Direct Search Methods for Nonsmooth Problems using Global Optimization Techniques

A THESIS
SUBMITTED IN PARTIAL FULFILMENT
OF THE REQUIREMENTS FOR THE DEGREE
OF
DOCTOR OF PHILOSOPHY IN MATHEMATICS
by

Blair Lennon Robertson

University of Canterbury
2010

To Peter

Abstract

This thesis considers the practical problem of constrained and unconstrained local optimization. This subject has been well studied when the objective function f is assumed to smooth. However, nonsmooth problems occur naturally and frequently in practice. Here f is assumed to be nonsmooth or discontinuous without forcing smoothness assumptions near, or at, a potential solution. Various methods have been presented by others to solve nonsmooth optimization problems, however only partial convergence results are possible for these methods.

In this thesis, an optimization method which use a series of local and localized global optimization phases is proposed. The local phase searches for a local minimum and gives the methods numerical performance on parts of f which are smooth. The localized global phase exhaustively searches for points of descent in a neighborhood of cluster points. It is the localized global phase which provides strong theoretical convergence results on nonsmooth problems.

Algorithms are presented for solving bound constrained, unconstrained and constrained nonlinear nonsmooth optimization problems. These algorithms use direct search methods in the local phase as they can be applied directly to nonsmooth problems because gradients are not explicitly required. The localized global optimization phase uses a new partitioning random search algorithm to direct random sampling into promising subsets of $\mathbb{R}^n$. The partition is formed using classification and regression trees (CART) from statistical pattern recognition. The CART partition defines desirable subsets where f is relatively low, based on previous sampling, from which further samples are drawn directly. For each algorithm, convergence to an essential local minimizer of f is demonstrated under mild conditions. That is, a point x_* for which the set of all feasible points with lower f values has Lebesgue measure zero for all sufficiently small neighborhoods of x_*. Stopping rules are derived for each algorithm giving practical convergence to estimates of essential local minimizers. Numerical results are presented on a range of nonsmooth test problems for 2 to 10 dimensions showing the methods are effective in practice.

Acknowledgements

It has been a pleasure working with Dr Chris Price over the years, learning from his extensive knowledge of mathematics. On many occasions Chris stepped above and beyond the role of primary supervisor and offered advice to me as a friend which I am very grateful for. I also thank my secondary supervisor Dr Marco Reale for his encouragement, advice and enthusiasm in my research. I thoroughly enjoyed our supervisory team discussions, both mathematical and non-mathematical.

There are too many friends to thank for whom I bounced ideas off, or bored to tears as the case may be, but I will mention Jason Bentley with whom I shared an office and many interesting discussions.

I would also like to thank my parents for their support and financial understanding over the years. I dedicate this thesis to my father Peter who is sadly no longer with us.

Finally, I would like to thank the Department for all the financial support including a Departmental PhD Scholarship and conference funds which allowed me to present some of my research at the 17^{th} International Conference on Computing in Economics and Finance in Cyprus and present a seminar at the Third University of Rome.

Contents

Notation

$\mathbf{0}$	The null vector (zero vector) in $\mathbb{R}^n$
a_k	Lower bound on a bracket
A	Low sub-region of an approximate level set
A_i	i^{th} low sub-region unless otherwise stated
$\|A\|$	Number of low sub-regions
$\mathcal{A}$	A singleton low sub-region
b_j	Lower bound on the j^{th} coordinate of a sub-region
b_{j+n}	Upper bound on the j^{th} coordinate of a sub-region
b_k	Upper bound on a bracket
B	Matrix containing sub-region bounds
B_i	Bounds for sub-region A_i
$\mathcal{B}(x, \epsilon)$	The open ball centered on x with radius ϵ
$\mathfrak{B}(x)$	Barrier function which maps $\mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$
c_i	i^{th} constraint function
$C(x)$	The set of constraint functions
$\mathcal{C}$	Finite set of category lables
d	Direction vector in $\mathbb{R}^n$
D	Node number
$\mathfrak{D}$	Training data transformation function
e_i	The i^{th} column of the identity matrix
E_k	Hooke and Jeeves exploratory phase vector
$\mathfrak{E}(z, \epsilon)$	The level set $L(z)$ in an ϵ-neighborhood of z
f	The objective function f that maps $\mathbb{R}^n \to \mathbb{R}$ (unless otherwise stated)
$f(x)$	Value of the objective at a point x
f_{inf}	See (4.6) and Definition 42 in Chapter 4

$f^o(x)$	The Clarke derivative of f at x
f_γ	Largest function value in the set Y
f_*	Abbreviation for $f(x_*)$
$\hat{f}_*$	Approximation to f_*
$\hat{f}_*^*$	The best approximation to f_*
F	The cumulative distribution function
$\hat{F}$	The empirical distribution function
$\mathcal{F}$	Sloping filter
$g(x, \sigma)$	Penalty function evaluated at x with penalty parameter σ
g_k	Penalty function with penalty parameter σ_k
g	Abbreviation for ∇f (unless otherwise stated)
G	The Hessian Matrix
GP	Global optimization procedure
$\mathcal{G}$	A grid in R^n
h	Mesh size parameter
$h_{\min}$	Terminating mesh size
h_Ω	Lower bound on localized global optimization region radius
$\mathfrak{h}$	Constraint violation function that maps $\mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$
$\mathfrak{h}_{\min}$	Lower bound on constraint violation
$\mathfrak{h}_{\max}$	Upper bound on constraint violation
H	The Householder matrix
H_Ω	Optimization region transformation Householder matrix
i, j, k	Iteration counters or indices
$i(D)$	Impurity measure at node(D)
I	The identity matrix
$\mathcal{I}_E$	Index vector for promising directions in the Hooke and Jeeves exploratory phase
$k_{\max}$	Maximum number of iterations
K_E	Kinetic energy of a particle of unit mass
ℓ_i	Lower bound on the i^{th} coordinate
$L(.)$	Level set: $L(z) = \{x : f(x) \leq f(z)\}$
$\mathcal{L}$	Approximate level set
$\mathcal{L}_k$	Approximate level set at iteration k

m	Iteration counter or index
$m(.)$	Lebesgue Measure
$M(x)$	A positive, bounded scaling function
$\mathcal{M}$	Scatter Matrix
n	Number of independent variables
N	Batch size
$\mathcal{N}$	Feasible Region
$\mathcal{O}$	Grid center
p	Number system base
P_E	Potential energy of a particle of unit mass
$\mathcal{P}_1$	One norm power fit
$\mathcal{P}_2$	Euclidean norm power fit
$\mathcal{P}_\infty$	Infinity norm power fit
$\mathcal{P}_\infty^*$	Optimal infinity norm power fit
$\mathfrak{P}_i$	Unique path vector to terminal node(i)
$Pr(.)$	Probability
r	Singleton sub-region radius
$\mathcal{R}$	Objective function value range in the set Y
$\mathbb{R}$	The real numbers
$\mathbb{R}^n$	Euclidean n-space
s	Scalar splitting value
$\mathcal{S}$	Optimization region
$\mathfrak{S}$	Simplex
t	Time parameter, usually
T	The training data set
$T(D)$	Training data at node(D)
T_0	Initial training data set
$\hat{T}$	The transformed training data set
T_γ	Subset of T containing the γ samples with the least function values
T^*	Training data set after a successful localized global optimization phase
$\mathcal{T}$	Classifier function which maps $\Omega \rightarrow \mathcal{C}$
$\mathfrak{T}_R$	Resent sample points

$\mathfrak{T}_\gamma$	Subset of T containing the sample points with the γ least $\mathfrak{h}$ and γ least g_k values
u	A unit vector in $\mathbb{R}^n$
u_i	Upper bound on the i^{th} coordinate
U_i	Uniform random variable of dimension i
$\mathcal{U}$	Upper bound on allowable function values
v	Velocity vector
v_0	Initial velocity vector
$\mathcal{V}$	Set of vectors which form a basis for $\mathbb{R}^n$
$\mathcal{V}^+$	Set of vectors which form a positive basis for $\mathbb{R}^n$
w	A unit vector in $\mathbb{R}^n$
x	A vector in $\mathbb{R}^n$
x_0	Initial point
$\{x_k\}$	Sequence of iterates
x_j^-	Minimum value of the j^{th} coordinate for the set $\{\omega_L \cap A_i\}$
x_j^+	Maximum value of the j^{th} coordinate for the set $\{\omega_L \cap A_i\}$
x_*	An essential local minimizer of f
$\bar{x}$	Geometric mean of a set of points
$\dot{x}$	First derivative of x with respect to time
$\ddot{x}$	Second derivative of x with respect to time
X	Batch of sample points
$X_\mathcal{D}$	Set of undominated points in $\mathbb{R}^n$
X_E	Set of points generated from a Hooke and Jeeves exploratory phase
X_R	Recent sample points
X_γ	γ sample points with least function values
$y(x)$	A real valued function the maps $\mathbb{R}^n \to \mathbb{R}$
Y	Set of γ lowest objective function values
z	A vector in $\mathbb{R}^n$
z_i	i^{th} coordinate of z
$\{z_k\}$	Sequence of iterates
z_*	An essential local minimizer of f
$\mathbb{Z}$	The set of integers
$\mathbb{Z}_+$	The set of positive integers

α	Forward-tracking face search step size parameter
β	Real number contained in the interval (0,1)
γ	Number of points with the least function values retained in T
δ	Minimum low sub-region radius
$\Delta i(D)$	Change in impurity measure at node(D)
ϵ	Small positive quantity
ζ	Small positive quantity
η	Real number contained in the interval [0,1]
θ	Various scalars
κ	Exponent of expected cumulative distribution function
κ^*	Optimal exponent of expected cumulative distribution function
λ	Positive scalar
λ_i	i^{th} positive scalar
μ	Sloping filter parameter
ν	Vector in $\mathbb{R}^n$, usually an element of a positive basis
ξ	KS test statistic
$\xi_{\eta,\gamma}$	Critical value for the KS test
ρ_k	Fraction of samples within ϵ of f_1 at iteration k
σ	Penalty Parameter
$\sigma_{\max}$	Maximum penalty parameter (finite)
σ_*	Terminating penalty parameter
τ_b	Terminating bracket size in GSS algorithm
τ_h	Maximum mesh reduction at each iteration parameter
τ_i	Misclassification parameter
ϕ_p	Radical inverse function of base p
Φ	Distribution function
φ	Householder transformation scalar
χ	Fraction of samples distributed into high sub-regions
$\{\omega_L\}$	Set of points with relatively low function values
$\{\omega_H\}$	Set of points with relatively high function values
Ω	Global optimization region
Ω_{f_ϵ}	The level set $L(f(z_*)+\epsilon)\cap\Omega$

Ω_m	Localized global optimization region at iteration m
Ω_ϵ	The ϵ-neighborhood of $x_* \cap \Omega$
$\nabla f(x)$	The gradient of f at x
$\emptyset$	The empty set
$\setminus$	Set difference operator: $x \in X \setminus Y \Rightarrow x \in X$ and $x \notin Y$
$\Rightarrow$	Implication symbol
$\overline{\Omega}$	The closure of Ω
Ω^{O}	The interior of Ω
$\lfloor . \rfloor$	Floor function: $\lfloor x \rfloor$ is the greatest integer less than or equal to x
$\lceil . \rceil$	Ceiling function: $\lceil x \rceil$ is the least integer greater than or equal to x
$[x]_+$	The maximum of zero and x
$\|.\|$	Euclidean norm
$\|.\|_\infty$	Infinity norm
$\square$	End of proof marker

Chapter 1

Introduction

So what exactly is optimization? The online dictionary Wiktionary defines the word optimization as:

> *the design and operation of a system or process to make it as good as possible in some defined sense.*

Thus, one may think of optimization as the art or science of determining the *best* solution to certain mathematically defined problems. An objective function defined by a set of independent decision variables is used to determine the *goodness* of a solution. The optimal value of the objective function represents the best solution to the problem. Many practical problems can be modeled by an objective function and hence, optimization problems can occur in many areas of research [76]. An economist, for example, may wish to maximize profits whereas an engineer may be interested in minimizing drag on a performance car.

The field of optimization is an interesting mix of mathematical theory and experimental simulations. One can study the field from a purely mathematical point of view, or use a blend of heuristic and theoretical ideas to construct and test practically implementable algorithms. In this thesis, the field is studied from the latter perspective.

Nonsmooth phenomena in mathematics and optimization occur naturally and frequently [13]. The interested reader is referred to [13] for examples of nonsmooth optimization problems. These problems are pulled from various fields including chemistry, physics, economics, engineering and of course mathematics, showing an extremely diverse range of practical situations. Thus, there is a need for provably convergent, implementable algorithms to solve such problems.

The main focus of this thesis is to develop practical algorithms for solving nonsmooth optimization problems. Currently, there exist algorithms which can be applied

to nonsmooth problems, however, only partial convergence results on nonsmooth problems are possible for these methods. Here practical algorithms with strong theoretical convergence to optimal points of nonsmooth problems are presented.

The remainder of this chapter is organized as follows. Firstly, the local optimization problem is introduced along with definitions of what points are considered optimal in this thesis. Sections 1.2 and 1.3 review direct search methods for local optimization. These methods are of interest here because no gradient information is required and hence, they can be applied to nonsmooth problems and partial nonsmooth convergence results exist for some methods. A brief introduction to trajectory following optimization is given in Section 1.4 as these methods have advantageous properties for nonsmooth problems. Section 1.5 introduces the global optimization problem and Section 1.6 shows a connection between nonsmooth local optimization and global optimization. This connection is exploited in Section 1.7 to provide an algorithmic framework for nonsmooth problems. The chapter concludes with a thesis overview.

1.1 Local Optimization

This thesis considers the local minimization problem. Given an objective function f that maps $\mathbb{R}^n \to \mathbb{R}$, the general unconstrained minimization problem is written as

$$\min_x f(x) \text{ subject to } x \in \mathbb{R}^n. \tag{1.1}$$

To find a local solution to (1.1) it is sufficient to find a point $x_* \in \mathbb{R}^n$ such that

$$f(x_*) \leq f(x) \text{ for all } x \in \mathcal{B}(x_*, \epsilon), \tag{1.2}$$

where $\mathcal{B}(x_*, \epsilon) = \{x \in \mathbb{R}^n : \|x - x_*\| < \epsilon\}$ is an open ball with center x_* and radius $\epsilon > 0$. A point x_* satisfying (1.2) is called a local minimizer of f and $f(x_*)$ is called a local minimum.

The problem of finding a local maximum is an equivalent problem since

$$\min\{f(x) : x \in \mathbb{R}^n\} = -\max\{-f(x) : x \in \mathbb{R}^n\}.$$

Both problems are referred to as optimization and throughout this thesis, unless otherwise stated, optimization implies minimization.

For the case where derivative information is available (even if it comes at large

computational expense) and f is smooth and free of noise, the local optimization problem has been well studied. Many efficient and provably convergent algorithms have been presented to solve such problems. Examples include Newton's Method, Quasi-Newton methods and Conjugate Gradient methods [55]. However, for the case when f is assumed to be nonsmooth or discontinuous, these existing methods are no longer provably convergent. In this thesis these problems are of primary interest. In particular, objective functions f that map $\mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$, where f is assumed to be nonsmooth or discontinuous are considered. The inclusion of $\{+\infty\}$ means the methods proposed here can be applied to certain constrained optimization problems directly using an extreme barrier function, of the form,

$$\mathfrak{B}(x) = \begin{cases} f(x) & \text{if } x \in \mathcal{N} \\ +\infty & \text{otherwise,} \end{cases}$$

where $\mathcal{N} \subset \mathbb{R}^n$ is a feasible region with positive Lebesgue measure and a minimizer over $\mathcal{N}$ is sought. It is not necessary to evaluate f at points outside $\mathcal{N}$, rather the barrier function assigns the value $+\infty$. In addition, functions which are not defined everywhere can be considered by assigning the value $+\infty$ where f is undefined.

Since f is assumed to be nonsmooth or discontinuous the standard definition of a local minimizer is modified.

Definition 1. (Essential local minimizer). *A point $x_* \in \mathbb{R}^n$ for which the set*

$$\mathfrak{E}(x_*, \epsilon) = \{x \in \mathbb{R}^n : f(x) < f(x_*) \text{ and } \|x - x_*\| < \epsilon\} \tag{1.3}$$

has Lebesgue measure zero for all sufficiently small positive ϵ is called an essential local minimizer of f.

Definition 2. (Essential local minimum). *If a point $x_* \in \mathbb{R}^n$ is an essential local minimizer of f, then $f(x_*)$ is called an essential local minimum of f.*

If the objective function is continuous at x_*, then x_* is also a local minimizer in the classical sense. If $x \in \mathbb{R}^n$ is not an essential local minimizer of f, then there exists a lower region with positive measure arbitrarily close to x. For completeness, a descent direction is defined.

Definition 3. (Descent direction). *The direction vector $d \in \mathbb{R}^n$ is a descent direction for f at x if there exists a $\hat{\lambda} > 0$ such that*

$$f(x + \lambda d) < f(x) \text{ for all } \lambda \in (0, \hat{\lambda}].$$

1.2 Direct Search Local Optimization

Direct search methods are of interest here because they do not require the explicit calculation of derivatives and thus can be applied to nonsmooth optimization problems. Although there is some ambiguity regarding which methods can be classified as direct search (see [41] for an interesting discussion), any method that does not calculate the gradient of f (∇f) directly is considered direct search. Therefore, a method which replaces derivative information with finite difference approximations to ∇f is considered direct search here, to the dismay of some authors.

Direct search methods date back to the 1950s but the field blossomed during the 1960s with the development of various algorithms, including the Hooke and Jeeves algorithm [35] for which the phrase *direct search* was first used. These early algorithms became popular amongst practitioners because their heuristic form was easy to understand and they often performed well in practice. Despite their popularity, by the early 1970s these methods were largely dismissed by the optimization community. Numerical optimizers became less interested in heuristics and more interested in formal theories of convergence [44]. Derivative based methods with convergence results then took center stage within the optimization community. These methods also offered superior performance on smooth problems, for example the quasi-Newton methods, whose success is now undisputed [44].

In 1991, direct search methods become popular once again when Torczon [79] developed a general convergence theory for a class of direct search methods. This lead to the development of a variety of new, provably convergent direct search algorithms. The interested reader is referred to [41] for a comprehensive review of the state-of-the-art in direct search until circa 2003. Further advances in the field until just recently can be found in [14].

This section concludes by briefly introducing simplex based methods, due to their undeniable popularity among practitioners when gradient information is not available or unreliable. Interestingly, the original Nelder-Mead simplex method [52] has become an official Science Citation Classic. Directional direct search methods are also considered because partial convergence results have been developed by others on nonsmooth problems. Furthermore, a method proposed in Chapter 5 for nonsmooth minimization uses a directional direct search method. These methods are considered in the next section.

1.2.1 Simplex Based Methods

A simplex is an n dimensional analog of a triangle, defined as the convex hull of $n+1$ independent points in $\mathbb{R}^n$. Unless otherwise stated, each simplex is assumed to be non-degenerate.

Definition 4. (Non-Degenerate Simplex). *A non-degenerate simplex in $\mathbb{R}^n$ is the convex hull of $n+1$ linearly independent points, where the set of edges from any vertex forms a basis for $\mathbb{R}^n$.*

If all points are mutually equidistant from each other, then the simplex is said to be regular. Therefore, an equilateral triangle and a regular tetrahedron define a regular simplex in $\mathbb{R}^2$ and $\mathbb{R}^3$ respectively.

Simplicial direct search methods are motivated by the fact that only $n+1$ function evaluations are required to approximate the gradient (for example via finite differences) and hence, can be used to approximate a direction of descent if f is continuously differentiable. Given an initial simplex, these methods transform and scale the simplex by replacing a vertex at each iteration. Rather than trying to reduce f at each iteration, these methods aim to replace the vertex with the largest function value.

The first simplex based method is due to Spendley, Hext and Himsworth [75] in 1962. Their method has two modes of operation: reflection and scaling. Given a regular simplex at iteration k, $\mathfrak{S}_k$, a new (regular) simplex is generated by reflecting the vertex with the largest function value through the centroid of the remaining n points. If the new vertex no longer has the largest function value, the simplex is accepted giving $\mathfrak{S}_{k+1}$ and the method repeats. Otherwise, the vertex in $\mathfrak{S}_k$ with second largest vertex is reflected and so on, until no descent is forthcoming or if cycling occurs. Cycling is evident when the vertex with the lowest function value remains constant for many iterations, see [44] for details. The authors suggest two alternatives at this stage, either reduce the side lengths of the simplex toward the vertex with the lowest function value (see Figure 1.2) and repeat, or employ a convergent local search method in the neighborhood identified by the simplex.

1.2.1.1 Nelder-Mead Method

The Nelder-Mead simplex method [52] is a generalization of the Spendley, Hext and Himsworth Algorithm. The interested reader is referred to [14] for a detailed description of this method and a clear algorithmic diagram. This section illustrates the basic

methodology of the Nelder-Mead method and explains how it can fail to converge to a local minimizer of f.

Like the Spendley, Hext and Himsworth Algorithm, the Nelder-Mead method proceeds by generating a new simplex from the previous. Let the simplex be defined by the sorted set of $n+1$ independent points

$$\mathfrak{S} = \{x_1, x_2, \ldots, x_{n+1}\}, \tag{1.4}$$

such that $f(x_i) \leq f(x_{i+1})$ for $1 \leq i \leq n$. Rather than simply reflecting x_{n+1} to generate the next simplex, the Nelder-Mead method considers a set of points on the line

$$l = \bar{x} + \theta(\bar{x} - x_{n+1}) \text{ such that } \theta \in \mathbb{R}, \tag{1.5}$$

where $\bar{x}$ denotes the geometric centroid of the remaining points, given by $1/n \sum_{i=1}^{n} x_i$. The typical points to consider are given by $\theta \in \{-1/2, 1/2, 1, 2\}$ [14]. With $\theta = 1$ the reflection move of Spendley, Hext and Himsworth is conducted leaving the simplex shape unchanged. An expansion move is conducted with $\theta = 2$, elongating the simplex in the direction $\bar{x} - x_{n+1}$. For both cases $\theta = \pm 1/2$ a contraction move is conducted, reducing the measure of the simplex. These moves are illustrated in Figure 1.1. The details and logic of which simplex is chosen at each iteration is left to others (see for example [14]) and note that a variety of simplicial shapes can be generated over successive iterations. If these moves fail to reduce $f(x_{n+1})$ below $f(x_n)$, then the simplex is *shrunk* towards the vertex with the lowest function value, usually by a factor of a $1/2$ as shown in Figure 1.2. The method terminates when the simplex diameter falls below a tolerance specified by the user.

Definition 5. (Simplex diameter). *Let $\mathfrak{S}$ be a simplex in $\mathbb{R}^n$ and $i, j \in \{1, 2, \ldots, n+1\}$, then the simplex diameter is defined as*

$$\operatorname{diam}(\mathfrak{S}) = \max_{i,j} \|x_i - x_j\|. \tag{1.6}$$

Other termination rules for the method can be found in [10, 14].

Allowing the simplex to deform means the simplex can adapt to the local landscape of the objective function taking a variety of simplicial shapes. For example, elongating down valleys and contracting around minimizers. Although this can lead to methods performing rather well in practice, it can have disastrous consequences. In particular, the simplices can become numerically degenerate or arbitrarily flat. McKinnon [48]

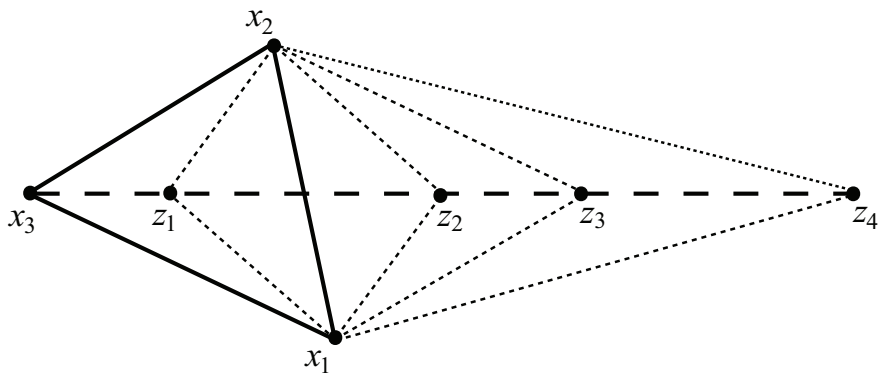


Figure 1.1: Possible simplices obtained for the Nelder-Mead method. The initial simplex is shown in bold and the reflection line is bold dashed. Points z_1 and z_2 define simplices from contraction moves. Point z_3 defines the simplex from the reflection move and z_4 is the simplex produced from the expansion move.

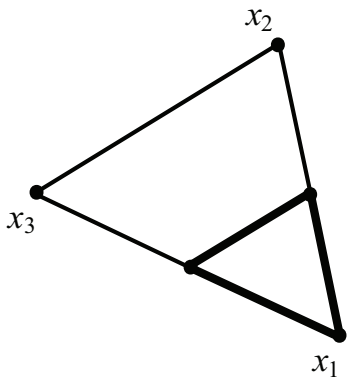


Figure 1.2: Simplex shrinking move of the Nelder-Mead method. The new simplex is shown in bold.

derived a family of functions for which this degeneracy is observed, causing the method to fail. Surprisingly this family includes a twice continuously differentiable convex function in $\mathbb{R}^2$. For this reason alone it is not possible to show that Nelder-Mead method converges to a minimizer of f and therefore, can only be considered a heuristic method.

Various authors have provided modified versions of the classical method for which convergence can be demonstrated, see for example [14, 59]. These results rely on f being continuously differentiable (and others) and do not extend to nonsmooth problems in an obvious way. Although any direct search method can be applied to nonsmooth problems, the author sees no advantageous reason in choosing a simplex method and they are considered no further in this thesis.

1.3 Directional Direct Search Methods

Directional direct search methods are methods where sampling is guided by sets of directions with appropriate features. The name is borrowed from [14] where a general algorithmic framework for this class of direct search methods is provided. Similar frameworks have been provided by others. Kolda, Lewis, and Torczon [41] provide a convergent framework on twice continuously differentiable functions called Generating Set Search. A popular algorithm conforming to both frameworks is called Mesh Adaptive Direct Search (MADS) [5]. The interested reader is referred to [5, 14] for further details. The MADS algorithm is of interest because it has partial convergence results on nonsmooth problems (see Section 1.3.3).

Of particular interest in this thesis are grid based optimization methods, where sampling is guided by points on a lattice. Before discussing these methods, Positive Bases are introduced, an underlying structure used in this approach.

1.3.1 Positive Bases

In 1954 Davis [19] introduced the theory of positive bases. Positive bases and positive spanning sets are a key feature in grid and frame based optimization methods [10]. The interested reader is referred to [14, 19] for a full treatment on the subject. Here a brief introduction is presented and a result which is used in the convergence analysis in Chapter 5 is given.

Definition 6. (Positive basis). *A positive basis $\mathcal{V}^+$ for $\mathbb{R}^n$ is a set of vectors for which the following conditions hold:*

(a) *Every vector in $\mathbb{R}^n$ can be written as a non-negative linear combination of vectors in $\mathcal{V}^+$;*

(b) *No member of $\mathcal{V}^+$ is expressible as a non-negative linear combination of the remaining vectors in $\mathcal{V}^+$.*

A finite set of vectors which satisfy condition (a), but not necessarily (b), is called a positive spanning set for $\mathbb{R}^n$. Interesting, the cardinality of a positive basis is $n+1 \leq |\mathcal{V}^+| \leq 2n$, unlike a basis for $\mathbb{R}^n$ which has exactly n vectors [85]. Positive bases with $n+1$ and $2n$ vectors are called minimal and maximal positive bases respectively. For example, if $\{\nu_1, \nu_2, \ldots, \nu_n\}$ is a basis for $\mathbb{R}^n$, then

$$\left\{\nu_1, \nu_2, \ldots, \nu_n, -\sum_{i=1}^{n} \nu_i\right\} \quad \text{and} \quad \{\nu_1, \nu_2, \ldots, \nu_n, -\nu_1, -\nu_2, \ldots, -\nu_n\}, \tag{1.7}$$

are minimal and maximal positive bases for $\mathbb{R}^n$.

Two attractive properties of positive bases for smooth optimization are made clear from the following theorem and corollary. For clarity the notation $\nabla f(x) = g$ is used and all vectors are column vectors.

Theorem 7. *If the set of vectors $\mathcal{V}^+$ forms a positive basis for $\mathbb{R}^n$, then*

$$\nu^\mathsf{T} g \geq 0 \text{ for all } \nu \in \mathcal{V}^+ \Rightarrow g = \mathbf{0}. \tag{1.8}$$

Proof. Let $\mathcal{V}^+ = \{\nu_i\}$, where $i = 1, 2, \ldots, |\mathcal{V}^+|$. Since any vector in $\mathbb{R}^n$ can be written as a non-negative linear combination of vectors in $\mathcal{V}^+$ let

$$-g = \sum_{i=1}^{|\mathcal{V}^+|} \lambda_i \nu_i,$$

where each $\lambda_i \geq 0$ and $i = 1, 2, \ldots, |\mathcal{V}^+|$. From (1.8) we have $\nu_i^\mathsf{T} g \geq 0$ for each $\nu_i \in \mathcal{V}^+$ and so

$$0 \leq \sum_{i=1}^{|\mathcal{V}^+|} \lambda_i \nu_i^\mathsf{T} g = -g^\mathsf{T} g \leq 0.$$

The only possibility is $g = \mathbf{0}$. □

Thus, positive bases can be used to confirm whether a point $x \in \mathbb{R}^n$ is a local minimizer of a smooth objective function or not. In addition, positive bases can be used to define a descent direction of a continuously differentiable function, if one exists.

Corollary 8. *Let f be a continuously differentiable function with $\nabla f(x) \neq \mathbf{0}$ for some $x \in \mathbb{R}^n$. Then given a positive basis $\mathcal{V}^+$, there exists a $\nu \in \mathcal{V}^+$ such that*

$$-\nabla f(x)^\mathsf{T} \nu > 0.$$

Proof. Let $g = -\nabla f(x)$ where $x \in \mathbb{R}^n$. Noting that $g^\mathsf{T} g > 0$ for all non-zero g and

$$g = \lambda_1 \nu_1 + \lambda_2 \nu_2 + \ldots + \lambda_i \nu_i,$$

where $\nu_i \in \mathcal{V}^+$, and λ_i is a non-negative scalar for $i = 1, 2, \ldots, |\mathcal{V}^+|$. Hence,

$$g^\mathsf{T}(\lambda_1 \nu_1 + \lambda_2 \nu_2 + \ldots + \lambda_i \nu_i) > 0 \tag{1.9}$$

and so there is at least one element of (1.9) for which $g^\mathsf{T} \nu_i > 0$, as required. □

Another way of thinking about Corollary 8 is that there exists at least one vector from a positive basis probing any open half space.

1.3.2 Grid Based Methods

Grid based direct search methods are a generalization of pattern search methods [16]. Grids can be generalized further into frame based methods. The interested reader is referred to [10, 15, 18, 60] for details on frame based methods and we consider them no further.

At each iteration, grid based methods evaluate f at a finite set of points on a rational lattice or grid. Each grid $\mathcal{G}(x_0, h, \mathcal{V})$ is defined by a point x_0 on the grid, a mesh size parameter $h > 0$ and a set of n linearly independent basis vectors $\mathcal{V} = \{\nu_i\}$ so that

$$\mathcal{G}(x_0, h, \mathcal{V}) = \left\{ x \in \mathbb{R}^n : x = x_0 + h \sum_{i=1}^{n} \lambda_i \nu_i \text{ for all } \lambda_i \in \mathbb{Z} \right\}.$$

The mesh size parameter h is adjusted from time to time to ensure a succession of finer grids is generated by the algorithm. This property is crucial for establishing convergence on smooth problems.

Coope and Price [16] provide a convergent template for grid based methods. Their framework generates a sequence of grid local minimizers.

Definition 9. (Grid Local Minimizer). *A point $x \in \mathcal{G}(x_0, h, \mathcal{V})$ is a grid local minimizer of an objective function f with respect to a positive basis $\mathcal{V}^+$ if*

$$f(x + h\nu) \geq f(x) \text{ for all } \nu \in \mathcal{V}^+.$$

From Theorem 7,

$$\nu^{\mathsf{T}} \nabla f(x) \geq 0 \text{ for all } \nu \in \mathcal{V}^+ \Rightarrow \nabla f(x) = \mathbf{0}$$

and thus, a grid local minimizer is a finite difference approximation to this. Under appropriate conditions, the convergence result shows that an infinite sequence of grid local minimizers are generated, for which each cluster point is a local minimizer of f. Furthermore, each cluster point has the same function value because the sequence of iterates have monotonically decreasing function values.

The framework provided by Coope and Price is quite flexible. Convergence holds even if the grids are arbitrarily translated, rotated or sheared relative to one another

and each grid axis may be rescaled independently of the others [16]. This means the grids can try to incorporate second order curvature information, for example, align the grid axes with respect to the principle axes of an approximation quadratic. The pattern search framework in [79] has a single set of grid axes, only rational scaling of the grid is allowed and arbitrary translations are not permitted. Thus, there is greater flexibility in the grid based framework.

1.3.2.1 Hooke and Jeeves Algorithm

This subsection introduces the classical direct search method of Hooke and Jeeves [35]. This method is of particular interest because no gradient information is required and has potential benefits on nonsmooth problems. These potential benefits are discussed in sections 1.4.1 and 1.4.2. An altered Hooke and Jeeves algorithm is presented in Chapter 5 which is provably convergent on nonsmooth problems.

A precise statement of the Hooke and Jeeves algorithm is given in Figure 1.3. The algorithm consists of three loops. The two inner loops (steps 2—3 and steps 2—4) search for points of descent on a grid $\mathcal{G}(x_0, h, \mathcal{V})$, where $\mathcal{V}$ is the canonical basis for $\mathbb{R}^n$. The outer loop (steps 2—5) reduces the mesh parameter h (typically by a factor of 10 [17]) when a grid local minimizer has been located. With x_0 and $\mathcal{V}$ fixed for all iterations, the sequence of grids is nested.

1. Initialize: Set $k = 0$ and $v_0 = \mathbf{0}$. Choose x_0, $h_0 > 0$ and $h_{\min} > 0$.
2. Exploratory Move: Calculate $f(x_k + v_k)$ and form the exploratory vector E_k from $x_k + v_k$.
3. Pattern Move: If $f(x_k + v_k + E_k) < f(x_k)$, then set

$$x_{k+1} = x_k + v_k + E_k \quad \text{and} \quad v_{k+1} = v_k + E_k,$$

   increment k and go to Step 2.
4. If $v_k \neq \mathbf{0}$, set $v_k = \mathbf{0}$ and go to Step 2.
5. If $h_k = h_{\min}$ stop. Otherwise, select $h_{\min} \leq h_{k+1} < h_k$, increment k and go to Step 2.

Figure 1.3: Hooke and Jeeves algorithm

Step 1 sets the iteration counter $k = 0$ and the initial Hooke and Jeeves pattern move v_0 to the zero vector. An initial point $x_0 \in \mathbb{R}^n$ (which is also the grid center), an initial mesh size $h_0 > 0$ and a minimum mesh size $h_{\min}$ are chosen by the user.

Step 2 conducts the exploratory phase of the Hooke and Jeeves algorithm [35]. Each exploratory phase tries to reduce f by altering each coordinate of the current exploratory search point $z = x_k + v_k$ in turn. Each element z_i of z is treated as follows. Firstly z_i is increased by h. If this yields a reduction in f then this increase is retained and the method moves on to the next coordinate. Otherwise the increase is removed, and z_i is decreased by h from its original value. Again, if this reduces f the decrease is retained, otherwise it is abandoned. In either case the process moves on to the next coordinate of z. The exploratory steps are retained in the vector E_k, where initially $E_k = \mathbf{0}$, as demonstrated in Figure 1.4.

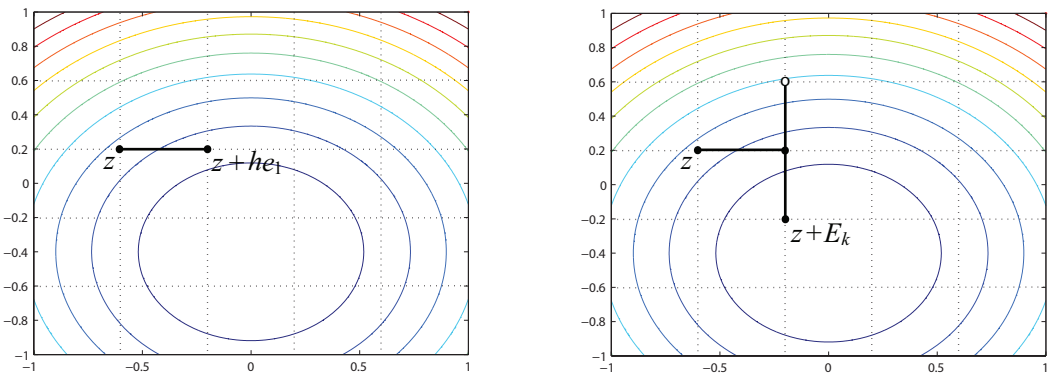


Figure 1.4: Hooke and Jeeves exploratory steps: Increasing z by he_1 gives descent and the step is retained. For the next coordinate $z + he_1$ is increased by he_2. This step increases f and the step is removed. $z + he_1$ is then decreased by he_2, which gives descent, and the step is retained. Thus, $E_k = [h, -h]$ in this example.

Step 3 is executed if descent is found, $f(x_k + v_k + E_k) < f(x_k)$. This step performs a pattern move, probing the promising direction of descent given by previous successful moves. Otherwise, no descent is found and the first inner loop terminates.

Step 4 is executed if v_k is non-zero and sets v_k to the zero vector. This means an exploratory phase is conducted from x_k at Step 2. Otherwise, v_k is the zero vector and a grid local minimizer has been located, terminating the inner loops.

The Hooke and Jeeves algorithm terminates when a grid local minimizer on $\mathcal{G}(x_0, h_{\min}, \mathcal{V})$ has been located. The final iterate, x_k, is the estimate of a local minimizer of f.

The Hooke and Jeeves algorithm is shown to converge to a local minimizer of a continuously differentiable function in [17]. Furthermore, relaxations to the grid are

made in [17], without affecting convergence.

1.3.3 Partial Nonsmooth Convergence Results

This section considers partial convergence results on nonsmooth problems that have been presented by others and shows these results are not sufficient for ensuring convergence to an essential local minimizer. These results are partial in the sense that they guarantee the non-negativity of the Clarke derivative [13] in all relevant directions at each cluster point of the sequence of iterates generated by an algorithm. The Clarke generalized derivative is the *generalized directional derivative* of f in the direction d, defined by

$$f^o(x;d) = \limsup_{\substack{z \to x \\ \lambda \downarrow 0}} \frac{f(z+\lambda d) - f(z)}{\lambda}, \tag{1.10}$$

where $z \in \mathbb{R}^n$ and λ is a positive scalar. However, the non-negativity of the Clarke derivative in all relevant directions (or indeed all directions) does not guarantee the non-existence of a set of descent directions from a cluster point.

In [14] a weak nonsmooth convergence result is presented for the Directional Direct Search framework. Under basic assumptions, the authors show that the sequence of iterates generated has a limit point x_* for which

$$f^o(x_*, \nu) > 0, \text{ for all } \nu \in \mathcal{V}^+, \tag{1.11}$$

where $\mathcal{V}^+$ is a positive basis for $\mathbb{R}^n$. However, (1.11) does not guarantee the non-existence of descent directions from x_*. This can be seen by considering, for example, the function

$$f(x) = \max\{a^\mathsf{T}x, b^\mathsf{T}x\} \text{ such that } x \in \mathbb{R}^2, \tag{1.12}$$

where $a \neq -b$ are unit vectors. The restriction $a \neq -b$ ensures there exists a descent direction for all values of x. Consider using the maximal canonical positive basis $\mathcal{V}^+$, choosing $a = [0.14, -0.98]$, $b = [-0.85, 0.53]$ in (1.12) and trying to locate descent from a point x such that $a^\mathsf{T}x = b^\mathsf{T}x$. The reader can immediately see from Figure 1.5 that a large cone of descent directions exists in such a situation. However, the directional derivative is positive in all directions $\nu \in \mathcal{V}^+$ (bold cross in Figure 1.5), which implies the Clarke derivative is also positive for all $\nu \in \mathcal{V}^+$. Note, this result is independent of the length of ν, i.e. it holds for $\lambda\nu$, where $\lambda > 0$ and finite and $\nu \in \mathcal{V}^+$.

A stronger convergence result is proposed by Audet and Dennis for the MADS algorithm [5]. Their method looks asymptotically in all directions at each cluster point

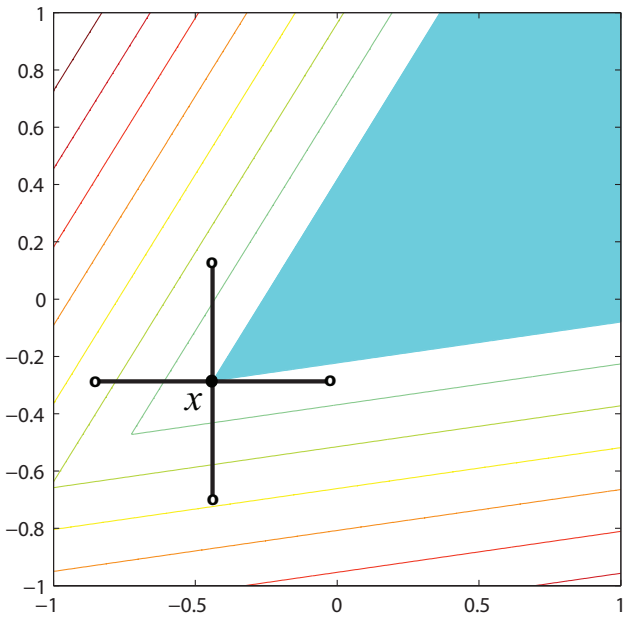


Figure 1.5: Contour plot of (1.12) with $a = [0.14, -0.98]$ and $b = [-0.85, 0.53]$. A large cone of descent directions from a point x exists (shaded region), centered on the line $a^\mathsf{T} z = b^\mathsf{T} z$ where $z \in \mathbb{R}^2$.

x_* of the sequence of iterates generated by the algorithm. They go on to show that

$$f^o(x_*, d) > 0, \text{ for all } d \in \mathbb{R}^n. \tag{1.13}$$

However, the non-negativity of the Clarke derivative in all directions still does not guarantee the non-existence of descent directions from x_*.

The non-negativity of the Clarke derivative in all directions at x is a necessary, but not a sufficient condition for x to be an essential local minimizer of f [62]. Consider a directionally differentiable function f. If $f^o(x, d) < 0$ for some direction d, then from the definition of the Clarke derivative we have

$$f(x + \epsilon d) < f(x) + \frac{\epsilon}{2} f^o(x, d), \tag{1.14}$$

for all sufficiently small $\epsilon > 0$. Hence, there exists a descent direction at x in the direction d. Therefore, $f^o(x, d) \geq 0$ is a necessary condition for x to be an essential local minimizer of f.

It can be seen that $f^o(x, d) \geq 0$ is not sufficient by considering, for example, the

function $f = -|x|$ in one dimension. The Clarke derivative at the origin is given by

$$f^o(0,d) = \limsup_{\substack{z\to 0\\ \lambda\downarrow 0}} \frac{-|z+\lambda d| + |z|}{\lambda}.$$

For $d > 0$ choose $z = -2\lambda d$ and the limit becomes

$$f^o(0,d) = \limsup_{\lambda\downarrow 0} \frac{-(\lambda d) + (2\lambda d)}{\lambda} = d = |d|. \qquad (d > 0)$$

Similarly for $d < 0$ choose $z = -2\lambda d$ giving

$$f^o(0,d) = \limsup_{\lambda\downarrow 0} \frac{+(\lambda d) - (2\lambda d)}{\lambda} = -d = |d|. \qquad (d < 0)$$

Hence, the Clarke derivative is positive in all non-zero directions from the origin, even though every non-zero d is a descent direction at $x = 0$. Therefore, $f^o(x,d) \geq 0$ is not a sufficient condition for x to be an essential local minimizer of f.

A more interesting function is presented by Price, Robertson and Reale [62],

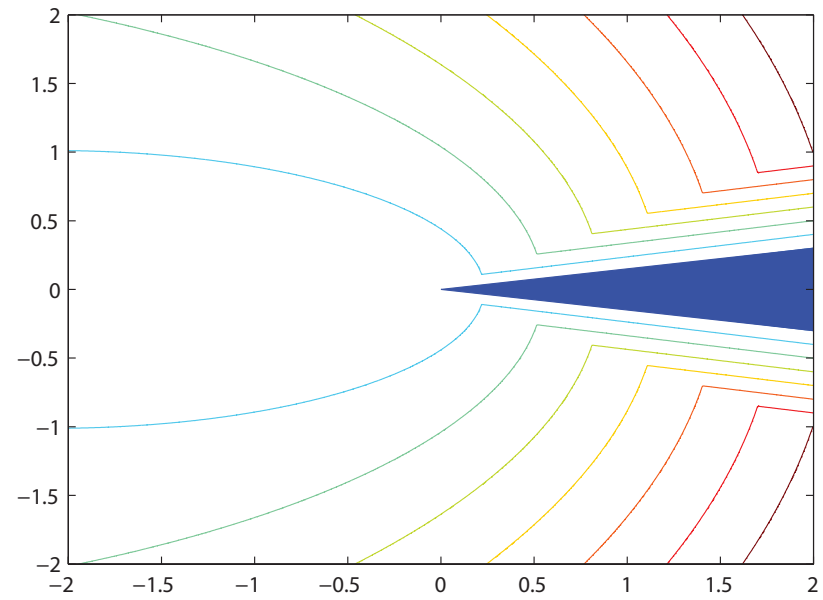
$$f = \begin{cases} 3(2|x_2| - x_1) + (0.9 + \sqrt{5}/2)x_1 & x_1 \geq 2|x_2| \\ 0.9x_1 + \sqrt{x_1^2 + x_2^2} & \text{otherwise.} \end{cases} \qquad (1.15)$$

The Clarke derivative at the origin, $f^o(\mathbf{0}, d)$, is positive in all non-zero directions d. However, there is a set of descent directions centered on e_1 emanating from the origin for which the directional derivative is negative, see Figure 1.6.

For general problems, the non-negativity of the Clarke derivative will not give convergence to essential local minimizers. Another way to obtain convergence, albeit a computationally intensive one, is to eventually look everywhere in the neighborhood of the terminating iterate x_k. However, although checking

$$f(x_k + d) \geq f(x_k) \text{ for all } \|d\| < \epsilon \qquad (1.16)$$

requires an n dimensional global optimization, checking $f^o(x_k, d) \geq 0$ for all d unit in $\mathbb{R}^n$ $(n > 1)$, requires an $n-1$ dimensional global optimization. Thus, even for relatively small n, the computational effort for both methods is similar. If (1.16) is satisfied no descent is found and x_k is an essential local minimizer of f. To this end, the Clarke derivative approach is replaced with a method that searches directly for points of descent using global optimization methods, details of which follow in Section



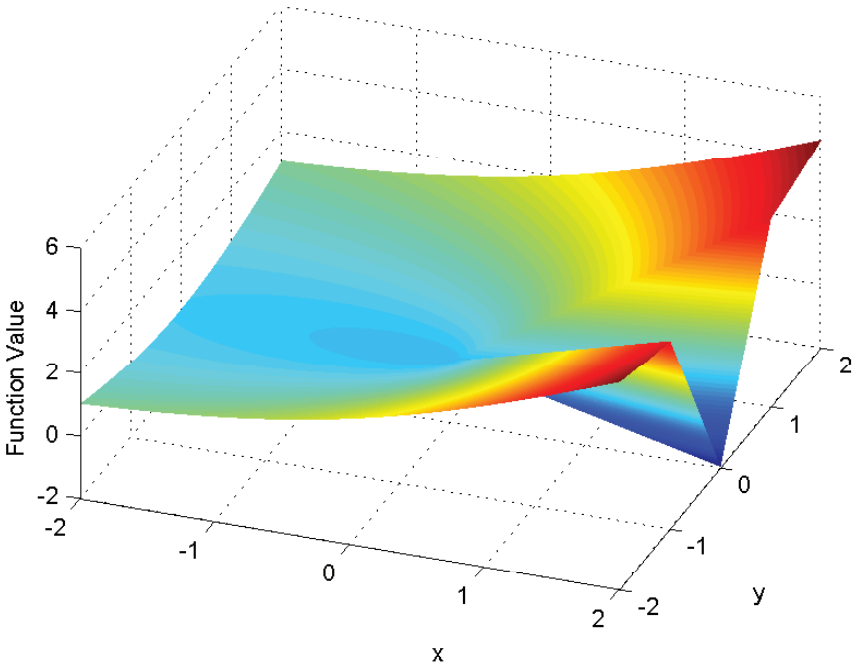


Figure 1.6: The contour plot and graph of the two dimensional function (1.15), where the Clarke derivative is positive in all directions at $x = \mathbf{0}$, is shown. The directional derivative along e_1 is negative and so there exists a cone of descent directions, indicated by the shaded region in the contour plot.

1.7.

1.4 Trajectory Following Optimization

This section considers trajectory following local optimization methods and illustrates a potential usage for nonsmooth optimization. These methods solve (1.1) by following a solution curve $x(t)$ of a system of ordinary differential equations (ODE). The interested reader is referred to [9, 67, 72] and references therein for a survey of trajectory following algorithms and numerical integration methods to obtain $x(t)$.

Trajectory following methods can be grouped into two categories depending on the order of the ODE that is used. Here $\dot{x}, \ddot{x} \in \mathbb{R}^n$ are vectors with elements dx_i/dt and d^2x_i/dt^2 for $1 \leq i \leq n$, respectively.

Definition 10. (First order ODE method.) *A trajectory following optimization method that solves an equation of the form $\dot{x} = y(x(t))$, with $t > 0$ and $x(0) = \mathbf{0}$ is called a first order ODE method.*

Definition 11. (Second order ODE method.) *A trajectory following optimization method that solves an equation of the form $\ddot{x} = y(x(t))$, with $t > 0$, $x(0) = \mathbf{0}$ and $\dot{x}(0) = \mathbf{0}$ is called a second order ODE method.*

The first order ODE method defined by choosing $y(x(t)) = -\nabla f(x(t))$, yields the continuous steepest descent trajectory [9]. Furthermore, with

$$y(x(t)) = -G^{-1}(x(t))\nabla f(x(t))$$

the continuous Newton trajectory [9] is obtained, where G is the (non-singular) Hessian matrix. Assuming f is twice continuously differentiable and $G(x(t))$ is positive definite, both the Newton and steepest descent trajectory define a path $x(t)$ to a local minimizer of f.

The primary interest here is minimizing nonsmooth objective functions. Therefore, having a first order ODE optimization method that replies on having derivative information (possibly second order) is problematic. Although this information can be approximated using finite differences, it is the author's opinion that there is no advantage in doing so for nonsmooth minimization. For these reasons, first order ODE methods are considered no further. However, second order ODE methods do have an interesting property which can be exploited in nonsmooth minimization, details of which follow in the next subsection.

1.4.1 Physical Analog to Function Minimization

This subsection considers a physical analog to function minimization proposed by Snyman [70], where the motion of a particle of unit mass in an n-dimensional conservative force field is considered. Here, the objective function f represents the potential energy of the particle and the total energy, consisting of potential and kinetic energy, is conserved. The motion of such a particle is described by the second order system

$$\ddot{x} = -\nabla f(x(t)), \tag{1.17}$$

$$x(0) = x_0, \qquad \dot{x}(0) = \mathbf{0}, \qquad t \geq 0.$$

The conservation of energy condition in the system leads to,

$$K_E(x_t) + P_E(x_t) = K_E(x_0) + P_E(x_0) = f(x_0), \tag{1.18}$$

where

$$K_E(x) = \frac{1}{2}\|\dot{x}\|^2 \text{ and } P_E(x) = f(x) \tag{1.19}$$

define the kinetic and potential energy of the particle respectively. From (1.18), the particle is in continuous motion and bounded above by the initial function value $f(x_0)$ ($K_E = 0$ and P_E is maximized). Therefore, the trajectory can surmount *ridges* with function values less than $f(x_0)$ and hence leave the neighborhood of local minimizer if visited by the trajectory. To ensure motion towards a local minimizer an artificial damping term $\lambda\dot{x}$, where λ is a positive constant, can be added to the left hand side of (1.17). However, the choice of λ can be problematic. Too small and the particle may endure a long period of oscillation about a local minimizer x_*. Whereas, for λ too large $\ddot{x} + \lambda\dot{x} \approx \lambda\dot{x}$, which reduces the system to

$$\dot{x} \approx -\nabla f(x(t))/\lambda.$$

Thus, for large λ intolerably slow progress is made. From (1.18) and considering two consecutive times on the trajectory, the following relation is obtained

$$-\Delta f(x) = \Delta K_E(x). \tag{1.20}$$

Therefore, by monitoring the sign and magnitude of $\Delta K_E(x)$ motion toward a minimizer can be obtained. For example, if $K_E(x_i) - K_E(x_{i+1}) < 0$ is observed, then restarting from x_i with $K_E(x_i)$ set to zero forces motion toward x_* provided $x(t)$ is

bounded.

For nonsmooth minimization there may be some merit in allowing the second order trajectory to follow its *natural path* allowing f to increase along $x(t)$. The second order term gives the particle momentum, allowing the particle to potentially roll though nonsmooth regions of f because uphill steps are possible. Provided some damping is applied to the system, an interesting optimization method is proposed. This is investigated further in Chapter 5. The interested reader is referred to Figure 5.3, where the relative merits of the second order trajectory are shown on a nonsmooth function.

1.4.2 Connections with Classical Direct Search

This subsection proposes a derivative free approximation to a second order trajectory following optimization method. With the dependence on derivative information removed, this method can be applied to nonsmooth problems. Interestingly, this method is a special case of the classical Hooke and Jeeves algorithm [35].

Consider the second order trajectory defined by (1.17). Solving equation (1.17) using a central difference approximation for $\ddot{x}$, one obtains a trajectory defined by

$$x_{k+1} - 2x_k + x_{k-1} = -h^2 \nabla f(x_k). \tag{1.21}$$

Replacing the right hand side of (1.21) with the vector of the Hooke and Jeeves exploratory moves E_k (see Section 1.3.2.1) operating on a grid $\mathcal{G}(x_0, h)$ with canonical basis vectors, a crude approximation to the direction and magnitude of $h^2 \nabla f(x_k)$ is obtained. Substituting E_k into (1.21) we have

$$x_{k+1} = 2x_k - x_{k-1} + E_k. \tag{1.22}$$

Using (1.22) and terminating when $x_{k+1} = x_k$ and $E_k = 0$, an iterative grid based approximation to a second order trajectory is obtained. Noting that (1.22) can be expressed by the following system

$$x_{k+1} = x_k + v_k + E_k, \quad \text{and} \quad v_{k+1} = v_k + E_k,$$

where $v_0 = \mathbf{0}$, the trajectory defined by (1.22) is that of the Hooke and Jeeves algorithm. It is simply one iteration (fixed h) of the classical algorithm whereby uphill steps are taken. Using this connection the classical Hooke and Jeeves algorithm is extended into

a non-descent algorithm in Chapter 5. Furthermore, the relative merits of allowing for uphill steps are empirically analyzed in Chapter 7.

1.5 Global Optimization

Although this thesis is primary interested in local nonsmooth optimization, the method proposed uses global optimization techniques. The problem of global optimization has a deceptively simple mathematical description,

$$\min_x f(x) \text{ such that } x \in \Omega, \tag{1.23}$$

where the objective function f maps $\mathbb{R}^n \to \mathbb{R}$. Here Ω is a compact subset of $\mathbb{R}^n$ and is called the optimization region. If $\Omega \equiv \mathbb{R}^n$ then (1.23) is called an *unconstrained* problem, otherwise the problem is referred to as *constrained*. Finding a global optimum is usually harder than finding a local optimum. To solve (1.23) a global minimizer is sought.

Definition 12. (Global Minimizer). *Let $\Omega \subset \mathbb{R}^n$. A point $x_* \in \Omega$ such that*

$$f(x_*) \leq f(x) \text{ for all } x \in \Omega, \tag{1.24}$$

is called a global minimizer of f.

Definition 13. (Global Minimum). *If a point $x_* \in \Omega$ is a global minimizer of f, then $f(x_*)$ is called the global minimum of f.*

There can only be one global minimum for an objective function f, but there can be multiple global minimizers. The notation $f(x_*) = f_*$ is used throughout the thesis to denote the minimum function value.

In practice the problem is somewhat relaxed to finding a *global minimum to within* ϵ, i.e. given an $\epsilon > 0$ find a function value f_* such that $f_* \leq f(x) + \epsilon$ for all $x \in \Omega$. A point x_* such that $f(x_*)$ is a global minimum to within ϵ is called a *global minimizer to within* ϵ. A further relaxation is to find a global minimum to within ϵ with high probability.

There have been many methods proposed to solve (1.23). These methods can be largely classified as being either deterministic or stochastic. Stochastic methods have an element of randomness whereas deterministic methods do not.

Two popular deterministic methods amongst practitioners are Interval methods and Branch and Bound methods. Interval methods use interval arithmetic [50] to provide bounds on functions by means of interval operations. An example of such a method is the interval Newton algorithm proposed by Moore [50]. Such methods are beyond the scope of this thesis, but the interested reader is referred to [53] for details. Interval arithmetic can also be applied to the Branch and Bound framework. These methods divide Ω into finitely many sub-domains and bounds on the global minimum in each sub-domain are calculated. This information is used to reject sub-domains that cannot contain the global minimum of Ω. Eventually only sub-domains containing the global minimum remain. The interested reader is referred to [76] for further details on Branch and Bound methods.

Another deterministic approach is to sample Ω using a quasi-random sequence of numbers. The use of quasi-random sequences is revisited in Section 3.6.

Stochastic methods have become popular amongst practitioners. The element of randomness in these methods can make them very successful on problems for which deterministic methods fail. The interested reader is referred to [28] for a detailed review of three popular stochastic methods: Simulated Annealing (see later), Tabu Search and Genetic algorithms. Of particular interest in this thesis are Random Search global optimization methods. These methods require no gradient information and thus, can be directly applied to nonsmooth or discontinuous problems. In particular, these methods can be applied to Barrier functions.

Random Search methods are discussed in more depth in Chapter 2 where a partitioning Random Search framework is presented.

1.6 Nonsmooth Local Optimization vs Global Optimization

This section shows that the problem of finding a descent direction of a nonsmooth function in $\mathbb{R}^n$ is closely related to global optimization in $(n-1)-$dimensions. This connection was first proposed by Price, Reale and Robertson [61] and is included here for completeness of the thesis. Exploiting this connection provides a powerful method to solve nonsmooth optimization problems, details of which follow in the next section.

Consider the global optimization problem

$$\min y(x) \text{ subject to } x \in [-1,1]^{n-1}, \tag{1.25}$$

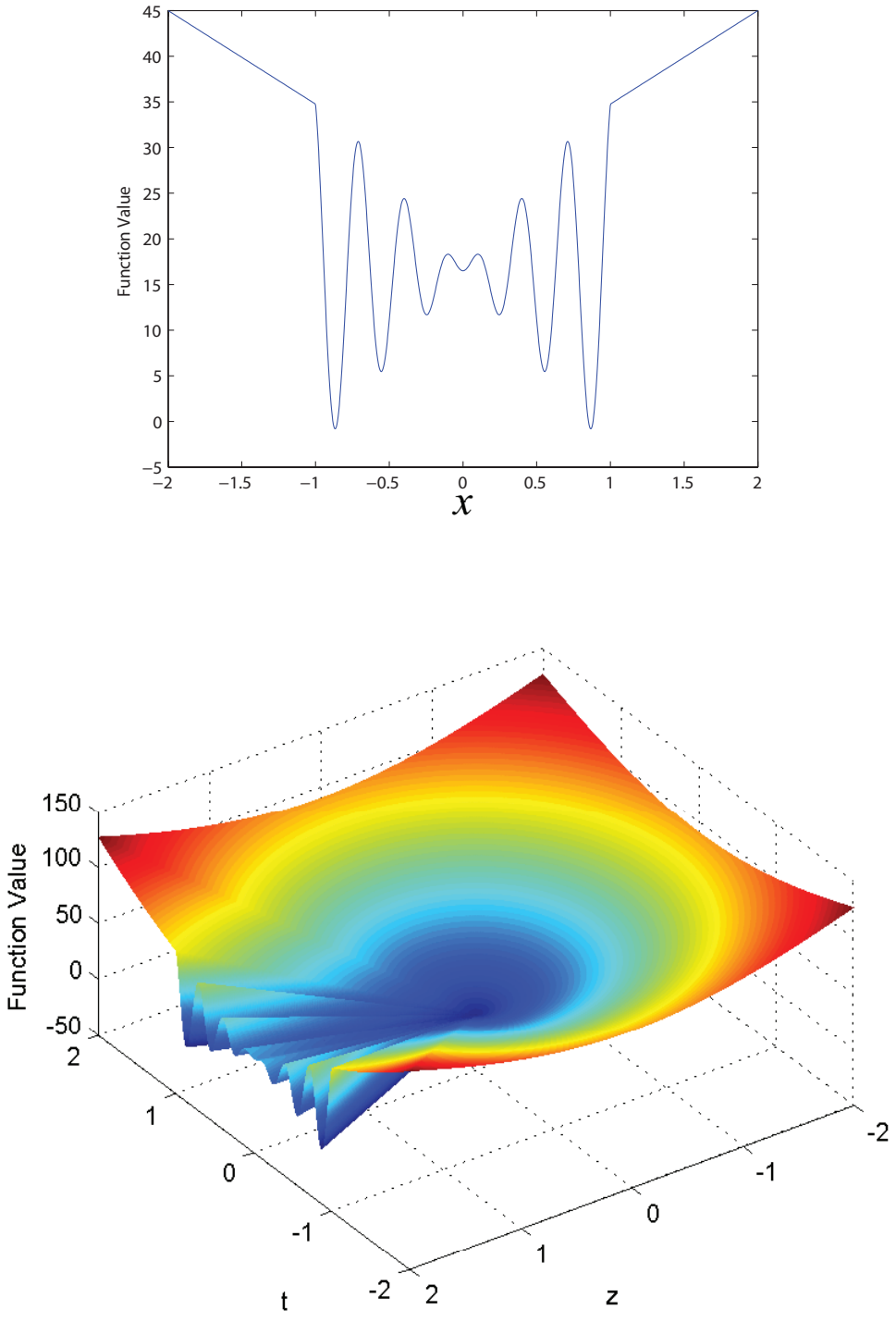


Figure 1.7: The top figure shows ψ and bottom shows Ψ with $y(x)$ defined by (1.26). From ψ it is clear that two global minimizers with negative function values exist. The connection between finding a descent direction from the origin of Ψ and the global minimum of ψ is clear from the front left edge of the graph of Ψ.

with a global minimum y_*. To solve (1.25) a point to within $\epsilon > 0$ of y_* is sought, i.e. a function value no more than $y_* - \epsilon$. Noting that the addition of a constant to (1.25) leaves the optimization problem unchanged, $y_* - \epsilon$ is subtracted from y. Now any negative function value is also a global minimum to within ϵ of (1.25). This problem can be expressed as the problem of finding a descent direction of a nonsmooth function in n dimensions. The nonsmooth function $\Psi(t, z)$ is defined in terms of an intermediate function

$$\psi(x) = \begin{cases} y(x), & \|x\|_\infty \leq 1, \\ (\|x\|_\infty - 1)\lambda + (2 - \|x\|_\infty)y(x/\|x\|_\infty), & 1 < \|x\|_\infty \leq 2, \end{cases}$$

where λ is a positive constant such that

$$\lambda > \max y(x) \text{ subject to } x \in [-1, 1]^{n-1}.$$

The continuous function ψ extends y to $[-2, 2]^{n-1}$ in such a way that y remains unchanged on $[-1, 1]^{n-1}$. Outside this region ψ rises linearly to λ, where $\|x\|_\infty = 2$ (see Figure 1.7). Using $x = (t, z)$ where $z \in \mathbb{R}$, define

$$\Psi(\psi) = \begin{cases} \psi(2t/z)\sqrt{z^2 + \|t\|^2} & \|t\|_\infty < z \\ \lambda\sqrt{z^2 + \|t\|^2} & \text{otherwise.} \end{cases}$$

Ψ is linear along each ray emanating from the origin and so locating a descent direction for Ψ is equivalent to locating a point (t, z) such that $\Psi(t, z) < 0$. Clearly $\Psi(t, z) < 0$ if and only if $\psi(2t/z) < 0$, which can only occur when $\|t\|_\infty < z$. From the form of ψ there are two ways $\psi(2t/z) < 0$ depending on the magnitude of $\|2t/z\|_\infty$. If $2\|t\|_\infty \leq z$ then we require $y(2t/z) < 0$, otherwise $y(t/\|t\|_\infty) < 0$ is required. In both cases a solution to (1.25) is obtained (the $(n-1)-$dimensional global optimization problem).

The reader is referred to Figure 1.7 for an illustrative example of $\Psi(t, z)$ in 2 dimensions to accompany the description above. The objective function $y(x)$ in the 1 dimensional global optimization problem is given by

$$y(x) = 16.5 + 20x\sin(20x) \text{ on } x \in [-1, 1]. \tag{1.26}$$

The positive constant $\lambda = 45$ has been used in $\psi(x)$.

Applying a global optimization method to the $(n-1)-$dimensional set of directions emanating from a point $x \in \mathbb{R}^n$ can locate a descent direction. However, this assumes that f is directionally differentiable at x. To remove this restriction a global optimiza-

tion method is applied to an $n-$dimensional neighborhood of x, which aims to locate a point of descent directly. This type of optimization is referred to as *localized global optimization.*

1.7 Optimization Method

This section presents the general method for solving nonsmooth optimization problems used throughout this thesis. These methods replace the Clarke derivative approach of Audet and Dennis [5] with one using a series of localized global optimization phases. Recall from Section 1.3.3 that in the Clarke derivative approach, only partial nonsmooth convergence results are possible. In particular, the non-negativity of the Clarke derivative in all directions at x_* is necessary, but not sufficient for x_* to be an essential local minimizer f and implicity requires $(n-1)-$dimensional global optimization. In our approach an exhaustive search is performed in an $n-$neighborhood of a cluster point, x_*. The exhaustive search ensures x_* is an essential local minimizer of f.

The methods proposed in this thesis have two modes of operation: a local phase and a localized global phase. A precise statement of the framework is given in Figure 1.8. At each local phase, a local optimization method is applied until no descent is forthcoming from an iterate x_{i^*}. A localized global optimization phase is then performed in a neighborhood Ω of x_{i^*} such that $x_{i^*} \in \Omega$ and $m(\Omega) > 0$. If a point $z \in \Omega$ such that $f(z) < f(x_{i^*})$ is located, the localized global optimization phase terminates. A new local phase is then initiated from the point z. Otherwise, no points of descent are found in Ω, confirming that x_{i^*} is an essential local minimizer of f.

The local phase in this approach is used to potentially increase the numerical performance of the framework on parts of f which are smooth. Typically local search algorithms are computationally cheaper than global algorithms. Therefore, it is advantageous to make as much progress toward an essential local minimizer of f as possible before conducting the computationally expensive localized global optimization phase. Ideally an algorithm conforming to the above framework spends most of its time in the local phase. There is great freedom in choosing which local search method to use, however, some methods are preferred over others. Choosing a direct search method, for example, is advantageous because no gradient information is required and here f is assumed to be nonsmooth. The implementation of the local search is unspecified. For example, one could run the local search until its stopping conditions are satisfied, or terminate, for example, before a mesh reduction. There are many possibilities.

1. Initialize: Choose $z_0 \in \mathbb{R}^n$ and set $k = 0$.
2. Local phase: Execute iterations of a local optimization procedure from z_k, generating a sequence of iterates $\{x_i\}_{i=1}^{i^*}$.
3. Localized global phase: Execute a global optimization procedure (GP) in a neighborhood Ω of x_{i^*} until a point z such that $f(z) < f(x_{i^*})$ is located, or the stopping conditions of GP are satisfied.
4. If stopping conditions of GP are satisfied, terminate algorithm with $x_* = x_{i^*}$. Otherwise set $z_{k+1} = z$, increment k and goto (2).

Figure 1.8: Nonsmooth optimization framework

The computationally expensive localized global phase gives convergence on nonsmooth or discontinuous problems, provided the sequence of iterates generated in the local phase $\{x_i\}$ is bounded. In fact, the local phase can be replaced with: set $x_{i^*} = z_k$, effectively removing the local phase, without affecting convergence properties. Convergence is demonstrated in both Chapters 3 and 5, where two algorithms conforming to this framework are presented.

1.8 Thesis Overview

This chapter concludes with a brief overview of the remainder of this thesis. Chapter 2 considers the localized global optimization phase explicitly, where a new random search algorithm is presented. This algorithm is extended further in Chapter 3, producing a nonsmooth local optimization algorithm called CARTopt. Stopping rules are developed in Chapter 4 to make CARTopt a practical algorithm. Chapter 5 develops a trajectory following nonsmooth local optimization method. This algorithm uses an altered Hooke and Jeeves algorithm as the local search procedure and the CARTopt algorithm as a localized global optimization phase. Chapter 6 extends the CARTopt algorithm into a filter based algorithm for solving nonsmooth nonlinear programming problems. Empirical testing of the algorithms on a variety of nonsmooth optimization problems is given in Chapter 7. The thesis concludes with a summary and future research ideas.

Chapter 2

Localized Global Phase

This chapter considers the localized global optimization phase in the two phase approach to nonsmooth minimization (see Section 1.7). The use of global optimization here is strange in the sense that a series of such problems are solved. The objective function remains the same in each problem, but the search region over which optimization occurs changes. The latter fact can make the use of deterministic methods problematic. Consider, for example, exploring the search region using a quasi-random number sequence like the Halton sequence [31] (see also Section 3.6.1). This sequence is very efficient at covering a space in low dimensions. In our approach successive search regions may overlap, displacing various parts of the Halton sequence relative to one another. This can potentially destroy the efficiency of the covering, leaving parts of the search region unexplored for multiple iterations. The use of stochastic methods removes this problem, particularly random search methods.

Each localized global optimization phase is conducted in a subset of the optimization region $\mathcal{S} \subset \mathbb{R}^n$ centered on x_k, defined by the set

$$\Omega = \{x \in \mathcal{S} : \ell_i \leq x_i \leq u_i \text{ for all } i = 1, \ldots, n\},$$

where $\ell_i < u_i$ are finite. Here x_k is an iterate for which no descent was forthcoming in the local phase. Interestingly, there is only one localized global phase in which the actual global minimum f_* is sought, where $f_* \leq f(x)$ for all $x \in \Omega$. For all other localized global phases only a point of descent is required, from which a new local phase is started. This chapter considers locating the global minimum on Ω. Although there is only one global minimum for f, there can be multiple global minimizers. The primary goal of the localized global optimization phase is to confirm that an essential local minimizer has been located and so locating any global minimizer x_* is sufficient.

Firstly, existing random search global optimization algorithms are considered. Section 2.2 reviews partitioning random search algorithms. A new partitioning strategy based on classification techniques from statistical pattern recognition is introduced in Section 2.3. Section 2.4 proposes a particular partition based on Classification and Regression Trees (CART). This partition has desirable properties and can be used in a new global optimization framework, Adaptive Partition Random Search (APRS), presented in Section 2.5. A particular APRS algorithm, CARTopt, is developed in Chapter 3 and an algorithm in Chapter 5 uses CARTopt for the localized global optimization phase. The APRS algorithm is shown to converge to a globally optimal point when the objective function is assumed to be nonsmooth.

2.1 Random Search Optimization

Random search global optimization methods are of interest here because they do not require the use of gradients and thus, can be directly applied to nonsmooth or discontinuous problems. Furthermore, they are not affected by a succession of overlapping search regions. This section and the next provide a review of random search methods, which is by no means exhaustive. The interested reader is referred to [2, 74, 83, 89] and references therein for more algorithms and to [91] for a comprehensive treatment of random search.

The simplest and most robust of all random search algorithms is Pure Random Search (PRS) [38]. PRS samples the objective function f uniformly over Ω and when stopping conditions are satisfied, the lowest function value obtained is the estimate of the global minimum. Although PRS can be shown to converge to the global minimum with probability one, notoriously it is often slow in practice. PRS can be generalized by sampling f from a non-uniform distribution over Ω. Such an algorithm can be used when (i) information about f is available, allowing for subsets of Ω to be considered more promising than others, or when (ii) the sampling problem for the uniform distribution on Ω is hard or practically impossible.

There has been considerable work done by numerous authors to increase the efficiency of PRS yielding a variety of random search algorithms. One method that has gained prominence in the optimization community is the Multi-Start algorithm [38]. Multi-Start alternates between local and global phases. Each global phase generates a batch of points from a uniform distribution over Ω, where each point is called a seed. A local search procedure is then conducted from each seed, yielding a local minimum

if the objective function is smooth, completing the local phase. If stopping conditions are not satisfied another global phase is conducted and the method repeats. Otherwise, the local minimum with the smallest function value is the estimate for the global minimum.

The Multi-Start algorithm has also been extended to more advanced algorithms, for example, Multi-Level Single Linkage [38, 39]. The main idea behind these methods is to apply a clustering technique to the batch of random points. The clustering aims to group points together that would share the same local minimizer, if a local search procedure was applied from each element of the cluster. Each cluster is defined by one seed point and hence, only one local search is performed from each cluster.

There have also been various point-to-point random search methods. These methods generate a single point using some randomized scheme and a decision is made on whether to accept the point as the next iterate or generate another. One popular method is the Simulated Annealing (SA) algorithm [91]. This algorithm mimics the behavior of the physical process of annealing in metallurgy. At each iteration, the current sample point is replaced by a nearby point, chosen with a probability that depends on the difference between the corresponding function values and on a global parameter t (temperature). Initially t is large meaning points are chosen at random (including points of ascent) but as the algorithm proceeds, $t \to 0$ and only points of descent are accepted. Allowing for points of ascent potentially stops the algorithm getting stuck at local minima.

Another method, Accelerated Random Search [3, 63], systematically reduces the hypercube shaped search region (initially Ω) over the current point x_k with the lowest function value. Points are generated in the successively smaller search regions centered on x_k until descent is made, the search then returns back to Ω. This counter intuitive approach allows the algorithm to focus the search in the neighborhood of local minima, yielding high accuracy solutions. An automatic restart feature occurs when a minimum search region size is reached, returning the search to Ω. This feature reduces the risk of missing the global solution.

Recursive Random Search [86] also systematically reduces the search region. This algorithm applies PRS until the probability of reducing f further is sufficiently small. The search region is then reduced and centered over the point with the lowest function value. PRS is applied in the new, smaller search region until the probability of reducing f further is sufficiently small once again. The method repeats until the smallest search region is obtained. The main idea behind this algorithm is to keep PRS searching in the *high efficiency* phase of random sampling. The idea of high efficiency sampling is

investigated further in Section 3.5.1 of this thesis.

To conclude this section the largely theoretical algorithm Pure Adaptive Search (PAS) [88] is mentioned. Although this algorithm can only be implemented efficiently on very special functions, it provides a theoretical ideal for random search. At the k^{th} iteration PAS evaluates $f(z_k)$, where z_k is drawn from a uniform distribution over the level set

$$L(z_{k-1}) = \{z_k \in \Omega : f(z_k) < f(z_{k-1})\}.$$

If the set level set of $f(z_k) - \epsilon$ is empty, then a global minimum to within ϵ is known and the algorithm terminates.

To implement PAS efficiently it is necessary to know all the level sets of f and a point must be drawn uniformly from a level set, both non-trivial requirements. PAS can be implemented, although extremely inefficiently, by simply applying PRS and only accepting points which reduce the current lowest function value. Attempts have been made to construct realizations of PAS. These include the Hit and Run algorithm [87], Hide and Seek algorithm [65] and Somewhat Adaptive Search [6]. The interested reader is referred to the papers above for details and also to Z. B. Zabinsky's book, Stochastic Adaptive Search for Global Optimization [89]. In Section 2.5, a new algorithmic framework is presented, which may be considered as a new *realization* of PAS.

2.2 Partitioning Algorithms

Another technique for increasing the efficiency of random search algorithms is to form a partition on Ω. To illustrate the advantages of forming a partition consider maximizing an objective function f using the following two approaches:

(i) applying PRS on Ω using N points; and

(ii) partitioning Ω into N sub-regions A_i of equal positive measure and drawing one point randomly from each, where $1 \leq i \leq N$.

Let $F(y)$ and $F_i(y)$ be the cumulative distribution functions of objective function values induced from uniform sampling over Ω and each A_i, respectively, so that

$$F(y) = \frac{1}{N} \sum_{i=1}^{N} F_i(y).$$

Let $f_{\max}$ denote the largest function value obtained out of N draws over Ω. Then for approach (i) and approach (ii) after N draws we have

$$Pr(f_{\max} \leq y|\text{approach (i)}) = \left(\frac{1}{N}\sum_{i=1}^{N} F_i(y)\right)^N \tag{2.1}$$

and

$$Pr(f_{\max} \leq y|\text{approach (ii)}) = \prod_{i=1}^{N} F_i(y). \tag{2.2}$$

Noting that the arithmetic mean of a list of non-negative real numbers is greater than or equal to the geometric mean of the same list,

$$\frac{1}{N}\sum_{i=1}^{N} F_i(y) \geq \left[\prod_{i=1}^{N} F_i(y)\right]^{1/N} \tag{2.3}$$

and

$$Pr(f_{\max} > y) = 1 - F(y), \tag{2.4}$$

we have

$$Pr(f_{\max} > y|\text{approach (i)}) \leq Pr(f_{\max} > y|\text{approach (ii)}). \tag{2.5}$$

That is,

$$E(f_{\max}|\text{approach (i)}) \leq E(f_{\max}|\text{approach (ii)}),$$

where $E(.)$ denotes mathematical expectation. Therefore, we would expect to obtain a larger f value by simply partitioning Ω into a set of sub-regions of equal positive measure and drawing one sample from each. Furthermore, with

$$\min\{f(x) : x \in \Omega\} \equiv -\max\{-f(x) : x \in \Omega\}, \tag{2.6}$$

a similar result is obtained for minimization,

$$E(f_{\min}|\text{approach (i)}) \geq E(f_{\min}|\text{approach (ii)}),$$

where $f_{\min}$ is the lowest value obtained from sampling. Thus, the efficiency of random search can be increased by forming a partition on Ω.

The partition can also be used to identify promising subsets of Ω where f is found to be relatively low. The sampling distribution can then be updated accordingly, drawing more samples from promising sub-regions, rather than simply sampling Ω uniformly.

These partitioning algorithms can be either adaptive, whereby the partition is updated at each iteration, or use a fixed partition defined during the first iteration.

To successfully implement a fixed partition random search algorithm information about the objective function f is usually required in advance, however, such information is usually not available. To gain information about f a general partition is imposed, often a collection of $i > 1$ hyper-rectangular sub-regions A_i of equal positive measure such that

$$\bigcup_i A_i = \Omega \text{ and } A_i \cap A_j = \emptyset \text{ for all } i \neq j.$$

A phase of random sampling is usually performed in each sub-region giving an indication of the behavior of f. This information is used to identify promising subsets of Ω. One method, for example, is Stratified Random Search [25]. Using both the partition and function values from the random sampling phase, a non-uniform sampling distribution is obtained. All subsequent samples are drawn from the updated distribution, concentrating numerical effort where f is relatively low.

Adaptive partitioning algorithms update the partition at each iteration by dividing promising subsets of Ω further. Thus, a sequence of nested partitions is generated by the algorithm. Choosing which sub-region(s) to partition further gives rise to a variety of algorithms, many based on partitioning sub-regions with relatively low objective function values further. The interested reader is referred to [20, 54, 78] for a variety of *promise measures* for selecting which sub-regions to partition further and a host of algorithms. Interestingly, Shi et al. [69] introduced the Nested Partitions algorithm which combines a global and local search phase in a similar way to the Multi-Start algorithm. Each sub-region of the partition is sampled, followed by a local search from each point generated. The set of points produced from each local search are used to determine which regions are promising and require further partitioning.

Partitioning random search algorithms are of interest here not only because they increase the efficiency of random search, but because they provide an efficient way of identifying promising subsets of Ω at minimal cost. For most iterations, the localized global optimization phase is only required to locate a point of descent. Therefore, adapting the sampling distribution to increase computational effort in promising sub-regions is advantageous, rather than sampling Ω uniformly. However, the use of nested (hyper-rectangular) partitions, as described above, is somewhat restrictive. Here a more dynamic partitioning strategy is considered, where an entirely new partition is formed at each iteration using useful information obtained about f from previous iterations. Furthermore, each partition is not necessarily nested with respect to previous

partitions. This partition on Ω is formed using classification methods from statistical pattern recognition.

2.3 Classification and Pattern Recognition

Statistical classification is a procedure for assigning unknown objects to one of several predefined categories using quantitative information inherent to the objects and a training data set of previously classified objects. The finite set of categories has the form $\mathcal{C} = \{\omega_1, \omega_2, \ldots, \omega_j\}$, with each ω_i referred to as a category label. This thesis is interested in optimization and so the objects to be classified are sample points $x \in \Omega$. A classifier is the mapping $\mathcal{T}$ that assigns a category label to each $x \in \Omega$, more formally $\mathcal{T}(x) : \Omega \to \mathcal{C}$. Another way to think about a classifier is to define A_i as the subset of Ω on which $\mathcal{T}(x) = \omega_i$, that is,

$$A_i = \{x \in \Omega : \mathcal{T}(x) = \omega_i\}.$$

The sets A_i, where $1 \leq i \leq j$, are disjoint and $\cup_i A_i = \Omega$, forming a partition on Ω. The purpose of the partition is to define subsets on Ω where f is relatively low. Here we choose two categories $\mathcal{C} = \{\omega_L, \omega_H\}$, points with relatively low and high function values, respectively. Therefore, Ω is partitioned into two categories. A more sophisticated partition could be formed by choosing more categories. However, for optimization purposes two is sufficient and simple.

There have been many classifiers proposed, however, there is no general classifier that performs best on a general classification problem. Usually the user decides upon which technique may best suit the problem, exploiting known structure or employing a method which gives a desired structure in the partition. Examples of classifiers include decision tree classifiers, neural networks, support vector machines and k-nearest neighbor [24, 77]. Each classification method produces a different partition on Ω. However, decision tree classifiers form a partition with a desirable structure that can be exploited and other classifiers are considered no further.

2.3.1 Classification Trees

An intuitive way to classify an object is through a sequence of questions, where by the next question depends on the previous answer. Such a procedure is displayed in a directed decision tree, or simply a tree. A decision or classification tree represents

a multi-stage decision process, whereby a decision is made at each node. Trees have either binary or multivalued decisions at each node. Binary decisions produce binary splits giving two descendent nodes, whereas, multivalued decisions produce multiple (greater than 2) descendent nodes. However, any tree can be represented using just binary decisions and thus, only binary splits are required. These trees are referred to as binary trees. The interested reader is referred to [21] for a particular mapping, oldest-child/next-sibling binary tree, which maps a multivalued tree onto an equivalent binary tree. Thus, without loss of generality, binary trees are used.

2.3.1.1 Tree Anatomy

The tree consists of *nodes* and *branches*, where by convention the first or *root* node is at the top of the tree. Nodes are linked to other nodes with branches. A node is either *internal* or *terminal*, where internal nodes have descendant or *child* nodes. Internal nodes branch to left and right child nodes, while terminal nodes have no descendant nodes. Each terminal node has an associated category and observations that end on a particular terminal node are assigned to that category.

2.3.1.2 Using a Tree as a Classifier

Starting from the root node, a particular object $x \in \Omega$ is classified in the following way. A binary decision 'true/false' is made with respect to a particular variable $s \in \mathbb{R}$. If the decision is true proceed to the left child node, otherwise proceed to the right child node. Continuing in this manner, eventually reaching a terminal node, the object x is given the category $\omega_i \in \mathcal{C}$ of that terminal node.

2.3.1.3 Unique Node Numbering

Here all descendant nodes are numbered with reference to their parent node. Specifically, if node(D) is internal then the left and right child nodes are numbered node($2D$) and node($2D + 1$), respectively. Hence, if a node number is even, the decision at the parent node was true. This unique numbering facilitates a backtracking strategy to determine the unique path from the root node to each terminal node, using only the terminal node number. This means the bounds on each sub-region of the partition can be determined in a straightforward manner. Details are left until Section 2.4.4.

2.3.2 Induced Partitions on Ω

Classification trees partition Ω into sub-regions using hyperplane decision boundaries, where each node in a classification tree represents a sub-region of Ω. The root node represents Ω itself and all descendent nodes satisfy the following requirements. If node(D) represents the sub-region A, with descendent nodes $2D$, $2D+1$ with sub-regions a_1, a_2 respectively, then $A = a_1 \cup a_2$ and $a_1 \cap a_2 = \emptyset$. Hence, the union of all sub-regions A_i defined by terminal nodes alone, where $i = 1, \ldots, |\text{terminal nodes}|$, partitions Ω into $|\text{terminal nodes}|$ non-empty sub-regions, with $\cup_i A_i = \Omega$.

Each tree method produces a different partition on Ω based on the form of the decision or query at each node. The reader is referred to Figure 2.1, where various partitions resulting from different queries on the same data set is shown. The notation x_j is used to denote the j^{th} coordinate of a point $x \in \Omega$ and $s \in \mathbb{R}$ is a scalar splitting value. The simplest approach to consider is binary classification trees, which use queries of the form: Is $x_j < s$? Such queries lead to hyper-rectangular sub-regions parallel to the coordinate axes. Binary Space Partition Trees (BSP) use queries of the form: Is $\lambda_1 x_1 + \lambda_2 x_2 + \ldots + \lambda_n x_n < s$? This partitions Ω into convex polyhedral sub-regions. Although there is more flexibility in how Ω is partitioned, evaluating many linear combinations can be computationally expensive in practice. Another approach, sphere trees, uses queries of the form: Is $\|x - z\| < s$? (where $z \in \Omega$ is chosen at each node). The resulting partition on Ω has sub-regions defined by pieces of spheres.

To construct an efficient adaptive partitioning random search algorithm, the partition must be computationally cheap to evaluate and it would be advantageous if the sub-regions were simple to draw samples from. The choice is obvious, binary classification trees, as they are computationally cheap and produce hyper-rectangular sub-regions. Thus, samples can be drawn directly from a uniform distribution over each sub-region. Other partitions would require an acceptance/rejection sampling method to draw samples from a particular sub-region.

Drawing samples from a uniform distribution over sub-regions using an acceptance/rejection sampling technique can be inefficient, even in relatively low dimensions in $\mathbb{R}^n$. Consider, for example, drawing samples from a hyper-spherical sub-region A of unit radius by applying acceptance/rejection sampling in a hypercube of equal radius. The probability of drawing a sample from A is obtained by comparing the measure of A and the cube, given by

$$Pr(x \in A) = \frac{1}{2^n} \frac{\pi^{n/2}}{\Gamma(n/2+1)}, \tag{2.7}$$

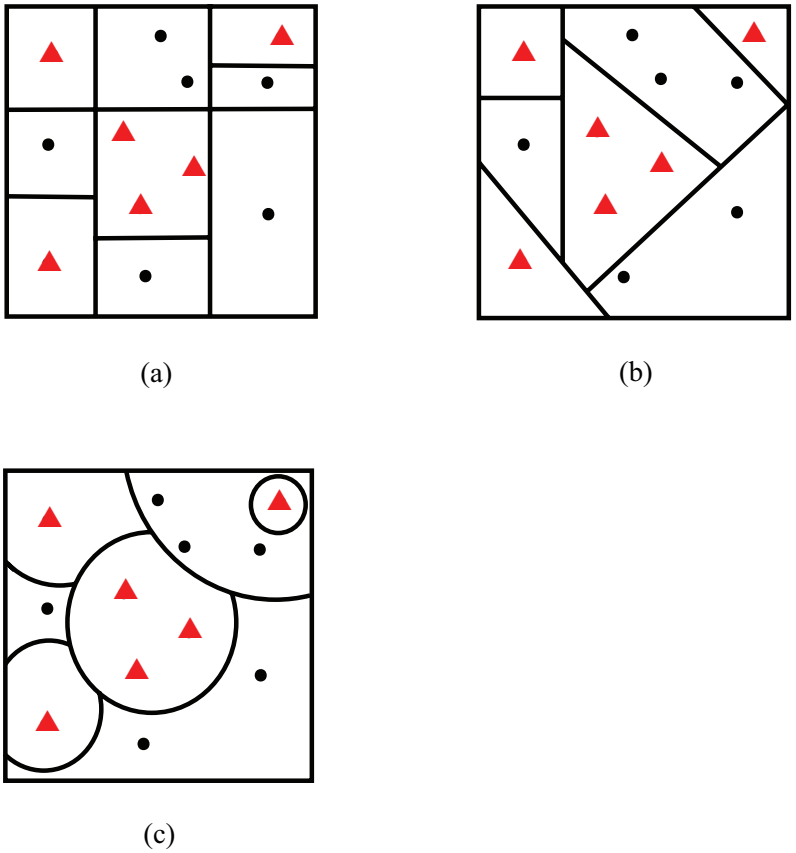


Figure 2.1: Partitions induced from a binary tree (a), BSP tree (b) and sphere tree (c) on data of two categories, denoted • and ▲.

where Γ is the Gamma function. It is clear from (2.7) that as dimension n increases, the probability of generating an $x \in A$ decreases rather quickly. Evaluating (2.7) for $n = 2, 4, 10$ the probabilities $0.8, 0.3, 0.002$ are obtained, respectively. Clearly, the acceptance/rejection sampling technique becomes increasingly inefficient and is largely impractical even at dimension 10. Hence, the hyper-rectangular structure obtained from binary classification trees is desirable here as samples can be drawn directly from each sub-region. One sampling strategy is described in Section 3.5 of the next chapter.

2.4 CART

The practical question of how to build a binary classification tree using a training data set is now considered. This section uses a training data set T of $N > 0$ sample points x, defined by

$$T = \{x^{(i)} \in \Omega : i = 1, \ldots, N\}.$$

T is assumed to be the union of two classified non-empty sets $\{\omega_L\}$ and $\{\omega_H\}$. The notation $x_j^{(i)}$ is used to denote the j^{th} coordinate of the i^{th} sample point.

In principle, many different binary classification trees can be constructed from a training data set however, some efficient algorithms exist. These algorithms proceed in a greedy manner, whereby a series of locally optimal decisions are made. Such algorithms include ID3, C4.5, and classification and regression trees (CART) [8, 24]. The latter is considered here. CART provides a general framework that can be implemented in many ways to produce different classification trees. A particular strategy designed for partitioning optimization spaces is considered here. In this approach, four general questions arise:

1. Where are potential splits in the training data?
2. Which potential split should be used to split a node?
3. When should a node be declared a terminal node?
4. How are the bounds of each sub-region obtained?

Each of these questions is considered in the subsections which follow.

2.4.1 Locating Potential Splitting Hyperplanes

The partition on Ω is formed using a set of hyperplanes, each orthogonal to a coordinate axis. There are up to $n(N-1)$ possible splitting hyperplanes (splits) in a training data set consisting of N distinct points. If the training data set consists of N randomly generated points, there are $n(N-1)$ possible splits with probability one because the probability of two points having the same coordinate value is zero. Often all the possible splits in T are considered potential in CART tree growing procedures [24]. However, as both N and n increase this method can become computationally expensive. All potential splits in the data occur between an element from $\{\omega_L\}$ and an element from $\{\omega_H\}$ — not between two elements from the same set. Thus, if $|\{\omega_H\}|, |\{\omega_L\}|$ are vastly different, considering all possible splits is computationally wasteful. Here we use a method which considers splits of the form $s = (x_j + z_j)/2$ such that $x \in \{\omega_L\}$ and $z \in \{\omega_H\}$, where x_j denotes the j^{th} coordinate of a point $x \in T$.

The method proposed here uses the Matlab functions 'sort' and 'find' [47]. For the reader who is unfamiliar with Matlab we provide some simple examples to aid the discussion: Let A be the vector $A = [7, 9, 6, 3]$ and let B be the vector of sorted

(ascending) elements. Using the sort function:

$$[B, I] = \text{sort}(A)$$

we obtain $B = [3, 6, 7, 9]$. The sort function also outputs an index vector $I = [4, 3, 1, 2]$. This index vector gives the position in A that each element of B holds. For example, the first element of B is the fourth element in A, that is,

$$3 = B(1) = A(I(1)) = A(4) = 3.$$

The Matlab notation for reordering elements of a vector is also used. For example, $A(I) = A([4, 3, 1, 2])$ reorders the vector A so that the fourth element becomes the first element, the third element becomes the second element and so on. Using the Matlab reordering notation we see that, $B = A(I)$.

The other Matlab function we use is the 'find' function. Let $A = [1, 2, 3, 2, 3]$, for example, and let's say we want to know the position of all the 3's in A. The command:

$$I = \text{find}(A == 3)$$

will return the index vector $I = [3, 5]$ — the third and fifth elements of A are 3's. Using the Matlab functions and notation our method for locating potential splits can be described.

Consider locating all potential splits in the j^{th} dimension. The notation $\mathbf{1}_\lambda$ ($\mathbf{2}_\lambda$) is used to denote a vector of ones (twos) with length λ. Let T_j denote the set consisting of the j^{th} coordinates of each element from the set $T = \{x \in \omega_L, z \in \omega_H\}$ and let $T_j^{\mathcal{O}}$ be the ordered set (ascending) of T_j with index vector $\mathcal{I}_T$ such that $T_j^{\mathcal{O}} = T_j(\mathcal{I}_T)$. Setting three vectors

$$\mathcal{I}_1 = [\mathbf{1}_{|\omega_L|}, \mathbf{2}_{|\omega_H|}], \quad \mathcal{I}_2 = [\mathcal{I}_1(\mathcal{I}_T), 0] \quad \text{and} \quad \mathcal{I}_3 = [0, \mathcal{I}_1(\mathcal{I}_T)],$$

the position of each potential split in $T_j^{\mathcal{O}}$ is related to the position of the 3's in the vector sum $\mathcal{I}_2 + \mathcal{I}_3$. Using the 'find' command

$$I = \text{find}(\mathcal{I}_2 + \mathcal{I}_3 == 3)$$

each potential split is located at minimal cost using the set

$$\mathcal{I}_s = \{I(i) : 1 \leq i \leq \dim I\},$$

where $\dim I$ is the dimension of I. Specifically, for each $i \in \mathcal{I}_s$ a potential split occurs at

$$s = (T_j^{\mathcal{O}}(i) + T_j^{\mathcal{O}}(i-1))/2. \tag{2.8}$$

Applying the above procedure for all n dimensions of f, potential splits in the training data are located at minimal cost. This procedure is illustrated in the next subsection.

2.4.1.1 Locating Potential Splits Example

To illustrate the above procedure we consider locating potential splits in one dimension using the following training data set:

$$\omega_L = \{1, 9, 11, 15, 17\} \text{ and } \omega_H = \{3, 5, 7, 13, 19\}.$$

Combining the classified sets ω_L and ω_H we obtain

$$T_1 = \{\omega_L, \omega_H\} = [1, 9, 11, 15, 17, 3, 5, 7, 13, 19]$$

and sorting into ascending order gives

$$T_1^{\mathcal{O}} = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]$$

with index vector

$$\mathcal{I}_T = [1, 6, 7, 8, 2, 3, 9, 4, 5, 10].$$

Setting the three vectors $\mathcal{I}_1$, $\mathcal{I}_2$ and $\mathcal{I}_3$ we obtain

$$\mathcal{I}_1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2];$$

$$\mathcal{I}_2 = [1, 2, 2, 2, 1, 1, 2, 1, 1, 2, 0]; \text{ and}$$

$$\mathcal{I}_3 = [0, 1, 2, 2, 2, 1, 1, 2, 1, 1, 2].$$

All potential splits in $T_1^{\mathcal{O}}$ are related to the position the 3's in the vector sum $\mathcal{I}_2 + \mathcal{I}_3$, given by the set $\mathcal{I}_s = \{2, 5, 7, 8, 10\}$. Therefore, there are five potential splits in the

training data. Using (2.8) we see a split occurs at

$$s = \frac{T_1^{\mathcal{O}}(7) + T_1^{\mathcal{O}}(6)}{2} = \frac{13 + 11}{2} = 12.$$

Furthermore, we see that $T_1^{\mathcal{O}}(6) \in \omega_L$ and $T_1^{\mathcal{O}}(7) \in \omega_H$ — different categories — as required.

2.4.2 Choosing a Potential Split and Node Impurity

When growing a classification tree the fundamental principle is that of simplicity. Specifically, decisions that lead to a compact tree with few nodes are preferred. Thus, at each node(D) the split that makes descendant nodes as pure as possible is sought. By convention a node's impurity is measured rather than how pure it is. There are various measures of node impurities, all of which satisfy the following requirements. Let $i(D)$ denote the impurity at node(D), then $i(D)$ must be zero when node(D) is pure and a maximum when the categories are equally represented. Impurity measures include Gini, Classification Error and Entropy [24]. Here the most popular measure, entropy measure, is used

$$i(D) = -P(\omega_L)\log_2 P(\omega_L) - P(\omega_H)\log_2 P(\omega_H), \tag{2.9}$$

where $P(\omega_L)$ is the fraction of points at node(D) which are elements of the set $\{\omega_L\}$ and similarly for $P(\omega_H)$. Here $0\log_2(0) = 0$ in entropy calculations.

Given a partial tree down to node(D), the question now arises: which split is optimal? There are up to $n(N-1)$ potential splits to consider. Each query is of the form "Is $x_j^{(i)} \leq s$?", where $s \in \mathbb{R}$ is the scalar splitting value given by (2.8). Using a greedy strategy, the split that decreases the impurity as much as possible is chosen. The drop in impurity is simply,

$$\Delta i(D) = i(D) - P_L i(D_L) - (1 - P_L) i(D_R), \tag{2.10}$$

where D_L and D_R are left and right child nodes, $i(D_L)$ and $i(D_R)$ their impurities and P_L is the fraction of points at node(D) that will go to D_L after the split. By means of exhaustive search over all potential splits, the optimal split which maximizes (2.10) is found. Sometimes there are several optimal splits which yield the same $\Delta i(D)$, in which case the first instance is usually chosen, as is done here. Although each split is locally optimal, the fully grown tree is not necessarily optimal, i.e. a series of locally

optimal decisions does not imply global optimality. To obtain a more *optimal* tree pruning methods can be employed. The interested reader is referred to, for example, [24] for further details on pruning. Relatively small data sets are used here (small trees) and pruning is considered no further.

There are two general splits that can be considered in tree growing algorithms. Firstly, the *forced split*, where a split in the j^{th} dimension is required. The j^{th} dimension split which maximizes (2.10) is chosen as the optimal split. Secondly, the *free split*, where a split from any dimension is chosen to maximize (2.10). Here only free splits are used.

2.4.3 When to Stop Node Splitting

The problem of when to stop node splitting is now considered. One strategy is to continue splitting nodes until all terminal nodes are pure. However, such a strategy can lead to large complicated trees with many nodes and hence, a complicated partition on Ω. Another method is to stop splitting when a predefined maximum number of nodes is reached. Here splitting continues until an *optimal partition* is achieved, defined formally in the following definition.

Definition 14. (Optimal partition). *Let* $|\{\omega_L(D)\}|, |\{\omega_H(D)\}|$ *denote the number of low/high points at node(*D*) and let* $0 < \tau_i < 1$ *be an impurity tolerance. Then an optimal partition is achieved if each terminal node(*D*) in the current tree satisfies one of the following conditions:*

- $i(D) = 0$*, the node is pure; or*
- $|\{\omega_L(D)\}| > |\{\omega_H(D)\}|$ **and** $i(D) < \tau_i$*.*

Definition 14 allows some terminal nodes to be impure by misclassifying high points. This onesided misclassification is used to potentially simplify the partition on Ω. Here function minimization is of primary interest and hence, if a few *rogue* high points over complicate the model they could be ignored. Misclassifying low points can simplify the partition but may be problematic for optimization purposes. Consider, for example, an $x \in \{\omega_L\}$ such that $\|x - x_*\| < \epsilon$, where ϵ is a sufficiently small positive constant and $x_* \in \Omega$ a global minimizer. Misclassifying x could result in the classification model asserting points sufficiently close to the solution have relatively high function values, even though a point with an exceedingly low function value has been sampled there.

A tolerance value $\tau_i = 0.45$ is used herein. Thus, a terminal node with $|\{w_L\}| \geq 10$ and $|\{w_H\}| = 1$ would be split no further and the corresponding low sub-region on Ω would contain one point with a relatively high function value.

2.4.4 Defining CART Sub-Regions

Each terminal node in the classification tree corresponds to a sub-region of the partition on Ω. As mentioned earlier, each node has a unique node number associated with it (see Section 2.3.1.3). Such numbering facilitates a backtracking procedure to obtain the bounds on each sub-region of the CART partition by retracing the unique path from each terminal node to the root node.

The unique path vector $\mathfrak{P}$ to a terminal node(D), is calculated as follows. Set $\mathfrak{P}(1) = D$. Calculate each internal node number on the path sequentially using $\mathfrak{P}(j) = \lfloor \mathfrak{P}(j-1)/2 \rfloor$ for integer $j > 1$, until the root node is found, $\lfloor \mathfrak{P}(j) \rfloor = 1$. Here $\lfloor \lambda \rfloor$ is the largest integer not exceeding the real number λ. Sorting $\mathfrak{P}$ into ascending order gives the unique path from the root node to terminal node(D).

A matrix B is used to store the bounds on each sub-region A_i. The notation B_i is used to denote the i^{th} row of the matrix B. The size of B is |terminal nodes| by $2n$ and each row has the following structure,

$$B_i = [b_1, \ldots, b_n, b_{n+1}, \ldots, b_{2n}].$$

Here $b_j, b_{j+n} \in \{B_i(q) : 1 \leq q \leq 2n\}$, where $1 \leq j \leq n$, denote lower and upper bounds for sub-region A_i in dimension j, respectively. Initially each row of B contains the bounds of the optimization region Ω,

$$B = \begin{bmatrix} \ell_1 & \ell_2 & \ldots & \ell_n & u_1 & u_2 & \ldots & u_n \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ \ell_1 & \ell_2 & \ldots & \ell_n & u_1 & u_2 & \ldots & u_n \end{bmatrix}.$$

Each row B_i is updated iteratively using queries along the unique path vector $\mathfrak{P}_i$ to the terminal node corresponding to sub-region A_i. Each query at an internal node(D) is of the form: Is $x_j < s$. Thus, if node($2D$) $\in \mathfrak{P}_i$ (left child node) the query at node(D) was true. That is, s is an upper bound for the j^{th} dimension of A_i and so $B_i(n+j) = s$. Otherwise node($2D+1$) $\in \mathfrak{P}_i$, in which case $B_i(j) = s$, a new lower bound. Elements of B_i are updated until the terminal node is reached. In this approach an element of B_i may be updated more than once. This procedure is demonstrated in Section 2.4.6.

2.4.5 CART Algorithm

The specific CART tree growing algorithm using a training data set T is now presented. A precise statement of the algorithm is given in Figure 2.2. The notation $T(D)$ is used to denote the elements of T that satisfy each query on the unique path to node(D).

The algorithm contains two nested loops. The outer loop (steps 2 - 5) grows the tree by considering splitting each terminal node of the current tree. The inner loop (Step 2) searches for an optimal split which maximizes (2.10) at the current terminal node(D). Step 4 splits the training data $T(D)$ using the optimal split found at Step 3, creating two new terminal nodes in the current tree. The tree is fully grown when an optimal partition is found (see Definition 14) and no further splitting is performed. Step 6 defines the lower and upper bounds on each hyper-rectangular sub-region of the partition on Ω, as described in the previous section.

1. Initialize: Set misclassification tolerance $\tau_i > 0$. Define training data at root node $T(1) = \{\omega_L, \omega_H\} = T$.

2. If a terminal node(D) in the current tree fails the optimal partition requirements, then search for the best split in data $T(D)$. **For** $j = 1, 2, \ldots, n$ **do**

   (a) Determine all potential splits in the j^{th} dimension using (2.8).

   (b) Choose the split that maximizes $\Delta i(D)$. Increment j and goto 2(a).

3. Out of the n best splits in Step 2, split node(D) on the one that maximizes $\Delta i(D)$ over all n dimensions.

4. For the split found in Step 3, determine $T(2D)$ and $T(2D + 1)$, points that go to the left and right child nodes.

5. Repeat steps 2 through 4 until an optimal partition is obtained and then goto Step 6.

6. Define bounds on each hyper-rectangular sub-region of the partition on Ω and store in matrix B.

Figure 2.2: CART tree growing algorithm

2.4.6 Classification Tree and Partition on Ω

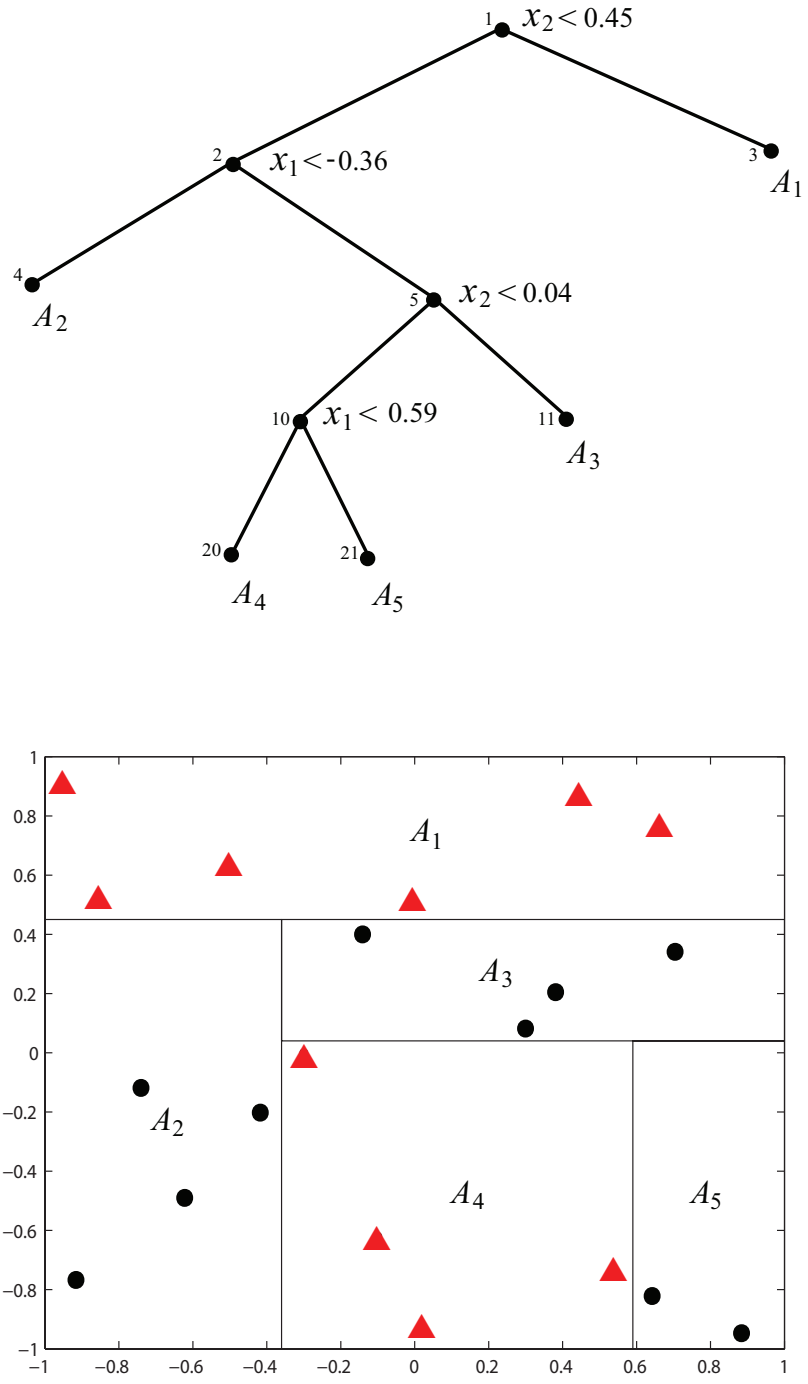


Figure 2.3: Classification tree grown by CART procedure and the resulting partition on Ω. Here $x \in \{w_L\}$ and $z \in \{w_H\}$ are denoted • and ▲, respectively.

To conclude this section, a classification tree is grown using the CART algorithm given in Figure 2.2. The training data set T was obtained by drawing 20 points from a uniform distribution over $\Omega = [-1, 1]^2$ and applying a two category classification as follows;

$$\omega_L = \left\{ \begin{array}{cccccccccc} -0.62 & -0.74 & -0.92 & 0.30 & -0.14 & -0.42 & 0.64 & 0.70 & 0.38 & 0.88 \\ -0.48 & -0.12 & -0.76 & 0.08 & 0.40 & -0.20 & -0.82 & 0.34 & 0.21 & -0.95 \end{array} \right\}$$

and

$$\omega_H = \left\{ \begin{array}{cccccccccc} -0.50 & -0.86 & -0.96 & -0.01 & 0.54 & 0.02 & -0.30 & -0.10 & 0.44 & 0.66 \\ 0.62 & 0.52 & 0.90 & 0.50 & -0.74 & -0.94 & 0.00 & -0.62 & 0.86 & 0.76 \end{array} \right\}.$$

Figure (2.3) shows the fully grown classification tree and the corresponding partition on Ω. Each hyperplane decision boundary corresponds to a particular query at an internal node of the tree. Four hyperplanes, each orthogonal to one coordinate axis, have partitioned Ω into five hyper-rectangular sub-regions A_i, where $i = 1, 2, \ldots, 5$. The set $\{x \in A_2 \cup A_3 \cup A_5\}$ is classified as ω_L and the set $\{x \in A_1 \cup A_4\}$ is classified as ω_H.

Using the backtracking procedure, the bound calculations on sub-region A_5 (terminal node(21)) is demonstrated. The unique path to node(21) is given by $\mathfrak{P} = [1, 2, 5, 10, 21]$ and so the queries at node(1) and node(5) are upper bounds, and the queries at node(2) and node(10) are lower bounds. Thus,

$$B_5 = [0.59, -1.00, 1.00, 0.04],$$

where $0.59 < x_1 < 1.00$ and $-1.00 < x_2 < 0.04$ for sub-region A_5.

2.5 Global Optimization Algorithmic Framework

This section provides a general framework for a class of Adaptive Partitioning Random Search global optimization algorithms (APRS). This framework is quite flexible allowing for a variety of algorithms to be constructed with only mild conditions to be satisfied, details of which follow. A specific algorithm conforming to this framework is given in the next chapter.

A general statement of APRS is given in Figure 2.4. The method consists of an initialization step and then alternates between two phases: a partition phase and a

1. Initialize: Choose $N > 0$, $T_{\max} > 0$ and $1/N \leq \chi \leq (N-1)/N$. Generate a batch of N points $x \in \Omega$, X_1, and set x_1 as the point with the lowest function value in X_1. Set $k = 1$ and $T_1 = X_1$.

2. Classification: Classify a fraction of elements from T_k with the least function values as low, ω_L, and the rest of the points in T_k as high, ω_H, such that $\{\omega_L\}, \{\omega_H\} \neq \emptyset$ and $x_k \in \{\omega_L\}$.

3. Partition phase: Form a partition on Ω using the classified training data set $T_k = \{\omega_L, \omega_H\}$ and a statistical classification method which is a finite process.

4. Sampling phase: Generate $\lceil (1-\chi)N \rceil$ points uniformly from the low sub-regions of the partition and $\lceil \chi N \rceil$ points uniformly from the high sub-regions using a finite procedure sampling method. Call the new batch of points X_k.

5. Update T: Set $T_{k+1} = T_k \cup X_k$ and $x_{k+1} = \arg\min\{f(x) : x \in T_{k+1}\}$. If $|T_{k+1}| > T_{\max}$ discard a fraction of points with the largest function values. If stopping conditions are satisfied stop, otherwise increment k and go to Step 2.

Figure 2.4: Adaptive Partitioning Random Search algorithm

sampling phase. The user chooses the batch size $N > 0$, an upper bound on the training data set $T_{\max} > 0$ and χ (the parameter determining the fraction of samples to be drawn from the high sub-regions). The initialization phase is completed by drawing a batch of N points, $X_1 \in \Omega$, and evaluating f at each point. This initial batch of points gives the initial unclassified training data set T_1. The point with the lowest function value is set as the initial iterate x_1.

Step 2 classifies the training data set T_k into two categories: points with relatively low function values $\{\omega_L\}$ and points with relatively high function values $\{\omega_H\}$, where both $\{\omega_L\}$ and $\{\omega_H\}$ are non-empty and $x_k \in \{\omega_L\}$. The training data set is re-classified at each iteration by choosing $\{\omega_L\}$ as the fraction of elements from T with the least function values. The remaining elements of T_k are classified as high points, $\{\omega_H\} = T_k \setminus \{\omega_L\}$. It is advantageous to promote clustering in $\{\omega_L\}$ to allow successive partitions to focus down on low sub-regions. Choosing $|\{\omega_H\}| > |\{\omega_L\}|$, for example,

can cause $\{\omega_L\}$ to cluster, potentially in subsets of Ω where f is low. Promoting clustering in $\{\omega_L\}$ is investigated further in the next chapter. More categories can be included in the classification and the algorithm is updated in the obvious way.

The partition phase at Step 3 forms a partition on Ω using a finite procedure statistical pattern recognition technique. For example, the CART technique described in the previous sections could be used although other choices are possible. The only requirement is that the union of all sub-regions is Ω itself so sampling both the low and high sub-regions is equivalent to sampling Ω. This property is crucial for establishing convergence to a global minimizer of f.

The sampling phase at Step 4 draws samples uniformly from either the low or the high sub-regions of the partition. The sampling method depends explicitly on which partition method is chosen in Step 3 and may require an acceptance/rejection sampling method to draw samples uniformly from particular sub-regions. Any finite procedure sampling method that draws samples uniformly from the low and high sub-regions of the partition is sufficient. If the CART partition is used, sampling sub-regions uniformly is straightforward due to the hyper-rectangular structure of the partition, see for example, Section 3.5 for a sample point delivery method.

The user chooses a fraction χ of sample points to be drawn from the high sub-regions of the partition, where

$$\frac{1}{N} \leq \chi \leq \frac{N-1}{N}. \tag{2.11}$$

Choosing $\chi < 0.5$ concentrates numerical effort where f is presumed to be relatively low based on the partition, potentially increasing the rate of convergence. With χ satisfying (2.11), at least one element is drawn from a uniform distribution over each of the two classified subsets of the partition at each iteration. Thus at each iteration, there exists a non-zero probability of sampling any subset of Ω with positive measure, a property crucial for convergence.

Step 5 updates the training data using the new batch of points X_k and existing sample points in T_k. All sample points are retained until a maximum training data set size of $T_{\max}$ is reached. Once reached, a fraction of points with the largest function values are discarded to ensure T remains bounded.

The treatment of stopping rules is deferred to Chapter 4, but when a rule is satisfied the estimate for a global minimizer is

$$x_* = \arg\min\{f(x) : x \in T_k\}.$$

2.5.1 Analogs with Pure Adaptive Search

There are similarities between APRS and PAS [88] (see Section 2.1). To implement PAS level sets of f must be known in advance and each level set requires a point to be drawn from a uniform distribution over it. Using the partition obtained from the chosen classification technique, an approximation to a level set can be obtained, defined by the set

$$A = \{x \in \Omega : \mathcal{T}(x) = \omega_L\}.$$

That is, A is the set of sub-regions classified as low in the partition. Furthermore, if the CART partition is used, A is a union of hyper-rectangles which is simple to sample. Thus, although APRS with the CART partition cannot guarantee each sample point is drawn from successively lower level sets, level sets can be approximated and samples can be drawn from them. At each iteration the level set approximation is updated, potentially defining a lower approximate level set from which samples can be drawn uniformly. Thus, one may consider APRS with the CART partition as a practical variant of the PAS algorithm.

2.5.2 Convergence

The convergence properties of APRS are analyzed with the stopping conditions removed. This allows us to examine the asymptotic properties of the sequence of iterates generated by the algorithm. The convergence result shows that every cluster point of the sequence $\{x_k\}$ is an essential global minimizer of f with probability one.

Definition 15. (Essential global minimizer). *An essential global minimizer x_* is a point for which the set*

$$L(x_*) = \{x \in \Omega : f(x) < f(x_*)\} \tag{2.12}$$

has Lebesgue measure zero.

If the objective function is continuous, then $f(x_*)$ is also a global minimum in the classical sense.

Assumption 16. *Let the following conditions hold:*

(a) The sequence of function values $\{f(x_k)\}$ is bounded below,

(b) The objective function f is lower semi-continuous.

The first condition of Assumption 16 ensures $f(x_k) \not\to -\infty$ as $k \to \infty$. The second condition precludes the existence of a sequence $\{x_k\}$ converging to a point x_* for which $f(x_*) > f(x)$ for all $\|x - x_*\| \leq \epsilon$, where $\epsilon > 0$ and sufficiently small.

Theorem 17. *The sequence of iterates $\{x_k\}$ generated by the APRS algorithm is an infinite sequence.*

Proof. For $\{x_k\}$ to be an infinite sequence the main loop of the APRS algorithm (steps 2—5) must be a finite process. The cardinality of the training data set T_k is bounded above by $T_{\max}$ and so steps 2 and 5 are finite processes. The partition phase uses a finite training data set and the classification method is a finite process so Step 3 is a finite process. Step 4 uses a finite process sampling method to draw N points from Ω and so Step 4 is a finite process. □

Theorem 18. *Let Assumption 16 hold. Each cluster point x_* of the sequence $\{x_k\}$ generated by APRS is an essential global minimizer of f with probability one.*

Proof. Theorem 17 and the fact that Ω is bounded ensure the existence of cluster points in $\{x_k\}$.

The proof is by contradiction. Assume x_* is not an essential global minimizer of f. Then there exists a subset $L(x_*) = \{z \in \Omega : f(z) < f(x_*)\}$ with positive Lebesgue measure. The probability that L is sampled from a single draw at iteration k is bounded below by:

$$\begin{aligned}
Pr(z \in L) &= Pr(z \in L \cap \omega_L) + Pr(z \in L \cap \omega_H) \\
&= Pr(z \in \omega_L) Pr(z \in L | z \in \omega_L) + Pr(z \in \omega_H) Pr(z \in L | z \in \omega_H) \\
&= (1 - \chi) \frac{m(L \cap \omega_L)}{m(\omega_L)} + \chi \frac{m(L \cap \omega_H)}{m(\omega_H)} \\
&\geq (1 - \chi) \frac{m(L \cap \omega_L)}{m(\Omega)} + \chi \frac{m(L \cap \omega_H)}{m(\Omega)} \\
&\geq \frac{1}{N} \frac{m(L)}{m(\Omega)} \\
&> 0,
\end{aligned}$$

where $1/N$ is the lowest value χ, or $1 - \chi$ can take. At iteration k, the probability that at least one sample is drawn from L after k batches is bounded below by

$$1 - \left(1 - \frac{1}{N} \frac{m(L)}{m(\Omega)}\right)^{kN}. \tag{2.13}$$

Since there is an infinite number of iterations, as k tends to infinity, (2.13) tends to one and L is sampled almost surely. Hence, in the limit as $k \to \infty$, $f(x_k) < f(x_*)$ almost surely, contradicting Assumption 16(b). Thus, x_* must be an essential global minimizer of f, almost surely.

□

Although this convergence result is strong, it is similar to the Pure Random Search result in the sense that as $k \to \infty$ every subset of Ω with positive measure is sampled with probability one. Therefore, one may consider such a method computationally expensive. However, it is the author's opinion that APRS would be more efficient in practice when most of the samples are drawn from low sub-regions ($\chi > 0.5$ say). This claim is empirically backed up for a particular APRS algorithm (presented in the next chapter) in the numerical results chapter of this thesis.

Chapter 3

CARTopt: A Random Search Nonsmooth Optimization Method

This chapter introduces a new method called CARTopt to solve the nonsmooth minimization problem given by (1.1), where the objective function $f : \mathcal{S} \to \mathbb{R} \cup \{+\infty\}$. Here $\mathcal{S}$ is called the optimization region and both bound constrained and unconstrained regions are considered. With $\mathcal{S} = \mathbb{R}^n$, a method for unconstrained minimization is presented. The bound constrained version searches in an n-dimensional box of the form

$$\mathcal{S} = \{x \in \mathbb{R}^n : \ell_i \leq x \leq u_i \text{ for all } i = 1, \ldots, n\},$$

where $\ell_i < u_i$ are finite. Under appropriate scaling, the constrained optimization region can be modified to $\mathcal{S} = [-1, 1]^n$. Therefore, without loss of generality, the bound constrained method is described with reference to $\mathcal{S} = [-1, 1]^n$.

The methods conform to the framework described in Section 1.7, whereby, a series of local and localized global phases are conducted. The local phase of the algorithm forms a partition on $\mathcal{S}$, using the CART method described in the previous chapter. The localized global phase performs Pure Random Search (PRS) on a subset of $\mathcal{S}$ defined by the partition.

Firstly, the bound constrained version of the CARTopt method is introduced. Section 3.2 defines the training data set which is used to form a partition on $\mathcal{S}$. An invertible training data transformation is proposed in Section 3.3, which potentially simplifies the partition on $\mathcal{S}$. The partition itself is introduced in Section 3.4. A sampling method is presented in Section 3.5, which draws samples from various parts of the partition. Section 3.6 introduces a deterministic instance of CARTopt using the Halton sequence [31]. The algorithm is then generalized to an unconstrained method

in Section 3.7. This chapter concludes with a convergence analysis for all stochastic instances of the CARTopt method, where the objective function is assumed to be nonsmooth or discontinuous.

3.1 Bound Constrained CARTopt

In this section the bound constrained CARTopt algorithm is presented. The unconstrained instance proposed later differs slightly. However, as most of what follows applies to both methods the term bound constrained is dropped. Section 3.7 makes it explicitly clear where the methods differ.

CARTopt is a random search method that uses classification methods to form a partition on $\mathcal{S}$. As the name suggests, Classification and Regression Trees (CART) [24] is the method used to form the partition. Recall from the previous chapter, that such a partition divides $\mathcal{S}$ into a set of non-empty hyperrectangular sub-regions. Furthermore, sub-regions of the partition can be classified based on observed function values contained in each. In particular, each sub-region can be classified as either high or low. Using the CART partition, an approximate level set $\mathcal{L}$ is obtained, defined formally below.

Definition 19. (Approximate level set). *Let A_i denote the i^{th} low sub-region of the CART partition on $\mathcal{S}$, where $1 \leq i \leq |$low sub-regions$|$, then an approximate level set is*

$$\mathcal{L} = \{x \in \bigcup_i A_i\}. \tag{3.1}$$

The approximate level set defines a subset of $\mathcal{S}$ where f is relatively low, based on where f has been sampled. An extremely useful property of $\mathcal{L}$ is that it consists of a union of hyperrectangles. Thus, it is simple to draw further samples from $\mathcal{L}$. Therefore, alternating between partition and sampling phases in an algorithmic manner provides a method for sampling subsets of $\mathcal{S}$ which are relatively low, based on known function values. This is the basis for the new method.

A statement of CARTopt is given in Figure (3.1). The algorithm consists of an initialization phase (steps 1 and 2) and a single loop (Step 3). Step 1 sets the iteration counter $k = 1$, the batch size $N > 0$ and a minimum hyperrectangle radius $\delta > 0$. Choosing $\delta > 0$ ensures $m(\mathcal{L}) > 0$ for all iterations — a property needed for convergence on nonsmooth problems. In addition, an input training data set T_1 with at least one finite function value is chosen if the user has a priori knowledge of f. If no input

1. Initialize: Set $k = 1$. Choose $N > 0, \delta > 0$ and $T_1 \subset \mathcal{S}$.

2. Generate $\max\{2N - |T_1|, 0\}$ random sample points $x \in \mathcal{S}$, store in X_1, and evaluate $f(x)$ at each $x \in X_1$. If $f(x) = \infty$ for each x continue sampling until one finite value is obtained. Set $z_1 = \arg\min\{f(x) : x \in X_1\}$.

3. **while** stopping conditions are not satisfied **do**

   (a) Update the training data set: $T_{k+1} \subset \{X_k \cup T_k\}$ (see Section 3.2)

   (b) Classify T_{k+1}: Set $\{\omega_L\}$ as the $\min\{\lfloor 0.8N \rfloor, |\{x \in T_{k+1} : f(x) \neq \infty\}|\}$ elements of T_{k+1} with the least function values, and set $\{\omega_H\} = T_{k+1} \setminus \{\omega_L\}$.

   (c) Transform Training data set: Calculate the Householder matrix H_k and set

   $$\hat{T} = \frac{1}{\varphi} H_k T_{k+1}^{\mathsf{T}} \text{ where } \varphi = \max_{i=1,\ldots,n} \sum_{j=1}^{n} |H_{k(ij)}|.$$

   (d) Partition optimization region with CART using $\hat{T}$ to identify low sub-regions whose union is $\mathcal{L}_k$.

   (e) Localized Global Phase: Generate next batch of N random points X_{k+1} totally, or mainly in $\mathcal{L}_k$.

   (f) Apply inverse transform: Set $X_{k+1} = \varphi H_k X_{k+1}^{\mathsf{T}}$.

   (g) Evaluate $f(x)$ at each $x \in X_{k+1}$ and call the best point $\hat{x}$. If $f(\hat{x}) < f(z_k)$, set $z_{k+1} = \hat{x}$, otherwise $z_{k+1} = z_k$. Check stopping conditions and increment k.

   end

Figure 3.1: CARTopt algorithm

training data exists, T_1 is set as empty. Step 2 generates the first set of points X_1, drawn from a uniform distribution over $\mathcal{S}$ and corresponding objective function values. At least one element of X_1 has a finite function value. Each point is stored as a row vector of the matrix X_1.

The loop generates a sequence of iterates $\{z_k\}_{k=1}^{\infty} \subset \mathcal{S}$. Each iterate z_{k+1} is generated from its predecessor z_k by evaluating f at a finite set of points. Rather than simply evaluating f uniformly over $\mathcal{S}$, f is evaluated mainly or totally in the approximate level set $\mathcal{L}_k$, defined at Step 3(d) from the CART partition. An attempt to simplify the partition is made at Step 3(c), where a transformation is applied to the training data. Before f is evaluated at the new points generated in Step 3(e), an inverse transform is applied so f is evaluated in the original space. The loop is executed until the stopping conditions are satisfied, where the estimate of an essential local minimizer is z_{k+1}.

The steps of the loop are discussed in detail in the sections that follow. The stopping conditions are described in Chapter 4.

3.2 Training Data and Classification

The training data set T is used as a model from which CART forms a partition of $\mathcal{S}$. Here a dynamic training data set which reflects information about f obtained during the previous sampling phase is proposed. Of particular interest is locating subsets of $\mathcal{S}$ where f is relatively low. Sampling such sets further and updating the training data set allows a new partition to be formed, defining a new subset to be explored.

An initial classification must be placed on T before any partition can be formed. There is a great deal of freedom when imposing a classification on T. Here two categories $\mathcal{C} = \{\omega_L, \omega_H\}$, points with relatively low and relatively high function values respectively are chosen, defined formally below. The notation $\lfloor \lambda \rfloor$ denotes the greatest integer less than or equal to λ.

Definition 20. (Low points). *Given* $0 < \beta < 1$, *the* $\min\{\lfloor \beta N \rfloor, |\{x \in T : f(x) \neq \infty\}|\}$ *elements of* T *with the least function values are classified as low and form the set* $\{\omega_L\}$.

Definition 21. (High points). *The set of points* $T \setminus \{\omega_L\}$ *is classified as high and form the set* $\{\omega_H\}$.

If f is finite at all training data points, $\{\omega_L\}$ is simply the βN elements of T with the least function values. The best iterate, z_k, is always included in $\{\omega_L\}$. Furthermore,

each element of $\{\omega_L\}$ is finite and $\{\omega_L\} \neq \emptyset$ (see Figure 3.1 Step 2). Hereafter, $\beta = 0.8$ used although other choices are possible. Therefore, $|\{\omega_L\}| \leq 0.8N$ for all k, which has two important consequences. Firstly, the number of distinct low sub-regions is bounded by $0.8N$, even if $|T|$ becomes large. Secondly, $|\{\omega_L\}| < |\{\omega_H\}|$ for all k, which can cause $\{\omega_L\}$ to cluster as k increases.

Let us now consider how T is updated. The primary interest here is local (not global) optimization and so it is sufficient for T to reflect local information about f. Forming the k^{th} partition using all $(k+1)N$ points generated from all k sampling phases (global information), can complicate the partition and is computationally expensive. A sufficient number of high points near $\{\omega_L\}$ are required to define the approximate level set $\mathcal{L}$ effectively. Loosely speaking, at least $2n$ high points are required to define each low hyperrectangular sub-region, one for each face. Therefore, as dimension increases, CART requires a larger training data set to effectively partition $\mathcal{S}$. Fortunately the number of faces grows linearly with n, as does the maximum training data size used here. Numerical simulations performed by the author found a maximum size $|T| \leq \max\{2N, 2(n-1)N\}$, performed well in practice on a variety of problems. The interested reader is referred to Appendix A for details on these problems.

Definition 22. (Full size training data). *A training data set T such that,*

$$|T| = \max\{2N, 2(n-1)N\}, \tag{3.2}$$

is called a full size training data set.

The training data is updated iteratively using the batch of newly generated points X_k via $T_{k+1} = \{X_k, T_k\}$. The most recent sample points appear first in the update. This gives an indication of the *age* of points as T grows. When $T_{k+1} = \{X_k, T_k\}$ exceeds full size, the oldest points with relatively high function values are discarded. Specifically, $T_{k+1} = \{X_\gamma, X_R\}$, where $X_\gamma \subset \{X_k \cup T_k\}$ is the $\gamma > 0$ sample points with the least function values and $X_R \subset \{X_k \cup T_k\} \setminus X_\gamma$ are the most recent points. This update ensures T_{k+1} is full size. Hereafter $\gamma = 2N$ is chosen, which is the number of function values required for the stopping condition (see Chapter 4). Numerical simulations performed by the author support this choice of γ although other choices are possible.

Ideally, updating and classifying T causes $\{\omega_L\}$ to cluster in the neighborhood of an essential local minimizer x_*. Fixing $|\{\omega_L\}|$, keeping points with the least f values and the most recent sample points tries to achieve this. Numerical simulations performed

by the author found that although initially $\{\omega_L\}$ may be disjoint, $\{\omega_L\}$ tends to cluster into a hyper-elliptical cloud, eventually in the neighborhood of x_*.

3.3 Transforming Training Data Sets

When forming a partition on $\mathcal{S}$ the fundamental principle is that of simplicity. The CART partition performs best, when the components of T have an alignment with the coordinate axes. Since all potential splits are orthogonal to the coordinate axes, such an alignment can simplify the partition on $\mathcal{S}$. Multivariate splits, as in BSP (see Section 2.3.2), can also simplify the partition on $\mathcal{S}$. However, such splits are computationally expensive and it is difficult to draw samples directly from sub-regions of the partition (see Section 2.3.2). Another approach is preprocessing T before the partition is formed. However, choosing a transform to simplify the partition for a general problem is often difficult. The interested reader is referred to [24] for various preprocessing techniques.

Ideally, a transform $\mathfrak{D} : x \in T \to \hat{T}$ is desired such that, the partition induced from $\hat{T}$ is *simpler* than one using T. Here simpler means fewer splits in the training data and hence, less sub-regions in the partition. The CARTopt approach, alternates between partition and sampling phases. Therefore, the transform $\mathfrak{D}$ must be invertible so f can be evaluated at points generated in the transformed space.

Numerical simulations performed by the author found that although initially $\{\omega_L\}$ may be disconnected, $\{\omega_L\}$ often forms a hyper-elliptical cloud of points. Thus, transforming T so that principle axis of the hyper-elliptical cloud aligns with a coordinate axis can be advantageous. Such a transform can dramatically simplify the partition, particularly in the neighborhood of some essential local minimizers where hyper-elliptical contours are present. This is illustrated in Figure 3.2.

The principle axes of the hyper-elliptical cloud $\{\omega_L\}$ are obtained using Principle Components Analysis (PCA) [24]. Using $\omega_L^{(i)}$ to denote the i^{th} element of $\{\omega_L\}$, expressed as a row vector, the scatter matrix is defined by

$$\mathcal{M} = \sum_{i=1}^{|\omega_L|} [\omega_L^{(i)} - \bar{\omega}_L]^{\mathsf{T}} [\omega_L^{(i)} - \bar{\omega}_L],$$

where $\bar{\omega}_L$ is the sample mean,

$$\bar{\omega}_L = \frac{1}{|\omega_L|} \sum_{i=1}^{|\omega_L|} \omega_L^{(i)}.$$

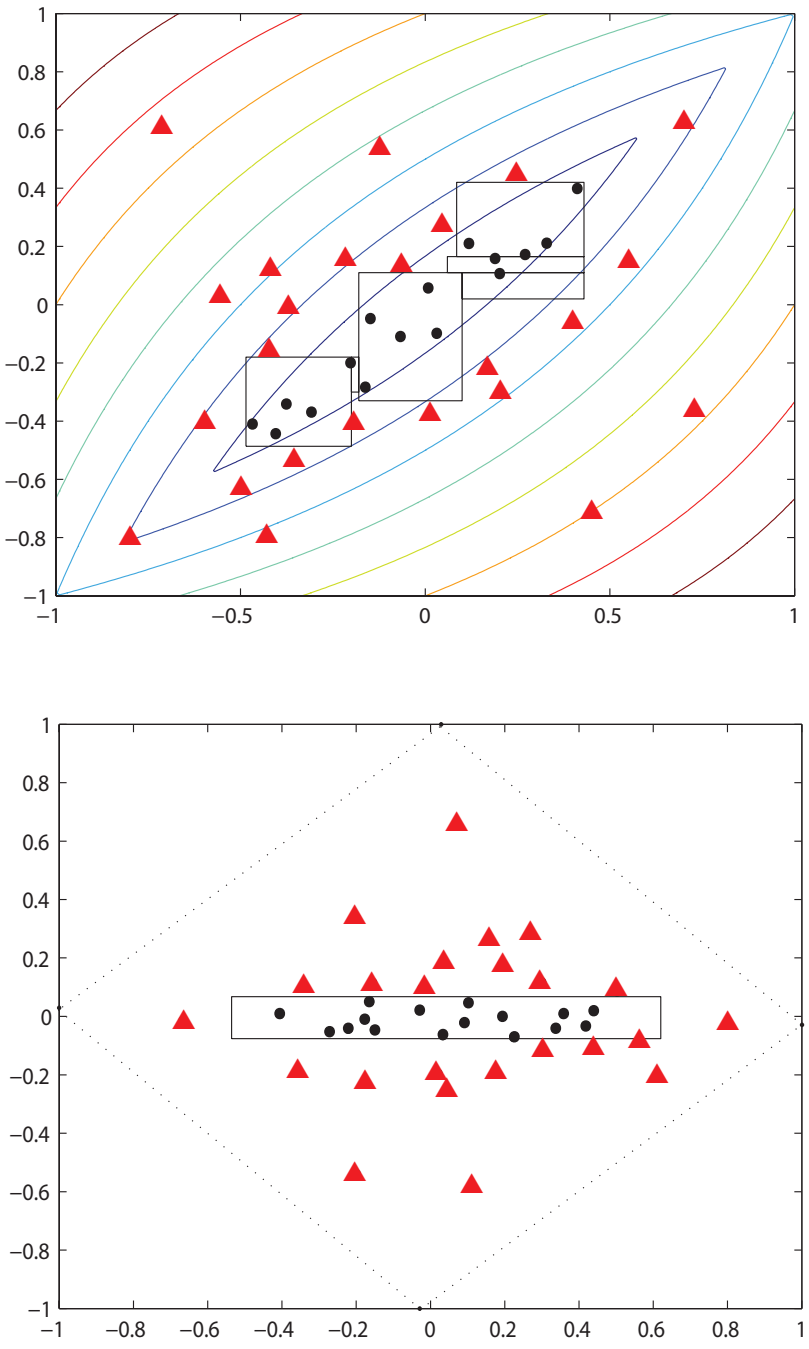


Figure 3.2: The first figure shows the level curves of $f = 2|x_1 - x_2| + x_1x_2$, along with the training data set $T = [\omega_L, \omega_H]$, with $x \in \{\omega_L\}$ and $z \in \{\omega_H\}$ denoted $\bullet$, $\blacktriangle$ respectively. The six low sub-regions formed in the CART partition are shown as black boxes. The second figure shows the transformed training data using (3.6). The transformed boundary of T is shown by the dotted lines. The transformation simplifies the partition to one sub-region, producing a good approximate level set $\mathcal{L}_k$.

The dominant eigenvector d, of the scatter matrix $\mathcal{M}$, is the direction vector for the principle axis of the hyper-elliptical cloud.

Using the Householder transformation,

$$H = I - 2uu^{\mathsf{T}},$$

$$u = (e_1 - d)/\|e_1 - d\|, \tag{3.3}$$

the x_1 axis is set parallel to d, i.e. $He_1 = d$. Pre-multiplying each element of T with H gives the desired transformed training data $\hat{T}$. This transformation is not only appealing because it keeps the coordinate directions orthogonal, but also the inverse transformation is trivial. Let, $\hat{x} = Hx$ be a point in the transformed space, then multiplying both sides by H,

$$(I - 2uu^{\mathsf{T}})\hat{x} = (I - 2uu^{\mathsf{T}})(I - 2uu^{\mathsf{T}})x, \tag{3.4}$$

where u is defined in (3.3). Simplifying the right hand side of (3.4),

$$\begin{aligned}(I - 2uu^{\mathsf{T}})\hat{x} &= \left(I - 4uu^{\mathsf{T}} + 4(uu^{\mathsf{T}})(uu^{\mathsf{T}})\right) x \\ &= \left(I - 4uu^{\mathsf{T}} + 4u(u^{\mathsf{T}}u)u^{\mathsf{T}}\right) x \\ &= x,\end{aligned}$$

where $u^{\mathsf{T}}u = 1$ (u is a unit vector). That is, $H\hat{x} = x$ and hence, f is evaluated at points generated in the transformed space at minimal cost.

However, pre-multiplying by H can reflect elements of T outside the $[-1, 1]$ optimization box. This is not problematic for the unconstrained instance of CARTopt, but for the bound constrained version it is convenient to operate in $[-1, 1]^n$. This allows post partition modifications to be made (see next section). Furthermore, always operating in $[-1, 1]^n$ makes it simple to ensure the search remains in the bound constrained region $\mathcal{S}$ after the transform.

To ensure all transformed points remain elements of $[-1, 1]^n$, each transformed point is multiplied by the scalar $1/\varphi$, defined as follows. The Householder matrix H is an elementary reflector which reflects points in the hyperplane given by (3.3). Noting that $\|Hx\|_2 = \|x\|_2$, the points reflected the greatest distance outside $\mathcal{S}$ (in the infinity norm sense) are elements of the vertex set

$$\mathbb{V} = \{z \in [-1, 1]^n : \|z\| = \sqrt{n}\}.$$

Therefore, multiplying by the reciprocal of

$$\max_z \|Hz\|_\infty, \tag{3.5}$$

where $z \in \mathbb{V}$, will give the desired scaling. Noting that $z_j \in \{-1, 1\}$, where $j = 1, \ldots, n$, solving (3.5) is simple. Specifically,

$$\varphi = \max_z \|Hz\|_\infty = \max_{i=1,\ldots,n} \sum_{j=1}^{n} |H_{ij}|,$$

the maximum absolute row sum of H. The elements of $\mathbb{V}$ which solve (3.5) are given by $z = \pm\text{sign}(H_i)$, where i is a row with maximum absolute row sum. The scalar φ takes values $1 \leq \varphi \leq \sqrt{n}$ and gives the minimal scaling such that $1/\varphi Hx \in [-1, 1]^n$ for all $x \in \mathcal{S}$. The reader is referred to Figure 3.2 for an example of the minimal scaling of the transformed space.

In summary, the transformed training data $\hat{T}$ is defined as

$$\mathfrak{D} = \frac{1}{\varphi} H T^\mathsf{T} = \hat{T}, \tag{3.6}$$

which is implemented in Step 3(c) of CARTopt. The inverse transform is simply

$$\mathfrak{D}^{-1} = \varphi \hat{T}^\mathsf{T} H = T, \tag{3.7}$$

which is implemented in Step 3(f) of CARTopt. Numerical simulations performed by the author found this transform to be extremely effective in practice, increasing the numerical performance of the algorithm significantly.

3.4 Partitioning the Optimization Region

CART partitions $\mathcal{S}$ into a set of non-empty hyperrectangular sub-regions A_i, such that $\cup_i A_i = \mathcal{S}$ and $A_i \cap A_j = \emptyset$. The interested reader is referred to Chapter 2 for a detailed description of the partition (see Section 2.4). CARTopt uses the same partition, but also performs post-partition modifications. Each modification is chosen to either increase the numerical performance of CARTopt, or ensure the algorithm is provably convergent on nonsmooth optimization problems. The modifications are simple to implement and keep the hyperrectangular structure of the partition. Furthermore, these modifications are not necessarily implemented at each iteration, only when required. Each is discussed

in the subsections which follow.

The matrix B, which contains the bounds of each sub-region A_i of the partition, is reintroduced here for convenience. Each row of B defines the lower and upper dimension bounds of A_i and has the following structure

$$B_i = [b_1, \ldots, b_n, b_{n+1}, \ldots, b_{2n}], \tag{3.8}$$

where $b_j \leq x_j \leq b_{j+n}$ for all $x \in A_i$ and $j = 1, \ldots, n$.

3.4.1 Defining Low Sub-Regions Only

Sampling subsets of $\mathcal{S}$ where f is known to be relatively low is of primary interest. Therefore, to increase computational efficiency low sub-regions are included in B and $\mathcal{S}$ is set as a single high region. In addition, drawing a point from a uniform distribution over $\mathcal{S}$ is cheaper than uniformly sampling the union of high sub-regions.

This simplification is reasonable because, typically, $m(\mathcal{L}_k) \ll m(\mathcal{S})$ for sufficiently large k. The probability of sampling $\mathcal{L}_k$ from drawing a sample from a uniform distribution over $\mathcal{S}$ is given by

$$Pr(x \in \mathcal{L}_k) = \frac{m(\mathcal{L}_k)}{m(\mathcal{S})}. \tag{3.9}$$

Hence, with $m(\mathcal{L}_k) \ll m(\mathcal{S})$, (3.9) is small and a high region is sampled uniformly simply by drawing an $x \in \mathcal{S}$. Furthermore, it would be efficient to apply acceptance/rejection sampling if required.

3.4.2 Minimum Sub-Region Radius

To exploit the connection between nonsmooth local optimization and global optimization (see Section 1.4.2), it is necessary for the $m(\mathcal{L}_k)$ to be bounded away from zero for all k. Therefore, pure random search is applied in a subset of $\mathcal{S}$ with positive measure for all iterations. To enforce this condition a minimum splitting distance between elements of $\{\omega_L\}$ and $\{\omega_H\}$ is imposed, post-partition.

Recall from Section 2.4.1, the partition on $\mathcal{S}$ is formed by considering a splitting hyperplane in each dimension j of the form, $s = (x_j + z_j)/2$, where $x \in \{\omega_L\}$ and $z \in \{\omega_H\}$. A minimum splitting distance $2\delta > 0$ between elements of $\{\omega_L\}$ and $\{\omega_H\}$ is imposed in CARTopt, after the CART partition phase is complete. Formally, for each low sub-region A_i, $\min \|x - z\| \geq \delta$ such that $x \in \{\omega_L \cap A_i\}$ and $z \in \mathcal{S} \setminus A_i$ is required. For sufficiently close splits, the splitting hyperplane is *pushed* away from the

closest $x \in \{\omega_L \cap A_i\}$. Specifically, each lower bound $b_j \in \{B_i(j) : 1 \leq j \leq n\}$ and each upper bound $b_{j+n} \in \{B_i(j+n) : 1 \leq j \leq n\}$ is set to

$$B_i(j) = \min\{b_j, \max\{-1, x_j^- - \delta\}\}, \tag{3.10}$$

$$B_i(j+n) = \max\{b_{j+n}, \min\{1, x_j^+ + \delta\}\}, \tag{3.11}$$

for each A_i, where $j = 1, \ldots, n$. Here the notation $x_j^+ = \max(x_j \in \{\omega_L \cap A_i\})$, and $x_j^- = \min(x_j \in \{\omega_L \cap A_i\})$, has been used. The constants -1 and 1 in (3.10) and (3.11) ensure each bound is not extended beyond the $[-1, 1]^n$ bound constrained CARTopt optimization region.

The update given by (3.10) and (3.11) has two important consequences. Firstly, it removes the possibility of samples converging to an impassable hyperplane boundary. Secondly, a minimum sub-region radius $\delta > 0$ on each A_i is forced. Therefore, $m(\mathcal{L}_k) \geq \delta^n$ for all k and so is bounded away from zero as required. The latter giving convergence on nonsmooth problems, see Section 3.8. Hereon in $\delta =$ 1e-10 is chosen.

Unfortunately, applying the minimum sub-region radius update can destroy the desirable property, $A_i \cap A_j = \emptyset$ for all $i \neq j$, inherent to the CART partition. To remove this problem the boundary of $\mathcal{L}$ could be extended, rather than individual sub-regions. However, this update is usually only applied when $m(\mathcal{L})$ is approaching the limiting size and has no adverse effects on the algorithm.

3.4.3 Updating Problematic Sub-Region Bounds

Numerical simulations performed by the author identified a weakness in the CART partition for optimization purposes. The CARTopt algorithm would be making great progress, consistently reducing f, then without warning, long periods of stagnation endured before f was reduced further. The problem stemmed directly from the training data set. Specifically, there did not exist an element(s) from $\{\omega_H\}$ between the cloud of points $\{\omega_L\}$ and some bound(s) of $\mathcal{S}$ in the partition. Therefore, a bound of $\mathcal{S}$ would also be a bound of some A_i. This can dramatically increase the measure of $\mathcal{L}_{k+1}$ from iteration k, forcing the algorithm to search away from the neighborhood of the known points with low function values. This is illustrated in 2-dimensions in Figure (3.3)

The partition considered here is always conducted in the $[-1, 1]^n$ box. Therefore, if $|b| \in \{B_i(q) : 1 \leq q \leq 2n\} = 1$ there is no $x \in \{\omega_H\}$ used to define the corresponding boundary of A_i. Such a bound is considered problematic, defined formally below.

Definition 23. (Problematic bound). *A sub-region bound* $b \in \{B_i(q) : 1 \leq q \leq 2n\}$

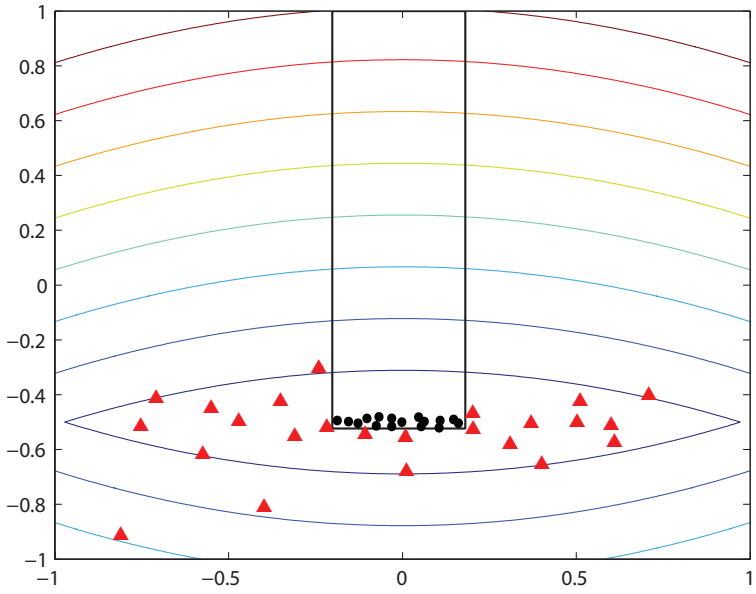


Figure 3.3: The level curves of $f = 5|x_2 + 0.5| + x_1^2$, along with the training data set $T = [\omega_L, \omega_H]$, with $x \in \{w_L\}$ and $z \in \{w_H\}$ denoted •, ▲ respectively, is shown. Note how the lack of an $x \in \{\omega_H\}$, between the elliptical cloud of points $\{\omega_L\}$ and the x_2 upper bound, has resulted in a poorly defined $\mathcal{L}$.

for a low sub-region A_i is considered a problematic bound if $|b| = 1$.

All problematic bounds warrant further investigation. Here two distinct low sub-regions that can be formed with the CART partition are considered.

Definition 24. (Singleton sub-region). *A low sub-region A_i is considered a singleton sub-region if $|\{x : x \in \omega_L \cap A_i\}| = 1$.*

Definition 25. (Non-singleton sub-region). *A low sub-region A_i is considered a non-singleton sub-region if $|\{x : x \in \omega_L \cap A_i\}| > 1$.*

Firstly, non-singleton low sub-regions with problematic bounds are considered and singleton low sub-regions are left until the next subsection.

To remove problematic bounds from non-singleton low sub-regions A_i the following method is used. The method is computationally cheap to evaluate and maintains the hyperrectangular structure of $\mathcal{L}$. Each problematic bound is replaced by a new bound which fits the training data best. The reader is referred to Figure 3.4 to accompany the description.

Consider a low sub-region A_i with at least one problematic bound. Each problematic bound corresponds to a face on the hyperrectangle which defines A_i. Initially, each

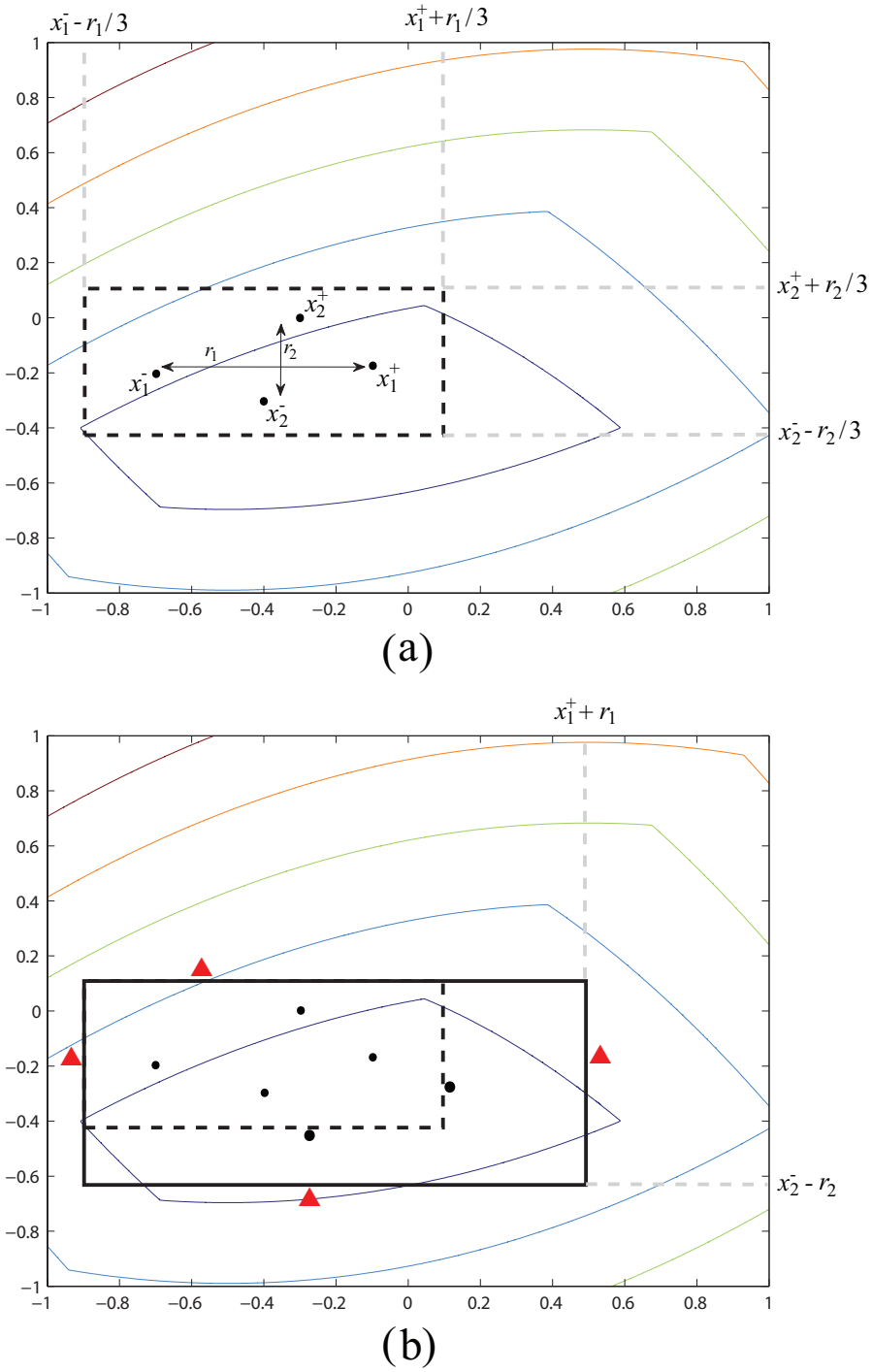


Figure 3.4: The level curves of $f = x_1^2 + 3|x_2 + 0.4| + |x_1 - x_2|$ are shown. Figure (a) shows the initial phase of the non-singleton sub-region update. Note all bounds are problematic for illustrative purposes (a situation not possible in practice). Figure (b) shows the completed update with the new sub-region shown in bold. Points of ascent and descent are denoted ▲, • respectively, where each point is drawn randomly from each face.

problematic lower bound $b_j \in \{B_i(j) : 1 \leq j \leq n\}$ is updated to

$$B_i(j) = \max\{-1, x_j^- - \alpha \max(r_j, \delta)\} \tag{3.12}$$

and each problematic upper bound $b_{j+n} \in \{B_i(j+n) : 1 \leq j \leq n\}$ is updated to

$$B_i(j+n) = \min\{1, x_j^+ + \alpha \max(r_j, \delta)\}, \tag{3.13}$$

where $\alpha = \frac{1}{3}$, δ is the minimum hyperrectangle radius and $r_j = x_j^+ - x_j^-$ is the range in dimension j of the set $\{x : x \in \omega_L \cap A_i\}$. Other choices of $\alpha > 0$ are possible provided they are finite. This initial step brings all problematic bounds closer to the convex hull of points in A_i (see Figure 3.4 (a)).

If $|b_j|$ or $|b_{j+n}| = 1$ after this initial step, the bound is fixed by (3.12) or (3.13) and is not considered problematic for the remainder of the update. Otherwise, the updated bounds are tested by drawing a point x from a uniform distribution over each updated face of A_i and evaluating $f(x)$.

Definition 26. (Upper-face). *For a low sub-region A_i with bounds B_i, the face defined by setting $b_j \leftarrow b_{j+n}$, where $b_j, b_{j+n} \in \{B_i(q) : 1 \leq q \leq 2n\}$ and $1 \leq j \leq n$, is called the j^{th} upper-face of A_i.*

Definition 27. (Lower-face). *For a low sub-region A_i with bounds B_i, the face defined by setting $b_{j+n} \leftarrow b_j$, where $b_j, b_{j+n} \in \{B_i(q) : 1 \leq q \leq 2n\}$ and $1 \leq j \leq n$, is called the j^{th} lower-face of A_i.*

The j^{th} upper and lower-faces define subsets of the hyperplanes $x_j = b_{j+n}$ and $x_j = b_j$ respectively. Therefore, generating an x uniformly on j^{th} face is equivalent to uniformly sampling A_i with the j^{th} coordinate fixed(see Section 3.5.2). If a delivered point on the j^{th} upper or lower-face is an element of the set $\{x : \|\varphi H_k x\|_\infty > 1\}$, then $x \notin \mathcal{S}$ with $m(A_i \setminus \mathcal{S}) > 0$. In this case, the updated bound is fixed and no longer considered problematic as it is sufficiently close to the bounds of $\mathcal{S}$. Otherwise, $f(x)$ is evaluated at each x on the remaining faces. If $f(x) > f(x_j^+)$ on the j^{th} upper-face of A_i, a higher function value has been generated and x is included in $\{\omega_H\}$. In this case the upper bound b_{j+n} is fixed (by 3.13). Similarly, if $f(x) > f(x_j^-)$ on the j^{th} lower-face, the lower bound b_j is fixed (by 3.12). Otherwise, descent was made, x is added to $\{\omega_L\}$ and the bound remains problematic.

For all remaining problematic bounds, α is increased iteratively in (3.12) and (3.13), in a standard forward-tracking manner. Here the sequence $\alpha = \frac{1}{3}, 1, 3, \ldots, 3^{10}$ is used

although other choices are possible. The method terminates when either every $b \in \{B_i(q) : 1 \leq q \leq 2n\}$ is not considered problematic, or 3^{10} is reached. Each non-singleton $A_i \subset \mathcal{L}_k$ is considered in turn and the update is applied. The points generated throughout the update are retained and included in the training data set as recent points. Including these points in T reduces the risk of having problematic bounds when the next CART partition is formed.

Numerical simulations performed by the author found that the CART partition generally works well, resulting in few problematic bounds. However, for the few cases where a problematic bound occurs, the performance of CARTopt is dramatically reduced if the update is not applied. The reader is referred to Figure 3.3 once more, where the x_2 upper bound is problematic. From the contours of f, it is clear that any point generated on the first upper-face ($\alpha = 1/3$) would be a point of ascent. Thus, the boundary is fixed at the cost of one function evaluation and the updated $\mathcal{L}$ now matches the training data. If the boundary is not fixed, the probability that a point drawn uniformly from $\mathcal{L}$ is an element of $\{\omega_H\}$ exceeds 90%, compromising efficiency.

3.4.4 Singleton Low Sub-Regions

The CART method can form a partition on $\mathcal{S}$ such that, singleton low sub-regions exist. These sub-regions can be problematic as they are often defined by two close, parallel hyperplanes that extend to the boundary of $\mathcal{S}$ (see Figure 3.5). Thus, the inclusion of these sub-regions in $\mathcal{L}$ can give a poor approximate level set. Furthermore, a singleton low sub-region can have relatively huge measure, when compared to the measure of the union of non-singleton sub-regions. Therefore, sampling $\mathcal{L}_k$ uniformly can mislead the algorithm, drawing most samples from the singleton sub-region. To maintain algorithm efficiency, directing the search in the neighborhood of points with known relatively low function values, a post-partition modification is made.

In the extremely unlikely case when every low sub-region is a singleton sub-region, the partition is updated as follows. For each $x \in \{\omega_L\}$, a hypercube of radius r and center x defines each sub-region A_i, where $i = 1, \ldots, |\{\omega_L\}|$. The hypercube radius r at iteration k is defined as

$$r = \frac{1}{2} \max \left\{ \left(\frac{m(\mathcal{L}_{k-1})}{|\{\omega_L\}|} \right)^{1/n}, \delta \right\},$$

where $\mathcal{L}_0 = \mathcal{S}$ and $\delta > 0$ is the minimum sub-region radius. The approximate level set $\mathcal{L}_k$ is defined by the union of hypercubes. The inclusion of δ ensures $m(\mathcal{L}_k) > 0$

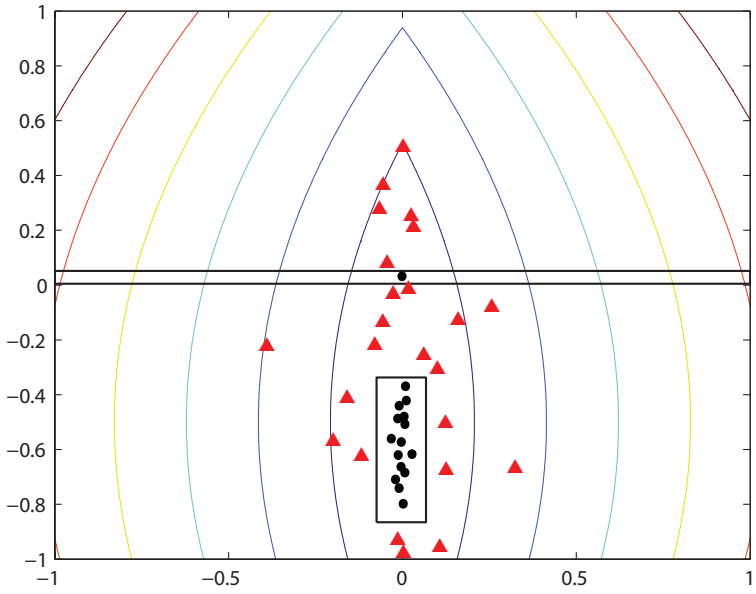


Figure 3.5: The level curves of $f = 5|x_1| + (x_2 + 0.5)^2$, along with the training data set $T = [\omega_L, \omega_H]$, with $x \in \{w_L\}$ and $z \in \{w_H\}$ denoted •, ▲ respectively, is shown. Note how the singleton low sub-region is defined by a thin band, resulting in a poor approximate level set.

for all k. Basing each hypercube radius on $m(\mathcal{L}_{k-1})$ ensures the algorithm searches at a similar level to the previous iteration, keeping the search focused. Each singleton sub-region has equal measure and hence, has equal probability of being sampled during the next sampling phase (see next section).

When there exists at least one non-singleton low sub-region, the partition is updated as follows. The notation $\mathcal{A}_i$ is used to denote a singleton sub-region of $\mathcal{L}_k$. For each $x \in \{\omega_L \cap \mathcal{A}_i\}$, a hypercube replaces each $\mathcal{A}_i$ with center x and radius r, defined by

$$r = \frac{1}{2} \max \left\{ \left(\frac{m(\mathcal{L}_k) - m(\mathcal{A})}{|\omega_L| - |\mathcal{A}|} \right)^{1/n}, \delta \right\}, \tag{3.14}$$

where $m(\mathcal{A})$ and $|\mathcal{A}|$ denote the measure and number of singleton sub-regions, respectively. Replacing each $\mathcal{A}_i$ with a hypercube, $\mathcal{L}_k$ is updated as the union of low sub-regions. After applying (3.14), each singleton sub-region occupies approximately $1/|\{\omega_L\}|$ of the total measure of $\mathcal{L}_k$. This is chosen because the $x \in \mathcal{A}_i$ occupies $1/|\{\omega_L\}|$ of the point space of $\{\omega_L\}$.

3.5 Generating The Next Batch

At each iteration the batch of N points X_k is distributed mainly or totally into the approximate level set $\mathcal{L}_k$. At the user's discretion, a specified number of samples $\lfloor \chi N \rfloor$ can be distributed into the high region $\mathcal{S}$, where $0 \leq \chi \leq 1$. Furthermore, the user can specify the number of iterations $K_S \geq 0$ for which sampling $\mathcal{S}$ is required. With $\chi > 0.5$ the algorithm generates more samples in $\mathcal{S}$ and becomes more of a global, rather than local optimization method. In fact, with $\chi = 1$ and $K_S = \infty$, CARTopt reduces to Pure Random Search.

Local nonsmooth optimization is of primary interest here, rather than global optimization. Therefore, a greedy strategy of sampling only $\mathcal{L}_k$ at each iteration ($\chi = 0$) is used hereafter. This increases the chance of reducing f and the measure of $\mathcal{L}$ at each iteration. Ideally, each $\mathcal{L}$ would be nested. However, as each is an approximate level set this is usually not true.

3.5.1 Batch Size

Before describing how each batch of points is generated, the batch size $N > 0$ is chosen. To choose a suitable N the efficiency of uniform sampling over $\mathcal{L}$ is considered. Firstly, let us define the η-percentage set of $\mathcal{L}$.

Definition 28. (η-percentage set). *Let f_* and f^* denote the minimum and maximum of f on $\mathcal{L}$, respectively. For any $\eta \in [0, 1]$, assume there exists an $f_\eta \in [f_*, f^*]$ for which the set*

$$\mathcal{S}(\eta) = \{x \in \mathcal{L} : f(x) < f_\eta\},$$

has $m(\mathcal{S})/m(\mathcal{L}) = \eta$. Such a set is called the η-percentage set of $\mathcal{L}$.

Therefore, $\mathcal{S}(1) = \mathcal{L}$ and in the limit as $\epsilon \to 0$, $\mathcal{S}(\epsilon)$ converges to the set of minimizers $\{x \in \mathcal{L} : f(x) = f_*\}$.

Let us now consider how many sample points are required to generate at least one point in each $\mathcal{S}(\eta)$ as η is decreased. The probability that there exists at least one point $x \in \mathcal{S}(\eta)$ out of N uniform draws over $\mathcal{L}$, is given by,

$$Pr = 1 - (1 - \eta)^N. \tag{3.15}$$

Choosing a high probability, $Pr = 0.99$ and rearranging (3.15) with respect to N,

$$N = \frac{\ln(0.01)}{\ln(1 - \eta)},$$

the expected number of points required to generate an $x \in \mathcal{S}(\eta)$ is obtained.

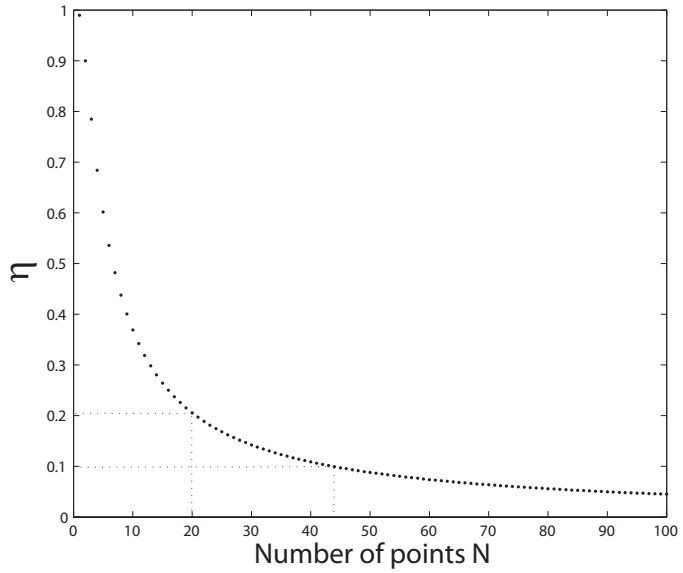


Figure 3.6: The expected number of sample points required to generate an $x \in \mathcal{S}(\eta)$ from uniform sampling over $\mathcal{L}$, with probability 0.99.

From Figure 3.6 it is clear that uniform sampling is very effective at reducing f in the early stages of sampling, only requiring 44 sample points to generate an $x \in \mathcal{S}(0.1)$ with probability 0.99. However, to reduce f significantly further, generating an $x \in \mathcal{S}(0.05)$ say, an extremely large sample size is required.

At each iteration $\mathcal{L}_k$ is sampled using a near-uniform distribution. Therefore, to maintain algorithm efficiency, it would be advantageous to alternate between partition and sampling phases with relatively small batch sizes. Sampling $\mathcal{L}_k$ for too long would cause CARTopt to become inefficient, with a large number of samples generated in $\mathcal{L}_k$ failing to reduce f below the current lowest function value. Furthermore, each partition phase defines a new, possibly smaller, promising subset of $\mathcal{S}$ to search. Here the value $N = 20$ is chosen. Hence, $\mathcal{L}_k$ is sampled until an $x \in \mathcal{S}(0.2)$ has been obtained with probability 0.99, before another partition phase is implemented. Numerical simulations performed by the author supports this batch size.

For the initial batch $2N = 40$ samples are used. Therefore, a point $x \in \mathcal{S}(0.1)$ will be an element of $\mathcal{L}_1$ with high probability. This gives the algorithm at least one sufficiently low point in the initial partition and hence, a promising sub-region to begin the search for an optimal point.

3.5.2 Delivery Method

Generating samples in the high region (if required) is trivial, simply, $\lfloor \chi N \rfloor$ points are drawn uniformly from the set $\{x : x \in [-1,1]^n\}$. These points need not be generated in the transformed space and can simply be generated after the inverse transform in Step 3(f) of CARTopt (see Figure 3.1).

To sample the approximate level set, a near-uniform distribution over $\mathcal{L}_k$ is used. The delivery method requires a two stage process. Firstly a low sub-region $A_i \subset \mathcal{L}_k$ is selected, then a point x is drawn from a uniform distribution over A_i. To select each A_i, a simple discrete inverse transform method is used [46]. The cumulative distribution function F for the sub-region measure of $\mathcal{L}_k$ is given by

$$F(A_i) = \sum_{q=1}^{i} m(A_q)/m(\mathcal{L}_k), \tag{3.16}$$

where $m(.)$ denotes Lebesgue measure. The measure for each A_i is given by $m(A_i) = \prod_{j=1}^{n} (b_{n+j} - b_j)$, where $b_j, b_{j+n} \in \{B_i(q) : 1 \leq q \leq 2n\}$. A particular A_i is selected using

$$A_i = \max(i : U \leq F(A_i)), \tag{3.17}$$

where $U \in [0,1]$ is a random variable. Hence, more samples are expected to be drawn from larger sub-regions of $\mathcal{L}_k$.

Upon selecting a low sub-region A_i a point $x \in A_i$ is delivered using

$$x = [b_{n+1} - b_1, \ldots, b_{2n} - b_n]U_n + [b_1, \ldots, b_n],$$

where U_n is a diagonal matrix of rank n with each non-zero element $u_{jj} \in [0,1]$ a random variable. The method is repeated until the batch of N (or $\lfloor (1-\chi)N \rfloor$ if sampling $\mathcal{S}$) points are obtained.

Performing the partition in the transformed space (Step 3(c) of CARTopt) means the set

$$\{y \in \mathcal{L}_k : \varphi H_k y \notin \mathcal{S}\} \tag{3.18}$$

can have positive measure at iteration k. To ensure f is evaluated only at points within $\mathcal{S}$, an acceptance/rejection sampling method is applied. Specifically, a delivered point belonging to the set $\{x \in A_i : \|\varphi H_k x\|_\infty > 1\}$ is rejected and the process repeats, selecting a new sub-region. In practice, the measure of (3.18) tends to be small (if it exists at all) and so acceptance/rejection sampling is an effective strategy. Clearly,

if $\mathcal{S} \equiv \mathbb{R}^n$ (unconstrained CARTopt algorithm) acceptance/rejection sampling is not required.

If no post-partition modifications were required at iteration k, $A_i \cap A_j = \emptyset$ for all $i \neq j$. Therefore, the proposed delivery method samples $\mathcal{L}_k$ uniformly. Whereas, a modified partition can have an overlap such that $A_i \cap A_j \neq \emptyset$. All overlaps have a greater probability of being sampled as they can be sampled from multiple sub-regions. However, such an overlap tends to be small, giving near-uniform sampling on $\mathcal{L}_k$ and does not have any adverse effects on the algorithm.

In both cases, the sampling method generates N points in $\mathcal{L}_k \cap \mathcal{S}$, a property needed for convergence.

Proposition 29. *The sampling method described above generates N points in the feasible approximate level set $\mathcal{L}_k \cap \mathcal{S}$ with probability one.*

Proof. Clearly if $\{\mathcal{L}_k \cap \mathcal{S}\} = \mathcal{L}_k$ each sample drawn is feasible and the result follows.

Otherwise, $\{\mathcal{L}_k \cap \mathcal{S}\} \neq \mathcal{L}_k$ and acceptance/rejection sampling is required to reject infeasible samples. Generating a feasible point is a Bernoulli trial — a feasible point is generated or not — where the probability of generating a feasible sample is given by

$$P = \frac{1}{\sum_{i=1}^{|A|} m(A_i)} \sum_{i=1}^{|A|} m(A_i \cap \mathcal{S}) > 0,$$

where $|A|$ is the number of low sub-regions. Noting that repeated trials are independent with constant probability P, the number of points generated in $\{\mathcal{L}_k \cap \mathcal{S}\}$ after j trials is a Binomial Random Variable. Thus, the probability that at least N samples are drawn from $\{\mathcal{L}_k \cap \mathcal{S}\}$ after $j > N$ trials is

$$Pr(\text{number of feasible points} \geq N) = 1 - \sum_{q=0}^{N-1} \frac{j!}{q!(j-q)!} P^q (1-P)^{j-q}. \tag{3.19}$$

The proposed sampling method draws samples from $\mathcal{L}_k$ indefinitely until N are obtained. Therefore, we consider the limit of (3.19) as $j \to \infty$. That is, the limit of each term in the summation of (3.19), given by

$$\frac{P^q}{q!} \lim_{j \to \infty} \frac{j!}{(j-q)!} (1-P)^{j-q}. \tag{3.20}$$

Expanding the numerator of (3.20)

$$\frac{P^q}{q!}\lim_{j\to\infty}\frac{j(j-1)\dots(j-(q+1))(j-q)!}{(j-q)!}(1-P)^{j-q}$$

and simplifying, the limit is expressed as

$$\frac{P^q}{q!}\lim_{j\to\infty}\frac{j(j-1)\dots(j-(q+1))}{\frac{1}{(1-P)^{j-q}}}. \tag{3.21}$$

The numerator of (3.21) is a polynomial of degree j^{q+2} and hence, in the limit as $j\to\infty$ the numerator tends to infinity. With $(1-P)<1$ and $0<q<N<j$, in the limit as $j\to\infty$, $(1-P)^{j-q}\to 0$ and so the denominator of (3.21) also tends to infinity. That is, (3.21) is in indeterminant form and thus, applying l'Hôpital's rule $q+2$ times we obtain

$$\frac{P^q}{q!}\lim_{j\to\infty}\frac{(q+2)!}{\frac{(-\ln(1-P))^{q+2}}{(1-P)^{j-q}}},$$

which simplifies to

$$\frac{(q+2)(q+1)P^q}{(-\ln(1-P))^{q+2}}\lim_{j\to\infty}(1-P)^{j-q}=0.$$

Hence, each term of the summation in (3.19) tends to zero in the limit as $j\to\infty$ and thus, N feasible points are drawn from $\mathcal{L}_k\cap\mathcal{S}$ with probability one. □

3.5.3 Effectiveness of Distribution Method

This sub-section illustrates the effectiveness of the distribution method via an example. Consider, for example, solving the nonsmooth Rosenbrock function, given by,

$$f(x)=|10(x_2-x_1^2)|+|1-x_1| \text{ such that } x\in[-2,2]^2, \tag{3.22}$$

with $x_*=(1,1)$ the unique essential local minimizer. With reference to Figure 3.7 it is clear that after only ten iterations (240 points) the algorithm has successfully identified a subset containing x_*, where the density of points is greatest. As the greedy sampling strategy is applied here, all subsequent samples will be drawn in the neighborhood of x_* refining the current estimate of x_*. Clearly this sampling strategy is more effective than uniform sampling over $[-2,2]^2$. To obtain a similar density of points in the neighborhood of x_*, far more points would be required.

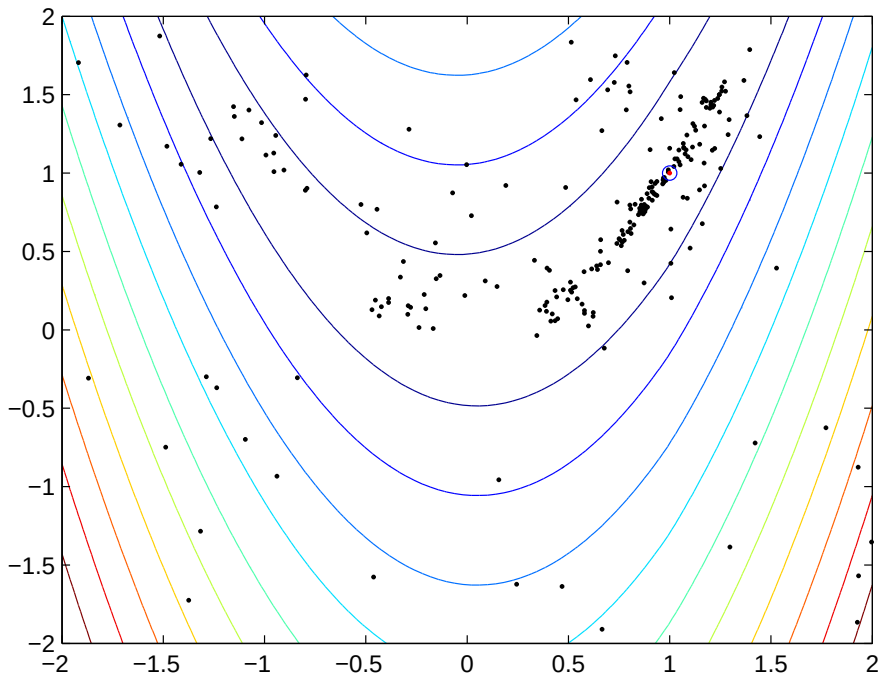


Figure 3.7: Ten iterations of the CARTopt method applied to (3.22), showing all points generated using the greedy sampling strategy. The essential local minimizer $x_* = (1, 1)$ is shown in red.

3.6 A Deterministic CARTopt Instance

This section considers sampling the approximate level set $\mathcal{L}$ as evenly as possible. Previously, CARTopt was described using uniform random sampling on $\mathcal{L}_k$. Here a deterministic approach to try and achieve a more even covering of $\mathcal{L}_k$ is considered. In doing so, a deterministic instance of the CARTopt method is proposed. For simplicity, let $\mathcal{L}_k$ be a unit hypercube in what follows.

Intuitively, one may think the most uniformly distributed set in a hypercube is given by a regular, rational lattice of points. However, this is only true for $n = 1$ and is far from best for $n \geq 2$ [73]. Consider minimizing a function f of n variables, which depends largely on $\hat{n} < n$ leading variables, a situation common in practice. Evaluating f on a regular lattice of N points in the hypercube would only give approximately $N^{\hat{n}/n}$ different function values, due to the linearity of weakly interacting variables. Therefore, if f is computationally expensive to evaluate, efficiency is lost and many evaluations are wasted. The use of uniform random sampling avoids this and embedding a regular lattice on $\mathcal{L}_k$ is considered no further.

Uniform random sampling on $\mathcal{L}_k$ is an efficient way to obtain information about

f, but clustering can occur leaving subsets unexplored. One approach to remove clustering is to discard points falling too close to already accepted points. The interested reader is referred to [80] for further details. However, these methods can struggle to generate samples in the later stages of sampling, discarding many generated points. Furthermore, deciding on which points are deemed *too close* can be problematic [81].

Another approach is to sample $\mathcal{L}_k$ using a quasi-random sequence. Quasi-random sequences satisfy dispersion conditions, the net effect generating points in a highly correlated manner, giving a more even distribution. For example, for sufficiently large N, uniform random sampling in a hypercube can potentially leave a half space empty, whereas, a quasi-random sequence will not. Here the Halton sequence [31] is of particular interest.

3.6.1 Halton Sequence

The Halton sequence is a quasi-random sequence of numbers which generates evenly distributed points in low dimensions. These sequences are based on van der Corput sequences. The van der Corput sequence in base p ($p \geq 2$) is constructed by reversing the base p representation of the sequence of natural numbers. More precisely, the sequence $\{x_1, x_2, \ldots\}$ with $x_k = \phi_p(k)$ for all $k \geq 1$, where $\phi_p(k)$ is the radical inverse function

$$\phi_p(k) = \sum_{j=0}^{\infty} \frac{\lambda_j(k)}{p^{1+j}}$$

and $\lambda_j(k) \in Z_+$ are the unique coefficients of the base p expansion of k

$$k = \sum_{j=0}^{\infty} \lambda_j(k) p^j.$$

For example, consider the binary system with $p = 2$. The binary expansion of $k = 6$ is 110 with radical inverse $\phi_2(6) = 0.011$ and converting back to the decimal system, $x_6 = 3/8$. The first ten members of the van der Corput sequence with base 2 and 3 are given by the sets

$$\left\{\frac{1}{2}, \frac{1}{4}, \frac{3}{4}, \frac{1}{8}, \frac{5}{8}, \frac{3}{8}, \frac{7}{8}, \frac{1}{16}, \frac{9}{16}, \frac{5}{16}\right\}$$

and

$$\left\{\frac{1}{3}, \frac{2}{3}, \frac{1}{9}, \frac{4}{9}, \frac{7}{9}, \frac{2}{9}, \frac{5}{9}, \frac{8}{9}, \frac{1}{27}, \frac{10}{27}\right\}.$$

Thus, the van der Corput sequence partitions the unit interval with respect to the

chosen base. In particular, $1/p, 1/p^2, 1/p^3$ etc. Furthermore, the sequence generates a dense set of points in $(0, 1)$ [1].

The Halton sequence in $\mathbb{R}^n$ is

$$\{\phi_{p_1}(k), \phi_{p_2}(k), \ldots, \phi_{p_n}(k)\}_{k=1}^{\infty}, \tag{3.23}$$

where $p_1 = 2, p_2 = 3$ and p_j is the j^{th} prime. The sequence for 100 points in $[0, 1]^2$ is shown in Figure 3.8. The covering is more even than that obtained from uniform random sampling.

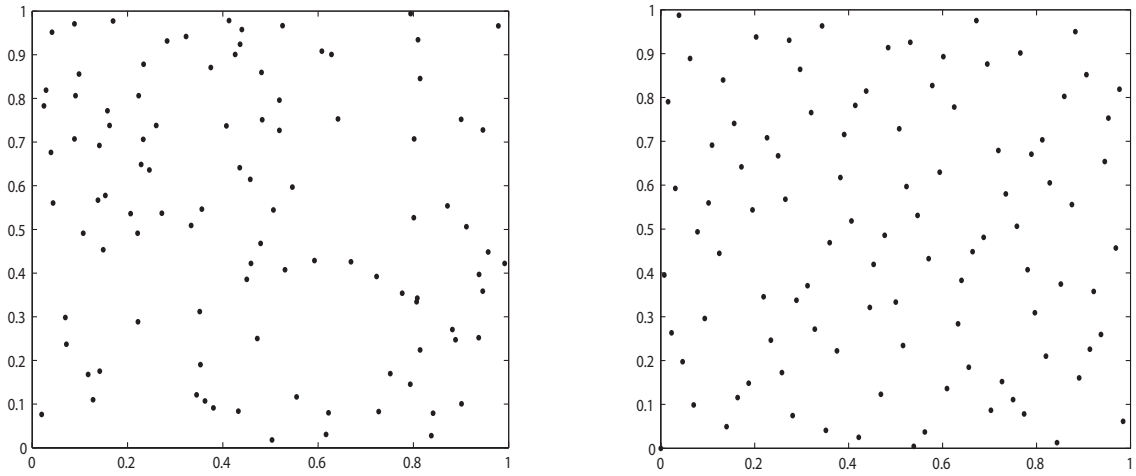


Figure 3.8: Uniform random distribution (left) and Halton sequence (right) for 100 points in $[0, 1]^2$.

Using the Halton sequence in high dimensions can be problematic. When using particular primes, individual Halton sequences can be highly correlated for lengthy periods. The interested reader is referred to [81] for further details and to [33] for modified sequences, including the scrambled and shuffled Halton sequences, which aim to minimize correlation. However, relatively low dimensions and relatively small subsequences of the Halton sequence are used here and the modified versions are considered no further.

3.6.2 Implementation

Deterministic CARTopt uses elements of the Halton sequence, rather than uniform random sampling, to generate each sample point and replaces each instance where a random variable is used. The algorithm is essentially the same as that given in Figure

3.1 and only differs at steps 2 and 3(d,e). The method draws elements sequentially from the one dimensional Halton sequence

$$\{\phi_{p_1}(m)\}_{m=1}^{\infty} = \{r_1, r_2, \ldots\} \text{ on } [0,1] \tag{3.24}$$

and the n dimension Halton sequence

$$\{\phi_{p_2}(m), \phi_{p_3}(m), \ldots, \phi_{p_{n+1}}(m)\}_{m=1}^{\infty} = \{x_1, x_2, \ldots\} \text{ on } [0,1]^n, \tag{3.25}$$

where $p_1 = 2, p_2 = 3, p_3 = 5$ and p_j is the j^{th} prime number. Here (3.24) and (3.25) are indexed with m, where initially $m = 1$.

The first $2N$ elements of (3.25) are extended to $\mathcal{S} = [-1,1]^n$, using the scaling $2x_m - 1$, to give the initial batch of points (see Step 2, Figure 3.1) and the sequence index counter is set to $m = 1 + 2N$.

To remove the stochastic sampling of $\mathcal{L}_k$ (Step 3(e), Figure 3.1), each approximate level set is sampled using a similar approach to the method proposed in Section 3.5.2. Here the m^{th} element of (3.24) selects a sub-region $A_i \subset \mathcal{L}_k$ and the m^{th} element of (3.25) is used to sample A_i. Consider generating a point $z \in \mathcal{L}_k$. Firstly, a sub-region is selected using

$$A_i = \max(i : r_m \leq F(A_i)),$$

where $F(A_i)$ is the sub-region measure CDF of $\mathcal{L}_k$ (see (3.16)). A point z is then delivered into A_i using

$$z = [b_{n+1} - b_1, \ldots, b_{2n} - b_n]U_n + [b_1, \ldots, b_n], \tag{3.26}$$

where U_n is a diagonal matrix with $U_{jj} = x_m(j)$ and b_j, b_{j+n} are the lower/upper bounds in dimension j of A_i. Increment m. If $\|\varphi H_k z\|_\infty > 1$, the delivered point is not an element of $\mathcal{S}$ and z is rejected. The process repeats until N points are generated.

Rather than restarting the sequence for each batch, m is incremented indefinitely. This is done from a practical point of view. If successive approximate level sets are similar in shape, size and location, using the same subsequence of the Halton sequence can generate similar iterates. This is discussed further in Section 3.6.3.

The sampling method is illustrated in Figure 3.9, where an approximate level set in $\mathbb{R}^2$ is shown. The proposed sampling method generates an even covering over $\mathcal{L}$, despite its irregular shape. Sampling successive approximate level sets may displace various parts of the Halton sequence relative to one another, thus potentially destroying the

efficiency of the covering over successive sampling phases. However, it does provide an even covering at each iteration and provides an interesting deterministic method for sampling $\mathcal{L}_k$. Furthermore, this method generates a dense set of points in $\mathcal{L}$. To prove this result, the following proposition from [1] is used.

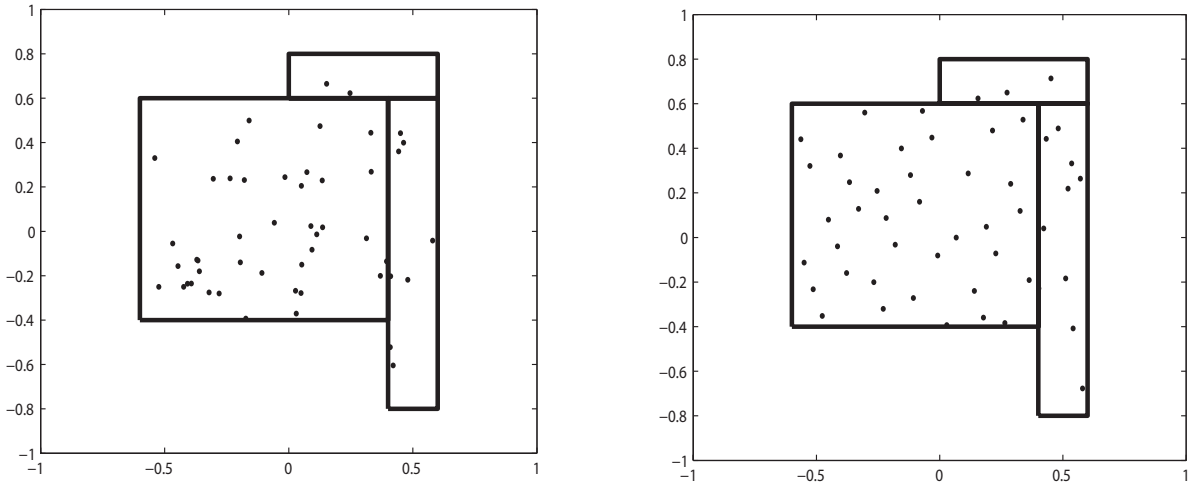


Figure 3.9: Uniform random distribution method (left) and Halton sequence method (right) for generating 50 points in $\mathcal{L}$ (bold region).

Proposition 30. *The Halton sequence* (3.23) *is dense in* $[0,1]^n$.

The interested reader is referred to [1] for a simple proof and [31] for a more detailed proof.

The reader is referred to Figure 3.10 to accompany the proof of the following proposition.

Proposition 31. *The proposed deterministic sampling method is dense in* $\mathcal{L}_k$ *if sampling is performed indefinitely.*

Proof. It is sufficient to show that the proposed sampling method is equivalent to sampling $[0,1]^{n+1}$ using the Halton sequence. At each iteration, r_m selects a sub-region and a point x_m is delivered (under appropriate scaling, see (3.26)). The pair $\{r_m, x_m\}$ is simply the m^{th} element of the $n+1$ dimensional Halton sequence. Proposition 30 states the sequence

$$\{r_m, x_m\}_{m=1}^{\infty}$$

is dense in $[0,1]^{n+1}$. Clearly, if a finite number of elements are removed from a dense set, the set of remaining elements form a dense set. Thus, the subset of the Halton

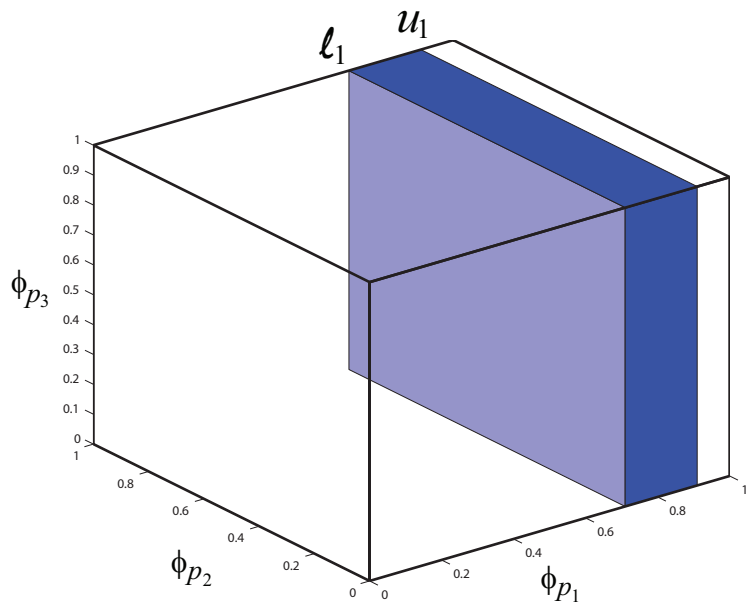


Figure 3.10: $[0,1]^{n+1}$ search space for sampling the approximate level set $\mathcal{L}_k$ given in Figure 3.9. The ϕ_{p_1} axis denotes the normalized measure of each sub-region $0 < m(A_i) \leq 1$ and the ϕ_{p_2}, ϕ_{p_3} axes give the normalized x_1, x_2 sides of each sub-region.

sequence

$$\{r_m, x_m\}_{m=\lambda}^{\infty} \tag{3.27}$$

is also dense in $[0,1]^{n+1}$, where λ is the sequence index counter of the Halton sequence used by the deterministic CARTopt algorithm. That is, the current position in the Halton sequence at iteration k.

With the number of sub-regions finite, $1 \leq |A| \leq |\{\omega_L\}|$, and $m(A_i) > 0$ for each $A_i \subset \mathcal{L}_k$, selecting and sampling a sub-region A_i is simply a subset of $[0,1]^{n+1}$ with positive measure. Specifically, for an A_i it is the set

$$\{[\ell_1, u_1], [0,1]^n\}, \tag{3.28}$$

where

$$[\ell_1, u_1] = \left[\frac{1}{\sum_{i=1}^{|A|} m(A_i)} \left(\sum_{j=0}^{i-1} m(A_j), \sum_{j=1}^{i} m(A_j) \right) \right]$$

and $m(A_0) = 0$ (see Figure 3.10). Therefore, (3.27) is also dense in the subset (3.28). That is, each sub-region A_i is selected infinitely many times and densely sampled. With $\mathcal{L}_k = \cup A_i$, the proposed sampling method is dense in $\mathcal{L}_k$. □

To remove all stochastic components from CARTopt, elements of (3.25) are used to remove problematic bounds during the partition phase (Step 3(d), Figure 3.1). Recall from Section 3.4.3, a sample needs to be drawn from the upper/lower face of a sub-region A_i if the upper/lower bound on A_i is problematic. Given the current position m in the sequence (3.25), the element x_m is scaled to fit the face and m is incremented. All the required points are drawn from (3.25) until the post-partition modification is complete.

With all the stochastic elements of the CARTopt method removed, a deterministic CARTopt instance is established.

3.6.3 Notes on Convergence

The current form of the deterministic instance of the CARTopt algorithm has no convergence proof. This is because approximate level sets are offset relative to each other. This can cause various parts of the Halton sequence to be displaced relative to one another, which can potentially leave parts of the search region unexplored. Using Proposition 31 and fixing $\mathcal{L}_k$ for some sufficiently large k, one could establish convergence to a minimizer of f. However, one would have to assume $z_* \in \mathcal{L}_k$, which is difficult to verify on general functions. Thus, choosing a sufficiently large k is problematic in practice.

Although no convergence result is given, the deterministic instance of CARTopt performs well in practice. When stopping conditions are included (see Chapter 4), the algorithm terminates at accurate approximations to essential local minimizers. The reader is referred to the numerical results chapter of this thesis for further details.

3.7 An Unconstrained CARTopt Instance

This section introduces the unconstrained instance of CARTopt, where $\mathcal{S} = \mathbb{R}^n$ is used. Like the deterministic instance described in the previous section, the unconstrained algorithm only differs slightly from the algorithm in Figure 3.1. Each change is detailed in the sections which follow, as they appear in Figure 3.1. Combining the ideas from this and the previous section in the obvious way, gives an unconstrained deterministic CARTopt instance.

3.7.1 Initialization

Unconstrained CARTopt generates an initial training data set T_1, in Step 1. To get an initial batch of points the user chooses an $x_0 \in \mathbb{R}^n$ and draws a further $2N-1$ points X_0, uniformly from the set

$$X = \{x \in \mathbb{R}^n : \|x - x_0\|_\infty < h\}, \tag{3.29}$$

where h is a finite positive constant. If all elements of X have infinite functions values, continue sampling until at least one finite function value exists. Setting $T_1 = \{X_0 \cup x_0\}$, gives CARTopt an initial training data set to form a partition on $\mathbb{R}^n$. With $|T_1| \geq 2N$, Step 2 of the algorithm is not implemented.

3.7.2 Training Data Transformation

The transformation given in Section 3.3 is still applied to the training data set to potentially simplify the partition on $\mathbb{R}^n$. However, the scalar φ in equations (3.6) and (3.7) are no longer needed because $\mathcal{S}$ is now unconstrained. There is no need to restrict the partition to $[-1,1]^n$ as any $x \in \mathbb{R}^n$ is also an element of $\mathcal{S}$. Hence, with $1/\varphi$ removed from Step 3(c), φ is also removed from Step 3(f) where the inverse transform is applied.

3.7.3 CART Partition

The CART partition can be applied to $\mathbb{R}^n$ because the hyperplane decision boundaries that form the partition are infinite in extent. The matrix B in Section 2.4.4 containing the bounds of $\mathcal{S}$ is now replaced with

$$B = \begin{bmatrix} -\infty & \dots & -\infty & +\infty & \dots & +\infty \\ \vdots & & \vdots & \vdots & & \vdots \\ -\infty & \dots & -\infty & +\infty & \dots & +\infty \end{bmatrix},$$

because $\mathcal{S}$ is unconstrained. Thus, sub-region bounds given by (3.8) can contain infinite elements however, the post-partition modifications described in Section 3.4 are applied to remove such elements.

The minimum sub-region radius $\delta > 0$, given by equations (3.10) and (3.11), is still needed to ensure $m(\mathcal{L}_k)$ is non-zero for all iterations. However, the bounds of $[-1,1]^n$ are dropped because $\mathcal{S} = \mathbb{R}^n$. Therefore, for each sub-region $A_i \subset \mathcal{L}_k$, the bounds B_i

are updated using

$$B_i(j) = \min\{b_j, x_j^- - \delta\}$$

and

$$B_i(j+n) = \max\{b_{j+n}, x_j^+ + \delta\},$$

where $j = 1, \ldots, n$.

To remove problematic bounds the same methods described in sections 3.4.3 and 3.4.4 are used. However, with $\mathcal{S} = \mathbb{R}^n$, problematic bounds now have the values $\pm\infty$, rather than ± 1 as in the constrained case.

For non-singleton sub-regions with problematic bounds the method proposed in Section 3.4.3 is used. However, the dependence on $[-1, 1]^n$ is removed from the update. Initially, the updates given by (3.12) and (3.13) are replaced with

$$B_i(j) = x_j^- - \alpha \max\{r_j, \delta\}$$

and

$$B_i(j+n) = x_j^+ + \alpha \max\{r_j, \delta\},$$

where $j = 1, \ldots, n$. Operating in $\mathbb{R}^n$ means all problematic bounds are extended in the standard forward-tracking manner until ascent is made, or $\alpha = 3^{10}$ is reached.

Non-singleton sub-regions are updated as described in Section 3.4.4. However, in the extremely unlikely case when every sub-region is singleton at the first iteration ($k = 1$), the measure of each individual hypercube is based on $m(\mathcal{L}_0) = m(X)$ rather than $m(\mathcal{S})$, where X is defined by (3.29).

The post-partition modifications ensure that all problematic bounds are removed. Thus, the approximate level set $\mathcal{L}$ is a finitely bounded subset of $\mathbb{R}^n$.

3.7.4 Sampling Phase

With each bound of $\mathcal{L}_k$ finite, the sampling method described in Section 3.5 is applied. However, every $x \in \mathcal{L}_k$ is also an element of $\mathbb{R}^n$ and so acceptance/rejection sampling is not required in the unconstrained method.

3.8 Convergence Analysis

The convergence properties of CARTopt are analyzed with the stopping conditions removed. This allows us to examine the asymptotic properties of the sequence of

iterates generated by the algorithm. Stopping conditions are included from a practical point of view (see Chapter 4 for details).

In the sections which follow, convergence results for the bound constrained and unconstrained stochastic instances of CARTopt are presented. Note, Theorem 18 cannot be applied to the CARTopt method proposed here. Theorem 18 requires the high region to be sampled and demonstrates convergence to a globally optimal point. The results here demonstrate convergence to locally optimal points. Convergence of the deterministic instances of CARTopt cannot be demonstrated without modifications (see Section 3.6.3). However, we show that an infinite sequence of iterates is generated by the deterministic instances of CARTopt and numerical simulations performed by the author show that the algorithm produces final iterates that are good approximations to essential local minimizers.

Before the results are presented, we show that each instance of CARTopt generates an infinite sequence of iterates.

Theorem 32. *The sequence $\{z_k\}$ generated by each stochastic instance of CARTopt is an infinite sequence with probability one.*

Proof. For $\{z_k\}$ to be an infinite sequence, Step 3 of CARTopt must be a finite process. Noting that $|T_k|$ is finite for all k, steps 3(a, b, c, f, g) are finite processes.

The CART partition in Step 3(d) uses a finite data set T_k and only considers a finite number splits from the set $\{s = (x_j + z_j)/2 : x, z \in T_k\}$. The resulting partition on $\mathcal{S}$ is a finite set of non-empty hyperrectangles, where the number of distinct low sub-regions is bounded by $|\{\omega_L\}| = \lfloor \beta N \rfloor$. Each low sub-region update requires only a finite number of steps. Therefore, the partition phase is a finite process and thus, Step 3(d) is finite.

Proposition 29 ensures that N points are drawn from $\mathcal{L}_k \cap \mathcal{S}$ with probability one. Thus, the sampling phase in Step 3(e) terminates almost surely. □

If the sequence of iterates $\{z_k\}$ is bounded, it follows from Theorem 32 that cluster points exist in $\{z_k\}$ almost surely. In practice the stochastic CARTopt algorithm does generate an infinite sequence of iterates because random number generators satisfy various dispersion conditions. To ensure cluster points exist one could, for example, divide $\mathcal{L}_k$ into N sub-regions of equal measure and draw one sample from each. If N feasible points are not obtained, continue dividing $\mathcal{L}_k$ into a set of nested sub-regions and draw one sample from each until they are obtained. This would ensure the sampling phase in Step 3(e) terminates surely. The following assumption is made to ensure cluster points exist in the sequence $\{z_k\}$.

Assumption 33. *The acceptance/rejection sampling technique in Step 3(e) of the CARTopt algorithm generates N points surely.*

The unconstrained stochastic CARTopt algorithm does not require Assumption 33 because acceptance/rejection sampling is not required.

Corollary 34. *The sequence $\{z_k\}$ generated by the unconstrained stochastic CARTopt algorithm is an infinite sequence.*

Proof. The proof is similar to Theorem 32. Steps 3(a,b,c,d,f,g) are finite processes as demonstrated in Theorem 32.

The unconstrained CARTopt algorithm has $\mathcal{S} \equiv \mathbb{R}^n$ and so the approximate level set is feasible for all iterations. Hence, no acceptance/rejection sampling is required and N points are drawn from $\mathcal{L}$ — Step 3(e) is a finite process. □

Theorem 35. *The sequence $\{z_k\}$ generated by each deterministic instance of CARTopt is an infinite sequence.*

Proof. The proof is similar to Theorem 32. Steps 3(a, b, c, d, f, g) are finite process as demonstrated in Theorem 32.

Proposition 31 ensures a dense set of points is generated in $\mathcal{L}_k \cap \mathcal{S}$ if sampling is performed indefinitely. Thus, N samples are drawn from $\mathcal{L}_k \cap \mathcal{S}$ and Step 3(e) is a finite process. □

The convergence results for the bound constrained and unconstrained stochastic CARTopt algorithms can now be given. To establish convergence to an essential local minimizer the following assumption on f is required.

Assumption 36. *The objective function f is lower semi-continuous.*

Assumption 36 ensures that

$$\liminf_{z \to z_*} f(z) \geq f(z_*).$$

Without Assumption 36, the CARTopt algorithm does not always converge to an essential local minimizer. Consider, for example, the function

$$f(x) = \begin{cases} \|x\| & x \neq 0 \\ 1 & x = 0. \end{cases} \tag{3.30}$$

The origin is not an essential local minimizer of (3.30), but the algorithm will give $\{z_k\} \to 0$ as $k \to \infty$ almost surely. Modifying (3.30) so that $f = -1$ at $x = 0$ makes f lower semi-continuous and the origin is now an essential local minimizer.

3.8.1 Bound Constrained Nonsmooth Result

The convergence result for the constrained CARTopt algorithm shows that every cluster point of the sequence $\{z_k\}$ is an essential local minimizer of f with probability one. In this section the standard definition of an essential local minimizer is slightly modified because $\mathcal{S} = [-1, 1]^n$, rather than $\mathbb{R}^n$.

Definition 37. (Essential local minimizer). *A point $x_* \in [-1, 1]^n$ for which the set*

$$\mathfrak{E}(x_*, \epsilon) = \{x \in [-1, 1]^n : f(x) < f(x_*) \text{ and } \|x - x_*\| < \epsilon\}$$

has Lebesgue measure zero for all sufficiently small positive ϵ is called an essential local minimizer of f.

The convergence result is now given. For convenience let $m(.)$ denote the Lebesgue measure.

Theorem 38. *Let Assumptions 36 and 33 hold. Each cluster point z_* of the sequence $\{z_k\}$ is an essential local minimizer of f with probability one.*

Proof. Theorem 32, Assumption 33 and the fact that $\mathcal{S}$ is bounded ensure the existence of cluster points in $\{z_k\}$.

The proof is by contradiction. Assume z_* is not an essential local minimizer of f. Then, there exists a set

$$\mathfrak{E} \subset \{z \in [-1, 1]^n : f(z) < f(z_*) \text{ and } \|z - z_*\| < \delta/2\}, \tag{3.31}$$

with $m(\mathfrak{E}) > 0$. Also there exists an infinite subsequence $\mathcal{K} \subset \mathbb{Z}_+$ such that

$$\|z_k - z_*\| < \delta/2 \text{ for all } k \in \mathcal{K}, \tag{3.32}$$

where all members of $\mathcal{K}$ are sufficiently large to ensure (3.32) holds. Then,

$$\|z_k - z\| \leq \|z_k - z_*\| + \|z_* - z\| < \delta$$

for all $z \in \mathfrak{E}$. For all k sufficiently large, CARTopt samples near-uniformly in the set

$$\{x \in [-1,1]^n : \|z_k - x\| < \delta\} \tag{3.33}$$

with probability density greater than or equal to $(|\{\omega_L\}|.m(\mathcal{L}_k))^{-1}$. Therefore, CARTopt samples in $\mathfrak{E}$ with probability

$$Pr(x \in \mathfrak{E}) \geq \frac{m(\mathfrak{E})}{|\{\omega_L\}|.m(\mathcal{L}_k)} > 0$$

for all sufficiently large k. Hence, in the limit as $k \to \infty$, $f(z_k) < f(z_*)$ almost surely, contradicting Assumption 36. Thus, z_* must be an essential local minimizer of f, almost surely. □

3.8.2 Unconstrained Result

The unconstrained stochastic instance of CARTopt is now considered. This result follows directly from the constrained results. In order to establish convergence we require the following assumptions.

Assumption 39. *The following conditions hold:*

(a) The points at which f is evaluated at lie in a compact subset of $\mathbb{R}^n$; and

(b) The sequence of function values $\{f(z_k)\}$ is bounded below.

These assumptions ensure $\{z_k\}$ is bounded and excludes the case where $f(z_k) \to -\infty$ as $k \to \infty$.

Theorem 40. *Let Assumptions 36 and 39 hold. Each cluster point z_* of the sequence $\{z_k\}$ is an essential local minimizer of f with probability one.*

Proof. The proof is similar to Theorem 38. Assumption 39 and Corollary 34 ensure the existence of cluster points in $\{z_k\}$.

Replacing $[-1,1]^n$ in both (3.31) and (3.33) with $\mathbb{R}^n$ gives the desired proof. □

Chapter 4

Sequential Stopping Rule

This chapter considers the practical problem of deriving a stopping rule for the CARTopt algorithm. Although the stopping rule is derived for the convergent stochastic instances of CARTopt, it is also applicable to the deterministic instances. Asymptotically, convergence to an essential local minimizer z_* of f is demonstrated for the stochastic instances of CARTopt (see theorems 38 and 40). Appropriate stopping rules are vital to ensure these theoretical results are maintained in practice. However, it is often difficult to know whether the sequence $\{z_k\}$ has converged to a z_* of f.

To achieve practical convergence to an essential local minimizer z_* on a general nonsmooth function $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$, one must verify that there does not exist a subset arbitrarily close to z_* of positive measure with lower f values. This is a constrained global optimization problem of the form

$$\min f(x) \text{ such that } x \in \Omega, \tag{4.1}$$

where Ω is a compact subset of $\mathbb{R}^n$ and $z_* \in \Omega$.

At each iteration k, CARTopt samples an approximate level set $\mathcal{L}_k$ using a near-uniform distribution. If $z_{k^*} = z_*$ at iteration k^*, then there exists an open ball $\mathcal{B}(z_{k^*}, \delta)$ in the dynamic region

$$\Omega_{k^*} = \{x \in \bigcup \mathcal{L}_k \cap \mathcal{S} : k = k^*, k^* + 1, k^* + 2, \ldots\}, \tag{4.2}$$

for which CARTopt has a non-zero probability of sampling for all further iterations, where $\delta > 0$ is the minimum sub-region radius. That is, CARTopt searches globally in the neighborhood of an essential local minimizer with probability one. The union of successive level sets is used to define Ω_{k^*} rather than just $\mathcal{L}_{k^*}$ because each $\mathcal{L}$ is

not necessarily nested. However, $\mathcal{L}_k$ is sampled near-uniformly at each iteration rather than the neighborhood of z_* directly. Thus, if a global minimizer of Ω_{k^*} is found, there does not exist a subset of lower points with positive measure in Ω_{k^*}. That is, an essential local minimizer of f is found and hence, a good place to terminate the CARTopt algorithm.

Unfortunately, no algorithm can solve (4.1) exactly because numerical procedures only obtain approximate answers. Thus, (4.1) is considered solved if an $x \in \Omega_{f_\epsilon}$ is located, defined by

$$\Omega_{f_\epsilon} = \{x \in \Omega : f(x) \leq f(z_*) + \epsilon\}, \tag{4.3}$$

where $\epsilon > 0$ and sufficiently small. However, lower semi-continuous functions are considered here and hence, (4.3) can be a set of measure zero. A simple function, for example, is

$$f(x) = \begin{cases} \|x\| & x \neq 0 \\ -2\epsilon & \text{otherwise,} \end{cases} \tag{4.4}$$

where $f(x_*) = -2\epsilon$. The probability of generating a point in Ω_{f_ϵ} from random sampling is zero. Hence, an approximate solution to the global minimum of (4.4) is not obtained.

From a practical point of view, the definition of an essential local minimum is modified so (4.3) always has positive measure and hence, an approximate solution can be obtained. The definition of an essential local minimizer remains the same.

Definition 41. (Essential local minimizer). *A point $z_* \in \mathcal{S}$ for which the set*

$$\mathfrak{E}(z_*, \zeta) = \{x \in \mathcal{S} : f(x) < f(z_*) \text{ and } \|x - z_*\| < \zeta\} \tag{4.5}$$

has Lebesgue measure zero for all sufficiently small positive ζ is called an essential local minimizer of f.

To define the essential local minimum, let

$$f_{\inf} = \inf\{f_\# : m(L(f_\#, z_*, \zeta)) > 0\} \tag{4.6}$$

where

$$L(f_\#, z_*, \zeta) = \{x \in \Omega : f(x) < f_\# \text{ and } \|x - z_*\| < \zeta\}.$$

Definition 42. (Essential local minimum). *Let $z_* \in \Omega$ be an essential local minimizer of f, then $f(z_*) = f_{\inf}$ is an essential local minimum of f.*

For practical reasons this definition makes sense. If $z_k \to z_*$, then $\|z_k - z_*\|$ becomes small, but the function values may be vastly different. Using Definition 2, $f = -2\epsilon$ is an essential local minimum of (4.4), whereas Definition 42 has $f = 0$ as an essential local minimum. For sufficiently small ϵ these values are similar, but for large ϵ they are not.

It is clear from (4.6) that the modified definition of an essential local minimum ensures (4.3) is a set of positive measure for all $\epsilon > 0$. Under mild conditions, the stochastic instances of CARTopt will sample (4.3) with probability one (see theorems 38 and 40). The global minimum of (4.1) to within ϵ is a good approximation to an essential local minimum of f because even if the measure of $\{\mathfrak{E}(z_*, \zeta) \cap \Omega_{f_\epsilon}\}$ is non-zero, the function values in this set only differ by ϵ at most. Furthermore, with ϵ sufficiently less than ζ, the $m(\{\mathfrak{E}(z_*, \zeta) \cap \Omega_{f_\epsilon}\})$ is typically small.

Firstly, existing stopping rules for random search methods are introduced and comments on their effectiveness for general nonsmooth minimization are made. In Section 4.3 a new stopping rule is introduced based on an expected distribution of f values in the neighborhood of an essential local minimizer. This rule is developed over the sections which follow and explicitly stated in Section 4.8.

4.1 Resource Based Stopping Rules

A simple way to terminate an optimization algorithm is to halt when a predetermined finite resource quantity is reached. Examples include run algorithm for a finite number of iterations, finite CPU time, or a finite number of function evaluations. These stopping rules make no reference to valuable information obtained from previous iterations, for example, objective function values. Such methods often overestimate, or underestimate the work required to solve a general problem to the desired standard. However, these rules can be used as an upper bound, terminating an algorithm if a more sophisticated rule fails to halt within user resources, or if the sequence of iterates is unbounded.

4.2 Sequential Stopping Rules

Sequential stopping rules for global optimization algorithms use valuable information obtained about f at each iteration of an algorithm. These rules are often tested at each iteration. They should require minimal storage and be computationally cheap to

evaluate. Thus, not compromising the overall performance of an algorithm.

Here, sequential stopping rules for random search methods are of particular interest because CARTopt is a random search method on a subset of $\mathbb{R}^n$. The interested reader is referred to [22, 23, 30, 32, 42] for a variety of stopping rules. These rules use statistics of extreme values to calculate an associated confidence in the current candidate for the global minimum. The algorithm halts when sufficient confidence is reached. These rules can be broadly classified into two categories.

Definition 43. (Type A). *A Type A sequential stopping rule is a rule based on the probability of sampling the ϵ-level set, given by* (4.3).

Definition 44. (Type B). *A Type B sequential stopping rule is a rule based on the probability of sampling the ϵ-neighborhood of a global minimizer x_*, defined by the set*

$$\Omega_\epsilon = \{x \in \Omega : \|x - x_*\| \leq \epsilon\}.$$

Of particular interest are rules for the Pure Random Search algorithm (PRS). PRS generates a sequence of iterates $\{z_k, f_k\}$ as follows. Let $x_1, x_2, \ldots$ be independent and identically distributed vectors with common distribution Φ on Ω, with $\{z_0, f_0\} = \{x_1, f(x_1)\}$. Then, $\{z_{k+1}, f_{k+1}\} = \{x_{k+1}, f(x_{k+1})\}$ if $f(x_{k+1}) \leq f_k$, and $\{z_{k+1}, f_{k+1}\} = \{z_k, f_k\}$ otherwise. Thus, PRS generates a decreasing sequence of function values, only updating the current function value when descent is made.

4.2.1 Existing Stopping Rules

Dorea [23] develops sequential stopping rules for PRS. Dorea's rules are based on the probability that Ω_{f_ϵ} or Ω_ϵ is sampled in a single step (Type A and Type B rules respectively). In what follows, it is assumed that Ω is simply a compact subset of $\mathbb{R}^n$ with positive measure.

Dorea's Type B rule terminates the PRS algorithm at iteration k when, given an $\epsilon > 0$ and $0 < \beta < 1$,

$$k \geq -\frac{m(\Omega)\log(\beta)}{\epsilon}. \tag{4.7}$$

Dorea goes on the show that if (4.7) is satisfied, then

$$Pr(\|x_k - x_*\| \leq \epsilon) \approx 1 - \beta.$$

However, (4.7) can require an extremely large number of iterations to terminate PRS when ϵ is sufficiently small. Consider, for example, Ω defined by the unit hypercube

with $\beta = 0.01$, then $k \approx 4.6 * 10^{1/\epsilon}$ is required to solve f to within ϵ. In addition, the derivation of the rule requires the following assumption.

Assumption 45. *Assume $\Omega \subset \mathbb{R}^n$ with $m(\Omega) > 0$ and that Φ is the uniform distribution on Ω. Moreover,*

1. *There exists a unique interior point x_* of Ω such that $f(x_*) = f_*$.*
2. *There exists a positive function $v(t), t > 0$, and a constant $\lambda > 0$ such that for all $\beta > 0$, $\lim_{t \downarrow 0}(v(\beta t)/v(t)) = \beta^{1/\lambda}$ and the following limit*

$$U(z) = \lim_{t \downarrow 0} \frac{f(x_* + tz) - f_*}{v(t)}$$

exists and is strictly positive and finite for all $z \neq \mathbf{0}$.

Assumption 45 is too strong for general nonsmooth minimization. Consider, for example, the function

$$f = |x_1| + x_2^2. \tag{4.8}$$

Let $(x_1, x_2) = (at, bt)$, where $a, b \in \mathbb{R}$ and $t > 0$, giving $f = t|a| + t^2 b^2$. f is linear in t when t is sufficiently small unless $a = 0$ (where f is quadratic). However, when t is large, f is quadratic in t unless $b = 0$ (where f is linear). Hence, f follows a power law in all directions where $a, b \neq 0$, although the power changes as t is increased. Thus, there does not exist the positive function $v(t), t > 0$, and a constant $\lambda > 0$ such that for all $\beta > 0$, $\lim_{t \downarrow 0}(v(\beta t)/v(t)) = \beta^{1/\lambda}$ — failing the second requirement of Assumption 45. In addition, Assumption 45 requires each sample to be drawn from a uniform distribution Φ on Ω.

The CARTopt method is designed for nonsmooth minimization and thus, functions like (4.8) are of primary interest. In addition, the CARTopt method uses a non-uniform distribution on a subset of $\mathbb{R}^n$ defined by a union of approximate level sets which potentially changes from iteration to iteration, rather than uniform distribution over a fixed region Ω. For these reasons alone, rule (4.7) is considered no further.

Dorea [23] also provides two Type A stopping rules which can be applied to general nonsmooth problems. Furthermore, these rules do not require a uniform distribution Φ on Ω, only that each sample be independent and identically distributed with a common distribution Φ on Ω. The probability that f_k is an element of Ω_{f_ϵ} is approximated using the number of samples that are within ϵ of the lowest objective function value obtained. Formally,

$$\rho_k(\epsilon) = \sup\{i : \tau_i > 0, f_{\tau_i} \leq f_k + \epsilon\}$$

and for $j = 1, 2, \ldots, k-1$ we define

$$\tau_{j+1} = \tau_{j+1}(k) = \begin{cases} 0 & \text{if } \{i : 1 \leq i \leq \tau_j, f_i \neq f_{\tau_j}\} = \emptyset \\ \sup\{i : 1 \leq i \leq \tau_j, f_i \neq f_{\tau_j}\} & \text{otherwise,} \end{cases}$$

with $\tau_1(k) = k$. The first rule terminates PRS at iteration $k \geq 2$, if given an $\epsilon > 0$ and $0 < \beta < 1$

$$k \geq \log(\beta) / \log\left(1 - \frac{\rho_k(\epsilon)}{k}\right). \tag{4.9}$$

The second rule terminates at iteration $k \geq 2$ if f_k is repeated for the next m steps, that is, $f_k = f_{k+j}$ where $j = 1, 2, \ldots, m$, and m satisfies

$$k \geq \frac{\log(\beta)}{\log\left(1 - \frac{\rho_k(\epsilon)}{k}\right)} - m. \tag{4.10}$$

Dorea shows that if stopping rule (4.9) is applied, then

$$Pr(f_k - f_* \leq \epsilon) \geq 1 - \beta$$

and if stopping rule (4.10) is applied then

$$Pr(f_k - f_* \leq \epsilon : f_k = f_{k+j}, j = 0, 1, \ldots, m) \geq 1 - \beta.$$

Dorea's Type A stopping rules, given by (4.9) and (4.10), have been investigated by Hart [32]. Hart identified some deficiencies in Dorea's rules which can terminate PRS either too late or too early in particular situations. Firstly, consider $f_1 - f_* \leq \epsilon$ where $\epsilon > 0$ and sufficiently small. Rearranging (4.9) and taking exponentials of both sides we obtain

$$\left(1 - \frac{\rho_k(\epsilon)}{k}\right)^k \leq \beta. \tag{4.11}$$

If a lower point is not found by the k^{th} iteration, then $f_1 = f_k$, with $\rho_k(\epsilon) = 1$. Noting that

$$\lim_{k \to \infty} (1 - 1/k)^k = e^{-1} \approx 0.368,$$

the left hand side of (4.11) is bounded below by ≈ 0.368, while $\rho_k(\epsilon) = 1$. Therefore, even at a modest accuracy of $\beta = 0.1$, a point $x_j \in \Omega$ such that $f(x_j) < f_1$ is required

to terminate PRS. The probability of PRS finding a lower point is given by,

$$Pr(f < f_1) = \frac{m(\Omega_{f_\epsilon})}{m(\Omega)}, \tag{4.12}$$

where Ω_{f_ϵ} is defined by (4.3). With $\epsilon > 0$ and sufficiently small, a large number of iterations may be required to locate x_j, even though a point to within ϵ of f_* has been located. Thus, PRS can halt too late using (4.9). Secondly, consider $f_1 > f_2$ where $f_1 - f_2 \leq \epsilon$. Substituting $\rho_2(\epsilon) = 2$ into (4.11), PRS is terminated at the current iteration if $0 \leq \beta$. Since $0 < \beta < 1$ PRS is terminated after only two points are sampled, even if $f_2 - f_*$ is large. Therefore, PRS can halt too early using (4.9). In addition, Hart shows empirically that rule (4.10) is less reliable than rule (4.9) [32].

To remedy the problems described above, Hart modifies Dorea's rules by introducing two additional parameters. Hart argues the first modification yields a more accurate approximation to the probability that f_k is an element of Ω_{f_ϵ} than Dorea's approximation $\rho_k(\epsilon)$. The second forces k to be sufficiently large to avoid early termination. The interested reader is referred to [32] for details. Empirical evidence provided by Hart suggests the modified rules perform better in practice on the test problems used.

Unfortunately, the sequential stopping rules of Dorea and Hart cannot be applied directly to the CARTopt method. Each rule requires a random sample point to be drawn from a common distribution Φ over the global optimization region Ω. However, this is not the case for the CARTopt method. At each iteration, a batch of random points is drawn from a near-uniform distribution over an approximate level set $\mathcal{L}_k$. Furthermore, the construction of each $\mathcal{L}_k$ is conditionally dependent on the sample points generated in previous iterations. Thus, CARTopt uses a different conditional distribution on a subset of $\mathbb{R}^n$ at each iteration, rather than a common distribution over a fixed region Ω. Therefore, neither Hart's or Dorea's stopping rules can be applied to CARTopt.

4.3 New Sequential Stopping Rule

This section introduces the stopping rule for the CARTopt method. At iteration k, the rule approximates the probability of reducing f below $f(z_k)$ (lowest observed function value), based on observed function values up until iteration k. If this probability is sufficiently small, the algorithm terminates and $f(z_k)$ is an approximation to an essential local minimum of f.

To develop the new rule, the following assumptions are made so the sequence of iterates generated by the CARTopt method have interesting properties.

Assumption 46. *The following conditions hold:*

(a) The points at which f is evaluated at lie in a compact subset of $\mathbb{R}^n$; and

(b) The sequence of function values $\{f(x_k)\}$ generated by the CARTopt method is bounded below.

These assumptions ensure the existence of cluster points in $\{z_m\}$, asymptotically. Thus, the algorithm is converging to something, rather than having $\{z_m\}$ unbounded. For the unbounded case, a resource based rule halts the algorithm. Also, the case where $f(x_k) \to -\infty$ as $k \to \infty$ is excluded.

4.3.1 Empirical Data

The CARTopt method operates using a training data set T_k which provides valuable information about f. For $k > \max\{2, 2(n-1)\}$, CARTopt uses a full size training data set containing $\max\{2N, 2(n-1)N\}$ elements. Recall from Section 3.2, T_k is updated iteratively, keeping the $\gamma = 2N$ points with the least function values and most recent sample points. Let

$$Y = \{f_1, f_2, \ldots, f_\gamma\} \tag{4.13}$$

denote the ordered γ least function values from T_k, where $f_i \leq f_{i+1}$. The set Y is automatically updated during the CARTopt method. However, Barrier functions are considered here and thus, Y can contain infinite function values. The rule proposed here is not implemented if Y contains infinite function values. For the purposes of developing the rule, it is assumed Y contains only finite function values.

The set Y is used to test whether $f(z_k)$ is a global minimum to within ϵ on a subset of

$$\Omega_k = \{x \in \bigcup \mathcal{L}_j \cap \mathcal{S} : j = 1, 2, \ldots, k\}, \tag{4.14}$$

with positive measure. That is, a subset of the union of all approximate level sets. If $f(z_k)$ is the global minimum to within ϵ on a subset of Ω_k, a good approximation to an essential local minimum of f has been located because any sufficiently close sets of lower points only differ by ϵ in f. Verifying $f(z_k)$ is an essential local minimizer is a global optimization problem, so finding a value within ϵ is sufficient in practice. To test the suitability of z_k, an assumption about the expected cumulative distribution of function values in the neighborhood of an essential local minimizer(s) z_* is made.

Assumption 47. *The cumulative distribution of function values of points randomly generated in the level set*

$$L(z_*, \zeta) = \{x \in \Omega_k : f(x) \leq f(z_*) + \zeta\}, \tag{4.15}$$

where $\zeta > 0$ and sufficiently small, will follow a power law distribution.

Assumption 47 is not too restrictive, in fact, f can be discontinuous at z_*. Consider, for example, the function

$$f(x) = \begin{cases} \|x\| & \text{if } x_1 \geq 0 \\ \infty & \text{otherwise,} \end{cases}$$

in two dimensions, where $z_* = \mathbf{0}$. Clearly, all sufficiently low level sets are semi-circles in the positive half plane ($x_1 \geq 0$). The rate at which the area of each semi-circles grow as the radius increases follows squared power law. Thus, the cumulative distribution of function values is expected to also follow a squared power law also.

The example above also illustrates the applicability of the rule to Barrier functions. Specifically, the boundary of $\mathcal{S}$ can be dealt with directly using a discontinuous penalty for all $x \notin \mathcal{S}$ on particular constrained nonsmooth optimization problems. This use of barrier functions is investigated further in the numerical results chapter of this thesis.

Using Assumption 47, the expected cumulative distribution of Y in the neighborhood less than or equal to f_γ of an essential local minimizer is of the form

$$F(f, \kappa) = Pr(f_i \leq f) = \left(\frac{f - f_*}{f_\gamma - f_*}\right)^\kappa, \tag{4.16}$$

where f_* is the essential local minimum (Assumption 46 ensures f_* is finite). From (4.16), the probability of finding a lower point $f_1 - \epsilon$ is obtained and if sufficiently small, z_k is assumed to be a global minimizer to within ϵ on a subset of Ω_k. That is, an estimate of an essential local minimizer of f. However, to approximate this probability a power law function must be fitted to the empirical data Y. The data fitting problem is considered in the next section and this section concludes with the expected range of κ values.

4.3.2 Expected κ Values

Considering the rate at which the level sets of f grow, as f is increased from an essential local minimizer(s) z_*, gives an expected range on κ values. The analysis requires the

following assumption.

Assumption 48. *The following conditions hold;*

(a) The rate θ at which f grows along any ray emanating from z_* in all sufficiently low level sets is bounded by $1/2 \leq \theta \leq 2$; and

(b) Each sufficiently low level set of f in the neighborhood of an essential local minimizer z_* are scaled copies of each other.

Part (a) of Assumption 48 assumes most essential local minimizers fall within the range of being either a quadratic bowl, or a cusp type minimizer (see Figure 4.1). Part (b) does not assume hyper-spherical contours and thus, ill-conditioned minimizers and nonsmooth contours are considered. In what follows, let $L(f_\delta)$ denote a sufficiently low level set of f, where $z_* \in L(f_\delta)$ and let $m(.)$ denote the Lebesgue measure.

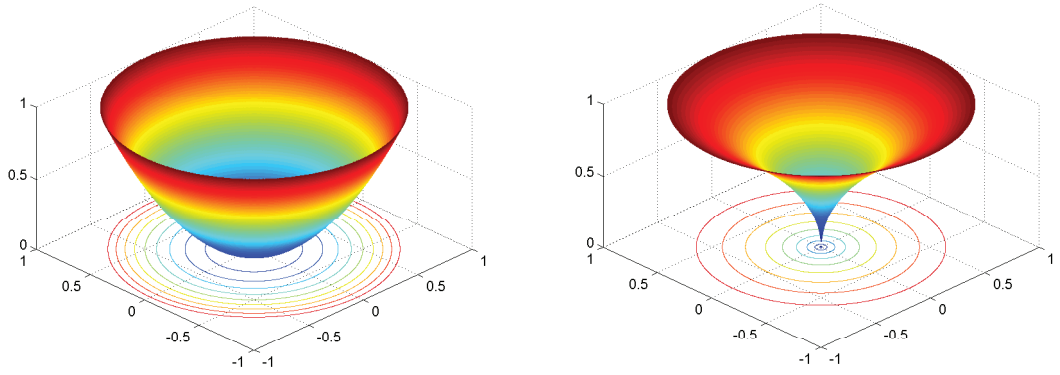


Figure 4.1: Quadratic bowl and cusp type essential local minimizers.

For a smooth quadratic bowl type z_*, f is approximately quadratic in all directions from z_* with

$$\|\lambda x - z_*\| \approx (f(\lambda x) - f_*)^{1/2} M(x), \tag{4.17}$$

where $\lambda > 0$ is sufficiently small and $M(x)$ is a positive, bounded scaling function for all $x \in \mathbb{R}^n$. Choosing $M \equiv 1$ gives the general quadratic and $M(x)$ extends to non-quadratic varieties. Consider increasing f_δ to $f_\delta + \epsilon$, where $\epsilon > 0$, along a ray emanating from z_*. Using the ratio $\|x_\delta + \epsilon - z_*\| / \|x_\delta - z_*\|$ and (4.17) the one dimensional scaling is obtained and raising to the n^{th} power, the scaling of Lebesgue measure, given by

$$m(L(f_\delta + \epsilon)) \approx \left(1 + \frac{\epsilon}{f_\delta - f_*}\right)^{n/2} m(L(f_\delta)).$$

Hence, the rate at which level sets grow from z_* follows a power law. The rate at which the measure of the level sets grow approximates the CDF of the expected distribution of objective function values in the neighborhood of z_*. Thus, the expected CDF of function values is given by

$$F \approx \left(\frac{f - f_*}{f_\gamma - f_*} \right)^{n/2}.$$

Whereas, for a nonsmooth cusp type z_*, we have

$$\|\lambda x - z_*\| \approx (f(\lambda x) - f_*)^2 M(x).$$

A similar argument to that above shows that

$$m(L(f_\delta + \epsilon)) \approx \left(1 + \frac{\epsilon}{f_\delta - f_*} \right)^{2n} m(L(f_\delta)).$$

Again the rate at which level sets of f grow follows a power law and the distribution of f values in the neighborhood of z_* is expected to be of the form

$$F \approx \left(\frac{f - f_*}{f_\gamma - f_*} \right)^{2n}.$$

Thus, if Assumption 48 holds, most essential local minimizers are expected to have $n/2 \leq \kappa \leq 2n$. If the user anticipates minimizers outside the range considered here, the analysis is easily updated by changing Assumption 48 (a). However, in what follows only this range is considered.

It should be noted that Assumption 48 is only required to determine the expected range of κ values. Minimizers failing Assumption 48 can also follow a power law within this range. Consider, for example, the function

$$f = |x_1| + x_2^2 \tag{4.18}$$

at the origin. This type minimizer fails Assumption 48 (b). However, the cumulative distribution of function values in $L(z_*, \zeta)$ (see (4.15)) does follow a power law. Furthermore, with (4.18) quadratic in all directions other than $x_2 = 0$ (where f is linear on a set of measure zero, that is, a set that is not sampled with probability one) a linear cumulative distribution of objective function values is expected, within the proposed range.

4.4 Fitting the Empirical Data

This section considers fitting a smooth curve to the empirical data Y. Firstly, the essential local minimum f_* is approximated with a variable $\hat{f}_*$. A method for choosing an optimal power κ^* for the empirical data is proposed in Section 4.4.2. This gives a the theoretical CDF for Y, of the form

$$F(f, \hat{f}_*, \kappa) = Pr(f_i \leq f) = \left(\frac{f - \hat{f}_*}{f_\gamma - \hat{f}_*} \right)^{\kappa^*} . \tag{4.19}$$

From (4.19), the probability of finding a lower function value $f_1 - \epsilon$ can be approximated, where $\epsilon > 0$.

4.4.1 Approximating f_*

Choosing a suitable $\hat{f}_*$ for (4.19) is crucial for determining tail probabilities, of primary interest here. Although the nature of an essential local minimizer is not known in advance, $n/2 \leq \kappa \leq 2n$ is expected. A general $\hat{f}_*$ for all essential local minimizers can be problematic as tail behavior varies a lot with respect to κ. Here a set of values is considered, rather than a single $\hat{f}_*$ value.

The set of possible $\hat{f}_*$ values is simply based on the range of observed objective function values,

$$\hat{f}_* \in \{f_1 - \mathcal{R}, f_1 - \mathcal{R}/2, f_1 - \mathcal{R}/4\}, \tag{4.20}$$

where $\mathcal{R} = \max\{f_\gamma - f_1, \epsilon/2\}$. With $\hat{f}_* < f_1$ for all $\hat{f}_*$ values, there exists a non-zero probability of finding a function value less than f_1. With (4.19) passing through $\hat{f}_*$, the set of values give a range of fits, one of which best fits the data Y.

The apparently crude set of $\hat{f}_*$ values given by (4.20) performs rather well in practice. The author considered various methods for approximating f_*, however, each method gave clustered $\hat{f}_*$ values at some iterations giving similar fits. Thus, a range of potential fits were not considered in general, whereas (4.20) consistently provided a range of possible fits relative to the data at each iteration.

4.4.2 Optimal Power Fit

Given an approximation $\hat{f}_*$ to the essential local minimum, a smooth curve is fitted through the empirical data $Y \cup \hat{f}_*$ to approximate the distribution of observed function values. An approximation to the distribution is obtained using the γ-sample empirical

distribution function (EDF). The EDF is the discrete distribution function which puts a probability mass of $1/\gamma$ at each sample point $f_i \in Y$, given by

$$\hat{F} = \frac{1}{\gamma} \sum_{i=1}^{\gamma} \mathcal{I}(f_i \leq f), \tag{4.21}$$

where $\mathcal{I}$ is an indicator function defined by

$$\mathcal{I}(f_i \leq f) = \begin{cases} 1 & \text{if } f_i \leq f \\ 0 & \text{otherwise.} \end{cases}$$

Asymptotically, as the number independent and identically distributed observations grows, (4.21) describes the true distribution function.

Theorem 49. *(Glivenko-Cantelli) If $f_1, \ldots, f_\gamma$ are independent and identically distributed random variables with cumulative distribution function F and empirical distribution function $\hat{F}$, then*

$$Pr\left(\lim_{\gamma \to \infty} \sup_t |\hat{F}(t) - F(t)| = 0\right) = 1.$$

The proof is omitted here but can be found in many elementary statistics textbooks. The interested reader is referred to [84] for details.

To obtain the smooth optimal power fit for the empirical data, minimizing various norm measures between (4.19) and (4.21) are considered, given by

$$\mathcal{P}(\hat{f}_*, \kappa) = \min_{\kappa} \|F(f, \hat{f}_*, \kappa) - \hat{F}(f)\|.$$

Here the 1, 2 and ∞ norm measures are used.

Firstly, the 1-norm measure between F and $\hat{F}$ is considered — the absolute area between the two curves. Noting that the EDF is piecewise continuous, the problem is formulated as a sum of integrals. An optimal κ is found by solving

$$\mathcal{P}_1(\hat{f}_*, \kappa) = \min_{\kappa} \sum_{i=0}^{\gamma-1} \int_{f_i}^{f_{i+1}} \left| \left(\frac{f - \hat{f}_*}{f_\gamma - \hat{f}_*} \right)^{\kappa} - \frac{i}{\gamma} \right| df, \tag{4.22}$$

where $f_i, f_{i+1} \in Y$ are the order statistics given by (4.13).

The second fit considered $\mathcal{P}_2$, is based on the 2-norm distance between F and $\hat{F}$ — the squared area between the curves. Once again the minimization problem is written as a sum of integrals, where the absolute value bars in (4.22) are replaced by $(.)^2$.

The final power fits use the infinity norm distance. Here two variations are considered. The infinity norm measures the maximum vertical displacement between F and $\hat{F}$ which occurs either just before, or just after a step in $\hat{F}$. Therefore, minimizing the infinity norm is equivalent to solving

$$\mathcal{P}_\infty(\hat{f}_*, \kappa) = \min_\kappa \left(\max_{i=1,\ldots,\gamma} \left(\max \left\{ F(f_i, \kappa) - \frac{i-1}{\gamma}, \frac{i}{\gamma} - F(f_i, \kappa) \right\} \right) \right). \tag{4.23}$$

Secondly, minimizing the sum of local infinity norm measures for each continuous part of $\hat{F}$ is considered, defined by

$$\mathcal{P}_4(\hat{f}_*, \kappa) = \min_\kappa \sum_{i=1}^{\gamma} \max \left\{ F(f_i, \kappa) - \frac{i-1}{\gamma}, \frac{i-1}{\gamma} - F(f_{i-1}, \kappa) \right\}.$$

4.4.2.1 Discussion

Numerical simulations performed by the author suggest each power fit method performs similarly in practice yielding similar values for κ. However, the computational effort required for each fit is not similar. Here the infinity norm power fit $\mathcal{P}_\infty$ is chosen because no integral evaluations are required, it is computationally inexpensive to calculate and it can be used as a goodness of fit measure (see Section 4.6). Furthermore, the following theorem shows the infinity norm minimization problem has a unique solution. For convenience let

$$g^+(\kappa) = \max_{i=1,\ldots,n} \left(\frac{i}{\gamma} - F(f_i, \kappa) \right)$$

and

$$g^-(\kappa) = \max_{i=1,\ldots,n} \left(F(f_i, \kappa) - \frac{i-1}{\gamma} \right),$$

denote the maximum vertical displacement between F and $\hat{F}$ above and below F, respectively. Furthermore, let i^* be the i which maximizes $\max\{g^+, g^-\}$ with objective function value f_{i^*}.

Theorem 50. *The minimization problem given by* (4.23) *has a unique solution for* $\kappa \geq 1$.

Proof: Differentiating F with respect to κ we obtain

$$F'(\kappa) = \ln \left(\frac{y - \hat{f}_*}{f_\gamma - \hat{f}_*} \right) \left(\frac{y - \hat{f}_*}{f_\gamma - \hat{f}_*} \right)^\kappa < 0$$

for all y such that $\hat{f}_* < y < f_\gamma$. Therefore, F is strictly monotonically decreasing with respect to κ such that $F(y,1) > F(y,\kappa)$ for all $\kappa > 1$. Hence, $g^+(\kappa)$ and $g^-(\kappa)$ are monotonically increasing and decreasing with respect to κ, respectively.

Firstly, consider $g^+(1) \geq g^-(1)$. Now $F(f_{i^*},1) > F(f_{i^*},\kappa)$ and so $g^+(1) < g^+(\kappa)$ for all $\kappa > 1$. Since $g^+(\kappa)$ is monotonically increasing and $g^+(1) \geq g^-(1)$, $\kappa = 1$ is a unique minimizer of (4.23).

Secondly, consider $g^+(1) < g^-(1)$. Since $g^+(\kappa)$ and $g^-(\kappa)$ are monotonically increasing and decreasing functions on the positive real line, respectively, there exits a unique intersection point (κ^*, g^*) such that $g^+(\kappa^*) = g^-(\kappa^*) = g^*$. Furthermore, $g^+(\kappa+\epsilon) > g^-(\kappa+\epsilon)$ for all $\epsilon > 0$, with $g^* < g^+(\kappa+\epsilon)$, and $g^-(\kappa-\epsilon) > g^+(\kappa-\epsilon)$ for all $\epsilon > 0$, with $g^* < g^-(\kappa-\epsilon)$. Thus, $\kappa = \kappa^*$ is the unique minimizer of (4.23). $\square$

4.5 Solving the ∞-norm Fit

To solve the minimization problem given by (4.23), the 1-dimensional optimization technique, Golden Section Search (GSS) [56] is used. This method is chosen because Theorem 50 ensures (4.23) has a unique solution and (4.23) is nonsmooth at the solution. Thus, the use of derivative based methods may be problematic. A precise statement of GSS is given in Figure (4.2).

GSS generates a sequence of nested brackets, each containing the desired solution κ^*. Each bracket is defined by three distinct points. Consider an interval $[a,b]$ containing κ^* and a point $\hat{a} \in (a,b)$, with

$$\mathcal{P}_\infty(a) > \mathcal{P}_\infty(b) > \mathcal{P}_\infty(\hat{a}). \tag{4.24}$$

$\mathcal{P}_\infty$ is continuous on the compact set $[a,b]$ which implies $\mathcal{P}_\infty$ is bounded and hence, $\mathcal{P}_\infty(x)$ has an infimum where $x \in [a,b]$. The continuity of $\mathcal{P}_\infty$ and compactness of $[a,b]$ ensures $\mathcal{P}_\infty$ achieves this infimum and thus, a minimum exists. Furthermore, from (4.24) the minimum is not at a or b. The interval $[a,b]$ is said to bracket κ^*.

Each nested bracket is obtained by considering two symmetric points $\hat{a}_k < \hat{b}_k \in (a_k, b_k)$. To shrink a bracket $\{a,\hat{a},b\}$, $\mathcal{P}_\infty$ is evaluated at a fourth point $\hat{b}$ where $a < \hat{a} < \hat{b} < b$. If $\mathcal{P}_\infty(\hat{a}) < \mathcal{P}_\infty(\hat{b})$, then $\{a,\hat{a},\hat{b}\}$ forms a bracket, otherwise $\{\hat{a},\hat{b},b\}$ forms a bracket. Placing these points at

$$\hat{a}_k = a_k + (1-\Xi)(b_k - a_k) \text{ and } \hat{b}_k = b_k - (1-\Xi)(b_k - a_k),$$

1. `Initialize: Compute initial bracket` $[a_0, b_0]$ `containing` κ^*. `Set` $k = 0$ `and choose` $\tau_b > 0$.

2. `Compute the pair of symmetric points,`

$$\hat{a}_k = a_k + (1 - \Xi)(b_k - a_k); \text{ and}$$

$$\hat{b}_k = b_k - (1 - \Xi)(b_k - a_k),$$

   `where` $\Xi = \frac{1}{2}(\sqrt{5} - 1)$.

3. `If` $\mathcal{P}_\infty(\hat{b}_k) \leq \min\{\mathcal{P}_\infty(\hat{a}_k), \mathcal{P}_\infty(b_k)\}$ `set`

$$a_{k+1} = \hat{a}_k \text{ and } b_{k+1} = b_k.$$

   `Otherwise set`

$$a_{k+1} = a_k \text{ and } b_{k+1} = \hat{b}_k.$$

4. `If` $b_{k+1} - a_{k+1} < \tau_b$, `set` $\kappa^* = \arg\min\{\mathcal{P}_\infty(\hat{a}_k), \mathcal{P}_\infty(\hat{b}_k)\}$ `and stop. Otherwise, increment` k `and goto Step (2).`

Figure 4.2: Golden Section Search algorithm

where $\Xi = \frac{1}{2}(\sqrt{5} - 1)$, ensures that either $\hat{a}_k = \hat{a}_{k+1}$ or $\hat{b}_k = \hat{b}_{k+1}$ holds. Thus, at each iteration three points with known function values are recycled and only one function evaluation is required to define the new nested bracket.

The algorithm terminates at iteration k when the minimal bracket size $b_k - a_k \leq \tau_b$ is reached. Hereafter $\tau_b = 0.001$ is chosen, giving κ^* to at least four significant figures.

4.5.1 Initial Bracket

The initial bracket is calculated from the expected range of κ values $[n/2, 2n]$, with $a_0 = 1$ and $b_0 = 2n$. Fixing $a_0 = 1$ for all n allows for a wide range of possible κ^* values, one of which best fits the data. Fixing $b_0 = 2n$ is crucial for the reliability of the stopping rule. In particular, allowing $\kappa^* > 2n$ can give a false indication of z_k being an essential local minimizer of f. Firstly consider $f_1 \ll f_2$. This suggests CARTopt has got *lucky*, generating a point with a significantly lower function value than other elements of Y. To yield an optimal fit an extremely large power is often required,

resulting in $F(f_1 - \epsilon)$ being extremely small. Thus, the probability of reducing f further is extremely small suggesting z_k is an essential local minimizer of f, even if only a single low function value has been located. Secondly, consider CARTopt generating a set $\{y \subset Y\}$ with exceedingly similar function values. This can result in a large jump in the empirical distribution function $\hat{F}$. Therefore, minimizing (4.23) reduces to minimizing the difference between F and $\hat{F}$ in the neighborhood of the large jump in $\hat{F}$. This can potentially give large powers and result in $F(f_1 - \epsilon)$ being extremely small, giving a false indication of f_1 being an essential local minimum of f.

Choosing the initial bracket as defined above may result in the interval $[1, 2n]$ not bracketing κ^*, suggesting a large power is required to fit $\hat{F}$. Thus, before the GSS algorithm is implemented, if

$$\mathcal{P}_\infty(2n - \lambda) > \mathcal{P}_\infty(2n),$$

where $0 < \lambda < \tau_b$, then $\kappa^* \geq 2n$ to within tolerance. In this case it is assumed the power is too large and κ^* is set to $2n$ (maximum value) and no optimization is required.

4.5.2 Choosing the Optimal κ^*

The GSS algorithm is applied using each value of $\hat{f}_*$ (defined in (4.20)), solving the minimization problem given by (4.23) three times producing the set,

$$\{\mathcal{P}_\infty^{(i)} = \mathcal{P}_\infty(\kappa^{(i)}, \hat{f}_*^{(i)}) : i = 1, 2, 3\},$$

where $\kappa^{(i)}$ denotes the power which minimizes (4.23) using $\hat{f}_*^{(i)}$. The optimal power fit is chosen as $\mathcal{P}_\infty^* = \min_i\{\mathcal{P}_\infty^{(i)}\}$ with corresponding κ^* and $\hat{f}_*^*$ values. Thus, the theoretical CDF is given by

$$F(f, \kappa^*, \hat{f}_*^*) = \left(\frac{f - \hat{f}_*^*}{f_\gamma - \hat{f}_*^*}\right)^{\kappa^*}, \tag{4.25}$$

from which, the probability of reducing f further is calculated. Note, with $\hat{f}_* < f_1$ this probability is non-zero. The reader is referred to Figure 4.3 for an example of an optimal power fit.

4.5.3 Computational Efficiency

The sequential stopping must be computationally cheap to evaluate so the overall efficiency of the CARTopt algorithm is not compromised. To this end, the optimization required in the fitting process must be computationally inexpensive. The construction of each bracket in GSS linearly shrinks the bracket range containing κ^* to zero at constant rate $\Xi = \frac{1}{2}(\sqrt{5}-1)$. Therefore, at the k^{th} iteration

$$b_k - a_k = \Xi^k(2n-1),$$

where $b_0 - a_0 = 2n-1$. To obtain the termination bracket range of 0.001,

$$k \geq \ln\left(0.001/(2n-1)\right) / \ln(\Xi)$$

iterations are required. Thus, for $n = 2, 10, 100$ the required number of iterations to locate κ^* to the desired accuracy are $k = 17, 21, 27$ respectively. Hence, the computational effort required to solve (4.23) is minimal and not adversely effected by the dimension of the objective function f.

4.6 Goodness of Fit

The theoretical optimal power fit (4.25) provides the best theoretical model for the empirical data Y. However, the best theoretical model does not necessarily fit the data well and a measure of goodness of fit is needed. If the fit is poor, probabilities calculated from (4.25) cannot be trusted and the rule fails to halt the algorithm.

Here, the Kolmogorov Smirnov (KS) test for continuous data is used to compare the EDF given by (4.21) to the theoretical CDF given by (4.25). The test is conducted under the null hypothesis that the two distributions are the same. The KS test statistic for continuous data is

$$\xi = \sup |F(f, \kappa^*, \hat{f}_*^*) - \hat{F}(f)|,$$

the largest deviation between the postulated and observed data. The test is conducted with respect to the critical value $\xi_{\eta,\gamma}$ of the distribution of the KS statistic, i.e. $Pr(\xi \geq \xi_{\eta,\gamma}) = \eta$, where η is the level of significance and γ is the number of sample points. The popular choice $\eta = 0.05$ is used. If $\xi \geq \xi_{\eta,\gamma}$, then the null hypothesis is rejected at the η level of significance.

Critical values $\xi_{\eta,\gamma}$ can be found in tables (see for example [90]) or approximated

by

$$\xi_{\eta,\gamma} \approx \sqrt{\frac{-\ln(\eta/2)}{2\gamma}} - \frac{0.16693}{\gamma}. \tag{4.26}$$

When dealing with small sample sizes (say $\gamma \leq 25$) the effectiveness of the KS test can be increased using a correction factor [90]. However, for $\gamma = 2N = 40$ (sample size used here) the two methods are indistinguishable.

This test is chosen because is it extremely cheap to calculate. The KS statistic is given by $\xi = \mathcal{P}^*_\infty(\kappa^*, \hat{f}^*_*)$, requiring no additional calculations and $\xi_{\eta,\gamma} < 1$ is constant. Thus, the goodness of fit is essentially free.

A good fit suggests the distribution of Y follows a power law and probabilities are determined from (4.25) with confidence. An example of a good fit is shown in Figure 4.3. Whereas, a poor fit indicates one of two things. Either, CARTopt is not sampling in a sufficiently small neighborhood of an essential local minimizer, or the distribution of function values near the essential local minimizer does not follow a power law. In either case, more random sampling is required in the neighborhood of z_k to obtain more information about f.

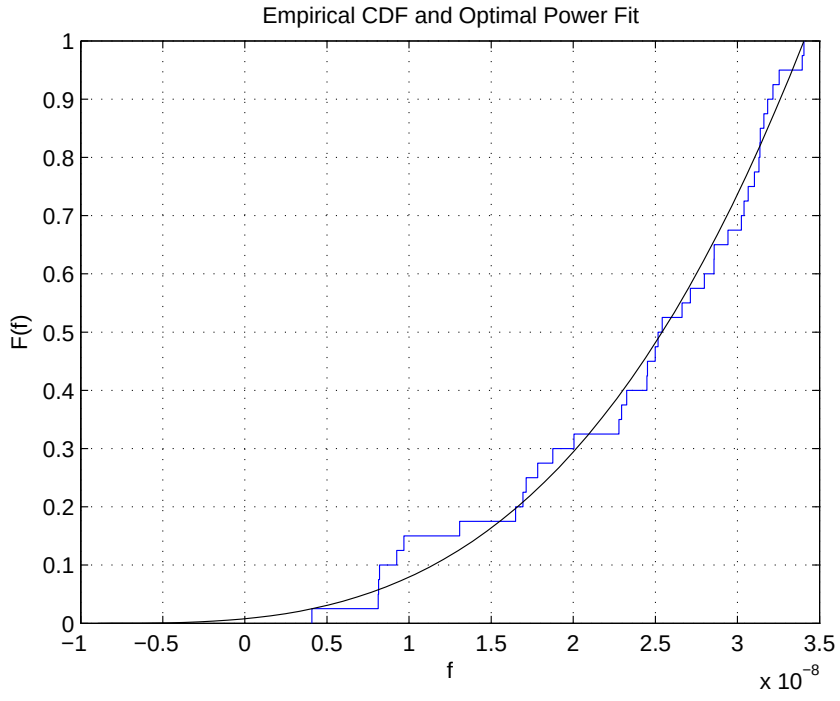


Figure 4.3: Empirical distribution function (blue staircase) and optimal power fit (smooth curve). This example illustrates an optimal power fit which terminated the CARTopt algorithm on the non-smooth Rosenbrock function (3.22), where $f_* = 0$. The fit is good and the probability of reducing f below $f_1 - \epsilon$ is sufficiently small for $\epsilon =$ 1e-8.

4.7 Deterministic Instances of CARTopt

Convergence results for the deterministic instances of the CARTopt method have not been derived. Therefore, convergence to an essential local minimizer of f cannot be guaranteed. The only difference between the stochastic and deterministic methods is how sample points are generated. Thus, the deterministic instances of CARTopt also generate a non-empty set of sample points with the least function values and thus, the rule proposed in the next section can also be applied.

4.8 Stopping Rule for the CARTopt Method

The sequential stopping rule for the CARTopt method is now presented. The rule is implemented at Step 3(g) in the main loop of the CARTopt algorithm (see Figure 3.1) when T_k is full size i.e. $k > \max\{2, 2(n-1)\}$. Furthermore, if f is a Barrier function, the rule requires T_k to be full size with at least $2N$ finite function values.

Definition 51. (Stopping rule). *Terminate the CARTopt algorithm at iteration k if T_k is full size with $|\{x \in T_k : f(x) \neq \infty\}| \geq 2N$ and one of the following conditions holds:*

(a) given an $\epsilon > 0$ and $0 < \beta < \frac{1}{\gamma}$, if $\mathcal{P}^*_\infty(\hat{f}^*_*, \kappa^*) < \xi_{\eta,\gamma}$ **and** *$F(f_1 - \epsilon, \hat{f}^*_*, \kappa^*) < \beta$, or*

(b) $k > k_{\max}$.

Here $\xi_{\eta,\gamma}$ is calculated from (4.26) and $k_{\max} \gg 1$ is a finite resource quantity chosen by the user. The default value is $k_{\max} = \max\{1000, 100n^2\}$. Thus, if Assumption 46 does not hold (the sequence of iterates $\{z_k\}$ is unbounded) the CARTopt method terminates.

Proposition 52. *The proposed stopping rule halts the CARTopt algorithm at iteration $k \leq k_{\max}$.*

Proof. From Theorem 32, the main loop of the CARTopt (see Figure 3.1, Step 3) is a finite process. This loop is implemented indefinitely until stopping conditions are satisfied. Thus, the algorithm halts at iteration $k \leq k_{\max}$, if not before. □

In summary, CARTopt is terminated at iteration k if the distribution of objective function values in a neighborhood of $f(z_k)$, given by Y, follows a power law and the probability of finding a point with function value $f_1 - \epsilon$ is sufficiently small, or if user

resource limits are reached. If CARTopt halts before $k_{\max}$ is reached, (4.25) asserts that the probability of reducing $f(z_k)$ further is small and $f(z_k)$ is an estimate of an essential local minimum of f. The reader is referred to Figure 4.3 for an example of a optimal power fit which halted CARTopt. Numerical simulations performed by the author found that choosing ϵ relative to the required decimal place accuracy and setting β = 1e-6 performed well in practice. Choosing ϵ = 1e-8, for example, gave approximately eight decimal accuracy on the nonsmooth problems considered.

Although this rule has been developed primarily for the CARTopt method, it can be applied to many random search algorithms. The only requirement is that a set of non-repeating, $\gamma > 0$ least function values are retained by the algorithm. With $\gamma = 40$, minimal additional storage is required to implement the rule on a different algorithm.

Chapter 5

Hooke and Jeeves / CARTopt Hybrid Method

This chapter introduces a new algorithm which finds a local solution to the unconstrained minimization problem given by (1.1). Here, the objective function $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ is assumed to be nonsmooth and may be discontinuous. The method conforms to the framework described in Section 1.7, where a series of local and localized global optimization phases are applied. The local phase of this algorithm is a modified Hooke and Jeeves method, which builds upon ideas described in Section 1.4 on trajectory following optimization. The localized global phase performs Pure Random Search on a subset of $\mathbb{R}^n$. This subset is defined and sampled using the bound constrained CARTopt algorithm. Choosing the CARTopt algorithm means the convergence results developed in Chapter 3 can be used to establish convergence on nonsmooth problems for this hybrid algorithm.

Firstly, the new algorithm's structure is introduced, where sections 5.1 – 5.4 detail the local phase and Section 5.5 describes the localized global phase. A method for generating a new search grid is proposed in Section 5.6. The chapter concludes with convergence results for both smooth and nonsmooth objective functions.

5.1 The Algorithm

The algorithm proposed here is a variant of the classical direct search algorithm of Hooke and Jeeves [35] (see Section 1.3.2.1). The new algorithm differs from the classical version in three major ways. Firstly, the algorithm allows for both grid perturbations and rotations as opposed to operating on a fixed grid. Secondly, the sequence of

objective function values $\{f(x_k)\}_{k=1}^{\infty}$ generated by the algorithm is not necessarily decreasing. Potential uphill steps are taken so the sequence of iterates $\{x_k\}$ approximates the path of a second order trajectory following method (see Section 1.4). Finally, the algorithm conducts a series of localized global optimization phases in a neighborhood of x_k before the reducing the grid mesh size. It is the localized global optimization phases that give convergence on nonsmooth optimization problems, by exploiting the connection between local nonsmooth optimization and global optimization [61] (described in Section 1.6).

5.1.1 Grid Structure

The local phase of the algorithm proposed here reduces f by searching on a succession of finer grids, indexed by m. Each grid $\mathcal{G}_m$ is defined by a mesh size $h_m > 0$, a grid center $\mathcal{O}_m$ (on the grid) and a set of n vectors $\mathcal{V}_m$ spanning $\mathbb{R}^n$, where

$$\mathcal{V}_m = \left\{ \nu_m^{(i)} = H_m e_i : i = 1, \ldots, n \right\}.$$

Here H_m is a Householder matrix and e_i is the i^{th} column of the identity matrix. Points on the grid $\mathcal{G}_m$ are given by

$$\mathcal{G}_m = \left\{ x \in \mathbb{R}^n : x = \mathcal{O}_m + h_m \sum_{i=1}^{n} \lambda_i \nu_m^{(i)} \right\},$$

where each λ_i is an integer. The vectors in $\mathcal{V}_m$ are parallel to the axes of the grid and steps between adjacent grid points are given by the vectors $h_m \nu_m^{(1)}, \ldots, h_m \nu_m^{(n)}$. The sequence of centers $\mathcal{O}_m$ means each grid can be offset relative to one another and the sequence of matrices H_m means each grid can be rotated relative to one another. Hence, the succession of finer grids is not necessarily nested. In the classical algorithm of Hooke and Jeeves, $\mathcal{O}$ is fixed and H is the identity matrix for all iterations.

The set of vectors

$$\mathcal{V}^+ = \{\mathcal{V}_m, -\mathcal{V}_m\} \tag{5.1}$$

form a positive basis for $\mathbb{R}^n$ [85] and hence, any vector in $\mathbb{R}^n$ can be written as a non-negative linear combination of vectors in $\mathcal{V}^+$. Using (5.1) a grid local minimizer is defined.

Definition 53. (Grid Local Minimizer). *A point $x \in \mathcal{G}(\mathcal{O}, h, \mathcal{V})$ is a grid local*

minimizer of an objective function f with respect to a positive basis $\mathcal{V}^+$ if

$$f(x + h\nu) \geq f(x) \text{ for all } \nu \in \mathcal{V}^+.$$

5.1.2 Algorithm Details

A precise statement of the main algorithm is given in Figure 5.1. The algorithm consists of an initialization phase and two nested loops. An exploratory phase (Step 3(a)) and a localized global phase (Step 3(e)) are listed as separate subroutines in sections 5.2 and 5.5 respectively. The dependence on $\mathcal{O}_m$ is suppressed in the following as $\mathcal{O}_m$ is the origin of $\mathcal{G}_m$. The notation $f_k = f(x_k)$ has been used for convenience.

The initialization phase sets the iteration counters k and m to zero, the Householder matrix H_0 is set as the identity matrix, a lower bound $h_{\min} > 0$ on the mesh size and v_0 to the zero vector. The user chooses an initial mesh size $h_0 > h_{\min}$ and an initial point $x_0 \in \mathbb{R}^n$, which is also set to the grid center $\mathcal{O}_0$. The initial velocity vector v_0 does not necessarily need to be set to zero. Setting a non-zero v_0 gives an initial *velocity* to the system, which may produce a different trajectory exploring different regions of $\mathbb{R}^n$ and possibly locating different minima.

The inner loop (steps 3(a) - 3(d), indexed by k) searches on the grid $\mathcal{G}_m$ until a grid local minimizer is found. Step 3(d) implements a Hooke and Jeeves automatic restart, removing unsuccessful pattern moves by setting $v_{k+1} = \mathbf{0}$. This forces an exploratory phase from $x_{k+1} = x_k$, a successful iterate. The outer loop (steps 3(a) - 3(f), indexed by m) defines the new grid $\mathcal{G}_{m+1}$ for the inner loop to search over. The grid is aligned with a promising direction of descent, $x_{k+1} - x_k$, using a Householder matrix H_k, where x_{k+1} is generated in Step 3(e). This alignment potentially reduces the number of grid local minimizers and can increase the computational efficiency of the method. Each component of Step 3 is discussed in the sections which follow.

The algorithm terminates when either $h_m \leq h_{\min}$ or if the stopping conditions of CARTopt are satisfied. Once terminated, z_m is the candidate solution for an essential local minimizer of f.

5.2 Exploratory Phase

The exploratory phase considered here is a slightly modified version of the classical Hooke and Jeeves exploratory phase [35]. A precise statement of the subroutine is given in Figure 5.2. At each execution, f is evaluated at a finite number of points

1. Initialize: Set $k = m = 0$, $v_0 = \mathbf{0}$, H_0 to the identity matrix and $h_{\min} > 0$. Choose $x_0 = \mathcal{O}_0 \in \mathbb{R}^n$ and $h^{(0)} > h_{\min}$.

2. Evaluate the objective function f_0, and set $\mathcal{U}_0 = f_0$.

3. **while** stopping conditions do not hold, **do**

   (a) Exploratory phase: Calculate the exploratory vector E_k from $x_k + v_k$.

   (b) Sinking lid: **If** $f(x_k + v_k + E_k) \geq f_k$ and $\mathcal{U}_k \neq f_k$, then choose $\mathcal{U}_{k+1} \in [f_k, \mathcal{U}_k)$, **else** $\mathcal{U}_{k+1} = \mathcal{U}_k$.

   (c) Pattern move: **If** $f(x_k + v_k + E_k) < \mathcal{U}_{k+1}$ then set

   $$x_{k+1} = x_k + v_k + E_k \quad \text{and} \quad v_{k+1} = v_k + E_k,$$

   increment k and goto (a).

   (d) **If** $v_k \neq \mathbf{0}$ set $v_{k+1} = \mathbf{0}$, and goto (a).

   (e) Localized global optimization phase: Set $z_m = x_k$. Choose $\Omega_m \subset \mathbb{R}^n$ such that $z_m \in \Omega_m$ and $m(\Omega_m) > 0$. Execute CARTopt in Ω_m until a lower point x_{k+1} is found or stopping conditions are satisfied.

   (f) Choose h_{m+1}, form H_{m+1} and set $\mathcal{O}_{m+1} = x_{k+1}$. Increment k and m and goto (a).

end

Figure 5.1: The main algorithm of the Hooke and Jeeves / CARTopt hybrid method

to define the exploratory vector E_k. This requires at least n, but no more than $2n$ function evaluations. In what follows, the m subscripts have been omitted on h and H as they remain unchanged throughout this subroutine.

Each exploratory phase is conducted from $z = x_k + v_k$. While operating on the grid $\mathcal{G}_m$, the following method is used to potentially reduce the number of function evaluations. If the decision variable z_i was successfully decremented during the previous exploratory phase, z_i is decremented first in the current exploratory. The classical exploratory phase increments each decision variable first, with no reference to previous iterations. Including this additional heuristic potentially saves n function evaluations at each phase, simply by exploring the promising direction defined by the trajectory first. To indicate which directions to explore first a diagonal matrix $\mathcal{I}_E$ is used, where the non-zero elements are given by

$$\mathcal{I}_E(i,i) = \text{sign}(HE_{k-1}(i)), \quad (k \geq 1)$$

where $E_0 = \mathbf{0}$ and

$$\text{sign}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{otherwise.} \end{cases}$$

The notation $\mathcal{I}_E(:,i)$ and $E_k(i)$ is used to denote the i^{th} column vector of $\mathcal{I}_E$ and i^{th} element of E_k respectively. When a new grid $\mathcal{G}_{m+1}$ is formed and for the special case $k = 0$, no promising direction exists and $\mathcal{I}_E$ is set as the identity matrix.

5.3 Sinking Lid

The sinking lid (Step 3(b)) is a strategy used to ensure the trajectory does not oscillate or cycle endlessly if the sequence $\{x_k\}$ is bounded. This behavior can occur when multiple uphill steps are taken. Recall that the sequence $\{x_k\}$ approximates a particle moving in conservative force field (see Section 1.4.1) and without damping can be in continual motion. The sinking lid systematically reduces an upper bound $\mathcal{U}$ on allowable function values each time ascent is made, creating a sequence of non-increasing upper bounds $\{\mathcal{U}_k\}$. Thus, at iteration k the trajectory cannot surmount ridges where $f(x) > \mathcal{U}_k$. Provided $\{x_k\}$ bounded, the lid eventually traps the particle in the neighborhood of an essential local minimizer, or at least terminates the trajectory in reasonable time.

Initially $\mathcal{U}_0 = f_0$ and thus, f cannot be increased above the initial function value.

1. `Initialize: Set` $E_k = \mathbf{0}$, $z = x_k + v_k$, `and` $f_E = f(z)$.

2. **For** $i = 1, \ldots, n$ **do**

   (a) `Set` $z = z + hH\mathcal{I}_E(:, i)$. **If** $f(z) < f_E$, **then set**

   $$f_E = f(z) \text{ and } E_k = E_k + hH\mathcal{I}_E(:, i).$$

   `Increment` i `and goto` **end**.

   (b) `Set` $z = z - 2hH\mathcal{I}_E(:, i)$. **If** $f(z) < f_E$, **then set**

   $$f_E = f(z) \text{ and } E_k = E_k - hH\mathcal{I}_E(:, i).$$

   `Increment` i `and goto` **end**.

   (c) `Set` $z = z + hH\mathcal{I}_E(:, i)$ `and increment` i.

   end

3. `Return to main algorithm with exploratory vector` E_k.

Figure 5.2: Modified exploratory phase of the Hooke and Jeeves algorithm

If descent is made ($f(x_k + v_k + E_k) < f_k$) the upper bound remains unchanged with

$$\mathcal{U}_{k+1} = \mathcal{U}_k. \tag{5.2}$$

Otherwise, $f(x_k + v_k + E_k) \geq f_k$ and the upper bound is systematically reduced as follows. If $f(x_k + v_k + E_k) < \mathcal{U}_k$ and $\|v_k\| \neq 0$

$$\mathcal{U}_{k+1} = (\mathcal{U}_k + f(x_k + v_k + E_k))/2, \tag{5.3}$$

else

$$\mathcal{U}_{k+1} = f_k. \tag{5.4}$$

Equation (5.3) sets the new upper bound halfway between the previous upper bound and the objective evaluated at the point of ascent. Whereas, (5.4) sets the upper bound to the previous successful iterate if the objective is increased above the current upper bound. Thus, the pattern move is rejected in Step 3(c) if (5.4) is satisfied.

The trajectory produced using this method can leave locally optimal points found

along the path. However, the primary interest here is in local optimization and hence locating any optimal point is sufficient. Setting $\mathcal{U}_{k+1} = f_k$ for all k gives a strictly descent version of the modified Hooke and Jeeves algorithm.

5.4 Pattern Move

The Hooke and Jeeves pattern move may be expressed by the pair of equations

$$x_{k+1} = x_k + v_k + E_k, \quad \text{and} \quad v_{k+1} = \theta(v_k + E_k), \tag{5.5}$$

where E_k is the exploratory vector, $v_0 = \mathbf{0}$ and θ is a positive integer. The pattern move used here differs from the classical move in two ways. Firstly, by choosing $\theta > 1$ a more aggressive pattern move is achieved, allowing the algorithm to transverse large distances in $\mathbb{R}^n$ in less iterations. Secondly, uphill steps are accepted provided f is not increased above the current upper bound $\mathcal{U}_k$. This allows f to temporarily increase along the trajectory approximating the path of a second order ODE trajectory (see Section 1.4). The relative merits of such an approach are illustrated in the following example.

5.4.1 Uphill Steps

Potential benefits of allowing uphill steps in a trajectory produced using the inner loop of the main algorithm with $\theta = 1$ in (5.5) is demonstrated. The problem considered here is a nonsmooth version of Rosenbrock's function, given by

$$f(x) = |10(x_2 - x_1^2)| + |1 - x_1|, \tag{5.6}$$

starting from $x_0 = [-0.5, 1.3]$, with $x_* = [1, 1]$ the unique essential local minimizer. This nonsmooth version of Rosenbrock's function shares all the difficulties of its smooth counterpart with an added sharp valley floor where f is non-differentiable. The reader is referred to Figure 5.3 where a contour plot of (5.6) is shown.

The benefits of allowing uphill steps are clear from Figure 5.3. Initially, both the classical and modified Hooke and Jeeves methods share the same trajectory. However, upon first contact with the nonsmooth valley floor the classical algorithm terminates at x_*^{HJ}. The classical algorithm then experiences a period of stagnation, requiring a series of mesh reductions before any progress is made toward x_*. Whereas, the modified algorithm *rolls* over the nonsmooth valley floor multiple times making progress toward

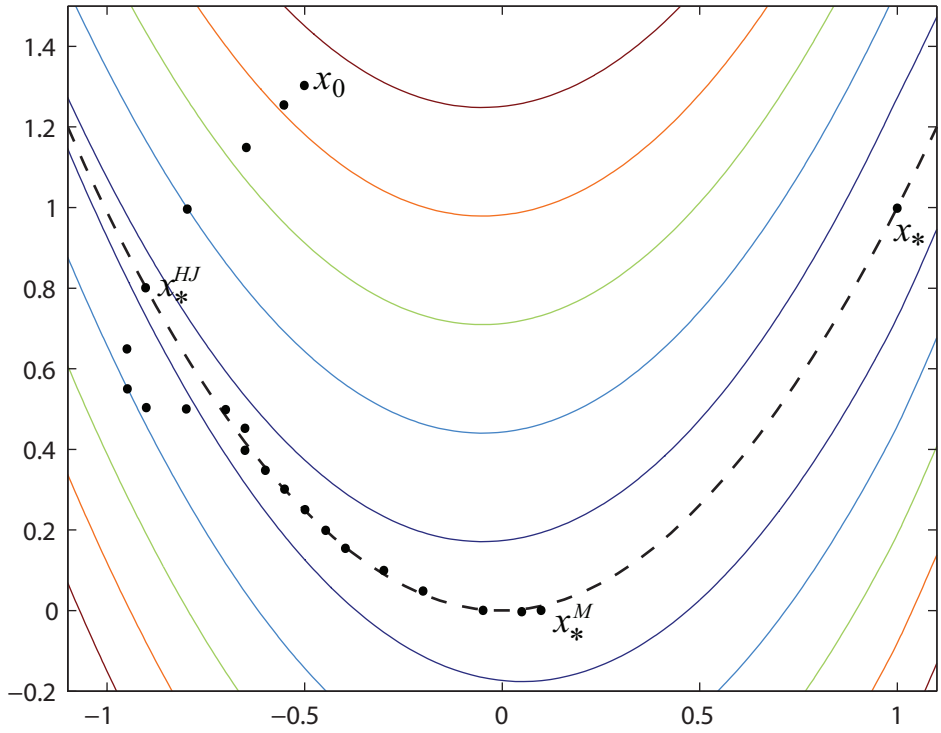


Figure 5.3: Trajectory generated by the inner loop of the main algorithm (dotted curve) for minimizing (5.6), terminating at x_*^M. The classical algorithm terminates at x_*^{HJ}. The dashed curve indicates the nonsmooth valley floor.

x_* while operating on the same grid. Although the trajectory terminates at x_*^M, still some distance from x_*, much more progress is made. Therefore, the first localized global optimization phase is performed closer to x_*, exploiting the *fast* local search component of the algorithmic framework better. In addition, Figure 5.3 illustrates the potential for this algorithm to escape local nonsmooth wedges in f, potentially reducing the risk of stagnation, when f is largely smooth.

5.5 Localized Global Phase

At each iteration of the localized global phase (Step 3(e)) the random search algorithm CARTopt is employed to either find a point of descent, or confirm that z_m is an essential local minimizer of f. The interested reader is referred to the three previous chapters for details on this optimization method, particulary Chapter 3.

Each localized global optimization phase is conducted in a sequence of hypercube

search regions $\Omega_m \subset \mathbb{R}^n$ centered on z_m, defined by

$$\Omega_m = \{x \in \mathbb{R}^n : \|H_\Omega x - z_m\|_\infty \leq \rho\}, \tag{5.7}$$

where

$$\rho = \max\{\sqrt{n}h_m, h_\Omega\} \tag{5.8}$$

is the hypercube radius and $h_\Omega > 0$ is a minimum radius imposed on Ω. The minimum radius ensures $m(\Omega)$ is bounded away from zero for all iterations, which is crucial for convergence on nonsmooth problems. If the objective function is known to be smooth, then $\rho = \sqrt{n}h_m$ is used. This allows $m(\Omega)$ to tend to zero as h tends to zero, potentially increasing the rate of convergence on smooth problems. The Householder matrix H_Ω in (5.7) is used to rotate the hypercube optimization region (see Figure 5.4), where

$$H_\Omega = I - \frac{2}{\|e_1 - w\|^2}(e_1 - w)(e_1 - w)^\mathsf{T}$$

and

$$w = \frac{1}{\sqrt{3+n}} H_m[2, 1, \ldots, 1]^\mathsf{T}. \tag{5.9}$$

This apparently strange transformation is used to remove collinear points from the input training data set that CARTopt uses, details of which follow in the next subsection.

The localized global phase can terminate in two ways. Firstly, if a point $x \in \Omega_m$ is found such that $f(x) < f(z_m)$, then x is set to the next iterate x_{k+1} and a new local phase begins. Secondly, if the stopping conditions of CARTopt are satisfied. The latter confirming z_m as an essential local minimizer of f, terminating the whole algorithm.

5.5.1 Recycling Points

Most of the computational effort in the approach to nonsmooth optimization proposed here is performed during the localized global phase. Therefore, to increase the numerical performance of CARTopt, points with known function values are reused where appropriate. This gives CARTopt an initial training data set from which a CART partition on Ω_m is formed.

Each localized global phase is conducted in the neighborhood of x_k, where an exploratory phase failed. Thus, there are at least $2n + 1$ points with known function values in the neighborhood of x_k, defined by the set

$$X_E = \{x_k, x_k + h_m\nu_m : \nu_m \in \mathcal{V}_m^+\}.$$

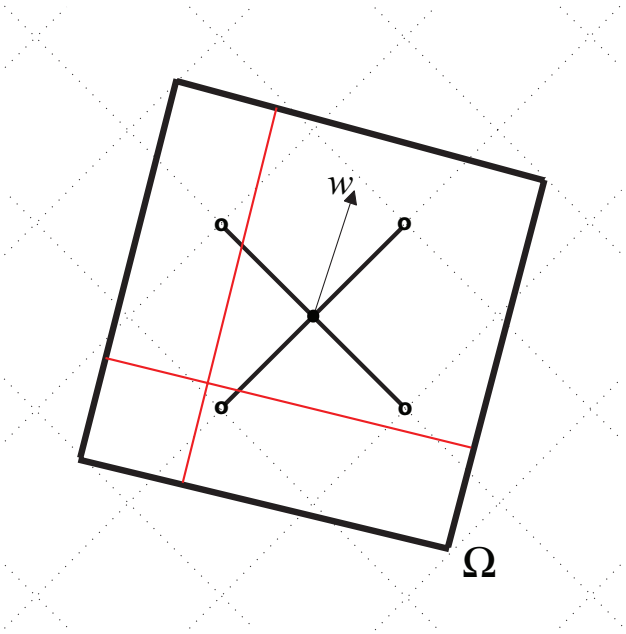


Figure 5.4: The localized global optimization region Ω is shown in bold. The dotted lines define the grid upon which the Hooke and Jeeves Exploratory phase failed, shown as the bold cross. Note, there does not exist two points from the cross which are collinear and parallel to boundaries of Ω. Two potential splitting hyperplanes are shown in red.

Noting that $\|x - x_k\| \leq h_m$ for all $x \in X_E$ and Householder transformations preserve the lengths of vectors,

$$\|H_\Omega x\|_\infty \leq h_m \leq \sqrt{n} h_m \text{ for all } x \in X_E.$$

Thus, $X_E \subset \Omega_m$ for all m.

Collinear points parallel to a boundary face of Ω_m can be problematic during a partition phase, particulary if they are in different categories — a point from $\{\omega_L\}$ and $\{\omega_H\}$. Recall that all potential splitting hyperplanes are orthogonal to boundary faces of Ω_m and between elements of different categories. Thus, there exists a potential hyperplane orientation for which no split exists. However, using H_Ω to define Ω_m ensures all collinear points in X_E are not parallel to any boundary face of Ω_m, see Figure 5.4. The direction vector w defined by (5.9) is chosen so that the minimum distance between potential splitting points in X_E is bounded above zero, see Figure 5.4.

In addition, if $\Omega_{m-1} \cap \Omega_m \neq \emptyset$ $(m \geq 2)$ there may be many points with known function values in Ω_m from previous localized global phases. Let T_m^* denote the terminating training data set from the m^{th} $(m \geq 1)$ localized global phase with $T_0^* = \emptyset$, then the input training data set is defined by

$$T_m = \{x \in T_{m-1}^* \cap \Omega_m\} \cup X_E. \tag{5.10}$$

Reusing or *recycling* these points can greatly increase the efficiency of CARTopt as a full size training data set may be known. With the input training data satisfying

$$2n + 1 \leq |T_m| \leq 2(n-1)N + 2n + 1,$$

where N is batch size used in CARTopt, less (or no) function evaluations are required to form the initial partition on Ω_m. Therefore, promising subsets of Ω_m can be sampled directly, without having to randomly sample Ω_m first.

5.6 Generating the Next Grid

The outer loop (steps 3(a) - 3(f) indexed by m) of the main algorithm generates a new grid for the inner loop to search over. In this section it is assumed that Step 3(e) is a finite process and a point of descent x_{k+1} is located. Therefore, Step 3(f) is implemented and a new grid is generated. Rather than simply having all grids nested, as in the classical algorithm, each grid is perturbed, rotated and scaled. Each is discussed in the subsections which follow.

5.6.1 Perturbation

The grid perturbation is straight forward, setting the grid center $\mathcal{O}_{m+1} = x_{k+1}$, the newly generated point of descent from the localized global phase. Since x_{k+1} is generated from randomly sampling subsets of Ω_m, $\mathcal{O}_{m+1} \notin \mathcal{G}_{m+1}$ $(m(\{x \in \mathcal{G}_m\}) = 0)$ and so $\mathcal{G}_{m+1}$ is offset relative to $\mathcal{G}_m$ with probability one. Hence, the modified Hooke and Jeeves algorithm is not confined to searching an admissible set of points.

5.6.2 Rotation

Each time a grid local minimizer is located, the computationally expensive localized global phase is initiated. Therefore, if transforming the grid has the potential to reduce

the number of grid local minimizers, the number of localized global phases can also be reduced. The interested reader is referred to [11] where the number of grid local minimizers for a strictly convex quadratic function in two dimensions is considered as the grid is transformed. In [11] it is shown that conjugate grids greatly reduce the number of grid local minimizers on such functions. Although these ideas could have potential here, a new approach is developed which potentially yields the same desired result.

The iterate x_{k+1} not only gives a point of descent, but also a promising direction of descent, given by $d = (x_{k+1} - x_k)/\|x_{k+1} - x_k\|$. Using the Householder transformation,

$$H_m = I - 2uu^{\mathsf{T}}, \tag{5.11}$$

$$u = (e_1 - d)/\|e_1 - d\|,$$

the x_1 axis of the grid $\mathcal{G}_{m+1}$ is set parallel to d, i.e. $H_m e_1 = d$. Such a transformation can dramatically reduce the number of grid local minimizers and allow the modified Hooke and Jeeves component of the algorithm to make more progress.

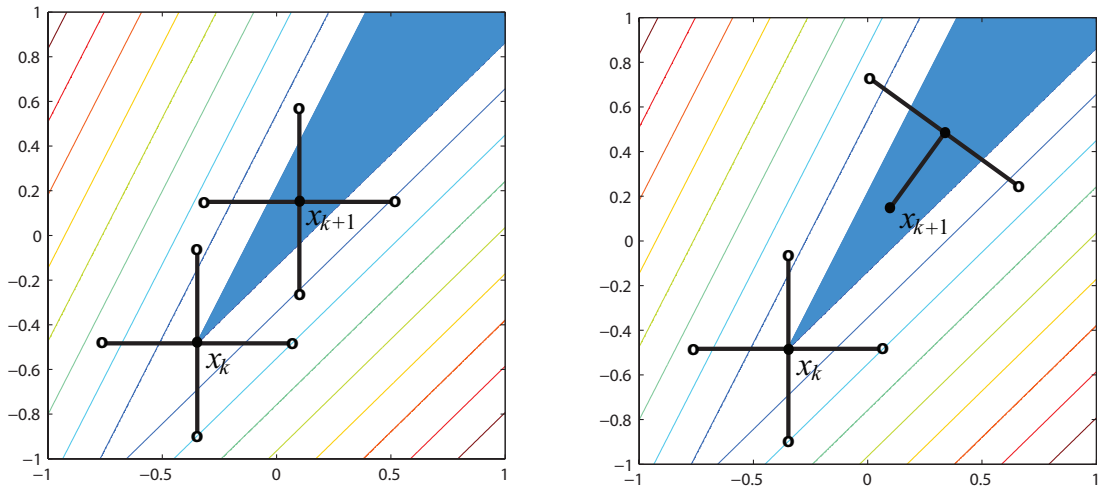


Figure 5.5: Grid perturbation with no rotation and grid perturbation with rotation. Points of descent and ascent are denoted •, o respectively and the shaded region indicates points of descent from x_k.

To illustrate the relative merits of such a transformation consider, for example, the function

$$f = \max\{a^{\mathsf{T}}x, b^{\mathsf{T}}x\} \text{ such that } x \in \mathbb{R}^2, \tag{5.12}$$

where $a = [0.7, -0.7]$ and $b = [-0.9, 0.5]$. The reader is referred to Figure 5.5 where

the contours of (5.12) are shown. With x_k a grid local minimizer of f, a localized global optimization phase is performed in the neighborhood of x_k, locating a point of descent x_{k+1}. If only grid perturbations are performed, x_{k+1} remains a grid local minimizer of f and another localized global phase is conducted. This can repeat for many iterations, making this method computationally expensive. However, x_{k+1} is no longer a grid local minimizer if the grid is transformed using (5.11). The modified Hooke and Jeeves exploratory moves have succeeded in locating a point of descent and a new trajectory is produced. Although only a heuristic, the grid transformation can dramatically increase the numerical performance of the algorithm, executing less localized global phases. This was observed during numerical simulations performed by the author.

5.6.3 Scaling

The mesh scaling heuristic used here is made with respect to the iterate x_{k+1} generated in the localized global phase. Firstly, consider the case when f is assumed to be nonsmooth. If

$$\|x_{k+1} - x_k\|_2 \geq h_m, \tag{5.13}$$

then $h_{m+1} = h_m$, otherwise

$$h_{m+1} = \max\{h_m/\tau_h, \|x_{k+1} - x_k\|\}, \tag{5.14}$$

where τ_h is a positive mesh reduction coefficient. Hereafter $\tau_h = 3$ is chosen, although any finite positive constant can be used. If (5.13) is satisfied it suggests there is still potential in operating with the current mesh size and no reduction is made. This may result in h_m not tending to zero as m tends to infinity, but convergence is still obtained through CARTopt with probability one. Otherwise the mesh is reduced using (5.14). The term h_m/τ_h is used as a bound on mesh reduction per iteration to prevent an unjustifiably small value for h_{m+1}, which may cause the algorithm to terminate early.

If f is known to be smooth, then the mesh size is updated using

$$h_{m+1} = \max\left\{\frac{h_m}{\tau_h}, \frac{\|x_{k+1} - x_k\|}{n + \epsilon}\right\}, \tag{5.15}$$

where ϵ is a small positive constant. This choice of h ensures a mesh reduction after each successful global optimization phase, a property needed for convergence on smooth problems.

Proposition 54. *If f is assumed to be smooth and the sequence of grid local minimizers $\{z_m\}$ is infinite, then $h_m \to 0$ as $m \to \infty$ for all $\tau_h > 1$.*

Proof. Clearly if $h_{m+1} = h_m/\tau_h$, the mesh size h is reduced for all $\tau_h > 1$.

For the second term in the maximization, the largest possible h value is obtained when $\|x_{k+1} - x_k\|$ is maximized. From the definition of Ω_m (5.7) and noting that Householder transformations preserve the length of vectors, this distance is maximized at the corners of Ω_m. With f assumed to be smooth, the radius of Ω_m is $\sqrt{n}h_m$ (see (5.8)) for all k and hence,

$$\max \|x_{k+1} - x_k\| = \sqrt{n(\sqrt{n}h_m)^2} = nh_m. \tag{5.16}$$

Substituting (5.16) into the second term of (5.15) we have

$$h_{m+1} = \frac{n}{n+\epsilon} h_m.$$

Therefore, both terms in the maximization reduce h at each pass and with $\{z_m\}$ an infinite sequence the result follows. □

5.7 Convergence

In this section it is assumed that the stopping rules for both CARTopt and the modified Hooke and Jeeves algorithm are never invoked. This allows us to examine the asymptotic properties of the full sequence of iterates generated by the algorithm. The stopping conditions are included in the algorithms from a practical point of view.

5.7.1 Smooth Results

Firstly, let us consider the case when the objective function is known to be smooth. These results is of interest here because the smooth version of the algorithm potentially converges faster on smooth problems than the nonsmooth version would. In order to establish the smooth convergence results the following assumptions are required.

Assumption 55. *The following conditions hold:*

(a) The points at which f is evaluated at lie in a compact subset of $\mathbb{R}^n$; and

(b) The sequence of function values $\{f(x_k)\}$ is bounded below.

These assumptions ensure the existence of cluster points in $\{z_m\}$ and excludes the case where $f(x_k) \to -\infty$ as $k \to \infty$.

The next theorem establishes the basic convergence result and follows closely from the results in [61]. This result uses Clarke's generalized derivative [13], which is, the *generalized directional derivative* of f in the direction d, defined by

$$f^o(x;d) = \limsup_{\substack{z \to x \\ \lambda \downarrow 0}} \frac{f(z+\lambda d) - f(z)}{\lambda},$$

where $z \in \mathbb{R}^n$ and λ is a positive scalar.

First we show that there exists a dense set points in neighborhood of z_m with larger f values if the m^{th} execution of Step 3(e) is infinite.

Proposition 56. *If Step 3(e) of the Hooke and Jeeves / CARTopt hybrid algorithm is an infinite process, then there exists a dense set of points with larger f values in the neighborhood of z_m with probability one.*

Proof. Step 3(e) is an infinite process and so CARTopt fails to generate a point with a function value less than $f(z_m)$. Noting that z_m is an element of CARTopt's input training data and no lower points are generated, the sequence of iterates $\{z_k\}_{k=1}^{\infty}$ generated by CARTopt remains constant with $z_k = z_m$ for all k (see Step 3(g) in Figure 3.1). Hence, z_m is a cluster point. It follows directly from Theorem 38 that z_m is an essential local minimizer of f with probability one. From the definition of an essential local minimizer (see Definition 1), the set

$$\mathfrak{E}(z_m, \epsilon) = \{x \in \Omega_m : f(x) < f(z_m) \text{ and } \|x - z_m\| < \epsilon\} \tag{5.17}$$

has Lebesgue measure zero for all sufficiently small $\epsilon > 0$. That is, there exists a dense set of points with larger f values in the neighborhood of z_m with probability one. □

The convergence result for the smooth version can now be given.

Theorem 57.

(a) Assume the sequences $\{z_m\}$ and $\{x_k\}$ are finite. If f is strictly differentiable at the final value z_{m_*} of $\{z_m\}$, then $\nabla f(z_{m_*}) = \mathbf{0}$.

(b) Let f be locally Lipshitz at z_*. If z_* is a cluster point of the sequence of grid local minimizers $\{z_m\}$ and if f is strictly differentiable at z_*, then $\nabla f(z_*) = \mathbf{0}$.

Proof. For part (a) the only way that the sequences $\{z_m\}$ and $\{x_k\}$ are finite is if Step 3(e) is an infinite process. Noting that f is strictly differentiable [13] at z_{m_*}, we have

$$\exists w \in \mathbb{R}^n \text{ such that } f^o(z_{m_*}; d) = w^\mathsf{T} d \text{ for all } d \in \mathbb{R}^n. \tag{5.18}$$

Proposition 56 implies that there exists a dense set of points in the neighborhood of z_{m_*} such that $f(z_{m_*}) \leq f(x)$ for all $\|x - z_{m_*}\| < \epsilon$. Therefore, the generalized directional derivative exists in all directions and is non-negative. From (5.18) the only possibility is $w = \mathbf{0}$, or equivalently $\nabla f(z_{m_*}) = \mathbf{0}$.

For part (b) we restrict our attention to a subsequence $\{z_j\}$ for which the corresponding subsequence $\{z_j, \mathcal{V}_j^+\}$ converges uniquely to $(z_*, \mathcal{V})$. In what follows, ν_j is an element of the positive basis $\mathcal{V}_j^+$ (see (5.1)). Using the definition of a grid local minimizer we have,

$$f(z_j + h_j \nu_j) - f(z_j) \geq 0 \text{ for all } \nu_j. \tag{5.19}$$

Rewriting (5.19), dividing by h_j and taking the limit we have

$$\limsup_{j \to \infty} \frac{f(z_j + h_j(w_j + \nu)) - f(z_j + h_j w_j) + f(z_j + h_j w_j) - f(z_j)}{h_j} \geq 0,$$

where $w_j = \nu_j - \nu$. Noting that $w_j \to \mathbf{0}$, $z_j \to z_*$ and $h_j \downarrow 0$ (Proposition 54) as $j \to \infty$, the first two terms give Clarke's generalized derivative at z_* and so we have

$$f^o(z_*; \nu) + \limsup_{j \to \infty} \frac{f(z_j + h_j w_j) - f(z_j)}{h_j} \geq 0. \tag{5.20}$$

With f locally Lipschitz at z_* we have

$$|f(z_j + h_j w_j) - f(z_j)| \leq K \|z_j + h_j w_j - z_j\|,$$

for a positive scalar K. Thus, after applying the Lipschitz condition and evaluating the limit, the second term of (5.20) vanishes leaving

$$f^o(z_*; \nu) \geq 0. \tag{5.21}$$

Recall that all smooth functions have an open half space of descent directions centered on x if $\nabla f(x) \neq \mathbf{0}$. Strict differentiability at z_* implies

$$\exists w \in \mathbb{R}^n \text{ such that } f^o(z_*; \nu) = w^\mathsf{T} \nu \text{ for all } \nu \in \mathbb{R}^n$$

and so if w is non-zero, there exists an open half space for which $w^\mathsf{T}\nu < 0$. However, every positive basis has at least one vector probing any open half space (Theorem 8) and (5.21) states that all such directions have $w^\mathsf{T}\nu \geq 0$. Therefore, w must be the zero vector and hence, $\nabla f(z_*) = \mathbf{0}$ as required. □

5.7.2 Nonsmooth Result

The nonsmooth convergence result is now given.

Theorem 58. *Exactly one of the following possibilities holds:*

(a) $\{z_m\}$ *is an infinite sequence and each cluster point* z_* *of the sequence is an essential local minimizer of* f *with probability one; or*

(b) both $\{z_m\}$ *and* $\{x_k\}$ *are finite sequences and the final* z_m *is an essential local minimizer of* f *with probability one; or*

(c) $\{z_m\}$ *is finite and* $\{x_k\}$ *is an infinite unbounded sequence.*

Proof. Noting that a sequence cannot be both infinite and finite only one of these possibilities can hold.

Case (a) is a direct consequence of Theorem 38.

Let $\{z_m\}$ be a finite sequence and let m_* be the final value of m. There are two possible ways this can happen; either the inner loop is an infinite process or the localized global phase is an infinite process. For the former we consider two cases. Firstly, if the upper bound $\mathcal{U}_k$ on f is eventually constant i.e. $\mathcal{U}_k = \mathcal{U}_{k_*}$ for all $k > k_*$, we have $f(x_k) \leq f(x_{k-1})$ for all $k > k_*$ by (5.2) and all $x_k \in \mathcal{G}_{m_*}$ for all k sufficiently large.

Secondly, if $\mathcal{U}_k$ is not eventually constant, then from (5.3) and (5.4) there exists a strictly decreasing subsequence $\{\mathcal{U}_i\}_{i=1}^\infty$ of $\{\mathcal{U}_k\}$ such that $\mathcal{U}_{i+1} < \mathcal{U}_i$ for all i. Furthermore, there exists an infinite subsequence $\{y_i\}_{i=1}^\infty$ of $\{x_k\}$ for which $f(y_i) \leq \mathcal{U}_i$ for all i and all $y_i \in \mathcal{G}_{m_*}$. In either case $\{x_k\}$ must be an infinite unbounded sequence, which is case (c).

For the latter we have the global phase being an infinite process. Theorem 38 implies that there does not exist a set

$$\mathfrak{E} = \{x \in \Omega_{m_*} : f(x) < f(z_{m_*}) \text{ and } \|x - z_{m_*}\| < \epsilon\}$$

with positive measure. Therefore, z_{m_*} is an essential local minimizer of f with probability one, which is case (b). □

Corollary 59. *If the sequence* $\{x_k\}$ *is bounded, then every cluster point of the sequence* $\{z_m\}$ *is an essential local minimizer of* f *with probability one.*

Proof. With $\{x_k\}$ a bounded sequence, case (c) is removed from Theorem 58 and the result follows. □

Chapter 6

A CARTopt Filter Method for Nonlinear Programming

This chapter considers finding a local solution of a Nonlinear Programming (NLP) problem. Many practical optimization problems have restrictions placed on acceptable solutions to the minimization problem (1.1). For example, one or more of the variables may represent physical quantities, such as quantities of manufacturing materials, that cannot take negative values. To enforce restrictions, a set of constraint functions $\{c_i(x)\}$ are included, defining a NLP problem. The optimization problem considered here can be written as

$$\min_{x \in \mathbb{R}^n} f(x) \text{ such that } C(x) \leq 0 \tag{6.1}$$

where $f : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ and $C : \mathbb{R}^n \to (\mathbb{R} \cup \{\infty\})^m$ are functions with $C = (c_1, c_2, \ldots, c_m)^\mathsf{T}$. The constraints define feasible points and the set of feasible points is called the feasible region.

Definition 60. (Feasible point). *A point $x \in \mathbb{R}^n$ is feasible if and only if $c_i(x) \leq 0$ for all $i = 1, \ldots, m$.*

Definition 61. (Feasible region). *The feasible region $\mathcal{N}$ is the set of all feasible points.*

If the objective function is linear and the constraints are linear, then the optimization problem is referred to as a linear programming problem or linear program. These problems occur frequently in management science and operations research. If the objective function is quadratic and the constraints are linear, the optimization problem is referred to as a quadratic programming problem or quadratic program. In this chapter,

the general NLP given by (6.1) is considered, rather than forcing linear constraints, for example.

General NLP problems can be solved using penalty function methods. Some of these methods try to create a smooth function with a local minimum near a solution to the constrained problem by penalizing infeasible regions. In this approach a series of unconstrained problems of the form

$$\min_{x\in\mathbb{R}^n} g(x,\sigma) = f(x) + \sigma\mathfrak{h}(x) \tag{6.2}$$

are solved. Here $\mathfrak{h}$ is a constraint violation function and σ is a penalty parameter. Choosing

$$\mathfrak{h}(x) = \sum_{i=1}^{m} (\max\{c_i(x), 0\})^2,$$

for example, gives the quadratic penalty function. The basic algorithm is defined as follows: Set $k = 0$. Given an $x_0 \in \mathbb{R}^n$ and $\sigma_0 > 0$, solve (6.2) starting from x_k and call the solution x_{k+1}. Choose $\sigma_{k+1} > \sigma_k$ and solve (6.2) once more starting from x_{k+1}. Repeat until stopping conditions are satisfied. Increasing the penalty parameter σ tries to drive the sequence of iterates $\{x_k\}$ to a solution of the constrained problem. However, choosing σ can be problematic [26]. If σ becomes too large, solving (6.2) can become increasingly ill conditioned, whereas too small and x_{k+1} can be a worse approximation than the current iterate x_k.

Other approaches include exact penalty functions and barrier methods. These methods penalize infeasible points but leave the solution to the original NLP unchanged. However, smoothness is lost making the use of gradient based methods (and others) problematic. These problems can be solved using the CARTopt methods described in the previous chapters if the following assumption is satisfied.

Assumption 62. *Let the closure of the interior of the feasible region $\mathcal{N}$ be $\mathcal{N}$ itself.*

Assumption 62 insures that for all $x \in \mathcal{N}$, there exists an $\epsilon > 0$ such that $\mathcal{B}(x,\epsilon)\cap\mathcal{N}$ with positive Lebesgue measure. If equality constraints are present in (6.1), Assumption 62 is not satisfied and the previous CARTopt methods fail — probability of sampling the feasible region is zero. Here an alternative approach using the concept of a filter is used to solve (6.1), including the case when equality constraints are present.

Firstly, the concept of a filter is introduced and its usage in constrained optimization is discussed. Section 6.1.2 defines the conditions imposed on the constraint violation function and provides some examples. The CARTopt filter algorithm is introduced

in Section 6.2 and developed over the subsections that follow. In subsection 6.2.1 the concept of a sloping filter is proposed. A stopping rule is given in Section 6.3 giving practical convergence to locally optimal points. Convergence to limit points with desirable properties is demonstrated in Section 6.4. Here the objective function and constraint violation function is assumed to be nonsmooth.

6.1 Filter Algorithms

Filter based algorithms were introduced by Fletcher and Leyffer [26] as a way to promote global convergence on sequential quadratic programming problems without the use of a penalty function. Filter algorithms treat the constrained optimization problem (6.1) as biobjective — one wishes to minimize both the objective function f and the constraint violation $\mathfrak{h}$. However, priority must be given to minimizing constraint violation to promote convergence to a feasible solution.

6.1.1 Filter

To solve the biobjective optimization problem the notion of dominance from multiobjective terminology is introduced. For a pair of distinct vectors $w, \hat{w}$ with finite elements, w *dominates* $\hat{w}$ if and only if $w_i \leq \hat{w}_i$ for each i and $w \neq \hat{w}$. This is written as $w \prec \hat{w}$ and the notation $w \preceq \hat{w}$ is used to indicate that either $w \prec \hat{w}$, or w and $\hat{w}$ are equivalent. The convention that any vector with finite elements dominates any vector with an infinite element is used. Furthermore, two vectors containing at least one infinite element are considered equivalent.

To simplify notation, dominance is defined for our particular situation.

Definition 63. (Dominance). *A point $x \in \mathbb{R}^n$ is said to dominate $z \in \mathbb{R}^n$, written $x \prec z$, if and only if*

$$(\mathfrak{h}(x), f(x))^{\mathsf{T}} \prec (\mathfrak{h}(z), f(z))^{\mathsf{T}}.$$

Clearly, if $x \prec z$, then either $\mathfrak{h}(x) < \mathfrak{h}(z)$ or $f(x) < f(z)$ (or both) must hold. Using the definition of dominance a filter is now defined.

Definition 64. (Filter). *A filter is a set of points in $\mathbb{R}^n$ such that no point dominates another.*

The filter can be represented geometrically in the $(\mathfrak{h}, f)$ plane as illustrated in Figure 6.2.

Fletcher *et al.* use the filter as a criterion for accepting or rejecting a step in a sequential quadratic programming method [26]. An iterate x_k is accepted by the filter if the corresponding pair $(\mathfrak{h}(x_k), f(x_k))$ is not dominated by any point in the filter and the filter is updated accordingly for the next iteration. Their filter method requires the explicit use of derivatives on both the objective and constraints. Audet and Dennis use a filter in a similar way to accept or reject points in a derivative free pattern search method [4]. This method is of interest here because it can be applied to NLP with discontinuous or nonsmooth objective functions. However, convergence is only demonstrated when smoothness assumptions are placed on both f and $\mathfrak{h}$. Karas *et al.* also provide a bundle-filter method for nonsmooth convex constrained optimization [40]. Similarly, the filter is used as a way of accepting or rejecting a step in their method. Convergence to optimal points is demonstrated when both f and h are assumed to be convex [40]. The method proposed here provides strong convergence results when f, $\mathfrak{h}$, or both are assumed to be nonsmooth or discontinuous.

6.1.2 Constraint Violation Function

The constraint violation function $\mathfrak{h}$ is measure of how feasible an iterate is. There are many possibilities. Constraint violation functions considered here satisfy two simple conditions. Firstly, $\mathfrak{h}(x) \geq 0$ for all x. Secondly, $\mathfrak{h}(x) = 0$ if and only if x is a feasible point. By convention, $\mathfrak{h}(x) = \infty$ if any component of $C(x)$ is infinite.

This thesis considers constraint violation functions of the form,

$$\mathfrak{h}(x) = \|[C(x)]_+\|,$$

where $\|.\|$ is a vector norm and $[y]_+ = \max\{y, 0\}$. Clearly, these functions satisfy the conditions stated above. Of particular interest are the standard vector norms — the 1, 2 and ∞ norm measures. The use of these constraint violation functions has been investigated recently by Griffin *et al.* on penalty function methods. The interested reader is referred to [29] for details.

The 1-norm violation gives the sum of violations

$$\mathfrak{h}_1(x) = \sum_{i=1}^{m} [c_i(x)]_+. \tag{6.3}$$

This choice is used by Fletcher and Leyffer in their paper introducing the filter approach, as it has convenient features exploited in their algorithm [26]. However, Audet

and Dennis provide a simple example for which their method fails when $\mathfrak{h}_1$ is used [4]. In their method, each step is obtained by polling directions using a positive spanning set and thus, a descent direction can be missed because $\mathfrak{h}_1$ is not differentiable everywhere.

Two variations of the 2-norm constraint violation function

$$\mathfrak{h}_2(x) = \left(\sum_{i=1}^{m} [c_i(x)]_+^2 \right)^{\theta}, \tag{6.4}$$

are considered here. With $\theta = 1/2$ the standard 2-norm is obtained. With $\theta = 1$, $\mathfrak{h}_2$ has the nice property of being continuously differentiable whenever C is [4]. This choice is preferred by Audet and Dennis because if there exists a descent direction in $\mathfrak{h}_2$, then a positive spanning set will detect it (see Theorem 8 of this thesis) and their method generates a successful step.

Finally, the ∞-norm constraint violation function is

$$\mathfrak{h}_\infty(x) = \max_i [c_i(x)]_+, \tag{6.5}$$

which gives the maximum constraint violation.

6.2 A CARTopt Filter Algorithm

In this section a CARTopt filter algorithm is presented for nonlinear programming. The algorithm is a variation on the unconstrained CARTopt instance proposed in Chapter 3 and is explicitly stated in Figure 6.1. It consists of an initialization phase (steps 1 and 2) and a single loop (Step 3).

Step 1 sets the iteration counter $k = 1$ and the user chooses a batch size $N > 0$, an initial hypercube radius $h > 0$, a minimum sub-region radius $\delta > 0$ and a constraint violation function $\mathfrak{h}$ with an initial penalty parameter $0 < \sigma_1 \leq \sigma_{\max}$, where $\sigma_{\max}$ is finite (see Section 6.2.5). If the user has an initial set of points for which f and $\mathfrak{h}$ are known, an initial training data set T_0 can be used. If no information is available, T_0 is set as empty. To complete Step 1 an initial point $x_0 \in \mathbb{R}^n$ such that both $f(x_0)$ and $\mathfrak{h}(x_0)$ are finite is required. This can be achieved, for example, by random polling. However, if T_0 is non-empty and contains at least one point for which both the objective and constraint violation are finite, x_0 is chosen from T_0. Specifically, x_0 is the element with the least $\mathfrak{h}$ value and if more than one exists, the one with the least f value.

1. Initialize: Set $k = 1$. Choose $N > 0$, $h > 0$, $\delta > 0$, T_0 and a constraint violation function $\mathfrak{h}$ with penalty parameter $0 < \sigma_1 \leq \sigma_{\max}$. Choose an $x_0 \in \mathbb{R}^n$ such that both $f(x_0)$ and $\mathfrak{h}(x_0)$ are finite. Set $T_1 = [x_0; T_0]$ and $z_0 = x_0$.

2. Generate initial batch of $\max\{2N - |T_1|, 0\}$ sample points $x \in x_0 + h[-1, 1]^n$ and store in X_1. Evaluate the objective $f(x)$ and the constraint violation $\mathfrak{h}(x)$ at each $x \in X_1$.

3. **while** stopping conditions are not satisfied **do**

   (a) Update filter, training data set and classify:

   $$T_{k+1} \subset \{X_k \cup T_k\} \text{ such that } T_{k+1} = \{\omega_1, \ldots, \omega_M\},$$

   where each classified subset ω_i is non-empty and the cardinality of T_{k+1} is finite.

   (b) Partition and sampling phase: Using T_{k+1} form a CART partition on $\mathbb{R}^n$ to define an approximate level set $\mathcal{L}_k$ of

   $$g_k(x, \sigma_k) = f(x) + \sigma_k \mathfrak{h}(x).$$

   Draw N points from $\mathcal{L}_k$, giving the next batch X_{k+1}.

   (c) Evaluate $f(x)$ and $\mathfrak{h}(x)$ at each $x \in X_{k+1}$. Choose z_{k+1} and x_{k+1} as the elements from $T_{k+1} \cup X_{k+1}$ that minimize $g_k(x, \sigma)$ and $\mathfrak{h}(x)$ respectively.

   (d) Update the penalty parameter by choosing $\sigma_k \leq \sigma_{k+1} \leq \sigma_{\max}$. Check stopping conditions and increment k.

   end

Figure 6.1: CARTopt Filter algorithm

Step 2 completes the initialization phase of the algorithm, generating $\max\{2N - |T_1|, 0\}$ points in a hypercube sub-region centered on x_0, defined by $x_0 + h[-1, 1]^n$. The hypercube radius h can be any positive number, however, numerical experience suggests that choosing a large h is advantageous. Choosing h small restricts the search space promoting convergence to points near x_0, whether they are feasible or not. Whereas, a large h gives the algorithm a *global* feel in the early stages of sampling, exploring a larger subset of $\mathbb{R}^n$. Furthermore, larger steps can be taken early on rather than lots of little ones. Noting that finding a feasible point is a global optimization problem, such an approach promotes convergence to feasible optimal points. In addition, if T_0 is non-empty, h should be chosen such that T_0 is a subset the hypercube. If a sufficiently large input training data set is known, Step 2 is not required. When complete, an initial (unclassified) batch of at least $2N$ sample points are known.

The main loop (Step 3) of the algorithm generates two infinite sequences of iterates $\{z_k\}$ and $\{x_k\}$. These sequences are obtained by evaluating the objective function and constraint violation at a finite number of points X_k in a neighborhood defined by a training data set. Each iterate of $\{z_k\}$ is chosen as the element from the training data set T_k and newly generated batch of points X_k that minimizes

$$g_k(x, \sigma_k) = f(x) + \sigma_k \mathfrak{h}(x), \tag{6.6}$$

where σ_k is a penalty parameter (see Section 6.2.5). If more than one element from $T_k \cup X_k$ minimizes g_k, an element that minimizes g_k with the least constraint violation is chosen. Each iterate of $\{x_k\}$ is chosen as an element of $T_k \cup X_k$ that minimizes the constraint violation $\mathfrak{h}(x)$ and if more than one exists, the one with the least f value is chosen. If a feasible iterate is generated at iteration k^* ($\mathfrak{h}(x_{k^*}) = 0$), the subsequence $\{x_k\}_{k=k^*}^{\infty}$ contains only feasible iterates and is necessarily monotonically decreasing with respect to f. Whereas, $\{z_k\}$ can take both feasible and infeasible iterates and nothing can be said about f. Each component of the main loop is described in the subsections which follow.

The algorithm terminates when stopping conditions are satisfied. The stopping rule can be implemented with respect to either $\{z_k\}$ or $\{x_k\}$. In either case, the terminating iterates have desirable properties with respect to $\mathfrak{h}$, f, or $g_k(x, \sigma_k)$. This is discussed further in Section 6.3.

6.2.1 Sloping Filter

This section proposes a variation on Fletcher's filter called the sloping filter. Both Fletcher *et al.* [26] and Audet *et al.* [4] use the filter as a way of rejecting iterates. That is, if a step generates a filtered iterate, it is rejected. In our approach elements of the filter are chosen to be interior points of a subset of $\mathbb{R}^n$ classified as low. Such regions are sampled further in an attempt to generate points that modify the filter, attempting to drive feasibility from multiple points. Only infeasible points are included in the filter and feasible points are treated separately. All feasible points are retained in the CARTopt filter algorithm until a full size training data set is obtained so that feasible low sub-regions are well defined in the partition, see next section.

In this approach it is important that the filter does not become too large. Here, a maximum cardinality of $3N/4$ is placed on the sloping filter, where N is the batch size. This bound is directly related to the cardinality of desirable points classified as low, which is N (see Section 6.2.3). To ensure the filter remains sufficiently small, elements are systematically removed from Fletcher's filter. In particular, elements that greatly increase the constraint violation but only slightly reduce f are removed. The resulting subset is called the sloping filter.

Definition 65. (Sloping filter). *A sloping filter $\mathcal{F}$ is a set of points in $\mathbb{R}^n$ such that for all $x \in \mathcal{F}$ there does not exist a $z \in \mathcal{F}$ satisfying*

$$\mathfrak{h}(x) > \mathfrak{h}(z) \text{ and } f(x) > f(z) - \mu(\mathfrak{h}(x) - \mathfrak{h}(z)),$$

where μ is a positive scalar.

The sloping filter can be represented in the $(\mathfrak{h}, f)$ plane as illustrated in Figure 6.2.

The sloping filter is calculated as follows. Firstly, the set of infeasible undominated points $X_{\mathcal{D}}$ from the union of the training data set and newly generated batch $T_k \cup X_k$ is calculated.

If $|X_{\mathcal{D}}| \leq 3N/4$ (sufficiently small), then $\mathcal{F}_k = X_{\mathcal{D}}$. Otherwise $X_{\mathcal{D}}$ contains too many elements and the slope parameter μ is used to remove elements from $X_{\mathcal{D}}$.

Starting with an initial $0 < \mu < \sigma_k$, μ is systematically increased until the cardinality of $\mathcal{F}_k$ is sufficiently small, or σ_k is reached. Here σ_k is the penalty parameter in g_k, which is bounded for all k (see Section 6.2.5). One strategy, for example, is to choose the sequence

$$\left\{ \frac{\sigma_k}{2^{10}}, \frac{\sigma_k}{2^9}, \ldots, \frac{\sigma_k}{2}, \sigma_k \right\}. \tag{6.7}$$

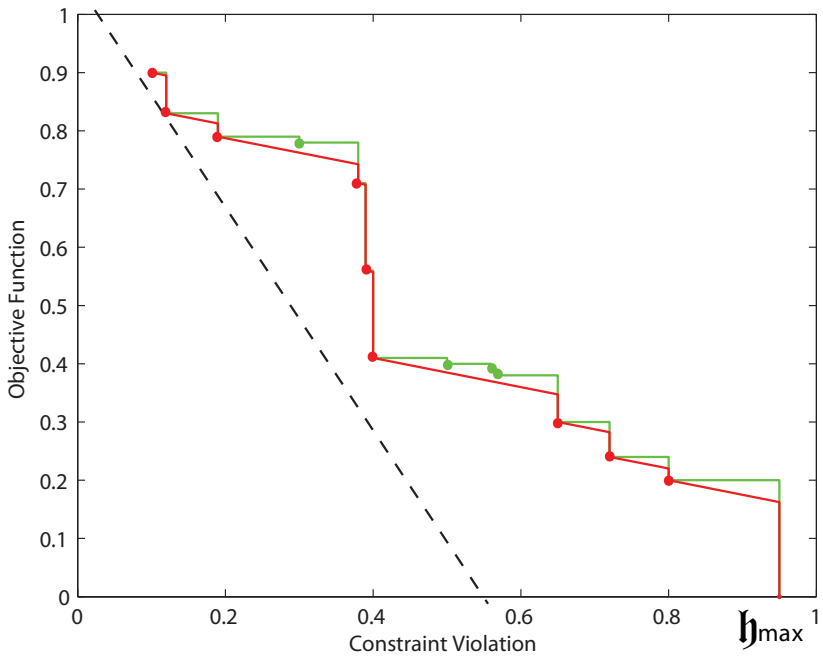


Figure 6.2: The sloping filter is shown in red and Fletcher's filter is shown green. Four elements (green dots) have been removed from $X_{\mathcal{D}}$. The dashed line is an example of an exact penalty function (6.6), which is also the upper bound on μ.

Such a sequence gives preference to *shallow* gradients with respect to σ_k, removing elements with relatively small changes in f and relatively large increases in $\mathfrak{h}$ first and is used here. With $\mu \leq \sigma_k$, an element of $X_{\mathcal{D}}$ that minimizes g_k will not be removed.

If the upper bound $\mu = \sigma_k$ is reached, an element of $X_{\mathcal{D}}$ that minimizes $g_k(x, \sigma_k)$ with the least constraint violation and the $3N/4 - 1$ most feasible elements are chosen to give $\mathcal{F}_k$. Clearly such an approach forces $|\mathcal{F}_k| \leq 3N/4$, as required. Furthermore, with only infeasible iterates included in the sloping filter, the most feasible infeasible iterate is always an element of $\mathcal{F}$. In addition, with (6.7) containing finitely many elements, this update is a finite process — a property needed for convergence.

6.2.1.1 Imposing an Upper Bound on Constraint Violation

An upper bound $\mathfrak{h}_{\max} > 0$ is placed on the constraint violation function. The inclusion of an upper bound prevents the unlikely case that a sequence of points is generated such that $f(z_{k+1}) < f(z_k)$ and $\mathfrak{h}(z_{k+1}) > \mathfrak{h}(z_k)$ for all k, with $\mathfrak{h}(z_k) \to \infty$ in the limit as $k \to \infty$. This is easily implemented by including $(\mathfrak{h}, f) = (\mathfrak{h}_{\max}, -\infty)$ in $X_{\mathcal{D}}$ (see Figure 6.2). Furthermore, $\mathfrak{h}_{\max}$ can be systematically reduced to remove the most

infeasible elements from $X_{\mathcal{D}}$. This can be achieved, for example, by setting $\mathfrak{h}_\infty$ as the most infeasible element of the sloping filter. Numerical results herein are generated using $\mathfrak{h}_{\max} = \max\{1, \mathfrak{h}(x_0)\}$, which remains constant for all iterations.

6.2.1.2 Imposing a Lower Bound on Constraint Violation

From a practical point of view a lower bound $\mathfrak{h}_{\min} > 0$ is placed on constraint violation. The inclusion of $\mathfrak{h}_{\min}$ precludes the existence of a sequence of infeasible iterates such that $f(x_{k+1}) > f(x_k)$ and $\mathfrak{h}(x_{k+1}) < \mathfrak{h}(x_k)$ for all k, with $\mathfrak{h}(x_k) \to 0$ as $k \to \infty$. Numerical simulations performed by the author on equality constrained problems (feasible region has measure zero) encountered such sequences, ultimately terminating the method at undesirable points. To increase numerical performance the following strategy was used. An element $z \in X_{\mathcal{D}}$ is filtered by another element $x \in X_{\mathcal{D}}$ if

$$\mathfrak{h}(x), \mathfrak{h}(z) < \mathfrak{h}_{\min} \text{ and } f(x) < f(z).$$

Here the value $\mathfrak{h}_{\min} =$ 1e-10 is used. However, if Assumption 62 is easily verified for a given problem, $\mathfrak{h}_{\min}$ is set to zero. This allows the algorithm to approach the feasible region of positive measure from the infeasible region if $\mathfrak{h}(x_k) \to 0$ as $k \to \infty$. This parameter is included from a practical point of view and is set to zero when convergence properties of the method are analyzed.

6.2.2 Training Data Set

There is great freedom in defining a training data set T for the CARTopt filter based method, only requiring mild conditions. Firstly, the cardinality of T must remain finite and contain at least two non-empty distinct sets of classified data, from which a CART partition on $\mathbb{R}^n$ can be formed, for all iterations. Secondly, the current iterates z_k and x_k must be elements of T_k. Finally, there must exist a non-zero probability of sampling a neighborhood of both z_k and x_k for all k. Here, as in the previous chapters of this thesis, two sets are considered — points with relatively low and high values. Forcing both z_k and x_k to be classified as low, satisfies the last condition placed on T (see (6.17)). Previously, T was chosen to reflect local information about where f is relatively low. However in the filter based approach, preference to feasibility is also included in T. Ideally, partitions induced from T locate feasible sub-regions where f is relatively low (if they exist), or modify the sloping filter to drive feasibility. This section focuses explicitly on the training data set and classification is left until section

which follows.

The training data set proposed here is similar to the set presented in Section 3.2. However, here we have three sets of desirable points instead of one. These are, the sloping filter, elements of $T_k \cup X_k$ with the least constraint violation and those with the least $g_k(x, \sigma_k)$ values. The training data set is defined via three subsets with the following structure

$$T_{k+1} = \{\mathcal{F}_k \cup \mathfrak{T}_\gamma, \mathfrak{T}_R\}. \tag{6.8}$$

The first set is the sloping filter $\mathcal{F}_k$ defined in the previous section. The sloping filter is iteratively updated with any infeasible points generated in the sampling phase, that is

$$\mathcal{F}_k \subset \{x \in T_k \cup X_k : \mathfrak{h}(x) > 0\}. \tag{6.9}$$

Provided an infeasible iterate is generated, the sloping filter remains non-empty and contains the most feasible infeasible solution for all iterations. If no infeasible points are generated, the sloping filter remains unchanged. The sloping filter's cardinality is bounded by $3N/4$ and thus, remains finite for all iterations.

The second set defines elements from $T_k \cup X_k$ with the least constraint violation $\mathfrak{h}$ and those with the least $g_k(x, \sigma_k)$ values. For clarity, let $\mathfrak{T}_\mathfrak{h}$ and $\mathfrak{T}_g$ denote the $\gamma > 0$ elements from $T_k \cup X_k$ with the least $\mathfrak{h}$ and $g_k(x, \sigma_k)$ values, respectively. If there exist multiple points with the same constraint violation, then preference towards elements with the least f values may be required to force $|\mathfrak{T}_\mathfrak{h}| = \gamma$. Similarly, if multiple points yield the same $g_k(x, \sigma_k)$ value, preference towards feasibility may be required to force $|\mathfrak{T}_g| = \gamma$. Then, we define the set

$$\mathfrak{T}_\gamma = \{\mathfrak{T}_\mathfrak{h} \cup \mathfrak{T}_g\} \tag{6.10}$$

where $\gamma \leq |\mathfrak{T}_\gamma| \leq 2\gamma$. Thus, (6.10) contains the γ best elements with respect to both $\mathfrak{h}$ and $g_k(x, \sigma_k)$ for all iterations. In particular, it contains the feasible point(s) with the least function value(s) if any exist. These sets of points are used in the stopping rule in Section 6.3.

The third set $\mathfrak{T}_R$ defines the most recent elements generated from successive sampling phases. Specifically, the

$$\min\{|\{T_k \cup X_k \setminus \mathcal{F}_k \cup \mathfrak{T}_\gamma\}|, T_{\max} - |\{\mathcal{F}_k \cup \mathfrak{T}_\gamma\}|\} \tag{6.11}$$

most recent elements from the set $\{T_k \cup X_k\} \setminus \{\mathcal{F}_k \cup \mathfrak{T}_\gamma\}$ are retained. Here

$$T_{\max} = \max\{2\gamma + N, 2N \max\{1, n-1\}\}, \tag{6.12}$$

is an upper bound placed on the cardinality of T, although other choices are possible. A training data set with cardinality $T_{\max}$ is called full size. This definition of full size is larger than our previous definition for small values of n to ensure the sloping filter and $\mathfrak{T}_\gamma$ are always elements of the training data set. This is crucial for the stopping rule and defining the set of desirable points.

In summary, all points generated are retained in T until full size is reached and then the oldest points with relatively large $\mathfrak{h}$ and $g_k(x, \sigma_k)$ values are discarded. The training data set contains feasible points (if they exist), the sloping filter and selected filtered points. With the cardinality of T finite, Step 3 of the CARTopt filter algorithm is a finite process. This property is needed to establish convergence.

6.2.3 Classification

To form a partition on $\mathbb{R}^n$ a classification must be imposed on the training data set. At least two non-empty classified subsets of T are required to form a CART partition on $\mathbb{R}^n$. The classification must include the current iterates z_k and x_k in the set of most desirable points so that there exists a non-zero probability of sampling a neighborhood of both z_k and x_k during the next iteration — a property needed to establish convergence. There are many possibilities.

The classified training data set proposed here consists of two sets — low and high points. Attention is focused primarily on defining the set of low points $\{\omega_L\}$, as they will be interior points of the subset of $\mathbb{R}^n$ that is explored further during the next iteration. To this end, promoting convergence to optimal feasible points is the primary goal. To achieve this, sampling neighborhoods of feasible points (if they exist) and elements of sloping filter is advantageous.

The set $\{\omega_L\}$ proposed here consists of both feasible and infeasible points. The inclusion of infeasible points allows the method to approach an optimal point from either the feasible or infeasible region. If at least one constraint is active at a solution to the NLP (i.e. $c_i(x_*) = 0$ for some $1 \leq i \leq m$), then it can be advantageous to classify particular infeasible iterates as low to yield a desirable partition. That is, a partition which has x_* as an interior point of the low sub-region. The interested reader is referred to Figure 7.1 in the next chapter, which illustrates the relative merits of

including infeasible points in $\{\omega_L\}$.

For clarity, $\{\omega_L\}$ is defined via three subsets $\{\omega_{\mathfrak{T}}\}, \{\omega_{\mathcal{F}}\}$ and $\{\omega_{\bar{\mathcal{F}}}\}$. With $z_k = x_k$ possible, the notation $\lambda = |\{z_k \cup x_k\}|$ is used for convenience.

The first subset gives the feasible points $\mathfrak{T} = \{x \in T_{k+1} \cap \mathcal{N}\}$ to be included in $\{\omega_L\}$. Specifically, the

$$\min\{|\mathfrak{T}|, \max\{\lceil N/2 - \lambda \rceil, N - |\mathcal{F}_k| - \lambda\}\} \tag{6.13}$$

elements of $\mathfrak{T} \setminus \{z_k, x_k\}$ with the least objective function values form the set $\{\omega_{\mathfrak{T}}\}$. If there are no feasible points (6.13) is empty. This is the case when equality constraints are included in the nonlinear programme, as all samples generated are infeasible with probability one. If there are no infeasible points ($\mathcal{F}_k = \emptyset$), the N points with the least objective function values form $\{\omega_L\}$.

The second subset gives elements from the sloping filter to be included in $\{\omega_L\}$. These are the

$$\min\{|\mathcal{F}_k|, N - |\mathfrak{T}| - \lambda\} \tag{6.14}$$

elements of $\mathcal{F}_k \setminus \{z_k, x_k\}$ with the least constraint violation and form the set $\{\omega_{\mathcal{F}}\}$. If (6.13) is empty, then all elements of the sloping filter are included in $\{\omega_L\}$ ($|\mathcal{F}_k| \leq 3N/4$ for all k).

Finally, a subset of filtered elements $\bar{\mathcal{F}} \in \{x \in T_{k+1} \setminus \{\mathfrak{T}, \mathcal{F}_k\}\}$ are included in $\{\omega_L\}$, given by, the

$$[N - |\mathfrak{T}| - |\mathcal{F}_k| - \lambda]_+ \tag{6.15}$$

elements of $\bar{\mathcal{F}}$ with the least g_k values and form the set $\{\omega_{\bar{\mathcal{F}}}\}$. These points are only included if both $\{\omega_{\mathfrak{T}}\}$ and $\{\omega_{\mathcal{F}}\}$ are sufficiently small.

Definition 66. (Low points). *Let $\omega_{\mathfrak{T}}, \omega_{\mathcal{F}}$ and $\omega_{\bar{\mathcal{F}}}$ be defined as above, then the set of points $\{\{z_k\} \cup \{x_k\}, \omega_{\mathfrak{T}}, \omega_{\mathcal{F}}, \omega_{\bar{\mathcal{F}}}\}$ are classified as low and form the set $\{\omega_L\}$.*

Definition 67. (High points). *The set of points $T_{k+1} \setminus \{\omega_L\}$ is classified as high and form the set $\{\omega_H\}$.*

From the definitions above it is clear that both $\{\omega_L\}$ and $\{\omega_H\}$ are non-empty for all k and so there exists a CART partition on $\mathbb{R}^n$ for all iterations. Clearly, both z_k and x_k are elements of $\{\omega_L\}$ for all iterations. The cardinality of $\{\omega_L\}$ is N and remains constant for all iterations. Whereas, $|\{\omega_H\}| \geq N$ for all iterations and holds strictly for $k > 1$. Having $|\{\omega_H\}| > |\{\omega_L\}|$ aims to promote clustering in $\{\omega_L\}$, see Section 3.2.

6.2.4 Partition and Sampling Phase

Step 3(c) of the CARTopt filter algorithm conducts the CART partition and sampling phase on $\mathbb{R}^n$. For a detailed discussion the interested reader is referred to Chapter 3, focusing particularly on the unconstrained instance of CARTopt. This section summarizes important features and necessary properties for convergence.

A CART partition on $\mathbb{R}^n$ is performed using the classified training data set T_k, defined in Step 3(a). Before the partition is made, a Householder transformation is applied to T_k to potentially simplify the partition (see Section 3.3). Taking the union of hyper-rectangular low sub-regions identified by the partition, a bounded approximate level set $\mathcal{L}_k$ with respect to $g_k(x, \sigma_k)$ is defined. To ensure $\mathcal{L}_k$ is bounded the post-partition modifications in Section 3.7.3 are made. However, the forward tracking face search is conducted with respect to $g_k(x, \sigma_k)$, rather than f. That is, a problematic bound is fixed if ascent in $g_k(x, \sigma_k)$ made. If a problematic bound is a subset of the feasible region ($\mathfrak{h} = 0$), the update is equivalent to Section 3.7.3.

A minimum sub-region radius $\delta > 0$ forces the measure of $\mathcal{L}_k$ to remain positive and $\{\omega_L\}$ to be interior points of $\mathcal{L}_k$ for all k (see Section 3.4.2). This is crucial for convergence and guarantees

$$\{x + \delta[-1, 1]^n\} \subset \mathcal{L}_k \text{ for all } x \in \{\omega_L\}. \tag{6.16}$$

In particular, (6.16) holds for both z_k and x_k for all k.

To sample $\mathcal{L}_k$ the greedy sampling strategy proposed in Section 3.5 is used, whereby samples are drawn directly from $\mathcal{L}_k$ using a near uniform distribution. No acceptance/rejection sampling is required because the CARTopt filter method is unconstrained. After N samples are drawn from $\mathcal{L}_k$ the batch X_k is complete and sampling is finished. Most importantly, with reference to (6.16), there exists a non-zero probability of sampling a neighborhood of z_k or x_k for all k, given by

$$Pr(x \in \{x_k + \delta[-1, 1]^n\}) \geq \frac{(2\delta)^n}{|\{\omega_L\}|.m(\mathcal{L}_k)} > 0. \tag{6.17}$$

Before f and $\mathfrak{h}$ are evaluated at the new batch of points, the Householder transform is applied to X_k. This applies the inverse transform converting each point to the original sample space.

6.2.5 Penalty Parameter

The sequence of iterates $\{z_k\}$ generated by the CARTopt filter method is obtained by selecting the most feasible element of $T_k \cup X_k$ that minimizes

$$g_k(x, \sigma_k) = f(x) + \sigma_k \mathfrak{h}(x), \tag{6.18}$$

where σ_k is a penalty parameter. With reference to (6.18) the reader may mistake the CARTopt filter method as a penalty function method in disguise. Clearly, generating a feasible iterate is of primary interest, but simply updating the most feasible iterate can be problematic — especially when a non-empty training data set exists from which iterates can be selected. Choosing the most feasible iterates does not preclude, for example, generating a sequence of iterates such that $\{f(z_k)\}$ is monotonically increasing and $\{\mathfrak{h}(z_k)\}$ is monotonically decreasing. This problem was also evident to Fletcher *et al.* and a North-West corner rule is introduced to the filter [26]. This rule requires sufficient reduction in $\mathfrak{h}$ in the leftmost corner of the filter for an element to be included in the filter. However, such an approach can filter desirable elements if the sufficient reduction parameter is chosen incorrectly.

Here the primary goal is to iteratively update the sloping filter and ultimately drive feasibility. The penalty parameter is used to ensure a sequence of iterates satisfying sufficient decrease in $\mathfrak{h}$ are included in the training data set. However, to promote convergence to a feasible iterate σ is increased from time to time. Here σ is bounded above by $\sigma_{\max} = 1\text{e}{+5}$. It is not necessary force $\sigma_{\max} \to \infty$ in limit as $k \to \infty$, as is common in standard penalty methods. As σ is increased, penalty methods give preference to minimizing $\mathfrak{h}$ rather than f. However, the sequence of least constraint violations $\{x_k\}$ includes this case, where preference to minimizing $\mathfrak{h}$ is made each iteration.

There are many possible heuristics for updating σ. Requiring sufficient reduction in constraint violation at each iteration, for example, would attempt to drive feasibility in $\{z_k\}$. Here elements of the sloping filter are used to approximate a suitable penalty parameter at iteration k. Firstly, a linear best fit (in the least squares sense) to the sloping filter is calculated with slope parameter θ. If $-\theta > \sigma_k$ it suggests the penalty parameter may be too small and σ is increased, otherwise σ remains unchanged. Specifically, if $|\mathcal{F}_k| > 1$ and $\sigma_k < -\theta$

$$\sigma_{k+1} = \min\{\sigma_{\max}, 2\sigma_k, -\theta\}, \tag{6.19}$$

otherwise $\sigma_{k+1} = \sigma_k$. For the special case when $|\mathcal{F}_k| = 1$, any penalty parameter value

would select the single sloping filter element and hence, σ_k remains unchanged.

6.3 Stopping Rule

The stopping rule used here is similar to the rule used for the unconstrained CARTopt instance described in Chapter 4. The interested reader is referred to Chapter 4 for full treatment on this stopping rule. This section outlines necessary changes only. There are two modes of termination for the CARTopt filter algorithm depending on whether a feasible iterate is obtained or not. Firstly, let us consider the case when a feasible iterate is generated and then consider the case when T_k contains infeasible iterates only.

Ideally, convergence to a feasible iterate is of primary interest. However, to generate a feasible iterate the closure of the interior of $\mathcal{N}$ must be a set of positive measure, from which a sample is drawn. Clearly if equality constraints are present this is not possible. If a feasible point is drawn from $\overline{\mathcal{N}^{\mathsf{O}}}$, the training data set T keeps all feasible points generated until at least γ are obtained. Furthermore, of these feasible points the $3N/4$ with the least function values are included in $\{\omega_L\}$. This feasible subset of T is similar to the training data set that the unconstrained CARTopt instance used to solve the NLP satisfying Assumption 62 expressed as the barrier function

$$\mathfrak{B}(x) = \begin{cases} f(x) & \text{if } x \in \mathcal{N} \\ +\infty & \text{otherwise,} \end{cases} \tag{6.20}$$

where $\overline{\mathcal{N}^{\mathsf{O}}} = \mathcal{N}$. Convergence to an essential local minimizer of (6.20) is demonstrated in Chapter 3 under mild conditions (see Theorem 40) provided at least one point with a finite function value is drawn. Furthermore, the stopping rule in Chapter 4 can be applied to barrier functions of the same form as (6.20). Thus, if a feasible point is generated, the existing rule can be applied directly to the CARTopt filter algorithm using the feasible elements of T only. That is, if the probability of reducing f below $f(x_k)-\epsilon$ is sufficiently small and the γ feasible, least function values follow a power law distribution, x_k is the candidate solution for an essential local minimizer of f on $\overline{\mathcal{N}^{\mathsf{O}}}$. The infeasible elements of T are useful in forming the partition, but are not required for the stopping rule.

For the case when no feasible iterates exist ($\mathfrak{h}(x) > 0$ for all $x \in T_{k+1}$), the user can choose whether driving feasibility or minimizing $g_k(x, \sigma_k)$ is of most importance. Firstly, let us consider driving feasibility. With T_{k+1} containing the γ elements with

the least $\mathfrak{h}$ values generated up until iteration k, the probability of reducing $\mathfrak{h}$ further can be approximated. The empirical data set (4.13) in Section 4.3.1 is replaced with

$$Y = \{\mathfrak{h}(\hat{x}_1), \mathfrak{h}(\hat{x}_2), \ldots, \mathfrak{h}(\hat{x}_\gamma) : \hat{x}_i \in T_{k+1}\} \tag{6.21}$$

with $\mathfrak{h}(\hat{x}_i) \leq \mathfrak{h}(\hat{x}_{i+1})$ and $\hat{x}_1 = \arg\min\{\mathfrak{h}(x) : x \in T_{k+1}\} = x_k$. Updating the analysis in Chapter 4 from Section 4.3.1 onwards in the obvious way, an approximation to the probability of reducing the constraint violation below $\mathfrak{h}(x_k) - \epsilon$ is obtained, given by,

$$Pr\left(\mathfrak{h}(x) < \mathfrak{h}(x_k) - \epsilon\right) = \left[\left(\frac{\mathfrak{h}(x_k) - \epsilon - \mathfrak{h}^*_*}{\mathfrak{h}(\hat{x}_\gamma) - \mathfrak{h}^*_*}\right)^{\kappa^*}\right]_+ . \tag{6.22}$$

Here κ^* and $\mathfrak{h}^*_*$ are the optimal power and $\mathfrak{h}_*$ approximation, respectively, obtained during fitting a power law to the empirical data (6.21). If (6.22) is sufficiently small and the set (6.21) follows a power law distribution, then x_k is a candidate solution for an essential local minimizer of the constraint violation function.

Secondly, preference to minimizing $g_k(z, \sigma_k)$ is reflected in the stopping rule by monitoring the γ elements with the least $f + \sigma_k\mathfrak{h}$ values generated up until iteration k. The training data update ensures these elements are always a subset T_{k+1} (see Section 6.2.2). Replacing the empirical data (4.13) with

$$Y = \{g_k(\hat{z}_1), g_k(\hat{z}_2), \ldots, g_k(\hat{z}_\gamma) : \hat{z}_i \in T_{k+1}\} \tag{6.23}$$

with $g_k(\hat{z}_i) \leq g_k(\hat{z}_{i+1})$ and $\hat{z}_1 = \arg\min\{f(z)+\sigma_k\mathfrak{h}(z) : z \in T_{k+1}\} = z_k$, an approximate probability of reducing $g_k(x, \sigma_k)$ below $g_k(z_k) - \epsilon$ is obtained. Replacing each instance of $\mathfrak{h}$ and x with g_k and z in (6.22) gives the probability, where κ^* and g^*_* are the optimal parameters obtained from fitting (6.23). If this probability is sufficiently small and the set (6.23) follows a power law distribution, then z_k is a candidate solution for an essential local minimizer of $g_k(z, \sigma_*)$, where $\sigma_k \to \sigma_*$ as $k \to \infty$.

The final method of termination is a resource based stopping rule. The user can choose a predefined resource quantity to terminate the algorithm if the rules above fail to halt within user limits. Here a maximum number of iterations $k_{\max} = \max\{1000, 100n^2\}$ is used to ensure the CARTopt filter algorithm always terminates.

Numerical simulations performed by the author indicated a preference towards termination based on minimizing $g_k(x, \sigma_k)$ as opposed to minimizing $\mathfrak{h}$ when no feasible iterates were generated. Minimizing with respect to $\mathfrak{h}$ fails to include valuable information about f, terminating the algorithm at approximations to essential local minimizers

of $\mathfrak{h}$ far from the solution to the NLP. This method of termination is considered no further.

6.4 Convergence Analysis

This section analyzes the convergence properties of the CARTopt filter algorithm. The stopping conditions are removed to allow us to examine the asymptotic properties of the sequence of iterates generated by the algorithm. In addition, the lower bound on constraint violation $\mathfrak{h}_{\min}$ is set to zero. The stopping rule and $\mathfrak{h}_{\min}$ are included in the algorithm from a practical point of view.

Theorem 68. *The sequences of iterates $\{z_k\}$ and $\{x_k\}$ generated by the CARTopt filter algorithm are infinite sequences.*

Proof. For both $\{z_k\}$ and $\{x_k\}$ to be infinite sequences, Step 3 of the CARTopt filter algorithm must be a finite process.

With the cardinality of X_k finite for all k, steps 3(c,d) are finite processes. The cardinality of the training data set is bounded by

$$|T_k| \leq \max\{2\gamma + N, 2N \max\{1, n-1\}\} \tag{6.24}$$

for all iterations. Thus, it follows directly from Corollary 34 that Step 3(b) is a finite process.

The sloping filter is a subset of T_k (finite set) which is calculated using a finite set (6.7) of slope parameters and is thus a finite process. Furthermore, with the training data bounded above by (6.24), the training data update and classification are both finite processes. Thus, Step 3(a) is also a finite process. □

In order to establish convergence, similar assumptions to those used in the unconstrained instance of CARTopt are required.

Assumption 69. *The following conditions hold:*

(a) The objective function f and the constraint violation function $\mathfrak{h}$ are both lower semi-continuous;

(b) The points at which f and $\mathfrak{h}$ are evaluated at lie in a compact subset of $\mathbb{R}^n$; and

(c) The sequences of function values $\{f(z_k)\}$ *and* $\{f(x_k)\}$ *is bounded below.*

The first assumption ensures that $\liminf_{z \to z_*} f(z) \geq f(z_*)$ for all cluster points. The other assumptions ensure $\{z_k\}$ is bounded and excludes the case where $f(z_k) \to -\infty$ as $k \to \infty$ and similarly for $\{x_k\}$. The constraint violation function is automatically bounded below due to its non-negativity.

The first result is concerned with the constraint violation function. Examining cluster points x_* in the sequence $\{x_k\}$ shows that $\mathfrak{h}(x_*)$ satisfies locally optimal properties.

Theorem 70. *Let Assumption 69 hold. If* x_* *is a cluster point of the sequence* $\{x_k\}$*, then* x_* *is an essential local minimizer of* $\mathfrak{h}$ *with probability one.*

Proof. Assumption 69 and Theorem 68 ensure the existence of cluster points in $\{x_k\}$.

If x_* is feasible $\mathfrak{h}(x_*) = 0$. The non-negativity of $\mathfrak{h}$ implies the set of lower points is the empty set. Thus, x_* is an essential local minimizer of $\mathfrak{h}$.

The proof for the case when x_* is infeasible is a direct consequence of Theorem 38. □

The next result shows that cluster points z_* in the sequence $\{z_k\}$ have locally optimal properties.

Theorem 71. *Let Assumption 69 hold. If* z_* *is a cluster point of the sequence* $\{z_k\}$ *and* σ_* *is the terminating penalty parameter, then* z_* *is an essential local minimizer of* $f + \sigma_* \mathfrak{h}$ *with probability one.*

Proof. Assumption 69 and Theorem 68 ensure the existence of cluster points in $\{z_k\}$.

The remainder of the proof is a direct consequence of Theorem 38. □

The following corollaries show convergence to feasible essential local minimizers when the feasible region satisfies Assumption 62. That is, for all $x \in \mathcal{N}$, there exists an $\epsilon > 0$ such that $\mathcal{B}(x, \epsilon) \cap \mathcal{N}$ with positive Lebesgue measure. Clearly, Assumption 62 removes the possibility of including equality constraints in the NLP.

Definition 72. (Feasible essential local minimizer). *Let Assumption 62 hold. A point* $x_* \in \mathcal{N}$ *for which the set*

$$\mathfrak{E}(x_*, \epsilon) = \{x \in \mathcal{N} : f(x) < f(x_*) \text{ and } \|x - x_*\| < \epsilon\}$$

has Lebesgue measure zero for all sufficiently small positive ϵ *is called a feasible essential local minimizer of* f*.*

Corollary 73. *Let Assumptions 69 and 62 hold. If z_* is a feasible cluster point of the sequence $\{z_k\}$, then z_* is a feasible essential local minimizer of f with probability one.*

Proof. Assumption 69 and Theorem 68 ensure the existence of cluster points in $\{z_k\}$.

The proof is similar to Theorem 38. Assumption 62 ensures that

$$m(\mathcal{B}(z_*, \epsilon) \cap \mathcal{N}) > 0 \tag{6.25}$$

for all $\epsilon > 0$. Thus, (3.31) is a set of positive measure. Replacing $[-1, 1]^n$ in both (3.31) and (3.33) with $\mathcal{N}$ gives the desired proof. □

Corollary 74. *Let Assumptions 69 and 62 hold. If x_* is a feasible cluster point of the sequence $\{x_k\}$, then x_* is a feasible essential local minimizer of f with probability one.*

Proof. The proof is similar to Corollary 73. Replacing $\{z_k\}$ with $\{x_k\}$ and z_* in (6.25) with x_* gives the desired result. □

The final two results show that Assumption 62 can be relaxed to a local property of $\mathcal{N}$, rather than a global one, without effecting convergence.

Corollary 75. *Let Assumption 69 hold. If z_* is a feasible cluster point of the sequence $\{z_k\}$ such that $\overline{\{\mathcal{B}(z_*, \epsilon) \cap \mathcal{N}\}^{\mathsf{O}}} = \{\mathcal{B}(z_*, \epsilon) \cap \mathcal{N}\}$, then z_* is a feasible essential local minimizer of f with probability one.*

Proof. The proof is similar to Corollary 73. For all $\epsilon > 0$, if $\overline{\{\mathcal{B}(z_*, \epsilon) \cap \mathcal{N}\}^{\mathsf{O}}} = \{\mathcal{B}(z_*, \epsilon) \cap \mathcal{N}\}$ then

$$m(\mathcal{B}(z_*, \zeta) \cap \mathcal{N}) > 0$$

for all $\zeta \leq \epsilon$. The result then follows directly from Corollary 73. □

Corollary 76. *Let Assumption 69 hold. If x_* is a feasible cluster point of the sequence $\{x_k\}$ such that $\overline{\{\mathcal{B}(x_*, \epsilon) \cap \mathcal{N}\}^{\mathsf{O}}} = \{\mathcal{B}(x_*, \epsilon) \cap \mathcal{N}\}$, then x_* is a feasible essential local minimizer of f with probability one.*

Proof. The proof is similar to Corollary 75. Replacing z_* with x_*, the result follows directly from Corollary 74. □

Chapter 7

Empirical Testing of Algorithms

This chapter empirically investigates the performance of the algorithms proposed in the preceding chapters of this thesis. Numerical simulations are important to verify that theoretical results are achieved in practice. Test problems are taken from Schittkowski *et al.* [34, 68], Moré *et al.* [51] and Luksan *et al.* [45]. The interested reader is referred to Appendix A for further details on the test problems considered here. The algorithms also performed well on a selection of smooth test problems, but only nonsmooth results are presented here as these problems are of primary interest.

To measure performance of the various algorithms the following considerations were made. Firstly, an algorithm that produces an accurate approximation to the solution of each test problem considered is better than an algorithm that does not. Furthermore, an algorithm that produces more accurate approximate solutions to each problem is better than one that provides less accurate values. Finally, an algorithm that requires fewer function evaluations to achieve accurate approximations to the solutions of each problem is better than one that requires more. It is the author's view that although keeping the number of function evaluations low is important, the primary goal of an algorithm is to produce accurate answers on a range of problems.

Most of the nonsmooth problems considered here are nonlinear least squares problems, where the squares are replaced by absolute values. Furthermore, these problems have optimal function values of zero. However, these modified functions can make the results look deceptively poor. A final function value of 1e-5 on the nonsmooth function, for example, corresponds to a function value of approximately 1e-10 on the original problem. Therefore, any final function value less than 1e-3 is considered acceptable here. The interested reader is referred to Appendix A for full details on these problems.

All the algorithms and test problems have been coded in Matlab 7.9.0 [47]. The interested reader is referred to Appendix B for complete algorithm codes. The test problem codes are available from the author upon request. The stochastic algorithms use the uniform pseudorandom number generator in Matlab called RAND. The sequence of numbers produced by RAND is determined by the internal state of the generator that underlies RAND. To prevent results being generated from similar subsequences, the initial state of the sequence was set to the sum(100*CLOCK) at each session, where CLOCK is a six element Matlab vector [year, month, day, hour, minute, second] in decimal form. All the stochastic results are averaged over ten runs to give a measure of average performance.

Firstly, the bound constrained instance of CARTopt is empirically tested. Secondly, the algorithms for unconstrained optimization are considered. These are the unconstrained CARTopt instance, and the strictly descent and non-descent versions of the Hooke and Jeeves / CARTopt hybrid algorithm. Finally, the unconstrained CARTopt instance and the CARTopt filter algorithm are empirically tested on constrained nonlinear programming problems.

7.1 Bound Constrained Optimization using the CARTopt Method

The algorithm was implemented with a batch size $N = 20$, cardinality of low points $|\{\omega_L\}| = 0.8N = 16$ and a minimum sub-region radius $\delta =$ 1e-10. Numerical simulations performed by the author found that the impurity condition (see Definition 14) had no significant impact on the numerical results and only pure partitions are considered hereafter. That is, low sub-regions of the CART partition contain elements from $\{\omega_L\}$ only. However, if the user chooses a larger batch size and/or a different classification scheme, the impurity condition may be advantageous and simplify the partition. The greedy sampling strategy was used, whereby all samples are drawn directly from low sub-regions only. Non-greedy methods were also investigated by the author but all required more function evaluations to obtain solutions of similar accuracy to those presented here.

The values $\epsilon =$ 1e-8 and $\beta =$ 1e-6 were used for the stopping rule. The stopping rule successfully terminated the CARTopt algorithm before the maximum number of iterations was reached on all problems considered.

The bound constrained optimization region is defined by the hypercube

$$(x_0 + x_*)/2 + h[-1, 1]^n, \tag{7.1}$$

where x_0 and x_* are the optimization starting point and minimizer for the particular problem. Here the hypercube radius h is chosen so that both x_0 and x_* are interior points of the hypercube. The interested reader is referred to Appendix A for exact values.

Table 7.1 lists the results for the nonsmooth problems considered. The legend for this table is defined as follows. The first two columns list the function and its dimension. The columns headed with f_* and 'nf' list the absolute error in function value at the final iterate ($|f - f_*|$) and the number of function evaluations respectively. This notation is used consistently throughout this chapter. The final column lists results from applying Pure Random Search (PRS) in (7.1) using 20 000 function evaluations.

Both the stochastic and deterministic instances of CARTopt were superior to PRS, requiring fewer function evaluations to produce far more accurate approximations to the solution of each problem. Although the deterministic instance does not have a convergence proof, all problems were solved by the deterministic instance with similar performance to the stochastic instance. Most importantly, no failures were observed when generating the numerical results. These results verify that theoretical convergence is achieved in practice for the bound constrained CARTopt algorithm.

7.2 Unconstrained Optimization

In this section the unconstrained CARTopt algorithm and Hooke and Jeeves / CARTopt hybrid algorithm (with and without uphill steps) are tested on nonsmooth unconstrained optimization problems. The deterministic instances of these algorithms were also tested by the author and each algorithm performed similarly to the results presented here. However, convergence was not demonstrated for these algorithms and these results are not presented.

The Hooke and Jeeves / CARTopt hybrid algorithm was implemented with an initial mesh size $h_0 = e/2$, a minimum localized global optimization radius h_Ω = 1e-4 and a minimum mesh size $h_{\min}$ = 1e-8. This apparently strange h_0 value was chosen over the popular choice of $h_0 = 1$ because the latter allowed the algorithm to step exactly to the solution on some of the test problems considered. This gave a misleading impression on the performance of the algorithm. The standard optimization starting point x_0 (see

Table 7.1: Bound Constrained Optimization using CARTopt and Pure Random Search

		Stochastic CARTopt		Deterministic CARTopt		PRS
Problem	n	f_*	nf	f_*	nf	f_*
Beale	2	4e-9	986	8e-10	1031	0.03
CB2	2	5e-9	835	5e-9	732	0.01
QL	2	7e-10	912	2e-10	887	0.01
Rosenbrock	2	3e-9	1102	8e-10	1085	0.04
Wolfe	2	1e-9	957	8e-10	903	0.15
Gulf	3	7e-9	1869	8e-9	1896	3.90
240	3	1e-8	1800	7e-9	1862	6.01
Helical Valley	3	7e-9	1722	3e-9	1783	0.41
Powell	4	1e-8	2329	7e-9	2681	2.14
261	4	9e-9	3483	6e-9	3547	0.60
Rosen-Suzuki	4	9e-5	5359	5e-7	4591	0.78
Trigonometric	5	2e-8	3945	1e-8	3367	0.84
Variably Dim.	8	4e-8	11508	3e-8	11137	1.43
291 (Quartic)	10	9e-9	5152	7e-9	5520	2.22

Appendix A) for each problem was used as the initial point for each problem. The localized global optimization phase uses the (stochastic) bound constrained CARTopt instance.

The unconstrained stochastic CARTopt instance was implemented with the parameters defined as in the previous section. The initial hypercube search region size was chosen relative to the first localized global phase search region in the Hooke and Jeeves / CARTopt Hybrid algorithm, given by

$$x_0 + \frac{e}{2}\sqrt{n}[-1, 1]^n.$$

This choice allows us to make fair comparisons between the two methods.

Table 7.2 lists the results for the nonsmooth problems considered. The legend for this table is defined as follows. Columns headed with 'nf HJ' list the number of function evaluations performed during the Hooke and Jeeves phase of the algorithm up until the final iterate. Columns headed with 'term' indicate the method of termination. This can be either through the CARTopt algorithm 'CART' or if the minimum mesh size is reached 'HJ'. The multicolumns headed with 'Descent' and 'Non-Descent' list the results for the strictly descent and non-descent versions of the Hooke and Jeeves algorithm.

The Hooke and Jeeves / CARTopt hybrid algorithm and the unconstrained CARTopt instance solved all the problems considered to the desired standard. The results show little difference between the strictly descent and non-descent versions of the Hooke and Jeeves algorithm. The hybrid algorithm terminated when the minimum Hooke and Jeeves mesh size was reached on approximately half the problems. Otherwise the algorithm terminated through the CARTopt algorithm. Most of the computational effort was conducted in the localized global optimization phase of the hybrid algorithm. The only exception was the Gulf problem, where a huge number of Hooke and Jeeves iterations were conducted between locating grid local minima. The unconstrained CARTopt instance produced the ten most accurate approximate solutions to the fourteen problems considered, using a similar number of function evaluations. Thus, the unconstrained CARTopt instance is the preferred method.

The previous results show that the CARTopt based methods are effective, but are they competitive in practice? To show these methods are competitive a comparison between the unconstrained CARTopt instance and two direct search methods for nonsmooth unconstrained optimization from Price, Reale and Robertson [61, 62] is given in Table 7.3. The algorithm in [61] is a frame based algorithm which performs a ray search along either a direct search quasi-Newton direction, or along a ray through the best frame point at each iteration. Random perturbations of the frames from time to time gives convergence on nonsmooth problems. The algorithm in [62] is similar to the Hooke and Jeeves / CARTopt hybrid algorithm presented in Chapter 5, using a series of local and localized global optimization phases. The classical Hooke and Jeeves algorithm is used in the local phase and the DIRECT algorithm of Jones, Perttunen and Stuckman [37] in the localized global optimization phase. This algorithm is deterministic and is provably convergent on nonsmooth problems [62].

The CARTopt algorithm was superior to the algorithm from [61], producing more accurate approximate solutions on all problems and only required more function evaluations on the Gulf problem. However, a more accurate approximate solution was produced by CARTopt on the Gulf problem. For all other problems approximately one third of the function evaluations were required.

In comparison to the algorithm from [62], a slightly more accurate approximate solution to Helical Valley problem was obtained, otherwise CARTopt produced more accurate approximate solutions. Only the Rosenbrock function required more function evaluations although a more accurate approximate solution was produced. The CARTopt algorithm performed much better than the algorithm from [62] in the higher dimensional problems, particularly on the Variably Dimensioned problem requiring less

Table 7.2: Unconstrained Optimization using Hooke and Jeeves / CARTopt Hybrid Algorithm and the Unconstrained CARTopt Algorithm

		HJ/CARTopt								CARTopt	
		Descent				Non-Descent				(unconstrained)	
Problem	n	f_*	nf HJ	nf	term	f_*	nf HJ	nf	term	f_*	nf
Beale	2	8e-9	157	950	HJ	7e-9	158	935	HJ	1e-9	1061
CB2	2	5e-9	151	979	CART	5e-9	168	1056	CART	5e-9	837
QL	2	9e-10	144	1177	CART	1e-9	321	1560	CART	2e-9	874
Rosenbrock	2	6e-9	250	1213	HJ	3e-9	168	1152	HJ	3e-9	1240
Wolfe	2	2e-9	169	974	HJ	2e-9	142	904	HJ	3e-10	966
Gulf	3	7e-8	11691	18041	CART	7e-8	16380	21406	CART	1e-6	17252
240	3	9e-9	389	1462	HJ	9e-9	476	1572	HJ	5e-9	1948
Helical Valley	3	2e-8	308	1842	HJ	2e-8	328	1904	HJ	4e-9	1856
Powell	4	3e-8	522	2669	HJ	5e-8	509	2658	HJ	7e-9	2725
261	4	1e-8	528	3751	HJ	2e-8	502	3510	HJ	9e-9	3718
Rosen-Suzuki	4	1e-5	1610	8996	CART	1e-5	1178	8157	CART	4e-4	5434
Trigonometric	5	6e-8	750	3745	HJ	6e-8	798	3870	HJ	2e-8	4652
Variably Dim.	8	1e-7	1579	12712	CART	2e-7	1590	12642	CART	6e-9	9218
291 (Quartic)	10	4e-9	1435	4124	CART	1e-8	1556	4181	CART	1e-8	6257

Table 7.3: Comparison with two other Direct Search Methods for Nonsmooth Unconstrained Optimization

		Results from [61]		Results from [62]		CARTopt (Unconstrained)	
Problem	n	f_*	nf	f_*	nf	f_*	nf
Beale	2	4e-8	3638	2e-8	1119	1e-9	1061
Rosenbrock	2	5e-8	4438	2e-8	1154	3e-9	1240
Gulf	3	1e-5	15583	6e-6	31306	1e-6	17252
Helical Valley	3	7e-8	8406	1e-9	2773	4e-9	1856
Powell	4	4e-7	11074	3e-3	3659	7e-9	2725
Trigonometric	5	5e-8	14209	4e-8	6678	2e-8	4652
Variably Dim.	8	2e-7	34679	5e-7	55647	6e-9	9218

than one fifth the function evaluations to produce an approximate solution two orders of magnitude better. It is the author's opinion that this is largely due to the fact that the algorithm in [62] is deterministic, whereas CARTopt is stochastic and explores higher dimensions more efficiently.

7.3 Nonlinear Programming using the CARTopt Filter Method

The CARTopt filter algorithm was implemented with the parameters similar to the CARTopt algorithm. The only difference is the slightly larger set of low points $|\{\omega_L\}| = N = 20$. The author investigated the use of various constraint violation functions but found the squared 2-norm violation to perform the best, given by

$$\mathfrak{h}(x) = \sum_{i=1}^{m} [c_i(x)]_+^2. \tag{7.2}$$

Here all results presented use (7.2). The test problems are taken from Schittkowski *et al.* [34, 68] and [45]. These problems have linear and nonlinear constraints. Furthermore, the objective function f is made nonsmooth by replacing the sum of squares formulation with a sum of absolute values in all cases. Hence, an approximate solution presented here of the order 1e-5 is approximately of the order 1e-10 on the smooth version. The interested reader is referred to Appendix A for further details on these problems and their formulation.

7.3.1 Inequality Constrained Problems

In this section three methods for solving nonsmooth inequality constrained nonlinear programmes are considered. These are the CARTopt filter algorithm, the unconstrained CARTopt algorithm — where the problem is formulated as a barrier function — and using Pure Random Search (PRS). The closure of the interior of the feasible region $\mathcal{N}$ is $\mathcal{N}$ itself for the problems considered. Thus, $\mathcal{N}$ is a set of positive measure.

The initial search region for both the filter and unconstrained CARTopt algorithms is given by the hypercube

$$(x_0 + x_*)/2 + h[-1, 1]^n, \tag{7.3}$$

where x_0 and x_* are the optimization starting point and minimizer for the particular problem. The hypercube radius h is chosen so that both x_0 and x_* are interior points of the hypercube. The interested reader is referred to Appendix A for exact values. This choice allows a comparison between PRS and the CARTopt methods to be made.

The inequality constrained problems considered here can be solved using the unconstrained CARTopt instance when the problem is expressed as a barrier function. Specifically,

$$\mathfrak{B}(x) = \begin{cases} f(x) & \text{if } \mathfrak{h}(x) = 0 \\ +\infty & \text{otherwise.} \end{cases} \tag{7.4}$$

Clearly, if equality constraints are present, the probability of sampling the feasible region is zero ($m(\mathcal{N}) = 0$) and thus, all function values are infinite with probability one. Equality constrained problems are considered in the next section. It is not necessary to evaluate f at infeasible points, rather the barrier function assigns the value $+\infty$. For comparison purposes, each barrier function evaluation is considered as one function evaluation, even though considerably less computation may be required to evaluate (7.4).

Table 7.4 lists the results for the nonsmooth nonlinear programming problems considered. The legend is defined as follows. The first three columns list the function name, dimension and number of inequality constraints present. The values in parentheses indicate the number of constraints that are nonlinear. The columns headed with σ_* list the penalty parameter value at the final iterate and values in parentheses state the bound on the penalty parameter $\sigma_{\max}$. The column headed 'nf_∞' lists the number of times the barrier function assigned to value $+\infty$ without evaluating f directly. The final column lists results from applying PRS in (7.3) using 20 000 function evaluations. The constraint violation was zero at all the final iterates and hence, are not listed.

Both the CARTopt filter and barrier algorithms were superior to PRS, requiring fewer function evaluations to obtain far more accurate approximate solutions to each problem considered. The barrier method performed better than the filter method on seven of test functions, requiring fewer function evaluations to obtain similar accuracy approximate solutions. Furthermore, with reference to the 'nf_∞' column, although more barrier function evaluations were required to solve test problem 225, fewer f evaluations were required. The barrier method assigns the value $+\infty$ rather than evaluating f directly and thus, the 'nf' column for the barrier method requires the computational effort of approximately $nf - nf_\infty$ function evaluations — less computational effort compared to the 'nf' column for the filter method.

The CARTopt filter method performed better on three of the problems considered. Setting a larger bound on the penalty parameter of $\sigma_{\max} = 1\text{e}{+10}$ gave better results for the filter algorithm. This allowed the method to perform like a barrier method in the final stages of sampling if a constraint(s) were active at the solution, producing good approximate solutions with fewer f evaluations.

A close inspection of the training data set on the problems for which the filter method was superior showed that classifying some infeasible points as low was advantageous. This meant subsets of the infeasible region were classified as low and hence, were sampled further during the next iteration. This allowed the algorithm to approach the solution from the infeasible region. In contrast, the barrier method assigned the value $+\infty$ to such points which resulted in the solution being bounded away from low sub-regions. The algorithm would then make slow progress toward the solution and ultimately terminated at modest accuracy when compared to the filter algorithm. This behavior is illustrated in Figure 7.1 for a simple nonlinear programming problem in 2 dimensions. It is clear that the barrier approach fails to include the optimal point x_* in the low sub-region. Whereas, the filter based approach classifies four additional sloping filter elements as low, producing a desirable partition which includes x_* in the low sub-regions. Thus, there is a non-zero probability of sampling a neighborhood of x_* during the next iteration for the filter approach and there is zero probability for the barrier approach.

7.3.2 Equality Constrained Problems

In this section the CARTopt filter algorithm is used to solve nonlinear programming problems with equality constraints present. The algorithm setup differs from the inequality constrained case with a lower bound of $\mathfrak{h}_{\min} = 1\text{e-}10$ placed on constraint

Table 7.4: Nonlinear Programming using the CARTopt Filter Algorithm, Barrier Method and Pure Random Search

			CARTopt Filter						CARTopt Barrier			PRS
Problem	n	IC	f_*	σ_* (1e+5)	nf	f_*	σ_* (1e+10)	nf	f_*	nf_∞	nf	f_*
225	2	5(4)	3e-9	1e+5	1703	5e-9	5e+7	1501	1e-9	884	2136	0.03
228	2	2(1)	1e-9	34	1576	-	-	-	2e-9	300	1271	0.02
MAD1	2	1	8e-9	70	1406	-	-	-	6e-9	244	1237	8e-3
MAD2	2	1	2e-9	40	1142	-	-	-	2e-9	165	929	2e-3
MAD5	2	1	3e-9	40	1738	-	-	-	4e-9	337	1495	0.01
249	3	2(1)	6e-9	1e+5	3038	4e-9	8e+6	2528	7e-9	264	1847	0.23
43	4	3(3)	3e-8	1e+5	14500	2e-8	2e+8	7501	7e-3	1479	5155	0.97
B1	4	4	3e-8	1e+5	9235	2e-8	5e+8	6415	7e-8	1099	3910	4.9
B2	6	6	6e-8	1e+5	19002	3e-8	2e+9	14156	2e-8	2042	7975	11.5
Pentagon	6	15	5e-8	1e+5	17877	5e-8	1e+8	14423	1e-4	2980	9555	0.22

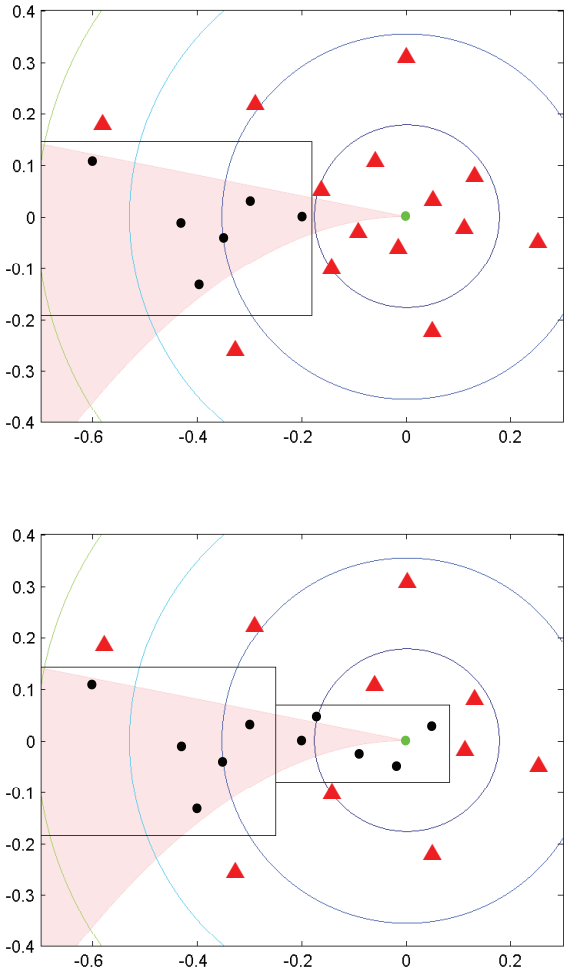


Figure 7.1: The figures above show the level curves of $f = \|x\|$, along with two inequality constraints $c_1 = x_2 + x_1^2 \leq 0$ and $c_2 = x_2 + 0.2x_1 \leq 0$. The feasible region is shaded red and solution x_* is denoted •. The training data set $T = [\omega_L, \omega_H]$, with $x \in \{w_L\}$ and $z \in \{w_H\}$ denoted •, ▲ respectively, is shown. The low sub-region bounds from the partition are shown as black lines. The first figure shows the barrier approach and the second shows the filter approach.

violation. The initial search region is the same hypercube defined by (7.3). However, if $(x_0 - x_*)/2$ is feasible, the hypercube center is perturbed slightly, otherwise misleading and uninteresting results about the most feasible iterate are obtained. Each CARTopt algorithm evaluates f and $\mathfrak{h}$ at $(x_0 - x_*)/2$ and thus, if the hypercube center is feasible, $x_k = (x_0 - x_*)/2$ for all iterations because the probability of generating a feasible iterate using CARTopt is zero. The algorithm terminates with respect to the sequence $\{z_k\}$, the iterates that minimize $f + \sigma\mathfrak{h}$.

Table 7.5 lists the results for the nonsmooth equality constrained problems considered. The legend is similar to Table 7.4 and the additional columns are defined as

Table 7.5: Nonlinear Programming using the CARTopt Filter Algorithm and Pure Random Search

				CARTopt Filter Method						PRS	
				z_*			x_*			σ = 1e+5	
Problem	n	EC	IC	f_*	$\mathfrak{h}_*$	σ_*	f_*	$\mathfrak{h}_*$	nf	f_*	$\mathfrak{h}_*$
6	2	1(1)	-	2e-9	7e-14	5e+3	2e-6	2e-16	1243	0.39	7e-7
28	3	1	-	4e-9	1e-13	9e+4	1e-4	2e-18	4225	2.43	0.08
32	3	1	4(1)	7e-6	8e-11	1e+5	1e-5	8e-17	2965	1.77	0.15
46	5	2(2)	-	2e-8	2e-14	1e+5	1e-7	1e-16	9038	2.28	0.24
48	5	2	-	1e-8	2e-13	4e+4	1e-7	4e-16	8027	7.32	15.6
51	5	3	-	1e-8	1e-14	8e+4	3e-8	3e-16	6676	3.20	0.24
MAD6	7	2	9	3e-6	3e-10	1e+4	5e-5	2e-13	14170	3.18	1.20

follows. The column headed with 'EC' lists the number of equality constraints present and values in parentheses indicate the number that are nonlinear. Columns headed with $\mathfrak{h}_*$ list the constraint violation at the final iterate. The multicolumn headings z_* and x_* list the terminating f, $\mathfrak{h}$ and σ values of the sequences $\{z_k\}$ and $\{x_k\}$. The last two columns list the results from applying PRS in (7.3) for 20 000 function evaluations. The terminating iterate minimizes $f + \sigma\mathfrak{h}$, where the maximum penalty σ = 1e+5 is used. Values of σ less than 1e+5 were tested but did not change the optimal iterate generated by PRS on all problems considered.

The CARTopt filter algorithm was superior to PRS, requiring fewer function evaluations to obtain far more accurate approximate solutions. The final iterate of $\{z_k\}$ produced high accuracy approximations to the solution of each problem, with absolute errors in f less than 1e-6 and constraint violations less than 1e-10. Whereas minimizing with respect to $\mathfrak{h}$ (the sequence $\{x_k\}$) produced more feasible final iterates, but with less accurate absolute errors in f.

7.4 Concluding Remarks

The numerical results presented show the CARTopt algorithms are effective at solving a variety of nonsmooth optimization problems. High accuracy approximate solutions have been produced on both constrained and unconstrained optimization problems ranging in dimension from $n = 2$ to 10. Clearly, random search CARTopt methods are vastly superior to PRS. Comparison with existing direct search nonsmooth optimization methods show the CARTopt methods are competitive in practice.

The assessment of efficiency (and comparison with methods in [61] and [62]) is based on the number of function evaluations required to solve a problem to the desired accuracy. The author is aware that the reader might be interested in how fast (in terms of computer time) the algorithms are. Clearly, computer time is dependent on the machine used, number of persons sharing a server, the programming language used and the programming skills of the programmer. The results were generated on my (modest) personal laptop, which has a 1.8GHz processor and 3GB of RAM. To give an indication of speed, the Rosenbrock function ($n = 2$) took less than two seconds to solve and the Trigonometric function ($n = 5$) took less than thirty seconds to solve. Each problem considered in this thesis took less than two minutes to solve.

Chapter 8

Summary and Conclusions

A basic introduction to both local and global optimization was presented in Chapter 1 along with a survey of algorithms. In addition, a basic introduction to the theory of positive bases was presented. Most importantly, Chapter 1 showed that ensuring the non-negativity of the Clarke derivative in some, or all, directions at cluster points of an algorithm is only a partial result on nonsmooth problems and does not preclude the existence of descent directions at such cluster points. A new algorithmic framework was presented which replaced the Clarke derivative approach with one consisting of a series of local and localized global optimization phases.

The localized global optimization phase was considered in Chapter 2. Stochastic methods were chosen over deterministic methods because successive search regions potentially overlap. This overlap may be problematic for deterministic methods, but not for stochastic methods. A review of partitioning random search methods was presented and a new algorithm, APRS, was proposed. A particular partitioning technique using classification and regression trees (CART) was also presented. The CART partition has the desirable property that further samples can be drawn directly from subsets of the partition. The APRS algorithm is quite flexible and forms a new partition at each iteration irrespective of the previous partition. Furthermore, successive partitions are not necessarily nested. Under mild conditions, convergence to an essential global minimum with probability one was demonstrated when f was assumed to be nonsmooth or discontinuous.

Chapter 3 extended the CART partitioning ideas presented in Chapter 2 into a nonsmooth local optimization algorithm called CARTopt. A particular classification and updating technique was applied to the training data set T to promote clustering in $\{\omega_L\}$. Also, an invertible transformation was applied to T, which potentially simplified

the partition. The partition defined a bounded subset of $\mathbb{R}^n$ where f was presumed to be low, from which further samples were drawn. Alternating between partition and sampling phases proved to be an effective method for bound constrained and unconstrained nonsmooth optimization. Convergence to an essential local minimizer of f with probability one was demonstrated under mild conditions, where f was assumed to be nonsmooth or discontinuous. Deterministic instances of these algorithms were also presented which used the Halton sequence to sample the approximate level set $\mathcal{L}$, instead of uniform random sampling, to promote more evenly distributed points in $\mathcal{L}$. Convergence was not proved for these deterministic instances.

A stopping rule for the CARTopt algorithms was presented in Chapter 4. If the distribution of the γ least function values generated by CARTopt followed a power law and the probability of reducing f further was sufficiently small, the algorithm terminated. This rule ensured that the strong theoretical results of Chapter 3 were achieved in practice giving practical convergence to estimates of essential local minimizers of f.

A Hooke and Jeeves / CARTopt hybrid algorithm was presented in Chapter 5. The hybrid algorithm uses an altered Hooke and Jeeves algorithm as a local search phase. The Hooke and Jeeves phase operates on a transformed grid and uphill steps are permitted under certain conditions. A localized global optimization phase was conducted in the neighborhood of a grid local minimizer, x_k, found in the Hooke and Jeeves phase. This localized global phase used the bound constrained CARTopt algorithm to locate and sample promising sub-regions in the neighborhood of x_k. If descent was found, the method reverts back to the Hooke and Jeeves phase. Otherwise, x_k was shown to be an essential local minimizer f with probability one, where f was assumed to be nonsmooth or discontinuous. Convergence to a local minimum was also demonstrated when f was assumed to be smooth.

Chapter 6 extended the unconstrained CARTopt instance into an algorithm for constrained nonsmooth nonlinear programming problems. The concept of a sloping filter was introduced and used to define a new training data set from which a partition on $\mathbb{R}^n$ was formed. Convergence to essential local minimizers of the constraint violation function $\mathfrak{h}$ and a penalty function g_k with probability one was demonstrated under mild conditions when both $\mathfrak{h}$ and g_k were assumed to be nonsmooth.

Numerical simulations reported in Chapter 7 showed that the CARTopt methods solved all the test problems considered and thus, seems to be robust in practice. Comparison with two existing direct search methods showed that the bound constrained and unconstrained CARTopt methods were competitive in practice. Furthermore, all the CARTopt methods were vastly superior to Pure Random Search on all problems

considered.

8.1 Where to Next?

In this section the author highlights some areas that could benefit from further research.

Classification

The classification imposed on the training data set T used in this thesis consisted of two categories $\{\omega_L\}$ and $\{\omega_H\}$ — points with relatively low and high function values. It would be interesting to include more stratification in T, for example, three categories giving more detail in the relatively low points. In addition, different classifiers could be investigated, for example, Multivariate Adaptive Regression Splines (MARS) [27]. With each classifier producing a different partition on $\mathbb{R}^n$, many possibilities exist.

Sampling Phase

At each iteration a batch of points was drawn from the approximate level set $\mathcal{L}$ using a near uniform distribution over $\mathcal{L}$. Thus, more samples are drawn from the larger sub-regions of the partition with probability one. It would be interesting to consider different sampling strategies, for example, drawing more samples from sub-regions with the least observed f values. Clearly, there are many possibilities. Furthermore, if a different classifier is used, MARS for example, new sampling strategies may be required to sample sub-regions of the partition efficiently.

Local Phase

In this thesis the altered Hooke and Jeeves algorithm was used as a local phase algorithm to potentially increase numerical performance where f was smooth. It would be interesting to consider other local optimization methods that can exploit derivative information if it becomes available, but are not reliant on it. Simply using a standard forward tracking ray search in the direction of the pattern move in the Hooke and Jeeves algorithm is worth considering. Price, Robertson and Reale made this simple modification in another algorithm and found a significant improvement in the algorithm's performance [62].

Rate Theorem for CARTopt

Strong theoretical convergence results have been presented for the CARTopt algorithms, however, the rate of convergence has not been analyzed. One can empirically analyze the rate of convergence by testing the CARTopt algorithm on a variety of problems. However, it would be interesting to mathematically derive a lower bound on the rate of convergence, for example, at least linear.

Global CARTopt Algorithm

Another interesting area of research would be to extend the CARTopt algorithm into a global rather than a localized global optimization algorithm. One could, for example, use the Multi-Start approach whereby a local search is applied from each seed point (see Section 2.1). The unconstrained CARTopt instance could be applied from each seed to produce a nonsmooth global optimization algorithm.

Parallel Processing

The CARTopt algorithms could be coded to exploit multiple processors, for example, when possible splitting hyperplanes in partition phase are be calculated (see Section 2.4.1). All potential splits in each dimension are considered separately and hence, each dimension could be treated separately on its own processor. Parallel processing may make the CARTopt methods run faster in practice, particularly on higher dimensional problems. Furthermore, if the Multi-Start global CARTopt algorithm was developed, a single processor could be used for each unconstrained CARTopt instance.

Clearly, there is plenty of interesting work to be done...

Bibliography

[1] M. A. Abramson, C. Audet, J. E. Dennis Jr., and S. Le Digabel, *Orthomads: a deterministic mads instance with orthogonal directions*, SIAM J. Optimization, **20** (2008), pp.948–966.

[2] M. M. Ali, C. Khompatraporn, and Z. B. Zabinsky, *A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems*, Journal of Global Optimization, **31** (2005), pp.635–672.

[3] M. J. Appel, R. Labarre, and D. Radulovic, *On accelerated random search*, SIAM J. Optimization, **14** (2003), pp.708–731.

[4] C. Audet and J. E. Dennis Jr., *A pattern search filter method for nonlinear programming without derivatives*, SIAM J. Optimization, **14** (2004), pp.980–1010.

[5] C. Audet and J. E. Dennis Jr., *Mesh adaptive direct search algorithms for constrained optimization*, SIAM J. Optimization, **17** (2006), pp.889–903.

[6] W. P. Baritompa, Zhang Baoping, R. H. Mladineo, G. R. Wood, and Z. B. Zabinsky, *Towards pure addaptive search*, Journal of Global Optimization, **7** (1995), pp.93–110.

[7] C. A. Botsaris and D. H. Jacobson, *A newton-type curvilinear search method for optimization*, Journal of Mathematical Analysis and Applications, **54** 1976, pp.217–229.

[8] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and regression trees*, Chapman and Hall/CRC, 1993.

[9] A. A. Brown and M. C. Bartholomew-Biggs, *Some effective methods for unconstrained optimization based on the solution of systems of ordinary differential equations*, Journal of Global Optimization, **62** 1989, pp. 211–224.

[10] D. Byatt, *Convergent variants of the Nelder-Mead algorithm*, Master's thesis, University of Canterbury, Christchurch, New Zealand, January, 2003.

[11] D. Byatt, I. D. Coope, and C. J. Price, *Conjugate grids for unconstrained optimization*, Computational Optimization and Applications, **29** (2003), pp.49–68.

[12] D. Byatt, I. D. Coope, C. J. Price, *Conjugate grids for unconstrained optimisation*, Computational Optimization and Applications, **29** (2004), pp. 49–68.

[13] F. H. Clarke, *Optimization and Nonsmooth Analysis*, Classics in Applied Mathematics, SIAM, Philadelpha, 1990.

[14] A. R. Conn, K. Scheinberg, and L. N. Vicente, *Introduction to derivative-free optimization*, MPS-SAIM series on Optimization, 2009.

[15] I. D. Coope and C. J. Price, *Frame based methods for unconstrained optimization*, Journal of Optimization Theory and Applications, **107** (2000), pp. 261–274.

[16] I. D. Coope and C. J. Price, *On the convergence of grid-based methods for unconstrained optimization*, SIAM J. Optimization, **11** (2001), pp. 859–869.

[17] I. D. Coope and C. J. Price, *Positive bases in numerical optimization*, Computational Optimization and Applications, **21** (2002), pp. 169–175.

[18] I. D. Coope and C. J. Price, *A direct search frame-based conjugate gradients method*, Journal of Computational Mathematics, **22** (2004), pp. 489–500.

[19] C. Davis, *Theory of positive linear dependence*, American Journal of Mathematics, **76** (1954), pp. 733–746.

[20] M. Demirhan, and L. Ozdamar, *A note on a partitioning algorithm with reference to Tang's statistical promise measure*, IEEE Transactions on Automatic Control, **45** (2000), pp.510–515.

[21] L. Devroye, L. Gyorfi, G. Lugosi, *A probabilistic theory of pattern recognition*, Springer, 1996.

[22] C. C. Y. Dorea, *Expected number of steps of a random search method*, Journal of Optimization Theory and Applications, **39**, (1983), pp.165–171.

[23] C. C. Y. Dorea, *Stopping rules for a random optimization method*, SIAM J. Control and Optimization, **28**, (1990), pp.841–850.

[24] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification*, Wiley-Interscience, New York, 2001.

[25] S. Ermakov, A. Zhigyavskii, and M. Kondratovich, *Comparison of some random search procedures for a global extremum*, U.S.S.R Comput. Math. Math. Phys., **29** (1989), pp.112–117.

[26] R. Fletcher and S. Leyffer, *Nonlinear programming without a penalty function*, Dundee University, Dept. of Mathematics, Report NA/183 (1997).

[27] J. H. Friedman, *Multivariate adaptive regression splines*, Annals of Statistics, **19** (1991), pp.1–67.

[28] D. Fouskakis, and D. Draper, *Stochastic optimization: a review*, International Statistical Review, **70** (2002), pp 315–349.

[29] J. D. Griffin and T. G. Kolda, *Nonlinearly constrained optimization using heuristic penalty methods and asynchronous parallel generating set search*, Applied Mathematical Research eXpress, **1** (2010), pp.1–62.

[30] L. Haan, *Esitmation of the minimum of a function using order statistics*, Journal of the American Statisical Association, **76**, (1981), pp.476–469.

[31] J. H. Halton, *On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals*, Numerische Mathematik, **2** (1960), pp.84–90.

[32] W. E. Hart, *Sequential stopping rules for random optimization methods with applications to multistart local search*, SIAM J. Optimization, **9**, (1998), pp.270–290.

[33] S. Hess and J. W. Polak, *On the performance of the shuffled Halton sequence in the esitmation of discrete choice models*, European Transport Conference Proceedings, 2003.

[34] W. Hock and K. Schittkowski, *Test examples for nonlinear programming codes*, Lecture Notes in Economics and Mathematical Systems (**187**), Springer-Verlag, 1981.

[35] R. Hooke and T. A. Jeeves, *Direct search solution of numerical and statistical problems*, Journal of the Association for Computing Machinery, **8** (1961), pp. 212–229.

[36] S. Incert, V. Parisi and F. Zirilli, *A new method for solving nonlinear simultaneous equations*, SIAM J. Numerical Analysis, **16** 1979, pp.779–789.

[37] D. Jones, C. D. Perttunen and B. E. Stuckman, *Lipschitzian optimization without the Lipschitz constant*, Journal of Optimization Theory and Applications, **79** (1993), pp.157–181.

[38] A. H. G. Rinnooy Kan and G. T. Timmer, *Stochastic global optimization methods part I: Clustering methods*, Mathematical Programming, **39** (1987), pp. 27–56.

[39] A. H. G. Rinnooy Kan and G. T. Timmer, *Stochastic global optimization methods part II: Multi level methods*, Mathematical Programming, **39** (1987), pp. 57–78.

[40] E. Karas, A. Ribeiro, C. Sagastizabal and M. Solodov, *A bundle-filter method for nonsmooth convex constrained optimization*, Mathematical Programming, **116** (2009), pp.297–320.

[41] 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** (2003), pp.385–482.

[42] I. E. Lagaris and I. G. Tsoulos, *Stopping rules for box-constrained stochastic global optimization*, Applied Mathematics and Computation, **197** (2008), pp662–632.

[43] R. M. Lewis and V. J. Torczon, *Rank ordering and positive bases in pattern search algorithms*, Institute for Computer Applications in Science and Engineering, Mail Stop 132C, NASA Langley Research Center, Hampton, Virginia 23681-2199, 1996.

[44] R. M. Lewis, V. J. Torczon, and M. W. Trosset, *Direct search methods: Then and now*, Journal of Computational and Applied Mathematics, **124**, Isses 1-2 (2000), pp. 191–207.

[45] L. Luksan and J. Vlcek, *Test problems for nonsmooth unconstrained and linearly constrained optimization*, Institute of Computer Science, Academy of Sciences of the Czech Republic, Technical report, **798**, 2000.

[46] W. L. Martinez, and A. R. Martinez, *Computational statistics handbook with matlab*, Chapman and Hall/CRC, Florida, 2002.

[47] Matlab. The MathWorks Inc, 24 Prime Park Way, Natwik, Massachusetts, USA.

[48] K. I. M. McKinnon, *Convergence of the Nelder-Mead simplex method to a nonstatioary point*, SIAM J. Optimization, **9** (1998), pp.148–158.

[49] J. Mockus, *The simple bayesian algorithm for the multidimensional global optimization*, Numerical Techniques for Stochastic Systems, (1980), pp. 369–377.

[50] R. E. Moore, *On computing the range of values of a rational function of n vaiables over a bounded region*, Computing, 1976.

[51] J. J. Moré, B. S. Garbow, and K. E. Hillstrom, *Testing unconstrained optimization software*, ACM Transactions on Mathematical Software, **7** (1981), pp.17–41.

[52] J. A. Nelder and R. Mead, *A simplex method for function minimization*, The Computer Journal, **7** (1965), pp. 308–313.

[53] A. Neumaier, *Interval methods for systems of equations*, Cambridge Press, 1990.

[54] J. Pinter, *Convergence qualification of adaptive partition algorithms in global optimization*, Mathematical Programming, **56** (1992), pp.343–360.

[55] E. Polak, *Computational methods in optimization, a unified approach*, Academic Press, New York, 1971.

[56] E. Polak, *Optimization: algorithms and consistent approximations*, Applied Mathematical Sciences, **124**, Springer, New York, 1997.

[57] M. J. D. Powell, *An efficient method for finding the minimum of a function of several variables without calculating derivatives*, The Computer Journal, **7** (1964), pp. 155–162.

[58] M. J. D. Powell, *A view of algorithms without derivatives*, Report DAMTP 2007/NA03. CMS, University of Cambridge, Cambridge CB3 0WA, UK. April, 2007.

[59] C. J. Price, I. D. Coope, and D. Byatt, *A convergent variant of the Nelder-Mead algorithm*, Journal of Optimization Theory and Applications, bf 113 (2002), pp.5–19.

[60] C. J. Price and I. D. Coope, *Frames and grids in unconstrained and linearly constrained optimization: A nonsmooth approach*, SIAM J. Optimization, **14** (2003), pp. 415–438.

[61] C. J. Price, M. Reale, and B. L. Robertson, *A direct search method for smooth and nonsmooth unconstrained optimization*, ANZIAM J., **48** (2006), pp.927–948.

[62] C. J. Price, B. L. Robertson, and M. Reale, *A hybrid Hook and Jeeves-Direct method for nonsmooth optimization*, Advanced Modeling and Optimization, **11** (2009), pp.43–61.

[63] D. Radulovi, *Pure random search with exponential rate of convergency*, Optimization, **54** (2008), pp.1–15.

[64] R. G. Regis and C. A. Shoemaker, *Improved strategies for radial basis function methods for global optimization*, J Glob Optim., **37** (2007), pp. 113–135.

[65] H. E. Romeijn, and R. L. Smith, *Simulated annealing for constrained global optimization*, Journal of Global Optimization, **5** (1994), pp.101–126.

[66] H. H. Rosenbrock, *An automatic method for finding the greatest or least value of a function*, The Computer Journal, **3** (1960), pp. 175–184.

[67] S. Schäffler and H. Warsitz, *A trajectory-following method for unconstrained optimization*, Journal of Optimization Theory and Applications, **67** 1990, pp.133–140.

[68] K. Schittkowski, *More test examples for nonlinear programming codes*, Lecture Notes in Economics and Mathematical Systems (**282**), Springer-Verlag, 1987.

[69] L. Shi, and S. Olafsson, *Nested partitions method for global optimization*, Operations Research, **48** (2000), pp.390–407.

[70] J. A. Snyman, *A new and dynamic method for unconstrained minimization*, Applied Mathematical Modelling, **6** 1982, pp.449–462.

[71] J. A. Snyman, *An improved version of the original leapfrog dynamic method for unconstrained minimization: LFOP1(b)*, Applied Mathematical Modelling, **7** 1983, pp.216–218.

[72] J. A. Snyman and L. P. Fatti, *A multi-start minimization algorithm with dynamic search trajectories*, Journal of Optimization Theory and Applications, **54** 1987, pp.121–141.

[73] I. M. Sobol, *On the systematic search in a hypercube*, SIAM J. Numerical Analysis, **16** (1979), pp.790–793.

[74] F. J. Solis and R. J. Wets, *Minimization by random search techniques*, Mathematics of Operations Research, **6** (1981), pp.19–31.

[75] W. Spendley, G. R. Hext, and F. R. Himsworth, *Sequential application of simplex designs in optimisation and evolutionary operation*, Technometrics, **4** (1962), pp. 441–461.

[76] C. Stephens, *Global optimization theory versus practice*, PhD thesis, University of Canterbury, Christchurch, New Zealand, 1997.

[77] P. Tan, M. Steinbach, and V. Kumar, *Introduction to data mining*, Addison Wesley, 2006.

[78] Z. B. Tang, *Adaptive partitioned random search to global optimization*, IEEE Transactions on Automatic Control, **39** (1994), pp.2235–2244.

[79] V. J. Torczon, *On the convergence pattern search algorithms*, SIAM J. Optimization, **7** (1997), pp.1–25.

[80] A. Törn and S. Viitanen, *Topographical global optimization using pre-sampled points*, Journal of Global Optimization, **5** (1994), pp.267–276.

[81] A. Törn and S. Viitanen, *Interative topographical global optimization*, State of the Art in Global Optimization, Kluwer Academic Publishers, Netherlands (1996), pp.353–363.

[82] M. W. Trosset, *I know it when I see it: Toward a definition of direct search methods*, SIAG/OPT Views-and-News, **9** (1997), pp.7–10.

[83] Y. Wardi, *Random search algorithms with sufficient descent for minimization of functions*, Mathematics of Operations Research, **14** (1989), pp.343–354.

[84] L. Wasserman, *All of nonparametric statistics*, Springer, New York, 2006.

[85] Y. Wen-Ci, *Positive bases and a class of direct search techniques*, Scientia Sinica, Special Issue of Mathematics, **1**, (1979).

[86] T. Ye and Shivkumar Kalyanaraman, *A recursive random search algorithm for large-scale network parameter configuration*, Sigmetrics'03, Proc 2003 ACM Sigmetrics conf. on measurement and modelling of computer systems, 2003, pp.196–205.

[87] Z. B. Zabinsky, R. L. Smith, J. F. McDonalds, H. E. Romejeijn, and D. E. Kaufman, *Improving hit-and -run for global optimization*, Journal of Global Optimization, **3** (1992), pp.171–192.

[88] Z. B. Zabinsky and R. L. Smith, *Pure adaptive search in global optimization*, Mathematical Programming, **53** (1992), pp.323–338.

[89] Z. B. Zabinsky, *Stochastic Adaptive Search for Global Optimization*, Kluwer Academic Publishers, 2003.

[90] J. H. Zar, *Biostatistical Analysis*, Prentice Hall International Editions, 1996.

[91] A. A. Zhigljavsky, *Theory of Global Random Search*, Kluwer Academic Publishers, Netherlands, 1991.

Appendix A

Test Functions

This appendix lists the test problems used to generate the numerical results in Chapter 7. These problems are taken from Schittkowski *et al.* [34, 68], Moré *et al.* [51] and Luksan *et al.* [45]. The test problems are listed in the two tables which follow.

A.1 Unconstrained Test Problems

The unconstrained test problems selected from [45] are nonsmooth, of the form,

$$f(x) = \max_{1<i\leq m} \{f_i(x)\} \text{ such that } x \in \mathbb{R}^n, \tag{A.1}$$

where m is a positive integer. The interested reader is referred to [45] for further details.

The unconstrained test problems selected from [51] are smooth, nonlinear least squares problems of the form,

$$f(x) = \sum_{i=1}^{m} f_i^2(x) \text{ such that } x \in \mathbb{R}^n, \tag{A.2}$$

where m is a positive integer. In particular, problems with global solutions of zero were chosen. These problems were made into nonsmooth problems by replacing the squares in (A.2) with absolute values. The fact that $f_i = 0$ for all i at the global minimizer means the global solution remains unchanged in the nonsmooth version of (A.2).

The remaining unconstrained test problems were selected from Schittkowski *et al.* [34, 68]. These problems are expressed in the same form as (A.2) and are made nonsmooth in the same way described above. To avoid confusion, the nonsmooth version

Table A.1: Unconstrained Nonsmooth Problems. The first three columns list the function name, dimension and the optimization starting point. The next two columns list the optimal function value and optimal point. The final two columns list the box center and radius for testing the CARTopt algorithm subject to bound constraints.

Problem	n	x_0	f_*	x_*	$\mathcal{O}$	h
Beale [51]	2	(1, 1)	0	(3, 0.5)	(2, 0.8)	1.5
CB2 [45]	2	(1, 0.1)	1.9522245	—	(1, 0.5)	1
QL [45]	2	(-1, 5)	7.2	(1.2, 2.4)	(0.1, 3.7)	2
Rosenbrock [51]	2	(-1.2, 1)	0	(1, 1)	(-0.1,1)	1.5
Wolfe [45]	2	(3, 2)	-8	(-1, 0)	(1, 1)	2.5
25 (Gulf) [34]	3	(100, 12.5, 3)	0	(50, 25, 1.5)	(75, 19, 2.3)	see (A.3)
240 [68]	3	(100, -1, 2.5)	0	(0, 0, 0)	(50, -0.5, 1.2)	51
Helical Valley [51]	3	(-1, 0, 0)	0	(1, 0, 0)	(0, 0, 0)	1.5
Powell [51]	4	(3, -1, 0, 1)	0	(0, 0, 0, 0)	(1.5, -1, 0, 0.5)	2
261 [68]	4	(0, 0, 0, 0)	0	(0, 1, 1, 1)	(0, 0.5, 0.5, 0.5)	1
Rosen-Suzuki [45]	4	(0, 0, 0, 0)	-44	(0, 1, 2, -1)	(0, 0.5, 1, -0.5)	1.5
Trigonometric [51]	5	(0.2, . . ., 0.2)	0	—	(0, 0, 0, 0, 0)	0.5
Variably Dim. [51]	8	(7/8, 6/8, . . ., 1/8, 1)	0	(1, . . ., 1)	(0.5, . . ., 0.5)	1
291 (Quartic) [68]	10	(1, . . ., 1)	0	(0, . . ., 0)	(0.5, . . ., 0.5)	1

of test problem 261 is listed below because it is not expressed as a sum of squares in [68].

$$f(x) = |e^{x_1} - x_4| + 10|x_2 - x_3| + |\tan(x_3 - x_4)| + |x_1| + |x_4 - 1|.$$

The minimizer $x_* = (0, 1, 1, 1)$ with minimum value $f(x_*) = 0$ remains the same as in [68].

A constrained version of the Gulf problem from [34] was used to test the CARTopt algorithm subject to bound constraints. Under the scaling,

$$\begin{aligned} x_1 &\leftarrow 99.9(x_1 + 1)/2 + 0.01 \\ x_2 &\leftarrow 25.6(x_2 + 1)/2 \\ x_3 &\leftarrow 5(x_3 + 1)/2 \end{aligned} \tag{A.3}$$

optimization was performed in the $x_0 + [-1, 1]^n$ box.

A.2 Nonlinear Programming Test Problems

The nonsmooth constrained problems selected from [45] are linearly constrained minimax objective functions with the same form as (A.1). The interested reader is referred to [45] for further details.

The problems selected from Schittkowski *et al.* [34, 68] are problems expressed as a sum of squares which were made nonsmooth by replacing squares with absolute values. A mixture of problems with linear and nonlinear equality and inequality constraints are considered. Bound constrained problems were avoided as they can be solved directly using the bound constrained CARTopt instance under appropriate scaling. The interested reader is referred to [34] and [68] for further details on these problems.

Additional constrained problems from [34] and [68] were also selected. These problems had obvious nonsmooth analogs that left the feasible global solution unchanged. Two additional problems B1 and B2 were also created by the author. These problems are listed below to avoid confusion.

Nonsmooth version of test problem 228,

$$f(x) = |x_1| + x_2$$

subject to the constraints given in [68].

Nonsmooth version of test problem 43,

$$f(x) = |x_1| + |x_2| + 2|x_3| + |x_4| - 5x_1 - 5x_2 - 21x_3 + 7x_4$$

subject to the constraints given in [34].

Nonsmooth version of test problem 46,

$$f(x) = |x_1 - x_2| + |x_3 - 1| + |x_4 - 1| + |x_5 - 1|$$

subject to the constraints given in [34].

The nonsmooth function B1 is defined by

$$f(x) = 2x_1 + |x_2| + 2x_3 + |x_4|,$$

subject to $x_i \geq 0$ for $i = 1, \ldots, 4$.

The nonsmooth function B2 is defined by

$$f(x) = 2x_1 + |x_2| + 2x_3 + |x_4| + 2x_5 + |x_6|,$$

subject to $x_i \geq 0$ for $i = 1, \ldots, 6$.

Table A.2: Nonsmooth Nonlinear Programming Problems. The first four columns list the function name, dimension and the number of inequality and equality constraints. The values in the parentheses indicate the number of nonlinear constraints present. The next three columns list the optimization starting point, optimal function value and the optimal point. The remaining two columns list box center and radius for PRS and the initial box for the unconstrained CARTopt algorithm.

Problem	n	IC	EC	x_0	f_*	x_*	$\mathcal{O}$	h
6 [34]	2	-	1(1)	(-1.2, 1)	0	(1, 1)	(-0.1, 1)	1.5
225 [68]	2	5(4)	-	(3, 1)	2	(1, 1)	(2, 1)	1.5
228 [68]	2	2(1)	-	(0, 0)	-3	(0, -3)	(0, -1.5)	2
MAD1 [45]	2	1	-	(1, 2)	-0.38965952	$\approx$(-0.4, 0.9)	(0.3, 1.5)	1
MAD2 [45]	2	1	-	(-2, -1)	-0.33035714	$\approx$(-0.9, 0.2)	(-1.5,-0.4)	1
MAD5 [45]	2	1	-	(-1, 3)	-0.42928061	$\approx$(1.5, 2.4)	(0.3, 2.7)	1.5
28 [34]	3	-	1	(-4, 1, 1)	0	(0.5, -0.5, 0.5)	(-1.7, 0.2, 0.7)	2.5
32 [34]	3	4(1)	1	(0.1, 0.7, 0.2)	1	(0, 0, 1)	(0.1, 0.4, 0.8)	1
249 [68]	3	2(1)	-	(1, 1, 1)	1	(1, 0, 0)	(1, 0.5, 0.5)	1
43 [34]	4	3(3)	-	(0, 0, 0, 0)	-44	(0, 1, 2, -1)	(0, 0.5, 1, -0.5)	1.5
B1	4	4	-	(1, 1, 1, 1)	0	(0, 0, 0, 0)	(1, 1, 1, 1)	1.5
46 [34]	5	-	2(2)	$(1/\sqrt{2}, 1.75, 0.5, 2, 2)$	0	(1, 1, 1, 1, 1)	(0.9, 1.3, 0.8, 1.5, 1.5)	1
48 [34]	5	-	2	(3, 5, -3, 2, -2)	0	(1, 1, 1, 1, 1)	(2, 3, -1, 2, 0.5)	2.5
51 [34]	5	-	3	(2.5, 0.5, 2, -1, 0.5)	0	(1, 1, 1, 1, 1)	(1.7, 0.7, 1.5, 0, 0.7)	3.5
Pentagon [45]	6	15	-	—	-1.85961870	-	(0, 0, 0, 0, 0, 0)	1
B2	6	6	-	(1, 1, 1, 1, 1, 1)	0	(0, 0, 0, 0, 0, 0)	(1, 1, 1, 1, 1, 1)	1.5
MAD6 [45]	7	9	2	(0.5, 1, 1.5, 2, 2.5, 3, 3.5)	0.10183089	$\approx$(0.4, 0.8, 1.2, 1.7, 2.1, 2.7, 3.5)	(0.5, 0.9, 1.4, 1.8, 2.3, 2.9, 3.5)	0.5

Appendix B

Matlab Code

This appendix contains the Matlab [47] code used to generate the numerical results in Chapter 7. Firstly, the nonsmooth optimization algorithm CARTopt is given. This algorithm is called as a subroutine in the Hooke and Jeeves / CARTopt hybrid algorithm, which follows in the next section. Finally, the additional functions required to implement the CARTopt filter algorithm are given in Section B.3.

B.1 CARTopt Algorithm

The function CARTOPT makes call to the functions CART, POSTPARTITION, OPTIMALPOWERFIT, and PTDIST. The latter is listed at the end of CARTOPT, separated with dashed lines, while the other three are listed in the subsections which follow. In addition, CARTopt can be implemented as filter based algorithm for nonlinear programming. In this case CARTopt makes call to the function CARTOPT_FILTER_T, which is listed in Section B.3.

```
function [Optimal_value,x_star,CARTopt_term,T_k,f_T,f_count] = ...
     CARTopt(T_k,f_T,x_0,h_m,fname,hname,n,N,chi,k_S,f_k,f_count,H_Omega,...
     Halton,subroutine,unconstrained,CARTopt_filter)
%
% CARTOPT is a random search nonsmooth optimization algorithm. The method
% forms a partition on the optimzation region using CART, directing
% computational effort in promising regions. The algorithm is implemented
% with f_k = [], H_Omega = [] and subrountine = 0. The variables f_k and
% H_Omega are only active when subroutine = 1, when CARTopt is used as the
% localized global phase of the Hooke and Jeeves / CARTopt algorithm.
% The algorithm can also be implemented as filter based method for non-
% linear programming problems. This version is implemented with f_k = [],
% H_Omega = [], subrountine = 0, unconstrained = 1 and CARTopt_filter = 1.
%
```

```
% Variables:
%
% Optimal_value             = essential local minimum
% x_star                    = essential local minimizer
% fn                        = total function evaluations
% CARTopt_term              = Indicator variable, 1 = termination due to
%                             satisfied stopping rule or k_max satisfied
% T_k                       = training data set (set [] if not known)
% f_T                       = function values of T_k (set [] if not known)
% x_0                       = optimization region center
% h_m                       = optimization region radius
% fname                     = function name (enter as 'string')
% hname                     = constraint violation function name (enter as
%                             'string'). Set [] if CARTopt_filter = 0
% n                         = dimension of function
% N                         = batch size (N>0)
% chi                       = fraction of points distributed into high
%                             region, where 0 <= chi <= 1
% k_S                       = number of iterations to sample high region,
%                             set to 0 if no sampling is required
% f_count                   = number of function evaluations (= 0 in CARTopt)
% Halton                    = Deterministic instance using Halton sequenece,
%                             set to 1 for Halton, 0 otherwise
% subroutine                = indicator variable - 1 means CARTopt is being
%                             used as a subroutine in the hybrid HJ/CARTopt
%                             algorithm, 0 otherwise
% unconstrained             = indicator variable - 1 means unconstrained
%                             instance of CARTopt, 0 otherwise.

% Initialize:
f_epsilon = 1e-8;           % f_1 - epsilon tolerance for stopping rule
gamma = 40;                 % number of points for stopping rule
beta = 1e-6;                % terminating probability for stopping rule
k_max = max(1000,100*(n^2));% maximum number of iterations
if CARTopt_filter == 1
    wL_size = N;            % wL size for filter method
else
    wL_size = floor(0.8*N); % maximum wL size
end
sigma_k = 10;               % initial penalty parameter value
frac = floor((1-chi)*N);    % number of points to sample from low regions
impurity = 0.0;             % misclassification allowed in partition
CARTopt_term = 0;           % Updates to 1 if stopping rule satisfied
% Set post-partition vectors as empty
fn_postpart = 0; X_U = []; F_U = []; h_U = [];
% Set the scaled initial Lebesgue measure of the optimization region
measure_Lk = 2^n;
% Caclulate critical value for KS test in stopping rule
zeta = (-log(0.025)/(2*gamma))^(1/2) - (0.16693/gamma);
% Set dimension of Halton set for deterministic instance
Halton_counter = 2;
if Halton == 1
    Halton_set = haltonset(n+1);
```

```
else
    Halton_set = [];
end
if subroutine == 0 % No optimization region transformation matrix
    H_Omega = eye(n,n);
end
if k_S == 0 && chi == 1
   error('Cannot sample the high region with k_S = 0 and chi = 1, increase k_S.')
end
if CARTopt_filter == 1 && unconstrained == 0
   error('CARTopt filter method is unconstrained, set unconstrained == 1')
end


% Genterate first batch of points for the unconstrained instance of CARTopt
% giving an input training data set.
if unconstrained == 1
    if Halton == 1
        T_k = 2*Halton_set(Halton_counter:Halton_counter + 2*N-2,2:n+1) - 1;
        Halton_counter = Halton_counter + 2*N-1;
    else
        T_k = 2*rand(2*N-1,n) - 1;
    end
    T_k = [zeros(1,n);T_k];
    % Evaluate the objective function at x_0 and each x in X_N
    f_T = zeros(2*N,1);                 h_T = zeros(2*N,1);
    for i = 1:2*N
        f_T(i) = feval(fname, x_0 + h_m*(H_Omega*T_k(i,:)'));
        f_count = f_count + 1;
        if CARTopt_filter == 1 %evaluate the constraint violation at each x
            h_T(i) = feval(hname, x_0 + h_m*(H_Omega*T_k(i,:)'));
        end
    end
    if CARTopt_filter == 1 % Impose maximum constraint violation h(x_0)
        Max_violation = h_T(1);
        h_max = max(1,Max_violation);
    end
end

% Generate initial batch (X_N) of 40 random pts in [-1,1]^n box (using input
% training data if given).

Card_f_T = length(f_T);
if Card_f_T >= 2*N
    X_N = T_k;      f_X = f_T;      T_k = [];       f_T = [];
    if CARTopt_filter == 1
        h_X = h_T;      h_T = [];
    end
else
    if Halton == 1
        X_N = 2*Halton_set(Halton_counter:Halton_counter + 2*N - ...
                                                    Card_f_T-1,2:n+1) - 1;
        Halton_counter = Halton_counter + 2*N - Card_f_T;
```

```
    else
        X_N = 2*rand(2*N - Card_f_T,n) - 1;
    end
    % Evaluate the objective function at each x in X_N and return to
    % HJ_CARTopt if descent is made and subroutine = 1
    f_X = zeros(2*N - Card_f_T,1);
    for i = 1:2*N - Card_f_T
        f_X(i) = feval(fname, x_0 + h_m*(H_Omega*X_N(i,:)')');
        f_count = f_count + 1;

        if subroutine == 1
            if f_X(i) < f_k
                % Define outputs and update output training data set
                Optimal_value = f_X(i);
                x_star = x_0 + h_m*(H_Omega*X_N(i,:)')';
                T_k = [X_N(1:i-1,:); T_k];     f_T = [f_X(1:i-1); f_T];
                return
            end
       end
    end
    % Set initial batch of points
    X_N = [X_N;T_k];       f_X = [f_X;f_T];        T_k = [];       f_T = [];
end

% Ensure there exists at least on element with finite function value
inf_count = 1;
while min(f_X) == inf
    if Halton == 1
        X_N(2*N+inf_count,:) = 2*Halton_set(Halton_counter,2:n+1) - 1;
        Halton_counter = Halton_counter + 1;
    else
        X_N(2*N+inf_count,:) = 2*rand(1,n)-1;
    end
    f_X(2*N+inf_count) = feval(fname, x_0 + h_m*X_N(2*N+inf_count,:));
    f_count = f_count + 1;      inf_count = inf_count+1;
    if inf_count == 1000
        error('Unable to locate a finite function value in sufficient time')
    end
end

for k = 1:k_max % main loop
    k
    % add all points to the training data set
    T_k = [X_U; X_N; T_k];              f_T = [F_U; f_X; f_T];

    % update T_k for the CARTopt filter algorithm and check stopping rule
    if CARTopt_filter == 1
        % update constraint violations
        h_T = [h_U; h_X; h_T];
        % enter training data update, classification and stopping function
        [T_k,f_T,h_T,Terminate,wL,wH,F_wL,cardwL,Sloping_filter,...
         x_k_info,z_k_info] = CARTopt_filter_T(T_k,f_T,n,h_T,gamma,...
         N,sigma_k,zeta,f_epsilon,beta,h_max);
```

```
if Terminate == 1 % stopping rule satisfied -- terminate algorithm
    break
end

else % update T_k for CARTopt algorithm if T_k exceeds full size and
     % and check stopping rule

    if length(f_T) > max(2*N,2*(n-1)*N)
        [FT,I_FT] = sort(f_T);
        T_k_low = T_k(I_FT(1:gamma),:);     F_low = FT(1:gamma);
        T_k(I_FT(1:gamma),:) = [];          f_T(I_FT(1:gamma),:) = [];
        T_k = [T_k_low; T_k];               f_T = [F_low; f_T];
        T_k = T_k(1:2*(n-1)*N,:);           f_T = f_T(1:2*(n-1)*N);

        % Check stopping rule as minimum number of points is reached
        F = FT(1:gamma);
        % Calculate optimal power fit
        [kappa_star,f_0_star,KS] = optimalpowerfit(F,'infnorm',n);

        if KS < zeta      % sufficient power fit
            Probf_c = max(real(((F(1) - f_epsilon - f_0_star)/...
                                (max(F) - f_0_star))^kappa_star),0);
            if Probf_c < beta
                disp('  ')
                disp('Halt due to stopping condition being satisfied')
                CARTopt_term = 1;
                break % terminate algorithm, prob of lower pt small
            end
        end
    end
end

% Classify training data points into two categories wL and wH
if CARTopt_filter == 0

    [Y,IT] = sort(f_T);          cardwL = min(wL_size, sum(isfinite(Y)));
    wL = T_k(IT(1:cardwL),:);    wH = setdiff(T_k,wL,'rows');
    F_wL = f_T(IT(1:cardwL),:); % function values in wL
end

% Calculate Householder matrix H_k (data transform matrix)
m = (1/cardwL)*sum(wL);
M = (wL - ones(cardwL,1)*m)'*(wL - ones(cardwL,1)*m);
[V,D] = eig(M);                v = V(:,find(sum(D) == max(sum(D))))';
v = v(1,:); % remove eigenvectors with repeated eigenvalues
I = eye(n,n);                  u = (I(:,1) - v')/norm(I(:,1) - v');
H_k = I - 2*u*u';
% Calculate the transformation scaling coefficient theta
if unconstrained == 1
    theta = 1; % searching R^n -- no scaling required.
else
    theta = max(sum(abs(H_k')));
```

```
end
% transform the training data T = [wL,wH]
wL = (1/theta)*(H_k*wL')';    wH = (1/theta)*(H_k*wH')';

% Partition optimization region using CART
[B] = CART(wL,wH,impurity,n,N,unconstrained);

% Perform post-partition modifications on the CART partition
[B,X_U,F_U,fn_postpart,measure_Lk,post_descent,Halton_counter,h_U] =...
     Postpartition(B,wL,n,x_0,h_m,fname,H_k,F_wL,fn_postpart,theta,...
     measure_Lk,Halton,Halton_set,Halton_counter,H_Omega,subroutine,...
     unconstrained,CARTopt_filter,hname,sigma_k);

% Terminate if descent was found in post-partition phase. Only possible
% if sub-routine = 1.
if post_descent == 1
    [Optimal_value, O_v_I] = min(F_U);
    x_star = x_0 + h_m*(H_Omega*X_U(O_v_I,:)')';
    % Define outputs and update output training data set
    F_U(O_v_I) = [];        X_U(O_v_I,:) = [];
    T_k = [X_U; T_k];       f_T = [F_U; f_T];
    f_count = f_count + fn_postpart;
    return
end

% Distribute points in low regions using a near uniform distribution
[X_N,Halton_counter] = ...
                    ptDist(B,N,n,frac,H_k,theta,Halton,Halton_set,...
                    Halton_counter,unconstrained);

% Apply inversive transform to new batch of points
X_N = theta*(H_k*X_N')';

%Sample high region if required, i.e. sample [-1,1]^n
if k_S >= k
    if Halton == 1
        X_N(frac+1:N,:) = 2*Halton_set...
                 (Halton_counter: Halton_counter + N-frac-1,2:n+1) - 1;
        Halton_counter = Halton_counter + N-frac;
    else
        X_N(frac+1:N,:) = 2*rand(N-frac,n) - 1;
    end
end
if k_S == k % Stop sampling high region
    frac = N;
end

% Evaluate the objective function at each x in X_N and return to
% HJ_CARTopt if descent is made and subroutine = 1
f_X = zeros(N,1);           h_X = zeros(N,1);
for i = 1:N
    f_X(i) = feval(fname, x_0 + h_m*(H_Omega*X_N(i,:)')');
    if f_X(i) == -inf
```

```
            error('Objective function unbouned below, f(x_k) = -inf')
        end
        f_count = f_count + 1;

        if CARTopt_filter == 1 %evaluate the constraint violation at each x
            h_X(i) = feval(hname, x_0 + h_m*(H_Omega*X_N(i,:)')');
        end
        if subroutine == 1
            if f_X(i) < f_k
                Optimal_value = f_X(i);
                x_star = x_0 + h_m*(H_Omega*X_N(i,:)')';
                % Define outputs and update output training data set
                T_k = [X_N(1:i-1,:); T_k];     f_T = [f_X(1:i-1); f_T];
                f_count = f_count + fn_postpart;
                return
            end
        end
    end

    % Increase the penalty parameter if the linear approximation to the
    % sloping filter is greater than the current penalty paramter
    % (only required in the CARTopt_filter algorithm)
    if CARTopt_filter == 1
        if length(Sloping_filter(:,1)) > 1
            [P,S,MU] = polyfit(Sloping_filter(:,n+2),Sloping_filter(:,n+1),1);
            if -P(1)/MU(1) > sigma_k
            sigma_k = min(-P(1)/MU(1), min(2*sigma_k,1e10))
            end
        end
    end
end


% ---------------------    Define outputs    ------------------------------


if nargout > 0
    [Y_term,I_term] = sort(f_T);            Optimal_value = Y_term(1);
    x_star = x_0 + h_m*T_k(I_term(1),:);    f_count = f_count + fn_postpart;
    CARTopt_term = 1;
    if subroutine == 0 % don't output training data
        T_k = [];       f_T = [];
    end
else
    if subroutine == 0
        if CARTopt_filter == 1
            % least f + sigma*h value (z_k)
            Optimal_f_sigma_h_value = z_k_info(n+1) + sigma_k*z_k_info(n+2)
            f_sigma_h_x_star = x_0 + h_m*z_k_info(1:n);
            f_sigma_h_violation = z_k_info(n+2)
            sigma_star = sigma_k
            % least constraint violation (x_k)
            Optimal_h_value = x_k_info(n+1)
```

```
            f_sigma_h_x_star = x_0 + h_m*x_k_info(1:n)
            h_violation = x_k_info(n+2)
            % total objective function evaluations
            f_count = f_count + fn_postpart
        else
            % CARTopt algorithm
            [Y_term,I_term] = sort(f_T);
            Optimal_value = Y_term(1)
            x_star = x_0 + h_m*T_k(I_term(1),:)
            f_count = f_count + fn_postpart
        end
    end
end

% End of CARTOPT.m

%-------------------------------------------------------------------------


function [X_N,Halton_counter] = ...
  ptDist(B,N,n,frac,H_k,theta,Halton,Halton_set,Halton_counter,unconstrained)
%
% PTDIST distributes 'frac' points into LOW
% sub-regions. An invervse transform method is used to
% distribute the points. Outputs a matrix X_N containing
% new sample points.

% Initialize:
num_regions = length(B(:,1)); X_N = zeros(N,n); k = 1;
hypervolume = zeros(1,num_regions); side = zeros(1,n);
% Calculate hypervolume of all sub-regions
for i = 1:num_regions
    for j = 1:n
        side(j) = B(i,n+j) - B(i,j);
    end
    hypervolume(i) = prod(side);
end
% Calculate hypervolume CDF for inverse transform
hyp_cdf = cumsum(hypervolume)*(1/sum(hypervolume));

while k < frac + 1

    if Halton == 1
        subregion = find(hyp_cdf > Halton_set(Halton_counter,1));
        X_Nk = (B(subregion(1),n+1:2*n) - B(subregion(1),1:n)).*...
            Halton_set(Halton_counter,2:n+1) + B(subregion(1),1:n);
        Halton_counter = Halton_counter + 1;
    else
        subregion = find(hyp_cdf > rand(1,1));
        X_Nk = (B(subregion(1),n+1:2*n) - B(subregion(1),1:n)).*rand(1,n)...
            + B(subregion(1),1:n);
    end
```

```
    if unconstrained == 1

        X_N(k,:) = X_Nk;                k = k+1;
    else
        % Reject sample point if outside [-1,1]^n box
        if norm(theta*H_k*X_Nk',inf) < 1

            X_N(k,:) = X_Nk;            k = k+1;
        end
    end
end

% End of PTDIST.m
```

B.1.1 The CART Partition

The function CART makes call to the functions NODESPLIT and LOWREGIONS. These functions are listed at the end of CART, separated by dashed lines.

```
function [B] = CART(wL,wH,impurity,n,N,unconstrained)
%
% CART performs a partition on Omega using Classification and Regression
% Trees. A matrix, TreeMatrix, is systematically updated by assigning points
% to particular nodes. The matrix has the following structure:
%
% TreeMatrix = [nodenumber : sample points : classification]
%
% When a terminal node is found each row of TreeMatrix corresponding to that
% terminal node is removed. The method terminates when TreeMatrix is empty.
% A matrix B containing the bounds on each low sub-region of the partition
% is the only output.
%
% Variables:
%
% wL                = low points
% wH                = high points
% iD_vector         = vector containing impurities at each node
% node              = vector containing all nodes of the tree
% termNodes         = vector containing all terminal nodes of the tree
% Classification    = vector containing classification of each terminal
%                     node, whereby wL = 0 and wH = 1
% CharSplit         = Matrix containing splitting values. Each row element
%                     not equal to 2 is the splitting value and its column
%                     position is the splitting dimension.


% Initialize: Preallocate storage
termNodes = zeros(1,2*(n-1)*N);     Classification = 2*ones(1,2*(n-1)*N);
iD_vector = zeros(4*(n-1)*N,2);         CharSplit = zeros(4*(n-1)*N,n);
```

```
node = zeros(1,4*(n-1)*N);          node(1) = 1;
% Counters
k = 1;  j = 0;  i = 0;
% Sizes
Card_wL = length(wL(:,1));  Card_wH = length(wH(:,1));

% TreeMatrix at root node
TreeMatrix = [ones(Card_wL,1),wL,zeros(Card_wL,1);...
    ones(Card_wH,1),wH,ones(Card_wH,1)];

% Calculate impurity at root node (Note 0*log2(0) = 0)
p_wL = Card_wL/(Card_wL + Card_wH); p_wH = Card_wH/(Card_wL + Card_wH);
iD_vector_k = -(min(0,p_wL*log2(p_wL)) + min(0,p_wH*log2(p_wH)));
iD_vector(1) = iD_vector_k;
% Grow tree until all terminal nodes are pure or satisfy the impurity cond.
while ~isempty(TreeMatrix)

    % Determine points at node(k)
    node_k_wL = find(TreeMatrix(:,1) == node(k) & TreeMatrix(:,n+2) == 0);
    nodek_wH = find(TreeMatrix(:,1) == node(k) & TreeMatrix(:,n+2) == 1);

    if iD_vector(k) == 0 || (length(node_k_wL) > length(nodek_wH) ...
                        && iD_vector(k) <= impurity)

        j = j+1;
        termNodes(j) = node(k);
        % Update TreeMatrix by deleting rows containing terminal nodes
        % which are pure or satisfy impurity condition
        TermNoderows = find(TreeMatrix(:,1) == node(k));
        % Classify terminal node
        Classification(j) = TreeMatrix(TermNoderows(1),n+2);
        TreeMatrix(TermNoderows,:) = [];
        k = k + 1; % Increment iteration counter

    else % Grow Tree further

        wL = TreeMatrix(node_k_wL,2:n+1);
        wH = TreeMatrix(nodek_wH,2:n+1);
        node_k = node(k);   iD_vector_k = iD_vector(k);

        % Split node(k) to find child nodes
        [SPLIT,ChildNodes,iD_best_k,LeftChild,RightChild] = ...
            NodeSplit(wL,wH,node_k,n,iD_vector_k);

        i = i+2;
        iD_vector([i,i+1]) = iD_best_k;
        node([i,i+1]) = ChildNodes;
        CharSplit(i/2,:) = SPLIT;
        % Update node(k) in TreeMatrix after split
        Update_node = find(TreeMatrix(:,1) == node(k));
        TreeMatrix(Update_node,1) = [LeftChild,RightChild]*ChildNodes';

        k = k + 1; % Increment iteration counter
```

```
    end
end

% Resize vectors for LowRegions input.
termNodes = termNodes(1:j);          Classification = Classification(1:j);
node = node(1:i+1);                  CharSplit = CharSplit(1:i/2,:);

% Define matrix B with low sub-region bounds
[B] = LowRegions(n,node,CharSplit,Classification,termNodes,unconstrained);

% End of CART.m


%-----------------------------------------------------------------------


function [SPLIT,ChildNodes,iD_best_k,LeftChild,RightChild] =...
    NodeSplit(wL,wH,node_k,n,iD_vector_k)
%
% NODESPLIT splits the current node using the entropy impurity
% relation -sum(P_wj*log_2(P_wj)) where j = L,H. Each
% potential split is considered between elements wL and wH.
%
% Variables:
%
% S                     = vector of possible splits for data
% SPLIT                 = optimal split, expressed as vector. Element not
%                         equal to 2 is the splitting value and position
%                         indicates which dimension is split
% LeftChild             = point(s) at left child node after the split
% RightChild            = point(s) at right child node after the split
% iD_best_k             = vector with impurities at each child node


% Initialize:
best_impurity = 0;  parentNode = node_k;
ChildNodes = [2*parentNode, 2*parentNode + 1];
Card_wL = length(wL(:,1));  Card_wH = length(wH(:,1));
num_pts = Card_wL + Card_wH;

for i = 1:n
    % find all potential splits in dimension i
    [Y,IY] = sort([wL(:,i);wH(:,i)]);
    I_1 = [ones(Card_wL,1);2*ones(Card_wH,1)];    I_2 = [I_1(IY);0];
    I_3 = [0;I_1(IY)];      I_4 = I_2 + I_3;
    S = find(I_4 == 3);

    for j = 1:length(S)

        % Splitting value
        Split = (Y(S(j)) + Y(S(j) - 1))/2;

        % determine where points go after the split
```

```
        wL_left = zeros(Card_wL,1); wL_right = ones(Card_wL,1);
        wH_left = zeros(Card_wH,1); wH_right = ones(Card_wH,1);

        % wL points
        wL_left(find(wL(:,i) < Split)) = 1;
        wL_right = wL_right - wL_left;

        % wH points
        wH_left(find(wH(:,i) < Split)) = 1;
        wH_right = wH_right - wH_left;

        % define variables for change in impurity calcultion
        num_wL_left = sum(wL_left);        num_wH_left = sum(wH_left);
        num_wL_right = sum(wL_right);      num_wH_right = sum(wH_right);

        p_left = (num_wL_left + num_wH_left)/num_pts;

        num_pts_left = num_wL_left + num_wH_left;
        num_pts_right = num_wL_right + num_wH_right;
        p_wL_left = num_wL_left/num_pts_left;
        p_wL_right = num_wL_right/num_pts_right;
        p_wH_left = num_wH_left/num_pts_left;
        p_wH_right = num_wH_right/num_pts_right;

        iD_left = -(min(0,p_wL_left*log2(p_wL_left)) +...
                    min(0,p_wH_left*log2(p_wH_left)));
        iD_right = -(min(0,p_wL_right*log2(p_wL_right)) +...
                    min(0,p_wH_right*log2(p_wH_right)));

        % Calculate change in impurity delta_iN
        delta_impurity = iD_vector_k - p_left*iD_left - (1-p_left)*iD_right;

        if delta_impurity > best_impurity    % better split found

            % Update best impurity
            best_impurity = delta_impurity;
            % Update outputs
            iD_best_k = [iD_left, iD_right];
            SPLIT = 2*ones(1,n);  SPLIT(i) = Split;
            LeftChild = [wL_left;wH_left];
            RightChild = [wL_right;wH_right];
        end
    end
end

% End of NodeSplit.m


%-------------------------------------------------------------------------


function [B] = LowRegions(n,node,CharSplit,Classification,termNodes,...
                          unconstrained)
```

```
%
% LowRegions defines all low sub-regions defined in the CART partition. The
% output matrix B contains the bounds on each sub-region where each row is
% of the form: [x1_min, ... , xn_min, x1_max, ... , xn_max]

% Initialize
num_splits = length(CharSplit(:,1)); node(1) = []; % Remove root node
Odd = 1:2:2*num_splits - 1;    Even = 2:2:2*num_splits;

% determine terminal nodes with classification wL
termNodes_wL = termNodes(find(Classification == 0));
% Set up initial structure for output Matrix B
if unconstrained == 1
    B = [-inf*ones(length(termNodes_wL),n),inf*ones(length(termNodes_wL),n)];;
else
    B = [-ones(length(termNodes_wL),n),ones(length(termNodes_wL),n)];
end

% Set up matrix Split_M such that even rows correspond to upper bounds
% and odd rows correspond to lower bounds using splitting values
MxMn = zeros(1,2*num_splits);   MxMn(Odd) = 1;
Split_M = zeros(2*num_splits,n);
Split_M(Odd,:) = CharSplit;          Split_M(Even,:) = CharSplit;

for k = 1:length(termNodes_wL) % Determine path to each terminal node

    tN = termNodes_wL(k);            Path = tN;
    while floor(tN/2) > 1
        tN = floor(tN/2);            Path = [tN, Path];
    end

    % define region bounds from path
    L_P = length(Path);      region = zeros(L_P,n);
    for j = 1:L_P
        region(j,:) = Split_M(find(node == Path(j)),:);
    end

    % Update the output matrix B
    for i = 1:L_P
        if MxMn(find(node == Path(i))) == 1
            B(k,n+find(region(i,:)~=2)) = region(i,find(region(i,:)~=2));
        else
            B(k,find(region(i,:)~=2)) = region(i,find(region(i,:)~=2));
        end
    end
end

% End of LowRegions.m
```

B.1.2 Post-Partition Modifications

```
function [B,X_U,F_U,fn_postpart,measure_Lk,post_descent,Halton_counter,h_U] = ...
    Postpartition(B,wL,n,x_0,h_m,fname,H_k,F_wL,fn_postpart,...
    theta, measure_Lk,Halton,Halton_set,Halton_counter,H_Omega,...
    subroutine,unconstrained,CARTopt_filter,hname,sigma_k)
%
% POSTPARTITION performs a post-partition modfication on the CART
% partition of Omega. The method uses the forward-tracking face search
% (FTFS) technique and defines singleton sub-regions with hypercubes. If the
% indicator variable subroutine is 1 (subroutine of the hybrid HJ/CARTopt
% algorithm), the method terminates if descent is found, returning to the
% local HJ phase. If CARTopt_filter = 1 (nonlinear programming method)
% problematic bounds are calculated with respect to f + sigma*h, not f.
%
% Variables:
%
% F_wL                      = f values at the set of points wL
% fn_postpart               = number of f evaluations performed
% X_U                       = points generated in FTFS
% F_U                       = f values generated in FTFS
% delta                     = minimum subregion radius
% measure_Lk                = measure of union of low sub-regions from the
%                             previous iteration
% post_descent              = terminates post-partition phase if descent
%                             is found, post_descent = 1, 0 otherwise

% Initialize:
delta = 1e-10;          U_counter = 0;          singleton_counter = 0;
measure_counter = 0;    post_descent = 0;
% Preallocate storage
X_U = zeros(length(wL)*10*n,n);         F_U = zeros(length(wL)*10*n,1);
h_U = zeros(length(wL)*10*n,1);
singleton_B_i = zeros(length(wL),n+1);   measure = zeros(length(wL)/2,1);

for i = 1:length(B(:,1))

    pts = wL;               f_pts = F_wL;
    % Determine wL points in sub-region B_i along with function values
    for j = 1:n
        Y = find(pts(:,j) > B(i,j) & pts(:,j) < B(i,n+j));
        pts = pts(Y,:);         f_pts = f_pts(Y);
    end

    if length(pts(:,1)) == 1 % Singleton sub-region found

        singleton_counter = singleton_counter + 1;
        singleton_B_i(singleton_counter,:) = [pts,i];
    else    % Non-singleton sub-region found
        % Preallocate storage
        dim_range = zeros(1,2*n);   dim_min_max = zeros(1,2*n);
        dim_f = zeros(1,2*n);
        B_i = B(i,1:2*n); % temperary updated sub-region B_i
```

```
for j = 1:n % Find max and min points for each dimension of B_i

    [min_f,mf_I] = min(pts(:,j));    [max_f,Mf_I] = max(pts(:,j));

    dim_min_max(j) = min_f;          dim_min_max(n+j) = max_f;
    dim_f(j) = f_pts(mf_I);          dim_f(n+j) = f_pts(Mf_I);
    dim_range(n+j) = dim_min_max(n+j) - dim_min_max(j);
    dim_range(j) = -dim_range(n+j);
    % Impose minimum splitting distance, forcing minimum box size
    if unconstrained == 1
        B_i(j) = min(B_i(j),dim_min_max(j) - delta);
        B_i(j+n) = max(B_i(j+n),dim_min_max(j+n) + delta);
    else % ensure approximate level set remains in [-1,1]^n
        B_i(j) = min(B_i(j),max(-1, dim_min_max(j) - delta));
        B_i(j+n) = max(B_i(j+n),min(1, dim_min_max(j+n) + delta));
    end
end

% find problematic boundaries
if unconstrained == 1
    prob_bound = find(abs(B(i,:)) == inf);
else
    prob_bound = find(abs(B(i,:)) == 1);
end
Update = zeros(1,2*n);               Update(prob_bound) = 1;

alpha = 1/3; % initial forward-tracking value

for ij = 1:10 % Forward-tracking face search method (max 10 its)

    B_i(prob_bound) = dim_min_max(prob_bound) + ...
                                        alpha*dim_range(prob_bound);
    if unconstrained == 0
    % Reset nonproblematic bounds to penultermate values (-1 or 1)
        nonprob_bound = find(abs(B_i) > 1);
        if numel(nonprob_bound) > 0
            Update(nonprob_bound) = 0;
            B_i(nonprob_bound) = B(i,nonprob_bound);
        end
    end

    if sum(Update) == 0
        break % no problematic bounds, consider next sub-region
    end

    % Evaluate f at a randomly generated pt on each problematic face
    prob_bound = find(Update > 0);
    for ii = 1:length(prob_bound)
        face = B_i;
        if prob_bound(ii) <= n % Lowerface

            face(n + prob_bound(ii)) = B_i(prob_bound(ii));
```

```
else  % Upperface
    face(prob_bound(ii) - n) = B_i(prob_bound(ii));
end

% Generate a random point on face
if Halton == 1
    x_face = (face(1,n+1:2*n) - face(1,1:n)).*...
        Halton_set(Halton_counter,2:n+1) + face(1,1:n);
    Halton_counter = Halton_counter + 1;
else
    x_face = (face(1,n+1:2*n) - face(1,1:n)).*rand(1,n)...
         + face(1,1:n);
end

if unconstrained == 1

    f_face = feval(fname, x_0' + ...
                           theta*h_m*H_Omega*H_k*x_face');
    fn_postpart = fn_postpart + 1;

    % keep point and function value for training data
    U_counter = U_counter + 1;
    X_U(U_counter,:) = (theta*H_k*x_face')';
    F_U(U_counter) = f_face;
    if CARTopt_filter == 1
        % evaluate constraint violation
        h_face = feval(hname, x_0' + ...
                           theta*h_m*H_Omega*H_k*x_face');
        % keep constraint violation value for training data
        h_U(U_counter,:) = h_face;
        % fix face if f + sigma*h is increased
        if f_face + sigma_k*h_face > dim_f(prob_bound(ii))
            Update(prob_bound(ii)) = 0;
        else
            dim_f(prob_bound(ii)) = f_face + sigma_k*h_face;
        end
    else % fix face if f is increased
        if f_face > dim_f(prob_bound(ii))
            Update(prob_bound(ii)) = 0;
        else
            dim_f(prob_bound(ii)) = f_face;
        end
    end

else
    % fix bound if x_face is not an element of Omega
    if norm(theta*H_k*x_face',inf) < 1 % in Omega
        f_face = feval(fname, x_0' + ...
                           theta*h_m*H_Omega*H_k*x_face');
        fn_postpart = fn_postpart + 1;

        % keep point and function value for training data
        U_counter = U_counter + 1;
```

```
                        X_U(U_counter,:) = (theta*H_k*x_face')';
                        F_U(U_counter) = f_face;

                        if subroutine == 1
                            if f_face < F_wL(1) % terminate if descent is found
                                post_descent = 1;
                                X_U = X_U(1:U_counter,:);
                                F_U = F_U(1:U_counter);
                            return
                            end
                        end

                        if f_face > dim_f(prob_bound(ii))
                            Update(prob_bound(ii)) = 0;
                        else
                            dim_f(prob_bound(ii)) = f_face;
                        end
                    else
                        Update(prob_bound(ii)) = 0;
                    end
                end
            end

            alpha = 3*alpha; % Increase forward-tracking coefficient
            prob_bound = find(Update > 0);
        end

        B(i,1:2*n) = B_i; % Update sub-region bounds

        % Calculate Lebesgue measure of updated sub-region
        side = zeros(1,n);
        for j = 1:n
            side(j) = B(i,n+j) - B(i,j);
        end
        measure_counter = measure_counter + 1;
        measure(measure_counter) = prod(side);
    end
end

% Resize output vectors
singleton_B_i = singleton_B_i(1:singleton_counter,:);
measure = measure(1:measure_counter);

% Construct a hypercube for each singleton sub-region
if numel(singleton_B_i) > 0

    %determine box radius
    if numel(singleton_B_i) < length(wL)
        cube_measure = sum(measure)/(length(wL)-length(singleton_B_i(:,1)));
        cube_radius = max((cube_measure^(1/n))/2,delta/2);
    else % All sub-regions are singleton, base hypercube on measure_Lk
        cube_measure = measure_Lk/length(wL);
        cube_radius = max((cube_measure^(1/n))/2,delta/2);
```

```
    end

    for j = 1:length(singleton_B_i(:,1))
        for p = 1:n

            if unconstrained == 1
                % update max
                B(floor(singleton_B_i(j,n+1)),n+p) = ...
                                        singleton_B_i(j,p) + cube_radius;
                % update min
                B(floor(singleton_B_i(j,n+1)),p) = ...
                                        singleton_B_i(j,p) - cube_radius;
            else % ensure approximate level set remains in [-1,1]^n
                % update max
                B(floor(singleton_B_i(j,n+1)),n+p) = ...
                                    min(1, singleton_B_i(j,p) + cube_radius);
                % update min
                B(floor(singleton_B_i(j,n+1)),p) = ...
                                    max(-1, singleton_B_i(j,p) - cube_radius);
            end
        end
    end
    measure_Lk = sum(measure) + numel(singleton_B_i)*cube_measure;
else
    measure_Lk = sum(measure);
end

% Define outputs: Resize X_U and F_U for output
X_U = X_U(1:U_counter,:);          F_U = F_U(1:U_counter);
h_U = h_U(1:U_counter);

% End of Postpartition.m
```

B.1.3 Stopping Rule for CARTopt

The function OPTIMALPOWERFIT makes call to the function INFNORM which is listed immediately below OPTIMALPOWERFIT.

```
function [kappa_star,f_0_star,KS] = optimalpowerfit(F,fittype,n)
%
% OPTIMALPOWERFIT calculates the optimal power fit approximation to a
% theoretical CDF using the empircal data given by F. The method minimizes
% the infinity norm distance between the EDF and theoretical CDF using
% golden section search.
%
% Variables:
%
% kappa_star        = optimal power for theoretical CDF
% f_0_star          = optimal approximation to f_*
```

```
% KS                = the Kolomogorov Smirnov statistic
% tau_1             = termination bracket size

% Initialize:
Card_F = length(F);          tau_1 = 0.001;
% determine largest distance d to smallest f_* approximation
d = F(40) - F(1);
% Define set of f_0_star values
F_0 = [F(1)- d,F(1) - d/2, F(1) - d/4];
Card_F0 = length(F_0);
kappa_best = zeros(1,Card_F0);      Fnorm_best = zeros(1,Card_F0);

for j = 1:Card_F0

    f_0_star = F_0(j);
    f_1 = feval(fittype,1,f_0_star,F,Card_F);
    f_2 = feval(fittype,2*n,f_0_star,F,Card_F);
    f_e = feval(fittype,2*n - tau_1,f_0_star,F,Card_F);
    % initial bracket and function values
    B = [1, 2*n];    F_B = [f_1,f_2];

    if f_e >= f_2 % bracket does not exist to within tolerance

        L_B = 2*tau_1; % Don't enter loop, no optimization required
        B = [2*n,2*n];
    else
        L_B = B(2) - B(1);
    end

    % Apply golden section search to find optimal power kappa_star
    while L_B > tau_1

        aa = B(1) + (1 - 0.5*(sqrt(5) - 1))*L_B;
        bb = B(2) - (1 - 0.5*(sqrt(5) - 1))*L_B;

        f_aa = feval(fittype,aa,f_0_star,F,Card_F);
        f_bb = feval(fittype,bb,f_0_star,F,Card_F);
        % Update bracket
        if f_bb <= min(f_aa, F_B(2))
            B(1) = aa;              F_B(1) = f_aa;
        else
            B(2) = bb;              F_B(2) = f_bb;
        end

        L_B = B(2) - B(1);
    end

    kappa_best(j) = min(B);      Fnorm_best(j) = F_B(B == min(B));
end
% Define outputs
[Y,I] = sort(Fnorm_best);
KS = Y(1);      kappa_star = kappa_best(I(1));       f_0_star = F_0(I(1));
```

```
% end of OPTIMALPOWERFIT.m

%                       ----------------------

function [FK] = infnorm(kappa_star,f_0_star,F,Card_F)
%
% INFNORM calculates the Infinity norm distance between CDF and EDF function

FK = zeros(1,30);
for i = 1:30

    F_K = ((F(i) - f_0_star)/(F(Card_F)-f_0_star))^(kappa_star);
    FK(i) = norm([F_K - (i-1)/Card_F,F_K - i/Card_F],inf);
end

FK = max(FK);

% end of INFNORM.m
```

B.2 Hooke and Jeeves / CARTopt Hybrid Algorithm

The function HJ_CARTOPT makes call to the functions EXPLORE and CARTOPT. The latter is listed in the previous section and EXPLORE is listed in the subsection which follows.

```
function [f_OPT,x_OPT,f_evals,f_values_of_HJ,Global_f_values] = HJ_CARTopt...
                (x_0,fname,h_min,h,n,Uphill,N,chi,k_S,Halton,Smooth)
% HJ_CARTopt is a hybrid algortihm for solving nonsmooth optimization
% problems. A modified Hooke and Jeeves local phase is used to
% give the algorithm speed and CARTopt is used as the localized global
% optimization phase. The localized global phase gives convergence on
% nonsmooth problems.
%
% Variables:
%
% x_0               = Initial sample point
% fname             = Objective function name ('string')
% h_min             = Minimum mesh size
% h                 = Mesh size
% n                 = Objective function dimension
% Uphill            = Set to 1 to allow for uphill steps, 0 otherwise
% N                 = Batch size for CARTopt
% chi               = Fraction of points distributed into high
%                     region, where 0 <= chi <= 1
% k_S               = Number of iterations to sample high region
```

```
% Halton            = Deterministic version using Halton sequenece,
%                     set to 1 for Halton, 0 otherwise
% Smooth            = Implements the smooth algorithm if set to 1 and set
%                     to 0 for the nonsmooth algorithm
% f_OPT             = Final function value
% x_OPT             = Final point
% f_evals           = Total number of function evaluations

% Initialize:
x_k = zeros(n,1);       v_k = zeros(n,1);       E_k = zeros(n,1);
H = eye(n,n);           h_Omega = 1e-4;         pattern_scalar = 1;
T_k = [];               f_T = [];               f_count = 0;
subroutine = 1; % Inform CARTopt that it is a subroutine of this algorithm
max_grid_reduction = 3; % Maximum grid reduction coefficient
HJ_its = 0; HJ_maxits = 50000; % HJ iterations and maximum number of HJ its
x = x_0; % Initial grid origin
% Calculate initial function value at x_0
f_k = feval(fname,x);
FE_count = 1;                   % HJ function evaluation counter
U_k = f_k;                      % Set upper bound on allowable function values
XFG = [zeros(1,n),f_k];         % Grid point with known function value
Repeat_flag = 0;                % Potential repeated element in T_k flag


while h > h_min
    % Conduct exploratory phase about x_k + v_k
    z = x_k + v_k;
    [E_k,f_E,T_E,f_TE,FE_count,XFG] = ...
                                Explore(fname,z,x_0,h,n,E_k,H,FE_count,XFG);
    % Sinking Lid step:
    if Uphill == 1 % Potential ascent algorithm
        if f_E >= f_k

            if U_k - f_E >= 1e-15 && norm(v_k) > 1e-13
                U_k = (U_k + f_E)/2;
            else
                U_k = f_k;
            end
        end
    else
      U_k = f_k; % Strictly descent algorithm
    end

    % Pattern Move: implimented if f is less than current upper bound U_k
    if f_E < U_k

        x_k = x_k + v_k + E_k;      v_k = pattern_scalar*(v_k + E_k);
        f_k = f_E;

    elseif norm(v_k) > 0

        v_k = zeros(n,1);
```

```
else % enter the localized global optimization phase

    if Repeat_flag == 1 % Remove x0_hold from T_E (already in T_k)
        [EK,EK_I] = min(sum(abs((T_E - ...
                                (x0_hold'*ones(1,length(f_TE)))')')));
        if EK < 1e-14
            T_E(EK_I,:) = [];       f_TE(EK_I) = [];
        end
        Repeat_flag = 0;
    end

    % Set optimization region (Omega) radius and center
    if Smooth == 1 % smooth algorithm
        h_m = h*sqrt(n); % Allow Omega to shrink with h
    else
        h_m = max(h*sqrt(n), h_Omega); % Impose minimum Omega size.
    end
    x_0 = (x_0 + h*x_k)';   x_k = zeros(n,1);
    x0_hold = x_0; % hold onto optimization region center

    % Calculate transformation matrix H_Omega to remove colinear points
    % resulting from using T_E as an input training data set
    w = (1/sqrt(3+n))*H*[2;ones(n-1,1)];        I = eye(n,n);
    uw = (I(:,1) - w)/norm(I(:,1) - w);         H_Omega = I - 2*(uw)*uw';

    % Determine points from T_k that are elements of Omega
    T_k = [T_E; T_k];          f_T = [f_TE;f_T];
    T_k = (1/h_m)*(H_Omega*((T_k) - (x_0'*ones(1,length(f_T)))')')';
    NotIn_Omega = [];
    for j = 1:length(f_T)
        if norm(T_k(j,:),inf) > 1
            NotIn_Omega = [NotIn_Omega, j];
        end
    end
    T_k(NotIn_Omega,:) = [];        f_T(NotIn_Omega) = [];

    % Enter CARTopt with non-empty training data set
    [Optimal_value,x_star,CARTopt_term,T_k,f_T,f_count] =...
        CARTopt(T_k,f_T,x_0,h_m,fname,[],n,N,chi,k_S,f_k,f_count,...
        H_Omega,Halton,subroutine,0,0);

    % convert training data back to original coordinate system
    T_k = (x_0'*ones(1,length(f_T)))' + h_m*(H_Omega*T_k')';

    if CARTopt_term == 1 % Essential local minimizer found by CARTopt
        break % terminate algorithm
    end

    % Calculate grid transformation matrix H, update mesh size h and set
    % the new grid center.
    promising_direct = (x_star - x_0)/norm(x_star - x_0);      I = eye(n,n);
    u = (I(:,1) - promising_direct')/norm(I(:,1) - promising_direct');
    H = I - 2*(u)*u';
```

```
        if Smooth == 1 % force grid reduction for smooth version
            h = max(h/max_grid_reduction, norm(x_star - x_0)/(n+10e-10));
        else
            if norm(x_star - x_0) < h
                h = max(h/max_grid_reduction, norm(x_star - x_0));
            end
        end
        % Update grid points with known function values matrix
        if h == norm(x_star - x_0)
            XFG = [zeros(1,n), Optimal_value; (1/h)*(x_0 - x_star), f_TE(1)];
            Repeat_flag = 1;
        else
            XFG = [zeros(1,n), Optimal_value];
        end
        x_0 = x_star';      f_k = Optimal_value;        U_k = f_k;
    end
    HJ_its = HJ_its + 1;
    if HJ_its > HJ_maxits
        disp('  ')
        disp('Maximum number of Hooke and Jeeves iterations reached')
        break % terminate algorithm
    end
end

% Define outputs.
f_OPT = f_k;        x_OPT = x_0;        f_evals = f_count + FE_count;
f_values_of_HJ = FE_count;          Global_f_values = f_evals - FE_count;

% End of HJ_CARTopt.m
```

B.2.1 Exploratory Phase

```
function [E_k,f_E,T_E,f_TE,FE_count,XFG] = ...
                                Explore(fname,z,x_0,h,n,E_k,H,FE_count,XFG)

% EXPLORE conducts the modified exploratory moves of the Hooke and Jeeves
% algorithm. Neighboring grid points are considered in turn to define a
% direction of descent. The previous exploratory vector is used to indicate
% which directions are explored first.
%
% Variables:
%
% fname                 = objective function name ('string')
% z                     = current iterate (x_k + v_k)
% x_0                   = current grid origin
% h                     = mesh size
% n                     = dimension of objective function
% H                     = Grid transformation Householder Matrix
% E_k                   = exploratory vector
% f_E                   = function value at f(x_k + v_k + E_k)
% T_E                   = training data from exploratory phase
```

```
% f_TE                  = function values for training data T_E
% XFG                   = Known grid points and f values matrix, where
%                         XFG(i) = [x_i,f(x_i)] (row i)
% FE_count              = function evaluation counter

%Initialize:
T_E = zeros(2*n+1,n);           f_TE = zeros(2*n+1,1);
% Set up indicator matrix I, defining a promising search direction
if length(XFG(:,1)) <= 2 % new grid, no promising direction
    I = eye(n,n);
else
    E_k = sign(H*E_k);      E_k(find(E_k == 0)) = 1;        I = diag(E_k);
end
E_k = zeros(n,1);
% Evaluate f at current iterate. Check if f value is known at grid point
[XK,XK_I] = min(sum(abs((XFG(:,1:n) - (z*ones(1,length(XFG(:,1))))')')));

if XK < 1e-14 % known grid point
    f_k = XFG(XK_I(1),n+1);
else
    x = x_0 + h*z;      f_k = feval(fname,x);
    FE_count = FE_count + 1;
    % New grid point function value is known
    XFG = [z',f_k;XFG];
end
% Update training data
T_count = 1;    T_E(T_count,:) = (x_0 + h*z)';        f_TE(T_count) = f_k;

for i = 1:n   % Conduct search along +/-He_i
    z = z + H*I(:,i);
    % Check if f value is known at grid point
    [XK,XK_I] = min(sum(abs((XFG(:,1:n) - (z*ones(1,length(XFG(:,1))))')')));

    if XK < 1e-14 % known grid point
        f_E = XFG(XK_I(1),n+1);
    else
        x = x_0 + h*z;     f_E = feval(fname,x);    FE_count = FE_count + 1;
        % New grid point function value known
        XFG = [z',f_E;XFG];
    end
    % Update training data
    T_count = T_count + 1;
    T_E(T_count,:) = (x_0 + h*z)';         f_TE(T_count) = f_E;

    if f_E < f_k
        f_k = f_E;          E_k = E_k + H*I(:,i);

    else % Conduct search along -/+He_i
        z = z - 2*H*I(:,i);
        % Check if f value is known at grid point
        [XK,XK_I] = min(sum(abs((XFG(:,1:n) - ...
                                    (z*ones(1,length(XFG(:,1))))')')));
```

```
        if XK < 1e-14 % known grid point
            f_E = XFG(XK_I(1),n+1);
        else
            x = x_0 + h*z;  f_E = feval(fname,x);   FE_count = FE_count + 1;
            % New grid point function value known
            XFG = [z',f_E;XFG];
        end
        % Update training data
        T_count = T_count + 1;
        T_E(T_count,:) = (x_0 + h*z)';          f_TE(T_count) = f_E;

        if f_E < f_k
            f_k = f_E;              E_k = E_k -H*I(:,i);
        else % No descent obtained, reset to inital value
            z = z + H*I(:,i);
        end
    end
end

% Resize training data and define outputs
T_E = T_E(1:T_count,:);     f_TE = f_TE(1:T_count);         f_E = f_k;
return

% End of Explore.m
```

B.3 CARTopt Filter Algorithm

The function CARTOPT_FILTER_T makes call to the function OPTIMALPOWER-FIT, listed in the Section B.1.3.

```
function [T_k,f_T,h_T,Terminate,wL,wH,F_wL,cardwL,Sloping_filter,...
          x_k_info,z_k_info] = CARTopt_filter_T(T_k,f_T,n,h_T,gamma,N,...
                                          sigma_k,zeta,f_epsilon,beta,h_max)
%
% CARTOPT_FILTER_T updates the training data set, classifies and tests
% the stopping rule for the nonlinear programming algorithm CARTopt_filter.
% The gamma elements with the least h and f + sigma*h values, the
% sloping filter and the most recent elements are retained ensuring T_k
% does not exceed full size.
%
% Variables:
%
% T_k                       = training data points
% f_T                       = objective function values of T_k
% h_T                       = constraint violation values of T_k
% x_k_info                  = point, f value and h value that minimizes h
% z_k_info                  = point, f value and h value that minimizes
%                             f + sigma*h
```

```
% initialize:
Terminate = 0;                  % set termination flag to zero
tau = 0;                        % feasiblity tolerence parameter
cardwL = N;                     % cardinality of low points wL
T_max = max(2*gamma...          % cardinality of full size training data set
    + N,max(2*N,(n-1)*N));

% firstly, the sloping filter is calculated from the infeasible points
X = [T_k,f_T,h_T];
X_infeasible = X(find(X(:,n+2) > 0 & X(:,n+2) < h_max),:);

% remove points that are within tau of being feasible, keeping
% the element with the lowest function value only
h_minimized = X_infeasible(find(X_infeasible(:,n+2) < tau),:);

% find element within tau with the least objective function value
if numel(h_minimized) > 0
    [a,best_h_I] = min(h_minimized(:,n+1));
    best_h = h_minimized(best_h_I,:);
    X_infeasible = [setdiff(X_infeasible,h_minimized,'rows'); best_h];
end

% set of mu values for sloping filter
mu = [0, sigma_k/(2^10), sigma_k/(2^9), sigma_k/(2^8), sigma_k/(2^7),...
      sigma_k/(2^6),sigma_k/(2^5),sigma_k/(2^4),sigma_k/(2^3),...
      sigma_k/(2^2),sigma_k/(2^1),sigma_k];

for i = 1:length(mu)
    X = X_infeasible;
    Filter = zeros(size(X));          Filter_size = 0;
    while numel(X) > 0
        % test to see if current point is an element of the sloping filter
        x_filter = X(1,:);              X(1,:) = [];
        % remove elements filtered by x_f
        filtered = find(X(:,n+2) >= x_filter(n+2) & X(:,n+1) >= ...
                        x_filter(n+1) - mu(i)*(X(:,n+2) - x_filter(n+2)));
        X(filtered,:) = [];
        % add x_f to the filter if it is not filtered by another element
        filter_element = find(X(:,n+2) < x_filter(n+2) & X(:,n+1) < ...
                        x_filter(n+1) - mu(i)*(X(:,n+2) - x_filter(n+2)));
        if numel(filter_element) == 0
            Filter_size = Filter_size + 1;
            Filter(Filter_size,:) = x_filter;
        end
    end
    % resize the filter
    Filter = Filter(1:Filter_size,:);

    % keep set of undominated points (mu = 0 case)
    if i == 1
        X_D = Filter;
    end
```

```
    % test to see if filter is sufficiently small or if mu = sigma_k
    if Filter_size <= 3*N/4
        % sort sloping filter with respect to constraint violation
        [a,order_filter_I] = sort(Filter(:,n+2));
        Sloping_filter = Filter(order_filter_I,:);
        break
    end
    if i == length(mu) % keep 3N/4 most feasible
        % sort sloping filter with respect to constraint violation
        [most_feasible,most_feasible_I] = sort(X_D(:,n+2));
        Sloping_filter = X_D(most_feasible_I(1:3*N/4),:);
        Filter_size = 3*N/4;
    end
end


% ------------ the training data set is now updated ----------------------


if length(f_T) <= T_max % keep all points (including sloping filter)

    T_gamma = [T_k,f_T,h_T];

    % define least f + sigma*h element (z_k)
    [a, f_sigma_h_best_I] = min(f_T + sigma_k*h_T);
    z_k_info = T_gamma(f_sigma_h_best_I,:);
    % define least h with least f value (x_k)
    [h_best, h_best_I] = min(h_T);
    if h_best == 0
        feasible = T_gamma(find(T_gamma(:,n+2) == 0),:);
        [feas,feasible_I] = min(feasible(:,n+1));
        x_k_info = feasible(feasible_I,:);
    else
        x_k_info = T_gamma(h_best_I,:);
        feasible = [];  % no feasible points exist
    end

else % remove elements from T_k, maintaining full size

    %working matrix with all relevant elements
    MatrixW = [T_k,f_T,h_T];

    % firstly, keep the gamma elements with least f + sigma*h values
    [f_sigma_h, f_sigma_h_I] = sort(f_T + sigma_k*h_T);
    f_sigma_h_low = MatrixW(f_sigma_h_I(1:gamma),:);
    % define z_k output
    z_k_info = MatrixW(f_sigma_h_I(1),:);

    % secondly, keep the gamma elements with least constraint violations
    [a,h_T_I] = sort(h_T);
    if h_T(h_T_I(gamma)) == 0 % least f values as well

        feasible = MatrixW(find(MatrixW(:,n+2) == 0),:);
```

```
    [feas,feasible_I] = sort(feasible(:,n+1));
    h_T_low = feasible(feasible_I(1:gamma),:);
    % define x_k output
    x_k_info = feasible(feasible_I(1),:);
else
    h_T_low = MatrixW(h_T_I(1:gamma),:);
    % define x_k output
    if h_T(h_T_I(1)) == 0
        feasible = MatrixW(find(MatrixW(:,n+2) == 0),:);
        [feas,feasible_I] = min(feasible(:,n+1));
        x_k_info = feasible(feasible_I(1),:);
    else
        x_k_info = MatrixW(h_T_I(1),:);
        feasible = [];      % no feasible points exist
    end
end

% determine how many recent elements are required to keep T_k full size
T_gamma = union(f_sigma_h_low,h_T_low,'rows');
% the sloping filter is also included in this set
T_gamma = union(T_gamma,Sloping_filter,'rows');
card_T_gamma = length(T_gamma(:,1));
% collect unused points, preserving order
[unused_pts,unused_pts_I] = setdiff(MatrixW,T_gamma,'rows');
[a,II] = sort(unused_pts_I);            unused_pts = unused_pts(II,:);
% finally, keep the required number of recent points
recent_pts = unused_pts(1:T_max - card_T_gamma,:);

% Output training data
T_k = [T_gamma(:,1:n); recent_pts(:,1:n)];
f_T = [T_gamma(:,n+1); recent_pts(:,n+1)];
h_T = [T_gamma(:,n+2); recent_pts(:,n+2)];

% Check stopping rule. If a feasible iterate exists impliment existing
% CARTopt stopping when gamma feasible points are obtained. Otherwise
% terminate with respect to f + sigma*h.

[a,I_h_T] = sort(h_T);
if h_T(I_h_T(2)) == 0
    if h_T_low(gamma,n+2) == 0 % sufficient number of feasible points
        F = feas(1:gamma);
    else
        F = [];
    end
else
    F = f_sigma_h(1:gamma);
end

% Calculate optimal power fit for empirical data F
if numel(F) > 0
    [kappa_star,f_0_star,KS] = optimalpowerfit(F,'infnorm',n);

    if KS < zeta      % sufficient power fit
```

```
            Probf_c = max(real(((F(1) - f_epsilon - f_0_star)/...
                                (max(F) - f_0_star))^kappa_star),0);
            if Probf_c < beta
                disp('  ')
                disp('Halt due to stopping condition being satisfied')
                Terminate = 1; % terminate algorithm, prob of lower pt small
            end
        end
    end
end


% -------- Classify the training data set into sets wL and wH ------------


% firstly, include x_k and z_k in wL
wL_xkzk = union(x_k_info,z_k_info,'rows');
alpha = length(wL_xkzk(:,1));

% determine the feasible points to be included in wL (if they exist)
if ~isempty(feasible)
    feasible = setdiff(feasible,x_k_info,'rows');
    card_feasible = length(feasible(:,1));
    [a,feasible_I] = sort(feasible(:,n+1));
    wL_feas = feasible(feasible_I(1:min(card_feasible,max...
                        (floor(N/2 - alpha),N - Filter_size - alpha))),:);

    wL_feas_size = length(wL_feas(:,1));
else
    wL_feas = [];               wL_feas_size  = 0;
end

% determine the sloping filter elements to be in wL (if they exist)
if Filter_size > 0
    wL_filter = Sloping_filter(1:min(Filter_size,...
                                    N - wL_feas_size - alpha),:);
    wL_filter = setdiff(wL_filter,wL_xkzk,'rows');
else
    wL_filter = [];
end

% define the current set of low points wL
wL = [wL_xkzk; wL_feas; wL_filter];             wL_size = length(wL(:,1));

% determine elements with the least f + sigma*h values to be included in
% wL to ensure the cardinality of wL = N
if wL_size < cardwL

    X_f_sigma_h = setdiff(T_gamma,wL,'rows');
    [a,f_sigma_h_I] = sort(X_f_sigma_h(:,n+1) + sigma_k*X_f_sigma_h(:,n+2));
    wL_f_sigma_h = X_f_sigma_h(f_sigma_h_I(1:N-wL_size),:);
    % define wL and function values in the form f + sigma*h
    F_wL = [wL(:,n+1) + sigma_k*wL(:,n+2);...
```

```
            wL_f_sigma_h(:,n+1) + sigma_k*wL_f_sigma_h(:,n+2)];
    wL = [wL(:,1:n);wL_f_sigma_h(:,1:n)];
else
    % define wL and function value in the form f + sigma*h
    F_wL = wL(:,n+1) + sigma_k*wL(:,n+2);
    wL = wL(:,1:n);
end

% define the high points as the remaining elements of the training data
wH = setdiff(T_k,wL,'rows');

% end of CARTOPT_FILTER_T.m
```