic function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ over a box-bounded domain of interest $\mathcal{C}$. We assume that the derivatives of f are neither symbolically available nor numerically computable, and that bounds, such as Lipschitz constants, for the derivatives of f are also unavailable.

The derivative-free optimization algorithm *Branch and Model* (B&M) is proposed and shown to be convergent in the limit to a global minimum. The proposed algorithm is based on modeling the function of interest around each one of the evaluated points. Models are fitted only using information from other nearby evaluated points, not requiring further points to be evaluated. Algorithm (B&M) is shown to perform a dense search. A local search algorithm, Model and Search (M&S), was proposed in the previous chapter to refine the best solution of B&M. Algorithm M&S is employed after a given number of evaluated points fail to improve the best known solution.

The B&M algorithm models the function of interest using a collection of local surrogate models around each evaluated point. Multiple local surrogate models permit the assumption that the behavior of the function is simple relative to a single surrogate function that models the entire feasible space. In a small neighborhood, the function is assumed to behave independently for each variable. The feasible space is partitioned in a set of hypercubes that define the area of influence of each point's model.

The surrogate model considered is a function of the distance to the evaluated point in a given hypercube and its objective value. Radial basis function based methods use a function of the radius, often considering a single surrogate model [55, 113].

Section 4.2 presents a detailed description of the algorithm, including an auxiliary routine for partitioning the feasible space. Section 4.3 presents convergence results for the algorithm. Convergence results rely on showing that the search is dense over the feasible space. Thus, the algorithm is shown to find a global minimum in the limit. Furthermore, the use of a locally convergent solver to refine solutions guarantees convergence if the global search approachhes the basin of the global minima. Section 4.4 presents computational experiments including comparisons to six state-of-the-art black-box solvers over a set of 30 test problems from publicly available sources. Finally, Section 3.5 presents conclusions and future directions for research.

4.2 Branch and Model: A Model-based global search algorithm

4.2.1 Hypercube subdivision

The *Hypercube subdivision* algorithm [**Algorithm 6**] partitions the hypercube $\mathcal{H}$ into a collection of $|\mathcal{P}|$ hypercubes $\{\mathcal{H}_i\}$ such that each hypercube $\mathcal{H}_i$ contains a single point $p^{(i)} \in \mathcal{P}$. The algorithm iteratively partitions a hypercube into two new hypercubes, each of them containing at least one point. Thus, since the set of points $\mathcal{P}$ is finite, the original hypercube $\mathcal{H}$ will be split at most a finite number of times. When partitioning a hypercube, a coordinate to be split and the value of the coordinate must be determined. The algorithm starts by including the hypercube $\mathcal{H}$ that defines the box-bounded feasible region into a working set of hypercubes T. The algorithm then enters a loop, and starts by selecting a hypercube T from T.

Let $\mathcal{P}_T = \{p \in \mathcal{P} \cap T\}$ be the set of points p contained in hypercube T. For each coordinate $j = 1, \ldots, n$, sets β_j are assembled with the j coordinates of points $p \in \mathcal{P}_T$. These sets β_j are sorted increasingly to obtain sets $\hat{\beta}_j$ where values β_j^* are determined to be the maximum difference between two consecutive elements of $\hat{\beta}_j$. The splitting coordinate s is selected to be the largest value of the expression $\beta_j^*/(\mathrm{ub}_j - \mathrm{lb}_j)$, scaling the differences β_j^* to its range of values defined by its lower and upper bounds. The coordinate s is split at the value r_s, determined by averaging the two values used in determining β_s^*. The hypercube T is then split. The resulting hypercubes which contain multiple points are included in the set T. However, if a resulting hypercube contains a single point $p^{(i)}$, then that hypercube is labeled as $\mathcal{H}_i$.

Algorithm 6 Hypercube subdivision

Initialize $\mathrm{T} = \{\mathcal{H}\}$
while $\mathrm{T} \neq \varnothing$ **do**
    Select a hypercube $T \in \mathrm{T}$ and remove it from the set $\mathrm{T} = \mathrm{T} \backslash T$
    Let $\mathcal{P}_T = \{p \in \mathcal{P} \cap T\}$ be a set of points p contained in hypercube T
    Let β_j be a set containing the scalars p_j, where $p \in \mathcal{P}_T$.
    Sort increasingly the elements of β_j and obtain $\hat{\beta}_j$
    Let β_j^* be the maximum difference between two consecutive elements in $\hat{\beta}_j$. Let r_j be the average of those elements.
    Find $s = \mathrm{argmax}_j \ \beta_j^*/(\mathrm{ub}_j - \mathrm{lb}_j)$
    Divide hypercube T at position r_s of coordinate s, obtaining two hypercubes T_1 and T_2.
    for $k = 1$ to 2 **do**
        Let $\mathcal{P}_k = \{p \in \mathcal{P} \cap \mathrm{T}_\mathrm{k}\}$
        if $|\mathcal{P}_k| > 1$ **then**
            $\mathrm{T} = \mathrm{T} \cup \mathrm{T}_\mathrm{k}$
        else
            Determine the index i of the point: $p^{(i)} \in \mathcal{P}_k$.
            Let $\mathcal{H}_i = \mathcal{P}_k$
        end if
    end for
end while

4.2.2 A Model-based global search algorithm

The Global solver algorithm [**Algorithm 7**] starts with the user providing an initial set of points, which can include both evaluated and unevaluated points. The initial set of points $\mathcal{I}$ are then evaluated and included in the set of evaluated points $\mathcal{P}$. All points evaluated at later stages of the algorithm will also be included in $\mathcal{P}$, using a variable *count* to control the number of function evaluations.

A well-designed initial sample set is needed to ameliorate the strong dependency on the set of initial points $\mathcal{I}$. Since no information is available about the effect of the variables over the objective function, the sample set should be spread over the entire feasible space. Various sample designs have been proposed, ranging from random samples, factorial, fractional factorial, central composite, Latin hypercube, among others. For the purpose of this algorithm, random samples and classical Design of Experiment designs are not the best choice, because the former does not guarantee that the sample covers the entire feasible space and the latter often requires a high number of sample points. The high number of required sample points is detrimental, because the majority of the points are concentrated at corners of the feasible space, corresponding to extreme scenarios.

Latin hypercube sampling (LHS) [95] produces designs that are uniformly distributed throughout the domain and have the flexibility to be determined for any sample size desired. Upon determining a sensible sample size $|\mathcal{L}|$ for a LHS design $\mathcal{L}$, each variable range is divided into $|\mathcal{L}|$ segments, sampling each segment once. Points $p \in \mathcal{L}$ are determined by selecting one sampled valued for each variable without repetition. LHS designs used in this algorithm are determined by maximizing the minimum distance between sample points. If the objective function is not dependent on a variable, each point in a LHS design still contributes information, as opposed to classical designs where a subset of points will contain duplicate information. LHS is used in multiple derivative-free algorithms, such as response surface methods (RSM)

based on kriging functions [133], radial basis functions [63] and the Efficient Global algorithm (EGO) [73, 126]. The size of the LHS design should be adjusted to the nature of the problem solved, aiming to keep the sample size to a minimum, leaving the majority of the computational budget to the subsequent global search. In the context of the EGO algorithm, Jones *et al.* [73] suggested a sample size of ten times the number of variables. In the context of RSM, Sóbester *et al.* [129] recommended a sample size between 35% and 60% of the computational budget.

At each iteration, the proposed algorithm considers a collection of local surrogate models around each evaluated point. Multiple local surrogate models permit the assumption that the behavior of the function is simple relative to a single surrogate function that models the entire feasible space. In a small neighborhood, the function is assumed to behave independently for each variable. The surrogate model $\hat{f}$ is the following:

$$\text{(M)} \qquad \hat{f}(x) = f(p) + \sum_{i=1}^{n} \left[\alpha_i |x_i - p_i|_+^{\gamma_i} + \beta_i |x_i - p_i|_-^{\lambda_i}\right] \tag{4.1}$$

Model (M) is a surrogate of the function f in a neighborhood of the evaluated point p. The surrogate is composed of two terms; the first term is the known function value at point p, and the second term is a weighted distance function, that allows different weights for positive and negative deviation with respect to the point p. The model parameters are $\alpha, \beta, \gamma, \lambda$, where each of them is a n vector resulting in a total of $4n$ model parameters. Before being able to predict candidate points, the algorithm needs to determine the feasible space and the parameter values for each surrogate model.

Algorithm Hypercube subdivision [**Algorithm 6**] is used to partition the hypercube that defines the box-bounded feasible space into a set of hypercubes, each containing a single evaluated point. Recognizing that the minimization of surrogate $\hat{f}$ over a hypercube will likely have its solution at a boundary point, the hypercubes

are partitioned at the largest unexplored segments for each variable. This partition choice is made with the expectation that the surrogate will explore promising areas and subsequent iterations will benefit from a set of points covering the feasible space. The result of **Algorithm 6** is a set of hypercubes $\{\mathcal{H}_j\}$, where each $\mathcal{H}_j$ contains the evaluated point $p^{(j)}$.

With a partitioned space $\{\mathcal{H}_j\}$, sets $\{\mathcal{N}_j\}$ are determined, where $\mathcal{N}_j$ includes the indices of the adjacent hypercubes to $\mathcal{H}_j$. Model parameters are fitted by solving (FIT($\mathcal{H}_j$)):

$$
\begin{aligned}
(\mathrm{FIT}(\mathcal{H}_j)) \quad \min \quad & \sum_{k \in \mathcal{N}_j} \left[\sum_{i=1}^{n} \alpha_i x_{k,i}^{\gamma_i} + \beta_i y_{k,i}^{\lambda_i} - g(z_k) \right]^2 && (4.2)\\
\text{s.t.} \quad & -\alpha^* \leq \alpha_i + \beta_i \leq \alpha^*, \quad i = 1, \ldots, n && (4.3)\\
& -\gamma^* \leq \gamma_i - \lambda_i \leq \gamma^*, \quad i = 1, \ldots, n && (4.4)\\
& x_{k,i} = |p_i^{(k)} - p_i^{(j)}|_+, \quad i = 1, \ldots, n; \quad k \in \mathcal{N}_j && (4.5)\\
& y_{k,i} = |p_i^{(k)} - p_i^{(j)}|_-, \quad i = 1, \ldots, n; \quad k \in \mathcal{N}_j && (4.6)\\
& g(z_k) = f(p^{(k)}) - f(p^{(j)}), \quad k \in \mathcal{N}_j && (4.7)
\end{aligned}
$$

Expressions (4.5) - (4.7) are constraints dependent on auxiliary parameters x and y, obtained from the evaluated points $p^{(j)}$ and $p^{(k)}$, where $k \in \mathcal{N}_j$. In order to obtain a smooth surrogate function at $\hat{f}(p^j)$, constraints (4.3) - (4.4) enforce $\alpha \approx \beta$, and $\gamma \approx \lambda$. The objective (4.2) minimizes the sum of squares of the error fitting the objective function values. The model parameters obtained from solving (FIT($\mathcal{H}_j$)) are used to minimize the expected value of the model over its feasible space $\mathcal{H}_j$. The

minimization problem $\text{MIN}(\alpha, \beta, \gamma, \lambda, \mathcal{H}_\text{j})$:

$$(\text{MIN}(\alpha, \beta, \gamma, \lambda, \mathcal{H}_\text{j})) \quad \min \sum_{i=1}^{n} \alpha_i x_i^{\gamma_i} + \beta_i y_i^{\lambda_i} \tag{4.8}$$

$$\text{s.t.} \quad \text{l}_\text{j} \leq \text{x} - \text{y} \leq \text{u}_\text{j} \tag{4.9}$$

$$x_i y_i = 0, \quad i = 1, \ldots, n \tag{4.10}$$

is considered, where l_j and u_j are the lower and upper bounds of the hypercube $\mathcal{H}_j$. The optimal point predicted by the model is $z = x - y$, where x and y contains the positive and negative components respectively. The complementarity constraint (4.10) is required to prevent x and y from both being active for the same variable.

Solving problems $(\text{FIT}(\mathcal{H}_\text{j}))$ and $\text{MIN}(\alpha, \beta, \gamma, \lambda, \mathcal{H}_\text{j})$ for all hypercubes at each iteration can be cumbersome and unnecessary. Assuming f is a Lipschitzian function, each hypercube can be lower bounded. Let $L > 0$ denote the Lipschitz constant of f. Then $|f(a) - f(b)| \leq L \parallel a - b \parallel$ for all a, b in the domain of f. As the Lipschitz constant L is not known, a hypercube $\mathcal{H}_k$ is likely to contain the optimal solution if there exists $\bar{L} > 0$ such that:

$$f(p^{(k)}) - \bar{L} d_k \leq f(p^{(j)}) - \bar{L} d_j, \quad j = 1, \ldots, |\{\mathcal{H}_j\}| \tag{4.11}$$

All hypercubes $\mathcal{H}_k$ that satisfy condition (4.11) are referred as *potentially optimal*. Assuming a Lipschitz constant $\bar{L}$, potentially optimal hypercubes have the lowest lower bound among all other hypercubes. The lower bound for a hypercube is given by $f(p^{(k)}) - \bar{L} d_k$, where d_k is the largest distance from the point p^k to the vertices of $\mathcal{H}_k$. The set of potentially optimal hypercubes is denoted $\mathcal{O} \in \{\mathcal{H}_j\}$, and the search for the remaining iterations is restricted to these hypercubes.

Model (4.1) is fitted and minimized for hypercubes in $\mathcal{O}$. As the optimization progresses, the set of potentially optimal hypercubes per iteration grows, consequently,

a limit on the number of evaluated points per iteration is enforced. Points with lower predicted values are given priority for evaluation. Finally, since the model (4.1) is not expected to be accurate for larger hypercubes, a group of hypercubes among the largest is evaluated. Instead of using the predicted point, the furthest vertex of the hypercube $\mathcal{H}_k$ from $p^{(k)}$ is evaluated.

If the evaluation of points from potentially optimal hypercubes fails to produce an improving point after a number of consecutive tries, the algorithm starts a local search from the point with the lowest objective value. Calls to a local search attempts to focus the optimization on a region and refine the solution. The local search algorithm [120] is a convergent model-based algorithm that evaluates points in the neighborhood of the best known point attempting to identify the local minimum.

Finally, while hypercubes $\mathcal{H}_k$ delimit spaces around each evaluated point, it can possibly change each $\mathcal{H}_k$ at every iteration. Moreover, Hypercube subdivision by itself does not guarantee that the maximum size of the hypercubes decreases with each iteration. A second set of partitioned hypercubes $\mathcal{C}_i$ for all $p^{(i)} \in \mathcal{P}$ is defined, where $\mathcal{C}_i$ is obtained by the same rules of Hypercube subdivision with the difference that $\mathcal{C}_i$ is only updated for hypercubes that contain newly evaluated points. The proposed algorithm evaluates a point from the largest hypercube in $\{\mathcal{C}_i\}$.

4.3 Convergence of the algorithm

The global algorithm search is shown to be dense, and thus, to converge to the global minima of f.

THEOREM 3. *The global algorithm search is dense.*

Proof. Assume the search is not dense. Then, there exists a region where no point is evaluated. Let C^* be a hypercube inscribed in the unevaluated region and d^* be its size.

Algorithm 7 Global solver algorithm

1: Determine and evaluate a Latin Hypercube Sample $\mathcal{L}$ and user provided starting points $\mathcal{I}$. Include evaluated points in $\mathcal{P}$
2: Initialize *count* $= |\mathcal{P}|$, $k = 1$, and $c_1 = \text{argmin}_{x \in \mathcal{P}} f(x)$
3: Let $\mathcal{H}$ be the box-bounded feasible space
4: Perform a *Hypercube subdivision* to $\mathcal{H}$. Obtain hypercubes $\{\mathcal{C}_i\}$
5: **while** *count* $\leq C_{max}$ **do**
6: Perform a *Hypercube subdivision* to $\mathcal{H}$. Obtain hypercubes $\{\mathcal{H}_j\}$
7: Determine the set of indices of adjacent hypercubes $\{\mathcal{N}_j\}$
8: Let $\mathcal{O} \subset \{\mathcal{H}_j\}$ be the set of potentially optimal hypercubes
9: **for** $\mathcal{H}_j \in \mathcal{O}$ **do**
10: Fit the model (4.1) parameters by solving (FIT($\mathcal{H}_j$))
11: Find the minimizer predicted by the model (4.1) by solving MIN($\alpha, \beta, \gamma, \lambda, \mathcal{H}_j$)
12: **end for**
13: Sort $\mathcal{O}$ by its model predicted value
14: Evaluate the first n_1 number of predicted points from $\mathcal{O}$
15: Sort $\mathcal{O}$ by its hypercube size
16: Evaluate n_2 number of furthest vertex of hypercubes from $\mathcal{O}$ not evaluated before
17: Update $\{\mathcal{H}_j\}$ by partitioning the hypercubes with new points evaluated
18: Evaluate the center of the largest hypercube in $\{\mathcal{H}_j\}$
19: **end while**

The set $\{\mathcal{C}_i\}$ is a partition of the feasible space. At every iteration, the largest hypercube $\mathcal{C}_i$ is evaluated. Thus, the sequence of largest hypercube values d^k is decreasing. After a finite number of iterations, d^k will be d^*. Therefore, the hypercube C^* will be selected for evaluation, which contradicts that C^* is not evaluated. □

4.4 Computational experiments

4.4.1 Test problems

Test problems were restricted to a maximum of 30 variables. A total of 30 test problems were selected from the `globallib` [52] and `princetonlib` [116] collections of nonlinear programming problems. Table 4.1 presents a breakdown of the number of problems as a function of the number of variables. The test problems are distributed in

Table 4.1: Distribution of test problems.

	number of variables	number of problems
type 1	1 to 3	16
type 2	4 to 30	14
average	5.9	

two types by their number of variables. The test problems are diverse, including sum of squares problems, quadratic and higher degree polynomials, convex and non-convex functions, continuous and discontinuous problems, 4 problems with trigonometric functions, and 4 problems with exponential or logarithmic functions.

Test problems with free variables were imposed bounds of $[-10000, 10000]$. Tighter bounds were used if the proposed bounds were found to produce numerical difficulties due to overflowing. Each problem was tested starting from ten randomly generated points inside the box-bounded region. The same randomly generated points were used across other solvers involved in the comparison.

4.4.2 Solvers selected for comparison

The following solvers were used for comparisons.

- `IMFIL`
- `NEWUOA`
- `TOMLAB/GLBSOLVE`
- `TOMLAB/LGO`
- `TOMLAB/RBFSOLVE`

The list includes the best performing solver `TOMLAB/LGO` as identified in Chapter 1.

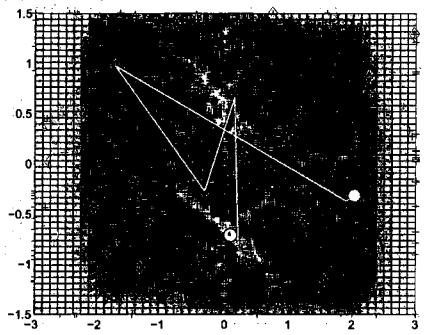

Figure 4.1: Optimization progress of B&M for test problem camel6.

4.4.3 Computational results

The 30 test problems were tested with a limit on the number of function evaluations set to 2500. The B&M algorithm presented is coded in MATLAB. Subproblems (FIT($\mathcal{H}_j$)) and MIN($\alpha, \beta, \gamma, \lambda, \mathcal{H}_j$) were solved using BARON. The computational experiments were performed on a cluster of 22 Intel Xeon 1700 Mhz processors running Linux and MATLAB 7.3 R2006b.

Information from test problem Web sites [52, 102, 103, 116] and results obtained by solving the test problems with the general purpose global optimization solver BARON [123,124] were used as obtain the global optimal solution for each test problem to be used for comparison of the results obtained by the solvers under study.

Figure 4.1 presents how optimization progressed for camel6. Evaluated points are marked with red crosses. The starting point and final solution are marked with a green and yellow circle respectively. The trajectory of the progress of the best point is marked with a cyan line. Finally, the initial LHS sample points are marked with a green diamond. As shown in Figure 4.1, the LHS samples evenly the feasible space. B&M will require an additional 40 evaluations to arrive to the global minima. There are two clouds of evaluation points around each global minima, points that were generated by the refinement calls to the local solver M&S.

Table 4.2 presents the test problem name, number of variables, global solution

and the solution reported by the solvers. The solvers solutions are the median values of the ten instances solved. Median values are presented instead of averages since there are instances with high objective values that distort the average value.

Figures 4.2 - 4.3 present a ranking of the median solutions by type of problem. The median solutions are ranked, where rank 1 is reserved for the best solution among the 5 solvers compared. The test problems are presented in an order such that the left part of the figures present problems for which the B&M algorithm achieves a good ranking (rank 1 or 2) and the right part presents the remaining problems.

Figure 4.2 presents problems with number of variables 1 to 3. The B&M algorithm was able to rank first in 13 out of the 16 test problems, and was able to solve 10 problems to global optimality, achieving the first rank, although sharing this rank with other solvers that also solved the problem. For the 3 remaining problems, B&M achieved second and third ranks, following only solvers TOMLAB/LGO and TOMLAB/GLB.

Figure 4.3 presents problems with number of variables 4 to 30. The B&M algorithm was able to rank first in 11 out of the 14 test problems, and was able to solve 8 problems to global optimality, achieving the first rank, although sharing this rank with other solvers that also solved the problem. The B&M algorithm outperformed TOMLAB/LGO in 5 of the problems. For the 3 remaining problems, B&M achieved second ranks, following only solver TOMLAB/LGO.

4.5 Conclusions

The Branch and Model (B&M) algorithm was proposed. The algorithm is based on modeling the function using surrogates around each evaluated point, resulting in a collection of surrogate models. The surrogate considers a function of the distance to an evaluated point. The area of influence is determined at each iteration by a partitioning algorithm. The algorithm is designed to avoid fitting and optimizing

Table 4.2: Median solution (10 instances) for test problems.

problem	vars	B&M	NEWUOA	FMINSEARCH	IMFIL	TOMLAB/GLB	TOMLAB/LGO	TOMLAB/RBF
3pk	30	1.10E+04	3.53E+11	8.87E+11	5.60E+07	5.34E+11	8.98E+09	9.00E+04
aircrftb	5	5.61E-01	1.90E+09	1.61E+10	4.93E+11	6.01E-01	0.00E+00	2.30E+01
allinit	3	1.67E+01	2.76E+14	6.05E+12	6.22E+09	1.67E+01	1.67E+01	9.99E+15
arglinb	10	4.63E+00	4.63E+00	4.63E+00	1.85E+04	2.00E+01	4.63E+00	2.00E+01
arglinc	8	6.14E+00	6.14E+00	6.14E+00	1.39E+05	2.00E+01	6.14E+00	2.00E+01
biggs5	3	1.00E-02	3.55E+00	1.66E-01	8.74E+00	8.70E+00	1.50E-02	3.38E+00
box2	5	0.00E+00	1.40E-02	0.00E+00	1.06E+00	9.30E-02	0.00E+00	5.34E-01
box3	2	0.00E+00	1.18E+08	5.00E-03	5.61E+02	0.00E+00	0.00E+00	0.00E+00
brownbs	3	9.80E+11	9.80E+11	9.80E+11	9.81E+11	9.80E+11	9.80E+11	9.81E+11
camel6	2	-1.03E+00	-4.72E-01	-1.03E+00	-1.03E+00	-1.03E+00	-1.03E+00	-1.03E+00
chi	2	-4.33E+01	-3.10E+01	-3.13E+01	-4.31E+01	-4.33E+01	-4.33E+01	-4.29E+01
cliff	2	2.00E-01	2.00E-01	2.00E-01	2.00E-01	2.00E-01	2.00E-01	2.03E-01
denschnb	2	0.00E+00	4.00E-03	2.00E-03	7.19E+02	0.00E+00	0.00E+00	5.00E+00
denschne	2	0.00E+00	1.00E+00	5.00E-01	7.60E+04	0.00E+00	0.00E+00	0.00E+00
denschnf	3	0.00E+00	2.03E+01	6.60E-02	4.91E+06	0.00E+00	0.00E+00	6.40E+01
expfit	2	2.40E+01	2.40E+01	2.41E-01	2.40E+01	2.40E+01	2.85E-01	2.40E+01
genhumps	2	0.00E+00	0.00E+00	0.00E+00	1.18E+01	0.00E+00	0.00E+00	0.00E+00
growth	5	4.97E+01	1.80E+03	3.54E+03	3.45E+03	2.00E+03	1.00E+00	0.00E+00
hart6	6	-3.32E+00	-3.26E+00	-3.26E+00	-3.32E+00	-3.32E+00	-3.32E+00	-3.31E+00
heart8ls	8	2.64E+02	5.62E+22	3.06E+22	1.38E+23	2.64E+02	3.67E+00	2.64E+02
hs003	2	0.00E+00	8.92E+01	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
hs110	10	-4.58E+01	-4.58E+01	-4.46E+01	-4.58E+01	-4.25E+01	-4.58E+01	-2.18E+00
median	1	4.63E+00	4.63E+00	4.63E+00	7.50E+02	4.63E+00	4.63E+00	4.63E+00
osbornea	5	1.00E+00	1.80E+08	4.29E+05	2.68E+03	1.43E+01	5.10E-02	1.43E+01
palmer1	4	2.82E+04	2.82E+04	2.82E+04	3.36E+04	2.86E+04	2.82E+04	4.48E+04
rbrock	2	0.00E+00	0.00E+00	0.00E+00	2.46E+00	0.00E+00	0.00E+00	9.00E-03
s243	3	7.97E-01	7.97E-01	7.97E-01	4.20E+09	7.97E-01	7.97E-01	7.97E-01
s287	20	2.28E+01	3.44E+09	6.88E+15	2.12E+09	2.10E+02	1.44E+10	2.10E+02
s288	20	0.00E+00	5.33E+08	8.05E+13	3.63E+10	0.00E+00	1.23E+10	0.00E+00
st_bsj3	6	-8.68E+04	-5.06E+04	-5.33E+04	-8.68E+04	-8.68E+04	-8.68E+04	-8.68E+04

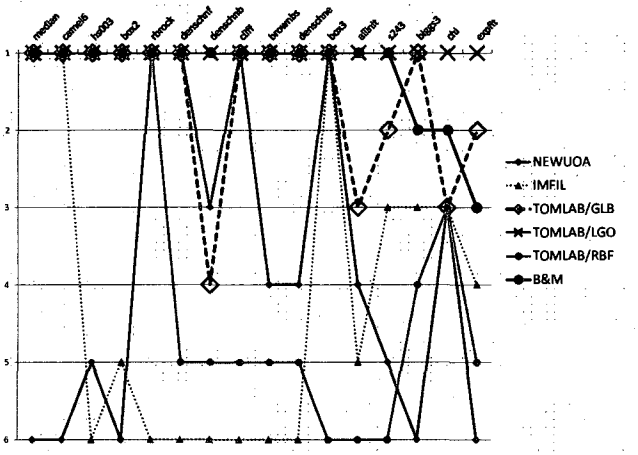

Figure 4.2: Ranking of solvers for problems of type 1.

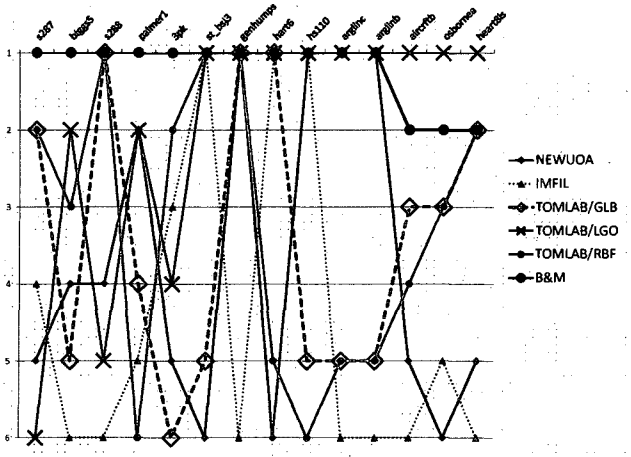

Figure 4.3: Ranking of solvers for problems of type 2.

each partition. Instead, only promising areas are analyzed and later evaluated. A scheme is proposed to ensure that the search is dense and thus, to converge to a global minimum in the limit. Extensive descriptions of the algorithms proposed were presented and the algorithm was implemented in MATLAB. Computational studies over a publicly available set of problems were presented. Algorithm B&M was able to solve to global optimality 18 out of the 30 problems, while obtaining better solutions than TOMLAB/LGO for 5 problems.

Chapter 5

Molecular docking by derivative-free optimization solvers

5.1 Introduction

Protein-ligand docking is pivotal to drug discovery and structure-based drug design. As a result, there has been a strong interest in developing algorithms for predicting the three dimensional arrangement of a protein and a ligand. In particular, various search methods have been proposed to find global minima of a scoring function over a multimodal feasible space.

In this chapter, we investigate the use of derivative-free optimization solvers to guide the search in docking problems. This includes the use of the `B&M` algorithm proposed in earlier chapters. The docking problem solved requires finding the best docking position between a receptor and a ligand, both of which are assumed to be rigid. The objective function was the score obtained by the change in binding free energy of the protein-ligand complex computed by `AutoDock 4`. The problem involves six variables, which completely determine the position and orientation of the ligand, allowing three variables for translation and three variables for rotation. The test problem set included twelve complexes from a broad class of families.

Computational experiments showed that DFO solvers obtained conformations better than the state-of-the-art `AutoDock` when the initial solution was close to the optimal solution. A two-phase scheme using `AutoDock` first, followed by DFO solvers to refine the solution was found to give good conformations when the initial solution was far from the optimal solution.

The results presented in this chapter are from joint work with S. Awasthi. A subset of the computational results presented here has appeared in Awasthi's MS thesis [12].

5.2 Background

The docking problem consists of predicting the binding arrangement of ligand molecules with a protein target, where potential conformations are evaluated by means of scoring functions. Initially proposed during the 1980s [82], molecular docking is used in the virtual screening of drug candidates and has contributed to the discovery of recent drugs. Improvements in X-ray crystallography and NMR spectroscopy have resulted in accurate predictions of the three dimensional structure of proteins, structures that are integral part in molecular docking.

Fueled by an interest of the pharmaceutical industry and life sciences firms, various docking algorithms have been developed and tested. The scoring function used to compare solutions is typically the binding free energy. Today challenges in the field include the development of a consensus model for scoring protein-ligand conformations, and the development of search techniques for the optimization problem. The optimization problem is difficult in nature due to the absence of a closed form expressions for the most reliable scoring functions and their multimodal landscape.

Currently, more than 20 docking programs are available involving the use of various search algorithms and scoring schemes. Table 5.1 lists the most cited programs.

5.3 Scoring functions

Scoring functions evaluate and rank ligand conformations that show some kind of interaction with the receptor. Scoring functions help to identify valid ligand con-

Table 5.1: Most cited docking programs (from ISI Web of Science).

AutoDock [100]
GOLD [74]
FlexX [117]
ICM [2]
DOCK [46]
LUDI [22]

formations. Scoring functions are based on models that involve various assumptions and simplifications. Halperin *et al.* [56] classified scoring schemes by their purpose in a search. The first class involves schemes with cheap evaluations and based on empirical assumptions. These functions are targeted for use with global searches, where filtering of regions is important and accuracy is not required. The second class involves more detailed schemes that include free energy simulations and is targeted to refine the search in local neighborhoods. Currently, there is no consensus on scoring schemes. The docking community has often criticized simplifications such as disregarding physical phenomena [56, 85, 136]. Three types of scoring schemes are implemented in docking tools: *force-field based scoring*, *empirical scoring functions*, and *knowledge-based scoring schemes.* For details of these functions, we refer to a recent review in [12].

5.4 Search algorithms

Search algorithms are used to explore the protein conformational space. Both the protein and the ligand are allowed to take a continuum of 3D orientations. Discretization of the search space followed by evaluation of these points is impractical. The most common search algorithms include *stochastic methods*, *systematic searches*, and *simulation methods.*

5.4.1 Stochastic methods

Stochastic methods rely on making random changes to a ligand or a population of ligands. The new ligand or population is then evaluated on the basis of a pre-defined probability function.

Monte Carlo (MC) algorithms contain a stochastic process and employ a metropolis criterion to accept or reject the randomly changed configuration. `AutoDock` [100] is among the various docking programs that use MC methods.

Genetic Algorithms (GA) begin with an initial sample of solutions and use genetic operators such as *mutations* and *crossovers* to obtain the next generation of solutions. The state of the ligand corresponds to the genotype, and its atomic coordinates represent the phenotype. Genetic operators are used to obtain a new population, which is then scored and ranked. `AutoDock` [100] uses a Lamarckian Genetic Algorithm (LGA) for automated docking.

Tabu search uses a list of prohibited moves that act to reduce the search spaces, avoiding previously generated solutions. Implemented in the context of stochastic search methods, tabu search involves making random changes to populations and obtaining new populations.

5.4.2 Systematic methods

Systematic methods attempt to search the complete configurational space, usually after discretization of the degrees of freedom, resulting in a large number of combinations [79]. To reduce the search space, ligands are often incrementally grown in the binding site of the protein. Fragment-based methods divide a ligand into various fragments that are docked into the active site and finally covalently linked. These methods are subjective. They require the choice of a base fragment, which can significantly alter the final result. `FlexX` [117] and `MATLAB` [46] use fragment-based methods with different fragment generation strategies.

Point complementarity methods represent the interacting molecules using spheres or cubes to model atoms. The molecular surface is represented by surface and volume cubes. The ligand cube is rotated and translated such that it maximizes the difference between the number of ligand and protein surface matches.

In distance geometry methods, a ligand binding to a certain receptor is considered as a collection of ligand points. Binding modes in distance geometry methods consider hydrogen bonding as the only factor for determining these modes. `DockIt` [21] uses distance geometry techniques to generate ligand conformations while representing the active site by a set of spheres.

5.4.3 Simulation methods

Molecular Dynamics (MD) methods solve Newton's equations of motion. Forces on each atom are calculated based on the change in the potential energy. Unlike stochastic methods, such as MC and GA, MD methods are deterministic in nature. MD methods often are unable to escape local minima and are usually complemented with other stochastic techniques. `DOCK` [46] uses MD simulations.

5.5 Computational Results

Twelve protein-ligand complexes were tested under two different initialization approaches using fifteen derivative-free solvers described earlier in the thesis as the search algorithm. Computational experiments were performed on an Intel Core 2 Duo 2.13 GHz processor with 3.2 GB RAM running Fedora 8. The methodology, test problems, and results are discussed below.

5.5.1 Methodology

The docking problem is formulated as an unconstrained optimization problem using the scoring function AD4 that was minimized using the derivative-free solvers described earlier in the thesis. The objective function is the change in binding free energy, ΔG_{bind}, which is minimized over six degrees of freedom:

$$\min_{x,y,z,\theta_x,\theta_y,\theta_z} \quad \Delta G_{\text{bind}}$$

where, x, y and z are the translations of the Center of Mass (CM) of the ligand. The rotations are carried out by performing the following steps:

- Calculate the geometric center of the ligand:

$$\begin{aligned} x_{\text{center}} &= \frac{\sum_{i=1}^{N} x_i}{N} \\ y_{\text{center}} &= \frac{\sum_{i=1}^{N} y_i}{N} \\ z_{\text{center}} &= \frac{\sum_{i=1}^{N} z_i}{N} \end{aligned}$$

  where x_i, y_i and z_i are the cartesian coordinates of the i^{th} atom of the ligand, respectively, and N is the number of atoms in the ligand.

- Translate the geometric center to the origin.

- Perform the rotations θ_x, θ_y, θ_z around the corresponding axes.

- Translate the geometric center back to its original position.

The optimum conformation of the ligand obtained by using this process was compared with the conformation obtained by performing docking by `AutoDock`.

Table 5.2: Characteristics protein-ligand complexes in the test set.

PDB code	Protein-ligand complex	Protein family	ΔG^{exp}_{bind} (kJ/mol)
1HBV	HIV-1 protease-SB203238	Aspartic Protease	-36.32
1HPV	HIV-1 protease-VX-478	Aspartic Protease	-52.61
1HTV	HIV-1 protease-GR126045	Aspartic Protease	-46.18
1BRA	Trypsin mutant-benzamidine	Serine Protease	-10.41
1TNH	Trypsin-4-fluorobenzylamine	Serine Protease	-19.21
3PTB	Trypsinbenzamidine	Serine Protease	-27.04
1CBX	CPA-L-benzylsuccinate	Metalloprotease	-36.21
6CPA	CPA-ZAAP(O)F	Metalloprotease	-65.74
7CPA	CPA-BZ-FVP(O)F	Metalloprotease	-79.64
2ABH	Phosphate-binding protein	Phosphate-binding	-37.13
1ULB	PNP-guanine	Pentosyltransferase	-30.24
2CMD	Malate dehydrogenase-citric acid	Dehydrogenase	-26.10

5.5.2 Test set: Protein-ligand complexes

Protein-ligand complexes from various families were selected from the PDB website [18]:

1. Aspartic proteases (three complexes)
2. Serine proteases (three complexes)
3. Metalloprotease (three complexes)
4. Other proteins (three complexes)

These complexes were chosen because they are well studied with reported experimental binding energy [45], binding site and other relevant details. Characteristics of these complexes are provided in Table 5.2.

5.5.3 Application of derivative-free optimization

Table 5.3 present the solvers compared on the docking problem.

Table 5.3: Solvers used in computational experiments.

Branch and Model (B&M) (Chapter 3.5)
Adaptive Simulated Annealing (ASA) [69]
DAKOTA/APPS [43]
DAKOTA/EA [43]
DAKOTA/DIRECT [43]
DAKOTA/PATTERN [43]
DAKOTA/SOLIS-WETS [43]
Derivative-Free Optimization (DFO) [26, 125]
IMFIL [76]
Multilevel Coordinate Search (MCS) [104]
FMINSEARCH [83]
NEWUOA [115]
Nonlinear Optimization using Mesh Adaptive Direct searches (NOMAD) [34]
PSWARM [134]
SID-PSM [38]

5.5.4 Approach I—Proof-of-concept

The objective of this section is to test the ability of derivative-free solvers as search algorithms in protein-ligand docking. The initial position of the ligand was chosen randomly near the binding site. The solvers were tested to determine if they could trace the energy minimum.

AutoDock used a Lamarckian Genetic Algorithm (LGA) with a minimum of 2.5e+7 energy evaluations per run. For each complex, ten AutoDock runs were performed and the best result was used for comparison with other solvers.

Tables 5.4 – 5.7 present optimal ΔG_{bind} values obtained by the solvers for different families of complexes. The first two rows contain solutions for Autodock and the B&M. The following 14 rows contain optimal results obtained by the solvers described in Section 5.5.3. Figures 5.1 – 5.4 compare the optimal values graphically. For convenience, positive ΔG_{bind} values are not graphed since they are far away from the optimal ΔG_{bind} values.

The ΔG_{bind} values for aspartic proteases, *i.e.*, for 1HBV, 1HPV and, 1HTF are

Table 5.4: ΔG_{bind} (in kJ/mol) for aspartic proteases.

Solver	1HPV	1HBV	1HTF
AutoDock	-34.4	-32.8	-41.5
B&M	-38.2	-33.4	-35.1
ASA	2420.2	2484.1	7727.6
DAKOTA/APPS	-38.8	-32.7	-36.5
DAKOTA/DIRECT	-38.6	-37.3	-35.7
DAKOTA/EA	12.8	254.9	22.7
DAKOTA/PATTERN	-38.8	-27.7	-36.5
DAKOTA/SOLIS-WETS	-4.8	-22.5	-30.7
DFO	-36.2	-29.6	-33.1
FMINSEARCH	-37.1	-39.1	-36.4
IMFIL	-38.8	-33.4	-36.7
MCS	-39.0	-38.3	-36.5
NEWUOA	-34.9	-27.2	-33.4
NOMAD	-38.8	-33.4	-36.5
PSWARM	-38.0	-24.2	-26.7
SID-PSM	-39.0	-39.0	-36.6

reported in Table 5.4. SID-PSM, FMINSEARCH, and IMFIL were the best performers for each of the three complexes.

SID-PSM obtained the best result for complex 1HPV, with 8 other solvers within 1 kJ/mol, including B&M. FMINSEARCH found the best solution for complex 1HBV, which was 20% better than the value obtained by AutoDock. For both 1HPV and 1HBV complexes, the optimal solution for B&M also outperforms AutoDock. AutoDock obtained the best ligand conformation for 1HTF, outperforming the best derivative-free solver IMFIL by almost 5 kJ/mol. Furthermore, 1HTF is the only complex out of the 12 complexes selected for which AutoDock was able to obtain the best conformation. Among the worst performers were ASA and COLINY EA. Their optimal ΔG_{bind} values were positive and were not shown in the Figure 5.1.

The ΔG_{bind} values for serine proteases, *i.e.*, for 1BRA, 3PTB, and 1TNH are reported in Table 5.5. For these complexes, PSWARM and FMINSEARCH proved to be the best performers.

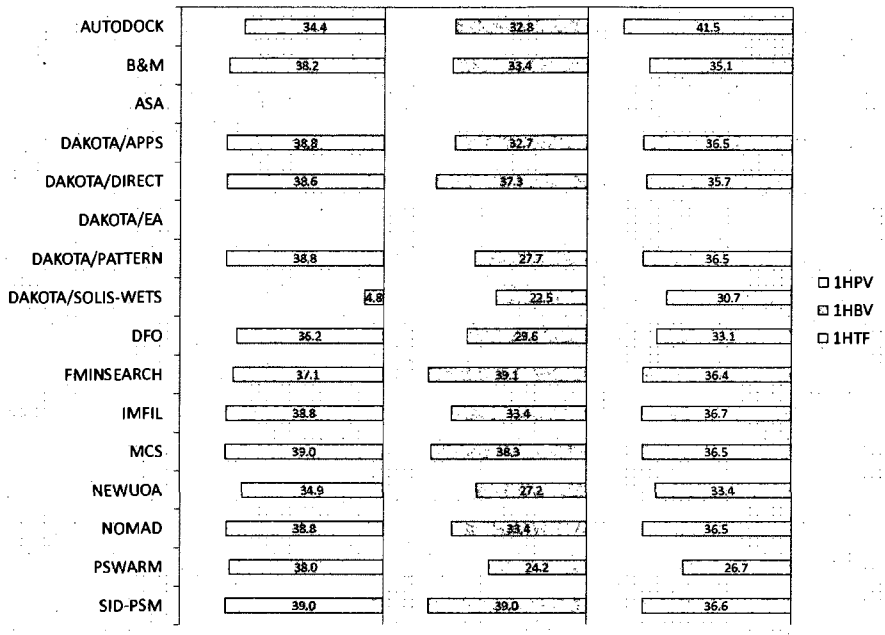

Figure 5.1: Comparison of solvers for aspartic proteases.

B&M and PSWARM obtained the best solution for complexes 1BRA and 1TNH respectively , whereas Autodock was only able to obtain a positive ΔG_{bind} value for complex 1BRA. Moreover, AutoDock was the worst performer among all the solvers for all 3 serine complexes, as presented in Figure 5.2. FMINSEARCH found the best solution for complex 3PTB, with 11 other solvers within 0.5 kJ/mol, including B&M.

The maximum scores obtained for 3PTB and 1TNH represent almost 100% improvement over the score obtained by AutoDock. Among derivative-free solvers, COLINY EA and ASA were the worst performers. However, both solvers converged to a feasible energy minimum for all the complexes.

The ΔG_{bind} values for metalloproteases, *i.e.*, for 1CBX, 6CPA, and 7CPA are reported in Table 5.6. DAKOTA/EA and MCS obtained the best solution for 1CBX and 6CPA, respectively. AutoDock was the worst performer among all the solvers and did not converge to a minimum for any of the three protein-ligand complexes. B&M obtained the best result for the complex 7CPA. Among the other solvers, COLINY

Table 5.5: ΔG_{bind} (in kJ/mol) for serine proteases.

Solver	1BRA	3PTB	1TNH
AutoDock	5.2	-10.7	-12.8
B&M	-22.1	-23.7	-24.1
ASA	-15.0	-16.3	-18.8
DAKOTA/APPS	-22.2	-23.8	-25.2
DAKOTA/DIRECT	-24.1	-23.8	-25.0
DAKOTA/EA	-18.7	-20.0	-16.1
DAKOTA/PATTERN	-22.2	-23.8	-25.2
DAKOTA/SOLIS-WETS	-21.8	-22.9	-22.0
DFO	-21.3	-23.6	-24.9
FMINSEARCH	-22.7	-23.8	-24.8
IMFIL	-22.2	-23.8	-25.3
MCS	-22.8	-23.7	-24.9
NEWUOA	-21.5	-23.7	-25.0
NOMAD	-22.2	-23.8	-24.7
PSWARM	-24.3	-23.5	-25.7
SID-PSM	-24.0	-23.8	-25.4

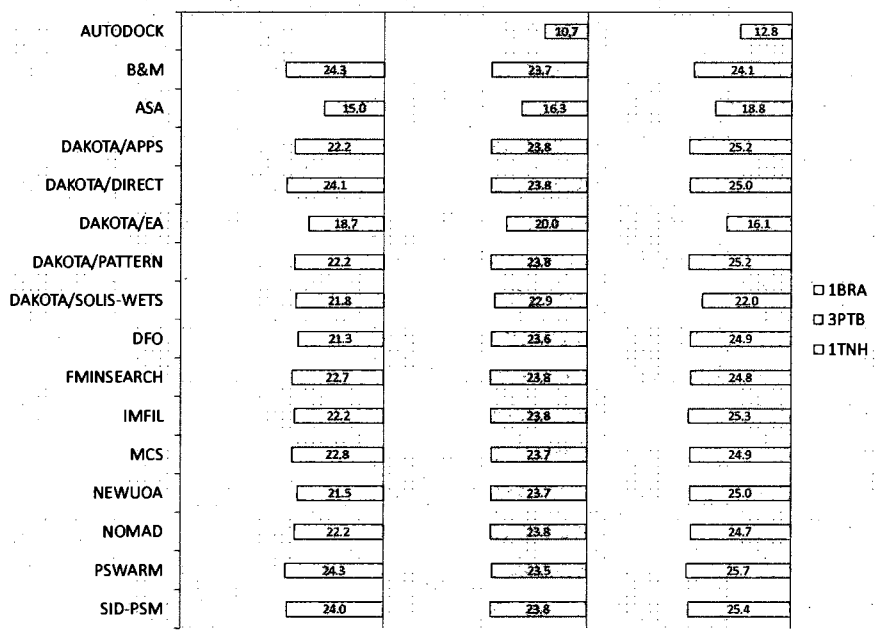

Figure 5.2: Comparison of solvers for serine proteases.

Table 5.6: ΔG_{bind} (in kJ/mol) for Metalloproteases.

Solver	1CBX	6CPA	7CPA
AutoDock	0.0	0.0	0.0
B&M	-17.1	-28.6	-32.4
ASA	3.0	37.6	878.4
DAKOTA/APPS	-16.2	-30.2	-32.3
DAKOTA/DIRECT	-17.2	-29.6	-32.2
DAKOTA/EA	-28.5	41.6	139.5
DAKOTA/PATTERN	-16.3	-28.4	-30.1
DAKOTA/SOLIS-WETS	-13.5	31.3	3.0
DFO	-15.3	-27.4	-28.6
FMINSEARCH	-17.5	-30.1	-29.2
IMFIL	-16.3	-30.1	-32.2
MCS	-16.3	-31.3	-32.2
NEWUOA	-17.2	-23.6	-27.3
NOMAD	-16.3	-30.2	-30.7
PSWARM	-16.4	-21.5	-23.9
SID-PSM	-17.3	-5.6	-30.7

SOLIS-WETS and COLINY EA did not perform well for these metalloproteases and gave positive ΔG_{bind} values for 6CPA and 7CPA. Three other complexes, all from different protein families, were included in the test set and presented in Table 5.7. B&M obtained the best result for complex 2ABH, whereas AutoDock obtained the second best result, outperforming 8 derivative-free solvers. PSWARM and SID-PSM had the best result for complexes 2CMD and 1ULB, respectively. All derivative-free solvers converged to a ΔG_{bind} that corresponds to a feasible docking mode, as shown in Figure 5.4.

5.5.5 Approach II

Approach II was used to test the solvers when the starting ligand position is far from the binding site. 2ABH was randomly picked from the test set and the ligand was randomly moved far from the binding site. The initial position of the ligand was chosen randomly far from the binding site. The results obtained by using all the solvers have been compiled in Table 5.8.

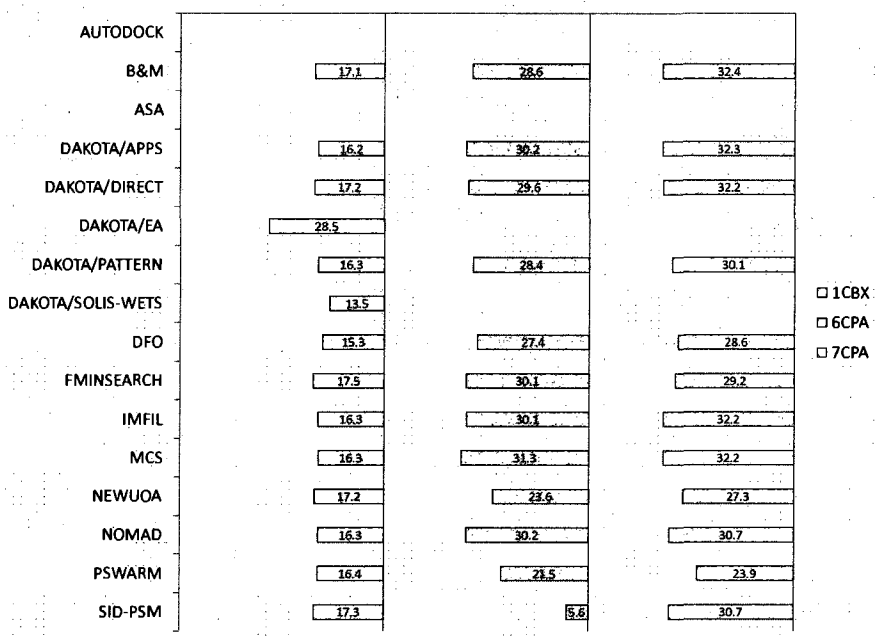

Figure 5.3: Comparison of solvers for Metalloproteases complexes.

Table 5.7: ΔG_{bind} (in kJ/mol) for other complexes.

Solver	**2ABH**	**2CMD**	**1ULB**
AutoDock	-22.5	-17.1	31.5
B&M	-24.4	-32.6	-23.7
ASA	-15.8	-10.0	-21.8
DAKOTA/APPS	-23.0	-32.1	-26.7
DAKOTA/DIRECT	-23.7	-32.5	-26.3
DAKOTA/EA	-21.8	-20.3	-21.9
DAKOTA/PATTERN	-20.6	-32.1	-27.0
DAKOTA/SOLIS-WETS	-11.6	-31.2	-23.8
DFO	-22.0	-31.5	-24.4
FMINSEARCH	-21.0	-31.4	-24.2
IMFIL	-23.5	-32.0	-26.6
MCS	-23.6	-32.4	-26.7
NEWUOA	-22.1	-31.6	-24.6
NOMAD	-23.4	-32.1	-26.7
PSWARM	-22.7	-32.9	-26.6
SID-PSM	-22.1	-32.6	-26.8

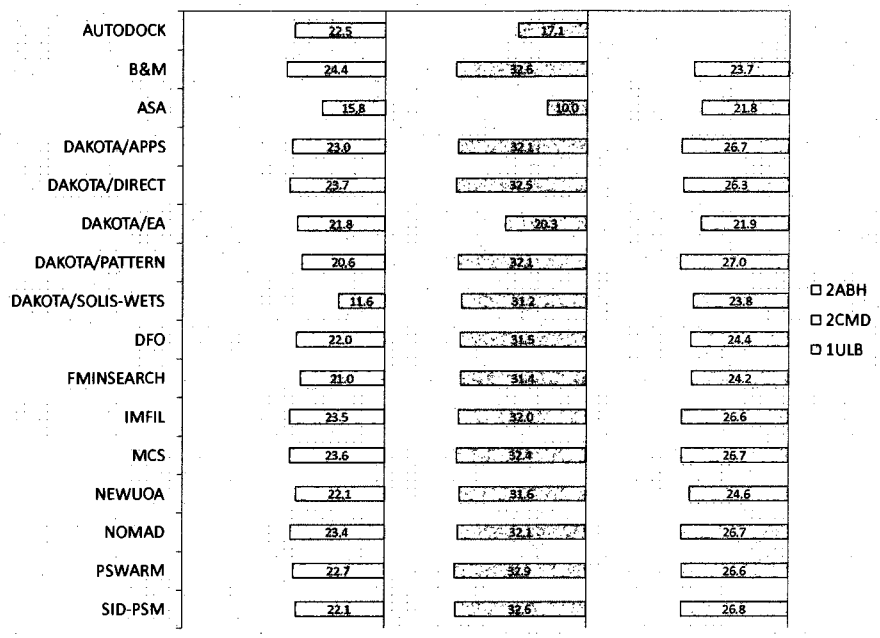

Figure 5.4: Comparison of solvers for other complexes.

All solvers converged to a local minimum but none reached the best known minimum for the known ligand conformation. The ligand in 2ABH is a phosphate ion, the smallest ligand among the ligands in the test set. The reason for `AutoDock`'s ability to find the optimum binding mode may be the small size of the ligand.

The ligand conformation obtained by `AutoDock` was then used as the starting point and the results from derivative-free solvers were analyzed. All solvers converged to the same values as shown in Table 5.7.

5.6 Conclusions

This chapter investigated the use of derivative-free optimization solvers as search algorithms for protein-ligand docking. Derivative-free solvers were applied on twelve different complexes with known 3D structures. Computational experiments were conducted to compare the performance of these solvers, and compared them to the

Table 5.8: ΔG_{bind} (in kJ/mol) for 2 ABH by starting the ligand far from the binding site.

Solver	2ABH
AutoDock	-22.49
B&M	-19.61
COLINY APPS	-3.77
COLINY DIRECT	-3.19
COLINY EA	-3.82
COLINY PATTERN	-3.27
COLINY SOLIS-WETS	-4.82
DFO	-2.74
FMINSEARCH	-3.37
IMFIL	-3.76
MCS	-1.28
NEWUOA	-2.78
NOMAD	-3.73
PSWARM	-3.68
SID-PSM	-3.42
ASA	-2.96

stand-alone software AutoDock.

Two different approaches were taken. The first approach aimed to prove that these solvers can obtain better scores or better poses than those obtained by AutoDock. For ligands near the binding site, most of the solvers converged to ΔG_{bind} values better than the values obtained by AutoDock. B&M, FMINSEARCH, MCS, PSWARM and SID-PSM performed consistently better than AutoDock. In particular, B&M outperformed AutoDock in 11 out of the 12 complexes. B&M obtained the best conformation for 3 out of the 12 complexes, followed by PSWARM, FMINSEARCH and SID-PSM, each with 2 best conformations out of the 12 complexes.

ASA and COLINY EA did not perform well and gave positive ΔG_{bind} values for some complexes. Moreover, for cases where these two solvers obtained feasible ligand configurations corresponding to a negative ΔG_{bind}, the values were much higher than the ones obtained by other solvers.

For cases where the starting ligand position was far from binding site, all solvers

converged to a local minimum but failed to converge to the best known minimum. To obtain better minima with these solvers, `AutoDock` was used to perform an initial docking and provide a ligand conformation which was later used by these solvers as the starting point. For cases where `AutoDock` converged to a negative ΔG_{bind}, these solvers were able to get a better configuration and, in some cases, even the best known minimum.

Based on the results obtained by using the two different approaches, we can conclude that the derivative-free solvers studied can be used to predict a docked complex. `AutoDock` can be employed in a first phase to explore the feasible space, whereas the other solvers can be used in a second phase to refine the solution.

Chapter 6

Conclusions

This thesis presented a comprehensive study of derivative-free algorithms and software, proposed two new algorithms for derivative-free optimization, and applied derivative-free algorithms to the problem of protein-ligand docking.

Twenty existing implementations were tested over a publicly available problem set composed of 236 problems. This computational comparison represents the first effort of such a large magnitude in this area, both in terms of the number of test problems and the number of solvers considered. In the process of performing this computational comparison, we developed a unified interface to derivative-free solvers that will be made available at `http://eudoxus.scs.uiuc.edu/dfo/`.

A new derivative-free optimization algorithm, Branch and Model (B&M), was proposed. This algorithm is based on modeling the function of interest around each evaluated point, using information from other nearby evaluated points. B&M was shown to perform a dense search and thus converge in the limit to a global minimum. A local search algorithm, Model and Search (M&S), was also proposed to refine the best solution found by B&M. The M&S algorithm is employed after a given number of evaluated points in the course of B&M fail to improve the best known solution. The search in M&S is guided by identifying descent directions from a quadratic model fitted around the best known point using information from other evaluated points. Algorithm M&S was shown to converge to a stationary point. Both algorithms were implemented in MATLAB. Computational studies over a publicly available set of problems were presented and displayed competitive performance compared to other available

solvers.

Finally, B&M and other derivative-free solvers were employed as search algorithms for protein-ligand docking. These solvers were applied to twelve different protein-ligand complexes with known 3D structures and the results were compared to those obtained by AutoDock, a state-of-the-art software for protein-ligand docking. Most of the derivative-free solvers performed very well when the ligand is initialized near the binding site. Some of the solvers were able to give conformations corresponding to the best known energy minimum. Most of the solvers outperformed AutoDock. In particular, B&M outperformed AutoDock in 11 out of the 12 complexes. Compared to the other derivative-free solvers currently available, B&M obtained the best conformation for 3 out of the 12 complexes, followed by PSWARM, FMINSEARCH and SID-PSM each of which obtained the best conformations for 2 out of the 12 complexes.

Future work in derivative-free optimization could consider a number of meaningful directions:

1. Most of the algorithms considered in this thesis were designed for unconstrained optimization. Constrained optimization problems naturally arise in science and industry. Typically, derivative-free solvers are extended to constrained problems via a penalty function. Constrained problems should be studied more extensively and alternative algorithms to treat constraints should be studied systematically.

2. The derivative-free solver M&S purposed seeks to refine solutions by searching through directions obtained from a model. This model uses previously evaluated points around a sphere centered at the best known point. The radius of the sphere is evidently affected by the scale of the variables. It is customary to scale the feasible space to a hypercube $[0, 1]$. Since the user often has little knowledge of sensible bounds for the variables, very large bounds are often provided. Hence, scaling to a hypercube $[0, 1]$ is not guaranteed to provide a

better scale than the original scale of the problem. A dynamic scaling method performed during the optimization process should be explored as a means to improve convergence.

3. Algorithm B&M approximates the real problem function by using a collection of models around each evaluated point. Each model is assumed valid and optimized over a hypercube. The algorithm used to partition the feasible space is of fundamental importance since it determines the scope of the next evaluated points. Partitioning schemes should be explored to emphasize regions with high potential.

4. There are numerous applications in science and engineering that could be tackled with the algorithms considered in this thesis. In the context of the docking problem, an approach that considers the protein and/or ligand as flexible molecules would considerably increase the complexity of the problem and could serve as a further application of the proposed techniques.

Appendix A

Interface for derivative-free optimization solvers

A.1 Introduction

`mydfo` is a `bash` script that provides a unified interface to various derivative-free black-box optimization solvers and is available at `http://eudoxus.scs.uiuc.edu/dfo/`.

The problem to be solved is of the form:

$$\min f(x)$$
$$l < x < b$$

where $f(x)$ is a black-box function that is minimized over the hypercube defined by the vectors l and b.

The `mydfo` usage is:

`mydfo[algorithm][executable][problemdata][input][output][parameters][printopt]`

The call to `mydfo` produces a standard output message and three result files. The standard output message has the following format:

[`executable`] algorithm :[`algorithm`][`minimum`][`calls`][`best_call`]

The [`minimum`] value is the best objective function value found by the solver. During

the optimization process, a total of [calls] function evaluations were used, obtaining a best objective value of [minimum] at the [best_call] attempt.

The optimization results are written into three files. The best objective function and the best set of variable values are stored respectively in files best_objective and best_solution. A file evals.res contains more details about the progress of the optimization, where each line starts with the function evaluation attempt (integer), the best point evaluated (float array), and the incumbent objective function value. The user can set the number of records printed through [printopt].

The following is a description of the arguments required by mydfo:

A.1.1 [algorithm]

The user needs to provide an integer number corresponding to the following list of supported solvers:

1. ASA [69]
2. NOMAD [34]
3. DFO [125]
4. NEWUOA [114]
5. FMINSEARCH
6. IMFIL [76]
7. MCS [104]
8. SID-PSM [38]
9. PSWARM [134]
10. DAKOTA/APPS [44, pp. 60]

11. DAKOTA/DIRECT [44, pp. 61–63]

12. DAKOTA/EA [44, pp. 63–67]

13. DAKOTA/PATTERN [44, pp. 67–69]

14. DAKOTA/SOLIS-WETS [44, pp. 70]

15. SNOBFIT [67]

16. TOMLAB/EGO [62, pp. 11–24]

17. TOMLAB/GLBSOLVE [61, pp.125–127]

18. TOMLAB/GLCSOLVE [61, pp.142–146]

19. TOMLAB/LGO [111]

20. TOMLAB/RBFSOLVE [62, pp.5–10]

21. B&M

22. M&S

A.1.2 [executable]

The user needs to provide the name of a standalone program that reads the input parameter values from the [input] file, evaluates the black-box function of interest, and writes the objective function value into an [output] file. The program is expected to run by a system call './[executable]'.

A.1.3 [problemdata]

The file [problemdata] contains various problem features in the following format:

- number of variables (integer)

- lower bounds (float array)
- upper bounds (float array)
- starting point (float array)

The following is an example for a 2-variable problem, where the first variable has bounds $[-10, 15]$ and starting value 0, and the second variable has bounds $[-20, 30]$ and starting value 5.

problemdata	
2	
-10	-20
15	30
0	5

A.1.4 [input]

The format for the [input] file is one variable value per row. The following is an example for a 2-variable problem:

input
4.999
5.999

A.1.5 [output]

The [output] file should only contain the objective function value. As the majority of solvers expect a numerical value as a result from the function evaluation and to prevent the solver from unexpected termination, the standalone program should contain mechanisms to return a value even when difficulties arise while evaluating the function. The following is an example of an [output] file:

output
3.639295166e-12

A.1.6 [parameters]

The [parameters] file contains a set of parameter values as follows:

- limit on number of function calls (integer)
- limit on variable bounds (float)
- limit on constraints bounds (float)
- tolerance on global solution (float)
- tolerance on feasibility (float)
- penalty for non-satisfied constraints (float)

Solvers are limited to a given number of function calls, though solvers may evaluate a few more function calls beyond this limit. The limit on variable bounds is a large value ϕ that is used to set the variable bounds within a $[-\phi, \phi]$, overwriting bounds provided in [problemdata]. The remaining parameters (limit on constraint bounds, tolerance on global solution, tolerance on feasibility and penalty for non-satisfied constraints) are not yet implemented and are expected to be used for constrained black-box solvers.

A.1.7 [printopt]

[printopt] (integer) is an optional parameter that specifies the detail level of the file evals.res. Only when the number of function evaluations is a multiple of [printopt] are records included in evals.res.

A.2 Interfaces

The interfaces are designed to be non-intrusive to the optimization operations of each solver. Each interface reads the user-provided problem information, produces the input required by the solver, runs the solver, and collects the results. The interfaces do not interfere while the solver is running.

The interface starts by reading the `[problemdata]` and `[parameters]` files. The starting point, lower and upper bounds are verified to be within $[-\phi, \phi]$. If they are not, their values are changed to the nearest bounds.

The black-box executable is used through a wrapper. The heterogeneous nature of the solvers, particularly on their features to control the function calls, was the motivation to use wrappers. Instead of directly calling the executable, the solvers call the wrapper that in turn calls the executable. The wrapper counts the number of function calls through a variable `iteration.number`. Upon evaluation, if the objective is the better than the incumbent, the incumbent is updated in variable `best_objective` and the `iteration.number` value is stored in variable `best_iteration`. If `iteration.number` is a multiple of `[printopt]`, then `iteration.number`, the last evaluated point and `best_objective` are printed in one line of the file `evals.res`.

Interface details for particular solvers are presented in the sequel.

A.2.1 ASA, DFO and NEWUOA

The interface writes an auxiliary file which includes the limit of function calls, number of variables, bounds, starting point and the name of input and output files. This file is later read by a program obtained from compiling the source code along with a custom main program and a custom makefile.

A.2.2 NOMAD

The interface writes a description, bounds, starting point and preferences files that are given as input to the NOMAD executable batch_nomad.

A.2.3 MATLAB solvers

Interfaces for MATLAB solvers (FMINSEARCH, IMFIL, MCS, SID-PSM, PSWARM, SNOBFIT and TOMLAB solvers) start by writing a m-file MATLAB program that prepares the input arguments of the solvers function. Additional MATLAB paths are included in this file and added at runtime to the current MATLAB path. The wrappers have the form of a m-file MATLAB function with the same functionalities. Additional function files are produced for SID-PSM for the evaluation of constraints and the gradient of the constraints. Since the only constraints are variable bounds, these additional files are simple functions.

A.2.4 DAKOTA solvers

The interface writes an input file for DAKOTA that specifies multiple solver parameters, including the choice of the solver, its optional parameter values, limit of function calls, number of variables, bounds, starting point and name of input and output files.

A.3 Installation

The user is expected to place mydfo in a directory included in their PATH. The solvers need to be installed and included in the PATH. In the case of MATLAB solvers, the user needs to modify the MATLAB PATH to include the solvers directories. In addition, mydfo permits the option to provide directories and include them in the MATLAB PATH at runtime.

A.4 Exit codes

Upon incurring an error, `mydfo` terminates and returns the corresponding exit code:

1 Incomplete set of input arguments

2 algorithm choice is not available, outside of the range $[1, 22]$

7 `[printopt]` is invalid or negative

13 `[problemdata]` file not found

16 parameters file not found

A.5 Additional files

`ASA`, `DFO` and `NEWUOA` distributions contain source code which requires a separate program or routine to specify features of the black-box function to optimize, such as number of variables, bounds, method of use, among others. Makefiles and additional source code files for these programs and routines are provided. Instructions on how to compile them are provided in the file `INSTALL_SCRIPT`.

Finally, `modtoblack` is an AMPL solver that produces a black-box standalone executable program and its corresponding `[problemdata]` file from an AMPL model in .mod format. `modtoblack` consists of a main program, which provides input, output, and execution for the `nlc` function by Gay [53] that reads the .mod file.

References

[1] E. H. L. Aarts and P. J. M. van Laarhoven. Statistical cooling: A general approach to combinatorial optimization problems. *Phillips Journal of Research*, 40:193–226, 1985.

[2] R. Abagyan, M. Totrov, and D. Kuznetzov. ICM–A new method for protein modeling and design: Applications to docking and structure prediction from the distorted native conformation. *Journal of Computational Chemistry*, 15:488–506, 1994.

[3] M. A. Abramson. *Pattern search algorithms for mixed variable general constrained optimization problems.* PhD thesis, Department of Computational and Applied Mathematics, Rice University, Houston, TX, August 2002.

[4] M. A. Abramson. NOMADm *version 4.5 User's Guide.* Air Force Institute of Technology, Wright-Patterson AFB, OH, 2007.

[5] M. A. Abramson, T. J. Asaki, J. E. Dennis Jr., K. R. O'Reilly, and R. L. Pingel. Quantitative object reconstruction via Abel transform X-ray tomography and mixed variable optimization. Technical report, Department of Computational and Applied Mathematics, Rice University, 2007.

[6] M. A. Abramson and C. Audet. Convergence of mesh adaptive direct search to second-order stationary points. *SIAM Journal on Optimization*, 17:606–609, 2006.

[7] M. A. Abramson, C. Audet, and J. E. Dennis Jr. Generalized pattern searches with derivative information. *Mathematical Programming*, 100:3–25, 2004.

[8] M. A. Abramson, C. Audet, and J. E. Dennis Jr. Filter pattern search algorithms for mixed variable constrained optimization problems. *Pacific Journal of Optimization*, 3:477–500, 2007.

[9] C. Audet. Convergence results for generalized pattern search algorithms are tight. *Optimization and Engineering*, 5:101–122, 2004.

[10] C. Audet and J. E. Dennis Jr. A pattern search filter method for nonlinear programming without derivatives. *SIAM Journal on Optimization*, 14:980–1010, 2004.

[11] C. Audet and J. E. Dennis Jr. Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on Optimization*, 17:188–217, 2006.

[12] S. Awasthi. Molecular docking by derivative-free optimization solvers. Master's thesis, Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, Pennsylvania, 2008.

[13] P. A. Barros Jr., M. R. Kirby, and D. N. Mavris. Impact of sampling techniques selection on the creation of response surface models. *SAE Transactions–Journal of Aerospace*, 113:1682–1693, 2004.

[14] M. C. Bartholomew-Biggs, S. C. Parkhurst, and S. P. Wilson. Using DIRECT to solve an aircraft routing problem. *Computational Optimization and Applications*, 21:311–323, 2002.

[15] R. R. Barton. Metamodeling: A state of the art review. *Proceedings of the 1994 Winter Simulation Conference*, pages 237–244, 1994.

[16] V. Béchard, C. Audet, and J. Chaouki. Robust optimization of chemical processes using a MADS algorithm. *Optimization and Engineering*, to appear.

[17] C. J. Bélisle, H. E. Romeijn, and R. L. Smith. Hit-and-run algorithms for generating multivariate distributions. *Mathematics of Operations Research*, 18:255–266, 1993.

[18] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne. The Protein Data Bank. *Nucleic Acids Research*, 28:235–242, 2000. `http://www.rcbs.org`.

[19] J. D. Bethke. *Genetic algorithms as function optimizers.* PhD thesis, Department of Computer and Communication Sciences, University of Michigan, Ann Arbor, MI, 1980.

[20] M. Björkman and K. Holmström. Global optimization of costly nonconvex functions using radial basis functions. *Optimization and Engineering*, 1:373–397, 2000.

[21] J. M. Blaney and J. S. Dixon. ***DockIt**, version 1.0.* Metaphorics, LLC, Mission Viejo, CA. `www.metaphorics.com/products/dockit.html`.

[22] H. J. Bohm. The computer program `LUDI`: A new method for the de novo design of enzyme inhibitors. *Journal of Computer-Aided Molecular Design*, 6:61–78, 1992.

[23] A. Boneh and A. Golan. Constraints' redundancy and feasible region boundedness by random feasible point generator (RFPG). In *Third European Congress on Operations Research (EURO III)*, Amsterdam, 1979.

[24] A. J. Booker, J.E. Dennis Jr., P. D. Frank, D. B. Serafini, V. J. Torczon, and M. W. Trosset. A rigorous framework for optimization of expensive functions by surrogates. *Structural Optimization*, 17:1–13, 1999.

[25] T. Chiang and Y. Chow. A limit theorem for a class of inhomogeneous Markov processes. *The Annals of Probability*, 17:1483–1502, 1989.

[26] COIN-OR Project. Derivative Free Optimization, Current as of 20 December, 2008. `http://projects.coin-or.org/Dfo`.

[27] COIN-OR Project. `IPOPT 2.3.x` A software package for large-scale nonlinear optimization, Current as of 20 December, 2008. `http://www.coin-or.org/Ipopt/ipopt-fortran.html`.

[28] A. R. Conn, N. I. M. Gould, and P. L. Toint. A globally convergent augmented lagrangian algorithm for optimization with general constraints and simple bounds. *SIAM Journal on Numerical Analysis*, 28:545–572, 1991.

[29] A. R. Conn, K. Scheinberg, and Ph. L. Toint. On the convergence of derivative-free methods for unconstrained optimization. In M. D. Buhmann and A. Iserles (eds.), *Approximation Theory and Optimization, Tribute to M. J. D. Powell,* Cambridge University Press, Cambridge, UK, pages 83–108, 1996.

[30] A. R. Conn, K. Scheinberg, and Ph. L. Toint. Recent progress in unconstrained nonlinear optimization without derivatives. *Mathematical Programming*, 79:397–345, 1997.

[31] A. R. Conn, K. Scheinberg, and Ph. L. Toint. A derivative free optimization algorithm in practice. *Proceedings of AIAA St Louis Conference*, pages 1–11, 1998.

[32] A. R. Conn, K. Scheinberg, and L. N. Vicente. Geometry of interpolation sets in derivative free optimization. *Mathematical Programming*, 111:141–172, 2008.

[33] A. R. Conn, K. Scheinberg, and L. N. Vicente. *Introduction to derivative-free optimization.* SIAM, 2009.

[34] G. Couture, C. Audet, J. E. Dennis Jr., and M. Abramson. The Nomad Project, Current as of 20 December, 2008. `http://www.gerad.ca/NOMAD/`.

[35] D. D. Cox and S. John. SDO: A statistical method for global optimization. In *Multidisciplinary design optimization (Hampton, VA, 1995)*, pages 315–329. SIAM, Philadelphia, PA, 1997.

[36] A. L. Custódio, J. E. Dennis Jr., and L. N. Vicente. Using simplex gradients of nonsmooth functions in direct search methods. *IMA Journal of Numerical Analysis*, 28:770–784, 2008.

[37] A. L. Custódio and L. N. Vicente. Using sampling and simplex derivatives in pattern search methods. *SIAM Journal on Optimization*, pages 537–555, 2007.

[38] A. L. Custódio and L. N. Vicente. *SID-PSM: A pattern search method guided by simplex derivatives for use in derivative-free optimization.* Departamento de Matemática, Universidade de Coimbra, Coimbra, Portugal, 2008.

[39] C. Davis. Theory of positive linear dependence. *American Journal of Mathematics*, 76:733–746, 1954.

[40] S. N. Deming, L. R. Parker Jr., and M. B. Denton. A review of simplex optimization in analytical chemistry. *Critical Reviews in Analytical Chemistry*, 7:187–202, 1974.

[41] D. di Serafino, S. Gomez, L. Milano, F. Riccio, and G. Toraldo. A genetic algorithm for a global optimization problem arising in the detection of gravitational waves. *Optimization Online Digest*, pages 1–18, 2008.

[42] R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pages 39–43, Nagoya, Japan, 1995.

[43] M. S. Eldred, B. M. Adams, D. M. Gay, L. P. Swiler, K. Haskell, W. J. Bohnhoff, J. P. Eddy, W. E. Hart, J-P Watson, J. D. Griffin, P. D. Hough, T. G. Kolda, P. J. Williams, and M. L. Martinez-Canales. DAKOTA, *A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 4.1 User's Manual.* Sandia National Laboratories, Albuquerque, NM and Livermore, CA, 2007.

[44] M. S. Eldred, B. M. Adams, D. M. Gay, L. P. Swiler, K. Haskell, W. J. Bohnhoff, J. P. Eddy, W. E. Hart, J-P Watson, J. D. Griffin, P. D. Hough, T. G. Kolda, P. J. Williams, and M. L. Martinez-Canales. 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 and Livermore, CA, 2007.

[45] M. D. Eldridge, C. W. Murray, T. R. Auton, G. V. Paolini, and R. P. Mee. Empirical scoring functions: I. The development of a fast empirical scoring function to estimate the binding affinity of ligands in receptor complexes. *Journal of Computer-Aided Molecular Design*, 11:445–445, 1997.

[46] T. J. A. Ewing and I. D. Kuntz. Critical evaluation of search algorithms for automated molecular docking and database screening. *Journal of Computational Chemistry*, 18:1175–1189, 1997.

[47] S. S. Fan and E. Zahara. A hybrid simplex search and particle swarm optimization for unconstrained optimization. *European Journal of Operational Research*, 181:527–548, 2007.

[48] D. E. Finkel and C. T. Kelley. Convergence analysis of the DIRECT algorithm. Technical report, North Carolina State University, 2004.

[49] K. R. Fowler, J. P. Reese, C. E. Kees, J. E. Dennis Jr., C. T. Kelley, C. T. Miller, C. Audet, A. J. Booker, G. Couture, R. W. Darwin, M. W. Farthing, D. E. Finkel, J. M. Gablonsky, G. Gray, and T. G. Kolda. A comparison of derivative-free optimization methods for groundwater supply and hydraulic capture community problems. *Advances in Water Resources*, 31:743–757, 2008.

[50] J. M. Gablonsky. *Modifications of the DIRECT Algorithm.* PhD thesis, Department of Mathematics, North Carolina State University, Raleigh, North Carolina, 2001.

[51] P. Gilmore and C. T. Kelley. An implicit filtering algorithm for optimization of functions with many local minima. *SIAM Journal on Optimization*, 5:269–285, 1995.

[52] GLOBAL Library, Current as of 20 December, 2008. `http://www.gamsworld.org/global/globallib.htm`.

[53] M. X. Goemans. Semidefinite programming in combinatorial optimization. *Mathematical Programming*, 79:143–161, 1997.

[54] G. Gray, T. Kolda, K. Sale, and M. Young. Optimizing an empirical scoring function for transmembrane protein structure determination. *INFORMS Journal on Computing*, 16:406–418, 2004.

[55] H.-M. Gutmann. A radial basis function method for global optimization. *Journal of Global Optimization*, 19:201–227, 2001.

[56] I. Halperin, B. Ma, H. Wolfson, and R. Nussinov. Principles of Docking: An Overview of search algorithms and a guide to scoring functions. *PROTEINS: Structure, Function, and Genetics*, 47:409–443, 2002.

[57] J. Han, M. Kokkolaras, and P. Y. Papalambros. Optimal design of hybrid fuel cell vehicles. *Journal of Fuel Cell Science and Technology*, to appear.

[58] W. E. Hart. The Coliny Project, Current as of 20 December, 2008. `http://software.sandia.gov/Acro/html/Projects/Coliny.html`.

[59] R. E. Hayes, F. H. Bertrand, C. Audet, and S. T. Kolaczkowski. Catalytic combustion kinetics: Using a direct search algorithm to evaluate kinetic parameters from light-off curves. *The Canadian Journal of Chemical Engineering*, 81:1192–1199, 2003.

[60] J. H. Holland. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, 1975.

[61] K. Holmström, A. O. Göran, and M. M. Edvall. *User's Guide for* TOMLAB. Tomlab Optimization, 2007. http://tomopt.com.

[62] K. Holmström, A. O. Göran, and M. M. Edvall. *User's Guide for* TOMLAB/CGO. Tomlab Optimization, 2007. http://tomopt.com.

[63] K. Holmström, N.-H. Quttineh, and M. M. Edvall. An adaptive radial basis algorithm (ARBF) for expensive black-box mixed-integer constrained global optimization. *Optimization and Engineering*, 9:311–339, 2008.

[64] R. Hooke and T. A. Jeeves. Direct search solution of numerical and statistical problems. *Journal of the Association for Computing Machinery*, 8:212–219, 1961.

[65] M. F. Hussain, R. R. Barton, and S. B. Joshi. Metamodeling: Radial basis functions, versus polynomials. *European Journal of Operational Research*, 138:142–154, 2002.

[66] W. Huyer and A. Neumaier. Global optimization by multilevel coordinate search. *Journal of Global Optimization*, 14:331–355, 1999.

[67] W. Huyer and A. Neumaier. SNOBFIT – Stable noisy optimization by branch and fit. *ACM Transactions on Mathematical Software*, 35:1–25, 2008.

[68] L. M. Hvattum and F. Glover. Finding local optima of high-dimensional functions using direct search methods. *European Journal of Operational Research*, 195:31–45, 2009.

[69] L. Ingber. *Adaptive Simulated Annealing (*ASA*)*, 2006. http://www.ingber.com/#ASA.

[70] R. Jin, W. Chen, and T. W. Simpson. Comparative studies of metamodelling techniques under multiple modelling criteria. *Structural and Multidisciplinary Optimization*, 23:1–13, 2001.

[71] D. R. Jones. A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization*, 21:345–383, 2001.

[72] D. R. Jones, C. D. Perttunen, and B. E. Stuckman. Lipschitzian optimization without the Lipschitz constant. *Journal of Optimization Theory and Application*, 79:157–181, 1993.

[73] D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13:455–492, 1998.

[74] G. Jones, P. Willett, R. C. Glen, A. R. Leach, and R. Taylor. Development and validation of a genetic algorithm for flexible docking. *Journal of Molecular Biology*, 267:727–748, 1997.

[75] C. T. Kelley. Detection and remediation of stagnation in the Nelder-Mead algorithm using a sufficient decrease condition. Technical report, North Carolina State University, 1997.

[76] C. T. Kelley. *Users Guide for* `IMFIL` *version 0.7*, 2008.

[77] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 1942–1948, Piscataway, NJ, USA, 1995.

[78] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[79] D. B. Kitchen, H. Decornez, J. R. Furr, and J. Bajorath. Docking and scoring in virtual screening for drug discovery: Methods and applications. *Nature Reviews: Drug Discovery*, 3:935–949, 2004.

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

[81] T. G. Kolda and V. J. Torczon. On the convergence of asynchronous parallel pattern search. *SIAM Journal on Optimization*, 14:939–964, 2004.

[82] I. D. Kuntz, J. M. Blaney, S. J. Oatley, R. Langridge, and T. E. Ferrin. A geometric approach to macromolecule-ligand interactions. *Journal of Molecular Biology*, 161:269–288, 1982.

[83] J. C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright. Convergence properties of the Nelder-Mead simplex method in low dimensions. *SIAM Journal on Optimization*, 9:112–147, 1998.

[84] G. M. Laslett. Kriging and splines: An empirical comparison of their predictive performance in some applications. *Journal of the American Statistical Association*, 89:391–400, 1994.

[85] A. R. Leach, B. K. Shoichet, and C. E. Peishoff. Prediction of protein-ligand interactions. Docking and scoring: Successes and gaps. *Journal of Medicinal Chemistry*, 49:5851–5855, 2006.

[86] R. M. Lewis and V. J. Torczon. Pattern search algorithms for bound constrained minimization. *SIAM Journal on Optimization*, 9:1082–1099, 1999.

[87] R. M. Lewis and V. J. Torczon. Pattern search algorithms for linearly constrained minimization. *SIAM Journal on Optimization*, 10:917–941, 2000.

[88] R. M. Lewis and V. J. Torczon. A globally convergent augmented lagrangian pattern search algorithm for optimization with general constraints and simple bounds. *SIAM Journal on Optimization*, 12:1075–1089, 2002.

[89] G. E. Liepins and M. R. Hilliard. Genetic algorithms: Foundations and applications. *Annals of Operations Research*, 21:31–58, 1989.

[90] S. Lucidi and M. Sciandrone. On the global convergence of derivative-free methods for unconstrained minimization. *SIAM Journal on Optimization*, 13:97–116, 2002.

[91] L. Lukšan and J. Vlček. Test problems for nonsmooth unconstrained and linearly constrained optimization. Technical report, Institue of Computer Science, Academy of Sciences of the Czech Republic, 2000.

[92] A. L. Marsden, J. A. Feinstein, and C. A. Taylor. A computational framework for derivative-free optimization of cardiovascular geometries. *Computer Methods in Applied Mechanics and Engineering*, 2006.

[93] A. L. Marsden, M. Wang, J. E. Dennis Jr., and P. Moin. Optimal aeroacustic shape design using the surrogate management framework. *Optimization and Engineering*, 5:235–262, 2004.

[94] A. L. Marsden, M. Wang, J. E. Dennis Jr., and P. Moin. Trailing-edge noise reduction using derivative-free optimization and large-eddy simulation. *Journal of Fluid Mechanics*, 5:235–262, 2007.

[95] M. D. McKay, R. J. Beckman, and W. J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21:239–245, 1979.

[96] K. I. M. McKinnon. Convergence of the Nelder-Mead simplex method to a nonstationary point. *SIAM Journal on Optimization*, 9:148–158, 1998.

[97] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21:1087–1092, 1953.

[98] M. Mongeau, H. Karsenty, V. Rouzé, and J. B. Hiriart-Urruty. Comparison of public-domain software for black box global optimization. *Optimization Methods & Software*, 13:203–226, 2000.

[99] J. Moré and S. Wild. Benchmarking derivative-free optimization algorithms. *Optimization Online Digest*, pages 1–20, 2008.

[100] G. M. Morris, D. S. Goodsell, R. S. Halliday, R. Huey, W. E. Hart, R. K. Belew, and A. J. Olson. Automated docking using a lamarckian genetic algorithm and an empirical binding free energy function. *Journal of Computational Chemistry*, 19:1639–1662, 1998.

[101] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[102] A. Neumaier. Library 1 Sources: GLOBALLib and Floudas' Collection, Current as of 20 December, 2008. `http://www.mat.univie.ac.at/~neum/glopt/coconut/Benchmark/Library1_new_v1.html`.

[103] A. Neumaier. Library 2 Source: Vanderbei's CUTE Test Collection, Current as of 20 December, 2008. `http://www.mat.univie.ac.at/~neum/glopt/coconut/Benchmark/Library2_new_v1.html`.

[104] A. Neumaier. `MCS`: Global Optimization by Multilevel Coordinate Search, Current as of 20 December, 2008. `http://www.mat.univie.ac.at/~neum/software/mcs/`.

[105] A. Neumaier, O. Shcherbina, W. Huyer, and T. Vinkó. A comparison of complete global optimization solvers. *Mathematical Programming*, 103:335–356, 2005.

[106] R. Oeuvray. *Trust-region methods based on radial basis functions with application to biomedical imaging.* PhD thesis, Institute of Mathematics, Swiss Federal Institute of Technology, Lausanne, Switzerland, March 2005.

[107] Oracle. `Oracle Berkeley DB` Product Family, Current as of 20 December, 2008. `http://www.oracle.com/database/berkeley-db/`.

[108] J. E. Orosz and S. H. Jacobson. Finite-time performance analysis of static simulated annealing algorithms. *Computational Optimization and Applications*, 21:21–53, 2002.

[109] J. Pintér. Homepage of Pintér Consulting Services, Current as of 20 December, 2008. `http://www.pinterconsulting.com/`.

[110] J. D. Pintér. *Global Optimization in Action: Continuous and Lipschitz Optimization. Algorithms, Implementations and Applications.* Nonconvex Optimization and Its Applications. Kluwer Academic Publishers, 1995.

[111] J. D. Pintér, K. Holmström, A. O. Göran, and M. M. Edvall. *User's Guide for* `TOMLAB/LGO`. Tomlab Optimization, 2006. `http://tomopt.com`.

[112] M. J. D. Powell. A direct search optimization method that models the objective by quadratic interpolation. In *Presentation at the 5th Stockholm Optimization Days*, 1994.

[113] M. J. D. Powell. Recent research at Cambridge on radial basis functions. Technical report, Department of Applied Mathematics and Theoretical Physics, University of Cambridge, 1998.

[114] M. J. D. Powell. The NEWUOA software for unconstrained optimization without derivatives. In G. Di Pillo and M. Roma (eds.), *Large-Scale Nonlinear Optimization,* Springer, New York, NY, pages 255–297, 2006.

[115] M. J. D. Powell. Developments of NEWUOA for minimization without derivatives. *IMA Journal of Numerical Analysis*, 28:649–664, 2008.

[116] Princeton Library of Nonlinear Programming Models, Current as of 20 December, 2008. `http://www.gamsworld.org/performance/princetonlib/princetonlib.htm`.

[117] M. Rarey, B. Kramer, T. Lengauer, and G. Klebe. A fast flexible docking method using an incremental construction algorithm. *Journal of Molecular Biology*, 261:470–489, 1996.

[118] R. G. Regis and C. A. Shoemaker. Constrained global optimization of expensive black box functions using radial basis functions. *Journal of Global Optimization*, 31:153–171, 2005.

[119] R. G. Regis and C. A. Shoemaker. Improved strategies for radial basis function methods for global optimization. *Journal of Global Optimization*, 37:113–135, 2007.

[120] L. M. Rios and N. V. Sahinidis. Derivative-free optimization: A review of algorithms and comparison of software implementations. *INFORMS Journal on Computing*, to be submitted.

[121] F. Romeo and A. Sangiovanni-Vincentelli. A theoretical framework for simulated annealing. *Algorithmica*, 6:302–345, 1991.

[122] J. Sacks, W. J. Welch, T. J. Mitchell, and H. P. Wynn. Design and analysis of computer experiments. *Statistical Science*, 4:409–423, 1989.

[123] N. V. Sahinidis. BARON: A general purpose global optimization software package. *Journal of Global Optimization*, 8:201–205, 1996.

[124] N. V. Sahinidis and M. Tawarmalani. *BARON 7.5: Global Optimization of Mixed-Integer Nonlinear Programs,* User's Manual, 2005.

[125] K. Scheinberg. *Manual for Fortran Software Package* `DFO` *v2.0*, 2003.

[126] M. Schonlau. *Computer Experiments and Global Optimization.* PhD thesis, Department of Statistics, University of Waterloo, Waterloo, Ontario, Canada, 1997.

[127] B. O. Shubert. A sequential method seeking the global maximum of a function. *SIAM Journal on Numerical Analysis*, 9:379–388, 1972.

[128] R. L. Smith. Efficient Monte Carlo procedures for generating points uniformly distributed over bounded regions. *Operations Research*, 32:1296–1308, 1984.

[129] A. Sóbester, S. J. Leary, and A. J. Keane. On the design of optimization strategies based on global response surface approximation models. *Journal of Global Optimization*, 33:31–59, 2005.

[130] W. Spendley, G. R. Hext, and F. R. Himsworth. Sequential application for simplex designs in optimisation and evolutionary operation. *Technometrics*, 4:441–461, 1962.

[131] V. J. Torczon. On the convergence of multidirectional search algorithms. *SIAM Journal on Optimization*, 1:123–145, 1991.

[132] V. J. Torczon. On the convergence of pattern search algorithms. *SIAM Journal on Optimization*, 7:1–25, 1997.

[133] A. C.M. van Beers and J. P. C. Kleijnen. Kriging interpolation in simulation: A survey. *Proceedings of the 2004 Winter Simulation Conference*, 1:121–129, 2004.

[134] A. I. F. Vaz. `PSwarm` Home Page, Current as of 20 December, 2008. `http://www.norg.uminho.pt/aivaz/pswarm/`.

[135] A. I. F. Vaz and L. N. Vicente. A particle swarm pattern search method for bound constrained global optimization. *Journal of Global Optimization*, 39:197–219, 2007.

[136] G. L. Warren, C. W. Andrews, A. Capelli, B. Clarke, J. LaLonde, M. H. Lambert, M. Lindvall, N. Nevins, S. F. Semus, S. Senger, G. Tedesco, I. D. Wall, J. M. Woolven, C. E. Peishoff, and M. S. Head. A critical assessment of docking programs and scoring functions. *Journal of Medicinal Chemistry*, 49:5912–5931, 2006.

[137] S. M. Wild, R. G. Regis, and C. A. Shoemaker. ORBIT: Optimization by radial basis function interpolation in trust-regions. *Optimization Online Digest*, pages 1–23, 2008.

[138] T. A. Winslow, R. J. Trew, P. Gilmore, and C. T. Kelley. Simulated performance optimization of gaas mesfet amplifiers. In *IEEE/Cornell Conference on Advanced Concepts in High Speed Semiconductor Devices and Circuits*, pages 393–402, Piscataway, NJ, 1991.

[139] Z. Zhao, J. C. Meza, and M. Van Hove. Using pattern search methods for surface structure determination of nanomaterials. *Journal of Physics: Condensed Matter*, 18:8693–8706, 2006.

Author's Biography

Luis Miguel Rios was born in Lima, Peru. He received his bachelor degree in Industrial Engineering from Pontificia Universidad Católica del Perú in 1999. After working three years in an insurance company, he moved to Champaign to start a PhD Program in Industrial Engineering at UIUC under Professor Nick Sahinidis. During an internship at the Research Company at Exxon-Mobil, he developed interest in derivative-free optimization, which later become the main topic of his research. This interest was reassured a year later while returning for another project at Exxon-Mobil. Luis Miguel became happily married in the summer of 2006. Luis Miguel will be joining Carnegie Mellon University during the summer to work at a post-doctoral position.