AN IMPROVED UNCONSTRAINED GLOBAL OPTIMIZATION ALGORITHM

by

Ronald John Van Iwaarden

B. A., University of Colorado at Denver, 1989

M. A., University of Colorado at Denver, 1992

A thesis submitted to the

University of Colorado at Denver

in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Applied Mathematics

1996

This thesis for the Doctor of Philosophy

degree by

Ronald John Van Iwaarden

has been approved

by

Weldon Lodwick

Harvey Greenberg

George Corliss

Jennifer Ryan

Tom Russell

Date

Acknowledgement

I would like to recognize the guidance of Dr. Weldon Lodwick, without whom the ideas for this dissertation would not have grown. I recognize the criticism of Dr. Harvey Greenberg who pushed me farther than I thought I could go. I recognize the critique by Dr. George Corliss who improved the presentation of this dissertation and has been a continual source of inspiration and support. Finally, I recognize the love, support and devotion of my wife, Stephanie Van Iwaarden. She have given be the strength to endure and the love that was both necessary and sufficient to complete this work. Thanks to all who supported and instructed me through the years and have not been directly mentioned here.

Van Iwaarden, Ronald John (Ph.D., Applied Mathematics)

An Improved Unconstrained Global Optimization Algorithm

Thesis directed by Associate Professor Weldon Lodwick

ABSTRACT

Global optimization is a very hard problem especially when the number of variables is large (greater than several hundred). Recently, some methods including simulated annealing, branch and bound, and an interval Newton's method have made it possible to solve global optimization problems with several hundred variables. However, this is a small number of variables when one considers that integer programming can tackle problems with thousands of variables, and linear programming is able to solve problems with millions of variables. The goal of this research is to examine the present state of the art for algorithms to solve the unconstrained global optimization problem (GOP) and then to suggest some new approaches that allow problems of a larger size to be solved with an equivalent amount of computer time. This algorithm is then implemented using portable C++ and the software will be released for general use.

This new algorithm is given with some theoretical results under which the algorithm operates. Numerical results that demonstrate the improvement that can be obtained by using this new algorithm.

This abstract accurately represents the content of the candidate's thesis.

I recommend its publication.

Weldon Lodwick

CONTENTS

FIGURES

1. Introduction: Unconstrained Global Optimization Problems

1.1 Introduction

The general problem to be considered is $\min f(x) \in \Re$, where $x \in S$ and S is a non-empty compact set in $\Re^n$. Although general compact sets can have complex structure, S is assumed here to be a box $[\underline{x}_1, \overline{x}_1] \times [\underline{x}_1, \overline{x}_1] \times \cdots \times [\underline{x}_n, \overline{x}_n] \in \Re^n$. The general unconstrained global optimization is one of the problems in mathematics that is still considered to be very hard. Murty and Kabadi [71] construct a class of global optimization problems for which either identifying a given point as a local minimum or showing the objective to be unbounded is an NP-complete problem. This is also shown by Pardalos and Vavasis [75] for quadratic programming in 1991.

The class created by Murty and Kabadi [71] is contrived, but demonstrates that there are NP-complete global optimization problems. This has been confirmed in practice since extremely challenging problems relating to global optimization can be found in almost all sections of science and industry. Proving that these problems are NP-complete is difficult, but the difficulty in solving needs no proof. Examples of such nonlinear problems in which having the global optimum is either necessary or desirable include finding the direction and duration of radiation dosage of cancer tumors, the lowest energy state for a protein molecule, and the correct mixtures for optimal chemical plant operations.

1.2 Overview

The dissertation first reviews general global optimization methods. It explores both deterministic and stochastic methods and gives some of the advantages or disadvantages of these methods. It also includes some of the historical work in global optimization. Branch and bound methods (a type of deterministic method) are presented in depth because branch and bound methods are central to the development of the algorithm created herein. This sets the framework in which this research lies.

The dissertation then looks at some new developments in global optimization. The main contribution of this dissertation is the development of a method that we call Back-Boxing and the release of source code for global optimization and interval arithmetic. Back-Boxing allows problems to be solved to a greater degree of accuracy without an increase in the runtime.

The last section contains published or electronically posted test problems for nonlinear global optimization that are deemed to be difficult global optimization problems. These problems are then used to show the performance of the Back-Boxing algorithm. Tests are run showing that the Back-Boxing algorithm is faster for either high dimensional problems or for problems in which a high degree of accuracy is required.

2. A Review of General Global Optimization Methods

A general global optimization procedure is a technique that finds the global optima of any function, where it is assumed that the initial bounds on the values of the variables are given. It is also acceptable for the bounds on the variables to be infinite. That is, the acceptable values for the variables could be $(-\infty, \infty)$, $[k, \infty)$, or $(-\infty, k]$ where k is any real number. Thus, the ideal general global optimization solver is a black box into which one feeds the description of the function, its constraints, and the bounds on the variables. The output of this black box is a combination of the following states:

- The approximate values at which the function attains the optima with mathematically verified error bounds.
- Values which are the approximation of the global optimum along with a mathematically verified maximum error.
- The function is unbounded.

The character of the first two states could be a set of boxes that are guaranteed to contain the global optimum. Another possibility is to report a set of points for which one can give a mathematically correct probability as to how close the present best minimum function value is to the global optimum. These termination situations correspond to the types of output one would receive after completing a deterministic algorithm or terminating a stochastic process prematurely. These types of algorithms are studied in depth in the following sections.

One of the earliest papers on the subject of global optimization was published by Tuy [99]. The research has progressed quite quickly since then and has generated several books that are devoted only to global optimization [35, 38, 19, 18, 39, 88, 49].

2.1 Terminology

The following symbols and terms are used throughout this work.

NaN: (Not A Number) This is the IEEE definition of a number that does not exist or is undefined, such as $0/0$ or $\sin^{-1}(2)$. Note that numbers such as $1/0$ are not NaNs, but are represented as infinite values.

ξ**:** corresponds to a computer or an automaton on which the global optimization algorithm is applied. For example $\Re|\xi$ corresponds to the set of real numbers as represented on the automaton.

$x \in \Re^n$**:** A real vector.

X**:** An interval number. That is, $X = \{x|\underline{x} \leq x \leq \overline{x}\}$, where $\underline{x}$ is always the left endpoint, and $\overline{x}$ is the right endpoint. Note that this dissertation uses capitol letters to denote the interval version of any structure such as an interval matrix or vector.

I**:** The space of interval numbers.

$X \in I^n$**:** An interval vector. That is $X = [\underline{x}, \overline{x}]$ where $\underline{x}, \overline{x} \in \Re^n$.

$A \in I^{n \times n}$**:** An interval matrix. That is $A = [\underline{a}, \overline{a}]$ where $\underline{a}, \overline{a} \in \Re^{n \times n}$.

thick: An interval number, vector, or matrix X is said to be thick if there exists $x_1 \in X$ and $x_2 \in X$ with $x_1 \neq x_2$. That is, $\text{width}(X) > 0$.

thin: An interval number, vector, or matrix X is said to be thin if for all $x_1 \in X$ and $x_2 \in X$, $x_1 = x_2$. That is, $\text{width}(X) = 0$.

macheps: macheps is the smallest positive value ϵ such that 1+macheps $\neq$ 1 on the automaton being used.

machinf: machinf is the largest positive number that can be represented on the automaton.

mach-inf: mach-inf is the largest (in absolute value) negative number that can be represented on the automaton.

$\hat{X}$**:** The midpoint of the interval number or vector X.

$\square X$**:** The smallest interval number, vector, or matrix that contains the set X.

$]X[$**:** The smallest outward rounded interval number, vector, or matrix that contains X. On an automaton, this would correspond to rounding $\overline{x}$ up and rounding $\underline{x}$ down to the next machine representable number for that automaton.

vertex: $x^* \in R$ is said to be a vertex of a convex region R if the set $\{R - x^*\}$ is convex

2.2 Probabilistic Methods

Probabilistic methods for global optimization are infinite processes for which the probability of having visited the global optima approaches 1 as the number of steps tends to infinity. This means that if an uncountable number of test runs were made with the method, the global optimum would not be found in at most a countable number of the test runs. The most basic such method is as follows:

Algorithm 1 : Random Search

Initialization: $f_{min} = \infty$, x_{min} =NaN, and $i = 0$.

step 1: x^i = random number.

step 2: If $f(x^i) < f_{min}$ set $x_{min} = x^i, f_{min} = f(x^i)$.

step 3: Increment i and go to step 1.

If F is at least piecewise continuous, then as $i \to \infty$, $f_{min} \to \min_{\Re^n|\xi} f$. Hence, this is a general stochastic optimization technique. Rather than converging to the global minimum over $\Re^n$, the procedure converges to the minimum function value over all machine representable numbers if this procedure is implemented on a computer.

This method was studied by several researchers ([1, 10]). They show that if the function is evaluated at points which are drawn from a uniform distribution over the region S, then it can be shown that the smallest function value found in this way converges to the global minimum value y^* with probability 1.

2.2.1 Simulated Annealing

Simulated annealing (SA) is another probabilistic method that has been used with good results as an optimization technique for integer optimization and, more recently, for general global optimization.

SA is loosely based on the physical idea of annealing (cooling) metals in order to find the lowest energy state of that metal. For this reason, SA uses words such as temperature that relate back to this physical interpretation. The mathematical interpretation of annealing is a cooling schedule which is a decreasing function of time. For integer optimization, SA can be seen as a random walk with bias [57]. In general, a SA technique can be written as follows:

Algorithm 2 : Naive Simulated Annealing

Initialization: Set $x^0 = 0$, and $i = 0$.

step 1: Choose y in some neighborhood of x^i.

step 2: If $f(y) < f(x^i)$ set $x^{i+1} = y$, go to step 4.

step 3: If (Uniform Random Number) $< e^{-[f(y)-f(x^i)]/c(i)}$, set $x^{i+1} = y$.

step 4: Increment i and go to step 1.

where $c(i)$ is the temperature at time i [25], and f is the function being minimized. In general, one wants $c(i) \searrow c_\infty \neq 0$ as $i \to \infty$. This is a very naive implementation. Fox [25] recommends several methods for speeding up the convergence of this naive implementation. The convergence properties of this algorithm, as well as some of the more intelligent implementations, are discussed in [36].

Like most stochastic processes, this is not a true algorithm in the sense that it does not terminate in a finite number of steps. However, if one places a limit on the number of steps taken by the algorithm, then Heine [36] gives some probabilistic bounds on the percentage of the global optima that has been found based on the number of steps that have been taken.

The implementation of this technique is straight forward for discrete problems. One need only define the neighborhood of x_i and the cooling schedule, $c(i)$. When SA is applied to continuous problems, the idea of neighborhood is even more nebulous. For continuous SA, one idea is to first determine a search direction. Continuous SA would choose a point on the unit hyper-sphere at random about the point X_i that gives the search direction. The algorithm would then choose a random length to step in that direction. An obvious drawback of this procedure is that it does not use any local information (such as the derivatives). To avoid this problem, Dekkers and Aarts [15] randomly pick one of the following two methods. Either x_i is chosen at random, or x_i is chosen by applying a few steps of a local search procedure to x_{i-1}. Dekkers then goes on to show that as long as one implements the random search a sufficiently large number of times, the algorithm is still guaranteed to converge.

A general algorithm for the continuous nonlinear case with finite termination is as follows:

Algorithm 3 : Nonlinear Simulated Annealing

Initialization: Initialize (c, x) and set $i = 0$.

step 1: Generate another vector y from x.

step 2: If $f(y) - f(x) \leq 0$, accept y as the new state and set $x = y$.

step 3: Else, if $e^{-(f(y)-f(x))/c_i} >$ random [0, 1), then accept y as the new state and set $x = y$.

step 4: Lower C_i to c_{i+1} and increment i.

step 5: If i $\leq$ L, go to step 1.

step 6: Report all points x for which the best function value is achieved.

One immediately notes that, similar to the basic SA algorithm, there are several portions of this algorithm that are chosen at the discretion of the implementor. Firstly, one must choose an initial value for the cooling function. This parameter must be chosen sufficiently large so that approximately all transition stages are acceptable at this value. Let $\chi(c)$ be the ratio between the number of accepted transitions and the number of proposed transitions. The cooling parameter can be chosen by taking a sample set and requiring that the initial acceptance ratio $\chi_o = \chi(c_0)$ is close to 1. Next perform m_0 samples with $m_0 = m_1 + m_2$, where m_2 is the number of accepted transitions, and m_1 is the number of unacceptable transitions. Let $\overline{\Delta f^+}$ be the average value of those $\Delta f_{x,y}$ values for which $\Delta f_{x,y} = f(y) - f(x) > 0$. Then choose c_0 according to the formula

$$c_0 = \overline{\Delta f^+} \left(\ln \frac{m_2}{m_2 \chi_0 + (1 - \chi_0) m_1} \right)^{-1} .$$

Similarly, the cooling schedule can be formulated in many different ways. One such method is

$$c_{i+1} = c_i \left(1 + \frac{c_i \ln(1 + \delta)}{3\sigma(c_i)} \right)^{-1} ,$$

where $\sigma(c)$ is a small positive number and denotes the standard deviation of the values of the cost function of the points in the Markov chain of x at c_i. The constant δ is called the distance parameter and determines the speed of the decrement of the control parameter [15].

Finally, one must determine how to generate a new point y from x. Call this procedure $g_{x,y}$. There are several requirements that must be satisfied so that the method is guaranteed to converge in a probabilistic sense. These are beyond the scope of this paper but can be found in [15]. The following methods do satisfy the necessary criteria.

First, y can be generated from x by choosing y from a uniform distribution of points on S, where S is the region over which one wants the global optimum. That is, the probability that y is generated from any given point x is $g_{x,y} = 1/m(S)$ where $m(S)$ is the Lebesgue measure of the set S. If S is simply a box, then $m(S) = \prod_{i=1}^{n}(\overline{S_i} - \underline{S_i})$. However, as mentioned before, this does not take into account any structural information about the function.

Second, one could choose

$$g_{x,y} = \begin{cases} LS(x) & \text{if } w > t \\ 1/m(S) & \text{if } w \le t \end{cases},$$

where t is a fixed number in the interval (0,1], w is a uniform random number drawn from [0,1), and $LS(x)$ is a local search procedure that generates a point y in a descent direction from x such that $f(y) \le f(x)$ but y is not necessarily a local minimum.

2.2.2 Adaptive Simulated Annealing

Simulated annealing has been further developed and modified by many different people. One of this modifications is called Adaptive Simulated Annealing (ASA). This software was developed by Lester Ingber and other contributors first released for public use in 1989 as very fast simulated reannealing (VFSR) code. This code is presently available on the WWW at
http://alumni.caltech.edu/~ingber or via anonymous ftp at ftp.alumni.caltech.edu in the directory /pub/ingber.

One problem with standard SA is that each parameter or variable is annealed or cooled at the same rate, regardless of its range, value, or sensitivity. ASA allows for different annealing (cooling) schedules for the different parameters and variables.

SA uses a fixed cooling schedule in which the temperature decays monotonically to 0 (or to a temperature close to 0). In a real world setting, it would seem reasonable to stretch out the range over which the relatively insensitive parameters are being searched. For example, if the objective function is fairly insensitive to changes in a particular variable x_i, ASA decreases the rate at which variable x_i cools. This allows ASA to explore more of the region for that particular variable. This is accomplished by periodically rescaling the annealing time. Ingber [51] calls this reannealing.

2.2.3 Multi-level, Single Linkage (Clustering Methods)

Multi-level, single linkage is one of a group of stochastic methods that are often called clustering methods. A clustering method starts with a uniform sample of points from the region S and then creates groups or clusters of close points that correspond to a common region of attraction. A cluster C is a set of points which correspond to a region of attraction and has the following properties:

- C contains exactly one local minimum x^*.
- Any descent algorithm that is started from a point $x \in C$ converges to the point x^*.
- The first two properties imply that C can contain exactly one stationary point.

For example, consider the function $(x^2 - 1)^2$, which has global minima at the points $x = \pm 1$. There are two clusters of points for this function. The first cluster is $\{x | -\sqrt{2} < x < 0\}$, and the second cluster is $\{x | 0 < x < \sqrt{2}\}$ (see figure 2.1). Any descent method that starts in cluster 1 is guaranteed to converge to the point $x = -1$. Similarly, a descent method starting at any point in cluster 2 is guaranteed to converge to the point $x = 1$.

Clustering methods have several techniques to identify these clusters, but the basic algorithm is still the same. A seed point is chosen which

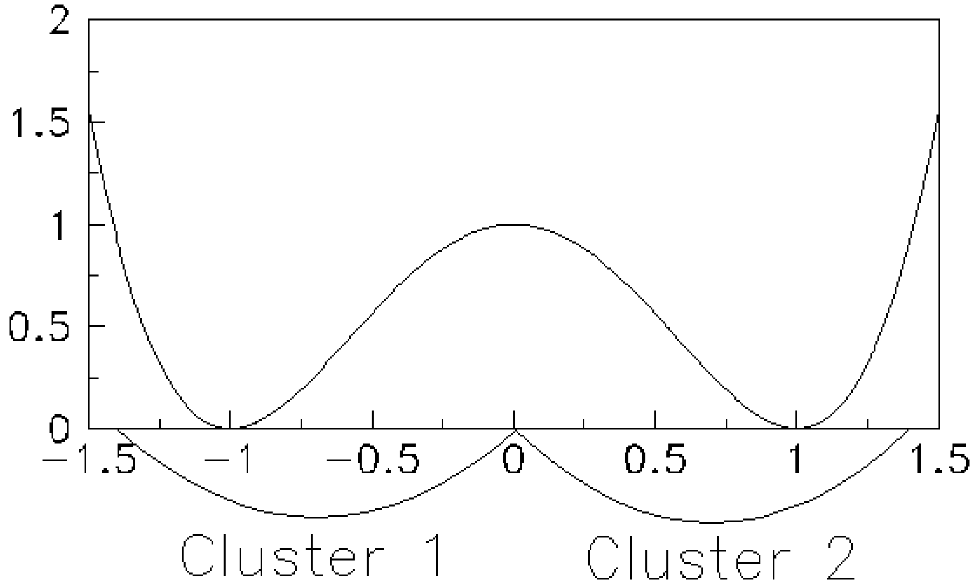

Figure 2.1: Clusters of the function $(x^2-1)^2$

is often the point with the lowest function value obtained so far or a local minimum. Points are added to this cluster through the application of a clustering rule or until some termination criterion is satisfied. Methods differ on the clustering rule and on the termination criterion.

Two of the more common methods are based on either the density [98] of the cluster or on the distances between the points. In the first method, the termination criterion states that the number of points per unit volume within the cluster should not drop below a certain value. For the second method, the termination criterion states that a certain critical distance between points of the cluster may not be exceeded.

Single linkage is the specific name given to one type of clustering method. It begins by evaluating the function at a set of sample points. It then applies a local optimizer procedure (LO) to the sample point with the lowest function value and starts a cluster about the resulting local minimum. It then adds points to the current cluster until the distance of the unclustered point to the cluster exceeds the critical distance. The distance between a point x and a cluster is defined as the minimum distance between x and any point in the cluster. Once no more points can be added to the cluster, the cluster is said to be terminated. If there are still unclustered points, then the (LO) is applied to the unclustered sample point with the lowest function value and a new cluster is formed around the resulting local minimum.

The method as found in [59] is actually implemented slightly differently in an iterative fashion. The points are sampled in groups of fixed size M, and a continually expanded sample set is re-clustered at each iteration. If X^* is the set of local minima from the previous iterations, the elements of X^* are used as the seed points for the first clusters.

Suppose the critical distance d_k for single linkage in iteration k for some $\sigma > 0$ is chosen such that

$$d_k = \pi^{1/2} \left[\Gamma(1 + \frac{n}{2}) m(S) \sigma \frac{\log(kM)}{KM} \right]^{1/n}, \tag{2.1}$$

where $m(S)$ is the Lebesgue measure of the region S and

$$\Gamma(n) = \int_0^\infty x^{n-1} e^{-x} \; dx$$

is the Gamma function. The resulting method has strong theoretical properties. If $\sigma > 4$, then even if the sampling continues forever, the total number of local searches started by single linkage can be proved to be finite with probability 1 [58]. Since the method cannot run forever, multiple local (even global) minima can remain undetected in a single given cluster and can remain undetected since only one local minimizer is applied to each cluster.

Multi Level Single Linkage (MLSL) [59] differs since it is guaranteed to find all global optima with probability 1. The method is applied to an ever expanding sample set like single linkage. The LO is then applied to every sample point unless there is another sample point within the critical distance with a smaller function value. Formulated in this way, MLSL does not form clusters and can be applied to the entire sample set without reduction or concentration. Reduction is the act of discarding a certain fraction of the sample points with the highest function values. Concentration transforms the sample by allowing one, or at most a few, steepest descent steps from every point.

MLSL may appear quite naive but, similar to single linkage, if the critical distance is chosen according to (2.1) with $\sigma > 4$, and if the sampling continues forever, the number of local searches started is finite with probability 1 [96]. Furthermore, single linkage is not guaranteed to find the global optima. MLSL has been shown in [96] to find all local minima (including the global minima) if the algorithm is allowed to run forever. More precisely, consider the connected component for a level set $L(y) = \{x|f(x) \leq y\}$ containing a local minimum x^*. Define the basin $Q(x^*)$ as the largest value y for which every application of LO converges to x^*. Timmer then shows that in the limit, if a point is sampled in $Q(x^*)$, then the local minimum x^* is found. If one chooses $\sigma > 0$ (2.1), any local minimum x^* is found by MLSL in a finite number of iterations with probability 1.

2.3 Deterministic Methods

Deterministic algorithms differ from stochastic algorithms in several ways. First, stochastic algorithms never give a 100% guarantee of having located the global optima since a guarantee of that quality requires that the algorithm run for infinite time. Second, deterministic algorithms always run in finite (although quite long) time and return a set of regions that are guaranteed to contain all global optima. Finally, stochastic algorithms are not guaranteed to find all global optima of a function but are guaranteed to find one global optima when there may be many alternative global optima. Deterministic algorithms return a guarantee that all global optima are contained in the regions that are returned at the algorithm's termination.

This section is not meant to be a complete guide to the history of global optimization. In fact, several methods are left out intentionally. The research contained herein strives to focus on problems that were more general in nature such as concave objective function with convex constraints rather than problems designed to solve a specific type of problem.

2.3.1 Cutting Plane Methods

Cutting plane methods are best applied to concave minimization. That is, they are best applied to minimizing a concave function over a convex set. According to Horst and Tuy [49] "Many applications lead to minimizing a concave function over a convex set. Moreover, it turns out that concave minimization techniques also play an important role in other fields of global optimization." There are three main techniques that are generally applied to concave minimization: cutting methods, successive approximation, and successive partition methods. This research explores only cutting plane methods, but a good reference for all these methods can be found in [49].

The basic concave program (BCP) can be written as follows:

$$\begin{array}{rccc} & \min & f(x) & \\ \text{s.t.} & Ax & \leq & b \\ & x & \geq & 0 \end{array}$$

where A is an $m \times n$ matrix, x is an m-vector, $f : \Re^n \rightarrow \Re$ is a concave function, and $D = \{x | Ax \leq b, \ \ x \geq 0\}$. It can be shown that the minimum of any bounded concave program occurs at a vertex of the polytope defined by A, b, and the non-negativity constraint. Therefore, the solution to the BCP can be re-written as:

$$\min f(x^i) \quad \text{where } x^i \text{ is a vertex of the polytope P}$$

Define a point x^0 to be a local BCP minimizer if $f(x^0) \leq f(x)$ for any x in the convex hull of x^0 and the adjacent vertices $y^1, y^2, \ldots, y^s$. Let $\gamma = f(x^0)$. For any $x \in \Re^n$ satisfying $f(x) \geq \gamma$, the point $x^0 + \theta(x - x^0)$ such that

$$\theta = \sup\{t | t \geq 0, f(x^0 + t(x - x^0) \geq \gamma\}$$

is called the γ-extension of x with respect to x^0. Now, let θ_i be the γ extension of x^0 with respect to y^i. It can be shown that any solution π of the system of linear inequalities

$$\theta_i \pi(y^i - x^0) \geq 1 \quad (i = 1, 2, \ldots, s) \tag{2.2}$$

where x^0 is a local BCP minimizer. This follows since $\min\{f(y^1), \ldots, f(y^s)\} \geq f(x^0)$. From this result the following theorem can be given.

Theorem 1 (Sufficient condition for global optimality see [49], page 179). Let π be a solution of the system 2.2. Then

$$\pi(x - x^0) > 1 \text{ for all } x \in D \text{ such that } f(x) \leq f(x^0)$$

Hence, if

$$\max\{\pi(x - x^0) | x \in D\} \leq 1,$$

then x^0 is a global optimal solution of BCP.

This gives a quick and easy method to check for optimality of a given point by solving the LP

$$\max\{\pi(x - x^0) | x \in D\},$$

where $\pi(x - x^0) \geq 1$ is a valid cut for the BCP. If the optimum value of this LP is less than 1, then x^0 is a global optimal solution. If not, the solution must be found in the region

$$D \bigcap \{x | \pi(x - x^0) \geq 1\}.$$

Hence the name "cutting plane" algorithms. The basic algorithm finds a local optimum. If that local optimum is not a global optimum, then the algorithm cuts away the region of the domain that contains that solution and proceeds to find another local optimum. This continues until the global optimum is found.

Algorithm 4 : Cutting Plane [49]

initialization: Locate a local BCP minimizer x^0. Set $\gamma = f(x^0)$, $D_0 = D$, $k = 0$.

step 1: At x^k, construct a γ-valid cut π^k for (f, D_k).

step 2: Solve the linear program $\max \pi^k(x - x^k)$ s.t. $x \in D_k$. Let ω^k be a basic optimal solution of this linear program. If $\pi^k(\omega^k - x^k) \leq 1$, then stop: x^0 is a global minimizer. Else, go to step 3.

step 3: Let $D_{k+1} = D_k \bigcap \{x | \pi^k(x - x^k) \geq 1\}$. Find a vertex x^{k+1} which is a local BCP minimizer of $f(x)$ over D_{k+1}. If $f(x^{k+1}) \geq \gamma$, set $k = k + 1$ and go to step 1. Else go to step 4.

step 4: Set $\gamma = f(x^{k+1})$, $k = k + 1$, and goto step 1.

The concavity cuts that are used in Algorithm 4 are easy to construct. The cuts that are constructed tend to become shallow as the algorithm progresses. The volume of the area removed by the cuts can approach zero long before enough is cut away to find the global minimum. This causes slow convergence and can cause the algorithm to not converge at all. This has made the research look to more expensive but deeper cuts in order to guarantee finiteness. This dissertation does not describe in detail the different methods for doing this but again refers the reader to [49].

2.3.2 Branch and Bound History

Branch and bound is one of the few deterministic methods that can be applied to a wide class of problems with little or no knowledge of the state of the function that is to be minimized. It can be applied quite easily to methods with concave or convex domains, concave or convex objective functions, and functions for which no information on continuity is available. While one does generally have some information about the function, some branch and bound methods do not require this in any form. The version we develop here exploits information about the function as it is actively discovered. However, the algorithm finds the information that it needs rather than requiring the user to provide the information at the start. The algorithm computes all the needed derivatives and then uses this information to exploit the structure of the function.

The earliest paper this author found that presents a branch and bound algorithm with finite convergence proofs is the 1969 paper by Falk and Soland [23]. Many people had thought about using branch and bound to solve the nonlinear optimization problem. In particular, the ideas of global optimization through branch and bound had been around for a few years at this point. In 1966, Lawler and Wood [62] wrote an article that surveyed branch and bound methods. They concentrated on methods for integer programming but do mention that several people had considered and suggested branch and bound methods for solving nonlinear programming problems. For example, they also refer to a 1963 presentation by Beale [4] in which he applies branch and bound to solve nonlinear transportation problems. Lawler and Wood state that this work was not investigated completely, but Beale's paper shows that the ideas for branch and bound have been considered for some time. The presentation by Beale was then accepted for publication in 1964. This is approximately the same time as Tuy's [99] first publication on the subject.

Falk and Soland's algorithm concentrated on finding the minimum of a separable objective function over a rectangular domain (bounds on the variables, i.e. a box). They subdivide the region using a rectangular partition, and they bound the objective function by using a convex underestimating nonlinear function. The nonlinear function is derived by computing the convex envelope of the objective function over each rectangular region. The convex envelope of f on D is the supremum of all convex functions that underestimate the function f on D. The resulting function is convex, and it can be shown that the minimum of a continuous function f on a compact set is the same as the minimum of the convex envelope of f on that same set. At each step, the algorithm splits along the variable for which the greatest difference between the convex envelope and the function is greatest. By continuing to reduce the size of each partition, the algorithm is able to improve the fit of the convex envelope at each iteration.

Greenberg [29] showed that Falk and Soland's result could be improved upon in terms of computational time as well as accuracy of the underestimating function. He shows that one can use the Lagrangian dual to compute a lower bound on the function rather than using a convex underestimating function. He goes on to show that in the worst case the Lagrangian dual gives the same result as the convex underestimating function and is potentially better. He also shows that the Lagrangian dual is less expensive to compute and gives some economic interpretation of the presence of gaps. Additionally, Greenberg gives some directions for further research.

Falk and Soland's algorithm was improved by Soland [92] so that it could handle non-convex constraints. The constraints are treated in a manner similar to that of the objective function. The constraints are replaced by their convex envelopes. Soland was able to prove convergence, but it was possible that the convergence was asymptotic.

At the same time, Beale and Tomlin [5] introduced their global optimization algorithm which was based on the concept of using ordered variables. The objective function was approximated with piecewise linear functions, and the problem was solved as an LP. For more information on the early work in branch and bound methods for global optimization, see McCormick [64] who provides a summary of the early work and feels that branch and bound methods held much promise for global optimization.

McCormick [65] extends the early work of Falk and Soland to allow for factorable functions. McCormick presents rules to generate convex and concave bounding functions for factorable functions which appear on many NLPs. These extensions allow branch and bound to solve most NLPs and is used by Epperly [21] to handle bilinear constraints.

In 1976, Horst [43] frees the original Falk and Soland algorithm of the restriction to rectangular partitions and the restriction to convex underestimating functions. Horst uses compact partitions and general underestimating functions in their places and then provides the necessary conditions on these partitions and functions to prove convergence. He provides an algorithm that uses n-dimensional simplices to partition the domain. Even though he proves that convex envelopes are unnecessary, he uses them in his algorithm for simplicity. The use of n-dimensional simplices also makes his algorithm much more efficient for concave functions since the n-dimensional simplices have only $n + 1$ vertices, while rectangles have 2^n vertices. He proves that the algorithm must give at least one accumulation point which is a solution of the original global optimization problem. In a later paper [44], he proves that every accumulation point must be a solution to the GOP.

Another algorithm was given by Thoai and Tuy [94]. They present a convex branch and bound algorithm which uses conical partitions. They develop the necessary consistency and criteria required for convergence. The problems which their algorithm attacks have a concave objective function and a polyhedral constraint set. They use an LP which is based on Tuy's cutting planes [99] to obtain lower bounds for each convex region.

We have now presented the three main partitioning methods used in branch and bound global optimization algorithms. These methods are rectangular, simplicial, and conical. Rectangular partitions are generally used by methods that use convex envelopes such as the algorithm developed by Falk and Soland [23] or algorithms that employ interval arithmetic such as the algorithm developed by Hansen [35]. Simplicial and conical partitions are generally used to solve concave minimization problems. This research examines these ideas more in depth in section 2.3.5.

In 1982, Benson [6] generalized Falk and Soland's [23] and Horst's [43] algorithms to an algorithm that was less restrictive on the types of underestimating functions used. Benson shows that every accumulation point is a solution of the GOP and, since Falk and Soland's work is a special case, shows that for their algorithm, every accumulation point is a solution of the GOP. Unfortunately, no computational results have been reported, so it is difficult to judge the utility of this work.

Rosen [86] presents a unique method of minimizing a concave function over a convex set. Rosen first computes a local maximum of the function by using a local search method. He then examines the eigenvectors of the Hessian matrix at the local maximum in order to estimate which feasible vertices are far from the maximum by using a multiple cost row LP. The vertex that yields the lowest objective functional value is then used an an upper bound for the global minimum and is used to create a set in which the minimum cannot occur due to the concavity of the objective. He can then exclude this region from the search leaving at most $2n$ polyhedral subsets that can be solved using the Falk-Hoffman algorithm [22]. It is hoped that the algorithm converges quickly since small simplices can be given from the first step of Rosen's method. Rosen applied his algorithm to several problems and provides a bound on the amount of work required.

Tuy, Thieu, and Thai [103] provide an improved version of the Thoai-Tuy algorithm that no longer required a bounded polyhedral feasible region. They develop a conical branch and bound algorithm that works for concave objective functions defined over a closed convex set with the possibility of the set being unbounded. They explore several different splitting strategies for the method including splitting the largest cone, splitting the oldest cone, and splitting the most promising cone. They use a supporting hyperplane of the convex set to obtain an underestimating function over a cone.

In [45], Horst generalizes the Benson-Falk-Horst-Soland algorithm. The latter is often referred to as the BFHS algorithm developed by Benson [6] (not to be confused with the BFGS local optimization method). It includes the branch and bound algorithm that Tuy, Thieu, and Thai [103] developed. Horst creates some extensions to the original method which allows for region deletion and different variable selection rules. He shows that the general algorithm has convergence in the limit and provides a concrete algorithm for minimizing concave functions over a compact and convex sets.

Horst and Tuy [42] show that many other algorithms are special cases of a general branch and bound algorithm. They show that Pintér's algorithms [79, 80], the algorithm by Zheng and Galperin [27, 28, 109], and the algorithms of Pijavskii [78], Shubert [91], and Mladineo [67] are all special cases of their general branch and bound algorithm. In a later paper, Tuy and Horst [102] review the convergence criterion for a general branch and bound algorithm. They present a restart branch and bound algorithm for use when the feasible set is overestimated using feasible cuts. Their algorithm introduces new cuts for the feasible region as it progresses. They provide concrete algorithms for concave D.C. programs (programs that can be expressed as the difference of two convex programs).

Benson, Horst, and Tuy [6, 45, 102] develop algorithms that assume it is easy to find feasible points inside each of the partitions. While this may be true for convex regions, it is not the case for general constraints. In fact, determining feasibility can be just as difficult as finding the global optimum. Horst [46] extends his previous general branch and bound algorithm and convergence proofs to apply when the feasibility of the partition sets is unknown. Additionally, Horst and Thoai [48] use this modification to solve systems of Lipschitzian equations and inequalities.

In [41], Horst, Thoai, and Benson refine their algorithm to present a faster version for branch and bound for concave minimization over a convex feasible region. They use conical partitions again along with polyhedral outer approximation to the convex region. Both of these modifications give a lower bound step that is a linear program rather than just a linear objective function with convex constraints. This is the reason for the speed-up observed in this modification.

Most of the algorithms presented thus far assume a bounded feasible region. Hamed and McCormick [33] investigate how to treat a problem for which there are no bounds on the variables. They investigate several procedures in an attempt to provide bounds on the variables. Each procedure depends on the type of NLP that is being solved.

2.3.3 Branch and Bound

Branch and bound, like SA, is a technique that has been successfully applied to integer optimization problems. In this context, a general integer program is given by (IP): $\max\{cx|x \in S\}$, where $S = \{x \in Z_+^n|Ax < b\}$. Let S_R be a relaxation of the set S to some larger set so that $z_R(x) \geq cx$. For example, a common relaxation of a integer program is to its linear counterpart (RP$_{lp}$): $\max\{cx|x \in S_{lp}\}$, where $S_{lp} = \{x \in \Re_+^n|Ax < b\}$. Then the general integer branch and bound algorithm is given in algorithm 5 (see [73]).

Algorithm 5 : Integer Branch and Bound

Initialization: Set $L = IP$ (L is a list of IPs, each with its own portion of the domain), and $S^0 = S$, $\overline{z}^0 =$ machinf, and $\underline{z}_{IP} =$ machinf.

step 1: (Termination test): If $L = \emptyset$, then the solution x^0 that yielded $\underline{z}_{IP} = cx^0$ is optimal. If there is no such x^0, then return infeasible.

step 2: (Problem selection and relaxation): Select and delete a problem from L. Solve a relaxation of (IP): RP^i. Let z_R^i be the optimal value of the relaxation and let x_R^i be an optimal solution if one exists.

step 3: (Pruning):

- **a** : If $z_R^i \leq \underline{z}_{IP}$, go to step 1.
- **b** : If $x_R^i \notin S^i$, go to step 4.
- **c** : if $x_R^i \in S^i$, and $cx_R^i > \underline{z}_{IP}$, let $\underline{z}_{IP} = cx_R^i$. Delete from L all problems with $\overline{Z}_i \leq \underline{z}_{IP}$. If $cx_R^i = z_R^i$, go to step 1. Otherwise, go to step 4.

step 4: Let $\{S^{ij}\}_{j=1}^k$ be a division of S^i such that $S^i = \bigcup_{j=1}^k S^{ij}$. Add problems $\{IP^{ij}\}$ to L, where $\overline{z}^{ij} = z_R^i$ for $j = 1, \ldots, k$. Go to step 1.

In step 2 if the entire set of subsets of S has been pruned, then the last integer solution found was optimal. If not, then a set is pulled off of the list, and an upper bound on the value of IP is computed for the set S^i.

If this upper bound is less than the best computed integer solution computed to this point, then the set S^i is pruned. If the solution is integral and better than the best solution so far, then the set of IPs that remain are searched for sets that can be pruned.

The branch portion of branch and bound becomes obvious when step 5 partitions S^i into k disjoint subsets. IPs are then generated that have these subsets for their domain, and they added to the list L and the algorithm continues.

This algorithm has been adapted to solve the Global NLP problem in a straight forward fashion by using some relaxation to bound the function value over a portion of the domain. Given the general NLP problem $\max f(x)$ st $x \in R \subseteq \Re^n$, a naive algorithm is as follows.

Algorithm 6 : Continuous Branch and Bound

Initialization: Set $i = 1$, $z^* =$ mach-inf, and list $= R$.

step 1: Pull a region R_i off of the list.

step 2: If the region R_i is too small, put it on the list of finished boxes and go to step 1.

step 3: Compute z_-, z^+ such that $z_- \leq f(x)|_{R_i} \leq z^+$.

step 4: If $z^- > z^*$ discard the region R_i and go to step 1.

step 5: If $z_+ \leq z^*$ set $z^* = z_+$.

step 6: Split the region R_i into k equal sized pieces and add them to the list.

step 7: Go to step 1.

step 8: Return the list of finished boxes.

Algorithm 6 covers the region R with smaller and smaller boxes, eliminating the ones where it is impossible for a global minimum to exist since we have bounded the values of that region to be less than some lower bound on the global minimum.

This basic algorithm suffers from several potential difficulties. First, the bounds computed for the function over a region that is used in step 2 are potentially not tight. If the bounds computed are sufficiently bad, then the algorithm could spend a large amount of time attempting to eliminate regions that are close to the global minimum, but whose objective function is very flat. Second, in step 6, the initial box is divided into k equal sized pieces which does not take into consideration any information such as local/global behavior of the function over any given region. A more intelligent algorithm would take such information into consideration when determining where to split. For example, if the box consists of the square [-1, 1]×[-1, 1] and $f = x^2$, the box could be divided into four boxes, each with one corner at [0, 0]. This may not be a good partition since each box still contains the global minimum of the function. However, if the box were split into the boxes

$$[-\epsilon,\epsilon]\times[-\epsilon,\epsilon],[\epsilon,1]\times[-1,1],[-1,-\epsilon]\times[-1,1],[\epsilon,\epsilon]\times[\epsilon,1],[-\epsilon,\epsilon]\times[-1,-\epsilon],$$

where ϵ is a small positive number, we now have five boxes, only one of which contains the global min, and the other four can be easily eliminated.

It would be an advantage to have an algorithm which could identify these boxes so that the region can be split more intelligently.

Third, regions can be eliminated on the basis of information other than just function values. If the first derivative is strictly positive (negative) over a region, that region can be eliminated as it cannot contain even a local minimum. Similarly, a region can be eliminated if the Hessian matrix cannot be positive definite on that region. Fourth a region could be partially eliminated using a combination of first and second order information. These ideas are explored in greater depth in section 2.5.1.

There are two basic features to any branch and bound algorithm. First, every algorithm requires some sort of bounding function for the objective. Second, the algorithm must determine how to partition the region after the bound has been computed. This research examines both these features and develop new approaches that allow high dimensional global optimization problems to be solved. We then present a new method for partitioning the region and give computational results that compare the new method with an older one.

2.3.4 Bounding Functions

It is desirable to have a bounding function that quickly gets accurate bounds on the values of a function when applying a branch and bound method for global optimization. Unfortunately, the desire to have both an accurate as well as a quickly computed bound is usually contradictory. We now explore a variety of bounding functions. Many of them are applicable to only specific types of problems such as concave minimization problems or separable functions.

2.3.4.1 Bounding Based on Lipschitz Constants

A potentially inexpensive but inaccurate bound on the function is given by

$$B_{L,R}f = f(x^*) - Lw(R) \leq f(x) \leq f(x^*) + Lw(R) = B^{L,R}f,$$

where $x^* \in R$, L is the Lipschitz constant for f on R, and $w(R)$ is an upper bound on the diameter of the region R. This bound has the advantage of being very inexpensive to compute provided that the Lipschitz constant is available at no cost. In addition it has the property that

$$\lim_{w(R)\to 0} B^{l,R} - B_{l,R}f = 0.$$

However, this method also has the disadvantage of potentially overestimating the bound if the constant l is unnecessarily large. Also, the bounds do not take advantage of the fact that, for most functions, a Lipschitz constant may vary greatly over different regions of the function.

2.3.4.2 Interval Arithmetic

Another method of bounding a function is based on interval arithmetic. Interval arithmetic was invented by Moore [68] and has been used recently for solving ordinary differential equations, linear systems, verifying chaos, and global optimization.

Define an interval $X = [\underline{x}, \overline{x}]$, where $\underline{x}$ and $\overline{x}$ are real numbers. Then, for any binary operation $\circ$ where $\circ$ is one of $+$, $-$, $/$, or $*$, and X and Y any two intervals, we desire

$$X \circ Y = [\min_{\substack{x\in X\\ y\in Y}} x \circ y, \max_{\substack{x\in X\\ y\in Y}} x \circ y].$$

Similarly, for a unary function $\phi(x)$, we desire

$$\phi(X) = [\min_{x \in X} \phi(x), \max_{x \in X} \phi(x)].$$

Let $X = [\underline{x}, \overline{x}]$ and $Y = [\underline{y}, \overline{y}]$ be intervals. We find that

$$\begin{aligned}
X + Y &= [\underline{x} + \underline{y}, \overline{x} + \overline{y}] \\
X - Y &= [\underline{x} - \overline{y}, \overline{x} - \underline{y}] \\
X * Y &= [a, b] \\
a &= \min\{\underline{x}\underline{y}, \overline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\overline{y}\} \\
b &= \max\{\underline{x}\underline{y}, \overline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\overline{y}\} \\
\frac{1}{Y} &= \left[\frac{1}{\overline{y}}, \frac{1}{\underline{y}}\right] \quad 0 \notin Y \\
\sqrt{X} &= [\sqrt{\underline{x}}, \sqrt{\overline{x}}] \quad X \geq 0 \\
\ln X &= [\ln \underline{x}, \ln \overline{x}] \quad X \geq 0 \\
e^X &= [e^{\underline{x}}, e^{\overline{x}}]
\end{aligned}$$

For non-monotonic computer functions such as $\sin(X)$, we can treat these functions by using the periodicity of these functions. Also, on an automaton, one must take care to round outwards the result of any operation on intervals so that the result is mathematically correct.

For example, let $f(X, Y) = 2X + 3XY - \sqrt{X}$ and let $X = [1, 4]$ and $Y = [-1, 2]$. One way of computing $f(X, Y)$ is as follows.

$$\begin{aligned}
T1 &= 2X = 2 * [1, 4] = [2, 8] \\
T2 &= X * Y = [1, 4] * [-1, 2] = [-4, 8] \\
T3 &= 3 * T2 = 3 * [-4, 8] = [-12, 24]
\end{aligned}$$

$$\begin{aligned}
T4 &= T1 + T3 = [2,8] + [-12,24] = [-10,32] \\
T5 &= \sqrt{X} = \sqrt{[1,4]} = [1,2] \\
T6 &= T4 - T5 = [-10,32] - [1,2] = [-12,31] \\
\text{true range} &= [f(4,-1), f(4,2)] \\
&= [-6,30]
\end{aligned}$$

We can now take the above operations and apply them to any function. The real operations are replaced by interval operations and all the intermediate results are intervals. This provides a function with guaranteed enclosures of the range of any function. Also (see [69]) we have the guarantee that

$$\lim_{w(X)\to 0} F(X) = F(x)$$

where F is the interval inclusion function of f over R and f is continuous on X.

Interval inclusion functions (of which there are many different types) have the advantage of being easy to implement using an object oriented language such as C++ or Ada. It also does not require any *a prioi* information about f. However, computing this interval bound carries a cost of 2 to 4 times as much effort as evaluating f [74].

2.3.4.3 Interval Taylor's Expansion

Any $n+1$ times continuously differentiable function f in an interval about x^* can be written as

$$f(x) = \sum_{i=0}^{n} \frac{f^{(i)}(x^*)(x-x^*)^i}{(n!)} + R_n(x,\xi), \tag{2.3}$$

where

$$R_n(x,\xi) = \frac{f^{n+1}(\xi)(x-x^*)^{n+1}}{(n+1)!},$$

and ξ lies on the line segment connecting x and x^*. Interval arithmetic can quite readily take advantage of this Taylor's expansion of f at any point x^* in an interval X. Evaluate (2.3) where $x^* \in X$, and $(x - x^*) = X - x^*$. Add the interval remainder term $R_n(x,\xi)$ by evaluating the $n+1$st derivative over the interval X and multiply this value by $(X - x^*)^{n+1}$. Define this value to be the n^{th} order Taylor polynomial $T_n(X)$ and write it as

$$T_n(X) = \sum_{i=0}^{n} \left(\frac{f^{(i)}(x^*)(X-x^*)^i}{(i!)} \right) + \frac{f^{n+1}(X)(X-x^*)^{n+1}}{(n+1)!}.$$

Note that for $n = 1$, this is similar to the Lipschitz method except that an interval of values is used for the Lipschitz constant. This is also known (see [74]) as the **generalized mean value form** and is written

$$T_1(X) = f_m(X,\tilde{z}) = f(\tilde{z}) + f'(X)(X - \tilde{z}),$$

where the Taylor's expansion is computed at the point $\tilde{x} \in X$.

As with the other methods,

$$\lim_{w(X)\to 0} T_n(X) = f(x).$$

Furthermore, if f^{n+1} is continuous and R is a box with sides parallel to the coordinate axis, then the interval Taylor's expansion has the property that

$$\lim_{n\to\infty} T_n(X) = \{y | y = f(x), x \in X\}.$$

Traditional methods for computing $f^{(n)}(X)$ require exponential computational effort as $n \to \infty$ if f has non-trivial n^{th} derivatives. Automatic differentiation [55] allows the computation of the n^{th} term of $T_n(X)$

for at most a constant times the work required to compute $f(x)$. In particular, for $n = 1$, $\nabla f(R)$ can be computed for at most 5 times the effort required to compute $f(R)$ itself [55].

2.3.4.4 Slope Form

Neumaier [74] suggests a method for including the range of a function that is similar to using a first order Taylor method. Let f be a differentiable function of one variable. Let the standard divided difference of f be defined by

$$f[x,y] = \begin{cases} \frac{f(y)-f(x)}{y-x} & \text{if } x \neq y \\ f'(x) & \text{if } x = y \end{cases}.$$

When f is twice continuously differentiable one can write

$$f(\tilde{x}) = f(\tilde{z}) + f[\tilde{z},\tilde{z}](\tilde{x} - \tilde{z}).$$

If this holds for $\tilde{x} \in x$ and if $f[x,y]$ is an arithmetical expression in x and y, the slope form is written as

$$f_s(x,\tilde{z}) = f(\tilde{z}) + f[\tilde{z},x](x - \tilde{z}).$$

For example, consider

$$\begin{aligned} f(x) &= x^3 - 2x^2 + 3x - 1 \\ f'(x) &= 3x^2 - 4x + 3 \\ f[x,y] &= x^2 + xy + y^2 - 2(x+y) + 3 \\ x &= [0,1]. \end{aligned}$$

Then

$$f[x,1/2] = [1/2,4]$$

and so,

$$f_s(x, 1/2) \;=\; 1/8 + [1/2, 4][-1/2, 1/2] = [-15/8, 17/8].$$

Note that the simple interval extension of f and the mean value form give

$$\begin{aligned} f([0,1]) &= [-3, 3] \\ f_m(x, 1/2) &= 1/8 + [1, 6][-1/2, 1/2] = [-23/8, 25/8] \end{aligned}$$

which are both larger than the slope form. In fact, for the same center, the slope form gives consistently more accurate results than the generalized mean value form [74].

This presentation of the slope form applies to functions of a single variable. If the number of variables is greater than 1, then the slope form is not unique. For example, for two variables ξ_1, ξ_2 and $f(\xi) = \xi_1\xi_2$, both $f[\tilde{x}, \tilde{x}] = (\tilde{x_2}, \tilde{z_1})$ and $f[\tilde{x}, \tilde{x}] = (\tilde{z_2}, \tilde{x_1})$ are slopes which shows that the slope form is not unique.

2.3.4.5 Convex Envelopes

In their first paper on the subject, Falk and Soland [23] by using the convex envelope (the supremum of all convex functions less than or equal to f) of the function. In fact, if the function f is separable and the set over which the convex envelope is desired is given by $C = \{x | l \leq x \leq L\}$, then the convex envelope Ψ of f can be computed as follows. Let

$$\Psi(x) = \sum_{i=1}^{n} \Psi_i(x) = \sum_{i=1}^{n} \sup_{t_i} \{x_i t_i - \rho_i(t_i)\},$$

where

$$\rho(t) = \sum_{i=1}^{n} \rho_i(t_i) = \sum_{i=1}^{n} \max_{l_i \leq x_i \leq L_i} \{x_i t_i - f_i(x_i)\}.$$

That is, the convex envelope of the function f over a rectangular partition is equal the sum of the convex envelopes of f_i taken over $C_i = \{x_i | l_i \leq x_i \leq L_i\}$

2.3.5 Partitioning Schemes

As mentioned in section 2.3.2, there are three methods of partitioning a region: cone, rectangle, and simplex. Conical partitions are generally used by methods for concave minimization problems, rectangular are used by methods that use interval arithmetic or convex envelopes, and simplicial partitions are generally only applied to concave minimization problems. One of the two main thrusts of this research is partitioning. In particular, we have developed a new method for a partitioning scheme that improves the performance of a validated branch and bound algorithm when the accuracy desired is quite small or when the dimension of the problem is large.

Falk and Soland [23] used rectangular partitions in their original algorithm. This has been the partitioning method of choice for any algorithm that employs either interval arithmetic or convex envelopes. Interval arithmetic uses rectangular partitions since an interval vector X in n-space is a rectangle. Hence, partitioning into rectangles is natural. Convex envelopes use rectangular partitions because they work so well for finding an underestimating convex function.

Schemes that are used for determining where to partition a given rectangle can be quite varied. One can bisect regions along the longest edge, bisect according to the most promising variable, or bisect according the size of the gradient. More generally, one can cut the region into k regions where k is any integer. Hansen [35] has implemented several of these. He recommends trisecting along the longest edge. These ideas were also explored by Csendes and Ratz [14, 84, 85]. They explored chopping the box into 2 to n pieces where n was the dimension of the function. They also explored using the Hansen-Sengupta operator (see section 2.4.3) to determine where to divide the box and they had some success with this method. In order to maintain the theoretical convergence, one must guarantee that the length of the longest edge tends to 0 as the algorithm progresses. Dividing the region into k equal sized pieces satisfies this requirement. We have developed a new method for dividing a rectangular partition that takes into account information about convexity and/or monotonicity of the function around a local optimum of a rectangular partition.

By contrast, Horst and Tuy [49] give a process which does not take into consideration any information about convexity or monotonicity.

This is a normal simplicial subdivision process for the optimization of a concave function over a convex but not necessarily polyhedral set. They let an n-simplex $M = [v^1, v^1, \ldots, v^{n+1}]$ be a subsimplex of M_0 which contains the feasible region D and let $\phi_M(x)$ be the convex envelope of the function f over M. $\phi_M(x)$ is obtained by finding the affine function that agrees with $f(x)$ at each vertex v^i of M. The linear program

$$\min \phi_M(x) \;\; \text{s.t.} \;\; x \in D \cap M$$

has a basic optimal solution $\omega(M)$ with value $\beta(B)$. Since $\phi_M(x)$ is a convex (also affine) underestimating function of f, we know that $\beta(M) \leq \min f(D \cap M)$.

A basic simplicial subdivision process is given by:

Algorithm 7 : Simplicial Subdivision Process

step 1: The initial simplex is M_0.

step 2: The subdivision of any n-simplex M is a radial subdivision with respect to a point $x = w(M) \in M$ which is distinct from any vertex of M.

Note that when the underestimating function is affine as above, the resulting LP is easy to solve since the n-simplex has exactly n vertices.

Conical partitioning is also generally implemented for the optimization of a concave function over a convex set. There are two main methods of conical partitioning: bisection and normal subdivision.

Following the development by Horst and Thoai [49], let S=conv$\{v^1, \ldots, v^{n+1}\}$ be an n-simplex defined as the convex hull of its vertices $v^1, \ldots, v^{n+1}$, and let $0 \in S$. Denote by F_i the facet of S that is opposite to v_i i.e. $v_i \notin F_i$. Clearly F_i is an $(n-1)$-simplex. Let $C(F_i)$ be the convex polyhedral cone generated by the vertices of F. It is well known that

$$\bigcup_{i=1}^{n+1} C(F_i) = \Re^n,$$

and that $\{C(f_i) | i = 1, \ldots, n+1\}$ is a partition of $\Re^n$.

Now let $C = C(u^1, \ldots, u^n)$ be a polyhedral cone generated by n independent vectors $u^i \in \Re^n$. Then every $r \in C$ can be written as

$$r = \sum_{i=1}^{n} \lambda_i u_i \quad i = 1, \ldots, n, \text{ where } \lambda_i \geq 0.$$

If $r \neq 0$ and $r \neq u^i$, $i = 1, \ldots, n$, then for each i satisfying $\lambda_i > 0$ in 2.3.5, we have

$$C_i = C(u^1, \ldots, u^{i-1}, r, u^{i+1}, \ldots, u^n).$$

It is easy to see that $\{C_i | \lambda_i > 0\}$ is a partition of C. We have

$$\bigcup_{\lambda_i > 0} C_i = C,$$

and

$$\text{int}(C_i) \bigcap \text{int}(C_j) = \emptyset, \quad \text{for } i \neq j.$$

A classic method for selecting the vector r is the bisection method. Choose

$$r = (1/2)(u^{i_1} + u^{i_2}),$$

where

$$||u^{i_1} - u^{i_2}|| = \max\{||u^i - u^j|| \; |i, j \in \{1, \ldots, n\}, i \neq j\}.$$

Geometrically, bisection corresponds to choosing the midpoint of one of the longest edges of the $(n-1)$-simplex $\text{conv}\{u^1, \ldots, u^n\}$. It has been shown that a conical branch and bound method that uses bisection conical partitioning is convergent. Many more choices of r that lead to the same convergence properties can be found in [104]. However, bisection tends to be quite slow in practice [49], and other methods for partitioning the region have been developed.

Horst and Tuy [49] give a method that is a hybrid of the bisection method and an ω-subdivision process. An ω-subdivision process subdivides a region K with respect to the basic optimal solution of the linear program associated with the n-simplex. ω-subdivisions take into consideration the values at the points of the n-simplex where bisection simply throws it away in favor of the longest edge. However, this information can cause ω-subdivision to degenerate and "jam". That is, ω-subdivision slows and make smaller and smaller cuts in the cone..

Define $Z(Q)$ to be the $(n-1)$-simplex which is the section of K by the hyperplane $H_0 = \{x|eQ^{-1}x = 1\}$, where e is the vector of all ones. Define $\sigma(Q)$ to be the eccentricity of K relative to the basic optimal solution of the linear program associated with Q. The following algorithm gives a hybrid of the ω-subdivision process with the bisection process for simplicial branch and bound.

Algorithm 8 : ω-subdivision, Bisection Hybrid Process

initialization: Let Q_0 be the initial n-simplex. Select η_0 and $\rho \in (\frac{\sqrt{3}}{2}, 1)$. Set $\eta(Q) = \eta_0$ for the initial cone $K_0 = \text{con}(Q)$ to be subdivided.

step 1: If $||eQ^{-1}|| \leq \eta(Q)$ and $\omega(Q) \leq \rho$, then perform an ω-subdivision of $K = \text{con}(Q)$.

step 2: Otherwise, perform a bisection of $K = \text{con}(Q)$.

This algorithm has the advantage of using some function information, but it is prevented from jamming like pure ω-subdivision by applying bisection when the region becomes eccentric.

2.4 Result Verification

Result verification methods are computational methods that mathematically guarantee existence of solutions. That is, result verification requires the verification of hypotheses of an existence theorem be carried out on a digital computer. This is generally done using interval arithmetic techniques with outward rounding. This research requires result verification in its partitioning scheme and for this reason, some ideas pertaining to this research and result verification are explored here.

Given a vector valued function $y = F(x)$ mapping $\Re^n \rightarrow \Re^n$ and an interval X, we would like to quickly and easily determine if there is a solution to the equation $F(x) = 0$ in the interval X. This research is interested in solutions to $F(x) = 0$ (where $F(x) = \nabla f(x)$) since these are points at which the function can achieve a global or local optimum. The points for which $\nabla F(y) = 0$ indicate that these points deserve further inspection. Result verification methods would allow one to place a small box around this point y and then mathematically verify that either there exists a point within that box which is a local minimum, some point in the box is a saddle point, or nothing can be verified.

This is important for global optimization since using result verification and a convexity test can either verify or dismiss the possibility of a particular region containing a local optima rather than simply containing a saddle point. Also, it is shown that some of the result verification methods can also improve an approximate solution that was found by a local optimization method.

This research now introduces three operators that verify a solution in a mathematically correct fashion. These operators are the Interval Newton Operator, the Krawczyk operator, and the Hansen-Sengupta operator.

2.4.1 Interval Newton Operator

Let $A^H = \Box\{\tilde{A}^{-1} | \tilde{A} \in A\}$, and define $N(x, \tilde{x}) = \tilde{x} - A^H F(\tilde{x})$.

Theorem 2 Let $F : D_0 \subseteq \Re^n \rightarrow \Re^n$ be Lipschitz continuous on $D \subseteq D_0$ and let $F(\tilde{x}) - F(\tilde{y}) = \tilde{A}(\tilde{x} - \tilde{y})$ for $\tilde{A} \in A$ and $\tilde{y}, \tilde{y} \in D_0$. Then, if $\tilde{x} \in X \in D$, every X' satisfying

$$N(X, \tilde{x}) = \tilde{x} - A^H F(\tilde{x}) \subseteq X'$$

has the following three properties:

(1) Every zero $x^* \in X$ of F satisfies $x^* \in X'$.

(2) If $X' \bigcap X = \emptyset$, then F contains no zero in X.

(3) If $\tilde{x} \in \mathrm{int}(X)$ and $X' \subseteq X$, then F contains a unique zero in X and, hence in X'.

Proof: See [74]. ■

$N(X, \tilde{x})$ is called the Interval Newton operator since, if $A = F'(X)^{-1}$, the operator looks like the Newton update $x_{n+1} = x_n - F'(x_n)^{-1}F(x_n)$. The disadvantage to this method is that A must be regular. That is, $\forall \tilde{A} \in A$, $\tilde{A}^{-1}$ must exist. A closed, convex and bounded set A of $m \times n$ matrices is called a Lipschitz set for F on $D \in D_0$ if $F(\tilde{x}) - F(\tilde{y}) = \tilde{A}(\tilde{x} - \tilde{y})$ for some $\tilde{A} \in A$. A is called a Lipschitz matrix if A is a Lipschitz set and $A \in IR^{n \times n}$. It follows that A must be a regular Lipschitz matrix. This strongly restricts the set of matrices that work for this method for a particular choice of F.

2.4.2 Krawczyk Operator

An improved operator was developed by Krawczyk. Define

$$K(X, \tilde{x}) := \tilde{x} - CF(\tilde{x}) - (CA - I)(X - \tilde{x}),$$

where C is any $n \times n$ real matrix Then the following theorem holds.

Theorem 3 Let $F : D_0 \subseteq \Re^n \to \Re^n$ be Lipschitz continuous on $D \subseteq D_0$, and let $A \in I\Re^{n \times n}$ be an interval Lipschitz matrix for F on D. If $\tilde{x} \in X \in ID$ and $X' = K(X, \tilde{x})$ then:

(1) Every zero $x^* \in X$ of F satisfies $x^* \in X'$.

(2) If $X' \bigcap X = \emptyset$ then F contains no zero in X.

(3) If $\tilde{x} \in \mathrm{int}(X)$ and $\emptyset \neq X' \subseteq X$, then A is strongly regular (that is, $\hat{A}^{-1}A$ is regular), F contains a unique zero in X, and hence in X'.

Proof: See [74]. ■

If, as with the interval Newton operator, we replace C by $\widehat{F(\tilde{x})}^{-1}$ and let $A = F(\tilde{x})$, then this operator can be seen as a Newton step plus a contraction of the interval. A distinct advantage of this method is that A is not required to be regular. If not, then we cannot verify uniqueness, but it is still possible to verify that no solution exists in the interval X. Neumaier [74] goes on to demonstrate that the best possible value for C is $\hat{A}$, that the best possible value of $\tilde{x}$ is $\hat{x}$, and that these values are independent of each other.

2.4.3 Hansen-Sengupta Operator

Define the operator

$$\Gamma(A, B, X) = \begin{cases} B/A \bigcap X & \text{if } 0 \notin a, \\ \square(x \,]\underline{B}/\underline{A}, \underline{B}/\overline{A}[) & \text{if } B > 0 \in A, \\ \square(X \,]\overline{B}/\overline{A}, \overline{B}/\underline{A}[) & \text{if } B < 0 \in A, \\ X & \text{if } 0 \in A, B. \end{cases}$$

Then $\Gamma(A, B, X) = \square\{\tilde{x} \in X | \tilde{x} = \tilde{b} \text{ for some } \tilde{a} \in A,\ \tilde{b} \in B\}$ [74]. Given a matrix A, a vector b, and an approximate solution X^* to the equation $AX = b$, an improved guess $y_1, y_2, \ldots, y_n$ can be obtained by the following process,

$$Y_i = \Gamma\left(A_{ii}, b_i - \sum_{k<i} A_{ik}Y_k - \sum_{k>i} A_{ik}X_k, X_i^*\right), \quad i = 1, \ldots, n.$$

This operator can only be applied to matrices for which $A_{ii} \neq 0$. If at some point, Y_i becomes $\emptyset$, then we assign the value of $\emptyset$ to Y_i, and have verified that no solution to $Ax = b$ exists in the interval X. Note that A, b, and X can all be either real or interval. If they are all real, this operator becomes the standard Gauss-Sidel process for solving systems of linear equations.

For nonlinear result verification, define

$$H(X, \tilde{x}) = \tilde{x} + \Gamma(CA, -CF(\tilde{x}), (X - \tilde{x})).$$

The following theorem then holds for the nonlinear Hansen-Sengupta operator.

Theorem 4 Let $F : D_0 \subseteq \Re^n \to \Re^n$ be Lipschitz continuous on $D \subseteq D_0$, and let $A \in I\Re^{n\times n}$ be an interval Lipschitz matrix for F on D. If $\tilde{x} \in X \in ID$ and $X' = H(X, \tilde{x})$ then:

(1) Every zero $x^* \in X$ of F satisfies $x^* \in X'$.

(2) If $X' \bigcap X = \emptyset$ then F contains no zero in X.

(3) If $\tilde{x} \in \mathrm{int}(X)$ and $\emptyset \neq X' \subseteq X$, then A is strongly regular (that is, $\hat{A}^{-1}A$ is regular), F contains a unique zero in X, and hence in X'.

Proof: See [74]. ∎

Neumaier goes on to show that the Hansen-Sengupta operator provides enclosures for the zero of F that are guaranteed to be at least as good as those of the Krawczyk operator. Also, the Hansen-Sengupta operator gives sharper non-existence and existence tests. However, it can be shown that the existence test with $H(X, \tilde{x})$ is weaker than with the Newton operator $N(X, \tilde{x})$. Also, there is not yet a known result that gives the optimal values for C and $\tilde{x}$ for the Hansen-Sengupta operator.

2.5 Interval Iteration

The methods in the previous section can be seen as a means to verify whether a given box contains a possible local optimum. They can all be used as the basis of a global optimization algorithm. Historically, the earliest of these algorithms used the Newton operator, but any of the operators K or Γ can be used.

Algorithm 9 : Interval Iteration (Newton operator)

Initialization: Put the box X on the list L of boxes, where X is the region over which the global optimum is desired, and set the DONE list to NULL.

step 1: If the L is empty, go to step 3; otherwise, remove a box B from L.

step 2: If $N(B, \tilde{x}) \not\subset B$, where $\tilde{x} \in B$, discard B, and go to step 1.

- **step 2a:** Else if $N(B, \tilde{x})$ is smaller than some pre-determined tolerance, put $K(B)$ on the DONE list. Go to step 1.
- **step 2b:** Else if $N(B, \tilde{x})$ is sufficiently smaller than B, put $N(B, \tilde{x})$ on L. Go to step 1.
- **step 2c:** Else, split $N(B, \tilde{x}) \bigcap B$ into k pieces, $B_1 \ldots B_k$ and put them on L. Go to step 1.

step 3: Output the DONE list.

A potential problem of this algorithm is the dependence on the matrix Y. The algorithm works best if Y is close to the inverse of the midpoint of the Hessian matrix and computing that inverse requires $O(n^3)$ operations. The approximate inverse can be used from step to step, but a good approximation to the inverse of the midpoint of the Hessian is important to the speed of convergence. When the second derivative is continuous on the box X, the third derivative is non-zero on X, and f has a fixed point in X, then the above algorithm converges quadratically in the limit [69]. Like a quasi-Newton's method for real numbers, this quadratic convergence rate is dependent on having a good approximation to the inverse of the midpoint of the Hessian.

All boxes on the DONE list consist of all the points of the original box for which it is possible that $\nabla F(x) = 0$. To complete the global optimization algorithm, each of these boxes must now be checked to determine which boxes contain only local optima and which boxes contain the global optima. To do this, let L be the DONE list and let

$$y^* = \min_{B_i \in L} \overline{F(B_i)}$$

and discard all B_i for which $\underline{F(B_i)} \geq y^*$. The global optima must be in the remaining boxes.

2.5.1 Hansen's Algorithm

Hansen has implemented an interval branch and bound algorithm which includes several different techniques to eliminate portions of a box. This research touches on the highlights of the particular methods used to eliminate part or all of a box as presented in [35]. His algorithm is guaranteed to find the global maximum of all local maximums, and he assumes that the global maximum occurs in the interior of the initial box and not on the boundary.

The most obvious method of eliminating a box is to evaluate the function at some point in a region. If that function value is better than any known function value, record this function value as the best known function value y^*. If the lowest function value of the function over a box is greater than y^*, that box can be eliminated. Similarly, if $0 \notin \nabla F(X)_i$, we know that it is impossible for there to be a stationary point of F in the box X. Thus X can be deleted. Finally, if $\nabla^2 F(X)_{i,i} < 0$, it must be true that $\nabla^2 F(x)_{i,i} < 0$ for any real vector $x \in X$. This implies that the Hessian matrix cannot be positive semi-definite (and therefore, the function cannot be convex) on the box X. Again the box X can be deleted. Enclosures of both the gradient and the hessian can be computed using interval arithmetic. Just as with evaluating the function itself, the enclosures for the gradient and the Hessian are guaranteed.

Hansen also implements some methods that are able to eliminate just part of a box. These methods require more work, but can result in a savings of time as large regions can be eliminated. It can be shown using a Taylor's expansion of the function F about a point $x = \hat{X}$ that for $y \in X$,

$$F(y) \in F(x) + \sum_{i=1}^{n}(y_i - x_i)g_i(X_1, \ldots, X_i, x_{i+1}, \ldots, x_n),$$

where g_i is the i^{th} component of the gradient of F. We denote $g_i^I = g_i(X_1, \ldots, X_i, x_{i+1}, \ldots, x_n)$ and rewrite equation(2.5.1) for some $j = 1, \ldots, n$, as

$$F(y) \in F(x) + ((y_j - x_j)g_j^I + \sum_{\substack{i=1 \\ i \neq j}}^{n}(y_i - x_i)g_i^I.$$

This equation is true for all $y_i \in X_i$. We can replace y_i by X_i in the right hand side and get

$$F(y) \in F(x) + (y_j - x_j)g_j^I + \sum_{\substack{i=1 \\ i \neq j}}^{n}(X_i - x_i)g_i^I. \tag{2.4}$$

Hence, $F(y) > \overline{F}$, where $\overline{F}$ is the best function value so far.

Define $t = y_j - x_j$,

$$U = F(x) + \sum_{\substack{i=1 \\ i \neq j}}^{n} (X_i - x_i) g_i^I - \overline{F}$$

and

$$V = g_j^I.$$

We can then delete the portions of X for which $U + Vt > 0$ and retain the portions for which $U + Vt \leq 0$. Let $T = \{t : U + Vt \leq 0\}$, $U = [a, b]$, and $V = [c, d]$. Then T is given by the following:

$$T = \begin{cases} [-a/d, \infty] & \text{if } a \leq 0 \text{ and } d > 0 \\ [-a/c, \infty] & \text{if } a > 0,\ c < 0, \text{ and } d \leq 0 \\ [-\infty, \infty] & \text{if } a \leq 0 \text{ and } c \leq 0 \leq d \\ [-\infty, -a/d] \bigcup [-a/c, \infty] & \text{if } a > 0 \text{ and } c < 0 < d \\ [-\infty, -a/c] & \text{if } a \leq 0 \text{ and } c > 0 \\ [-\infty, -a/d] & \text{if } a > 0,\ c \geq 0 \text{ and } d > 0 \\ \emptyset & \text{otherwise.} \end{cases}$$

Hansen sets $Y_j = X_j \bigcap (T + x_y)$. If this set is empty, all of X is deleted. Otherwise, the algorithm sets $X_j = Y_j$ and repeats this operation for $j = 1, \ldots, n$. It is also possible for Y_j to be made up of two intervals. If this is the case, the algorithm sets $X_j = \square Y_j$ but retains the "gap" information if the box is split. In this manner, it saves the problem of dealing with handling the branching that would occur if we split the box. It also allows the algorithm to discard the gap if the box is split at a later point in time.

Similarly, Hansen uses the second order Taylor's expansion to eliminate portions of the box X. If we let $x = \hat{X}$ and $z = y - x$, then we can delete the points y for which

$$f(x) + z^T g(x) + \frac{z^t \nabla^{2j}(x, X) z}{2} > \overline{F}$$

where $\nabla^{2j}(x, X)$ is the Hessian with arguments $(X_1, \ldots, X_j, x_{j+1}, \ldots, x_n)$. For example, when $n = 2$, we then want to retain the points y such that

$$f(x) + z_1 g_1 + z_2 g_2 + \frac{z_1^2 \nabla_{11}^2 + 2 z_1 z_2 \nabla_{21}^2 + z_2^2 \nabla_{22}^2}{2} \leq \overline{f}.$$

Hansen shows that $\nabla^2(x, X)$ is lower triangular(see [35], section 6.4) and so the above expression simplifies to

$$f(x) + z_1 g_1 + z_2 g_2 + \frac{z_1^2 \nabla_{11}^2 + z_1 z_2 \nabla_{21}^2 + z_2^2 \nabla_{22}^2}{2} \leq \overline{f}.$$

After simplifying the above and replacing z_2 by $Z_2 = X_2 - x_2$, one wants to solve the quadratic equation

$$A + B z_1 + C z_1^2 \leq 0, \tag{2.5}$$

where

$$A = f(x) - \overline{f} + Z_2 g_2 + \nabla_{22}^2 Z_2^2,$$
$$B = \frac{X_2 \nabla_{21}^2}{2},$$

and

$$C = \frac{\nabla_{11}^2}{2}.$$

The solution of 2.5 then becomes a matter of solving the equation $A + BT + CT^2 = 0$ where A, B, C, T are all intervals. The solution to this equation is quite complicated, is not presented here but can be found in [35]. The above equation is then also solved for the case in which one replaces z_1 by $Z_1 = X_1 - x_1$, and then solves for z_2.

Hansen applies the zeroth order check first, immediately followed by the test for monotonicity and the test for non-convexity. He then applies the first order elimination which is followed by the second order elimination. One pass of an interval Newton's method (presented in section 2.5) is also applied. If the box is reduced at all, then the function is re-evaluated at (approximately) its center, and $\overline{F}$ is updated. Hansen also implements a Newton-like search whenever a box yields a new upper bound on F is found, but this search is terminated if it leaves the box.

Hansen claims to have had a great deal of success with this combination of techniques. He states that run time appears to increase roughly with the cube of the dimension of the problem. Since a specially modified compiler was used by Hansen, this research was unable to compile Hansen's FORTRAN code, so direct comparisons on a single machine were not possible.

3. Back-Boxing Methods For Global Optimization

The algorithms for global optimization presented in the previous chapter work well, but often take a great deal of time for large numbers of variables or when high accuracy is required. We present a new method that can solve problems with a large number of variables to a high degree of accuracy in a short amount of time, even when the available memory for the automaton is limited.

Back-Boxing is a new partitioning scheme for global optimization with interval arithmetic and rectangular partitions. It takes advantage of the fact that local optimization is quick and easy and attempts to use ideas of both convexity and result verification to reduce the work expended when close to an optimum. As with global optimization algorithms, this scheme can be foiled and it can actually take significantly longer on some problems. When one desires a high degree of accuracy, a Back-Boxing partitioning scheme performs significantly better.

The algorithm that is developed here finds a local minimum for a given function subject to bound contraints on the variables. As before, we will continue to use capital letters to represent interval variables and non-capitols for point variables. First, consider the following local optimization algorithm of the function F where $\phi(\alpha) = F(x + \alpha p)$:

Algorithm 10 : Damped Newton's Method for Real Local Optimization

step 1: Solve $\nabla_F^2(x)p = -\nabla F(x)$ for p.

step 2: Find α so that the following are satisfied:

a) $\phi(\alpha) < \phi(0) + \epsilon\phi'(0)\alpha$.

b) $\phi(\alpha) > \phi(0) + (1-\epsilon)\phi'(0)\alpha$.

step 3: Set $x \leftarrow x + \alpha p$.

step 4: If $\|\nabla F(x)\| < \epsilon$, stop; else, go to step 1.

This is a basic damped Newton's method for local optimization. Given enough time and a function that is not too poorly behaved, this algorithm converges to a point x^* where $\nabla F(x^*) = 0$, which may or may not be a local optima. If we want the algorithm to converge to a point inside a box, this algorithm could easily fail. It is possible that the method would attempt to go outside the box when it applies the line search in step 2. To prevent this, consider the following modification to the algorithm to find the local optimum of a function over a box B:

Algorithm 11 : Constrained Damped Newton's Method for Real Local Optimization

step 1: Solve $H_F(x)p = -\nabla F(x)$

step 2: For $1 \leq i \leq n$ if $x_i = \overline{B_i}$ and $p_i > 0$, set $p_i = 0$.

step 3: For $1 \leq i \leq n$ if $x_i = \underline{B_i}$ and $p_i < 0$, set $p_i = 0$.

step 4: If $x + p \notin B$, find α such that $x + \alpha p \in \partial B$ and set $p \leftarrow \alpha p$.

step 5: Find α so that the following are satisfied:.

a) $\phi(\alpha) < \phi(0) + \epsilon\phi'(0)\alpha$.

b) $\phi(\alpha) > \phi(0) + (1-\epsilon)\phi'(0)\alpha$.

step 6: Set $x \leftarrow x + \alpha p$.

step 7: If $\|\nabla F(x)\| < \epsilon$, stop; else, go to step 1.

Steps 2 and 3 check to determine if x is on the side of the box. If it is, step 4 makes sure that the vector p does not take x outside the box. That is, if x_i is on the right side of the box and if $p_i > 0$, any value of α would place $x + \alpha p$ outside the box, so p_i is set to 0. Similarly, if x_i is on the left side of the box and $p_i < 0$, then any value of α would place $x + \alpha p$ outside of the box.

Step 4 then checks to see if the $x + p$ would remain inside the box. If not, p is reduced to a length that would, at most, allow $x + \alpha p$ be on the side of the box where $0 \leq \alpha \leq 1$. Finding that α is accomplished by the following theorem.

Theorem 5 Let $X \in I^n$, X not thin, $x^* \in X$, $x^* \in \Re^n$, and let a unit direction p be given. Then the line starting at x^* with direction p leaves X at the point $x^* + \alpha p$ where α is given by

$$\alpha = \min_{1 \leq i \leq n} \alpha_i(X, x^*, p),$$

where

$$\alpha_i(X, x^*, p) = \begin{cases} \frac{\overline{X}_i - x_i^*}{p_i} & \text{if } p_i > 0 \\ \frac{\underline{X}_i - x_i^*}{p_i} & \text{if } p_i < 0 \\ \infty & \text{if } p_i = 0. \end{cases}$$

Proof:

Finding the largest α is equivalent to solving the LP

$$\text{Max} \quad \alpha$$

st:

$$\begin{aligned} x_i^* + \alpha p_i &\leq \overline{X}_i \quad i = 1, \ldots, n \\ x_i^* + \alpha p_i &\geq \underline{X}_i \quad i = 1, \ldots, n \\ \alpha &\geq 0. \end{aligned}$$

Since $x^* \in X$, $\alpha = 0$ is a feasible point of this LP. Therefore, the largest α for which $x^* + \alpha p$ is inside or on the box $\overline{X}$ is given by $\alpha^* = \min_{i, p_i \neq 0} \alpha_i(X, x^*, p)$, providing the desired result. ■

On a digital computer, directed roundings must be used so that it is guaranteed that the point $x + \alpha p$ is in the box. For example, suppose $X = [0, 1]$, $x^* = 1/3$, and $p = 1$. Then $\alpha^* = 2/3$. Suppose also that $1/3$ is represented on our automaton as 0.334, and $2/3$ is represented as 0.667. Then, $x + \alpha p = 1.001$, which lies outside of X. While this example is indeed contrived, it is entirely possible for roundings to occur that force $x + \alpha p$ outside of X.

3.1 Back-Boxing

Some branch and bound global optimization algorithms are based on interval arithmetic (see [35]). While interval arithmetic is mathematically correct, it has an inherent trend to sometimes explosively overestimate the range of a function over a box. When the box width tends to zero, interval arithmetic guarantees that the width of the overestimation also tends to zero but only linearly in the box width. From this we obtain the convergence proof.

However, even though the overestimation tends to zero, this is only a theoretical result and often can cause a great deal of computer time to be expended for the boxes immediately surrounding a global minima. Therefore, it may be wise to identify the areas where minima are located. The algorithm slows on these areas and a fast local search procedure could applied instead. The following theorems are required to help identify these "basins of attraction" and are taken from [8].

Theorem 6 If Gaussian elimination can be performed on $A \in \Re^{n \times n}$ without pivoting, and all the diagonal elements of the resulting matrix B are positive, then A is positive definite.

Theorem 7 If $A \in \Re^{n \times n}$ is symmetric, has positive diagonal elements, and is diagonally dominant, then A is positive definite.

From these standard theorems for real linear algebra, one can quickly derive corollaries for interval linear algebra. Define the following:

$$\begin{aligned} |X| &= \max(|\underline{X}|, |\overline{X}|) \\ \langle X \rangle &= \begin{cases} \min(|\underline{X}|, |\overline{X}|) & \text{if } 0 \notin X \\ 0 & \text{otherwise.} \end{cases} \end{aligned}$$

Define an interval matrix A to be diagonally dominant if

$$\langle A_{ii} \rangle > \sum_{\substack{i=1 \\ i \neq j}}^{n} |A_{ij}|.$$

Corollary 1 If $A \in I^{n \times n}$ is symmetric, has positive diagonal elements, and is diagonally dominant, then every symmetric sub-matrix $A' \in A$ is positive definite.

Proof: Let $A' \in \Re^{n \times n}, A' \in A$, and $(A')^T = A'$. Since A has positive diagonal elements, A' has positive diagonal elements. Since A is diagonally dominant,

$$A'_{ii} \geq \underline{A_{ii}} \geq \langle A_{ii} \rangle > \sum_{\substack{i=1 \\ i \neq j}}^{n} |A_{ij}| \geq \sum_{\substack{i=1 \\ i \neq j}}^{n} |A'_{ij}|.$$

The first inequality is possible since we know that $A_{ii} > 0$. Therefore, A' is symmetric, diagonally dominant, and has a positive diagonal. Applying Theorem 7, A' is positive definite. ∎

Corollary 2 If $A \in I^{n\times n}$ can be Gaussian eliminated without pivoting, and the resulting matrix $B \in I^{n\times n}$ has a positive diagonal, then every symmetric sub-matrix $A' \in A$ is positive definite.

Proof: Let $B = AC$, where $C \in \Re^{n\times n}$, and C is the product of elementary matrices, none of which correspond to row exchanges. Furthermore, let $B_{ii} > 0, \;\; 1 \leq i \leq n$. Let $A' \in A$ be symmetric. Then $B' = A'C \in AC = B$ has a positive diagonal and is obtained from A' by Gaussian elimination without row exchanges and, by Theorem 6, is positive definite. ■

One interesting point in the context of the unconstrained optimization problem is that even though the Hessian should be symmetric if calculated using exact real arithmetic, the interval Hessian often is non-symmetric due to round-offs occurring in different places and in different orders. The algorithm that is developed here forces symmetry on the interval Hessian by taking $A_{i,j} = A_{i,j} \bigcap A_{j,i}$ which sometimes results in tighter values for the Hessian.

The reason that forcing symmetry is a valid procedure is that the Hessian of every twice continuously differentiable function is symmetric. Let A be a matrix such that $A = \nabla^2 F(X)$ and $A_{ij} \neq A_{ji}$ for some i, j. Without loss of generality, suppose that there exists $a \in A_{ij}$, $a \notin A_{ji}$. Then any matrix $A' \in A$, $A' \in \Re^{n\times n}$ with $A'_{i,j} = a$ cannot be symmetric and so could not be the Hessian of a twice continuously differentiable function. Therefore, we can eliminate any such matrices by applying the above theorems to $A \bigcap A^T$ rather than just the matrix A. We take the intersection rather than the union since the intersection gives a tighter bound on A.

Back-Boxing is defined to be the process of identifying a box that surrounds a given point in $\Re^n$ such that the objective function is convex on that box. This may not always be possible. For example, consider the function $f(x) = x^3$ at the point $x = 0$. For any interval that contains $x = 0$ as an interior point, f is not convex on that interval. For a case such as this one, a small box is placed around the point and it is set aside for later processing. The computational method developed here finds a box where F has a unique global optimum on that box and verifies mathematically that the minimum is unique using result verification methods unlike [56]. In [56], a stationary point is found, and then a "prohibited area" is placed around that point. Next, it is assumed that this stationary point is the global minimum of the prohibited area, and this portion of the domain is no longer considered. This has the advantage that one does not need to verify that the global minimum of the prohibited area occurs at the stationary point, but this is also its downfall. Since the region is not verified, it is possible that it could contain not only other local minima, but also a smaller local minima.

Recall that the majority of the work expended by a branch and bound algorithm is usually expended searching close to the local optima. This has been verified by several different people including Hansen, Walster, and Sengupta in [107]. They state that for most problems they tested, the time required to globally optimize did not go up as an exponential function of the size of the domain. This was also verified by Hansen [35] as well as in some private communication between Hansen and Moore. If the work required for a branch and bound algorithm was evenly distributed without regard to the existence or location of a local optima, then one would expect exponential growth in the time when the size of the domain increases. Since this does not happen, most of the work must be expended when close to the optima.

Another way to see this is to consider what happens to the interval extension of F, ∇F, and $\nabla^2 F$ when X is close to or contains a global minimum. If X contains a local minimum, and we then bisect X into two pieces, both pieces could now contain the global minimum and, neither can be deleted. Similarly, if a box G of width macheps is placed around the global minimum of X, and each of the boxes $X - G$ is evaluated with an interval extension, the evaluation (due to round off error) does not allow any of the boxes in $X - G$ to be eliminated. For example, consider the function $f(x) = x^2$ on the interval $X = [-10,\ 10]$. If we split X into the three boxes [-10, ϵ], [-ϵ, ϵ], [ϵ, 10] where ϵ is the smallest machine representable number, only one box contains the global minimum. However, $F([\epsilon, 10]) = [\epsilon^2, 100]$ and on the automaton, $]\epsilon^2, 100[= [0, 100]$ cannot be deleted. This may be a trivial example, but round-offs must be taken into consideration. Many more boxes that are close to the solution must be treated when tolerance is small. Similarly, inevitable overestimation in the evaluation of the gradient causes the enclosures to contains zero, even when there is no local minimum in that box.

It is relatively easy (when compared to global optimizing) to find the global optima of a convex function over a convex set. This can be done to a Back-Boxed region. Naturally, one wants to identify the largest box around a point such that the objective function is convex. This turns out to be a non-trivial problem and is discussed later. Since one does not want to spend time Back-Boxing regions that are not promising, it would be wise to first attempt to locate a local optima and then Back-Box that local optima. In this way, rather than spend work attempting to eliminate regions close to the global optima, this "difficult" region is treated using a real arithmetic algorithm which is inexpensive in terms of CPU time and also is fast. The following algorithm incorporates these ideas.

The Back-Boxing algorithm maintains three lists of boxes. The first list is the list of undetermined boxes which contains all boxes for which it has not been determined whether they contain a global minimum. The second list is the set of finished boxes. These are boxes that have been reduced to a size smaller than a preset tolerance. The global optima of the function may be contained within these boxes. Finally, there is the list of CX boxes. This is a list of boxes for which it has been determined that the objective function F is convex at each point of the box. For each of these boxes, the minimum over the box is verified, and the remainder of the box is deleted at the end of the algorithm.

The algorithm contains two loops. In the outer loop the algorithm checks to see whether any excessively large boxes remain on either the unchecked list or the CX list. If there are any boxes that are too large on either of these lists, the inner loop processes these boxes. These boxes are reduced in size or split either by Back-Boxing or by bisection. This is continued until the unchecked list is empty. The algorithm then returns to the outer loop. If any boxes B on the CX list are too large, a box B^* of width ϵ of the width of the box is located in the center. Then the boxes

B^* and $B - B^*$ are put on the unchecked list, and the algorithm continues. The box is located in the center because that is where the proposed minimum of the box was found.

At termination, the algorithm returns all the boxes that are on the CX list and all the boxes on the finished list. All the global minima must be contained within these boxes.

Algorithm 12 : Back-Boxing

Initialization: Set list of undetermined boxes to B_0, where B_0 is the set over which one wants the set of global minima. Assign the set of CX boxes NULL, the list of finished boxes NULL, and $i = 0$.

step 1: Choose a box B from the list of undetermined boxes. If the list of undetermined boxes is empty, go to step 4.

step 2: Attempt to eliminate all or part of the box using some of the techniques in section 2.3.3.

step 3: If either B is not large enough for Back-Boxing or if $(F(\hat{B}) - F^*)/F^* >$ BBTol, where F^* is the smallest upper bound of the function values on the region, split B into k pieces $B^1, \ldots, b^k$. Place the boxes for which $\max_i \text{width}(B^i) <$ pre-specified tolerance on the list of finished boxes and place the remainder on the unchecked list and go to step 1. Otherwise, continue.

Overview: In steps 3a through 3d, the algorithm attempts to find the largest box B' for which it is mathematically guaranteed that B' has a unique global optimum. This process uses the intermediate boxes B_g, B_h, and B_{hs} to complete its work.

step 3a: Apply a point algorithm to minimize f over B to find a point x that (nearly) satisfies the Kuhn-Tucker conditions. That is, find a point x' such that $\nabla F - \mu g \leq \epsilon$, where g represents the constraints $x_i \leq \overline{B}$ and $x_i \geq \underline{B}$, and ϵ is some specified tolerance.

step 3b: While $0 \notin \nabla f(B')$), enlarge B' to B_g. Then while $\nabla^2 f(B_g)$ is positive definite, enlarge B_g to B_h. Finally, while $H(X, \tilde{x}) \in X$, enlarge B_h to B_{hs} The strategy to enlarge B' is given below in Algorithm 13.

step 3c: If B_{hs} is larger than B_h or if B_h is sufficiently larger than B_g, set $B' = B_{hs}$ and place B_h on the list of CX boxes. Else set $B' = B_g$ and discard B_g.

step 3d: Let $B - B' = B^1, \ldots, B^j$. Place the boxes for which $\max_i \text{width}(B^i) <$ pre-specified tolerance on the list of finished boxes and place the remainder on the unchecked list and go to step 1. A discussion of how to do this in a mathematically verifiable way on a digital computer is given below.

step 4: All boxes have now been classified as either being too small to be processed further, or it has been determined that the function F is convex on that box. The algorithm now attempts to eliminate or reduce the size of the boxes that remain. This is the end of the inner loop as mentioned above. Remove a box B from the list of CX boxes. If the list of CX boxes is empty, go to step 9.

step 5: Attempt to eliminate part or all of B using some of the techniques in section 2.3.3.

step 6: If B is now empty, go to step 4. Otherwise apply a point descent

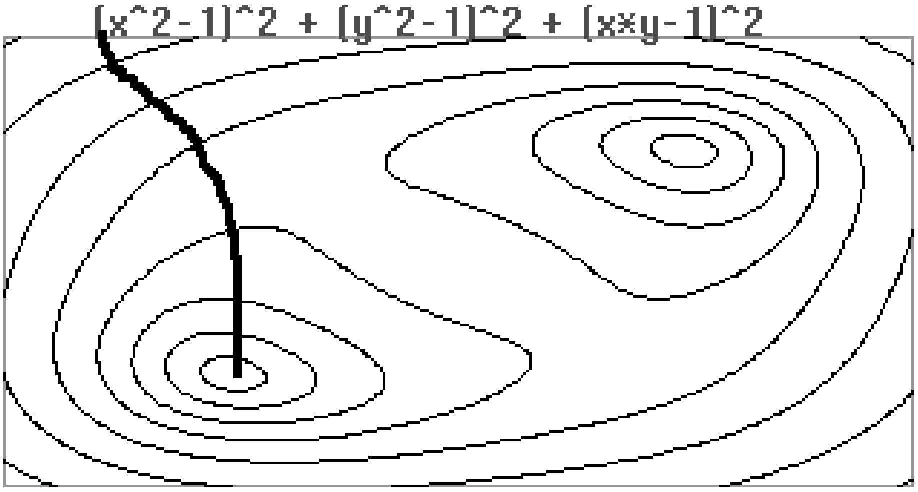

Figure 3.1: First locate a local minimum

method which is guaranteed in theory (given enough time and precision) to converge to the global minimum of a convex function on a convex set to what remains of the box. This returns a point $x^* \in B$, $x^* \in \Re^n$.

step 7: Let $Y = x^* + [-\epsilon_v, \epsilon_v]$ for some $\epsilon_v > 0$, and verify that Y contains a local minimum over the box B. If not, go to step 6 with x^* as the starting point for the minimizer.

step 8: Let $Y = x^* + [\epsilon_v, \epsilon_v]$. If $\underline{F(Y)} >$ best function value so far, discard B. Else, discard $B - Y$ and contract the box Y using one of the result verification routines in section 2.4 until it is smaller than the predescribed tolerance. Put Y on the list of finished boxes and go to step 4.

step 9: Discard all boxes on the finished list for which $\underline{F}(B_i) > F^*$.

step 10: Return the list of finished boxes.

This research desired an algorithm that worked as a black box

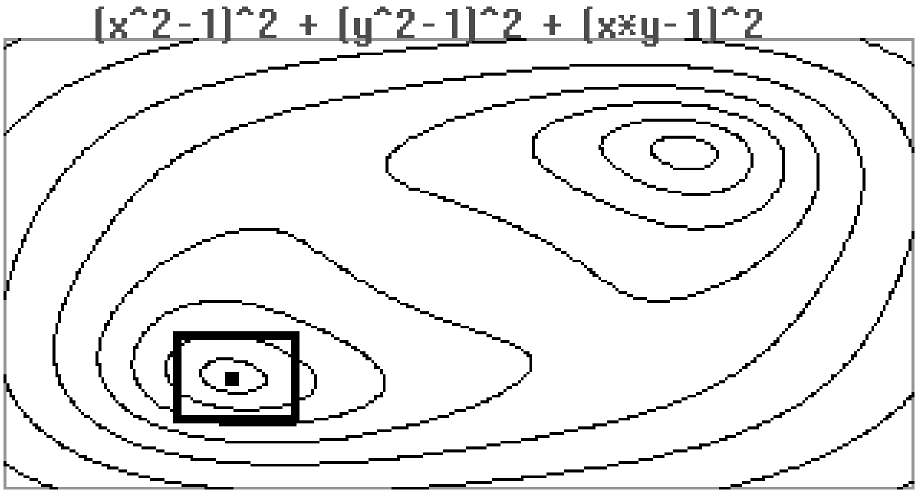

Figure 3.2: Place a box around that point such that f is convex

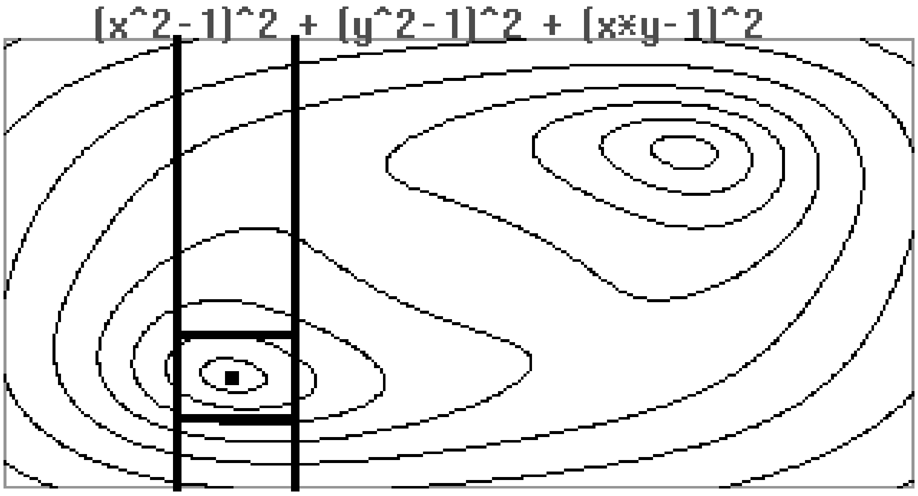

Figure 3.3: Partition the region around that box

requiring no information about the function other than its definition. This algorithm satisfies those goals. In fact (assuming the function is differentiable), derivatives are computed using automatic differentiation so one does not need to provide an algebraic definition for the derivative. As output, it gives a set of boxes for which are mathematically guaranteed to contain the global optima when this algorithm is implemented on a digital computer with directed roundings (available on any IEEE machine) for interval arithmetic and guaranteed function evaluations.

Step 4b of the Back-Boxing algorithm is accomplished in the following manner.

Algorithm 13 : Grow Box

Initialization: Let $c = \text{width}(B)/2$, $a = 0$, $b = (a + c/2)$, and ONE to be the vector of all ones. Finally, let x be the point about which one desires a box of maximal size for which F has a unique global optimum on that box.

step 1: $B_g = B \bigcap (x + b[-1, 1]ONE)$.

step 2: If $0 \notin \nabla f(B_g)$, $a = b$, else $c = b$.

step 3: If $(c - a) < \text{tol}$, go to step 4. Otherwise, go to step 1.

step 4: $B_g = B \bigcap (x + a[-1, 1]ONE)$, $c = \text{width}(B)/2$, $b = (a + c/2)$.

step 5: $B_h = B \bigcap (x + b[-1, 1]ONE)$.

step 6: If $\nabla^2 f(B_h)$ is positive semi-definite, $a = b$, else $c = b$.

step 7: If $(c - a) < \text{tol}$, go to step 8. Otherwise, go to step 4.

step 8: $B_h = B \bigcap (x + a[-1, 1]ONE)$.

step 9: $B_{hs} = B \bigcap (x + b[-1, 1]ONE)$.

step 10: If $H(B_{hs}, \tilde{x}) \in X$, $a = b$, else $c = b$.

step 11: If $(c - a) < \text{tol}$, go to step 12. Otherwise, go to step 8.

step 12: $B_{hs} = B \bigcap (x + a[-1, 1]ONE)$.

The Grow Box algorithm is a combination of three one dimensional bisection algorithms. The first identifies the largest value a for which $\nabla F(B_g = \nabla F(B \bigcap (x + a[-1, 1]ONE))$ does not contain 0. The second uses B_g as a starting point and then determines the largest box for which $\nabla^2 F(B_h) = \nabla^2 F(B \bigcap (x + a[-1, 1]ONE))$ is positive definite. The final bisection algorithm finds the largest box for which $H(X, \tilde{x}) \in X$. Since the cost of computing the gradient is an order of magnitude less than computing the Hessian [55], the box is first enlarged using the gradient rather than the Hessian or the Hansen-Sengupta operators.

Jansson and Knüpel proposed and implemented a similar method in [56]. They use a local minimizer to find a proposed local minimum and then place a "prohibited area" around this local minimum which would no longer be searched. They do not guarantee the existence or lack of a local or global minima in this prohibited area. Hence, other local or global minima could escape detection. They do compute bounds on the function value over this prohibited area and discard the prohibited area if the minimum function value exceeds the best function value discovered for the function on the region.

Back-Boxing is also similar in some respects to MLSL. In MLSL, a local minimizer is applied to a proposed "basin of attraction" exactly once. That region is then ignored since the best function value for that region has been found. There is always the potential problem that there might be two or more local minima in a proposed basin of attraction and that a global (or smaller local) minimum could be missed by MLSL.

Back-Boxing is guaranteed to not miss any local or global minima of the function since any region that is identified as convex has its local minimum verified. If the local minimum cannot be quickly verified, then the box is split and treated again until it is either discarded or the global minimum of the box is found. Regions that are not identified as having a convex objective function are eliminated only if they do not have the necessary characteristics of a global minimum. Some of these conditions are the gradient containing 0, having a smaller function value than other regions, having a positive definite Hessian, and others given in section 2.3.3.

We know that a real matrix is positive definite if it is symmetric, its diagonal is greater than zero, and it is diagonally dominant. The test for diagonal dominance of every symmetric sub-matrix becomes $A_{i,i} < \sum_{\substack{j=1 \\ j \neq i}}^{n} \min\left(|\overline{A_{i,j}}|, |\underline{A_{i,j}}|\right)$. Similarly, the test for the diagonal being greater than 0 is simply $\underline{A_{i,i}} > 0$. If both of these conditions are satisfied, then the every real sub-matrix A' of A is guaranteed to be positive definite.

Step 4d of Algorithm 12 can be accomplished in any number of ways. If we assume that B' is strictly interior to B, we can let

$$B - B' = \left\{X' | B' \neq X', X'_i \in \{[\underline{B_i}, \underline{B'_i}], [\underline{B'_i}, \overline{B'_i}], [\overline{B'_i}, \overline{B_i}]\}\right\}$$

There are three choices for each X_i, $i = 1, \ldots, N$ which yields 3^N possible boxes. 1 must be subtracted for the box B where X' =B which gives $3^N - 1$ possible boxes for $B - B'$ (see figure 3.4) and this may not be the best way to calculate the difference. The hope from steps 4a-d is that the box B has only a few local optima, and that each of these

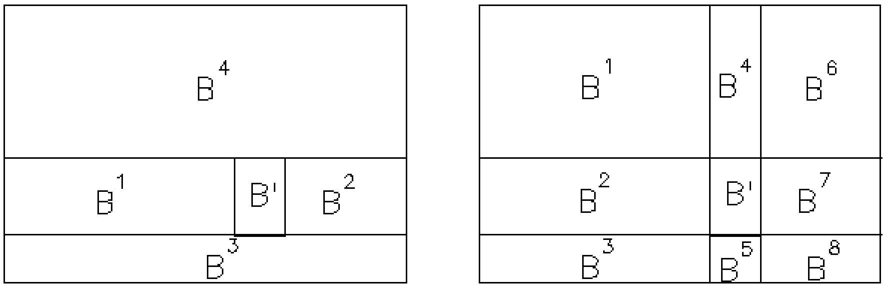

Figure 3.4. Partitions for $B - B'$. The first is from Algorithm 14 and the second is the memory intensive method.

can be captured in its own box, while the remainder of the box can be easily eliminated using any of the standard branch and bound techniques. Adding an exponential number of boxes to the list is not a good choice in this case as it requires an exponential number of boxes to be eliminated. For example, if $N = 100$ (not a large number of variables), 5.154x10^{47} boxes are added to the list. Considering that each box requires storing $2N$ double numbers to represent the upper and lower bounds of the box in each dimension and each double requires ten bytes, this requires in excess of 1.031x10^{45} megabytes of memory which exceeds the storage capacity of any known computer today (actually more than the sum of the storage capacity of all computers ever built).

Since this is unreasonable for a large number of variables, we compute $B - B'$ in the following fashion:

Algorithm 14 : Minus

Initialize: Set MinusList = NULL, $i = 0$.

step 1: $X = B$.

step 2: $X_i = [\underline{B_i}, \underline{B'_i}]$.

step 3: Add X to MinusList.

step 4: $X_i = [\overline{B'_i}, \overline{B_i}]$.

step 5: Add X to MinusList.

step 6: $X_i = [\underline{B'_i}, \overline{B'_i}]$.

step 7: If $i < N$, increment i, and go to step 2. Else return MinusList.

This algorithm creates boxes by cutting out portions of the box in each dimension. For example, if

$$B = \begin{bmatrix} [-10, 10] \\ [-5, 30] \end{bmatrix}$$

and

$$B' = \begin{bmatrix} [-1, 1] \\ [2, 3] \end{bmatrix},$$

the following boxes would be added to the list by the above method in the following order:

$$\begin{bmatrix} [-10, -1] \\ [-5, 30] \end{bmatrix}, \begin{bmatrix} [1, 10] \\ [-5, 30] \end{bmatrix}, \begin{bmatrix} [-1, 1] \\ [-5, 2] \end{bmatrix}, \begin{bmatrix} [-1, 1] \\ [3, 30] \end{bmatrix}$$

This method returns a total of $2N$ boxes of considerably different volumes. Moreover, the amount of memory required for this method is much lower. For example, for 10,000 variables and 10 bytes per double, only 200K of memory is required for these boxes. This is easily handled on today's computers. It is hoped that only a few of the boxes require significant time to process, and that many may be eliminated outright. Jansson and Knüpel use a branch and bound algorithm like Algorithm 12 and compute what they call a "prohibited area". They do not describe how they search the area that is not prohibited, nor do they describe how they divide up the region $B - B'$.

Algorithm 12 has similarities to many global optimization algorithms reviewed here. If the minimum size required for Back-Boxing is set to 0, then the algorithm degenerates into a standard branch and bound algorithm such as Hansen's method. If not, then at steps 4c and 4d of the algorithm, the procedure attempts to find as large a box as is possible that is still either convex or does not have 0 in the gradient. B' can either easily be discarded or a global maximum over the box B' can be quickly and easily found using a local optimization method and result verification.

4. Test Problems

This section contains the problem set that was compiled from problems found in the literature or specially created to test the Back-Boxing algorithm that was developed in this paper. Several problems were either created or found to test the Back-Boxing algorithm that was developed in this paper. The solutions (if known) are given. The majority of the test problems can be found in [39, 56, 88] but the original reference is provided when known.

4.1 Published Problem Sets

Test Problem 1 Extended Rosenbrock's function [87]

$$\begin{aligned} F(x) &= \sum_{i=1}^{n/2} 100(x_{2i} - x_{2i-1}^2)^2 + (1 - x_{2i-1})^2 \\ \text{s.t.} \quad & -10 \le x_i \le 10 \end{aligned}$$

$x^* = (1, 1, 1, \ldots, 1)^T$, and $f(x^*) = 0$.

Test Problem 2 Extended Rosenbrock's function, version 2 [88]

$$\begin{aligned} F(x) &= \sum_{i=1}^{n} 100(x_i - x_{i-1}^2)^2 + (1 - x_{i-1})^2 \\ \text{s.t.} \quad & -10 \le x_i \le 10 \end{aligned}$$

$x^* = (1, 1, 1, \ldots, 1)^T$, $f(x^*) = 0$.

Schittkowski [88] provides many different variations of the standard Rosenbrock's function, of which the above two examples are given. The original reference by Rosenbrock [87] is the first of the above two with $n = 2$. Rosenbrock's function is a classic problem in local optimization since it has a very narrow and very steep banana-shaped valley that surrounds the global minimum.

Test Problem 3 Six Hump Camel Back [56, 97]

$$
\begin{aligned}
F(x) &= 4x_1^2 - 2.1x_1^4 + x_1^6/3 + x_1 x * 2 - 4x_2^2 + 4x_2^4 \\
\text{s.t.} &\quad -5 \le x_i \le 5
\end{aligned}
$$

$$
x^* = \begin{cases} (0.0898400, -0.712659) \\ (-0.0898400, 0.712659) \end{cases}, \ f(x^*) = \text{-1.0316285.}
$$

Test Problem 4 Levy 3 [56, 107]

$$
\begin{aligned}
F(x) &= \sum_{i=1}^{5} i \cos((i+1)x_1 + i) \sum_{j=1}^{5} j \cos((j+1)x_2 + j) \\
\text{s.t.} &\quad -10 \le x_i \le 10
\end{aligned}
$$

$$
x^* = \begin{cases} (4.97648, 4.85806) \\ (4.97648, -1.42513) \\ (4.97648, -7.70831) \\ (-1.30671, 4.85806) \\ (-1.30671, -1.42513) \\ (-1.30671, -7.70831) \\ (-7.58989, 4.85806) \\ (-7.58989, -1.42513) \\ (-7.58989, -7.70831) \end{cases}, \ f(x^*) = \text{-176.542.}
$$

Test Problem 5 Levy 5 [56, 107]

$$
\begin{aligned}
F(x) &= \sum_{i=1}^{5} i \cos((i+1)x_1 + i) \sum_{j=1}^{5} j \cos((j+1)x_2 + j) \\
&\quad + (x_1 + 1.42513)^2 + (x_2 + 0.80032)^2 \\
\text{s.t.} &\quad -10 \le x_i \le 10
\end{aligned}
$$

$x^* = (-1.30685, -1.42485)$, $f(x^*) =$ -176.138.

Levy 3 and Levy 5 are very difficult problems for a local optimizer. The products of different periods of cosines lead to a very rugged landscape with hundreds of local optima. Levy 2 has 9 separate local optima, and Levy 5 is identical to Levy 3 except for the addition of a quadratic term. This quadratic term has the effect of raising the landscape on the edges so that the global minimum that is closest to the point (-1.42513, -0.80032) becomes unique.

Test Problem 6 Beale 1 [56, 107]

$$\begin{aligned} F(x) &= (1.5 - x_1 + x_1x_2)^2 + (2.25 - x_1 + x_1x_2^2)^2 + (2.625 - x_1 + x_1x_2^3)^2 \\ \text{s.t.} \quad & -4.5 \leq x_i \leq 4.5 \end{aligned}$$

$x^* = (3, 0.5)$, $f(x^*) = 0$.

Test Problem 7 Booth [56, 107]

$$\begin{aligned} F(x) &= (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2 \\ \text{s.t.} \quad & -10 \leq x_i \leq 10 \end{aligned}$$

$x^* = (1, 3)$, $f(x^*) = 0$.

Test Problem 8 Matyas [56, 107]

$$\begin{aligned} F(x) &= 0.26(x_1^2 + x_2^2) - 0.48x_1x_2 \\ \text{s.t.} \quad & -10 \leq x_i \leq 10 \end{aligned}$$

$x^* = (0, 0)$, $f(x^*) = 0$.

Matyas appears to be simple since it is just a quadratic. However, the Hessian matrix is very nearly singular along the line $x_1 = x_2$ which causes some problems eliminating regions using zeroth, first or second order information.

Test Problem 9 Branin [56, 97]

$$\begin{aligned} F(x) &= \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos x_1 + 10 \\ \text{s.t.} &\quad -5 \le x_1 \le 10 \\ &\quad 0 \le x_2 \le 15 \end{aligned}$$

$$x^* = \begin{cases} (-3.14159, 12.27500) \\ (3.14159, 2.27500) \\ (9.42478, 2.47500) \end{cases}$$

, $f(x^*) = 0.397887$.

Test Problem 10 Rastrigin [56, 97]

$$\begin{aligned} F(x) &= x_1^2 + x_2^2 - \cos(12x_1) - \cos(18x_2) \\ \text{s.t.} &\quad -1 \le x_i \le 1 \end{aligned}$$

$x^* = (0, 0)$, $f(x^*) = 2$.

Test Problem 11 Griewank 2 [56, 97]

$$\begin{aligned} F(x) &= \frac{x_1^2 + x_2^2}{200} - \cos x_1 \cos\frac{x_2}{\sqrt{2}} + 1 \\ \text{s.t.} &\quad -100 \le x_i \le 100 \end{aligned}$$

$x^* = (0, 0)$, $f(x^*) = 0$.

Test Problem 12 Exp2 [18, 56]

$$\begin{aligned} F(x) &= \sum_{i=1}^{10}\left(e^{-ix_i/10} - 5e^{-ix_2/10} - e^{-i/10} + 5e^{-i}\right)^2 \\ \text{s.t.} &\quad 0 \le x_i \le 20 \end{aligned}$$

$x^* = (1, 10)$, $f(x^*) = 0$.

Test Problem 13 **Treccani [18, 56]**

$$F(x) = x_1^4 + 4x_1^3 + 4x_1^2 + x_2^2$$
$$\text{s.t.} \qquad -5 \le x_i \le 5$$

$$x^* = \begin{cases} (0,0) \\ (-2,0) \end{cases}, \ f(x^*) = 0.$$

Test Problem 14 **Three Hump Camel Back [18, 56]**

$$F = 2x_1^2 - 1.05x_1^4 + \frac{1}{6}x_1^6 + x_1 * x_2 + x_2^2$$
$$\text{s.t.} \qquad -5 \le x_i \le 5$$

$x^* = (0,0)$, $f(x^*) = 0$.

Test Problem 15 **Branin2 [18]**

$$F = \left(1 - 2x_2 + \frac{1}{20}\sin(4\pi x_2) - x_1\right)^2 + \left(x_2 - \frac{1}{2}\sin(2\pi x_1)\right)^2$$
$$\text{s.t.} \qquad -10 \le x_i \le 10$$

$$x^* = \begin{cases} (1,0) \\ (0.148696, 0.402086) \\ (0.402537, 0.287408) \\ (1.59746, -0.287408) \\ (1.85130, -0.402086) \end{cases}, \ f(x^*) = 0.$$

Test Problem 16 **Chichinadze [12, 19, 56]**

$$F = x_1^2 - 12x_1 + 11 + 10\cos\frac{\pi}{2}x_1 + 8\sin(5\pi x_1) - \frac{1}{\sqrt{5}}\exp\left(-\frac{(x_2 - 1/2)^2}{2}\right)$$
$$\text{s.t.} \qquad -30 \le x_1 \le 30$$

$x^* = (5.90133, 0.5)$, $f(x^*) =$ -43.3159. $-10 \le x_2 \le 10$

Test Problem 17 Price [19, 56]

$$\begin{aligned} F(x) &= (2x_1^3x_2 - x_2^3)^2 + (6x_1 - x_2^2 + x_2)^2 \\ \text{s.t.} \quad & -10 \le x_i \le 10 \end{aligned}$$

$$x^* = \begin{cases} (0,0) \\ (2,4) \\ (1.464352, -2.506012) \end{cases}, \; f(x^*) = 0.$$

Test Problem 18 Mccormick [39, 65]

$$F(x) = \sin(x_1 + x_2) + (x_1 - x_2)^2 - 1.5x_1 + 2.5x_2 + 1$$

$$\begin{aligned} \text{s.t.} \quad & -1.5 \le x_1 \le 4 \\ & -3 \le x_2 \le 3 \end{aligned}$$

$x^* = (0,0)$, $f(x^*) = 1$, $-1.5 \le x_1 \le 4$, and $-3 \le x_2 \le 3$

Test Problem 19 Schwefel 3.2 [56, 107]

$$\begin{aligned} F(x) &= \sum_{i=2}^{3} \left[(x_1 - x_i^2)^2 + (1 - x_i)^2 \right] \\ \text{s.t.} \quad & -10 \le x_i \le 10 \end{aligned}$$

$x^* = (1,1,1)$, $f(x^*) = 0$.

Test Problem 20 Levy 8 [56, 107]

$$\begin{aligned} F(x) &= \sin^2(\pi y_1) + \sum_{i=1}^{k-1} (y_i - 1)^2 (1 + 10\sin^2(\pi y_i + 1)) \\ &\quad + (y_k - 1)^2 (1 + \sin^2(2\pi x_k)) \\ y_i &= 1 + \frac{x_i - 1}{4} \\ \text{s.t.} \quad & -10 \le x_i \le 10 \end{aligned}$$

$x_i^* = 1$, $f(x^*) = 0$

Levy 8, is difficult due to the combinations of different periods of the sine function. However, like Levy 5, they are coupled with a quadratic which allows the interval methods to quickly eliminate large regions.

Test Problem 21 Hartman 3 [56, 97]

$$\begin{aligned} F(x) &= -\sum_{i=1}^{4} c_i \exp\left[-\sum_{j=1}^{3} a_{i,j}(x_j - p_{i,j})^2\right] \\ \text{s.t.} & \quad 0 \le x_i \le 1 \end{aligned}$$

$$a = \begin{bmatrix} 0.36890 & 0.11700 & 0.26730 \\ 0.46990 & 0.43870 & 0.74700 \\ 0.10910 & 0.87320 & 0.55470 \\ 0.03815 & 0.57430 & 0.88280 \end{bmatrix}$$

$$p = \begin{bmatrix} 3 & 10 & 30 \\ 0.1 & 10 & 35 \\ 3.0 & 10 & 30 \\ 0.1 & 10 & 35 \end{bmatrix} \qquad c = \begin{bmatrix} 1 \\ 1.2 \\ 3 \\ 3.2 \end{bmatrix}$$

$x^* = (0.114614, 0.555649, 0.852547)$, $f(x^*)$ = -3.86278.

Test Problem 22 Shekel 5 [56, 97]

$$\begin{aligned} F(x) &= -\sum_{i=1}^{5} \frac{1}{\sum_{j=1}^{4}(x_j - a_{i,j})^2 + c_i} \\ \text{s.t.} & \quad 0 \le x_i \le 10 \end{aligned}$$

$$a = \begin{bmatrix} 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 3 & 7 & 3 & 7 \end{bmatrix} \qquad c = \begin{bmatrix} 0.1 \\ 0.2 \\ 0.2 \\ 0.4 \\ 0.4 \end{bmatrix}$$

$x^* = (4.00004, 4.00013, 4.00004, 4.00013)$, $f(x^*)$=-10.1532.

Test Problem 23 Shekel 7 [56, 97]

$$
\begin{aligned}
F(x) &= -\sum_{i=1}^{7} \frac{1}{\sum_{j=1}^{4}(x_j - a_{i,j})^2 + c_i} \\
\text{s.t.} & \quad 0 \le x_i \le 10
\end{aligned}
$$

$$
a = \begin{bmatrix} 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 3 & 7 & 3 & 7 \\ 2 & 9 & 2 & 9 \\ 5 & 5 & 3 & 3 \end{bmatrix} \qquad c = \begin{bmatrix} 0.1 \\ 0.2 \\ 0.2 \\ 0.4 \\ 0.4 \\ 0.6 \\ 0.3 \\ \end{bmatrix}
$$

$x^* = (4.00057, 4.00069, 3.99949, 3.99961)$, $f(x^*)$=-10.4029.

Test Problem 24 Shekel 10 [56, 97]

$$
\begin{aligned}
F(x) &= -\sum_{i=1}^{10} \frac{1}{\sum_{j=1}^{4}(x_j - a_{i,j})^2 + c_i} \\
\text{s.t.} & \quad 0 \le x_i \le 10
\end{aligned}
$$

$$
a = \begin{bmatrix} 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 3 & 7 & 3 & 7 \\ 2 & 9 & 2 & 9 \\ 5 & 5 & 3 & 3 \\ 8 & 1 & 8 & 1 \\ 6 & 2 & 6 & 2 \\ 7 & 3.6 & 7 & 3.6 \end{bmatrix} \qquad c = \begin{bmatrix} 0.1 \\ 0.2 \\ 0.2 \\ 0.4 \\ 0.4 \\ 0.6 \\ 0.3 \\ 0.7 \\ 0.5 \\ 0.5 \end{bmatrix}
$$

$x^* = (4.00075, 4.00059, 3.99966, 3.99951)$, $f(x^*)$=-10.4029.

Test Problem 25 Colville4 [37, 39]

$$\begin{aligned} F(x) &= 100(x_2 - x_2^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2 \\ &\quad +10.1((x_2 - 1)^2 + (x_4 - 1)^2) + 19.8(x_2 - 1)(x_4 - 1) \\ \text{s.t.} &\quad -10 \le x_i \le 10 \end{aligned}$$

$x^* = (1, 1, 1, 1)$, $f(x^*) = 0$.

Colville is a variation on the standard Rosenbrock function but turned out to be similarly difficult. This function has a banana-shaped valley in 4 dimensions as well as a dense 4×4 Hessian as opposed to the sparse Hessian of the 4 dimensional Rosenbrock function.

Test Problem 26 Hartman 6 [56, 97]

$$\begin{aligned} F(x) &= -\sum_{i=1}^{4} c_i \exp\left[-\sum_{j=1}^{6} a_{i,j}(x_j - p_{i,j})^2\right] \\ \text{s.t.} &\quad 0 \le x_i \le 1 \end{aligned}$$

$$p = \begin{bmatrix} 0.1312 & 0.1696 & 0.5569 & 0.0124 & 0.8282 & 0.5886 \\ 0.2329 & 0.4135 & 0.8307 & 0.3736 & 0.1004 & 0.9991 \\ 0.2348 & 0.1451 & 0.3522 & 0.2883 & 0.3047 & 0.6650 \\ 0.4047 & 0.8828 & 0.8732 & 0.5732 & 0.1091 & 0.0381 \end{bmatrix}$$

$$a = \begin{bmatrix} 10 & 17 & 3.5 & 1.7 & 8 & \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{bmatrix} \qquad c = \begin{bmatrix} 1 \\ 1.2 \\ 3 \\ 3.2 \end{bmatrix}$$

$x^* = (0.201690, 0.150011, 0.476874, 0.275332, 0.311652, 0.657300)$

$f(x^*)$ = -3.32237.

Test Problem 27 Sum Squares4 [37, 39]

$$
\begin{aligned}
F(x &= \sum_{i=1}^{n} i * x_i^2 \\
\text{s.t.} & \quad -10 \le x_i \le 10
\end{aligned}
$$

$x_i^* = 0$, $f(x^*) = 0$.

Test Problem 28 Powell's function [81]

$$
\begin{aligned}
F(x) = \textstyle\sum_{j=1}^{n/4} & \quad (x_{4j-3} + 10x_{4j-2})^2 + 5(x_{4j-1} - x_{4j})^2 \\
+ & \quad (x_{4j-2} - 2x_{4j-1})^4 + 10(x_{4j-3} - x_{4j})^4 \\
\text{s.t.} & \quad -4 \le x_i \le 5
\end{aligned}
$$

$X_0 = (3, -1, 0, 1, 3, -1, 0, 1, \ldots, 3, -1, 0, 1)^T$, $f(x^*) = 0$.

Test Problem 29 [17]

$$
\begin{aligned}
F(x) &= \sum_{i=2}^{n} i(2x_i^2 - x_{i-1})^2 + (x_1 - 1)^2 \\
\text{s.t.} & \quad -10 \le x_i \le 10
\end{aligned}
$$

$x^* = \left(1, \frac{1}{2^{1/2}}, \frac{1}{2^{3/4}}, \frac{1}{2^{7/8}}, \ldots, \frac{1}{2^{(2^{n-1}-1)/2^{n-1}}}\right)$, $f(x^*) = 0$.

Test Problem 30 Goldstein-Price [56, 97]

$$
\begin{aligned}
F(x) &= [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 - 6x_1x_2 + 3x_2^2)] \\
& \quad *[20 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)] \\
\text{s.t.} & \quad -2 \le x_i \le 2
\end{aligned}
$$

$x^* = (0, -1)$ $f(x^*) = 3$.

Test Problem 31 Griewank 10 [56, 97]

$$
\begin{aligned}
F(x) &= \sum_{i=1}^{10} \frac{x_i^2}{4000} - \prod_{i=1}^{10} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \\
\text{s.t.} & \quad -600 \le x_i \le 600
\end{aligned}
$$

$x_i^* = 0$, $f(x^*) = 0$.

Griewank 2 and Griewank 10 are both very difficult problems for local optimizers since they have a large number of local optima due to the products of the the cosines. However, they are not very difficult for an interval optimizer since they are raised slightly by a quadratic. This allows large regions to be quickly eliminated with each improvement in the present estimate of the global optima.

Test Problem 32 Schwefel 3.1 [56, 107]

$$\begin{aligned} F(x) &= \sum_{i=1}^{3} \left[(x_1 - x_i^2)^2 + (x_i - 1)^2 \right] \\ \text{s.t.} &\quad -10 \leq x_i \leq 10 \end{aligned}$$

$x^* = (1, 1, 1)$, $f(x^*) = 0$.

Test Problem 33 Kowalik [56, 107]

$$\begin{aligned} F(x) &= \sum_{i=1}^{11} \left(a_i - x_1 \frac{b_i^2 + b_i x_2}{b_i^2 + b_i x_3 + x_4} \right) \\ \text{s.t.} &\quad 0 \leq x_i \leq 0.42 \end{aligned}$$

$$a = \begin{bmatrix} 0.1957 \\ 0.1947 \\ 0.1735 \\ 0.1600 \\ 0.844 \\ 0.627 \\ 0.456 \\ 0.342 \\ 0.323 \\ 0.235 \\ 0.246 \end{bmatrix} \qquad b = \begin{bmatrix} 4 \\ 2 \\ 1 \\ 1/2 \\ 1/4 \\ 1/6 \\ 1/8 \\ 1/10 \\ 1/12 \\ 1/14 \\ 1/16 \end{bmatrix}$$

$x^* = (0.192833, 0.190836, 0.123117, 0.135766)$ $f(x^*) = 3.0748610^{-4}$.

Test Problem 34 Holzmann [37, 39, 40]

$$\begin{aligned}
F(x) &= \sum_{i=1}^{99} f_i^2(x) \\
f_i(x) &= -0.1i + e^{\left(\frac{1}{x_1}(u_i - x_2)^{x_3}\right)} \\
u_i &= 25 + (-50\log(0.01i))^{2/3} \\
\text{s.t.} \quad & .1 \le x_1 \le 100 \\
& 0 \le x_2 \le 25.6 \\
& 0 \le x_3 \le 5
\end{aligned}$$

$x^* = (50, 25, 1.5)$, $f(x^*) = 0$.

Holzman may not appear to be difficult at first glance, but is difficult if only because each evaluation of the function requires a large amount of computer time. It was difficult to solve the problem due to this restriction but, in some communication with Dr. Bennet Fox, it was suggested that one could evaluate only the first few terms and approximate the remainder with a Markov chain. As the size of the box became smaller, one would need a better approximation in order to eliminate a region, but this could be a useful direction of attack on this problem.

Test Problem 35 Holzmann [37, 39, 40]

$$\begin{aligned}
F(x) &= \sum_{i=1}^{n} i * x^4 \\
\text{s.t.} \quad & -10 \le x_i \le 10
\end{aligned}$$

$x_i^* = 0$, $f(x^*) = 0$.

This problem is trivial to solve for almost any gradient based method. However, the Hessian is identically 0 at the solution and this should cause problem for branch and bound methods that rely on gradient or hessian information to quickly eliminate large regions.

5. Results

5.1 Implementation

All of the code for this dissertation was coded in C++ and compiled using the GNU C/C++ compiler version 2.7.2. The programs were run on an IBM RS6000 running AIX version 3. The algorithm was terminated either when the desired tolerance was reached for every box or when 4 hours of CPU time had expired. If less than 4 hours of CPU time was used, then the research executed 10 runs of the algorithm and presents the average run time in this paper. The unit time for this machine as computed by 1000 executions of the Shekel 5 function is 0.03 seconds.

The code will be made available at ftp://interval.usl.edu/pub/interval__math/BackBox. The Back-Boxing software will be updated to include more of the techniques proposed by Hansen (see section 2.5.1) as well as will as support for general constraints rather than just bounded variables.

At the time this research began, no solid public interval domain package was available. For this reason, an interval package was developed expressly for this purpose. It is easily ported to any IEEE machine and requires the modification of only three small functions. The author will aid in the porting upon request if access to the platform is available. A new package for interval arithmetic called BIAS (Basic Interval Arithmetic Software) is available at http://www.ti3.tu-harburg.de/indexEnglisch.html.

Algorithm 11 is used for the local optimizer. The solution of the Newton's equations was implemented using a truncated Conjugate Gradient method. The line search was then implemented by checking if $\alpha = 1$ is acceptable. If it is not, then a cubic is fitted using $\phi(0)$, $\phi'(0)$, $\phi(1)$, and $\phi'(1)$, and the minimum of the cubic is chosen. If it is not acceptable, then the algorithm half steps back until an acceptable step size is found.

The standard branch and bound algorithm that was used for comparison used 0^{th} and 1^{st} order conditions for elimination of a box and used a local optimizer when the function value at the center was close to the best function value already found. This algorithm was then modified to implement Back-Boxing on every box that was large enough and had a function value that was close enough to the best function value already seen.

Early in the research, many methods were explored for storing the list of unchecked boxes. They were stored in a first in, first out method (corresponds to a depth first search), first in last out (corresponds to a breadth first search), sorted by increasing lower bounds, sorted by increasing upper bounds, sorted by increasing volume, sorted by decreasing volume, sorted by decreasing upper bound, and sorted by decreasing lower bound. Although each of these sorting methods may work best for a particular problem type, this research found that sorting the list according to increasing lower bound provided the best results in general.

All derivatives, both real and interval, are computed using reverse automatic differentiation [55]. This lead to a great savings of execution time when the number of variables was large, but it also required more memory than explicitly coding the derivatives. This did not turn out to be a significant problem as the amount of memory required was small. The automatic differentiation was accomplished by using operator overloading in C++. This overloading also imposed overhead above and beyond the overhead for interval arithmetic. On average, each function took approximately 3 times as long to evaluate due to the overhead of automatic differentiation. However, this was made up in the time savings in computing derivatives as well as the time that would be required to debug the derivatives that would need to be entered in to the computer.

The author realizes that this implementation of a standard branch and bound algorithm is by no means optimal. However, conclusions can be drawn from this implementation in terms of the general behavior of the algorithm.

Greenberg [30] states

> "The reasons to perform computational testing in the context of conducting and reporting research are to demonstrate:
> - correctness of model or algorithm
> - quality of solution
> - speed of computation
> - robustness
>
> These are, of course, not exhaustive, but they are enough to serve our purposes."

This algorithm is proven [35] to converge to the correct set of solutions to the GOP. Hence, correctness of the algorithm is not in question as it would be if this algorithm were a local optimization procedure.

Similarly, the quality of the solution is not in question since the algorithms are run until they achieve a predetermined accuracy which must contain the global optima. That leaves the questions of speed of computation, robustness, and whether my code implements the algorithm presented here that must be answered.

Verifying that the software written implements the algorithm presented here is a daunting task and is not possible for very large programs. In order to attempt to verify that this algorithm achieves it's stated goals of globally optimizing a function, the algorithm was executed on all the functions presented here and was able to both find and verify each of the global optima for each of the presented functions. We believe that this gives a strong case for believing that the software written achieves the goal of finding all global optima for a bound constrained function.

To this end, this research has assembled a set of test problems that are widely accepted as being difficult for global optimization algorithms to solve. This was determined in several ways. First a thorough review of the literature was performed and a set of problems that are used to test other global optimization algorithms was uncovered.

Second, the Frequently Asked Questions (FAQ) [90] about nonlinear optimization was consulted. This document is posted monthly on the Usenet Newsgroup sci.op-research as well is available on the WWW at http://www.skypoint.com/subscribers/ashbury/nonlinear-programming-faq.html. This resulted in locating a few more articles and books with global optimization problems as well as a few pieces of source code that are available on the internet.

Finally, this research consulted the Usenet newsgroup sci.op-research and put out a request for difficult global optimization problems. This request was met by several recommendations which duplicated many of the problems found in the first and second portions of this search. This set of problems listed in section 4 was then used to evaluate the performance of the algorithm at a variety of numbers of variables and with several different tolerance levels.

Late in this research, several libraries have started to become available as global optimization test suites. The first is CUTE (Constrained and Unconstrained Testing Environment) which is a set of FORTRAN subroutines, system tools and test problems in the area of nonlinear optimization and nonlinear equations. The package may be obtained via anonymous FTP at camelot.cc.rl.ac.uk (Internet 130.246.8.61), in the directory pub/cute, or at thales.math.fundp.ac.be (Internet 138.48.4.14) in directory cute. Argonne National Laboratories has also released the MinPack-2 Test Problem Collection [3]. This is a set of actual nonlinear optimization problems that have arisen in applied mathematics settings. These problems involve small dimensional application problems with interesting features that make them difficult to solve and so are useful to test the robustness of a global optimization algorithm. They are not large scale, but they can be computationally intensive (some require solving ODE's or PDE's via numerical techniques) and are often difficult to parallelize. The source code for both the function value as well as the gradient is available in FORTRAN. This research did not use the problems from CUTE or from the MinPack-2 test set because it found them late in the research and was not able to easily translate them into usable C++ code.

5.2 Numerical Results

This research applied the Back-Boxing algorithm to most of the problems in the test suite for various ranges of accuracy. The results of these executions can be found in Appendix A. Some of these results are analyzed here. This research also compares the results obtained here with those from Ratz [84] and those of Hansen [35].

For the standard form of Rosenbrock's function, when the number of variables is increased, the execution time of the standard branch and bound algorithm and the Back-Boxing algorithm increase. Also, the execution time for the standard branch and bound algorithm appears to increase exponentially for the higher levels of accuracy. On the other hand, for the Back-Boxing algorithm, the execution time is larger for the lower accuracy but is independent of the accuracy. This is because the global optimum is surrounded by a box of size (approximately) 0.004 and this box has its global local optimum verified.

The 6-hump camel back function again exhibited the standard behaviour for the two algorithms. The standard branch and bound algorithm increased its execution time as the tolerance decreased while the time for Back-Boxing remained constant.

The Shekel series all performed similarly. The Back-Boxing algorithm did not significantly decrease the amount of time required, even for the 10^{-15}case. However, the time required for the Back-Boxing algorithm is fairly constant for tolerances lower than 10^{-5}.

The Levy problems were quite interesting. Due to the extremely large number of local minimum (approximately 700), Back-Boxing was not expected to perform well. It still turned in a good performance, matching a standard branch and bound for low tolerances and it performed significantly better for the higher tolerances. The quadratic nature of Levy-5 was brought out in the faster solution times when compared to Levy-3

The Sum Squares algorithm is the exact sort of problem for which one would expect Back-Boxing to excel. That is exactly what happened. The Back-Boxing algorithm was able to identify the entire region as convex and immediately dismiss it. It should be noted that the standard branch and bound algorithm could have been just as fast if a good partitioning scheme was used. However, a better partitioning scheme is the point of Back-Boxing. Also, the addition of a second order test would not have significantly improved the performance of standard branch and bound since it was slow due to the exponential number of boxes that must be checked.

Historically, the second order information has been used only to attempt to eliminate regions where the function is not convex rather than identify regions where it is convex and leave those to an easier algorithm which is what the Back-Boxing procedure does. That is where the advantage of Back-Boxing appears.

The Beale function was mostly uninteresting since, again, Back-Boxing had constant times for higher tolerances while standard branch and bound increased its time as the tolerance decreased. Similarly the Booth function performed exactly as was expected. Since the function is quadratic, the Back-Boxing algorithm was able to quickly identify this fact, and the entire algorithm terminated with the global optimum the local optimizer had found.

Kowalick was an especially difficult problem for both problems and was not solvable to even 10^{-5} and so no data is provided here. The research feels that this would be a good test problem for all global optimization algorithms due to its simple nature as well as its difficulty.

Matyas is yet another problem that should be easy for the Back-Boxing algorithm to solve. Indeed it was but, surprisingly, it was quite difficult for the standard branch and bound algorithm. The branch and bound algorithm had problems with it due to the large number of local optimization procedures that are applied. They are applied so often because the function is very flat close the line $x_1 = x_2$. Since the function values there are so close to the values for the global best known function value, it continued to apply the local optimizer, even though it had already found the global optimum with the first application.

The Schweffel32 problem was interesting since the time required for Back-Boxing increased with each decrement in the tolerance. The research is unable to provide a good explanation for this behaviour except that the function may simply be too flat at the global optimum to identify a region of convexity.

Branin performed well for Back-Boxing as well in spite of having a large number of local optima. The time for Back-Boxing was mostly constant while the standard branch and bound increased quickly.

Griewank's second problem was quite surprising. It was solved for the same amount of time for each of the tolerances for the Back-Boxing algorithm. However, the algorithm (Back-Boxing and the branch and bound) found the global minimum on the first iteration by chance (design?) and then the Back-Boxing algorithm set the region around the global minimum aside for later processing. The surrounding area was the sum of a quadratic and a cosine which was quickly and easily eliminated. Similarly, the branch and bound function eliminated much of the region around the global optima but then spent the remainder of it's time trying to resolve the region that immediately surrounded the global optima.

However, when the number of variables was increased to 10, Back-Boxing out performed standard branch and bound by a large margin. Back-Boxing again was able to eliminate large regions due to a more intelligent partitioning while standard branch and bound took an extremely long time due to partitioning boxing exactly at the global optimum. The research also tried this problem with the box [-600,6000]. Each method took approximately 18 seconds for 10^{-15}due to standard branch and bound not partitioning exactly or very close to the global optimum.

The research expected the Back-Boxing algorithm to perform better on the Hartman 6 problem since the function appears to be quite well behaved. Again, the time for Back-Boxing was constant for the tolerances tested and standard branch and bound increased quite rapidly.

Exp2 is one of the problems on which Back-Boxing performed well. It was faster for each of the tolerances tested. Back-Boxing required more time as the size of the problem increased just as the non-Back-Boxing algorithm did but the increase was not as large.

Treccani is a well behaved quartic and Back-Boxing performed well on this one also. The time for Back-Boxing was constant over the tested tolerances while standard branch and bound increased rapidly.

The 3-hump camel back problem was perfect for Back-Boxing. The execution time was constant and was significantly faster than the branch and bound algorithm for all the tolerances tested. However, the problem is quite well behaved, has only two variables, and is quadratic in one of the variables. This is the type of problem for which Back-Boxing was designed. It did just as well as could be expected.

Branin 2 was another quartic polynomial and was quite difficult for both algorithms. Both solved the problem very quickly for low tolerances but, as the accuracy increased, each one required huge amounts of CPU time to solve the problem. In fact, neither was able to solve it to a tolerance of 10^{-15} within the 4 hour limit. It should be noted that both had located each global optimum and that the remaining time was spent in verifying to the required tolerance.

Chichinadze was another problem that displayed the standard behavior of the Back-Boxing algorithm. Back-Boxing was slower for 10^{-5} but was equal to or faster than branch and bound for $10^{-10}$and 10^{-15}. Again, this is due to the fact that it is able to identify a large region (or regions) in which the function is convex and then ignore those until later in the algorithm.

Price was a bit of a surprise. It is a 6^{th} order polynomial, but Back-Boxing still performed well. Back-Boxing was equally fast for 10^{-5} and significantly faster for 10^{-10} and 10^{-15}. The number of Back-Boxing processes increased along with the tolerance but the number of interval evaluations did not increase as rapidly as they did for the branch and bound algorithm.

McCormick was another problem where Back-Boxing performed very well. McCormick is the sum of a quadratic and a sine function. The Back-Boxing algorithm was able to identify the global optima quickly and then set a large region aside for later processing. It was significantly faster than branch and bound for all tolerances, partially due to the single local optimization procedure rather than the large number of them for Back-Boxing. Branch and bound applied so many local optimization procedures because it examined many boxes that are close to the global optima (Back-Boxing set these boxes aside in one large box). These boxes all had very low function values leading to the application of the local optimizer a large number of times.

Colville was another problem for which Back-Boxing was a good procedure. Colville is quite similar to Rosenbrock's function, but it has two more variables and a slightly more dense Hessian than the 4 dimensional Rosenbrock has. Back-Boxing out-performed branch and bound at every accuracy since it was able to identify a relatively large region around the global optima as convex. This led to constant times for tolerances of 10^{-5} , 10^{-10} and 10^{-15} as well as constant evaluations for both real and interval numbers.

When compared to results from other sources, the Back-Boxing algorithm still appears to perform well. Ratz [85] provides some comparisons between his algorithm and that of Hansen [35]. E_effort_1 is

Function Initial Box $[x]_i$	Levy 5 [-10,10]			Levy 8, 10 dimension [-10,10]		
$\epsilon = 10^{-12}$	Han	Ratz	New	Han	Ratz	New
FE	2166	59	1235	559	89	885
GE	2021	319	444	497	177	423
HE	729	69	48	184	41	89
E_effort_1	34704	3905	2267	144268	34099	75875
E_effort_2	26288	2853	2223	22787	5307	12367

Table 5.1. Comparison of Back-Boxing, Hansen, and Ratz applied to Levy functions

computed as FE + $3n$GE + $7n^2$HE for forward automatic differentiation. E_effort_2 is computed as FE + 4GE + $11n$HE, which corresponds to the effort required for reverse automatic differentiation. This data can be viewed in Tables 5.2 and 5.2

The Back-Boxing algorithm does not perform significantly better or worse than Ratz in Table 5.2, nor was it significantly faster than Ratz in Table 5.2 for the box $X_i = (-1.2, 1.2)$. When the box was increased to $X_i = (-10^6, 10^6)$, the Back-Boxing algorithm was significantly faster than Ratz or Hansen due to the better partitioning scheme. This reinforces the intuition that drove the development of Back-Boxing. It should be noted that the Back-Boxing algorithm outperformed the results of Hansen in every one of these test cases. As implemented for this research, the Back-Boxing algorithm does not take advantage of some of the techniques of Hansen (see section 2.5.1 and [35]), and the performance of the basic algorithm could be greatly enhanced with these techniques.

For the Back-Boxing algorithm, the interval evaluations + 1/2 of the real evaluations (loose approximation that interval evaluation is approximately twice as expensive as a real evaluation) are reported This

Function	Rosenbrock					
Initial Box $[x]_i$	[-1.2,1.2]			$[-10^6, 10^6]$		
$\epsilon = 10^{-12}$	Han	Ratz	New	Han	Ratz	New
FE	640	111	145	12321	1213	271
GE	583	187	103	12827	2399	193
HE	238	50	57	4949	593	57
E_effort_1	10802	2633	2359	227855	32211	3025
E_effort_2	9958	2520	1811	210988	31052	2297

Table 5.2. Comparison of Back-Boxing, Hansen, and Ratz, applied to Rosenbrock's Function

appears to be in spirit with the reporting of Ratz [85].

5.3 Conclusions and Directions for Further Research

This research has identified a new method for solving the global optimization problem. It is not always the fastest method when compared with other codes, but it appears to be faster than the standard branch and bound algorithm when a high degree of accuracy is required for a large number of variables. The problems for which the Back-Boxing algorithm performs poorly appear to be a class of pathological problems that contain a great number of both local and or global optima in a small region.

5.3.1 Open Questions on Back-Boxing

There are several ideas that are not considered in this algorithms development. First, the Back-Boxing increases the box in a uniform fashion. This, in general, probably is not the optimal process. It may be wiser to grow the box in different dimensions more rapidly than in others. The directions of growth of the box probably depend on the size of the first, second, and third derivatives, but this has not yet been explored. Also, some of the ideas of Ratz [85] could be adapted to the Back-Boxing algorithm in order to speed up the elimination of regions.

Second, the method for finding the largest box is presently based on a bisection type algorithm. Higher order methods could be developed to find the largest box which require fewer computations of the Hessian and/or fewer Gaussian eliminations of the Hessian matrix.

Third, this research has only used two methods for identifying when every symmetric sub-matrix of an interval matrix is positive definite. There may be more faster methods than are uncovered here. Most importantly, using Gaussian elimination to determine whether the Hessian matrix is definite is computationally expensive. It also does not take advantage of the fact that we are only considering symmetric sub matrices of the interval Hessian matrix. Methods that take less time or are more accurate in detecting definiteness would be useful to speeding up the Back-Boxing algorithm.

Fourth, the computation of $B - B^*$ in Algorithm 14 is simply done in the order of the variables. A more intelligent method of computing this set could lead to improved speed for the algorithm.

Finally, there are a great deal of improvements that can be made in the general algorithm. The local optimizer is quite trivial and could be improved with some work.

Appendix A. Data for Numerical Results

The time (calculated using the wall clock difference between the start and end of the algorithm run) presented is measured in seconds and is the average of at least 3 executions of the code on that problem. If the execution time is greater than 4 hours, the run was terminated. The other columns detail the following.

The time for each method is presented both when using Back-Boxing and when not using Back-Boxing. The results have also been split into tables for the different tolerances as well as for two dimensional problems versus higher dimensional problems. The unit time for the test platform (computed from 1000 evaluations of the Shekel 5 function) is 0.028 seconds.

time The average execution time. '*'s for the time mean that the algorithm was terminated after 4 hours of CPU time.

fctn The number of real function evaluations.

grad The number of real gradient evaluations.

hessv The number of real Hessian vector products computed.

lopt The number of implementations of the local optimizer.

ifctn The number of interval function evaluations.

igrad The number of interval gradient evaluations.

ihess The number of interval Hessian evaluations.

BB The number of times Back-Boxing was implemented.

#	n	time	Real fctn	Real grad	Real hessv	Real lopt	Interval fctn	Interval grad	Interval hess	BB
1	2	0.0700	64	68	58	1	400	133	0	0
1	2	0.0700	73	75	62	1	132	70	24	1
2	2	0.0633	64	68	58	1	400	133	0	0
2	2	0.0600	73	75	62	1	132	70	24	1
3	2	0.5933	425	223	164	8	2676	1181	0	0
3	2	0.5733	384	100	52	5	1830	923	93	5
4	2	8.8600	346	168	152	6	4930	1899	0	0
4	2	8.7933	963	316	238	12	3299	1443	233	12
5	2	2.1000	427	211	192	12	1105	386	0	0
5	2	1.9667	131	56	54	2	876	347	39	2
6	2	0.6633	406	73	76	1	3628	1222	0	0
6	2	0.4667	683	269	76	1	1976	684	23	1
7	2	0.0733	46	12	8	1	546	182	0	0
7	2	0.0400	80	16	8	1	15	33	24	1
8	2	2.3800	9708	8742	7908	813	3065	1095	0	0
8	2	0.0400	5	4	0	1	136	91	23	1
9	2	2.6167	15229	7259	7670	468	2257	601	0	0
9	2	0.3667	683	191	134	13	472	495	166	13
10	2	0.1200	312	270	160	41	277	79	0	0
10	2	0.0300	5	4	0	1	16	26	20	1
11	2	0.0700	4	2	0	1	381	127	0	0
11	2	0.0867	5	4	0	1	16	79	50	1
12	2	0.6633	261	165	170	3	807	270	0	0
12	2	0.6000	185	120	128	3	439	202	26	3
13	2	0.5467	622	240	228	18	2907	1398	0	0
13	2	0.3067	373	104	72	7	734	558	128	7
14	2	1.6800	6694	5266	2776	91	1613	681	0	0
14	2	0.2200	5	4	0	1	1016	490	22	1
15	2	0.8333	39	33	18	1	3736	1303	0	0
15	2	0.8867	44	37	18	1	3658	1303	23	1
16	2	0.1933	633	281	206	15	316	135	0	0
16	2	0.2600	395	146	116	6	209	227	87	6
17	2	2.7433	10752	8479	5780	334	2011	563	0	0
17	2	1.3633	1461	1064	584	33	1065	1188	409	33
18	2	0.2133	384	153	124	11	874	421	0	0
18	2	0.1000	93	24	12	1	100	102	40	1

Table Appendix A.1: Tolerance, 1e-5, 2 variables

#	n	time	Real				Interval			BB
			fctn	grad	hessv	lopt	fctn	grad	hess	
27	2	1.0467	5860	4754	4224	61	369	103	0	0
27	2	0.0400	5	4	0	1	16	29	23	1
29	2	0.0633	214	158	52	2	298	99	0	0
29	2	0.0800	206	165	56	2	193	92	22	2
30	2	2800.4733	220	113	104	4	872649	328203	0	0
30	2	2744.6567	2681	1538	1612	109	784782	289659	1550	109

Table Appendix A.2: Tolerance, 1e-5, 2 variables (continued)

#	n	time	Real				Interval			BB
			fctn	grad	hessv	lopt	fctn	grad	hess	
19	3	0.1100	151	115	22	1	670	223	0	0
19	3	0.1100	239	178	22	1	229	103	24	1
20	3	0.1033	99	101	60	1	195	64	0	0
20	3	0.1833	290	367	60	1	26	36	24	1
21	3	5.2267	10678	5972	9852	267	931	315	0	0
21	3	0.7400	833	320	456	9	324	306	36	9
20	4	0.1967	198	203	228	2	266	92	0	0
20	4	0.3133	334	370	224	2	82	84	23	2
22	4	0.3667	865	299	128	19	339	140	0	0
22	4	0.5433	86	30	12	1	74	87	42	1
23	4	0.2967	161	63	32	3	337	150	0	0
23	4	0.8200	94	30	18	1	156	113	42	1
24	4	0.4833	190	74	36	3	353	158	0	0
24	4	1.1733	124	34	20	1	180	121	42	1
25	4	219.1967	191	151	64	1	456462	193664	0	0
25	4	189.2967	742	550	72	1	384289	162187	24	1
28	4	0.3667	181	144	124	1	1810	615	0	0
28	4	0.4100	218	170	180	1	1617	574	22	1
20	5	0.2467	237	204	144	2	331	115	0	0
20	5	0.3667	300	296	140	2	86	85	23	2
26	6	4.7233	1015	571	1692	27	3386	1466	0	0
26	6	3.8367	409	244	684	8	1753	839	55	8
20	8	0.6300	370	340	358	2	525	183	0	0
20	8	0.9867	409	429	350	2	248	162	23	2
20	10	1.1067	388	410	602	2	655	229	0	0
20	10	1.5300	377	402	588	2	263	167	23	2
31	10	535.8100	4	2	0	1	558081	186367	0	0
31	10	1.4433	5	4	0	1	48	85	54	1
35	4	12.9533	42244	63362	23200	529	2913	799	0	0
35	4	0.0233	4	2	0	1	22	6	0	1
2	4	0.2300	185	147	124	1	850	283	0	0
2	4	0.3067	763	546	124	1	296	126	0	1
2	8	0.8000	351	279	336	1	1750	583	0	0
2	8	1.9533	874	679	336	1	3328	1154	0	1
2	16	3.7300	683	609	896	1	3550	1183	0	0
2	16	789.3933	1150	1003	922	1	853278	286642	0	1

Table Appendix A.3: Tolerance, 1e-5, large variable

#	n	time	Real				Interval			BB
			fctn	grad	hessv	lopt	fctn	grad	hess	
35	8	959.3	2375684	3563522	1095168	14849	89857	25087	0	0
35	8	0.027	4	2	0	1	38	6	0	1
35	16	*.***	4	2	0	1	1032338	387802	0	0
35	16	0.110	4	2	0	1	70	6	0	1
27	4	1.203	2180	2594	3200	401	2785	799	0	0
27	4	0.060	5	4	0	1	24	29	0	1
27	8	76.46	57092	75778	217088	10753	85761	25087	0	0
27	8	0.173	5	4	0	1	40	29	0	1
27	16	*.***	4	2	0	1	968222	366430	0	0
27	16	0.750	5	4	0	1	72	29	0	1
27	32	*.***	4	2	0	1	526601	263299	0	0
27	32	3.243	5	4	0	1	136	29	0	1

Table Appendix A.4: Tolerance, 1e-5, large variable (continued)

#	n	time	Real fctn	Real grad	Real hessv	Real lopt	Interval fctn	Interval grad	Interval hess	BB
1	2	0.1200	196	173	60	3	720	249	0	0
1	2	0.0800	73	75	62	1	132	91	24	1
2	2	0.1700	196	173	60	3	720	249	0	0
2	2	0.0767	73	75	62	1	132	91	24	1
3	2	0.8700	1195	339	164	25	3307	1489	0	0
3	2	0.7167	384	100	52	5	1830	939	93	5
4	2	25.6567	15923	4199	3974	379	6995	2745	0	0
4	2	10.2800	963	316	238	12	3299	1508	233	12
5	2	3.2567	1126	356	326	28	1272	462	0	0
5	2	2.2600	123	62	52	2	876	347	39	2
6	2	1.3800	406	73	76	1	6799	2273	0	0
6	2	0.4933	683	269	76	1	1976	684	23	1
7	2	0.1600	46	12	8	1	980	327	0	0
7	2	0.0433	80	16	8	1	15	33	24	1
8	2	5.2167	17738	16886	15236	1637	5573	1919	0	0
8	2	0.0433	5	4	0	1	136	91	23	1
9	2	11.1567	97721	17864	10520	2679	9083	2812	0	0
9	2	0.4233	683	191	134	13	472	532	166	13
10	2	0.3200	1592	706	360	105	533	143	0	0
10	2	0.0200	5	4	0	1	16	26	20	1
11	2	0.1267	4	2	0	1	597	199	0	0
11	2	0.0867	5	4	0	1	16	79	50	1
12	2	0.9433	261	165	170	3	1498	499	0	0
12	2	0.7633	185	120	128	3	868	365	26	3
13	2	0.8333	622	240	228	18	5593	2726	0	0
13	2	0.2967	373	104	72	7	734	558	128	7
14	2	1.9733	9660	7660	3664	187	1933	777	0	0
14	2	0.2100	5	4	0	1	1016	490	22	1
15	2	3094.4300	39	33	18	1	795672	293042	0	0
15	2	3321.3000	44	37	18	1	810634	298592	23	1
16	2	0.4533	2544	649	310	60	493	201	0	0
16	2	0.2833	395	146	116	6	209	264	87	6
17	2	5.9600	25996	20033	10486	783	3806	1012	0	0
17	2	1.6833	2270	1748	890	55	1325	1526	456	55
18	2	0.9567	7549	1359	124	193	1840	813	0	0
18	2	0.0867	93	24	12	1	100	121	40	1

Table Appendix A.5: Tolerance, 1e-10, 2 variables

#	n	time	Real				Interval			BB
			fctn	grad	hessv	lopt	fctn	grad	hess	
27	2	2.1733	12068	9810	6784	125	625	167	0	0
27	2	0.0367	5	4	0	1	16	29	23	1
29	2	0.1633	314	238	52	3	620	210	0	0
29	2	0.1367	250	193	56	2	345	164	22	2
30	2	*.****	220	113	104	4	2125184	956769	0	0
30	2	2886.6233	9266	7828	8466	702	750819	287208	6156	702

Table Appendix A.6: Tolerance, 1e-10, 2 variables (continued)

#	n	time	Real				Interval			BB
			fctn	grad	hessv	lopt	fctn	grad	hess	
19	3	0.2433	151	115	22	1	1177	392	0	0
19	3	0.1433	239	178	22	1	229	103	24	1
20	3	0.1800	200	203	60	2	350	116	0	0
20	3	0.2067	290	367	60	1	26	57	24	1
21	3	10.1133	23696	9974	14452	603	1939	651	0	0
21	3	0.8267	833	320	456	9	324	306	36	9
20	4	0.3333	198	203	228	2	470	160	0	0
20	4	0.3700	334	370	224	2	82	84	23	2
22	4	0.8300	2129	524	174	52	644	276	0	0
22	4	0.6100	86	30	12	1	74	87	42	1
23	4	0.8767	1324	318	74	37	643	286	0	0
23	4	0.9367	94	30	18	1	156	133	42	1
24	4	1.1233	913	205	72	22	644	294	0	0
24	4	1.3533	124	34	20	1	180	141	42	1
28	4	1.0667	220	171	126	2	4718	1780	0	0
28	4	1.2500	216	171	126	2	5411	2052	22	2
25	4	501.0700	2085	1533	68	10	974512	412796	0	0
25	4	187.1367	742	550	72	1	384289	162187	24	1
20	5	0.4100	237	204	144	2	586	200	0	0
20	5	0.4333	300	296	140	2	86	85	23	2
26	6	24.1200	34716	10889	19224	877	5938	2316	0	0
26	6	3.1700	409	244	684	8	1753	854	55	8
20	8	1.0900	370	340	358	2	933	319	0	0
20	8	1.1167	409	429	350	2	248	162	23	2
20	10	1.8300	388	410	602	2	1165	399	0	0
20	10	1.7400	377	402	588	2	263	167	23	2
31	10	721.8167	4	2	0	1	803841	268287	0	0
31	10	1.2367	5	4	0	1	48	85	54	1
2	4	0.5633	185	147	124	1	2136	738	0	0
2	4	0.3533	763	546	124	1	296	126	0	1
2	8	1.5500	351	279	336	1	3115	1038	0	0
2	8	2.3633	874	679	336	1	3328	1154	0	1
2	16	185.4800	1557	1249	902	2	170337	61314	0	0
2	16	838.4800	1150	1003	922	1	853278	286642	0	1

Table Appendix A.7: Tolerance, 1e-10, large variable

#	n	time	Real				Interval			BB
			fctn	grad	hessv	lopt	fctn	grad	hess	
35	4	21.94	81956	122930	23200	1041	4961	1311	0	0
35	4	0.013	4	2	0	1	22	6	0	1
35	8	1827	4997124	7495682	1096704	31233	155393	41471	0	0
35	8	0.040	4	2	0	1	38	6	0	1
35	16	*.***	4	2	0	1	936398	355822	0	0
35	16	0.103	4	2	0	1	70	6	0	1
27	4	2.657	4996	5922	7296	913	4833	1311	0	0
27	4	0.067	5	4	0	1	24	29	0	1
27	8	186.6	143108	190466	544768	27137	151297	41471	0	0
27	8	0.187	5	4	0	1	40	29	0	1
27	16	*.***	4	2	0	1	1033316	388128	0	0
27	16	0.640	5	4	0	1	72	29	0	1
27	32	*.***	4	2	0	1	562157	281077	0	0
27	32	2.763	5	4	0	1	136	29	0	1

Table Appendix A.8: Tolerance, 1e-10, large variable (continued)

#	n	time	Real				Interval			BB
			fctn	grad	hessv	lopt	fctn	grad	hess	
1	2	0.4167	366	213	60	9	2982	1209	0	0
1	2	0.0700	73	75	62	1	132	91	24	1
2	2	0.3767	366	213	60	9	2982	1209	0	0
2	2	0.0733	73	75	62	1	132	91	24	1
3	2	1.6133	9491	1883	164	325	4209	1789	0	0
3	2	0.5967	384	100	52	5	1830	986	93	5
4	2	27.7333	16207	4265	3974	397	11440	4953	0	0
4	2	8.5600	963	316	238	12	3299	1588	233	12
5	2	3.2267	1182	378	326	32	1617	622	0	0
5	2	1.8500	83	59	52	2	876	347	39	2
6	2	1.8700	482	115	76	11	10919	3795	0	0
6	2	0.4467	683	269	76	1	1976	704	23	1
7	2	0.1967	60	26	8	6	1588	569	0	0
7	2	0.0400	80	16	8	1	15	33	24	1
8	2	7.4267	32668	36962	18472	2481	8145	2763	0	0
8	2	0.0467	5	4	0	1	136	91	23	1
9	2	17.6667	198096	28462	10520	5337	17006	5470	0	0
9	2	0.3767	683	191	134	13	472	546	166	13
10	2	0.4500	2968	1090	488	169	789	207	0	0
10	2	0.0267	5	4	0	1	16	26	20	1
11	2	0.1433	4	2	0	1	801	275	0	0
11	2	0.0833	5	4	0	1	16	79	50	1
12	2	1.2667	261	165	170	3	2117	713	0	0
12	2	0.9467	185	120	128	3	1194	476	26	3
13	2	1.2367	622	240	228	18	8267	4062	0	0
13	2	0.2933	373	104	72	7	734	558	128	7
14	2	2.5867	13468	10794	4416	295	2293	885	0	0
14	2	0.2067	5	4	0	1	1016	490	22	1
15	2	*.****	39	33	18	1	1202166	475520	0	0
15	2	*.****	39	33	18	1	1204769	474466	21	1
16	2	0.5800	4068	1098	310	154	774	295	0	0
16	2	0.2500	395	146	116	6	209	286	87	6
17	2	6.3967	34813	26057	11508	1222	5548	1451	0	0
17	2	1.5467	2368	1924	1024	57	1343	1640	460	57
18	2	1.9967	19964	4193	124	801	3739	1421	0	0
18	2	0.0800	93	24	12	1	100	121	40	1

Table Appendix A.9: Tolerance, 1e-15, 2 variable

#	n	time	Real				Interval			BB
			fctn	grad	hessv	lopt	fctn	grad	hess	
27	2	2.8033	19052	15498	7384	197	913	239	0	0
27	2	0.0333	5	4	0	1	16	29	23	1
29	2	0.1933	344	258	52	6	1018	363	0	0
29	2	0.1533	248	193	56	2	556	266	22	2
30	2	*.****	220	113	104	4	2125184	956769	0	0
30	2	2575.0200	9266	7828	8466	702	750819	287243	6156	702

Table Appendix A.10: Tolerance, 1e-15, 2 variable (continued)

#	n	time	Real				Interval			BB
			fctn	grad	hessv	lopt	fctn	grad	hess	
20	3	0.266	497	337	60	14	563	199	0	0
20	3	0.180	290	367	60	1	26	57	24	1
19	3	0.366	210	140	22	6	2248	833	0	0
19	3	0.123	239	178	22	1	229	124	24	1
21	3	9.690	29924	11604	14452	891	2801	939	0	0
21	3	0.713	833	320	456	9	324	321	36	9
20	4	0.413	408	304	228	13	756	277	0	0
20	4	0.297	334	370	224	2	82	102	23	2
22	4	1.047	3517	800	174	109	875	363	0	0
22	4	0.543	86	30	12	1	74	107	42	1
23	4	1.510	3349	776	74	127	968	404	0	0
23	4	0.827	94	30	18	1	156	133	42	1
24	4	1.380	913	205	72	22	830	387	0	0
24	4	1.263	124	34	20	1	180	141	42	1
25	4	711.1	4439	2882	68	70	1550359	662500	0	0
25	4	159.9	742	550	72	1	384289	162208	24	1
28	4	0.080	0	0	0	0	0	0	0	0
28	4	1.647	241	191	126	3	8202	3176	24	3
20	5	0.550	410	266	144	13	943	348	0	0
20	5	0.377	300	296	140	2	86	103	23	2
26	6	30.38	65019	17163	19224	2003	9322	3442	0	0
26	6	3.083	409	244	684	8	1753	854	55	8
20	8	1.420	536	394	358	10	1484	544	0	0
20	8	0.993	409	429	350	2	248	180	23	2
20	10	2.203	578	503	602	11	1876	703	0	0
20	10	1.460	377	402	588	2	263	185	23	2
31	10	1068	833540	103426	0	20481	1139713	399359	0	0
31	10	1.227	5	4	0	1	48	85	54	1
2	4	4.723	457	309	124	10	22627	10048	0	0
2	4	0.303	763	546	124	1	296	147	0	1
2	8	852.9	420	327	336	7	1518824	717835	0	0
2	8	1.990	874	679	336	1	3328	1175	0	1
2	16	*.***	2453	1953	902	11	2384142	1191176	0	0
2	16	734.4	1150	1003	922	1	853278	286663	0	1

Table Appendix A.11: Tolerance, 1e-15, large variable

#	n	time	Real				Interval			BB
			fctn	grad	hessv	lopt	fctn	grad	hess	
35	4	22.03	83108	124658	23200	1617	7265	1887	0	0
35	4	0.010	4	2	0	1	22	6	0	1
35	8	1825	5195780	7793666	1096704	49665	229121	59903	0	0
35	8	0.033	4	2	0	1	38	6	0	1
35	16	*.***	4	2	0	1	964778	365282	0	0
35	16	0.103	4	2	0	1	70	6	0	1
27	4	9.337	40836	36018	10752	1489	7137	1887	0	0
27	4	0.060	5	4	0	1	24	29	0	1
27	8	719.7	2424068	2025986	763904	45569	225025	59903	0	0
27	8	0.193	5	4	0	1	40	29	0	1
27	16	*.***	4	2	0	1	1044044	391704	0	0
27	16	0.680	5	4	0	1	72	29	0	1
27	32	*.***	4	2	0	1	525005	262501	0	0
27	32	2.710	5	4	0	1	136	29	0	1

Table Appendix A.12: Tolerance, 1e-15, large variable (Continued)

BIBLIOGRAPHY

[1] L.S. Anderssen, R.S. Jennings and D.M. Ryan. Global optimization. In R.S. Anderssen, editor, **Optimization**. University of Queensland Press, 1972.

[2] Kendall E. Atkinson. **An Introduction to Numerical Analysis**. John Wiley & Sons, New York, New York, 1989.

[3] Richard G. Averick, Brett M. Carter and Jorge J. Moré. The Minpack-2 test problem collection (preliminary version). Technical Memorandum 150, Argonne National Laboratory, Mathematics and Computer Science Division, 9700 South Cass Avenue, Argonne, Illinois, 60439, May 1991.

[4] E.M.L. Beale. Two transportation problems. In **Actes de la Beme Converence International de Recharche Operational, Oslo, 1963**, 1964.

[5] E.M.L. Beale and J.A. Tomlin. Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables. In John Lawrence, editor, **Proceeding of the Fifth International Conference on Operation Research**. Tavistock Publications, 1970.

[6] Harold P. Benson. On the convergence of two branch-and-bound algorithms for nonconvex programming problems. **Journal of Optimization Theory and Applications**, 36(1), 1982.

[7] Harold P. Benson and Sayin Serpil. A finite concave minimization algorithm using branch and bound and neighbor generation. **Journal of Global Optimization**, 5:1–14, 1994.

[8] William Briggs. Results on Gaussian elimination for diagonally dominant and positive definite matrices. Notes distributed in mathematics department, University of Colorado, Denver.

[9] Daniel G. Brooks and William A Verdini. Computational experience with generalized simulated annealing over continuous variables. **American Journal of Mathematical and Management Sciences**, 8(3 & 4):425+, 1988.

[10] S. H. Brooks. A discussion of random methods for seeking maxima. **Operations Research**, 6:244–251, 1958.

[11] Hue Sun Chan and Ken A. Dill. The protein folding problem. **Physics Today**, February 1993.

[12] V. Chichinadze. The ψ-transform for solving linear and nonlinar programming problems. **Automata**, 5:347–355, 1969.

[13] A. R. Colville. A comparative study on nonlinear programming codes. Technical Report 320-2949, IBM Scientific Center Report, New York, 1968.

[14] T. Csendes and D. Ratz. Subdivision direction selection in interval methods for global optimization. **SIAM Journal of Numerical Analysis**, 1995. To Appear.

[15] Anton Dekkers and Emile Aarts. Global optimization and simulated annealing. **Mathematical Programing**, 50:367, 1991.

[16] Ron S. Dembo and Trond Steihaug. Truncated-Newton algorithms for large-scale optimization. **Mathematical Programming**, 26:190–212, 1982.

[17] L. C. Dixon and Richard C. Price. The truncated Newton method for sparse unconstrained optimisation using automatic differentiation. **J. Opt. Theory and Appl.**, 60(2):261 +, February 1989.

[18] Szegö Dixon, L.C. Dixon, editor. **Towards global optimization**. North-holland Publishing Company, New York, 1975.

[19] Szegö Dixon, L.C. Dixon, editor. **Towards global optimization, volume 2**. North-holland Publishing Company, New York, 1978.

[20] Erik Willy Dravnieks. **A Methodology for the Formulation and Solution of Global Optimization Problems**. PhD thesis, Carleton University, 1994.

[21] Thomas Guthrie Weidner Epperly. **Global optimization of nonconvex nonlinear programs using parallel branch and bound**. PhD thesis, University of Wisconsin – Madison, 1995.

[22] J. E. Falk and K. R. Hoffman. A successive underestimation method for concave minimization problems. **Mathematics of Operations Research**, 1(3):251–259, August 1976.

[23] J. E. Falk and R. M. Soland. An algorithm for separable nonconvex programming problems. **Management Science**, 15(9):550–569, May 1969.

[24] C. A. Floudas and V. Visweswaran. A global optimization (GOP) for certain classes of nonconvex NLP's—I. Theory. **Computers & Chemical Engineering**, 14(12):1397–1417, 1990.

[25] Bennet L. Fox. Simulated annealing: Folklore, facts, and directions. October 1994.

[26] Bennett L. Fox. Faster simulated annealing. **SIAM Journal on Optimization**, 5(3):488,497, Aug 1995.

[27] E. A. Galperin. The cubic algorithm. **Journal of Mathematical Analysis and Applications**, 112:635–640, 1985.

[28] E. A. Galperin and Q. Zheng. Nonlinear observation via global optimization methods. In **Proceedings of the 1983 Conference on Information Sciences and Systems**, pages 620–628, 1983.

[29] Harvey J Greenberg. Bounding nonconvex programs by conjugates. **Operations Research**, 21:346–348, 1973.

[30] Harvey J. Greenberg. Computational testing: Why, how and how much. **ORSA Journal on Computing**, 2(1):94–97, Winter 1990.

[31] Andreas Griewank. Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation. **Optimization Methods and Software**, 1(1), 1993. Also appeared as Preprint MCS–P228–0491, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., April 1991.

[32] Andreas Griewank, David Juedes, , and Jean Utke. ADOL-C, a package for the automatic differentiation of algorithms written in C/C++. **ACM Trans. Math. Software**, to appear. Also

appeared as Preprint MCS–P180–1190, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., November 1990.

[33] A. S. E. Hamed and G. P. McCormick. Calculation of bounds on variables satisfying nonlinear inequality constraints. **Journal of Global Optimization**, 3:25–47, 1993.

[34] B. Hansen, P. Jaumard and J. Xiong. Decomposition and interval arithmetic. **Journal of Global Optimization**, 3:421–437, 1993.

[35] E. R. Hansen. **Global Optimization using Interval Analysis**. Springer-Verlag, Berlin, 1993.

[36] George W. Heine. **Smart Simulated Annealing, PhD Dissertation**. PhD thesis, University of Colorado at Denver, 1994.

[37] D. M. Himmelblau and R.V. Yates, editors. **Applied non-linear programming**. McGraw-Hill Book Company, New York, 1972.

[38] Willi Hock and Klaus Schittkowski. **Test examples for Nonlinear Programming Codes**. Springer-Verlag, New York, 1981.

[39] Willi Hock and Klaus Schittkowski, editors. **Test Examples for Nonlinear Programming Codes**. Springer-Verlag, Berlin, 1987.

[40] A. G. Holzman. Comparative analysis of nonlinear programming codes with the Weisman algorithm. Technical Report 113, University of Pittsburgh, Pittsburgh, 1969.

[41] N. V. Horst, Reiner Thoai and H. P. Benson. Concave minimization via conical partitions and polyhedral outer approximation. **Journal of Global Optimization**, 2:1–19, 1992.

[42] R. Horst and Hoang Tuy. On the convergence of global methods in multiextremal optimization. **Journal of Optimization Theory and Applications**, 54(2):253–271, August 1987.

[43] Reiner Horst. An algorithm for nonconvex programming problems. **Mathematical Programming**, 10:312–321, 1976.

[44] Reiner Horst. A note on the convergence of an algorithm for nonconvex programming problems. **Mathematical Programming**, pages 237–238, 1980.

[45] Reiner Horst. A general class of branch-and-bound methods in global optimization with some new approaches for concave minimization. **Journal of Optimization Theory and Applications**, 51(2):271–291, 1986.

[46] Reiner Horst. Deterministic global optimization with partition sets whose feasibility is not known: Applications to concave minimization, reverse convex constraints, dc-programming and Lipschitzian optimization. **Journal of optimization Theory and Applications**, 58(1):11–37, July 1988. Short Communication.

[47] Reiner. Horst and N. V. Thoai. Conical algorithm for the global minimization of linearly constrained decomposable concave minimization problems. **Journal of Optimization Theory and Applications**, 74(3):469–486, 1992.

[48] Reiner. Horst and N. V. Thoai. Constraint decomposition algorithms in global optimization. **Journal of Global Optimization**, 5:333–348, 1994.

[49] Reiner Horst and Hoang Tuy. **Global Optimization**. Springer-Verlag, Berlin, 1990.

[50] V. Horst, Reiner. Thoai and J. De Vries. A new simplicial cover technique in constrained global optimization. **Journal of Global Optimization**, 2:1–19, 1992.

[51] A.L. Ingber. Very fast simulated re-annealing. **Journal of Mathematical Computer Modeling**, 12(8):967–973, 1989.

[52] A.L. Ingber. Simulated annealing: Practice versus theory. **Journal of Mathematical Computer Modeling**, 18(11):29–57, 1993.

[53] A.L. Ingber. Adaptive simulated annealing (ASA): Shades of annealing. **Journal of Physics review**, 1995 (submitted).

[54] A.L. Ingber and A.B. Rosen. Genetic algorithms and very fast simulated reannealing: A comparison. **Journal of Mathematical Computer Modeling**, 16(11):87–100, 1992.

[55] Masao Iri and K. Kubota. Methods of fast automatic differentiation and applications. Research Memorandum RMI 87 – 02, Department of Mathematical Engineering and Information Physics, Faculty of Engineering, University of Tokyo, 1987.

[56] C. Jansson and O. Knüppel. A global minimization method: The multi-dimensional case. Technical Report 92-1, TU Hanburg-Harburg, January 1992.

[57] Mark E. Johnson. Simulated annealing. **American Journal of Mathematical and Management Sciences**, 8(3 & 4):205–206, 1988.

[58] A. H. G. Rinnooy Kan and G. T. Timmer. A stochastic approach to global optimization. In Richard H. Byrd and Robert B. Schnabel, editors, **Numerical Optimization**. SIAM, 1984.

[59] Rinnooy A.H.G. Kan and G.T. Timmer. Stochastic methods for global optimization. **American Journal of Mathematical and Management Sciences**, 4(1&2):7–40, 1984.

[60] R. Baker Kearfott. Interval Newton/generalized bisection when there are singularities near roots. **Annals of Operations Research**, 25:181–196, 1990.

[61] R. Baker Kearfott. An interval branch and bound algorithm for bound constrained optimization problems. **Journal of Global Optimization**, 2:259–280, 1992.

[62] E. L. Lawler and D. E. Wood. Branch and bound methods: A survey,. **Operations Research**, 14(3):699–719, 1966.

[63] David G. Luenberger. **Linear and Nonlinear Programming** 2^{nd} **ed**. Addison-Wesley Publishing Company, Reading MA, 1989.

[64] G.P. McCormick. Attempts to calculate global solutions of problems that may have local minima. In F.A. Lootsma, editor, **Numerical**

methods for non-linear optimization, pages 209–221. University of Dundee, Academic Press, 1972.

[65] G.P. McCormick. Computability of global solutions to factorable nonconvex programs: Part I - convex underestimating problems. **Mathematical Programming**, 10(2):147–175, 1976.

[66] Leo Michelotti. MXYZPTLK: A practical, user-friendly C++ implementation of differential algebra: User's guide. Technical Memorandum FN–535, Fermi National Accelerator Laboratory, Batavia, Ill., January 1990.

[67] R.H. Mladineo. An algorithm for finding the global maximum of a multimodal multivariate function. Presented at the 12th International Symposium on Mathematical Programming, Cambridge, Mass., 1985.

[68] R.E. Moore. Automatic error analysis in digital computation. Technical Report LMSD-48421, Lockheed Missiles and Space Division, Sunnyvale, CA, 1959.

[69] R.E. Moore. **Methods and applications of interval analysis**. SIAM, Philadelphia, 1979.

[70] R.E. Moore and H. Ratschek. Inclusion functions and global optimization ii. **Mathematical Programming**, 41:341–356, 1988.

[71] Katta G. Murty and Santosh N. Kabadi. Some NP-complete problems in quadratic and nonlinear programming. **Mathematical Programming**, page 117, 1987.

[72] Lê D. Muu and W. Oettli. Combined branch-and-bound and cutting plane methods for solving a class of nonlinear programming problems. **Journal of Global Optimization**, 3:377–391, 1993.

[73] Geroge L Nemhauser and Laurence A. Wolsey. **Integer and Combinatorial Optimization**. John Wiley & Sons, New York, New York, 1988.

[74] Arnold Neumaier. **Interval Methods for Systems of Equations**. Cambridge University Press, New York, New York, 1990.

[75] Panos M. Pardalos and Stephen A. Vavasis. Quadratic programming with one negative eigenvalue is NP-hard. **Journal of Global Optimization**, 1:15–22, 1991.

[76] A. T. Phillips and J. B. Rosen. Computational comparison of two methods for constrained global optimization. **Journal of Global Optimization**, 5:325–332, 1994.

[77] Thai Quynh Tao Phong, Dinh Pham, and Le Thi Hoai An. A method for solving D.C. programming problems. application to fuel mixture nonconvex optimization problem. **Journal of Global Optimization**, 6:87–105, 1995.

[78] S. A. Pijavskii. An algorithm for finding the absolute extremum of a function. **USSR Computational Mathematics and Physics**, 12:57–67, 1972.

[79] J. Pintér. A unified approach to globally convergent one-dimensional optimization algorithms. Technical Report IAMI-83.5, Istituto per la Applicazioni della Matematica e dell'Informatica, Milan, Italy, 1983.

[80] J. Pintér. Globally convergent methods for n-dimensional multiextremal optimization. **Optimization**, 17:1–16, 1986.

[81] M.J.D. Powell. An iterative method for finding stationary values of a function of several variables. **Computer Journal**, 5, 1964.

[82] H. Ratschek and J. Rokne. **Computer methods for the range of functions**. Halstead Press, New York, 1984.

[83] H. Ratschek and R.L. Voller. What can interval analysis do for global optimization. **Journal of Global Optimization**, 1:111–130, 1991.

[84] Dietmar Ratz. Box-splitting strategies for the interval gauss-Sidel step in a global optimization method. **Computing**, 53:1–16, 1994.

[85] Dietmar Ratz. On the selection of subdivision directions in interval branch-and-bound methods for global optimization. **Journal of Global Optimization**, pages 183–207, 1995.

[86] J. B. Rosen. Global minimization of a linearly constrained objective function by partition of feasible domain. **Mathematics of Operations Research**, 8(2):215–230, 1983.

[87] H.H. Rosenbrock. An automatic method for finding the greatest and least value of a function. **Computer Journal**, 3, 1969.

[88] Klaus Schittkowski, editor. **More Test Examples for Nonlinear Programming Codes**. Springer-Verlag, Berlin, 1987.

[89] Fabio Schoen. Stochastic techniques for global optimization: A survey of recent advances. **Journal of Global Optimization**, 1:207–228, 1991.

[90] Nonlinear programming - Frequently Asked Questions list. Posted monthly to Usenet newsgroup sci.op-research., September 1995. Also available at http://www.skypoint.com/subscribers/ashbury/nonlinear-programming-faq.html.

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

[92] R.M. Soland. An algorithm for separable nonconvex programming problems II: Nonconvex constraints. **Management Science**, 17(11):759–773, July 1971.

[93] B. Speelpenning. **Compiling Fast Partial Derivatives of Functions Given by Algorithms**. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana-Champaign, Ill., January 1980.

[94] N. V. Thoai and Hoang Tuy. Convergent algorithms for minimizing a concave function. **Mathematics of Operations Research**, 5(4):556–566, November 1980.

[95] N. V. Thoai and Hoang Tuy. Convergent algorithms for minimizing a concave function. **Mathematics of Operations Research**, 5(4):556–566, November 1980.

[96] G.T. Timmer. **Global optimization: A stochastic approach**. PhD thesis, Erasmus University Rotterdam, 1984.

[97] A. Törn and A. Žilinskas, editors. **Lecture Notes in Computer Science**. Springer-Verlag, Berlin, 1989.

[98] A.A. Torn. Cluster analysis using seed points and density determined hyperspheres with an application to global optimization. In **Proceedings of the third International Conference on Pattern Recognition, Coronado, California**, 1976.

[99] Hoang Tuy. Concave programming under linear constraints. **Soviet Mathematics**, pages 1437–1440, 1964.

[100] Hoang Tuy. Effect of the subdivision strategy on convergence and efficiency of some global optimization algorithms. **Journal of Global Optimization**, 1:23–36, 1991.

[101] Hoang Tuy. Normal conical algorithm for concave minimization over polytopes. **Mathematical Programming**, 51:229–246, 1991.

[102] Hoang Tuy and Reiner. Horst. Convergence and restart in branch-and-bound algorithms for global optimization. application to concave minimization and d.c. optimization problems. **Mathematical Programming**, 41:161–183, 1988.

[103] T. V. Tuy, Hoang Theiu and Ng. Q. Thai. A conical algorithm for globally minimizing a concave function over a closed convex set. **Mathematics of Operations Research**, 10(3):498–514, August 1985.

[104] V. Tuy, Hoang Khatchaturov and S. Utkin. A class of exhaustive cone splitting procedures in conical algorithms for concave minimization. **Optimization**, 18:791–807, 1987.

[105] V. Visweswaran and C. A. Floudas. Unconstrained and constrained global optimization of polynomial functions in one variable. **Journal of Global Optimization**, 2:73–99, 1992.

[106] V. Visweswaran and C.A. Floudas. A global optimization (GOP) for certain classes of nonconvex nlp's—II. application of theory and test

problems. **Computers& Chemical Engineering**, 14(12):1419–1434, 1990.

[107] E. R. Walster, G. W. Hansen and S. Sengupta. Test results for a global optimization algorithm. In Richard H. Byrd and Robert B. Schnabel, editors, **Numerical Optimization**. SIAM, 1984.

[108] R. E. Wengert. A simple automatic derivative evaluation program. **Comm. ACM**, 7(8):463–464, 1964.

[109] B. Zheng, Q Jiang and S. Zhuang. A method for finding the global extremum. **Acta Mathematicae Applagatea Sinica**, 2:164–174, 1978. In Chinese.

[110] Shen Zuhe and M.A. Wolfe. On interval enclosures using slope arithmetic. **Applied Mathematics and Computation**, 39(1):89–105, 1990.