NEW INTERVAL PARTITIONING ALGORITHMS FOR GLOBAL OPTIMIZATION PROBLEMS

NANYANG TECHNOLOGICAL UNIVERSITY

Chandra Sekhar Pedamallu

School of Mechanical and Aerospace Engineering

A thesis submitted to the Nanyang Technological University in fulfillment of the requirement for the degree of
Doctor of Philosophy

2007

Abstract

Global Optimization Problems are encountered in many scientific fields concerned with industrial applications such as kinematics, chemical process optimization, molecular design, and so on. When non-linear relationships among variables are defined by problem constraints resulting in non-convex feasible spaces the problem of identifying feasible solutions may become very hard. Consequently, finding the location of the global optimum in the problem is more difficult. The objective of this thesis is to develop a generic methodology, which can solve bound constrained optimization problems (*BCOP*), continuous constraint satisfaction problems (*CCSP*) and constrained optimization problems (*COP*). A new subdivision direction selection method is proposed in this research for these problems.

A variant for the proposed new subdivision selection method is also proposed for *BCOP*. The new variant considers the width of interval in addition to sub-expression bounds. The proposed two rules for *BCOP* target directly on the uncertainty degree of the objective function (with respect to the optimality). Reducing these uncertainties as such results in the reliable detection of sub-optimal boxes, thereby diminishing the number of boxes to be assessed.

The efficiency of the proposed variants is illustrated on well-known bound constrained test functions and compared with established subdivision direction selection methods from the literature.

For *CCSP* and *COP* a new adaptive search tree framework where nodes (boxes defining different variable domains) are explored using a restricted hybrid depth-first and best-first branching strategy is proposed. This hybrid approach is also used for activating local search in boxes with the aim of identifying different feasible

stationary points. The proposed search tree management approach improves the convergence of the interval partitioning method that is also supported by the new parallel subdivision direction selection rule used in selecting the variables to be partitioned in a given box.

The proposed rule targets directly the uncertainty degrees of constraints with respect to feasibility and the uncertainty degree of the objective function with respect to optimality. Reducing these uncertainties as such results in the early and reliable detection of infeasible and sub-optimal boxes, thereby diminishing the number of boxes to be assessed. Consequently, chances of identifying local stationary points during the early stages of the search increase.

For *CCSP*, the effectiveness of the proposed interval partitioning algorithm is compared with published results of established symbolic-numeric methods for solving *CCSP* on a number of state-of-the-art benchmarks. The effectiveness is also illustrated on several practical applications.

For *COP*, the effectiveness of the proposed interval partitioning algorithm is illustrated on several state-of-the-art benchmarks and also several practical applications and compared with professional commercial local and global solvers. Empirical results show that this approach is as good as available *COP* solvers.

Acknowledgments

I would like express my sincerest gratitude and appreciation to my supervisor, Associate Professor. Arun Kumar, for his invaluable advises and continued direction. I am particularly very grateful to Prof. Linet Özdamar, who supervised me for about two years, for her excellent guidance, support and encouragement in carrying out this research work. I am very thankful to her for introducing me to the field of Global Optimization and Interval Methods. Without their support and inspiration, this thesis would never be completed. Their patience, understanding and encouragement have enabled me to pursue my research effectively.

I wish to express my deepest gratitude and appreciation to Prof. Tibor Csendes for enabling the overseas research attachment (May 2005 to December 2005) with University of Szeged. He has always been there to help me in both technical and non-technical matters. His generous support and help enabled for successful completion of the overseas research attachment and the thesis. Without his support and inspiration, this thesis would never be completed.

I wish to thank Prof. Hermann Schichl (University of Vienna), Prof. Arnold Neumaier (University of Vienna) and Prof. Martine Ceberio (University of Texas) for their invaluable comments, suggestions and fruitful discussions on the thesis.

I wish to thank Prof. Andre Tits (Electrical Engineering and the Institute for Systems Research, University of Maryland, USA) for providing the source code of *CFSQP*.

I would also like to acknowledge the opportunity provided to me by the Division of Systems and Engineering Management, School of Mechanical and Aerospace Engineering, and Nanyang Technological University to carry out this research work.

Special thanks to the professors, staff, and colleagues in the Center for Supply Chain Management, Nanyang Technological University, Singapore for their assistance and kind cooperation.

I would like to thank Applied Informatics, University of Szeged, for providing the lab and other facilities to under take my research work during my overseas research attachment. I also wish to thank the system administrator (Mrs. Marianna and other system admins) and other staff in Applied Informatics for their kind help and support. I would like to thank Dr. Janos Posfai, and Dr. Kshitiz Chaudhary in editing this thesis.

Finally, I would thank my parents (Dr. P. Anjaneyulu and Anu Radha) and brothers (Dr. P. B. Srinivas, and Dr. P. Raghuveer) who have been extremely supportive during my research. And it would be great injustice if I didn't thank my friends in Singapore (Madhu, Damu, Manjula, Uma, Ramakrishna, Lakshmi, Wu Yong, and others) and Hungary (Tamas, Balázs, and others) for their kind support during my research.

Table of Contents

List of Figures

List of Tables

Notation

Through out the Thesis

x	Real variable.
$\mathbf{X}$	Cartesian product of intervals (also known as Box) .
X	Interval of a variable x.
$\overline{\mathrm{X}}$	Upper bound of variable x.
$\underline{\mathrm{X}}$	Upper bound of variable x.
z (.)	Real valued function z.
Z(.)	Inclusion function of z.
$\overline{Z}(\mathbf{X})$	Upper bound of the inclusion function Z over box $\mathbf{X}$.
$\underline{Z}(\mathbf{X})$	Lower bound of the inclusion function Z over box $\mathbf{X}$.
f (.)	Objective function.
g (.)	Inequality constraints.
h (.)	Equality constraints.
c (.)	Real continuous constraint.
C	Constraint System.
PF_Y	Degree of uncertainty with regard to optimality.
$PG^i_{\mathbf{Y}}$	Degree of uncertainty of an inequality constraint.
$PH^i_{\mathbf{Y}}$	Degree of uncertainty of an equality constraint.
$TINF_{\mathbf{Y}}$	Total degree of uncertainty.
$INF_{\mathbf{Y}}$	Total feasibility uncertainty degree of a box.
$\boldsymbol{A}$	Interval matrix.
$\mathbb{R}$	Set of reals.
$\mathbb{I}$	Set of Interval.
$\mathbb{I}^n$	n-dimensional interval (box).
w (X)	Width of an interval X.
Hull (X)	Hull of an interval X.
r (X)	Radius of an interval X.
m (X)	Midpoint of an interval X.
$\langle \mathrm{X} \rangle$	Mignitude of an interval X.

|X|	Magnitude of an interval X.
$\diamond$	Arithmetic operations (+, −, ×, ÷)
CLB	Current Lower Bound.
FE	Number of function calls.
GE	Number of Gradient calls.
x^*	Location of global optimum.
WLB	Working list / pending list of Boxes.
δ	Tolerance for final interval length.
ϕ	Empty set.
D^k	Parent sub-expression at tree level *k.*
L^{k+1} and R^{k+1}	Immediate Left and Right sub-expressions of D^k at tree level $k+1$.
STU	Standard time units.
D	Dimension of the problem.
NE	Number of nonlinear equations.
LE	Number of linear equations.
NIE	Number of nonlinear inequalities.
LIE	Number of linear inequalities.
IP	Interval partitioning algorithms

Chapter 1

Introduction

1.1. Background

Despite the advanced computer support we have at hand, optimization problems are still challenging for researchers working in computing sciences, mathematics and operations research fields. Limited success has been achieved in classifying and identifying global optima in nonlinear and discrete optimization systems. Often, the number of local optima in these problems is large; therefore, the standard nonlinear programming methods may fail to locate the global optimum. On the other hand, a numerical and exhaustive algorithm suffers from slow convergence when the feasible region defined by the given system of equations is not convex or covers a large space. Identifying the global optimum of a given system in a relatively fast and efficient way is very important both for engineering community and academics.

In engineering, economics and other scientific studies, quantitative decisions are frequently modeled by applying optimization concepts and tools. The decision maker or modeler typically wants to find the “absolutely best" decision, which corresponds to the maximum (or minimum) of a given objective function, while it satisfies a collection of feasibility constraints. The objective function expresses overall (modeled) system performance, such as profit, utility, loss, risk, or error. Constraints originate from physical, technical, economic or possibly some other considerations.

In the case of a possibly quite complex nonlinear system description, the associated decision model may – and frequently will - have multiple local optimal solutions. In most realistic cases, the number of such local solutions is not known *a priori*, and the

quality of local and global solutions may differ substantially. Sometimes the only information available is given computationally, i.e., only the function value is available, the derivative is either not available or is very expensive to compute. Therefore, these decision models can be very difficult, and standard optimization strategies may not be directly applicable to solve them. Hence, one needs to rely on other reliable global optimization (*GO*) concepts and techniques.

1.2. Categories of Global Optimization Problems

Global Optimization (*GO*) can be defined as the task of finding the absolutely best set of parameters to optimize an objective function. In general, there can be solutions that are locally optimal only but not globally optimal. Consequently, global optimization problems are typically quite difficult to solve analytically; in the context of combinatorial problems, they are often *NP-hard*. Global optimization problems fall within the broader class of Nonlinear Programming Problems (*NLP*).

A constrained optimization problem (*COP*) is defined by an objective function, $f(x_1,\ldots,x_n)$ to be maximized over a set of variables, $V=\{ x_1,\ldots,x_n \}$, with finite continuous domains for each variable x_i, $i = 1,\ldots,n$, the domain is defined as: $X_i = \{x_i : a \leq x_i \leq b\}$, where $a, b \in \mathbb{R}$ that are restricted by a set of constraints, $C=\{ c_1,\ldots,c_p \}$ where $p = k + m$ and k, m are number of inequality and equality constraints respectively.

Constraints in C are linear or nonlinear equations or inequalities that are represented as in equation (1.1):

$$\left.\begin{array}{l} g_i(x_1,\ldots, x_n) \leq 0, \forall i = 1,\ldots, k. \\ h_j(x_1,\ldots, x_n) = 0, \forall j = 1,\ldots, m. \end{array}\right\} \quad (1.1)$$

An optimal solution of a *COP* is an element (x^*) of the search space $\mathbf{X}$ ($\mathbf{X} = X_1 \times \ldots \times X_n$), that meets all the constraints, and whose objective function value, $f(x^*) \geq f(x)$ for all consistent elements $x \in \mathbf{X}$.

The generic *COP* model can be represented as:

$$\left.\begin{array}{l} \text{Objective function: } \textit{maximize } f(x) \\ \text{Subject to a set of constraints:} \\ g_i(x) \leq 0, \forall\ i = 1,\ldots, k \text{ (Inequality constraints)} \\ h_j(x) = 0, \forall\ j = 1,\ldots, m \text{ (Equality constraints)} \\ \text{Search region (box): } x \in \mathbf{X} \subseteq \mathbb{R}^n \end{array}\right\} \quad (1.2)$$

$f(x)$: $\mathbf{X} \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$, is the objective function, and can be convex or non-convex and linear or nonlinear functions.

The constraints can be of convex or non-convex type restricting over the domain $\mathbf{X}$.

The *COP* is a Bound Constrained Optimization Problem (*BCOP*)**,** if the constraints $g_i(x)$ and $h_j(x)$ are absent.

The *COP* is a Continuous Constraint Satisfaction Problem (*CCSP*)**,** if objective function $f(x)$ is absent.

The basic concepts of neighborhood, feasible point, local maximum, and global maximum are defined as follows (Wah and Wang 1999):

Definition 1.1. *$N(x)$, the **neighborhood** of point x in variable space X, is user defined set of points $\{x' \in \mathbf{X}\}$ such that $x \notin N(x)$ and that $x' \in N(x) \Leftrightarrow x \in N(x')$. Neighborhoods must be defined such that any point in the finite search space is reachable from any point through traversals of neighboring points.*

Definition 1.2. *Point* $x \in \mathbf{X}$ *is called a* ***feasible point*** *for equation* (*1.1*)*, if x satisfies all the constraints, i.e.,* $g_i(x) \leq 0 \; \forall \;\; i = 1,\ldots,k$ *and* $h_j(x) = 0 \; \forall \; j = 1,\ldots, m$.

Definition 1.3. *A* ***local maximum*** *for equation* (*1.2*)*, is defined as a point x′ such that* $f(x') \geq f(x) \; \forall \; x \in \mathbf{X} \cap N_\varepsilon(x')$. *Here* $\varepsilon > 0$ *and* $N_\varepsilon(x')$ *is an* ε*-neighborhood around x′ defined by* $N_\varepsilon(x') = \{x \mid \|x - x'\| < \varepsilon\}$.

Definition 1.4. *Point* $x^* \in \mathbf{X}$ is called a ***global maximum*** *for equation (1.2), iff a)* x^* *is a feasible point, and b) for every feasible point* $x \in \mathbf{X} \subseteq \mathbb{R}^n$, $f(x^*) \geq f(x)$.

Optimization techniques can be classified into two broad categories: local optimization and global optimization methods used in different fields of research. Efficient local optimization methods exist in the literature. It is harder to develop efficient Global Optimization methods.

1.3. Global Optimization Methods

The global optimization methods are broadly divided into deterministic and probabilistic global optimization methods.

a. Deterministic Global Optimization Methods:

Many deterministic methods have been developed in the past. Some of them apply deterministic heuristics, such as modifying the search trajectory in trajectory methods and adding penalties in penalty-based methods, to bring a search out of a local maximum. Other methods, such as branch-and-bound and interval methods partition a search space recursively into smaller subspaces and exclude regions containing no optimal solution. These methods do not work well when the search space is too large for deterministic methods to cover adequately.

Deterministic global optimization methods include covering methods, such as interval-based method, which are branch and bound method (Moore 1966, Ratschek

and Rokne 1988, Neumaier 1990, Hansen 1992) and generalized descent methods. Almost all the algorithms designed for solving constrained optimization problems and continuous constraint satisfaction problems are derived from the bound constrained optimization problems. However, there is a very limited amount of contribution made in solving the *COP*.

Cleary (1987) first proposed interval constraints for solving continuous constraint satisfaction problems. His approach associates propagation and search techniques developed in artificial intelligence and interval analysis based methods. However, Symbolic-Interval cooperation techniques are one of the most commonly used techniques in solving *CCSP*. The symbolic part in the cooperation deals only with the representation of the constraint expression and interval analysis computes the verified enclosures of the solution sets (Granvilliers et al. 2001).

A very limited amount of research is carried out in Symbolic-Interval cooperation approaches to handle non-polynomial *CCSP* problem (Granvilliers et al. 2001, Granvilliers and Benhamou 2006). This motivates in solving *CCSP* of non-polynomial type.

Covering methods (Evtushenko et al. 1992, Baritompa and Cutler 1994) are reliable since, to the extent they work, they have built-in guarantees of solution quality. However, they require some global properties of the optimization problem, such as Lipschitz condition. In the worst case, covering methods take an exponential amount of work. Many of the heuristic techniques used for searching global solutions can be adapted to or combined with branch and bound approach to take advantage of structural insights of specific applications.

Generalized descent methods (Anderson and Walsh 1986, Schaffler and Warsitz 1990) continue the search trajectory every time a local solution is found. Their problem is that as more local minima are found, the modified objective function becomes more difficult to minimize.

b. Probabilistic Global Optimization Methods:

Probabilistic global optimization methods rely on probability to make decisions. The simplest probabilistic algorithm uses restarts to bring a search out of a local maximum when little improvement can be made locally. Advanced methods use more elaborate techniques. Probabilistic global optimization methods are classified into clustering methods (Törn 1973, Boender et al. 1982, Törn and Viitanen 1992), random search methods such as single-start (Zabinsky and Smith 1992), multi-start (He and Polak 1993), random line search, adaptive random search, genetic algorithms, simulated annealing, and methods based on stochastic models for example, Bayesian methods.

Genetic algorithms (Goldberg 1989, Michalewicz 1994) and simulated annealing (Romeijn and Smith 1994) are the two popular stochastic global optimization methods. Genetic algorithms make use of analogies to biological evolution by allowing mutations and crossovers between candidates of good local optima in the hope to derive even better ones. However, simulated annealing takes its intuition from the fact that heating and slowly cooling (annealing) a piece of metal brings it into more uniformly crystalline state, which is believed to be the state where the free energy of bulk matter takes its global minimum.

More detailed review over the above solution approaches are available in Pardalos and Rosen (1987), Ratschek and Rokne (1988), Torn and Zilinskas (1989), Neumaier (1990), Floudas and Pardalos (1992), Horst and Pardalos (1995), Floudas and

Pardalos (1996), Grossmann (1996), Pinter (1996), Horst et al. (2000), Horst and Tuy (2003), Neumaier (2004), and so on.

1.4. Objective and Scope

The research objective of this thesis is to develop a reliable and generic approach that can deal with the following global optimization (non-convex and nonlinear) problems defined in equation (1.2):

- Bound constrained optimization problem (*BCOP*)
- Numeric / Continuous constraint satisfaction problem (*CCSP*)
- Constrained optimization problem (*COP*)

In *BCOP*, partitioning methods utilizing Interval Arithmetic are powerful techniques that produce reliable results. Subdivision direction selection is a major component of partitioning algorithms and it plays an important role in convergence speed. The subdivision rules proposed up to date is based on criteria such as the width of variable intervals (*Rule A* and *D*), or estimated function improvement by selected variables (gradient information such as *Rule B*, *C* and *E*). The performance of such rules is assessed extensively on standard test problems (Ratz and Csendes 1995, Csendes and Ratz 1996, Csendes and Ratz 1997, Csendes et al. 2000) resulting in the general conclusion that gradient based rules work much better.

Here, a new subdivision direction selection scheme is proposed that uses symbolic computing in interpreting interval arithmetic operations. We call this approach Interval Inference Rule (*IIR*). *IIR* targets the reduction of interval bounds of pending boxes directly by identifying the major impact variables and re-partitioning them in the next iteration. This approach speeds up the interval partitioning algorithms because it targets the pending status of sibling boxes produced. The proposed *IIR*

enables multi-section of two major impact variables at a time. Also, the new subdivision does not need any form of gradient information for the selection of subdivision directions. The efficiency of *IIR* is illustrated on well-known bound constrained test functions and compared with established subdivision direction selection methods from the literature.

In *CCSP* and *COP*, a cooperative solution methodology that integrates Interval Partitioning algorithms (*IP*) with a local search, Feasible Sequential Quadratic Programming (*FSQP*), is presented as a technique to enhance the solving of *CCSP* and *COP*. *FSQP* is invoked using a special search tree management system developed to increase search efficiency in finding feasible solutions and global optima of a *CCSP* and *COP* respectively.

In this framework, a new symbolic method is introduced for selecting the subdivision directions that targets immediate reduction of the uncertainty related to constraint infeasibility in child boxes. This subdivision method is compared against two previously established partitioning rules (also parallelized in a similar manner) used in the interval literature and shown to improve the efficiency of *IP*. Further, the proposed tree management system is compared with tree management approaches that are classically used in *IP*. The whole method is compared with the published results of established symbolic-numeric methods and other established software for solving *CCSP* and *COP* on a number of state-of-the-art benchmarks respectively.

The scope of this research is to

- ✓ develop a general Interval Partitioning Approach with special tree management techniques.
- ✓ integrate a new and efficient subdivision selection rule (*IIR*) that transforms

function expressions into workable trees (binary tree) on which sub-expression intervals can be numerically propagated to identify major impact variables for re-partitioning.

1.5. Outline of the Thesis

The organization of thesis is follows: Chapter 2 outlines the detailed literature related to this research, overview on Interval Arithmetic and the basic notation, and overview and motivation behind selecting the Feasible Sequential Quadratic Programming (*FSQP*) as the local search method. Chapter 3 briefly reviews the existing tree management systems and the new proposed tree management used in solving *CCSP* and *COP*. Chapter 4 first introduces the different subdivision direction selection strategies. Then the details on the new subdivision selection strategy (Interval Inference Rule (*IIR*)) and its convergence proof are presented**.** The performance of the *IIR* over the existing approaches and software is illustrated in Chapter 5 with numerical experiments. Some of the possible application problems and its numerical results are presented in Chapter 6 in brief. Chapter 7 summarizes the work done in this research and presents some possible extensions. The Appendices A, and B present the detailed computational results for *CCSP*, and complete list of *COP* benchmarks, respectively.

Chapter 2

Literature Review

General nonlinear optimization problems are difficult to solve due to the large number of local maxima in the search space. Good local maxima are difficult to be found by the local search methods because they stop at every local maximum. Thus, to obtain globally optimal solutions, global optimization techniques have been developed to escape from local maxima once the search gets there, and continue the search process further.

2.1. Pervious Work on Global Optimization

Due to the importance of optimal solutions for engineering, economics and social sciences applications, numerous optimization methods have been developed. In the recent decades, as computers have become more powerful, numerical optimization algorithms have been developed for many applications.

Solution methods for nonlinear optimization problems can be classified into local and global optimization methods. Local optimization methods such as gradient-descent and Newton's methods use local information (gradient or Hessian) to perform descents and converge to a local optimum. They can find local optima efficiently and work best in uni-modal problems. Global methods, in contrast, employ heuristic strategies to look for global optima and do not stop after finding a local optimum (Pardalos 1993). A taxonomy on global optimization methods can be found in Pardalos and Rosen (1987), Törn and Zilinskas (1989), Floudas and Pardalos (1992), Hansen (1992), Horst and Tuy (2003) and so on. It is noted that the gradients and Hessians can be used in both local and global methods.

2.1.1. Local Search Methods

Local optimization methods can be broadly classified into Zero-order methods, First-order methods, and Second-order methods based on the derivative information used during search (Shang 1997).

Zero-order methods do not use derivatives of objective functions during optimization. They include simplex search method, the Hooke and Jeeves methods, the Rosenbrock method, and conjugate direction method (Powell 1964, Nelder and Mead 1965, Chazan and Miranker 1970, Dennis Jr. and Torczon 1991, Lewis et al. 2000).

First-order methods use first-order derivatives of the objective function during the search. They include the gradient-descent method, the discrete Newton's method, the quasi-Newton methods, and conjugate gradient methods. The gradient-descent method performs a linear search along the direction of the negative gradient of the minimized function (Arminjo 1966, Wolfe 1967, Sturua and Zavriev 1991, Baldi 1995). The discrete Newton's method approximates the Hessian matrix by the finite difference of the gradient. Quasi-Newton methods approximate the curvature of the nonlinear function using information of the function and its gradient only, and avoid the explicit evaluation of the Hessian matrix (Broyden 1972, Bazaraa et al. 1993). Conjugate gradient methods combine the current gradient with the gradients of previous iterations and search direction to form the new search direction. They generate search directions without storing a matrix (Hestenes 1980, Kinsella 1992, Bazaraa et al. 1993).

Second-order methods make use of second-order derivatives. They include Newton's methods, Levenberg-Marquardt's method, and trust region methods (Dennis and Schnabel 1983, Bazaraa et al. 1993). In Newton's method, the inverse of Hessian

matrix multiplies the gradient, and a suitable search direction is found based on a quadratic approximation of the function. Newton's method converges quadratically if the initial point is close to a local optimum. Levenberg-Marquardt's method and trust region methods are modifications of Newton's method. However, line search or trust region algorithms converge when their starting point is not close to an optimum. Line search and trust region techniques are suitable if the number of variables is not too large. Truncated Newton's methods are more suitable for problems with a large number of variables. They use iterative techniques to obtain a direction in a line search or a step in a trust region method. The iteration is stopped when a termination criterion is satisfied.

The sequential quadratic programming (*SQP*) algorithm is a generalization of Newton's method for bound constrained optimization in that it finds a step away from the current point by minimizing a quadratic model of the problem (Shanno and Phua 1989, Zhou and Tits 1996, Lawrence et al. 1997, Murray 1997, Lawrence and Tits 2001).

Local search methods converge to local optima. For some applications, local optima may be good enough, particularly when the user provides a good starting point for local optimization algorithms. However, for many applications, globally optimal or near-optimal solutions are desired.

In a nonlinear optimization, objective functions are multimodal with many local optima. Local search methods converge to local optima close to the initial points. Therefore, the solution quality depends heavily on the initial point selected. When the objective function is highly nonlinear, local search methods may return solutions much worse than the global optima when starting from a random point.

Some of the software codes developed using local search techniques are as follows:

GAMS / Conopt

Algorithms:

Steepest Descent, Quasi-Newton, Sequential Linear Programming, Sequential Quadratic Programming using the new second order information. The SQP sub-method uses Reduced Hessians (when there are few superbasics) or Conjugate Gradients (when there are many superbasics) (Drud 1996).

GAMS / Minos

Algorithms:

GAMS / *Minos* solves linear programs using a reliable implementation of primal simplex method (Dantzig 1963). However, it solves nonlinear programs using reduced-gradient algorithm (Wolfe 1962, Wolfe 1967) combined with a quasi-Newton algorithm (Murtagh and Saunders 1978).

GAMS/ Snopt

Algorithms:

GAMS / Snopt applies primal simplex method (Dantzig 1963) for linear programs. For both linearly and nonlinearly constrained problems, *GAMS / Snopt* applies a sparse sequential quadratic programming (*SQP*) method (Gill et al. 1997), using limited-memory quasi-Newton approximations to the Hessian of the Lagrangian.

To overcome the deficiencies in local search methods, global optimization methods have been developed with global search mechanisms. Global search methods use local search methods to determine local maxima, and focus on bringing the search out of a local maximum once it gets there.

2.1.2. Global Search Methods

Global search algorithms are abundant in global optimization literature, for example, findings of Pardalos and Rosen (1987), Horst and Pardalos (1995), Pinter (1996), Horst et al. (2000), and Özdamar and Demirhan (2001) are important for reviews and comparisons. These algorithms can be classified as deterministic or probabilistic algorithms. Probabilistic methods evaluate the objective function at randomly sampled points from the solution space. Deterministic methods, on the other hand, involve no element of randomness.

Alternatively, global optimization algorithms can also be classified as reliable and unreliable. Reliable methods guarantee solution quality while unreliable methods do not. Probabilistic methods, including simulated annealing, clustering, and random searching fall into the unreliable category. However, Unreliable methods usually have the strength of efficiency and better performance in solving large-scale problems.

Deterministic methods can be further classified into covering methods, and generalized descent methods. Probabilistic methods can also be further divided into clustering methods, random search methods, and methods based on stochastic models.

1. Deterministic Methods

Numerous deterministic methods have been developed in the past. Some of them apply deterministic heuristics, such as modifying the search trajectory in trajectory methods and adding penalties in penalty-based methods, to bring a search out of a local maximum. Other methods, like branch-and-bound and interval methods, partition a search space recursively into smaller subspaces and exclude regions containing no optimal solution. These methods do not work well when the search space is too large for deterministic methods to cover adequately.

a. Covering Methods

Covering methods detect sub regions not containing the global maximum, and exclude them from further consideration. Covering methods provide guarantee of solution quality, and approximate the global solution by iteratively using tighter bounds (Evtushenko et al. 1992, Hansen 1992, Moore et al. 1992, Horst and Tuy 2003). Obtaining a solution with guaranteed accuracy implies an exhaustive search of the solution space for the global maximum. Thus, these methods can be computationally expensive; with a computation time that increases dramatically as the problem size increases (Baritompa 1993, Baritompa and Cutler 1994).

Branch and bound methods is one of the best examples for covering method. They evaluate upper bounds on the objective function of subspaces. They allow an assessment of the quality of the local optima obtained. Combining with computationally verifiable sufficient conditions for global optimality, they allow one to actually prove global optimality of the best solution obtained.

Branch and bound methods can be further classified into algorithms based on interval methods, algorithms based on certain prior assumptions on functions, such as Lipschitz functions and other methods.

Interval Methods

Interval Partitioning Algorithms (*IPA*) use interval arithmetic (Moore 1966) to produce reliable results for constrained and bound constrained optimization (Hansen 1992, Ratschek and Rokne 1995). Due to their reliability, interval applications take place in a wide scope of scientific fields (Kearfott and Kreinovich 1996). In bound constrained global optimization problems, *IPA* subdivides the given domain into smaller subspaces (boxes) that are assessed according to their function range

calculated by using an approximating inclusion function. Based on the function range bounds and a known best solution that is updated during the search, some subspaces are deleted reliably, because they cannot hold the global optimum solution (Pinter 1992, Hammer et al. 1993). Subdivision continues in remaining boxes so that the location of the global optimum solution can be enclosed within a small box of a given tolerance. The final report contains all such boxes in the given function domain.

Convergence rate of *IPA* depends on the use of accelerating devices such as monotonicity and concavity tests that help in discarding boxes (Ratschek and Rokne 1988, Ratschek and Rokne 1995) and on the selection of subdivision direction (variable whose domain is to be re-partitioned) (Moore 1966, Neumaier 1990, Hansen 1992, Ratz and Csendes 1995, Berner 1996, Csendes and Ratz 1996, Csendes and Ratz 1997, Csendes et al. 2000). In *IPA*, the latter issue has a major impact on convergence rate because reducing the domain size of a specific variable might enhance the reduction in the overestimated function range of the sibling boxes to a significant degree. Thereby, boxes that cannot be discarded due to their promising overestimated upper bounds may become disposable in a few re-partitioning iterations with a good subdivision direction selection strategy.

Subdivision rules proposed up to date are based on criteria such as the width of variable intervals, for example, *Rule A* (Moore 1966, Ratschek and Rokne 1988, Ratz and Csendes 1995, Berner 1996, Csendes and Ratz 1996, Ratz 1996, Csendes and Ratz 1997, Csendes et al. 2000) and *Rule D* (Hammer et al. 1993, Ratz and Csendes 1995, Berner 1996, Csendes and Ratz 1996, Ratz 1996, Csendes and Ratz 1997, Csendes et al. 2000), or estimated function improvement by selected variables that is gradient information (*Rule B* (Hansen 1992, Ratz and Csendes 1995, Csendes and Ratz 1996, Ratz 1996, Berner 1996, Csendes and Ratz 1997, Csendes et al. 2000),

Rule C (Ratz and Csendes 1995, Berner 1996, Csendes and Ratz 1996, Ratz 1996, Csendes and Ratz 1997, Csendes et al. 2000) and *Rule E* (Ratz and Csendes 1995, Berner 1996, Csendes and Ratz 1996, Ratz 1996, Csendes and Ratz 1997, Csendes et al. 2000). The performance of such rules is assessed extensively on standard test problems (Ratz and Csendes 1995, Csendes and Ratz 1996, Csendes and Ratz 1997, Csendes et al. 2000) resulting in the general conclusion that gradient based rules work much better.

In Berner (1996), these rules are converted into parallel multi-section rules by taking the first k number of variables from a list of variables sorted according to the rule (called *k-best* strategy here). Multi-section (subdivision of some variables in parallel) and multi-splitting (subdivision of a single variable's width into $s > 2$ pieces) approaches are proposed in Csallner et al. (2000a, 2000b). The latter studies investigate the efficiency related to specific values of s with regard to each subdivision rule. Casado et al. (2000) propose multi-section / multi-splitting hybrids by subdividing intervals of all variables into two or more pieces (s^n) in parallel. They propose a parametric method that involves the comparison of a box assessment criterion with given constants used in deciding which hybrid parallel scheme should be used for a given box. In Casado et al. (2000) the authors use the box assessment criterion as a box selection rule and utilize multi-section subdivision rules based on *k-best* strategy found in Berner (1996).

Further, the direction selection rules are classified into priori direction selection rules and posteriori selection rules. The rules based width of variable intervals and gradient information is classified as the priori selection rules (Csendes et al. 2000). The rules developed for understanding the functioning of these subdivision direction selections

are called as posteriori selection rules such as Rule 1, Rule 2, Rule 3 and Rule 4 (Csendes et al. 2000).

Ratz (1994, 1997) investigates the impact of gap-treating and box-splitting techniques on subdivision direction selection rules. The box splitting techniques can be divided into Brute force interval splitting, Selective interval splitting, Selective interval splitting with midpoint evaluation, Exploitation of single parameter occurrence, Exploitation of partial N-monotonicity, and Generalized Splitting Algorithm (*GSA*) (for an overview, see Lüthi and Lladó 2003)**.**

Accelerating devices are introduced to speedup the convergence of *IPA*. The midpoint test (Ratz 1992, Hammer et al. 1993), which evaluates the middle point of a box, is used to discard the boxes whose upper bounds are less than the midpoint's function value. The monotonicity test assumes that the given function is continuously differentiable (Ratschek and Rokne 1988, Vaidyanathan and Halwagi 1994, Ratschek and Ratschek 1995, Markót 2003). The other accelerating device includes non-convexity test and other discarding tests (Hansen 1980, Ratschek and Rokne 1988, Vaidyanathan et al. 1994, Markót 2000, Fernandez and Pelegrin 2001, Markót et al. 2006). More detailed review over the above accelerating devices is available in Ratschek and Rokne (1988), Vaidyanathan and Halwagi (1994), Ratschek and Ratschek (1995), Markót (2003). The box can be abdicated when this test is satisfied. However, this case is only true when dealing with bound constrained optimization. For constrained optimization, this test alone is not sufficient to discard a box. Moreover, the assumption of '*continuously differentiable*' cannot always be met in practice.

Convergence rate of *IPA* also depends on the order of boxes to be processed (Ratschek and Rokne 1988, Csendes and Pinter 1993, Csallner and Csendes 1996,

Csendes 2001). In interval methods, the order of boxes to be processed plays an important role; there exist different strategies for selecting the next box to be subdivided. Some strategies have been used in interval methods, such as the strategy to select the best upper bound, i.e., Moore-Skelboe rule (Skelboe 1974, Ratschek and Rokne 1988), the oldest interval from the list, i.e., Hansen rule (Hansen and Sengupta 1980, Hansen 1992) and so on. Moore-Skelboe rule ensures a quicker algorithm than Hansen rule. Jansson (1994) studies the combination of oldest first and best first selection strategy. Berner (1996) studied the influence of different strategies for selecting the next box for subdivision in *IPA*. Csendes (2001), and Casado et al. (2001a, 2001b) propose some more heuristic selection strategies, which utilize the information of global optima in solving bound constrained optimization problems.

Termination criterion also plays an important role in *IPA* to obtain solutions, which are close to the actual solutions. Kearfott and Walster (2000) introduce new termination criteria, i.e., thickness stopping criterion, which can be used for global optimization algorithms using interval analysis. The other stopping criterion is a heuristic domain and range stopping criteria, which is used to determine the accuracy tolerances (Moore 1966, Neumaier 1990, Hansen 1992, Ratscheck and Rokne 1995).

Theoretically, *IPA* has no difficulties in dealing with the *COP*; however, interval research on the *COP* is relatively scarce when compared with bound constrained optimization. Robinson (1973) uses interval arithmetic only to obtain bounds for the solution of the *COP*, but does not attempt to find the global optimum. Hansen and Sengupta (1980) first use *IPA* to solve the inequality *COP*. A detailed discussion on interval techniques for the general *COP* with both inequality and equality constraints is provided in Ratschek and Rokne (1988) and Hansen (1992), and some numerical

results using these techniques have been published later (Wolfe 1994, Kearfott 1996a).

Conn et al. (1994) transform inequality constraints into a combination of equality constraints and bound constraints and combine the latter with a procedure for handling bound constraints with reduced gradients. Computational examination of feasibility verification and the issue of obtaining rigorous upper bounds are discussed in Kearfott (1996d) where the interval Newton method is used for this purpose. In Hansen and Walster (1993), interval Newton methods are applied to the Fritz John equations that are used to reduce the size of sub-spaces in the search domain without bisection or other tessellation. Experiments that compare methods of handling bound constraints and methods for normalizing Lagrange multipliers are conducted in Kearfott (1996b). Dallwig et al. (1997) propose software for solving bound constrained optimization and the *COP* (so called GLOPT). GLOPT uses a branch and bound technique to split the problem recursively into subproblems that are either eliminated or reduced in size. The authors also propose a new reduction technique for boxes and novel techniques for generating feasible points. More recently, Kearfott (2003) presents the GlobSol, which is *IP* software that is capable of solving bound constrained optimization problems and the *COP*.

Markót (2003) developed a new *IP* for solving the *COP* with inequalities where new adaptive multi-section rules and a box selection criterion are presented (Markót et al. 2006). Kearfott (2006) provides a discussion and empirical comparisons of linear relaxations and alternate techniques in validated deterministic global optimization. Empirical results show that linear relaxations are of significant value in validated global optimization. Finally, in order to eliminate the subregion of the search spaces, Kearfott (2005) proposes a simplified and improved technique for validation of

feasible points in boxes, based on orthogonal decomposition of the normal space to the constraints. In the *COP* with inequalities, a point, rather than a region, can be used, and for the *COP* with both equalities and inequalities, the region lies in a smaller-dimensional subspace, giving rise to sharper upper bounds. More detailed review over the methods for constrained optimization is available in Ratschek and Rokne (1988), Hansen (1992), Ratschek and Rokne (1995), Kearfott (1996c) and so on.

For Continuous Constraint Satisfaction Problems:

CCSP's are solved by discarding inconsistent elements of the search space (this technique is known as a filtering technique). One of the main difficulties of filtering is that parts of the search space can be discarded only by proving that they do not contain any feasible solution. In particular, unlike when solving constrained optimization problems, filtering for *CCSP* cannot take advantage of a bound on the objective (*Branch* and *Bound* approach) upon a search space and discard it using this bound. It is hard to tackle the general nonlinear *CCSP* with computer algebra systems. In general, numeric algorithms cannot guarantee ***completeness*** (some solutions may be missed) and ***reliability*** of the solution set (the search might result with an infeasible response, despite the fact that a feasible solution exists). Neumaier et al. (2005) compare the solver performance in terms of reliability, efficiency, and so on among major complete/global and incomplete/local solvers using an extensive set of constrained optimization problems and *CCSP*. As far as solving *CCSP* is concerned, the reliability of interval-symbolic solver *ICOS* (Lebbah 2003) is praised, however, with a note on its slow convergence. Results on these benchmarks are reported for the identification of a first feasible solution, rather than all feasible solutions.

Completeness and reliability can be achieved by using interval methods. Intervals were first used (Moore 1966) to take rounding errors into account, guaranteeing thus reliability. In addition to this, interval-based methods were proved to be complete (Neumaier 1990, Hansen 1992, Ratschek and Rokne 1995). A drawback of interval methods is the dependence of a function interval on the syntactical form of the constraints /expressions. The latter affects the performance of the interval-based methods. This problem of intervals is known as the dependency problem. Fortunately enough, in some specific cases, it was shown that the input forms could be converted into more manageable/solvable expressions by different methods (Buchberger 1985, Rump 1992, Granvilliers 1998, Ceberio and Granvilliers 2002).

Interval techniques for the *CCSP* are basically Branch and Prune techniques where branching consists in splitting the search space into smaller boxes, and pruning in reducing the variables' domains. Splitting consists in bisecting the domains of selected variable(s) in a given search space (box), which results in child boxes. Variable selection is made according to different heuristics, such as largest width first (*Rule A*) or largest rate of change (product of absolute value of the corresponding Jacobian element and width of the variable- *Smear* rule by Kearfott and Manuel 1990).

Precision test checks out the precision of the current box. The test succeeds if the precision of the current box is smaller than or equal to an infinitesimally number ε ($\varepsilon > 0.0$). The splitting step divides the current box along one dimension. The splitting step can be performed using *Rule A*, *Rule B*, *Rule C*, *Rule D*, *Smear Rule* and so on given in the above section.

In the Branch and Prune approach, pruning test (or *filtering*) can be carried out in three basic ways.

The first method is the simplest one and involves *interval evaluation* of each constraint. If the interval of any constraint does not contain a root (namely, the zero), the box is discarded. Convergence of this method is slow due to overestimation of inclusion function ranges that leads to repetitive bisection of boxes.

The second method is the classical global pruning approach, the interval Newton method (Moore 1966, Hansen 1992). Interval Newton is based on the iterative Newton step involving the Jacobian matrix that represents interval rate of change of all constraints in the system. Hence, it may be called a *global filtering* method. Newton steps may result in variable domains that do not intersect with the box's domains (infeasible search space-to discard), it may narrow variable domains in a given box, or, it might fail in case of multiple roots. Convergence rate to a sufficiently small enclosure of the root is quadratic if a feasible solution exists in the given box.

The third and most important category of pruning approaches is that of *local consistency* methods, which aim to narrow variable domains by using their functional relationships in constraints, taking one constraint and one variable at a time. Such methods remove parts of the search space of variables that are inconsistent with the domains of other variables taking place in the same constraint. Once the domains of all variables in a given constraint are screened for consistency, reduced domains are substituted into other constraints sharing the variables whose domains have just been reduced. These steps result in a *constraint propagation* algorithm (Jaulin et al. 2001). The sequence in which constraints are handled in constraint propagation plays an important part in the efficiency of the method (L'homme et al. 1998). Two major types of consistency techniques exist, *hull consistency* and *box consistency*. Hull

consistency (Cleary 1987, Benhamou et al. 1994, Benhamou and Older 1997) is basically constraint inversion that uses relational interval arithmetic. Hull consistency is applicable to simple constraints due to difficulties arising during the inversion of complex expressions and also due to the *dependency* problem that takes place in the case of multiple occurrences of the same variable in a constraint expression. The dependency problem is partially eliminated by the *box consistency* technique (Benhamou et al. 1994, Van-Hentenryck et al. 1997a) where consistency of gradually expanding outermost sub-domains (starting from lower and upper bounds of domains) is identified by an iterative method that checks constraint feasibility by interval evaluation. When constraint propagation does not reduce a variable interval substantially, the branching module bisects its domain and inserts the two new sub-spaces (boxes) into the list of boxes waiting to be assessed for feasibility. Hull and box consistency methods work in conjunction with each other and with interval Newton method to improve overall efficiency. A comparison of consistency techniques and cooperative strategies are found in Benhamou et al. (1999), and Granvilliers (2001). The other consistency technique is higher order local consistency (Freuder 1978, Sam-Haroud and Faltings 1996, Lebbah and Lhomme 2002) deriving from k-consistency such as 3B Consistency, kB-Consistency (Lhomme 1993), and box (2)-Consistency (Puget and Van Hentenryck 1998).

An extensive discussion on how symbolic-interval cooperation can be carried out by appropriate constraint partitioning is given in Granvilliers et al. (2001). Most of the research up to date is focused on constraint representation such as Horner Rule, Factorization, Gröbner basis, and so on (Ceberio and Granvilliers 2002). These symbolic-interval cooperation techniques use interval analysis for computing verified enclosures of solution sets. Thus, research carried out till date is more promising to

the polynomial functions then the non-polynomial functions. Developing a generic approach for solving polynomial and non-polynomial functions is a worthwhile contribution to this field of research**.**

Schichl and Neumaier (2005) propose a new technique for global optimization, which is a combination of interval analysis on directed acyclic graph and constraint propagation. Vu et al. (2004) propose a new simple algorithm, which coordinates constraint propagation and exhaustive search for solving numerical constraint satisfaction problems (Vu et al. 2003).

ALIAS and *ICOS* are software, which is capable of finding all feasible solutions for a given *CCSP*. *COPRIN* project web page publishes results obtained through the usage of *ALIAS*, a comprehensive set of libraries that include 2B/3B box and hull consistency variants, linearization (Yamamura et al. 1998), interval Newton, and unicity operators, and numerical and interval root approximations for univariate polynomials (http://www-sop.inria.fr/coprin/logiciels/ALIAS/ALIAS-C++/ALIAS-C++.html). *ALIAS* is an advanced C++ library of symbolic-interval algorithms that deal with *CCSP's*.

Several interval based software have been developed, and most of them being referenced on the interval computations webpage (http://www.cs.utep.edu/interval-comp/intsoft.html). Interval-based constraint satisfaction techniques have been implemented in several constraint logic programming (*CLP*) systems, such as Eclipse (Meier and Schimpf 1993), Oz (Smolka 1995), Mozart (Mozart Consortium 1999) and Prolog IV (PrologIA 1996). Constraints are embedded into *CLP* programs, using Horn clauses. The operational semantics based on term unification and constraints solving techniques.

Pascal Van Hentenryck and his colleagues developed *Numerica* (Van Hentenryck et al. 1997a), which is the integration of constraint satisfaction techniques and interval methods. However, *Numerica* is not currently available, but some of their constraints solving functionalities have been implemented in the commercial object-oriented library *ILOG Solver* (ILOG 2001). *RealPaver* (Granvilliers and Benhamou 2006) is a C++ library for solving nonlinear systems of equations, implementing a fine-grained algorithm based on constraint satisfaction techniques. On the *COCONUT* web page, an extensive testing has been carried out for box-constrained and constrained optimization problems in addition to *CCSP's*. The solvers that are compared on the *CCSP* problems include *ALIAS*, *COCONUT, ICOS* (Lebbah 2003) and *QUAD* (for two problems reported).

More detailed review over the above solution approaches for *CCSP* is also available in Schichl (2003), and Vu (2005).

Some of the software codes or platforms developed using local search techniques are as follows:

1. COCONUT (Schichl 2003, Schichl 2004)

The *COCONUT* project (Schichl 2003, Schichl 2004) aims at the integration of existing approaches to continuous global optimization and constraint satisfaction. The solution algorithm is an advanced branch-and-bound scheme which proceeds by working on the search graph, a directed acyclic graph of search nodes (Schichl and Neumaier 2005), each representing an optimization problem, or a model. The several state of the art techniques already provided are *Donlp2* (Spellucci 2002), Box covering solver (*BCS*), *STOP* (a heuristic starting point generator), Karush-John Condition generator using symbolic differentiation, Point Verifier for verifying

solution points, Exclusion Box generator, calculating an exclusion region around local optima (Schichl and Neumaier 2004), Interval constraint propagation (Petrov and Benhamou 2002), Linear Relaxation, *CPLEX* (a wrapper for the state of the art commercial linear programming solver by *ILOG* (http://www.ilog.com/products/cplex/)), Basic Splitter, and Convexity detection for simple convexity analysis.

2. Realpaver (Granvilliers and Benhamou 2006**)**

The constraint solving engine of RealPaver implements a branch and prune algorithm. Given a *CCSP***,** a set of boxes that contain all the solutions of the *CCSP* is computed through splitting and reducing each box. The reduction eliminates inconsistent values from domains by means consistency techniques. The splitting step generates sub-boxes in order to separate the solutions.

3. Numerica (Van Hentenryck et al. 1997a**)**

Branch and bound is algorithm for constrained optimization (with mathematically rigorous results). This code (no longer available) was based on branching and box reduction using interval analysis and constraint satisfaction techniques. The box reduction and interval analysis algorithms of *Numerica* are now available in ILOG Solver.

4. GlobSol (http://interval.louisiana.edu/GlobSol/download_GlobSol.html, Kearfott 2003**)**

This is a Branch and bound code for global optimization with general factorable constraints, with rigorously guaranteed results (even round off is accounted for correctly). *GlobSol* is based on branching and box reduction using interval analysis to verify that a global maximizer cannot be lost.

5. Globopt (Dallwig et al. 1997**)**

Globopt is a Fortran77 program for global minimization of a block separable objective function subject to bound constraints and block-separable constraints. *Globopt* uses a branch and bound technique to split the problem recursively into subproblems that are either eliminated or reduced in their size.

6. ALIAS (http://www-sop.inria.fr/coprin/)

ALIAS is a software based on interval analysis and can be used for almost any system as long as it is composed of classical mathematical operators. Some algorithms may be used only for systems with specific structure such as algebraic, linear, distance, systems and so on. *ALIAS* may also deal with functions that involve determinants of matrices, without having to expand the determinants. This is mainly developed for solving constraint satisfaction problems.

7. ICOS (http://www-sop.inria.fr/coprin/ylebbah/icos/index.html, Lebbah et al. 2003**)**

ICOS (Interval COnstraints Solver) is a software package for solving nonlinear and continuous constraints. It is based on constraint programming and interval analysis techniques. This is mainly developed for solving constraint satisfaction problems and can find all the solutions for a given *CCSP* model.

8. **Unicalc** (http://www.rriai.org.ru/UniCalc/)

This is a solver based on interval constraint propagation. It allows tackling nonlinear algebraic systems with real and/or integer variables.

9. **ParaGlobSol** (http://happy.dt.uh.edu/~sun/ParaGlobSol.html)

Parallel/distributed implementations of the interval global optimization Fortran 90 package *GlobSol*, which solve global optimization problems with the interval branch-and-bound algorithm together with interval Newton/generalized bisection method.

Various software packages developed to support interval arithmetic are:

1. **Profil / Bias** (Knuppel 1994)

http://www.ti3.tu-harburg.de/Software/PROFILEnglisch.html

PROFIL (Programmer's Runtime Optimized Fast Interval Library) is a C++ class library supporting the most commonly needed interval and real operations in a user friendly way. *PROFIL* is based on Basic Interval Arithmetic Subroutines (*BIAS*). The development of *BIAS* was guided by the ideas of Basic Linear Algebra Subprograms (*BLAS*), to provide an interface for basic vector and matrix operations with specific and fast implementations on various machines, the latter is frequently provided by the manufacturers. The idea of *BIAS* is to give such an interface for interval operations with the objective, very efficient use of the underlying hardware, portability, independency of a specific interval representation.

2. **C-XSC** (Klatte et al. 1993)

http://www.rz.uni-karlsruhe.de/~iam/html/language/cxsc/cxsc.html

C-XSC is a tool for the development of numerical algorithms delivering highly accurate and automatically verified results. It provides a large number of predefined numerical data types and operators. These types are implemented as C++ classes. Thus, *C-XSC* allows high-level programming of numerical applications in C and C++.

Lipschitz Methods

Lipschitz methods solve global optimization problems in which the objective function and constraints are given explicitly and have bound slopes. The algorithm partitions feasible space into sub-spaces and conducts assessment for re-partitioning relying on assumed knowledge about the rate of change of the function, or the so-called Lipschitz constant (Gourdin et al. 1994, Hansen et al. 1995, Horst and Tuy 2003, Pinter 1988). In Lipschitz approach, functions are assumed to be continuous and smooth with finite slopes around stationary points. Lipschitzian methods guarantee convergence to the global optimum only if Lipschitz constant utilized in approximating the function is not underestimated. Heuristic estimations of the Lipschitz generation are found in Strongin (1978), Meewella and Mayne (1989), and Baritompa (1993) whereas exact approaches require the generation of a refined mesh to obtain an appropriate Lipschitz constant. It is known that a high degree of overestimation in Lipschitz constant results in a very slow convergence rate. Obviously, function properties should be known to utilize Lipschitzian approaches, and therefore, the problem cannot be considered as black box.

However, there exist other Lipschitzian approaches that eliminate the necessity of specifying the Lipschitz constant whether or not it is estimated or calculated. The latter are classified as black box optimization techniques. An example is *DIRECT* (Jones et al. 1993) where the Lipschitz constant is taken as a weighting parameter that balances global and local search. Parallel partitioning is conducted on boxes that are non-dominated with respect to the two criteria, the first being the box size (representing unexplored areas – a global search feature) and the second being the box value (representing function fitness – a local search feature). Efforts are made to enhance the computational complexity of *DIRECT* by using massive parallelism

(Watson and Baker 2000). He et al. (2002) propose modifications that involve termination and box selection criteria. A limitation of *DIRECT* is that it requires surface smoothness property for convergence.

Huyer and Neumaier (1999) propose another black box partitioning approach called Multilevel Coordinate Search (*MCS*). *MCS* performs non-uniform partitioning by introducing a partition bias that divides boxes in the vicinity of samples having better function values. Similar to *DIRECT*, *MCS* also requires a smooth and continuous surface in the close neighborhood of the global optimum.

Pinter (1996) proposes a new powerful algorithm that integrates deterministic and probabilistic global and local search within a global partitioning framework. This algorithm is used in Lipschitz (-Continuous) Global Optimizer (*LGO*) development, which is a commercial package for solving global optimization problems (Pinter 1996, 1997).

Other Methods

The other deterministic approaches including branch and bound methods are Al-Khayyal and Sherali (2000), cutting plane methods (Tuy et al. 1985), outer approximation (Horst et al. 1992), primal–dual method (Floudas and Visweswaran 1993, Ben-Tal et al. 1994), alpha-Branch and Bound approach (Androulakis et al. 1995), reformulation techniques (Sherali and Tuncbilek 1992, Smith and Pandelides 1999, Sherali and Wang 2001), interior point methods (Morales et al. 2001, Forsgren et al. 2002, Leyffer et al. 2004, Leyffer 2005) and interval methods (Hansen 1992).

Branch and Bound techniques (B&B) are partitioning algorithms that are complete and reliable in the sense that they explore the whole feasible domain and discard sub-spaces in the feasible domain only if they are guaranteed to exclude feasible solutions

and/or local stationary points better than the ones already found. B&B are exhaustive algorithms that typically rely on generating lower and upper bounds for boxes in the search tree, where tighter bounds result in early pruning of nodes. For expediting B&B, feasibility and optimality based variable range reduction techniques (Ryoo and Sahinidis 1995, 1996), convexification (Tawarmalani and Sahinidis 2002, Tawarmalani and Sahinidis 2004), outer approximation (Burkard et al. 1992) and constraint programming techniques in pre- and post-processing phases of branching have been developed (Ryoo and Sahinidis 1996). The latter resulted in an advanced methodology and software called Branch and Reduce algorithm (*Baron*, Sahinidis 1996, Sahinidis 2003).

Symbolic reformulations / spatial branch-and-bound algorithm, which is another variant of B&B for nonconvex optimization problems, is developed with bounds tightening, optimization based tightening, and feasibility based tightening tests to obtain tighter bounds (Sherali and Tuncbilek 1992, Smith and Pantelides 1996, 1999, Smith 1996, Sherali and Wang 2001).

Surveys on global optimization are abundant in the literature (Pardalos and Romeijn 2002).

Some branching codes using Function Values only:

The codes listed use black box function evaluation routines, and have heuristic stopping rules.

i. DIRECT, Divide Rectangles (in Fortran, by Gablonsky and Kelley 2001, Gablonsky 2001)

gblSolve, a MATLAB 5 implementation of *DIRECT*

DIRECT method uses branching and a Pareto principle for the box selection. Jones et al. (1993) implement a simple and efficient global optimization method for bound constrained problems using DIRECT method.

ii. MCS, Multilevel Coordinate Search (Huyer and Neumaier 1999)

MCS is a branching and sequential quadratic programming algorithm, which can be developed using Matlab. However, for a bound constrained global optimization problem it uses function values only.

iii. LGO, Lipschitz Global Optimization (Pinter 1997)

This is an integrated development environment for global optimization with Lipschtiz continuous objective and constraints. *LGO* is based on branching and estimation of Lipschitz constants and interior convex constraints by projection penalties.

Some Branch and Bound Codes for Continuous Global Optimization:

The codes listed below use global information generally from required symbolic problem input. They have finite termination with guarantee that the global maximizer is found; in difficult cases storage or time limits may be exceeded, however, leading to appropriate error messages.

i. GAMS / *Baron* (Ryoo and Sahinidis 1996, Sahinidis 1996)

Baron is a general purpose solver for global optimization problems with nonlinear constraints and / or integer variables. It is one of the fast specialized solvers for many linearly constrained problems. *Baron* is based on branching and box reduction using convex relaxation and Lagrange multiplier techniques.

ii. αBB (Adjiman et al. 1998a, 1998b, Androulakis et al. 1995, Adjiman and Floudas 2001)

αBB is a branch and bound code for nonlinear programs. It is based on branching and bound by convex underestimation, using interval analysis to write nonlinearities in DC (difference of convex function) form.

b. Generalized Descent Methods

These methods continue the search trajectory every time a local solution is found. There are two approaches. First, trajectory methods modify the differential equations describing the local-descent trajectory so that they can escape from local maxima (Anderson and Walsh 1986, Snyman and Fatti 1987, Diener and Schaback 1990, Schaffler and Warsitz 1990, Sturua and Zavriev 1991, Vincent et al. 1992). Their advantage is the large number of function evaluations spent in unpromising regions. Second, penalty methods prevent multiple determinations of the same local maxima by modifying the objective function, namely, by introducing a penalty term on each local maximum (Ge and Qin 1987, Cetin et al. 1993). Their problem is that as more local maxima are found, the modified objective function becomes more difficult to minimize. In existing generalized descent methods, the descent trajectory is modified using internal function information, e.g., local maxima along the search.

Alternatively, deterministic methods can be classified into two categories (Shang 1997): (a) point-based methods, methods that compute function values at sampled points, such as generalized descent methods, and (b) region based methods, methods that compute function bounds over compact sets, such as covering methods. Point-based methods are unreliable, but usually have less computational complexity.

Region-based methods are expensive, but can produce rigorous global optimization solutions when they are applicable.

2. Probabilistic Methods

Probabilistic global maximization methods rely on probability to make decisions. The simplest probabilistic algorithm uses restart to bring a search out of a local maxima when little improvement can be made locally. Advanced methods use more elaborate techniques. The probabilistic methods are classified into clustering methods, random-search methods and methods based on stochastic models (Shang 1997).

a. Clustering Methods

Clustering analysis (Törn 1977, Boender et al. 1982, Törn and Viitanen 1992) is used to prevent the re-determination of already known maxima. There are two strategies for grouping points around a local maximum: (i) retain only points with relatively low function values (Beckernad and Lago 1970, Törn 1973); (ii) push each point toward a local maximum by performing a few steps of a local search (Schoen 1991, Törn et al. 1999). They do not work well when the function terrain is very rugged or when the search gets trapped in a deep but suboptimal valley (Zabinsky and Smith 1992).

b. Random Search Methods

These include pure random search, single-start (Sarma 1990, Schoen 1991, Spaans and Luus 1992, Törn et al. 1999), multi-start (Goldberg 1989, Michalewicz 1994), random line search, adaptive random search, partitioning subsets, replacing the worst point, evolutionary algorithms (Kirkpatrick et al. 1983, Aarts and Korst 1989), and simulated annealing.

Simulated annealing (Kirkpatrick et al. 1983) and genetic algorithms (Michalewicz 1994) are two popular stochastic global optimization methods. Simulated annealing

(*SA*) takes its intuition from the fact that heating and slowly cooling (annealing) a piece of metal brings it into a more uniformly crystalline state, which is believed to be the state where the free energy of bulk matter takes its global maximum. The role of temperature is to allow the configurations to reach energy states with a probability given by Boltzmann's exponential law, so that they can overcome energy barriers that would otherwise force them into local maxima. Simulated annealing is provably convergent asymptotically in a probabilistic sense, but may be exceedingly slow. Various ad hoc enhancements make it much faster. Simulated annealing has been successfully applied to solve many nonlinear optimization problems (Ingber 1994).

Özdamar and Demirhan (2000, 2001) provide extensive computational surveys reflecting the performance of both types of approaches (deterministic adaptive partitioning approaches and probabilistic approaches including many SA versions and clustering methods) where a large number of test functions are used. The authors reach the following empirical conclusion: *SA* (*SA* with local search (Özdamar and Demirhan 2000), and Adaptive SA (*ASA*, Ingber 1996) and the fuzzy adaptive partitioning scheme (Özdamar and Demirhan 2001) are the best performing ones among the tested algorithms. However, when the number of variables increase (above ten), the performance of both *ASA* and the fuzzy partitioning scheme deteriorate considerably and the *SA* with local search scheme becomes best performing. Yet, the results of the best approach are far from satisfactory.

The performance of *SA* depends on the number of variables of the function under investigation, because, as a single point search technique, *SA* converges rather slowly in order to provide sufficient moves carried out in every direction (variable). Dekkers and Aarts (1991) provided a convergence proof for *SA* in the real domain. Various SA implementations exist in the literature (Corana et al. 1987, Ingberg 1994, Zacharias et

al. 1998). The strong points of *SA* and some pitfalls for potential *SA* users are indicated in an extensive review given by Ingber (1994) where a wide range of application areas from finance to combat analysis are described.

Genetic algorithms make use of analogies to biological evolution by allowing mutations and crossovers between candidates of good local optima in the hope to derive even better ones. At each stage, a whole population of configurations is stored. Mutations are performed as local search, whereas crossover operators provide the ability to leave regions of attraction of local maximizers. With high probability, the crossover rules produce offspring's of similar or even better fitness. The effect of interchanging coordinates is beneficial mainly when these coordinates have a nearly independent influence on the fitness, whereas if their influence is highly correlated, such as for functions with deep and narrow valleys not parallel to the coordinate axes, genetic algorithms have more difficulties. Successful tuning of genetic algorithms requires a considerable amount of insight into the nature of the problem at hand. Genetic algorithms have shown promising results in solving nonlinear optimization problems (Glover 1980, Le Grand and Merz 1993, Michalewicz 1994).

Random search methods are easy to understand and simple to realize. The simplest random algorithm uses restarts to bring a search out of a local maximum. Others, such as simulated annealing, rely on probability to indicate whether a search should ascend from a local maximum. Other stochastic methods rely on probability to decide which intermediate points to interpolate as new starting points, like in random recombinations and mutations in genetic algorithms. These algorithms are weak in either their local or their global search. For instance, gradient information useful in local search is not used well in simulated annealing and genetic algorithms. In contrast, gradient-descent algorithms with multi-starts are weak in global search.

These methods perform well for some applications. However, they usually have many problem-specific parameters, leading to low efficiency when improperly applied (Betro and Schoen 1987, 1992, Boender and Rinnooy kan 1987, 1991).

c. Methods Based on Stochastic Models

Most of these methods use random variables to model unknown values of an objective function. One example is the Bayesian method, which is based on a stochastic function and minimizes the expected deviation of the estimate from the real global maximum (Mockus 1989, Zilinskas 1992, Mockus 1994). Bayesian methods do not work well because most of the samples they collect randomly from the error surface are close to the average error value, and these samples are inadequate to model the behavior at maximal points. Other methods based on stochastic models include methods that approximate the level sets. Although very attractive theoretically, this class of methods are too expensive to be applied to problems with more than twenty variables (Törn and Zilinskas 1989).

2.2. Interval Arithmetic

This section mainly deals with the foundations of interval arithmetic, the notation used, concept of inclusion functions, interval methods for uncertainty and some implementation examples for interval arithmetic. More detailed review over the interval arithmetic is available in Moore (1966), Alefeld and Herzberger (1983), Ratschek and Rokne (1988), Neumaier (1990), Hansen (1992), Ratschek and Rokne (1995), Kearfott (1996c), Neumaier (2004), and so on.

2.2.1. Foundations

Moore (1966) introduced the Interval arithmetic in the late 1960s to deal with infiniteness, to model uncertainty, and to tackle rounding errors of numerical

computation. Interval analysis is a set of algorithms that have been extended from numerical analysis to intervals, such as solving linear or nonlinear systems, differentiation or integration. Given a problem over the real numbers, interval computations are said to be reliable (verified) since no solution is lost (Granvilliers 2004).

Interval arithmetic is an elegant tool for practical work with inequalities, approximate numbers, error bounds, and more generally with certain convex and bounded sets. Moreover, it can also be used for solving bound constrained optimization, constraint satisfaction problem, and constrained optimization problems.

2.2.2. Basics of Interval Arithmetic and Terminology

Definition 2.1: *Interval arithmetic (IA) is an arithmetic defined on convex sets of real numbers, called intervals.* (Moore 1966, Alefeld and Herzberger 1983, Kearfott 1996a) ■

The *set of intervals* is denoted by $\mathbb{I}\mathbb{R} := \{[a, b] \mid a \leq b, a, b \in \mathbb{R}\}$.

Note that, in order to represent the real line with closed sets, $\mathbb{I}\mathbb{R}$ is made compact in the obvious way with the infinities $\{-\infty, +\infty\}$. The usual conventions apply: $(+\infty) + (+\infty) = +\infty$, and so on. Every interval $X \in \mathbb{I}\mathbb{R}$ is denoted by $[\underline{X}, \overline{X}]$, where its bounds are defined by $\underline{X} = \inf X$ and $\overline{X} = \sup X$.

For every $a \in \mathbb{I}\mathbb{R}$, the *interval point* $[a, a]$ is also denoted by a.

Some important notions are as follows:

- Given a subset ρ of $\mathbb{R}$, the *convex hull* of ρ is the interval Hull $(\rho) = [\inf \rho, \sup \rho]$.

- The *width* of an interval X is the real number, and defined as $w(X) = \overline{X} - \underline{X}$.

  Example : The width of interval X = [2, 3] is defined as $w(X) = 3 - 2 = 1$.

- Given two real intervals X and Y, X is said to be *tighter than* Y if $w(X) \leq w(Y)$.

- The *radius* of an interval X is defined as $r(X) = 0.5 * w(X) = 0.5 * (\overline{X} - \underline{X})$.

- The *midpoint* of an interval X is defined as $m(X) = 0.5 * (\overline{X} + \underline{X})$.

- The *mignitude* of an interval X is the number $\langle X \rangle = \min_{x \in X} |x|$.

- The *magnitude* of an interval X is the number $|X| = \max \{|\underline{X}|, |\overline{X}|\}$.

Elements of $\mathbb{II}^n$ also define boxes. Given $X_i \in \mathbb{II}$, $i = 1, 2, \ldots, n$, the corresponding *box* **X** is the Cartesian product of intervals, $\mathbf{X} = X_1 \times \ldots \times X_n$, where $\mathbf{X} \in \mathbb{II}^n$. A subset of **X**, $\mathbf{Y} \subseteq \mathbf{X}$, is a sub-box of **X**. The notion of *width* is defined as follows in equation (2.1):

$$w(X_1 \times \ldots \times X_n) = \max_{1 \leq i \leq n} w(X_i) \tag{2.1}$$

Interval Arithmetic operations are set theoretic extensions of the corresponding real operations as defined in equation (2.2). Given $X, Y \in \mathbb{II}$, and an operation $\diamond \in \{+, -, \times, \div\}$, we have:

$$X \diamond Y = \text{Hull } X \diamond Y, \text{ where } (x, y) \in X \times Y. \tag{2.2}$$

For all $X, Y \in \mathbb{II}$ and $X = [\underline{X}, \overline{X}]$ and $Y = [\underline{Y}, \overline{Y}]$ it holds that

Addition rule:

$$X + Y = [\underline{X} + \underline{Y},\ \overline{X} + \overline{Y}]$$

Subtraction rule:

$$X - Y = [\underline{X} - \overline{Y},\ \overline{X} - \underline{Y}]$$

Multiplication rule:

$$X \times Y = [\ min(\underline{X}\,\overline{Y},\ \overline{X}\,\underline{Y},\ \underline{X}\,\underline{Y},\ \overline{X}\,\overline{Y}),\ max(\underline{X}\,\overline{Y},\ \overline{X}\,\underline{Y},\ \underline{X}\,\underline{Y},\ \overline{X}\,\overline{Y})]$$

Division rule:

$$1 \div Y = [1/\overline{Y},\ 1/\underline{Y}]$$

$$X \div Y = X \times 1/Y \text{ if } 0 \notin Y$$

n^{th} Power rule :

$$\mathbf{X^n} = \begin{cases} [1, 1] & \text{if } n = 0 \\ [\underline{X}^{\,n},\ \overline{X}^{\,n}] & \text{if } \underline{X} \geq 0 \text{ or } \underline{X} \leq 0 \leq \overline{X} \text{ and } n \text{ is odd} \\ [\overline{X}^{\,n},\ \underline{X}^{\,n}] & \text{if } \overline{X} \leq 0 \\ [0,\ max(\underline{X}^{n},\ \overline{X}^{n})] & \text{if } \underline{X} \leq 0 \leq \overline{X} \text{ and } n \text{ is even for } n = 0, 2, 4, 6, \ldots\ldots \end{cases} \quad (2.3)$$

Due to properties of monotonicity, these operations can be implemented by real computations over the bounds of intervals. Given two intervals X = [*a*, *b*] and Y = [*c*, *d*], we have for instance: X + Y = [*a*+*c*, *b*+*d*]. The associative and the commutative laws are preserved over $\mathbb{I}$. However, the distributive law does not hold. In general, only a weaker law is verified, called semi-distributivity.

2.2.3. Extended Interval Arithmetic

In the above rules of interval arithmetic, division by an interval containing zero is excluded. But it is often useful to remove this restriction. The resulting arithmetic is

called extended interval arithmetic. The arithmetic was first discussed (independently) by Hansen (1968) and Kahan (1968).

The rule for extended interval arithmetic must also satisfy associative and the commutative laws defined in equation (2.3). This condition gives rise to a set of rules extending to those in the section 2.2.2.

Popova (1996) investigates the algorithmic aspects for implementation of interval arithmetic involving NaN's or signed zero and developed a simple model for the interval arithmetic exceptions and their handling in IEEE non-trapping mode. Verdonk et al. (2002) propose a set based representation of non-real to remove the restrictions (for example: divisible by zero, interval domains containing points outside the domain of the underlying functions) on the domain of interval functions and to guarantee the inclusion property in all situations for 100% reliability of interval arithmetic.

Hyvönen (2001) investigates the conceptual and practical difficulties of the end users to interface with intervals. Further, Popova (2001) summarizes the distributive relations on multiplication and addition of generalized intervals.

Kearfott et al. (2002) propose a new standard notation for the interval arithmetic and to standardize the notation used for interval analysis.

For all $X, Y \in \mathbb{I}$, and $X = [\underline{X}, \overline{X}]$ and $Y = [\underline{Y}, \overline{Y}]$ it holds that

If $0 \in Y$, then

$$\frac{X}{Y} = \begin{cases} X\,[1/\overline{Y},\ 1/\underline{Y}\,] & \text{if } 0 \notin Y \\ [-\infty, \infty] & \text{if } 0 \in X \text{ and } 0 \in Y \\ [\,\overline{X}/\underline{Y}, \infty] & \text{if } \overline{X} < 0 \text{ and } \underline{Y} < \overline{Y} = 0 \\ [-\infty, \overline{X}/\overline{Y}\,] \cup [\,\overline{X}/\underline{Y}, \infty] & \text{if } \overline{X} < 0 \text{ and } \underline{Y} < 0 < \overline{Y} \\ [-\infty, \overline{X}/\overline{Y}\,] & \text{if } \overline{X} < 0 \text{ and } 0 = \underline{Y} < \overline{Y} \\ [-\infty, \underline{X}/\underline{Y}\,] & \text{if } 0 < \underline{X} \text{ and } \underline{Y} < \overline{Y} = 0 \\ [-\infty, \underline{X}/\underline{Y}\,] \cup [\,\overline{X}/\overline{Y}, \infty] & \text{if } 0 < \underline{X} \text{ and } \underline{Y} < 0 < \overline{Y} \\ [\,\underline{X}/\overline{Y}, \infty] & \text{if } \underline{X} < 0 \text{ and } 0 = \underline{Y} < \overline{Y} \\ \varnothing & \text{if } 0 \in X \text{ and } 0 = Y \end{cases} \quad (2.4)$$

2.2.4. The Dependency Problem

Suppose, the interval $X = [a, b]$ subtracts from itself.

Apply the subtraction rule defined in equation (2.3), the result obtained is

$$X - X = [a, b] - [a, b] = [a\text{-}b, b\text{-}a] \quad (2.5)$$

We might expect to obtain [0, 0]. However, we do not (unless $b=a$). The result is $\{x–y: x \in X, y \in x\}$ instead of $\{x–x : x \in X\}$.

In general, when a given variable occurs more than once in an interval computation, it is treated as a different variable in each occurrence. Thus, X – X is the same as X – Y with Y equal to but independent of X. This causes widening of computed intervals

and makes it more difficult to obtain sharp results in calculations. One should always be aware of this consideration and take appropriate steps to reduce its effect.

The n^{th} Power rule defined as above is used to overcome the dependency problem in multiplication. For example, for $n = 2$, the definition is equivalent to equation (2.6)

$$X^2 = \{x^2 : x \in X\} \tag{2.6}$$

rather than,

$$X \times X = \{x * y : x \in X, y \in X\} \tag{2.7}$$

If a particular interval variable occurs only once in a given form of a function, then it cannot give rise to dependency. Thus dependency can occur in evaluating a function $f(X,Y)$ of the form $(X - Y)/(X + Y)$, but not if it is rewritten as $1-2/(1 + X/Y)$.

If we evaluate $f(X, Y)$ in the latter form, the resulting interval is the exact range of $f(x, y)$ for $x \in X$ and $y \in Y$.

Overestimation for a function can be defined as the difference between actual and the exact bounds of a function. This is mainly due to the dependency problem. Neumaier (1982) and Stahl (1997) propose a technique for finding the overestimation and its bounds in computing the function values.

Various methods were proposed to handle dependency problems such as Gröbner basis (Buchbeger 1985), factorizations (Ceberio and Granvilliers 2002), substitution, and so on.

2.2.5. Interval Arithmetic Properties

Theorem 2.1 (Algebraic Properties) (Hansen 1992)

For all X, Y, and Z $\in$ $\mathbb{I}$.

- Associativity : $(X + Y) + Z = X + (Y + Z)$ and $(XY) Z = X(YZ)$

- Commutativity : (X + Y) = (Y + X), XY = YX
- Neutral Element : (0 + X) = X, 1*X = X*1

However, proper intervals do not have additive or multiplicative inverses. Further, the distributivity law does not hold for intervals. Instead, there is a weaker version of the same given as follows:

Theorem 2.2 (Subdistributivity) (Hansen 1992)

Interval arithmetic is subdistributive in the sense that, if X, Y, and Z are intervals, then

$$X(Y + Z) \subseteq XY + XZ$$

Thus, although addition or multiplication of intervals is commutative and associative, the distributive laws do not hold. Furthermore, although there is an additive identity [1, 1], additive and multiplicative inverses do not exist.

2.2.6. Inclusion Functions

Interval arithmetic is particularly appropriate to represent outer approximations of real quantities. The range of a real function *f* over a domain **D**, denoted by *F*(**D**), can be computed by interval extensions.

Definition 2.2 (Interval Extension): *An* interval extension *of a real function* $f : \mathbf{D}_f \subset \mathbb{R}^n \to \mathbb{R}$ *is a function* $\Im : \mathbb{R}^n \to \mathbb{R}$ *such that*

$$\forall\, \mathbf{X} \in \mathbb{I}^n, (\mathbf{X} \in \mathbf{D}_f \Rightarrow F(\mathbf{X}) = \{ f(x) \mid x \in \mathbf{X} \} \subseteq \Im(\mathbf{X})). \blacksquare$$

This inclusion formula is called Fundamental Theorem of Interval Arithmetic. *Interval extensions are also called* interval forms or inclusion functions.

This definition implies the existence of infinitely many interval extensions of a given real function. In particular, the weakest and tightest extensions are respectively defined by: $X \to [-\infty, +\infty]$ and $X \to$ Hull $F(X)$.

The most common extension is known as the *natural extension*. Natural extensions are obtained from the expressions of real functions, and are *inclusion monotonic* (this property follows from the monotonicity of interval operations). Hence, given a real function *f*, whose natural extension is denoted by *F*, and two intervals X and Y such that $X \subset Y$, the following holds: $F(X) \subset F(Y)$. We denote the lower and upper bounds of the function interval range over a given box Y as $\underline{F}(Y)$ and $\overline{F}(Y)$, respectively.

For a standard function h such as sin, exp, and so on, pre-declared in a given programming language, it is not too difficult to obtain a good inclusion function *H*, since monotonicity properties of these functions are well known and then $H(X) = \{h(x): x \in X\}$ for any $X \in \mathbb{I}$ in the domain of *h*. For a general function $f(x)$, $x \in \mathbb{R}^n$, the easiest method to obtain an inclusion function is so called natural interval extension, which is obtained by replacing each occurrence of variable *x* with a box including it, X, each occurrence of a predeclared function *h* by its corresponding interval operators.

Some important properties of the inclusion function are given as follows (Neumaier 1990, Hansen 1992, Ratschek and Rokne 1995, Stahl 1995).

The properties of the inclusion function are as follows:

Property 1: The inclusion function *F* is said to be an isotone inclusion function over X_0 if for any pair of boxes $Y, Z \subseteq X_0$, $Y \subseteq Z$ implies $F(Y) \subseteq F(Z)$.

Property 2: The inclusion function *F* is said to be a $\propto$-convergent inclusion function over X_0 if for any box $Y \subseteq X_0$, $w(F(Y))\text{-}w(f(Y)) \leq c\, w(Y)^{\alpha}$ holds, where $\propto$ and c are positive constants and $f(Y)$ is the range of f over Y.

Property 3: The inclusion function f has the zero convergence property, if $w\ (F\ (Z)) \rightarrow 0$ holds for all the $\{Z_i\}$ interval sequences for which $Z_i \subseteq X_0$ $\forall\ i = 1, 2, \ldots$ and $w\ (Z_i) \rightarrow 0$.

The following are the three methods used for developing the inclusion functions, which can easily be constructed (Ratschek and Rokne 1995, Neumaier 1990, Tóth 2002):

a. Natural Interval Extension

Natural interval extension expands the arithmetical operations in a straightforward way from real to intervals (Moore 1966). The natural inclusion function for all functions is created by substitution of the real variables for intervals and the real operations or standard functions for interval operations or respective inclusion functions. It was shown that the convergence order of the natural inclusion function is at least 1.

Natural interval extension inclusion functions are most widely used inclusion functions, which can be applicable for non-differentiable function also.

b. Centered Form

The centered form can be sub classified into two forms: 1) Mean value forms, and 2) Taylor forms (of second order). However, these inclusion functions depend on the derivative of the function, thus we will use this form if and only if the function if differentiable and also when the enclosure of the derivative is feasible to compute. The derivative of the given function can be easily calculated using automatic differentiation.

Mean value forms (F_c (X)) involving generalized gradients, can be defined as equation (2.8):

$$F_c(X) = f(c) + (X - c)^T F'(X) \text{ for } X \in \mathbb{I} \qquad (2.8)$$

where $c = m(X)$. In general, midpoint will be used for c, but it can be anywhere in X. If the convergence order of the inclusion function $F'(X)$ is at least one, then the convergence order of the centered form is at least a quadratic.

Taylor forms can only be used when the direct computation of the mean value form is not possible or if the Hessian inclusion $F''(X)$ is already available and can be incorporated without difficulties. Taylor form ($F_t(X)$) can be defined as equation (2.9):

$$F_t(X) = f(c) + (X - c)^T f'(c) + 0.5\,(X - c)^T F''(X)(X - c) \text{ for } X \in \mathbb{I} \qquad (2.9)$$

c. Baumann's Optimal Centered Form

Baumann introduced the optimal c for the centered form. He proved that $f_c(X) \leq f_{b^-}(X)$ $\forall c \in X$ for an optimal $b^- \in X$. The optimal lower bound is $\underline{f_{b^-}}(X)$, where b^- can be obtained in the one dimensional case as

$$b^- = \frac{\underline{X} - \overline{X}}{\overline{f'}(X) - \underline{f'}(X)},$$

In higher dimensions the above formula must be used component wise. For the optimal upper bound b^+ one should reflect b^- on the midpoint of the interval X.

The Baumann's form always gives at least as good enclosure as the general centered form; still its convergence order is also at least 2.

Nataraj and Kotecha (2002) propose an algorithm for global optimization using the Taylor Bernstein form as inclusion function for better convergence rate in the function. Also, the empirical convergence speed of inclusion functions are studied and found that natural interval extension of a given function can be as good as usual

quadratically convergent inclusion functions. Tóth (2002) compared different inclusion methods and found that the centered forms (second-order) as larger convergences order than that of the rest of the inclusion methods. Vinkó et al. (2002) propose a new inclusion function called Kite for one-dimensional and multi dimensional functions using the least gradient information. They also study the impact of the kite inclusion function over the convergence, which helps in further implementation.

2.2.7. Interval Computations and Mathematical Proofs

The powerful aspect of interval computations is tied to the Brouwer fixed-point theorem (Kearfott 1996a), and shown in the Theorem 2.3.

Theorem 2.3 (Brouwer fixed point theorem) (Kearfott 1996a)

Let **D** be homeomorphic to the closed unit ball in $\mathbb{R}^n$, and suppose P is a continuous mapping such that the P maps **D** into **D**, then P has a fixed point, i.e., there is an $X \in \mathbf{D}$ such that P (X) = X.

The Brouwer fixed point theorem combined with interval arithmetic enables numerical computations to prove existence of solutions to linear and nonlinear systems. The simplest context in which this can be explained is the one-dimensional interval Newton method.

Suppose f: $X = [\underline{X}, \overline{X}] \rightarrow \mathbb{R}$ has a continuous first derivative on X, suppose $\tilde{x} \in X$, $F'(X)$ is a set that contains the range of f' over X (such as when f' is evaluated at X with interval arithmetic). Then the operator

$$\boldsymbol{N}(f; X, \tilde{x}) = \tilde{x} - f(\tilde{x}) / F'(X)$$

is termed the univariate interval Newton method. Applying the Brouwer fixed point

theorem in the context of the univariate interval Newton methods leads to:

Theorem 2.4 (Miranda 1940)

If $N(f; \mathbf{X}, \tilde{x}) \subset \mathbf{X}$, then there exists a unique solution of $f(x) = 0$ in **X.**

Existence of Theorem 2.4 follows from Miranda's theorem (Miranda 1940), a corollary of Brouwer fixed point theorem. Uniqueness is as follows: Suppose there were two solutions $x \in X$ and $\tilde{x} \in X$. Then $f(x) = 0 = f(\tilde{x})$, so there is a $\xi \in X$ with

$$f(x) = f(\tilde{x}) + f'(\xi)(x - \tilde{x}) = f'(\xi)(x - \tilde{x}) = 0.$$

However, since $\boldsymbol{N}(f; X, \tilde{x}) \subset X$, $f'(\xi)$ cannot contain zero, so $0 \notin f'(X)(x - \tilde{x})$. But this contradicts $0 = f'(\xi)(x - \tilde{x}) \in f'(X)(x - \tilde{x})$.

Existence theory for multivariate interval Newton methods is similar. Uniqueness theory proceeds by proving that the intervals derivative matrix or interval slope matrix is regular. There are various ways of doing this computationally. For example, if a preconditioned interval version of Gaussian elimination completes without pivots that contain zero, and then the interval matrix cannot contain any singular matrices.

This computational existence uniqueness theory has wide use, from constructing narrow bounds around approximate solutions to linear systems, within which an actual solution must lie, to proving existence and uniqueness of solutions to operator equations.

2.2.8. Interval Newton Method

Interval Newton methods are excellent methods for determining all zeros of a continuously differentiable vector-valued function $\phi: \mathbf{X} \rightarrow \mathbb{R}^m$ where $\mathbf{X} \in \mathbb{I}^m$. These methods are important tools for nonlinear optimization problems since they can be

used for computing the critical points of ϕ by applying the methods of Jacobian ($\boldsymbol{J}_{\phi}(\mathbf{X})$), or for solving Kuhn-Tucker or John conditions in constrained optimization.

The interval Newton method was introduced by Moore (1966) and it has been further extensively developed by many researchers. The latest state of art for interval Newton methods may be found in Neumaier (1990). More detailed review of Interval Newton methods is available in Ratschek and Rokne (1988), Neumaier (1990), Hansen (1992), Ratschek and Rokne (1995), Kearfott (1996c) and so on.

2.2.9. Interval Methods for Uncertainity

In many real-life situations, the input data come from measurements. Measurements are not 100% precise (Tung 2001). Therefore, if we have made the measurements with accuracy Δ, and z obtained the measurement result, this means that the actual value y of the measured quantity can take any value from the interval $y = [z-\Delta, z+\Delta]$. The desired solution x depends on the exact value of y from this interval. Thus, it makes sense to produce the set of all possible solutions x that correspond to all possible values $y \in [z-\Delta, z+\Delta]$. By this way interval framework can also consider the uncertainty in the input data (Kearfott and Kreinovich 1996). This makes the Interval Analysis to tackle the uncertainty of data inherently, which makes the whole approach more powerful in real-time scenarios.

There are several useful quantities related to the concept of the interval: size, radius, and midpoint (Schwartz 1999). The size (or thickness) of an interval indicates the uncertainty in a value and is specified as a width ≥ 0. Intervals with zero thickness are crisp intervals whereas non-crisp intervals are said to be thick. The concepts of radius and midpoint are useful in describing intervals as well as constructing them.

To construct a new interval, one way is to use an original value, which is a value that supplies the midpoint point of a new interval. Then, a certain radius (uncertainty) can be added to and subtracted from the original value to obtain a new interval. Similarly, the midpoint can also serve as an approximation to a value with an error of plus or minus the radius. Using these definitions, the percentage uncertainty in a midpoint value would be:

$$p = \frac{r(\mathrm{X})}{m(\mathrm{X})} * 100$$

2.2.10. Motivation in Selection of Interval Methods

The following indicates the advantages of intervals.

- the control of all kinds of errors, especially rounding errors, truncation errors, etc (Ratschek and Rokne 1995),
- the processing of infinite data sets (Ratschek and Rokne 1995),
- With regard to the global optimization, interval based optimization techniques are able to continually delete portions of the search space with the objective of maintaining a final box of any desired width, which contains the global solution (Vaidyanathan and Halwagi 1994),
- Interval arithmetic can take care of uncertainty inherently, which is most common property of any real-time problem (Kearfott and Kreinovich 1996, Schwartz 1999), and
- Ease in integration to symbolic computing and consistency techniques.

2.2.11. Applications of Interval Analysis

The following list indicates some of the basic applications using interval analysis (Kearfott and Kreinovich 1996).

- In Engineering:
  - to manufacturing, including:
    - quality control;
    - detection of defects in computer chips;
    - flexible manufacturing.
  - to automatic control, including:
    - control of airplane engines;
    - control of electric power plants.
  - to robotics;
  - to airplane inertial navigation;
  - to civil engineering, including traffic control.
- In Ergonomics and Social Sciences:
  - to learning curves (that describe how people learn);
  - to project management;
  - to service systems;
  - to sociology.
- In Physics:
  - to laser beams;
  - to particle accelerators;
  - to astrophysics (planet atmospheres);
  - to image processing in radio astronomy.
- In Geology and Geophysics.

- In Chemistry , including:
    - to spectral analysis.
- In Computer Science and Engineering:
    - to expert systems;
    - to communication networks, especially computer networks.
- In Economics:
    - to planning;
    - to banking.

2.3. Feasible Sequential Quadratic Programming (FSQP)

This section deals with Feasible Sequential Programming (*FSQP*) and its related algorithms. Moreover, it also presents the basic motivation in selection of *FSQP* for the current research work. This section is greatly influenced by the information given in http://www.aemdesign.com/FSQPwhatis.htm.

2.3.1. Introduction

Portable standard C implementation *FSQP* (*CFSQP*) and Fortran77 implementation *FSQP* (*FFSQP*) were originally developed by Andre Tits' research group at the Institute for Systems Research (ISR), University of Maryland.

The algorithm's main architects were Panier and Tits (1988). The implementation was due to Zhou et al. (1997). The first version of the *CFSQP*, carried out by Craig Lawrence, followed in 1993 (Lawrence et al. 1997, Lawrence and Tits 2001). Responsibility for their further development and support was transferred to AEM Design in 2000.

FSQP is a source code for minimization of the maximum of a set of smooth objective functions subject to general smooth constraints.

If the initial guess provided by the user is infeasible for some inequality constraint or some linear equality constraint, *FSQP* first generates a feasible point for these constraints; subsequently, the successive iterations generated by *FSQP* all satisfy these constraints. Nonlinear equality constraints are turned into inequality constraints and the maximum of the objective function is replaced by an exact penalty function which penalizes nonlinear equality constraint violations only. The user has the option of either requiring that the objective function (penalty function if nonlinear equality constraints are present) decreases at each iteration after feasibility for nonlinear inequality and linear constraints has been reached (monotone line search), or requiring a decrease within at most four iterations (nonmonotone line search). The user must provide functions that define the objective functions and constraints and may either provide functions to compute the respective gradients or require that *FSQP* estimate them by forward finite differences.

When solving problems with numerous sequentially related constraints (or objectives), such as discretized semi-infinite programming (*SIP*) problems, the C version *CFSQP* gives the user the option to use an algorithm that efficiently solves these problems, greatly reducing computational effort.

FSQP is an implementation of two algorithms based on Sequential Quadratic Programming (*SQP*), modified so as to generate feasible iterates. In the first one (monotone line search), a certain Armijo type arc search is used with the property that the step one is eventually accepted, a requirement for superlinear convergence. In the second one the same effect is achieved by means of a nonmonotone search along a straight line. The merit function used in both searches is the maximum of the objective functions if there is no nonlinear equality constraint, or an exact penalty function if nonlinear equality constraints are present.

2.3.2. The Basic FSQP Algorithm

SQP (Sequential Quadratic Programming) type algorithm modified so as to generate feasible iterates. The basic problem solved is (where the variable x is n-dimensional)

$$\min_{x} \max_{i \in I} \{f_i(x)\}$$

subject to

$g_j(x) \leq 0, j = 1,\ldots,n_{i,}$ where, n_i is the number of inequalities

$h_j(x) = 0, j = 1,\ldots,n_{e,}$ where, n_e is the number of equalities.

Two phase operation:

Phase I - generate iterate satisfying all linear constraints and nonlinear inequality constraints.

Phase II - minimize the maximum of the objectives, while maintaining satisfaction of linear constraints and nonlinear inequality constraints, nonlinear equality constraints being satisfied asymptotically.

Feasibility

Consider the simple problem

$$\min_{x} f_i(x)$$

subject to $g(x) \leq 0,$

Feasibility requires $g(x_k) \leq 0,$ for all k.

FSQP generates iterates that satisfy all inequality constraints and linear equality constraints.

2.3.3. Why Feasibility?

From an application point of view:

- Objective may not be achieved if certain constraints are violated. For example, the steady-state errors of a dynamical system are undefined if the system is not stable. Important for real-time applications.
- Termination of the optimization process after a prescribed amount of time, in which case it may be crucial that the sub-optimal solution satisfy at least some hard constraints.
- In the context of optimal design, tradeoff exploration cannot meaningfully take place if some hard constraints are not first satisfied. It is thus of great interest to produce iterates that all satisfy these hard constraints.

From an algorithmic point of view:

- The line search criterion can be based on the decrease of the objective function, i.e., there is no need for an artificial "merit function".
- In the SQP context, whenever the current iterate is feasible, the QP sub problem has a feasible solution.

2.3.4. Nonlinear Equality Constraints

The basic *FSQP* algorithm was designed for nonlinear inequality constraints only. To handle nonlinear equality constraints, *FSQP* incorporates a modification of a scheme due to Mayne and Polak (1976). Equality constraints $h(x) = 0$ are turned into inequality constraints $h(x) \leq 0$ and $h(x) \geq 0$. Negative values are penalized, where the

objective function is replaced with $f(x)$- $\sum_{j=1}^{m_e} c^j h_j(x)$, where c^j, $j = 1,\ldots,m_e$, are positive penalty parameters (iteratively increased, but bounded).

The result is a *differentiable exact penalty function.*

2.3.5. Line Search

***FSQP* provides a choice of line searches:**

Armijo (monotone) - Requires decrease of objective function at every step (Arminjo 1966).

Drawback: requires evaluation of constraints at an intermediate point.

Nonmonotone - Requires decrease of objective in at most *four* steps (Bonnans et al. 1992).

Drawback: objective could increase between successive iterates.

Special features of line search: no further evaluation of constraints once a constraint is violated. Active or previously violated functions are evaluated first at each trial point.

2.3.6. Multiple Objectives/Constraints

Consider the following problem:

$$\min_{x} \max_{\omega \in \Omega} \{f_i(x, \omega)\}$$

$$\text{subject to } g(x, \xi) \leq 0, \ \forall \xi \in \Xi$$

where $\Omega \subset \text{IR}$, $\Xi \subset \text{IR}$ are large, but *finite*, sets. For example, finely discretized Semi infinite Programming (SIP) problems.

CFSQP is equipped to efficiently solve problems with large sets of objectives and/or constraints. Consequently, the sizes of the QP's to be solved as well as the number of gradient evaluations are drastically reduced.

2.3.7. Automatic Differentiation

ADIFFSQP Version 0.9 (experimental version), an interface between *FFSQP* and the automatic differentiation preprocessor *ADIFOR2.0*, is also available (Liu and Tits 1997).

2.3.8. Selected Applications (http://www.aemdesign.com/FSQPapplref.htm)

- RIOTS: A Matlab toolbox for solving optimal control problems,
- Magnetic Resonance Imaging ,
- Clutter noise in Over-The-Horizon radar,
- Robotic manipulation planners,
- Hub-and-shaft assemblies for dual-wheel excavators,
- Optimal Protein Separation,
- Parametric Surface Polygonization,
- Analysis of intermediately lethal tumors,
- Hierarchical traffic control systems,
- Failure detection and isolation,
- Multi-purpose reservoir systems,
- Neural net based predictive control, and
- Aerosol thermodynamics

2.3.9. Motivation in Selection of *FSQP*

- *FSQP* algorithm uses directly tackling optimization problems with: multiple competing linear/nonlinear objective functions (minimax), linear/nonlinear inequality constraints, and linear/nonlinear equality constraints. (http://www.enee.umd.edu/Newsletter/vol5_no1/fsqp.htm)
- It also contains special provisions for maintaining "semi-feasibility" of each iterate and efficiently handling problems with multiple "sequentially related" objectives and/or constraints. (http://www.enee.umd.edu/Newsletter/vol5_no1/fsqp.htm)
- *FSQP* methods are particularly useful for solving those problems arising from engineering design where the objective function might be undefined outside the feasible region (Yang et al. 2003).
- *FSQP* methods are that the objective function can be used as a merit function to avoid the use of a penalty function (Yang et al. 2003).
- The main advantage of this algorithm is a reduction in the amount of computation required in order to generate a new iterate (Lawrence and Tits 2001).
- *FSQP* is interfaced with Automatic differentiation (Liu and Tits 1997).

Chapter 3

Tree Management Approach

This chapter mainly describes the existing tree management systems and proposes a new tree management system. The existing tree management systems are originally developed by Körf (1998) is the basis for the proposed new system.

3.1. Introduction

The tree management systems can be broadly classified into: 1. Brute-force search tree, and 2. Heuristic search tree (Körf 1998).

1. Brute-force Search Tree

The most general search algorithms are brute-force searches, since they do not require any domain specific knowledge. All that is required for a brute-force search is a state description, a set of legal operators, an initial state, and the description of the goal state. The most important brute-force searches are breadth-first, uniform-cost, depth-first, depth-first iterative deepening, and bidirectional search. In the descriptions of the algorithms below, to generate a node means to create the data structure corresponding to that node, whereas to expand a node means to generate all the children of that node.

a. Breadth-First Search

Breadth-first search expands nodes in order of their distance from the root, generating one level of the tree at a time until a solution is found. This is shown in Figure 3.1. It is most easily implemented by maintaining a queue of nodes, initially containing just

the root, and always removing the node at the head of the queue, expanding it, and adding its children to the tail of the queue.

Since it never generates a node in the tree until all the nodes at shallower levels have been generated, breadth-first search always finds a shortest path to a goal. Since each node can be generated in constant time, the amount of time used by breadth-first search is proportional to the number of nodes generated, which is a function of the branching factor '*b*' and the solution depth *'d'*. Since the number of nodes at level '*d*' is b^d, the total number of nodes generated in the worst case is $b + b^2 + b^3 + \ldots + b^d$, which is $O(b^d)$, the asymptotic time complexity of breadth-first search.

The main drawback of breadth-first search is its memory requirement. Since each level of the tree must be saved in order to generate the next level, and the amount of memory is proportional to the number of nodes stored, the space complexity of breadth-first is also $O(b^d)$. As a result, breadth-first search is severely space-bound in practice, and will exhaust the memory available on typical computers in a matter of minutes.

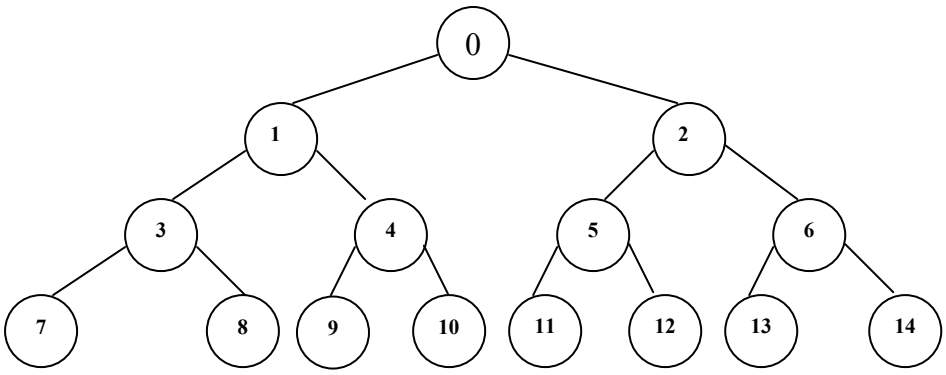

Figure 3.1 Order of Node Generation for Breadth-First Search

Source: Körf (1998)

b. **Uniform-Cost Search**

If all edges do not have the same cost, then breadth-first search generalizes to uniform-cost search. Instead of expanding nodes in order of their depth from the root, uniform-cost search expands nodes in order of their cost from the root. At each step, the next node n to be expanded is one whose cost $g(n)$ is lowest, where $g(n)$ is the sum of the edge costs from the root to node n. The nodes are stored in a priority queue. This algorithm is also known as Dijkstra's single-source shortest-path algorithm (Dijkstra 1959).

Whenever a node is chosen for expansion by uniform-cost search, a lowest-cost path to that node has been found. The worst-case time complexity of uniform-cost search is $O(b^{c/m})$, where c is the cost of an optimal solution, and m is the minimum edge cost. Unfortunately, it also suffers the same memory limitation as breadth-first search.

c. **Depth-First Search**

Depth-First Search remedies the space limitation of breadth-first search by always generating next a child of the deepest unexpanded node as shown in Figure 3.2. Both algorithms can be implemented using a list of unexpanded nodes; with the difference that breadth-first search manages the list as a first-in first-out queue, whereas depth-first search treats the list as a last-in first-out stack. More commonly, depth-first search is implemented recursively, with the recursion stack taking the place of an explicit node stack.

The advantage of depth-first search is that its space requirement is only linear with respect to the search depth, as opposed to exponential for breadth-first search. The reason is that the algorithm only needs to store a stack of nodes on the path from the

root to the current node. The time complexity of a depth-first search to depth '*d*' is $O(b^d)$, since it generates the same set of nodes as breadth-first search, but simply in a different order. Thus, as a practical matter, depth-first search is time limited rather than space-limited.

The disadvantage of depth-first search is that it may not terminate on an infinite tree, but simply goes down the left-most path forever. Even a finite graph can generate an infinite tree. The usual solution to this problem is to impose a cutoff depth on the search. Although the ideal cutoff is the solution depth '*d*', this value is rarely known in prior to solving the problem. If the chosen cutoff depth is less than '*d*', the algorithm will fail to find a solution, whereas if the cutoff depth is greater then '*d*', a large price is paid in execution time, and the first solution found may not be an optimal one.

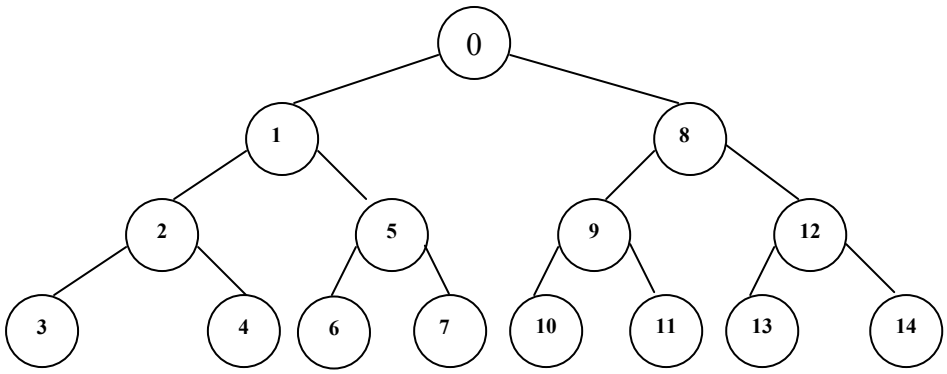

Figure 3.2 Order of Node Generation for Depth-First Search

Source: Körf (1998)

d. Depth-First Iterative-Deepening

Depth-first iterative deepening (*DFID*) combines the best features of breadth-first and depth-first search (Körf 1985, Stickel and Tyson 1985). *DFID* first performs a depth-first search to depth one, then starts over, executing a complete depth-first search to

depth two, and continues to run depth-first searches to successively greater depths, until a solution is found (see Figure 3.3).

Since it never generates a node until all shallower nodes have been generated, the first solution found by *DFID* is guaranteed to be along a shortest path. Furthermore, since at any given point it is executing a depth-first search, saving only a stack of nodes, and the algorithm terminates when it finds a solution at depth *'d'*, the space complexity of *DFID* is only $O(d)$.

Although it appears that *DFID* wastes a great deal of time in the iterations prior to the one that finds a solution, this extra work is usually insignificant. To see this, note that the number of nodes at depth *'d'* is b^d, and each of these nodes are generated once, during the final iteration. The number of nodes at depth d-1 is b^{d-1}, and each of these is generated twice, once during the final iteration, and once during the penultimate iteration. In general, the number of nodes generated by *DFID* is

$$b^d + 2b^{d-1} + 3\ b^{d-2} + \ldots + db.$$

This is asymptotically $O\ (b^d)$ if '*b'* is greater than one, since for large values of d the lower order terms become insignificant. In other words, most of the work goes into the final iteration, and the cost of the pervious iterations is relatively small. The ratio of the number of nodes generated by *DFID* to those generated by breadth-first search on a tree is approximately $b/(b-1)$. In fact, *DFID* is asymptotically optimal in terms of time and space among all brute-force shortest-path algorithms on a tree (Dillenburg and Nelson 1994).

If the edge costs differ from one another, then one can run an iterative deepening version of uniform-cost search, where the depth cutoff is replaced by a cutoff on the $g(n)$ cost of a node. At the end of each iteration, the threshold for the next iteration is

set to the minimum cost of all nodes generated on the previous iteration whose cost exceeded the previous threshold.

On a graph with cycles, however, breadth-first search may be much more efficient than any depth-first search. The reason is that a breadth-first search can check for duplicate nodes whereas a depth-first search cannot. Thus, the complexity of depth-first search depends on the number of paths of a given length. For example, in a square grid, the number of nodes within a radius *'r'* of the origin is $O(r^2)$, whereas the number of paths of length r is $O(3^r)$, since there are three children of every node, not counting its parent. Thus, in a graph with a large number of very short cycles, breadth-first search is preferable to depth-first search, if sufficient memory is available. Two approaches to the problem of pruning duplicate nodes in depth-first search are presented in Dillenburg and Nelson (1993) and Taylor and Körf (1993).

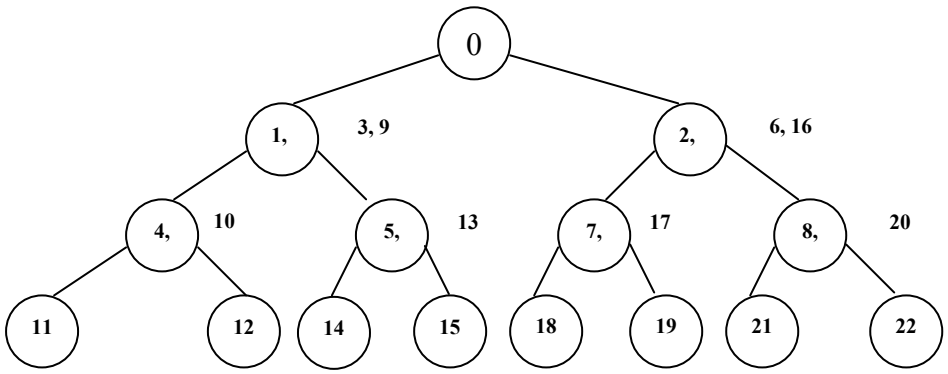

Figure 3.3 Order of Node Generation for Depth-First Iterative Deepening Search

Source: Körf (1998)

e. Bidirectional Search

Bidirectional search is a brute-force algorithm that requires an explicit goal state instead of simply a test for a goal condition (Pohl 1971). The main idea is to simultaneously search forward from the initial state, and backward from the goal state, until the two search frontiers meet. The path from the initial state is then

concatenated with the inverse of the path from the goal state to the complete solution path.

Bidirectional search still guarantees optimal solutions. Assuming that the comparisons for identifying a common state between the two frontiers can be done in constant time per node, by hashing for example, the time complexity of bidirectional search is $O(b^{d/2})$ since each search need only proceeds to half the solution depth. Since at least one of the searches must be breadth-first in order to find a common state, the space complexity of bidirectional search is also $O(b^{d/2})$. As a result, bidirectional search is space bound in practice.

f. Combinatorial Explosion

The problem with all brute-force search algorithms is that their time complexities grow exponentially with problem size. This is called combinatorial explosion, and as a result, the size of problems that can be solved with these techniques is quite limited.

2. Heuristic Search Tree

In order to solve larger problems, domain-specific knowledge must be added to improve the search efficiency. In Artificial Intelligence, heuristic search has a general meaning and a more specialized technical meaning. In a general sense, the term heuristic is used for any advice that is often effective, but is not guaranteed to work in every case. Within the heuristic search literature, however, the term heuristic usually refers to the special case of a heuristic evaluation functions.

a. Heuristic Evaluation Functions

In bound constrained optimization problem for example, the objective function is of maximization type, heuristic evaluation function estimates the cost as the upper bound of the objective function. The key properties of a heuristic function are that it

estimates actual cost, and that it is inexpensive to compute. In addition, most heuristic functions are derived from the original problem, for example, ratio of upper bound of the objective function to the range of objective function, and so on. Similarly, one can define heuristic function for constrained optimization and also for continuous constraint satisfaction problems.

A number of algorithms make use of heuristic functions, including pure heuristic search, the A* algorithm, iterative deepening-A*, depth-first branch and bound, and the heuristic path algorithm. In addition, heuristic information can be employed in bidirectional search as well.

b. Pure Heuristic Search

The simplest of these algorithms, pure heuristic search, expands nodes in order of their heuristic values $h(n)$ (Doran and Michie 1966).

Example:

For Bound Constrained Optimization Problem

$h(n)$ = Upper bound of the objective function,

For Continuous Constraint Satisfaction Problem

$h(n)$ = Sum of infeasibility over all constraints,

For Constrained Optimization problem

$h(n)$ = Upper bound of the objective function + Sum of infeasibility over all constraints

It maintains a closed list of those nodes that have already been expanded, and an open list of those nodes that have been generated but not yet expanded. The algorithm begins with just the initial state on the Open list. At each cycle, a node on the Open

list with the minimum or maximum (based on the type of problem) $h(n)$ value is expanded, generating all of its children, and is placed on the Closed list. The heuristic function is applied to the children, and they are placed on the Open list in order of their heuristic values. The algorithm continues until a goal state is chosen for expansion.

In a graph with cycles, multiple paths will be found to the same node, and the first path found may not be the shortest. When a shorter path to a closed node is found, the node is moved to open, and the shorter path is associated with it. The main drawback of pure heuristic search is that as it ignores the cost of the path so far to node n, it does not find optimal solutions.

Breadth-first search, uniform-cost search, and pure heuristic search are all special cases of a more general algorithm called best-first search. In each cycle of a best-first search, the node that is best according to some cost function is chosen for expansion. The best-first algorithms differ only in their cost functions: the depth of node n for breadth, cost of the path from initial state to node n ($g(n)$) for uniform-cost search, and $h(n)$ for pure heuristic search.

c. A* Algorithm

The A* algorithm (Hart et al. 1968) combines features of uniform-cost search and pure heuristic search to efficiently compute optimal solutions. A* is a best-first search in which the cost associated with a node is $f(n) = g(n) + h(n)$, where $g(n)$ is the cost of the path from the initial state to node n, and $h(n)$ is the heuristic estimate of the cost of a path from node n to a goal. Thus, $f(n)$ estimates the lowest total cost of any solution path going through node n. At each point a node with the lowest f value is chosen for expansion. Ties among nodes of equal f value should be broken in favor of nodes with

lower *h* values. The algorithm terminates when a goal node is chosen for expansion. The main drawback of A*, and indeed of any best-first search, is its memory requirement. Since at least the entire Open list must be saved, A* is severely space-limited in practice, and is no more practical than breadth-first search on current machines.

d. Iterative-Deepening-A*

Just as depth-first iterative-deepening solved the space problem of breadth-first search, iterative-deepening-A* (*IDA**) eliminates the memory constraint of A*, without sacrificing solution optimality (Körf 1985). Each iteration of the algorithm is a depth-first search that keeps track of the cost, $f(n) = g(n) + h(n)$, of each node generated. As soon as a node is generated whose cost exceeds a threshold for that iteration, its path is cut off, and the search backtracks before continuing. The cost threshold is initialized to the heuristic estimate of the initial state, and in each successive iteration it is increased to the total cost of the lowest-cost node that was pruned during the previous iteration. The algorithm terminates when a goal state is reached whose total cost does not exceed the current threshold.

Since *IDA** performs a series of depth-first searches, its memory requirement is linear with respect to the maximum search depth. In addition, if the heuristic function is admissible, *IDA** finds an optimal solution. Finally, by an argument similar to that presented for *DFID*, *IDA** finds an optimal solution. Moreover, *IDA** expands the same number of nodes, asymptotically, as A* on a tree, provided that the number of nodes grows exponentially with solution cost. These facts, together with the optimality of A*, imply that *IDA** is asymptotically optimal in time and space over all heuristic search algorithms that find optimal solutions on a tree. Additional benefits of

*IDA** are that it is much easier to implement, and often runs faster than A*, since it does not incur the overhead of managing the Open and Closed lists.

The other Heuristic Search algorithms are Depth-First Branch-and-Bound, Complexity of Finding Optimal Solutions, Heuristic Path Algorithm, and Recursive Best-First Search (Körf 1998).

3.2. Adaptive Tree Management

3.2.1. Introduction

The tree management system in the proposed *IP* maintains a stage-wise branching scheme that is conceptually similar to the **iterative deepening approach** (Körf 1985). The new adaptive tree management is applied in solving *CCSP* and *COP* problems.

This tree management comprises of two subunits such as best-first (in case of *CCSP*, worst-first is selected because we select the box with the maximum infeasibility degree) and restricted depth-first tree management systems. The best-first of the proposed tree management uses the following merit function to rank the boxes in the pending list.

In *CCSP*:

Merit function value = Sum of infeasibility over all constraints,

In *COP*:

Static penalty box ranking:

Merit function value = Upper bound of objective function + Sum of infeasibility over all constraints.

Non-penalty box ranking:

If a feasible solution (*CLB*) is not identified:

Merit function value = Sum of infeasibility over all constraints

If a feasible solution (*CLB*) is identified:

Merit function value = Upper bound of objective function

However, the depth-first unit of the proposed tree management utilizes the Total Area Deleted (*TAD*) by discarding boxes fails to improve in two consecutive partitioning iterations in this sub-tree.

Initial steps of the adaptive tree management procedure:

1. Box ranking

Rank the candidate boxes according to merit function defined as above.

2. Generating partial deeper tree levels

The new approach has the following feature related to the selection of child box to re-partition:

a. Once a parent box is selected for re-partitioning, the algorithm first assesses the child boxes before placing them in the current stage's candidate list.

b. It ranks child boxes according to the swapping dual criteria and then selects the first to re-partition.

c. Steps 2a and 2b are repeated until two consecutive re-partitioning steps do not result in at least one discarded box.

3. Maintaining stage-wise tree with two lists of candidate boxes and invoking local search

a. The new IP algorithm invokes FSQP in each unprocessed box once the algorithm fails to discard a box in two consecutive re-partitioning iterations.

b. If FSQP identifies a stationary point better than the *CLB* in any box, then the *CLB* is updated.

c. Boxes subjected to FSQP are placed back in the current candidate list (first candidate list).

d. After FSQP is invoked once in a given stage, all generated child boxes are placed in the candidate list of the next stage (second candidate list).

e. The candidate boxes in the next stage are not explored unless all boxes in the current stage's list is depleted.

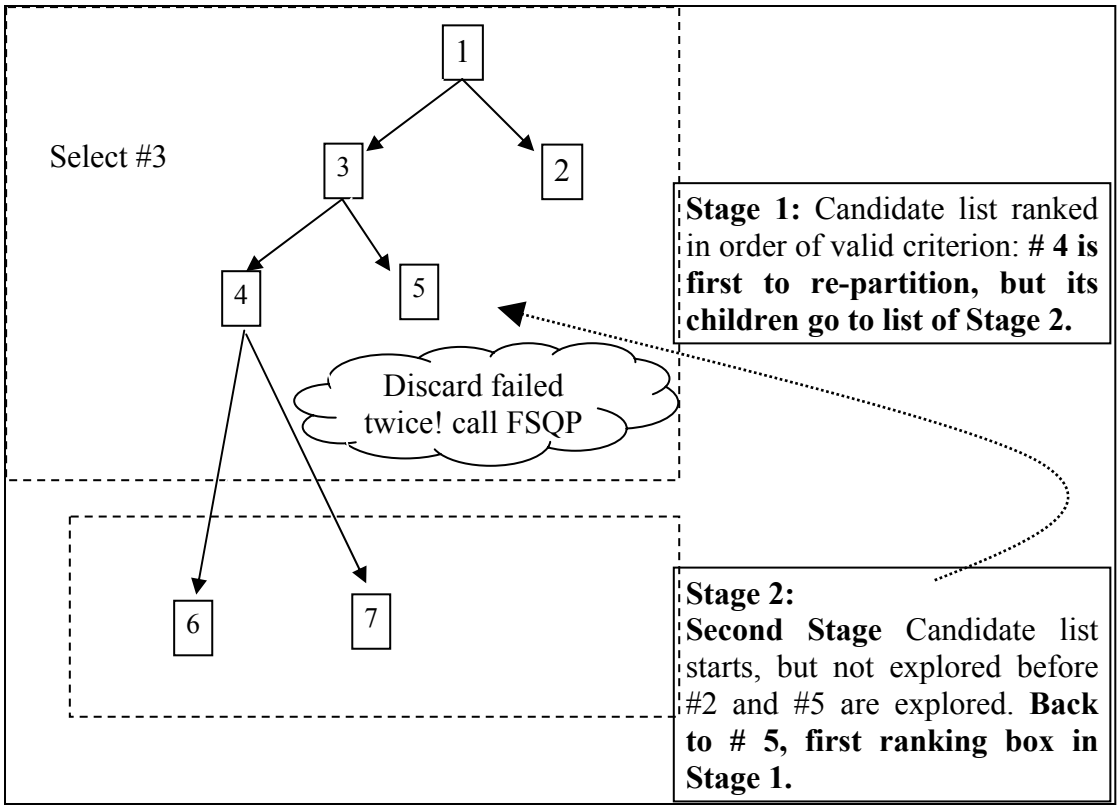

Figure 3.4. Initial steps of the adaptive iterative deepening procedure.

A more detailed description is presented in the following subsection 3.2.2.

3.2.2. Detailed description

The iterative deepening approach explores all nodes generated at a given tree level (stage) before it starts assessing the nodes at the next stage. Exploration of boxes at

the same stage can be done in any order, the sweep may start from best-first box or the one on the most right or most left of that stage. On the other hand, in the proposed adaptive tree management system, a node (parent box) at the current stage is permitted to grow a sub-tree forming partial succeeding tree levels and to explore nodes in this sub-tree before exhausting the nodes at the current stage. In the *COP*, if a feasible solution (*CLB*) is not identified yet boxes in the sub-tree are ranked according to ascending order of total feasibility uncertainty degree of a box criterion, otherwise they are ranked in descending order of upper bound of the objective function. In the *CCSP*, boxes are simply ranked according to descending order of total feasibility uncertainty degree of a box criterion and a box is selected among the children of the same parent according to maximum total feasibility uncertainty degree of a box (worst-first). In the *COP,* if a feasible solution is not identified yet a box is selected among the children of the same parent according to minimum total feasibility uncertainty degree of a box criterion, otherwise a box with maximum upper bound of the objective function. Then, the child box is partitioned again continuing to build the same sub-tree. This sub-tree grows until the *TAD* by discarding boxes fails to improve in two consecutive partitioning iterations in this sub-tree. Such failure triggers a call to local search where all boxes not previously subjected to local search are processed by the procedure *FSQP*, after which they are placed back in the list of pending boxes and exploration is resumed among nodes at the current stage. Feasible solutions found by *FSQP* are stored. If a box contains more than one feasible solution, then these are discovered in its child boxes.

The above adaptive tree management scheme is achieved by maintaining two lists of boxes, B_s and B_{s+1} that are the lists of boxes to be explored at the current stage s and the next stage $s+1$, respectively. Initially, the set of indeterminate boxes in the

pending list B_s consists of **X** only and B_{s+1} is empty. As child boxes are added to a selected parent box, they are ordered in descending order of merit function. Boxes in the sub-tree stemming from the selected parent at the current stage are explored and partitioned until there is no improvement in *TAD* in two consecutive partitioning iterations. At that point, partitioning of the selected parent box is stopped and all boxes that have not been processed by local search are sent to *FSQP* module and processed to identify feasible point solutions if *FSQP* is successful in doing so. From that moment onwards, child boxes generated from any other selected parent in B_s are stored in B_{s+1} irrespective of further calls to *FSQP* in the current stage. When all boxes in B_s have been assessed (discarded, stored as feasible boxes or partitioned), the search moves to the next stage, s+1, starting to explore the boxes stored in B_{s+1}. In this manner, a lesser number of boxes (those in the current stage) are maintained in primary memory and the search is allowed to go down to deeper levels within the same sub-tree, increasing the chances to discard boxes. On the other hand, by enabling the search to explore horizontally across boxes at the current stage, it might be possible to find feasible solutions faster by not partitioning parent boxes that are not so promising.

The tree continues to grow in this manner taking up the list of boxes of the next stage after the current stage's list of boxes is exhausted. The algorithm stops either when the theoretical number of feasible solutions are found (in equality problems) or CPU time reaches a given limit or when there are no boxes remaining in B_s and B_{s+1}. The proposed *IP* algorithm is described below.

IP with adaptive tree management

Step 0. Set tree stage, s=1 and future stage, r=1. Set non-improvement counter for *TAD*: nc=0. Set B_s, the list of pending boxes at stage s equal to **X**, B_s ={**X**}, and

$B_{s+1}=\varnothing$.

Step 1. *COP*: If the number of function evaluations or CPU time reaches a given limit, or, both $B_s=\varnothing$ and $B_{s+1}=\varnothing$, then stop.

CCSP: If the theoretical number of solutions is identified or CPU time reaches a given limit, or, both $B_s=\varnothing$ and $B_{s+1}=\varnothing$, then stop.

Else, if $B_s=\varnothing$ and $B_{s+1}\neq\varnothing$, then set $s\leftarrow s+1$, set $r\leftarrow s$, and continue.

Pick first box **Y** in B_s and continue.

1.1 If **Y** is infeasible or suboptimal (in the *COP*), discard **Y**, and go to Step 1.

1.2 If **Y** is sufficiently small, evaluate m, its mid-point, and if it is a feasible improved solution, update *CLB,* re-set $nc \leftarrow 0$, and store m. Remove **Y** from B_s and go to Step 1. (In the *CCSP*, omit the *CLB* update and store the feasible solution.)

1.3 Else go to Step 2.

Step 2. Select variable(s) to partition (use the subdivision direction selection rule *IIR*). Set v = number of variables to partition.

Step 3. Partition **Y** into 2^v non-overlapping child boxes. Check *TAD*, if it improves, then re-set $nc \leftarrow 0$, else set $nc \leftarrow nc + 1$.

Step 4. Remove **Y** from B_s, add 2^v boxes to B_r.

4.1. If $nc > 2$, apply *FSQP* to all (previously unprocessed by *FSQP*) boxes in B_s and B_{s+1}, re-set $nc\leftarrow 0$. If *FSQP* is called for the first time in stage s, then set $r\leftarrow s+1$. Go to Step 1.

4.2. Else, go to Step 1. ■

The adaptive tree management system in *IP* is illustrated in Figure 3.4 on a small tree where node labels indicate the order of nodes visited. The nodes 3, 5, 6, 8, 9, 11, 12, 13, 14, 15, 16 are in Stage 1's list and should be explored before moving to Stage 2. All their children are placed in Stage 2's list after the first *FSQP* call in Stage 1. There might be more than one *FSQP* calls in Stage 1, this does not affect the placement of the children.

The adaptive tree management strategy proposed here can also be used in non-interval partitioning algorithms such as *Baron* and *LGO*. It is effective in the sense that it allows going deeper into selected promising parent boxes while providing a larger perspective on how promising a parent box is by comparing it to all other boxes available in the current stage's box list.

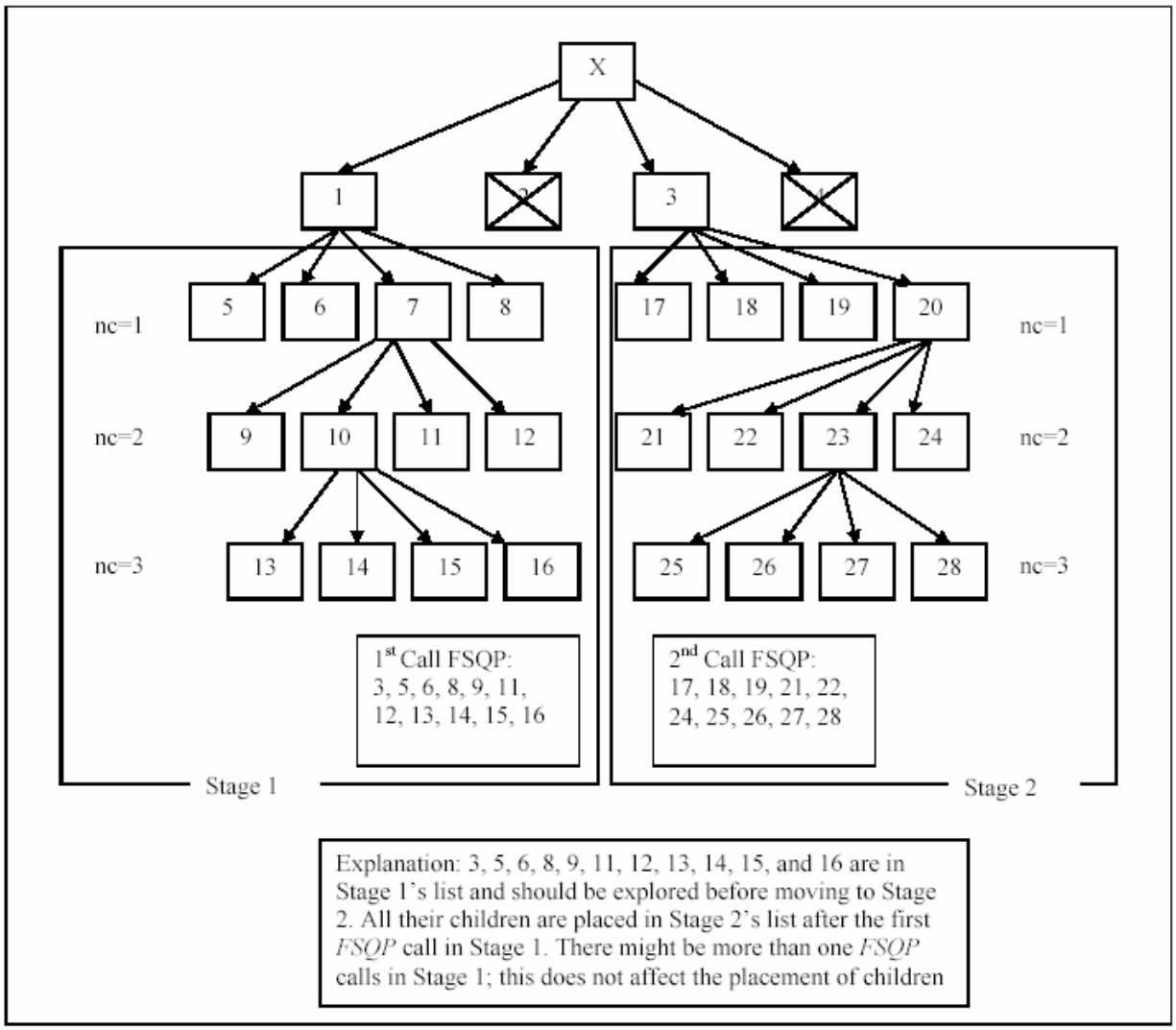

Figure 3.5. Implementation of the adaptive iterative deepening procedure.

Chapter 4

Interval Inference for Global Optimization

This chapter presents the details on Interval Inference Rule (*IIR*) for solving global optimization problems including *BCOP*, *CCSP* and *COP*. It also presents the details on Interval Partitioning Algorithms (*IPA*) for *BCOP*, *CCSP* and *COP* and different subdivision direction rules such as *Rule A*, *Rule B*, *Rule C*, *Rule D*, and *Smear Rule*.

4.1. Interval Partitioning Algorithms

Interval Partitioning Algorithms (*IPA*) use interval arithmetic (Moore 1966) to produce reliable results for constrained and bound constrained optimization (Hansen 1992, Ratschek and Rokne 1995). Due to their reliability, interval applications take place in a wide scope of scientific fields (Kearfott and Kreinovich 1996). In bound constrained global optimization problems, an *IPA* subdivides the given domain into smaller subspaces or boxes that are assessed according to their function range calculated by using an approximating inclusion function. Based on the function range bounds and a known best solution that is updated during the search, some subspaces are deleted reliably, because they cannot hold at the global optimum solution (Pinter 1992, Hammer et al. 1993). Subdivision continues in remaining boxes so that the location of the global optimum solution can be enclosed within a small box of a given tolerance. The final report contains all such boxes in the given function domain.

The prototype interval branch and bound algorithm for solving (1.1) is as follows:

Step 1: $\mathbf{Y} \leftarrow \mathbf{X_0}$, initialize the empty list L_w

Step 2: Choose the coordinate directions (i.e., *Rule A*, *B*, *C*, *D* and *E*)for the splitting of $\mathbf{Y}$.

Step 3: Split **Y** normal to the chosen directions, cutting the box a given number of times in each direction. Let $\mathbf{Y}_1,\ldots,\mathbf{Y}_s$ be the subboxes obtained.

Step 4: For i = 1 to *s* do

4.1 : Delete $\mathbf{Y}_i$ if it can be proven that $\mathbf{Y}_i$ contains no optimal solution or diminish $\mathbf{Y}_i$ if it can be proven that the respective part of $\mathbf{Y}_i$ contains no optimizer point.

4.2 : If $\mathbf{Y}_i$ is not deleted**,** then store it (as a whole or diminshed) into working list L_w.

Step 5: Choose a box from L_w and remove it from the working list. Let **Y** denote the chosen box.

Step 6: While termination criterion does not hold go to Step 2.

In Step 2 of IPA, subdivision direction selection step, subdivision rules proposed up to date are based on criteria such as the width of variable intervals (Rules A and D), or estimated function improvement by selected variables (gradient information such as Rules B, C and E). The performance of such rules is assessed extensively on standard test problems (Ratz and Csendes 1995, Csendes and Ratz 1996, 1997, Csendes et al. 2000) resulting in the general conclusion that gradient based rules work much better.

1. Rule A

This *Rule A* selects the subdivision direction based on the interval–width (Moore 1966, Ratschek and Rokne 1988, Ratz and Csendes 1995, Ratz 1996, Berner 1996, Csendes and Ratz 1996, 1997, Csendes et al. 2000). According to this rule, the coordinate direction with maximum $D(i)$ value will be selected, and the $D(\mathrm{X_i})$ can be defined as in equation (4.1):

$$D(\mathrm{X_i}) := w(\mathrm{X_i}) \qquad (4.1)$$

where, X_i is the variable vector

It is found that an algorithm with *Rule A* is convergent both with and without the monotonicity test (Ratschek and Rokne 1988, Ratz and Csendes 1995, Csendes and Ratz 1997, Csendes et al. 2000).

2. Rule B

Hansen (1992) proposes the *Rule B*, with initiation from G. W. Walster (Ratz and Csendes 1995, Ratz 1996, Berner 1996, Csendes and Ratz 1996, 1997, Csendes et al. 2000). According to this rule, a coordinate direction with maximum $D(i)$ is chosen for subdivision. However, the $D(i)$ can be defined as in equation (4.2):

$$D(\mathrm{i}) := w(F_i'(X_i)) * w(X_i) \tag{4.2}$$

where $F_i'(X_i)$ is the first order derivative with respect to variable X_i.

It is also found that this subdivision direction rule can also be carried out for many directions in a single iteration step (Csendes and Ratz 1997). This rule becomes nonconvergent if the monotonicity test is removed from the algorithm (Csendes and Ratz 1997, Csendes et al. 2000).

3. Rule C

Ratz proposes the *Rule C* (Ratz and Csendes 1995, Ratz 1996, Berner 1996, Csendes and Ratz 1996, 1997, Csendes et al. 2000), with an underlying idea of minimizing the width of the inclusion function. According to this rule, a coordinate direction with maximum $D(i)$ is chosen for subdivision. However, the $D(i)$ can be defined as in equation (4.3):

$$D(\mathrm{i}) := w\,(F_i'(X_i) * (X_i - m(X_i))) \tag{4.3}$$

where $F_i'(X_i)$ is the first order derivative with respect to variable x_i.

However, this formulation shows the model algorithm with the direction selection rule can be related to Lipschitzian partition methods for global optimization (Pinter 1986, 1992).

4. Rule D

Rule D is a derivative free method such as *Rule A*, and reflects the machine representation of the inclusion function $F(\mathrm{X})$ (Hammer et al. 1993, Ratz and Csendes 1995, Ratz 1996, Berner 1996, Csendes and Ratz 1996, 1997, Csendes et al. 2000). According to this rule, the coordinate direction with maximum $D(i)$ will be selected for subdivision from the expression (4.4).

$$\mathrm{D(i)} := \begin{cases} w(\mathrm{X_i}) & \text{if } 0 \in \mathrm{X_i} \\ \\ w(\mathrm{X_i}) \,/\, \min\{|x_i|;\, x_i \in \mathrm{X_i}\} & \text{otherwise} \end{cases} \tag{4.4}$$

5. Rule E

Ratz proposes *Rule E* (Ratz and Csendes 1995, Ratz 1996, Berner 1996, Csendes and Ratz 1996, 1997, Csendes et al. 2000,) similar to *Rule C*, with an underlying idea of minimizing the width of the inclusion function. According to this rule, a coordinate direction with maximum $D(i)$ is chosen for subdivision and the $D(i)$ can be defined as in equation (4.5):

$$D(\mathrm{i}) := w\,((\mathrm{X_i} - m(\mathrm{X_i}))\,(F_\mathrm{i}'(m(\mathbf{X}))+0.5 * \sum_{j=1}^{n}(F_{ij}''(\mathbf{X})\,(\mathrm{X_j} - m(\mathrm{X_j}))\,)) \tag{4.5}$$

where $F_\mathrm{i}'(m(\mathrm{X_i}))$ is the first order derivative with respect to variable midpoint of $\mathrm{X_i}$, and

$F_\mathrm{i}''(\mathbf{X})$ is the second order derivative with respect to variable x.

Kearfott and Manuel (1990) propose a new rule known as *smear rule* for finding the roots of nonlinear systems of equations. According to this rule, the variable with the largest rate of change (the absolute value of the Jacobian element) multiplied by the width of its domain is selected.

The coordinate of maximum smear is defined to be k such that $s_k = \max_{1 \le j \le n} s_j$, where s_j equals to $\max_{1 \le j \le n} \{|\boldsymbol{A}_{i,j}|\} w(X_j)$.

In Step 3 of *IPA,* multi-section of a selected box step, several strategies can be applied, such as *k-best* strategy; subdivision of a single variable's width into s > 2 pieces (Csallner et al. 2000a, 2000b, Casado et al. 2001a, 2001b).

In Step 5 of *IPA*, the box selection step, several strategies can be applied, such as, select the box with best upper bound (Skelboe 1974, Ratschek and Rokne 1988); select the box which has been longest in the list (Hansen and Sengupta 1980, Hansen 1992); and select the box which has maximum width (Hansen and Sengupta1980), and so on.

In Step 6 of *IPA* (Termination criterion), termination criterion also plays an important role in *IPA* to obtain solutions which are close to the actual solutions. Kearfott and Walster (2000) introduce a new termination criterion, i.e., thickness stopping criterion, which can be used for global optimization algorithms using interval analysis. The other stopping criteria are a heuristic domain and range stopping criteria, which is used to determine the accuracy tolerances (Moore 1966, Neumaier 1990, Hansen 1992, Ratscheck and Rokne 1995).

Interval Constraints for *CCSP*

The general scheme for solving the interval constraint is as follows. Interval techniques for solving *CCSP* are based on Branch and Prune / Splitting and Filtering

approaches:

- branching consists of splitting the search space into smaller parts and therefore easier to handle;
- pruning consists of filtering the current box to remove inconsistent elements.

Splitting is generally carried out by bisecting the domain(s) of the selected variable(s). It mostly results in two new domains to filter. In some cases though, it may be more efficient to split the current filtered domain in more than two smaller boxes (Chabert 2005). The bisection stage results in the creation of so-called child boxes. Variable selection is made according to different heuristics, such as choosing the variable with the domain of largest width (usually referred to as "largest first" or *Rule A*), or choosing the next variable on the pre-established sequence of variables ("round-robin"), or even choosing the variable with the largest rate of change (i.e., the absolute value of the Jacobian element) multiplied by the width of its domain ("*Smear rule*" by Kearfott and Manuel 1990).

Pruning / Filtering is performed by using consistency techniques. Basically consistency techniques check the satisfaction of the constraints. If the test is negative then the current box is discarded. A very naive version of such a technique is the following: suppose you want to check the consistency of an equality constraint of the form $c : f(x)=0$ over some box **X**. If the interval evaluation of f over **X** does not contain 0, then you can immediately conclude that c is inconsistent (i.e., is not feasible) over **X**, and you can discard **X**. Convergence of this method is very slow for it totally relies on interval evaluations that are known to overestimate ranges of functions.

4.2. Basic Terminologies and Definitions

Definition 4.1 (Continuous real constraint):

A continuous real constraint *is an atomic formula made of expressions and relations symbols, such as equality or an inequality*.

Let *c* be a real continuous constraint defined over **X** included in $\mathbb{R}^n$. Let ρ_c be the subset of $\mathbb{R}^n$ satisfying *c*. Then any element *s* of **X** intersected with ρ_c is called a *solution of c*. Such elements are also called *consistent* elements for *c*.

As a result, a constraint divides its domain of definition **X** into two distinct subsets: the subset of consistent elements and the subset of inconsistent elements ($\mathbf{X}/ (\mathbb{R}^n \cap \rho_c)$).

Definition 4.2 (Constraint system):

A constraint system *is the conjunction of a set of constraints* $C=\{c_1,..., c_p\}$ *defined over a set of variables* $V=\{ x_1,..., x_n \}$, *each variable* x_i *defined over a given domain* X_i *of* $\mathbb{R}$.

A constraint system is denoted by S= (V, C, **X**), where $\mathbf{X} = X_1 \times \ldots \times X_n$, and the corresponding problem to be solved is called a *CCSP*, usually referred to as *CSP*. Let us note that, when this is not ambiguous, we may refer to a *CCSP*, S= (V, C, **X**), simply as C.

The *solution set of* S is denoted ρ_S. A solution of S is a n-tuple $s \in \mathbf{X}$ such that, for all constraints $c_i \in C$, the restriction of s to the variables V_{c_i} of c_i belongs to $\rho_{c_i} \cap x_{j \in V_{c_i}} X_j$

Definition 4.3 (Interval Constraint):

An interval constraint is built from an atomic interval formula (interval function) and relation symbols, whose semantics are extended to intervals as well. ■

A constraint is being defined by its expression (atomic formula and relation symbol), variables, and their domains. Moreover, it is considered that an interval constraint has interval variables (variables that take interval values), and that each associated domain is an interval.

The main guarantee of interval constraint is that if its solution set is empty, it has no solution over a given box **Y**; then it follows that the solution set of the COP is also empty and box **Y** can be reliably discarded. In a similar manner, if the upper bound of the objective function range, $\overline{F}(\mathbf{Y})$, over a given box **Y** is less than or equal to the objective function value of a known feasible solution (the Current Lower Bound, *CLB*), then **Y** can be reliably discarded since it cannot contain a better solution than the *CLB*.

Below we formally provide the conditions where a given box **Y** can be discarded reliably based on the ranges of interval constraints and the objective function.

In a partitioning algorithm, each box **Y** is assessed for its optimality and feasibility status by calculating the ranges for *F*, *G* and *H* over the domain of **Y**.

Definition 4.4 (Indeterminate box with regard to optimality):

If $\underline{F}(\mathbf{Y}) \leq CLB$ *and* $\overline{F}(\mathbf{Y}) > CLB$, *then* **Y** *is called an* <u>*indeterminate*</u> *box with regard to optimality. Such a box holds the potential of containing x* if it is not an infeasible box.* ■

A box is called as an indeterminate box with regard to optimality when its objective function lower bound is less then or equal to the *CLB* and the upper bound is greater then *CLB*.

Definition 4.5 (Indeterminate box with regard to feasibility):

If ($\underline{G_i}(\mathbf{Y}) < 0$ AND $\overline{G_i}(\mathbf{Y}) > 0$) OR ($0 \in H_i(\mathbf{Y}) \neq 0$) *for any i, and other constraints are consistent over* **Y**, *then* **Y** *is called an indeterminate box with regard to feasibility and it holds the potential of containing x* if it is not a sub-optimal box.* ■

Definition 4.6 (Cut-off test based on optimality):

If $\overline{F}(\mathbf{Y})$ < Current Lower Bound (*CLB*), *then box* **Y** *is called a suboptimal box and it is deleted because it cannot contain x*.*

Definition 4.7 (Cut-off test based on feasibility):

If $\underline{G_i}(\mathbf{Y}) > 0$ *or* $0 \notin H_i(\mathbf{Y})$ *for any i, then box* **Y** *is called an infeasible box and it is discarded.*

Definition 4.8 (Uncertainty of an indeterminate box with regard to optimality):

The degree of uncertainty of an indeterminate box with respect to optimality is defined as:

$$PF_Y = \overline{F}(\mathbf{Y}) - CLB \qquad (4.6)$$

Definition 4.9 (Uncertainty of an indeterminate box with regard to feasibility):

The degree of uncertainty, $PG^i_{\mathbf{Y}}$ *(*$PH^i_{\mathbf{Y}}$*) of an indeterminate inequality (equality) constraint with regard to feasibility is defined by equations (4.7) and (4.8), respectively.*■

$$PG^{i}_{\mathbf{Y}} = \overline{G_i}(\mathbf{Y}) \tag{4.7}$$

$$PH^{i}_{\mathbf{Y}} = \overline{H_i}(\mathbf{Y}) + |\underline{H_i}(\mathbf{Y})| \tag{4.8}$$

Definition 4.10 (Total feasibility uncertainty degree of a box):

The total feasibility uncertainty degree of a box, $INF_{\mathbf{Y}}$*, is the sum of uncertainty degrees of equalities and inequalities that are indeterminate over* **Y**. ■

Definition 4.11 (Feasible box):

If $\overline{G_i}(\mathbf{Y}) \leq 0$, *and* $0 \in H_i(\mathbf{Y}) = 0$, *for* ∀i, *then box* **Y** *is a* feasible *box*.

4.3. New Interval Partitioning Algorithms

4.3.1. Bound Constrained Optimization Problems

In each box assessment, the function range estimate F (**M**) over a sufficiently small box **M** enclosing the mid-point (m(**Y**)) is calculated. In the assessment of the first box, *min* f(**M**) becomes the current lower bound (*CLB*) and each time a better mid-point solution is found, *CLB* is updated.

IPA continues to subdivide available pending boxes until either they are all deleted or interval sizes of all variables in existing boxes are less than a given tolerance, δ. All such boxes are reported that may contain x*. In Figure 4.1, a generic pseudocode is provided for *IPA*.

Notation :

WLB: Working List of Boxes; **M** : Point interval at the mid-point of a box;

$F(\mathbf{M})$: range estimate at M ; δ : tolerance for final interval length

Void *IPA*:

{

Construct tree structure for $f(x)$;

Initialize: initial box = $\mathbb{I}(\mathbf{X})$; $CLB = -\infty$; $WLB = \mathbb{I}(\mathbf{X})$;

While $WLB \neq \phi$ do

{

Select a box $\mathbf{Y} \in WLB$; Calculate $F(\mathbf{Y})$;

if ($\bar{F}(\mathbf{Y}) > CLB$) AND (At least for one variable interval, $w(x_i) > \delta$)

{

if ($\underline{F}(\mathbf{Y}) > CLB$), then CLB = $\underline{F}(\mathbf{Y})$;

Calculate the mid-point function value, F(M);

if ($\underline{F}(\mathbf{M}) > CLB$), then $CLB = \underline{F}(\mathbf{M})$;

Select subdivision direction; // *Activate Symbolic Interval Inference Rule*;

Subdivide **Y** to obtain four sibling boxes: S_1, S_2, S_3, S_4; // *Multisection - 4 siblings*

$WLB = WLB - \{\mathbf{Y}\}$; $WLB = WLB + \{S_1, S_2, S_3, S_4\}$;

} // *endif*

else

{

if ($w(x_i) < \delta, \forall i$), then store **Y**; $WLB = WLB - \{\mathbf{Y}\}$;

}

} // *endwhile*

Report all stored boxes;

} // *endprocedure*

Figure 4.1. Generic pseudocode for *IPA*.

In essence, *IPA* aims to discard suboptimal boxes and reduce the number of pending boxes with as few function calls as possible. This is facilitated by partitioning appropriate variables and generating sub boxes whose overestimation in P_Y is reduced. Then, the algorithm converges fast by discarding suboptimal boxes early and also by

partitioning promising boxes in a fitting direction to reach the global basin of attraction. While variable selection is made according to this criterion, box selection is carried out following a worst-first strategy, i.e., the box with the maximum P_Y is selected first.

We would like to mention that P_Y is a traditional box selection index used in *IPA*. A normalized version of this index (the *Reject Index*) is obtained by dividing P_Y by w ($F(\mathbf{Y})$) (Casado et al. 2001a. 2001b). The *Reject Index* aims at reducing the overestimation in smaller boxes with greater uncertainty whereas we target at discarding boxes as large as possible.

4.3.2. Continuous Constraint Satisfaction Problems and Constrained Optimization Problems

IP is a reliable convergent algorithm that sub-divides indeterminate boxes to reduce INF_Y and PF_Y by nested partitioning. The contraction and the α-convergence properties enable the reduction in the uncertainty levels. The reduction in the uncertainty levels of boxes finally lead to their elimination due to sub-optimality or infeasibility while helping *IP* in ranking remaining boxes in a better fashion.

In the *COP*, a box that becomes feasible after nested partitioning still has uncertainty with regard to optimality unless it is proven that it is sub-optimal. The convergence rate of *IP* might be very slow if we require nested partitioning to reduce a box to a point interval that is the global optimum. Hence, since a box with a high PF_Y holds the promise of containing the global optimum, we propose to use a local search procedure that can identify stationary points in such boxes. In a similar fashion, we use the local search procedure to identify feasible solutions in indeterminate (with regard to feasibility) boxes in the *CCSP*.

In the *COP*, *IP* continues to subdivide available indeterminate and feasible boxes until either they are all deleted or interval sizes of all variables in existing boxes are less than a given tolerance, δ. Termination can also be forced by limiting the number of function evaluations and/or CPU time. Here, we choose to terminate *IP* when the number of function calls outside the local search procedure reaches a given limit or when the CPU time exceeds the maximum allowable time. In the *CCSP*, *IP* can terminate either when there are no more indeterminate boxes or when the theoretical number of feasible solutions is identified. Here, we stop when the number of feasible solutions identified reaches the theoretically known number of solutions or when a CPU time limit is reached.

In the following, we describe our proposed *IP* that has a flexible stage-wise tree management feature. This stage-wise tree also enables us to apply the best-first box selection rule within a restricted sub-tree (economizing memory usage) as well as to invoke local search in a set of boxes.

The tree management system in the proposed *IP* maintains a stage-wise branching scheme that is conceptually similar to the iterative deepening approach (Körf 1985). The iterative deepening approach explores all nodes generated at a given tree level (stage) before it starts assessing the nodes at the next stage. Exploration of boxes at the same stage can be done in any order, the sweep may start from best-first box or the one on the most right or most left of that stage. On the other hand, in the proposed adaptive tree management system, a node (parent box) at the current stage is permitted to grow a sub-tree forming partial succeeding tree levels and to explore nodes in this sub-tree before exhausting the nodes at the current stage. In the *COP*, if a feasible solution (*CLB*) is not identified yet, boxes in the sub-tree are ranked according to ascending INF_Y, otherwise they are ranked in descending order of $\overline{F}(\mathbf{Y})$.

In the *CCSP*, boxes are simply ranked according to descending INF_Y. A box is selected among the children of the same parent according to either box selection criterion, and the child box is partitioned again continuing to build the same sub-tree. This sub-tree grows until the Total Area Deleted (*TAD*) by discarding boxes fails to improve in the two consecutive partitioning iterations in this sub-tree. Such failure triggers a call to local search where all boxes not previously subjected to local search are processed by the procedure Feasible Sequential Quadratic programming (*FSQP*) (Zhou and Tits 1996, Lawrence et al. 1997). The boxes that have undergone local search are placed back in the list of pending boxes and exploration is resumed among the nodes at the current stage. In the *COP*, feasible and improving solutions found by *FSQP* are stored (that is, if feasible solutions with a better objective function value is found, *CLB* is updated and the solution is stored). In the *CCSP*, all feasible point solutions and sub-spaces are stored.

The above adaptive tree management scheme is achieved by maintaining two lists of boxes, B_s and B_{s+1} that are the lists of boxes to be explored at the current stage s and at the next stage $s+1$, respectively. Initially, the set of indeterminate or feasible boxes in the pending list B_s consists only of $\mathbf{X}$ and B_{s+1} are empty. As child boxes are added to a selected parent box, they are ordered according to the current ranking criterion. Boxes in the sub-tree stemming from the selected parent at the current stage are explored and partitioned until there is no improvement in *TAD* in two consecutive partitioning iterations. At that point, partitioning of the selected parent box is stopped and all boxes that have not been processed by local search are sent to *FSQP* module and processed to identify feasible and improving point solutions if *FSQP* is successful in doing so. It is noted that, whether or not *FSQP* fails to find an improving solution, *IP* will continue to partition the box since it passes both cutoff tests as long as it has a

potential to contain an improving solution. Finally, the algorithm encloses potential improving solutions in sufficiently small boxes where *FSQP* can identify them. Thus, *FSQP* acts as a catalyst that occasionally scans larger boxes to identify improving solutions at the earlier stages of the search. From that moment onwards, child boxes generated from any other selected parent in B_s are stored in B_{s+1} irrespective of further calls to *FSQP* in the current stage. When all boxes in B_s have been assessed (discarded or partitioned), the search moves to the next stage, s+1, starting to explore the boxes stored in B_{s+1}.

In this manner, a lesser number of boxes (those in the current stage) are maintained in primary memory and the search is allowed to go down to deeper levels within the same stage, increasing the chances to discard boxes or identify stationary points. On the other hand, by enabling the search to also explore boxes horizontally across at the current stage, it might be possible to find feasible improving solutions faster by not partitioning parent boxes that are not so promising (because we are able to observe a larger number of boxes).

The tree continues to grow in this manner taking up the list of boxes of the next stage after the current stage's list of boxes is exhausted. The algorithm stops when the stopping criteria mentioned above for the *COP/ CCSP* are satisfied. The proposed *IP* algorithm is described below:

IP with adaptive tree management

Step 0. Set tree stage, s=1, and future stage, r=1. Set non-improvement counter for *TAD*: nc=0. Set B_s, the list of pending boxes at stage s equal to **X**, B_s ={**X**}, and B_{s+1}=∅.

Step 1. COP: If the number of function evaluations or CPU time reaches a given

limit, or, both $B_s=\varnothing$ and $B_{s+1}=\varnothing$, then stop.

CCSP: If the theoretical number of solutions is identified or CPU time reaches a given limit, or, both $B_s=\varnothing$ and $B_{s+1}=\varnothing$, then stop.

Else, if $B_s=\varnothing$ and $B_{s+1}\neq\varnothing$, then set $s\leftarrow s+1$, set $r\leftarrow s$, and continue.

Pick first box **Y** in B_s and continue.

1.1 If **Y** is infeasible or suboptimal (in the *COP*), discard **Y**, and go to Step 1.

1.2 If **Y** is sufficiently small, evaluate *m*, its mid-point, and if it is a feasible improving solution, update *CLB,* re-set $nc \leftarrow 0$, and store *m*. Remove **Y** from B_s and go to Step 1. (In the *CCSP*, omit the *CLB* update and store the feasible solution.)

1.3 Else go to Step 2.

Step 2. Select variable(s) to partition (use the subdivision direction selection rule *IIR*). Set *v* = number of variables to partition.

Step 3. Partition **Y** into 2^v non-overlapping child boxes. Check *TAD*, if it improves, then re-set $nc \leftarrow 0$, else set $nc \leftarrow nc + 1$.

Step 4. Remove **Y** from B_s, add 2^v boxes to B_r.

4.1. If $nc > 2$, apply *FSQP* to all (previously unprocessed by *FSQP*) boxes in B_s and B_{s+1}, re-set $nc\leftarrow 0$. If *FSQP* is called for the first time in stage *s*, then set $r\leftarrow s+1$. Go to Step 1.

4.2. Else, go to Step 1. ■

The adaptive tree management system in *IP* is illustrated in Figure 3.4 on a small tree where node labels indicate the order of nodes visited.

4.4. Framework of Interval Inference Rule (*IIR*)

4.4.1. General Overview

The order in which variable domains are partitioned has an impact on the performance of *IP*. In general, variable selection is made according to widest variable domain rule or largest function rate of change in the box. Here, we develop a new numerical subdivision direction selection rule, *Interval Inference Rule* (*IIR*), to improve *IP*'s performance by partitioning in parallel, those variable domains that reduce PF_Y and INF_Y in at least one immediate child box. (The related illustration of the latter reduction and exceptional situations where such reduction may not be achieved are found in the section 4.6 of this chapter.) Hence, new boxes formed with an appropriate partitioning sequence result in diminished uncertainty caused by overestimation. Before *IIR* is applied, the objective *f* and each constraint *g* and *h* are interpreted as binary trees that represent recursive sub-expressions hierarchically. Such binary trees enable interval propagation over all sub-expressions of the constraints and the objective function (Benhamou et al. 1994). Interval propagation and function trees are used by Kearfott (1991) in improving interval Newton approach by decomposition and variable expansion, by Smith and Pantelides (1999) in automated problem reformulation, by Sahinidis (2003) and by Talawarmani and Sahinidis (2004) where feasibility based range reduction is achieved by tightening variable bounds.

After interval propagation is carried out over the sub-expressions in a binary tree, *IIR* traverses this tree to label its nodes so as to identify the pair of variables (source variables) that are most influential on the constraint's or the objective's uncertainty degree. This pair of variables are identified for each constraint and the objective

function, and placed in the pool of variables whose domains will be possibly partitioned in the next iteration. In the *COP*, we make sure that the pool at least contains the source variables for the objective function and therefore, the number of variables to be bisected in parallel is at least two. The total pool resulting from the traversal of *f*, *g* and *h* is screened and its size is reduced by allocating weights to variables and re-assessing them.

4.4.2. Interval Inference Rule

The *IIR* Framework comprises of six basic components such as Parser, Interval Inference Rule Base, *IA* partitioning coordinator, Interval Arithmetic Library, Local Search Methods, and Branching Master. The integrated framework of *IIR* is illustrated in Figure 4.2 and the detailed description of each component is given as follows.

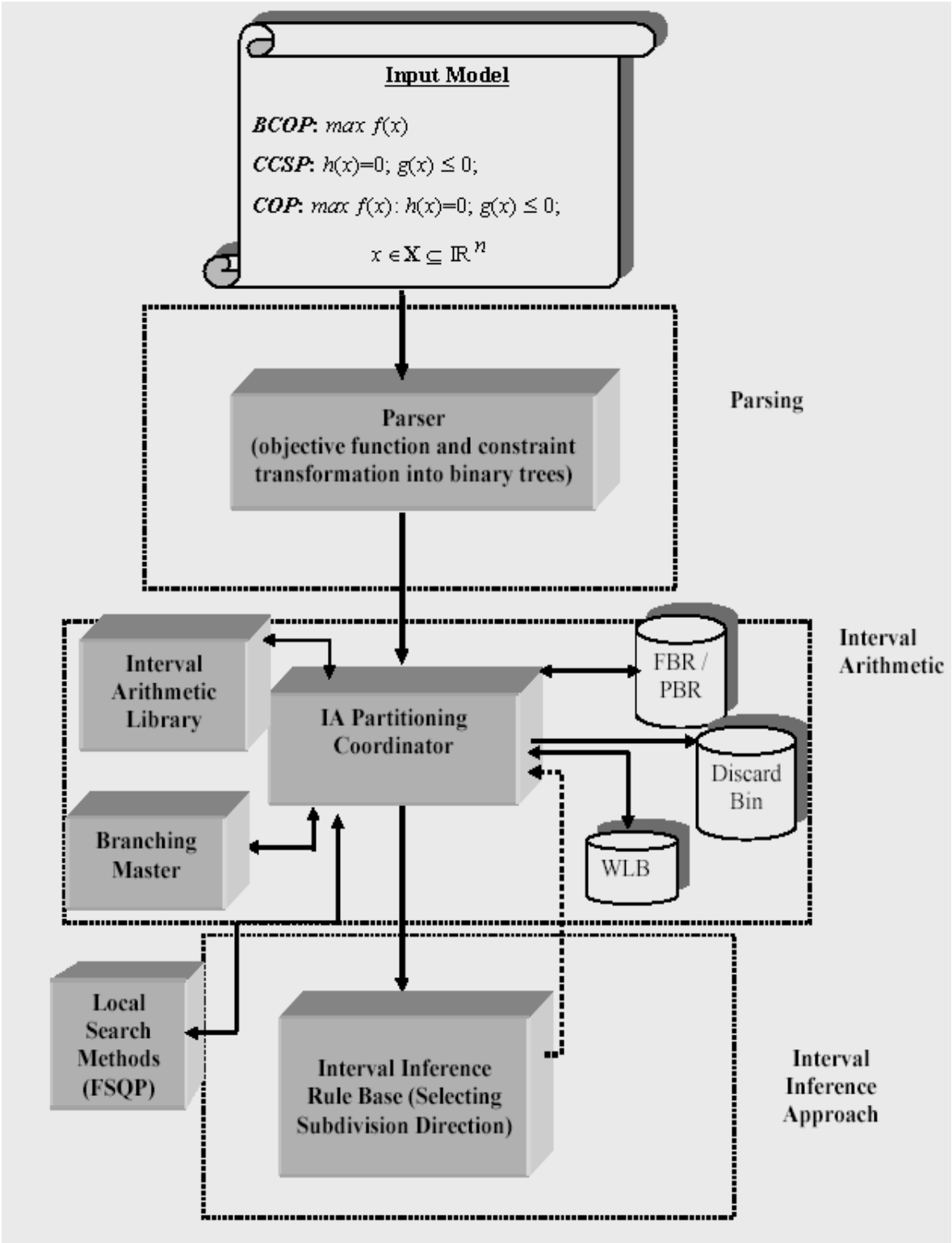

Figure 4.2. Integrated Framework of Interval Inference Rule

1. Parser

The parser is activated once before *IPA* is executed. It dissects the function expression and passes the output to the tree builder. The parser comprises of three sub-components: Expression parser, Expression translator, and Binary tree builder shown in Figure 4.3.

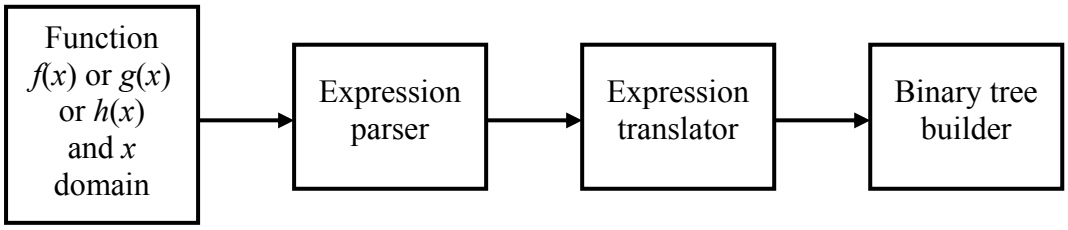

Figure 4.3. Sequence of parser functions

i. Expression Parser parses a given infix expression into machine understandable format as array of strings. It converts variable domains ($x = [\underline{X}, \overline{X}]$) into VarName ($x$), and VarValues **(**$[\underline{X}, \overline{X}]$). Expression parser also parses each constraint into three parts that are constraint expression, type of constraint (direction of inequality or equality), and the right hand side constant. The objective function is parsed into two parts, type of objective function, i.e., minimization (*min*) or maximization (*max*), and objective function expression.

ii. Expression Translator converts infix expressions obtained from **Expression parser** into post-fix expressions and passes the post-fix expressions to the tree builder.

iii. Binary Tree Builder

The binary tree builder constructs a binary tree using the information obtained from the parser. A binary tree that represents the function with all its subexpressions is then constructed. The contribution of subexpressions and atomic elements (variables) to

the function range are recursively calculated by calling an *Interval Library* at each (molecular) level of the hierarchical binary tree so that the impact of all terms can be assessed in descending order of complexity.

2. Interval Inference Rule Base

Interval Inference Rule Base is designed for identifying the variables to be re-partitioned for each constraint and objective functions whichever is applicable for a given optimization problem. The identified variables will be returned to *IA* partitioning coordinator for further processing. It comprises of two basic procedures: Interval propagation over a binary tree and *IIR* labeling procedure.

a. Interval Propagation over a Binary Tree

Before the labeling process can be applied on a constraint expression, it has to be parsed and converted into a binary tree where intervals at sub-expression levels are calculated in a bottom to top fashion starting from atomic levels (variables or constants).

A binary tree representing a constraint is built as follows. Leaves of the binary tree are atomic elements, i.e., they are either variables or constants. All other nodes represent binary expressions of the form (Left Θ Right). A binary operator is an arithmetic operator (*, +, -, /) having two branches ("Left", "Right") that are themselves recursive binary sub-trees. However, mathematical functions such as *ln*, *exp*, *sin*, etc. are unary operators. In such cases, the argument of the function is always placed in the "Left" branch. For instance, the binary tree for the expression in equation (4.9) is illustrated in Figure 4.4.

$$1-((10*x_1)+(6*(x_1*x_2))-(6*(x_3*x_4))) \qquad (4.9)$$

Variable intervals in the box are X_1 = [-2.0, 4.0], X_2 = [0.0, 10.0], X_3 = [-2.0, 1.0], and

X_4 = [-10.0, 0.0]. In Figure 4.4, dotted arrows linking arguments with operator nodes show how intervals are propagated starting from the bottom leaves (variables). Once a level of the tree is completed and corresponding sub-expression intervals are calculated according to basic interval operations, they are linked by next level operators. This procedure goes on until the top most "root" node representing the whole constraint is reached resulting in the constraint range of [-339, 261].

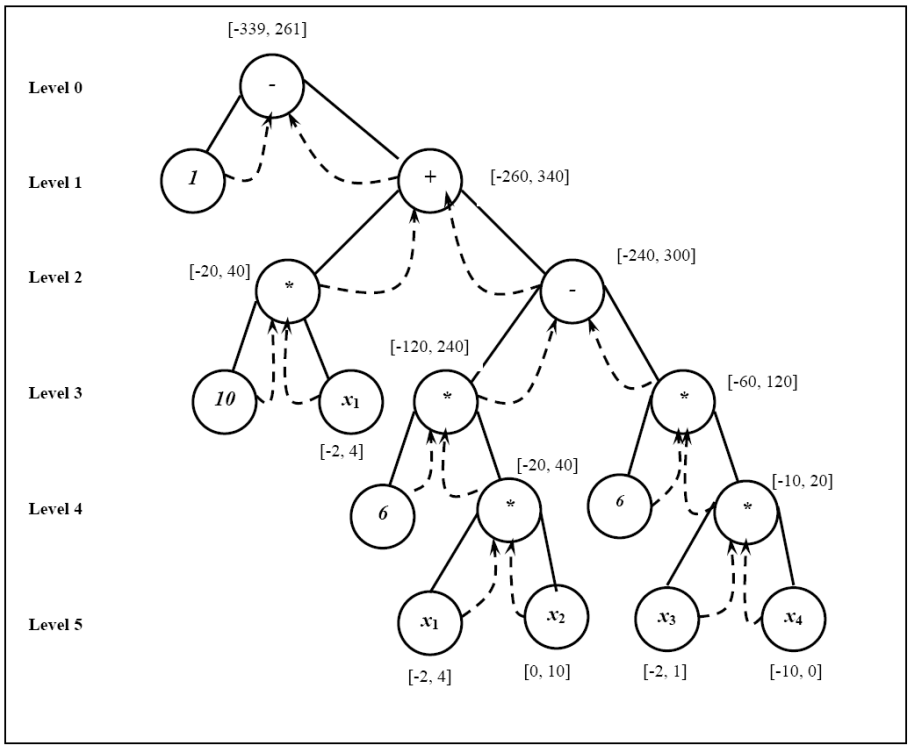

Figure 4.4. Interval propagation for the expression "1-((10*x_1)+(6*(x_1*x_2))-(6*(x_3*x_4)))".

b. *IIR* Labeling Procedure

Suppose a binary tree is constructed for an expression and its source variables are to be identified over a given box **Y**. This is accomplished by a labeling procedure via tree traversal. Let us assume the expression given in the equation (4.9) is an equality constraint. In Figure 4.5, the path constructed by *IIR* is illustrated graphically on the constraint given in equation (4.9). Straight lines in the figure indicate the propagation tree, dashed arrows indicate binary decisions, and bold arrows indicate the path

constructed by *IIR*.

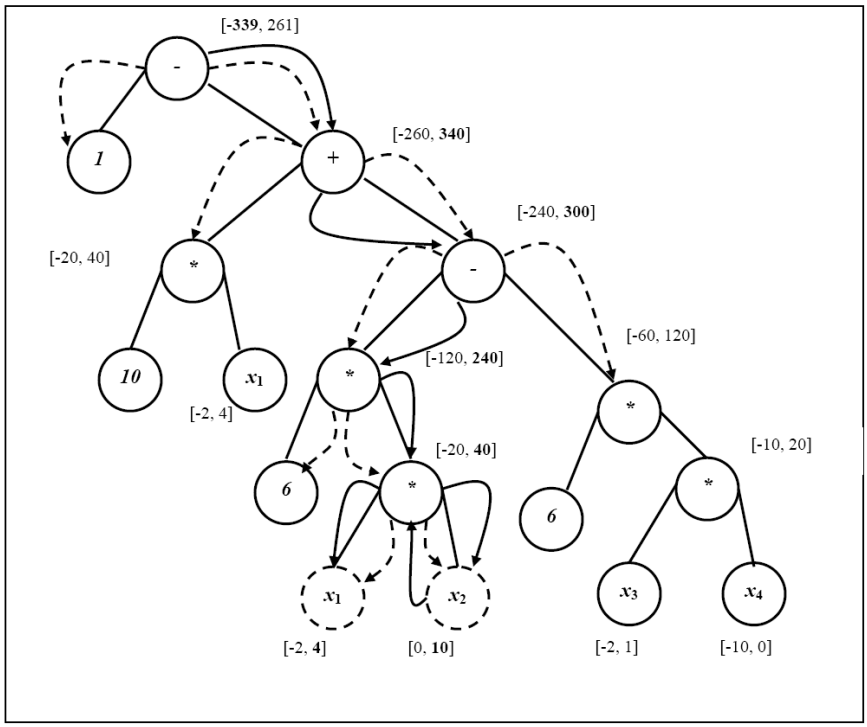

Figure 4.5. Implementation of *IIR_Tree* over the binary tree for "1-((10*x_1)+(6*(x_1*x_2))-(6*(x_3*x_4)))=0"

For illustrating how *IIR* works on a given constraint over domain **Y**, we introduce the following notation.

D^k: a parent sub-expression at tree level k (k=0 is root node),

L^{k+1} and R^{k+1}: immediate Left and Right sub-expressions of D^k at level k+1,

$[\underline{D}^k, \overline{D}^k]$: interval bounds of parent sub-expression D^k,

$[\underline{L}^{k+1}, \overline{L}^{k+1}]$ and $[\underline{R}^{k+1}, \overline{R}^{k+1}]$: interval bounds of immediate left and right sub-expressions at level k+1, and

Λ^k: labeled bound at level k.

IIR starts by labeling $\overline{D}^0$ if the expression is an inequality constraint or the expression is an objective function. The target is $\overline{G_i}(\mathbf{Y})$ for inequalities so as to reduce $PG^i{}_Y$,

and in equalities the target is the *max* $\{|\underline{D}^0|,\ \overline{D}^0\}$ (i.e., *max* $\{|\underline{H}_i(\mathbf{Y})|, \overline{H}_i(\mathbf{Y})\}$ is targeted to reduce PH^i_Y).

In summary,

Type of Expression	**Target at root node**
Objective Function (Maximization)	$\overline{D}^0$ i.e., $\overline{F}(\mathbf{Y})$
Inequality constraint ($\leq$)	$\overline{D}^0$ i.e., $\overline{G}i(\mathbf{Y})$
Equality constraint (=)	*max* $\{|\underline{D}^0|,\ \overline{D}^0\}$ (i.e., *max* $\{|\underline{H}_i(\mathbf{Y})|, \overline{H}_i(\mathbf{Y})\}$

Suppose, we have an equality constraint in equation (4.9), we label the bound that gives *max* $\{|\text{-}339|,\ |261|\}$, that is, we label "-339" as Λ^0 at the root node. Next, we determine the pair of interval bounds $\{\underline{L}^1 - \overline{R}^1\}$ which results in "-339". Hence, $\underline{L}^1 \Theta \overline{R}^1 = \underline{D}^0$. We then compare the absolute values of individual bounds in this pair and take their maximum as the label at level k+1. That is, Λ^1=*max* $\{|\underline{L}^1|, |\overline{R}^1|\} = \overline{R}^1 =$ 340. A formal description of *IIR* rule is given in a pseudocode in Figure 4.6.

The procedure is applied recursively from top to bottom; each time searching for the bound pair resulting in the labeled bound Λ^{k+1} till a leaf (a variable) is hit. Once this forward tree traversal is over, all leaves in the tree corresponding to the selected variable are set to "Closed" status. The procedure then backtracks to the next higher level of the tree to identify the other leaf in the couple of variables that produce the labeled bound. *IIR_Tree's* pseudocode is given in Figure 4.7.

```
Node_Type IIR _Tree (Node_Type Start_Node) {
 If ((Count>2) OR (All leaves are "Closed")) exit;
 Select_Node = IIR (Start_Node);     /*calls procedure IIR */
 If (Select_Node. Status = "Open Node")
    Start_Node = IIR_Tree(Select_Node);
 Else if (Select_Node. Status = "Open Leaf")  /*found a source variable*/
    {
      Store source variable "Open Leaf";
      Close all leaves of type "Open Leaf";
      Count++;
      Start_Node = IIR_Tree (Next_Up(Select_Node));   /*backtrack to identify
second source*/
     }
    Else Start_Node = IIR_Tree (Next_Up(Select_Node)); /*backtrack to identify
second source*/
 Return Start_Node;
}
```

Figure 4.6. Procedure *IIR _Tree*: Recursive tree traversal of *IIR*.(Input: Root node; Output: pair of source leaves - variables)

Node_Type *IIR (Node_Type Node)* {

if (node_level $k = 0$), $bnd = \overline{G}(\mathbf{Y})$; /* if equality $bnd = max\ \{|\underline{H}_i(\mathbf{Y})|, \overline{H}_i(\mathbf{Y})\}$

if objective function $bnd = \overline{F}(\mathbf{Y})$ */

else $bnd = \Lambda^k$;

Identify the pair a Θ b =

$\{\{\underline{L}^{k+1}\Theta\underline{R}^{k+1}\} \vee \{\underline{L}^{k+1}\Theta\overline{R}^{k+1}\} \vee \{\overline{L}^{k+1}\Theta\underline{R}^{k+1}\} \vee \{\overline{L}^{k+1}\Theta\overline{R}^{k+1}\}\}$: a Θ b = *bnd*;

$\Lambda^{k+1} = max\ \{|a|, |b|\}$;

if $\Lambda^{k+1} = |a|$, then return Left branch node as labeled at level k+1;

else then return Right branch node as labeled at level k+1;

}

Figure 4.7. Pseudocode for *IIR* (Input: node at level *k*; Output: labeled node at level *k+1*)

All steps of the labeling procedure carried out in the example are provided below in detail.

Level 0: $[\underline{D}^0, \overline{D}^0] = [-339, 261]$. $\Lambda^0 = \underline{D}^0$.

a Θ b $= \underline{L}^1 - \overline{R}^1 = \{1\text{-}340\} = -339$. $\Lambda^1 = max\ \{|\underline{L}^1|, |\overline{R}^1|\} = max\ \{|1|, |340|\} = 340$

$= \overline{R}^1$.

Level 1: $[\underline{D}^1, \overline{D}^1] = [-260, 340]$

a Θ b = {-20+(-240) or 40+300} = 340 $\Rightarrow$ a Θ b= $\overline{L}^2 + \overline{R}^2$. $\Lambda^2 = max\ \{|\overline{L}^2|, |\overline{R}^2|\}$

$= max\ \{|40|, |300|\} = 300 \Rightarrow \overline{R}^2$.

Level 2: $[\underline{D}^2, \overline{D}^2] = [-240, 300]$

a Θ b = { (-120)-120 or 240-(-60) } = 300 $\Rightarrow$ a Θ b= $\overline{L}^3 - \underline{R}^3$. $\Lambda^3 = max\ \{|\overline{L}^3|, |\underline{R}^3|\}$

$= max\ \{|240|, |-60|\} = 240 \Rightarrow \overline{L}^3$.

Level 3: $[\underline{D}^3, \overline{D}^3] = [-120, 240]$

a Θ b = {6*(-20) or 6*40 } = 240 $\Rightarrow$ a Θ b= $\overline{L}^4 * \overline{R}^4$. $\Lambda^4 = max\ \{|\overline{L}^4|, |\overline{R}^4|\}$

$= max\ \{|6|, |40|\} = 40 \Rightarrow \overline{R}^4$.

Level 4: $[\underline{D}^4, \overline{D}^4] = [-20, 40]$

a Θ b = {-2*0 or -2*10 or 4*0 or 4*10 } = 40 $\Rightarrow$ a Θ b= $\overline{L}^5 * \overline{R}^5$. Λ^5

$= max\ \{|\overline{L}^5|, |\overline{R}^5|\} = max\ \{|4|, |10|\} = 10 \Rightarrow \overline{R}^5$.

The bound $\overline{R}^5$ leads to leaf x_2. The leaf pertaining to x_2 is "Closed" from here onwards, and the procedure backtracks to Level 4. Then, the labeling procedure leads to the second source variable, x_1.

Note that the uncertainty degree of the parent box is 600 whereas when it is sub-divided into four sibling boxes by bisecting the two source variables, the uncertainty degrees of sibling boxes become 300, 330, 420, and 390. If the parent box were sub-divided using the largest width variable rule (x_2 and x_4), then the sibling uncertainty degrees would have been 510, 600, 330 and 420.

3. *IA* Partitioning Coordinator

The binary tree structure is transferred to the Interval Inference Rule base component for identifying the subdivision direction selection through the *IA* partitioning coordinator. It also coordinates the activities between the components such as Branching Master, Interval Library, Local Search Methods (*FSQP*), Working List Box (*WLB*), Feasible or Pending Box Repository (*FBR / PBR*), and Discard Bin. *IA* Partitioning Coordinator comprises of five sub-components such as Box Ranker, Tree Selector, Termination rule, variable screener, and box discarder.

i. Box Ranker

The sibling boxes generated from the Branching Master will be returned to the *IA* Partitioning Coordinator for inserting into the *WLB*. The Box ranker subcomponent is designed to rank the *WLB* based on the following box ranking strategies:

1. Ranking Based on Degree of Uncertainty

This strategy is applicable for *BCOP* and *CCSP* problems. The degree of uncertainty is defined as the PF_Y and $TINF_Y$ for *BCOP* and *CCSP* problems respectively.

Degree of uncertainty with regard to optimality: $PF_Y = \overline{F}(\mathbf{Y}) - CLB$

Degree of uncertainty of an inequality constraint: $PG^i_{\mathbf{Y}} = \overline{G_i}(\mathbf{Y})$

Degree of uncertainty of an equality constraint: $PH^i_{\mathbf{Y}} = \overline{H_i}(\mathbf{Y}) + |\underline{H_i}(\mathbf{Y})|$

Total degree of uncertainty: $TINF_{\mathbf{Y}} = \sum_{i \in \text{ all pending constraints}} (PG^i_{\mathbf{Y}} + PH^i_{\mathbf{Y}})$

where

F, G, H : the objective function, inequality and equality constraints respectively.

CLB : Current Lower bound

In case of Worst-first box selection strategy, the box with maximum degree of uncertainty will be selected for re-partitioning in the next iteration. However, in case of best-first strategy, a box with minimum degree of uncertainty will be selected for re-partitioning in the next iteration.

2. Ranking Based on Single Criterion

This strategy is applicable only to *COP* problems. The boxes are ranked based on the following criteria:

Criteria 1: If there is an initial feasible solution found for the given *COP* problem, and then sort the boxes in descending order of $\overline{F}(\mathbf{Y})$**.** However, ties in $\overline{F}(\mathbf{Y})$ are resolved according to the minimum $TINF_{\mathbf{Y}}$.

Criteria 2: If there is no initial feasible solution found for the given *COP* problem, then sort the boxes in ascending order of $TINF_{\mathbf{Y}}$**.** However, ties in $TINF_{\mathbf{Y}}$ are resolved according to the maximum $\overline{F}(\mathbf{Y})$.

3. Ranking Based on Static Penalty

This strategy is applicable only to *COP* problems. The boxes are ranked based on the following merit function value:

Merit function value (MFV)= $|\overline{F}(\mathbf{Y})| + \sum_{i \in \text{ all pending constraints}} (PG^{i}_{\mathbf{Y}})^2 + (PH^{i}_{\mathbf{Y}})^2$

According to this strategy, sort the boxes in the descending order of MFV.

ii. Tree Selector

Tree Selector subcomponent will influence the extent of repartitioning of a new box picked up from working list (*WLB*) and is based on the following tree management systems:

1. Worst-first Tree Management

This box selection strategy is applicable for *BCOP***,** and *CCSP*.

According to this tree management, the boxes are selected based on the maximum P_Y and maximum $TINF_Y$ for *BCOP* and *CCSP* respectively.

2. Best-first Tree Management

This box selection strategy is applicable only for *COP*. The box selection is carried out following a best-first strategy, i.e., the box with the least merit function as defined in the Box ranker is selected first. A detailed description of the above tree management system is provided in the previous chapter (i.e., Chapter 3 under Pure Heuristic Search).

3. Depth-first Tree Management

This strategy is applicable only for *COP* and *CCSP*. A detailed description of the above tree management system is provided in the previous chapter (i.e., Chapter 3 under Depth-First Search).

4. New Iterative deepening (*applicable for COP and CCSP*)

This strategy is applicable only for *COP* and *CCSP*. A detailed description of the above tree management system is provided in the previous chapter (i.e., Chapter 3 under Adaptive tree management).

iii. Termination Rule

The **Termination Rule** plays an important role in *IPA* to obtain solutions which are close to the actual solutions. This sub-component defines the constraint satisfaction tolerance, box elimination tolerance, maximum CPU time allocation, and the

Separation distance between solutions. This particular is responsible for termination of a given algorithm defined for solving *BCOP, CCSP* and *COP* problems.

In case of bound constrained optimization problems, a run is completed when for all non-discarded pending boxes the difference of the function upper bound over the box to the current lower bound ($|CLB - \overline{F}(\mathbf{X})|$) is less than 1 x 10^{-13}.

The other termination options are:

Box elimination tolerance : 0.0

Maximum CPU time allowed (seconds) : 300 seconds on 2 GB RAM, 2.4 GHz Intel Xenon CPU, under Windows OS system.

In case of *CCSP*, a run is completed when all solutions or a user defined number of solution for a given constraint system is found or the indeterminate boxes list (i.e., all non discarded pending boxes list) size equals to zero.

The other termination options are:

Box elimination tolerance : 0.0

Maximum CPU time allowed (in STU's) : 0.771 Standard Time Units (STU)

The other options are:

Maximum of subdivision direction selected : 8

Constraint satisfaction tolerance : ≤ 1E-8

Separation distance between solutions : ≥1E-6

In case of *COP*, a run is completed when the number of function evaluation reaches 2000*(Dimension of the problem + Total number of constraints) or the indeterminate boxes list size equals to zero.

The other termination options are:

Box elimination tolerance : 0.0

Maximum CPU time allowed (in STU's) : 2.827 Standard Time Units (STU)

(Note: 1 Standard Time Unit = (CPU Time in seconds for a given problem / Time required to complete the Shekel CPU (Törn and Zilinskas 1989, Scherbina et al. 2002))

The other options are:

Maximum of subdivision direction selected : 6

Constraint satisfaction tolerance : less then or equal to 1E-6

Separation distance between solutions : greater then or equal to 1E-6

iv. Variable Screener (Applicable for *COP* and *CCSP*)

The variables identified for each constraint and objective function (if applicable) by the Interval Inference Rule Base will be transferred back to the IA Partitioning Coordinator and to a list. The list of selected subdivision directions is re-screened by a Symbolic priority allocation defined as follows.

In case of *CCSP*:

IIR is applied to every constraint in the *CCSP* to identify a pair of source variables for each. The resulting set of variables, denoted by V, might be large, leading to a wide set of sibling boxes generated in parallel. We develop a symbolic priority allocation scheme to narrow down the size of V. A weight w_j is assigned to each variable $x_j \in$ V and the average $\overline{w}$ is calculated. The final set of variables to be re-partitioned in the next iteration of *IPA* is composed of all $x_j \in$ V with $w_j > \overline{w}$. However, the maximum

number of variables that can be selected for re-partition must be less then or equal to six and eight for *COP* and *CCSP* respectively.

Here, w_j is defined as a function of several criteria: PG^i_Y (PH^i_Y) of constraint g_i for which x_j is identified as a source variable, the number of times x_j exists in g_i, and total number of interactive terms in which x_j is involved within g_i. Furthermore, the existence of x_j in a trigonometric and even power sub-expression in g_i is included in w_j by inserting corresponding flag variables. When a variable x_j is a source variable to more than one constraint, the weight calculated for each such constraint is added to result in a final w_j, defined in equation (4.10).

$$w_j = \sum_{i \in IC_j} [PH^i_Y / PH_{\max} + PG^i_Y / PG_{\max} + e_{ji} / E_j + a_{ji} / A_j + t_{ji} + p_{ji}] / 5 \quad (4.10)$$

where:

IC_j : set of indeterminate constraints (over **Y**) where x_j is a source variable,

TIC : total set of indeterminate constraints,

PH_{max} : $\max_{i \in TIC} \{PH^i_Y\}$,

PG_{max} : $\max_{i \in TIC} \{PG^i_Y\}$,

e_{ji} : number of times x_j exists in constraint $i \in IC_j$,

E_j : $\max_{i \in ICj} \{e_{ji}\}$,

a_{ji} : number of interactive terms x_j is involved in constraint $i \in IC_j$,

A_j : $\max_{i \in IC_j} \{a_{ji}\}$,

t_{ji} : binary parameter indicating that x_j exists in a trigonometric or non-polynomial expression in constraint $i \in IC_j$, and

p_{ji} : binary parameter indicating that x_j exists in an even power or abs expression in constraint $i \in IC_j$.

This weighting method is illustrated on a *CCSP* with 4 variables and 3 equalities. The first constraint is the expression given in equation. (4.9). The other two constraints are

provided in equations (4.11) and (4.12). Variable domains are as listed for equation (4.9).

$$(6*(x_1*x_4))+(6*(x_2*x_3))- (10*x_3)-4 = 0 \qquad (4.11)$$

$$(sin(x_1*x_2)*cos((x_1^2)-x_2))+ (x_1* x_4) = 0 \qquad (4.12)$$

In Table 4.1, we provide a tabulated summary of symbolic characteristics pertaining to each variable and each constraint. This information is used in calculating w_j.

The pairs of maximum impact variables found for each constraint are (x_1, x_2), (x_1, x_4) and (x_1, x_4) for the first, second and third constraints, respectively. The set V consists of { x_1, x_2, x_4}. A sample weight calculation for x_1 in the first constraint is given as $\left(\frac{\frac{600}{600}+\frac{2}{3}+\frac{1}{3}+\frac{0}{1}+\frac{0}{1}}{5} \right) = 0.4$.

Constraint No. (*i*)	$\mathbf{x_j}$	e_{ji}	a_{ji}	p_{ji}	t_{ji}	$[\overline{H_i}(Y), \underline{H_i}(Y)]$	$PH^i{}_Y$
Constraint 1	x_1	2	1	0	0	[-339,261]	261+339=600
	x_2	1	1	0	0		
	x_3	1	1	0	0		
	x_4	1	1	0	0		
Constraint 2	x_1	1	1	0	0	[-374,196]	374+196=570
	x_2	1	1	0	0		
	x_3	2	1	0	0		
	x_4	1	1	0	0		
Constraint 3	x_1	3	3	1	1	[-41,21]	41+21=62
	x_2	2	2	0	1		
	x_3	0	0	0	0		
	x_4	1	1	0	0		

Table 4.1. Summary of symbolic characteristics pertaining to each variable and each constraint

The weight calculation of each variable in each constraint and their final weights are indicated in Table 4.2.

Variable	Weight in Constraint 1	Weight in Constraint 2	Weight in Constraint 3	Total Variable Weight (w_j)
x_1	0.4	0.32	0.82	1.54
x_2	0.40	0.59	0.42	1.41
x_4	0.60	0.59	0.42	1.61
			Average weight	**1.521**

Table 4.2. Weight calculation of each variable in each constraint and their final weights

Consequently, (x_1, x_4) are selected for re-partitioning. This results in three sibling boxes whose sum of IF_Y is indicated in bold in the table below (2nd column). One sibling box is found to be infeasible and discarded.

For comparison purpose, we also show the sum of IF_Y over sibling boxes that would result from re-partitioning other pairs of variables. It is observed that total IF_Y of the pair (x_1, x_4) is lower than all other variable couples.

Selected variables	**(x_1, x_4)**	(x_1, x_2)	(x_1, x_3)	(x_2, x_3)	(x_2, x_4)	(x_3, x_4)
Total IF_Y of sibling boxes	**2141**	3088	2758	3786	3778	3728

In case of COP

In each indeterminate box assessment, *IIR* is applied to every constraint in the *COP* to identify a pair of source variables. The resulting set of variables, denoted by V. In addition to that, if there is an initial feasible solution found then the *IIR* is applied to the objective function in the *COP* to identify a pair of source variables, denoted by V_s. However, in each feasible box assessment, *IIR* is applied to only objective function in the *COP* to identify a pair of source variables, denoted by V_s.

The resulting set of variables, denoted by V, might be large, leading to a wide set of sibling boxes generated in parallel. The symbolic priority allocation scheme defined as above is applied to narrow down the size of V, and the maximum of number of

variables selected from V is (6-size of V_s), denoted by V_f. The final set of variables selected for re-partitioning the box assessment is the set of V_f and V_s.

v. Box Discarder (Applicable for *COP* and *BCOP*)

The Box discarder sub-component is designed for *COP* and *BCOP* problems. The main purpose of this component is to discard the sub-optimal boxes from the list of indeterminate boxes to Discard Bin. This can also be called as cut-off test based on optimality.

The cut-off test is performed based on the following criterion:

Criteria 1: A box is said to be discarded if the difference of the function upper bound over the box to the current lower bound ($|CLB - \overline{F}(\mathbf{X})|$) is less than 1 x 10^{-13}.

Criteria 2: A box is said to be discarded if the function upper bound over the box is strictly less than the current lower bound ($\overline{F}(\mathbf{X}) < CLB$).

4. Interval Arithmetic Library

IA partitioning coordinator calls the Interval Arithmetic Library (*Profil / Bias*) to perform the interval arithmetic operations to calculate the expression and subexpression at each level of hierarchical binary tree. The information obtained from the Interval Arithmetic Library will be transferred to the Interval Inference Rule Base for the selection of the subdivision directions.

5. Local Search Method (FSQP)

IA partitioning coordinator triggers a call to Local Search Method where all boxes not previously subjected to local search are processed by the Feasible Sequential Quadratic programming procedure when the Total Area Deleted (*TAD*) by discarding boxes fails to improve in the two consecutive partitioning iterations in a given sub-

tree. The solution identified by the Local Search Method will be stored in *FBR / PBR* through the *IA* partitioning coordinator.

6. Branching Master

The *IA* partitioning coordinator activates the *Branching Master* for implementing the selected subdivision direction. Once sibling boxes are generated, they are returned to the *IA* partitioning coordinator for placement in the Pending/Feasible Box Repository, *PBR/FBR,* or discarded if infeasible. *WLB* is also updated if sibling boxes are fit to be re-partitioned.

The proposed interval subdivision direction selection rules can be well inserted into the directed acyclic graph framework developed by the *COCONUT* project (Schichl and Neumaier 2005).

4.5. New variant of *IIR* (*IIR_Widths*)

As an alternative to the above described rule, *IIR*, we have also developed another rule (called *IIR_Widths*), that chooses that branch of the computation tree which has the largest width of the expression inclusion related to the given node. In case the two widths are equal, we follow the branch, which belongs to the above given symbolic inference. This new variant is implemented for solving bound constrained optimization problems.

Let us consider the following expression given in equation (4.13) for illustration of *IIR* and *IIR_Widths* on bound constrained optimization problem:

$$((x_1 + x_2) * (x_3 + x_4)) + sin(x_1 + x_3) \qquad (4.13)$$

4.5.1. An Illustration of *IIR* and *IIR_Widths* Procedures

Suppose we have the example given in Figure 4.8 with the expression interval [-166, 451]. Then, "451" is selected as the labeled bound Λ^0 at the root node. In *IIR*, we next determine which pair of interval bounds ($\{\underline{L}^I + \underline{R}^I\}$, $\{\underline{L}^I + \overline{R}^I\}$, $\{\overline{L}^I + \underline{R}^I\}$, $\{\overline{L}^I + \overline{R}^I\}$) results exactly in $\overline{D}^0$. The pair of interval bounds that provides 451 is (450, 1) since "450+1= 451". Hence, $\overline{L}^I \Theta \overline{R}^I = \overline{D}^0$. We then compare the absolute values of individual bounds in this pair and take their maximum as the label at level *k+1*. Λ^{k+1}=max $\{\overline{L}^I, \overline{R}^I\} = \overline{L}^I$ = 450. All steps of *IIR_Tree* for *IIR* and *IIR_Widths* are provided below in detail and decisions are illustrated in Figures 4.8 and 4.9 with bold arrows respectively.

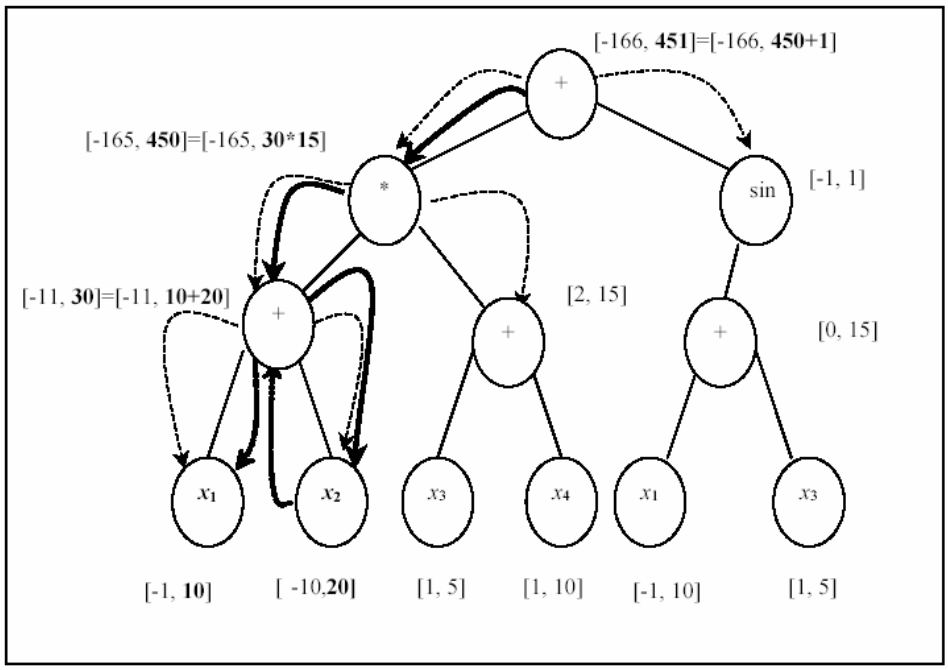

Figure 4.8. Demonstration of the run of *IIR* on the ($(x_1+x_2)*(x_3+x_4)$) + *sin* (x_1+x_3)

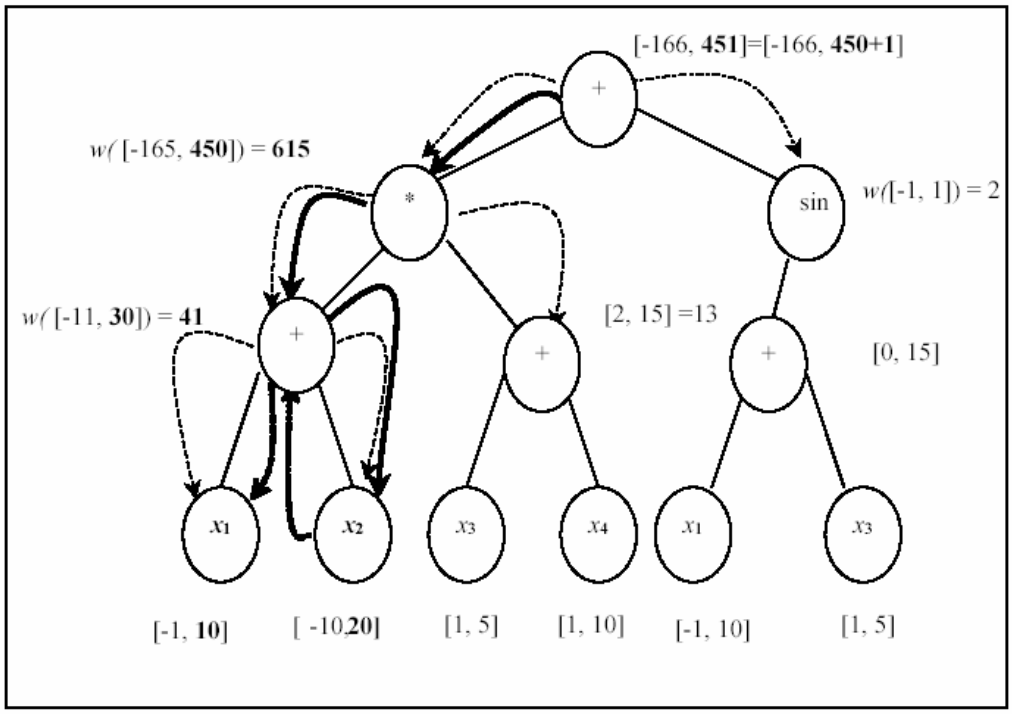

Figure 4.9. Demonstration of the run of *IIR_Widths* on the $((x_1+x_2)*(x_3+x_4)) + sin\ (x_1+x_3)$.

In case of *IIR*, this leads to $\overline{R}^3$, a bound of leaf x_2. The leaf pertaining to x_2 is "Closed" from here onwards, and the procedure backtracks to Level 2. Then, *IIR* leads to the second source variable, x_1.

IIR

Level 0: $[\underline{D}^0, \overline{D}^0]$= [-166, **451**]

$\Lambda^0 = \overline{D}^0$.

$a \ominus b$ = {(-165+1) or (*450+1*) or (-165-1) or (450-1) }

= **451.**

Hence, $a \ominus b = \overline{L}^1 + \overline{R}^1$, and

Λ^1 = max {| $\overline{L}^1$ |, | $\overline{R}^1$ |} = max {|450 |, |1|}

= **450** = $\overline{L}^1$.

Level 1: $[\underline{D}^1, \overline{D}^1]$= [-165, **450**]

$a \ominus b$ = {(-11*2) or (30*2) or (-11*15) or (*30*15*)}

IIR_Widths

Level 0: $[\underline{D}^0, \overline{D}^0]$= [-166, 451], $\Lambda^0 = \overline{D}^0$.

$w(L^1)$ = 615 and $w(R^1)$ = 2. Hence,

Λ^1 = max { $w(L^1)$, $w(R^1)$ } = **615** = L^1.

Level 1: $[\underline{D}^1, \overline{D}^1]$= [-165, 450]

$w(L^2)$ = 41 and $w(R^2)$ = 13.

Λ^2 = max { $w(L^2)$, $w(R^2)$ } = **41** = L^2.

Level 2: $[\underline{D}^2, \overline{D}^2]$ = [-11, 30]

$w(L^3)$ = 11 and $w(R^3)$ = 10.

Λ^3= max { $w(L^3)$, $w(R^3)$ } = **11** = L^3.

= **450**

$\Rightarrow a \ \Theta b = \overline{L}^2 * \overline{R}^2$,

Λ^2 = max {$|\overline{L}^2|$, $|\overline{R}^2|$} = max {|30|, |15|} = **30** $\Rightarrow \overline{L}^2$.

Level 2: $[\underline{D}^2, \overline{D}^2]$ = [-11, **30**]

$a \ \Theta b$ = {(-1-10) or (-1+20) or (*10+20*) or (10-10)}

= **30**

$\Rightarrow a \ \Theta b = \overline{L}^3 + \overline{R}^3$,

Λ^3 = max {$|\overline{L}^3|$, $|\overline{R}^3|$} = max {|10|, |20|} = **20** $\Rightarrow \overline{R}^3$.

In case of *IIR_Widths*, this leads to L^3, a bound of leaf x_1. The leaf pertaining to x_1 is "Closed" from here onwards, and the procedure backtracks to Level 2. Then, *IIR_Widths* leads to the second source variable, x_2.

As a final remark on this example, we would like to mention that the two *2-best* parallel gradient based rules from the literature (Berner 1996) (Rules B/C) select x_2 and x_4 in parallel for re-partitioning this box. This results in a 10% lower reduction in the total pending status of all four siblings as compared to the reduction achieved by *IIR* and *IIR_Widths***.**

4.6. Convergence Proof of *IIR*

Remarks 4.1 and 4.2 discuss *even power*, *abs* and *trig* operators where *IIR* cannot label an interval bound at level *k+1* symbolically if some ambiguous conditions hold on sub-expression intervals at relevant levels of the binary tree. Remark 4.3 indicates two exceptional cases for interval multiplication operator. Remark 4.4 shows that *IIR* symbolically identifies the correct pair of candidate bounds resulting in Λ^k at any tree level *k* as long as the ambiguities indicated in remarks 4.1, 4.2 and 4.3 do not exist in

a constraint expression.

It is noted that a variable can be a source variable for partitioning the parent box's domain only if its width exceeds a tolerance size. We assume throughout the following proofs that the given constraint's calculated interval is finite.

Remark 4.1.

Let the operator at any level k of a binary tree be Θ = "^m" (m is even) or Θ = "*abs*", and let $\Lambda^k = \underline{L}^k = 0$. Further, let $\underline{L}^{k+1} < 0$. Then, *IIR* may not be to identify Λ^{k+1}. ✓

The remark is described by providing a counter example showing that *IIR* cannot identify Λ^{k+1} when the operator at level k is *even power* and $\Lambda^k = 0$. Suppose at level k, we have the interval [0, 16] and $\Lambda^k = \underline{L}^k = 0$. The operator at level k is ^2. Since "power" is a unary operator, there is a single Left branch to this node at level $k+1$. The Left branch at level $k+1$ has a sub-expression interval [-4, 2]. It is obvious that neither $\underline{L}^{k+1}$ nor $\overline{L}^{k+1}$ results in Λ^k. ■

Remark 4.2.

Let *trig* denote any trigonometric function. Define *maxtrig* and *mintrig* as the maximum and the minimum values *trig* can take during one complete cycle. Further, let the operator at any level k of a binary tree be Θ = "*trig*", and $maxtrig \in [\underline{L}^k, \overline{L}^k]$ or $mintrig \in [\underline{L}^k, \overline{L}^k]$. Then, *IIR* may not be able to identify Λ^{k+1}. ✓

Similar to remark 4.1, a counter example is provided to describe it. Suppose we have Θ = "*sin*" operator at level k and the interval $[\underline{L}^k, \overline{L}^k] = [0.5, 1]$. The interval of the unary Left branch at level $k+1$ is $[\underline{L}^{k+1}, \overline{L}^{k+1}] = [\pi/6, 2\pi/3]$. Both $\underline{L}^{k+1}$ and $\overline{L}^{k+1}$ might result in $\underline{L}^k$ and none result in $\overline{L}^k$. ■

Remark 4.3.

Suppose the interval operator at a given level k is "$\Diamond = '\times'$, and, $\underline{L}^{k+1}, \underline{R}^{k+1} < 0, \overline{L}^{k+1}, \overline{R}^{k+1} > 0, \left|\underline{L}^{k+1}\right| = \overline{R}^{k+1}$ and $\left|\underline{R}^{k+1}\right| = \overline{L}^{k+1}$. Then, *IIR* might not be able to label a bound in the right or left sub-trees at level $k+1$. ✓

It is sufficient to show a counter example for *IIR*'s labeling procedure. Suppose '$\times$' type of interval operation exists at level k, with L^{k+1}=[-1,2] and R^{k+1}= [-2,1]. Then, at level k the $\times$ operator's interval is [-4, 2]. If the labeled bound is 2 at level k, then both $\underline{L}^{k+1} \times \underline{R}^{k+1} = \overline{L}^{k+1} \times \overline{R}^{k+1} = 2$ and we cannot choose among the two pairs of bounds at level $k+1$ that both provide the labeled bound at level k. ■

Remark 4.4 (Illustration of reduction in uncertainty).

For constraint expressions excluding the ambiguous sub-expressions indicated in remark's 4.1, 4.2 and 4.3, *IIR* identifies the correct couple of bounds at level $k+1$ that result exactly in Λ^k at level k. ✓

This remark is true by monotonicity property of elementary interval operations and functions. ■

Theorem 4.1 states that unless ambiguous sub-expressions indicated in remark's 4.1, 4.2, and 4.3 exist in a constraint expression c_i or an objective function f, partitioning a source variable identified by *IIR* in a parent box **Y** guarantees immediate reduction in the labeled bound of c_i or f at the root level of the binary tree, and hence, a reduction in the degrees of uncertainty (PG^i_Y, PH^i_Y, or PF_Y) of at least one immediate child box. The proof of Theorem 4.1 relies on inclusion isotonicity and symbolic processing.

Theorem 4.1.

Suppose a given constraint $g_i(x)$ or $h_i(x)$, or an objective function f does not contain

the sub-expression types indicated in remarks 4.1, 4.2 and 4.3. Let **Y** be a parent box whose domain is partitioned by at least one source variable identified by *IIR* and let S_t be its children. Then, $PG^i_{S_t} < PG^i_Y$, or $PH^i_{S_t} < PH^i_Y$, or $PF_{S_t} < PF_Y$ for at least one child S_t.

Proof:

First, we show that there is an immediate guaranteed reduction in the uncertainty degrees of three children S_t, t=1,..,3, assuming that there are two source variables, x^Y_r, x^Y_m, identified by *IIR* in box *Y* for a given constraint $g_i(x)$. We define S_1, S_2, S_3 and S_4 as four children produced by the parallel bisection of these two source variables. We denote intervals of x^Y_r and x^Y_m in box **Y** as: $\mathrm{I}^Y_r = [\underline{\mathrm{X}}^Y_r, \overline{\mathrm{X}}^Y_r]$ and $\mathrm{I}^Y_m = [\underline{\mathrm{X}}^Y_m, \overline{\mathrm{X}}^Y_m]$, respectively. Variable domains in a given child are denoted by I^S_j, j=1,2,...,n. We assume that $I^S_j = I^Y_j, \forall\, j \neq r, m$. In Table 4.3, all child domains are listed.

Let $\overline{\mathrm{X}}^Y_r$ and $\overline{\mathrm{X}}^Y_m$ be identified as most contributing source bounds to $\overline{G}_i(\mathbf{Y})$. Below, we show that $PG^i_{S_t} < PG^i_Y$ for children S_1, S_2 and S_3, and that $PG^i_{s_4} = PG^i_Y$. (The proof techniques for $h_i(x)$ or $f(x)$ are similar, therefore omitted.)

Sibling	I^S_r	I^S_m
S_1	$[\underline{\mathrm{X}}^Y_r, \underline{\mathrm{X}}^Y_r + w(\mathrm{I}^Y_r)/2]$	$[\underline{\mathrm{X}}^Y_m, \underline{\mathrm{X}}^Y_m + w(\mathrm{I}^Y_m)/2]$
S_2	$[\underline{\mathrm{X}}^Y_r + w(\mathrm{I}^Y_r)/2, \overline{\mathrm{X}}^Y_r]$	$[\underline{\mathrm{X}}^Y_m, \underline{\mathrm{X}}^Y_m + w(\mathrm{I}^Y_m)/2]$
S_3	$[\underline{\mathrm{X}}^Y_r, \underline{\mathrm{X}}^Y_r + w(\mathrm{I}^Y_r)/2]$	$[\underline{\mathrm{X}}^Y_m + w(\mathrm{I}^Y_m)/2, \overline{\mathrm{X}}^Y_m]$
S_4	$[\underline{\mathrm{X}}^Y_r + w(\mathrm{I}^Y_r)/2, \overline{\mathrm{X}}^Y_r]$	$[\underline{\mathrm{X}}^Y_m + w(\mathrm{I}^Y_m)/2, \overline{\mathrm{X}}^Y_m]$

Table 4.3. Domain Boundaries of Sibling Boxes

Case S_1:

Based on child domains defined in Table 4.3, $S_1 \subseteq \mathbf{Y}$. Then, by inclusion isotonicity, $w(G_i(S_1)) \leq w(G_i(\mathbf{Y}))$ and $\overline{G_i}(S_1) \leq \overline{G_i}(Y)$. Further, since $\overline{\mathrm{X}}_r^{S_1} \neq \overline{\mathrm{X}}_r^{Y}$ and $\overline{\mathrm{X}}_m^{S_1} \neq \overline{\mathrm{X}}_m^{Y}$, then, $\overline{G_i}(S_1) \neq \overline{G_i}(\mathbf{Y})$.

From the above, $\overline{G_i}(S_1) < \overline{G_i}(\mathbf{Y})$ holds as strict inequality which leads to $PG_{S_1}^i < PG_Y^i$.

One can show by similar reasoning that $PG_{S_2}^i < PG_Y^i$ and $PG_{S_3}^i < PG_Y^i$. However, $PG_{S_4}^i = PG_Y^i$, because $\overline{\mathrm{X}}_r^{S_4} = \overline{\mathrm{X}}_r^{Y}$ and $\overline{\mathrm{X}}_m^{S_4} = \overline{\mathrm{X}}_m^{Y}$.

The above proof is applicable to all bound combinations (four in total) of contributing source bounds other than $(\overline{\mathrm{X}}_r^S, \overline{\mathrm{X}}_m^S)$ pair. In each case, three of the children result in reduced $PG_{s_t}^i$.

When only one source variable is partitioned and two children are obtained, then, S_1 is guaranteed to have reduced $PG_{s_1}^i$. ■

We now describe supporting rules that are applied by *IIR* in case labeling ambiguities described in remark 4.1 and 4.2 arise during tree traversal. For the exceptional case found in remark 4.3, the choice in the two pairs of bounds is arbitrary.

Corollary 4.1.

Let there exist a sub-expression of the type indicated in Remark 4.1 at level *k* of a binary tree with $\Lambda^k = \underline{L}^k = 0$ and interval bound at level *k+1*, $\underline{L}^{k+1} < 0$. The bound labeling rule to be applied by *IIR* at level *k+1* is $\Lambda^{k+1} = \underline{L}^{k+1}$. This rule supports *IIR*'s reduction of INF_Y or PF_Y. ✓

Proof:

Under the conditions indicated in remark 4.1, labeling $\underline{L}^{k+1}$ at level *k+1*, results in the selection of the bound pair targeting $\underline{L}^{k+1}$ at level *k+2*. The binary sub-tree below level *k* is analogous to the full constraint binary tree, and by induction, the principle of identifying the correct bound pair in that sub-tree (Remark 4.4) for reducing $|\underline{L}^{k+1}|$ is valid as proved in Theorem 4.1. Hence, source variable pair selected by *IIR* (using this rule) in forthcoming partitioning iterations target $|\underline{L}^{k+1}|$. Then, $|\underline{L}^{k+1}| \rightarrow 0+$, that eliminates the ambiguity problem at level *k*, after which Theorem 4.1's guarantee of immediate (or in next iteration) reduction in INF_Y or PF_Y holds. ■

Corollary 4.2.

Let there exists a trig type sub-expression at level k of a binary tree with *maxtrig* $\in[\underline{L}^k, \overline{L}^k]$ or *mintrig* $\in[\underline{L}^k, \overline{L}^k]$. The bound labeling rule to be applied by *IIR* at level *k+1* is $\Lambda^{k+1} = \max\{|\underline{L}^{k+1}|, |\overline{L}^{k+1}|\}$. This rule supports *IIR*'s reduction of INF_Y or PF_Y. ✓

Proof:

Similar to the proof of Corollary 4.1, setting $\Lambda^{k+1} = \max\{|\underline{L}^{k+1}|, |\overline{L}^{k+1}|\}$ targets at finding (in the corresponding sub-tree) the source variables that reduce the part of the interval containing the maximum number of repetitive trigonometric cycles. The ambiguity at level *k* is resolved in forthcoming partitioning iterations when $[\underline{L}^k, \overline{L}^k]$ excludes *maxtrig* or *mintrig*. ■

Theorem 4.2.

The *IPA* algorithm is convergent both with the *IIR* and with the *IIR_Widths* interval subdivision selection rules in the sense that the sequence of leading intervals converges only to global maximizer points.

Proof:

Consider first the case when the *IIR* rule is applied. Assume that there exists such a subsequence $\{X_i\}$ of the leading boxes that X_i is a subset of X_{i-1}, and there exists a point x' in the search interval such that $f(x') < f(x^*)$, and x' is in each X_i. We demonstrate that it will imply a contradiction.

We prove first that during the subdivision in the subsequence $\{X_i\}$ every such variable that appears in the computation tree will be halved. It is so since otherwise when a variable that is used during computation would keep the original width while the width of others converges to zero. As a consequence, then $\{X_i\}$ converges to a point regarding those variables that appear in the computed expression. This fact provides the contradiction, since the selection of the subinterval with the largest upper bound on the objective function cannot converge to a point x' in the search interval such that $f(x') < f(x^*)$, due to the assumed α-convergence.

For the case of the *IIR_Widths* subdivision direction selection rule the proof is similar, but it is more straightforward that the respective interval subsequence has such intervals for which the width converges to zero for all variables used within the computation. ■

It is noted that the leading interval subsequences do not necessarily converge to points of the search space. It may happen when there is at least a variable that does not contribute to the objective function, i.e., it is not used in the computation tree. In such

cases there is a continuum of global maximizer points and the resulting intervals will highlight this phenomenon, since such variables will keep their width in the original search interval. This is true for both introduced selection rules, and this indicates that these are as sophisticated as the rules B and C that also have this feature.

Chapter 5

Computational Results

5.1. Bound Constrained Optimization (*BCOP*)

The numerical experiments are conducted on well-known test problems from the literature in order to assess the performance of *IIR* against *k-best* (for a fair comparison, *2-best*) parallel version of established subdivision direction selection rules and against the standard 2^n multi-section rule.

Testing environment

The runs were executed on a PC with 2 GB RAM, 2.4 GHz Intel Xenon CPU, under Windows OS system. All codes were developed with Visual C++ 6.0 interfaced with the *PROFIL* interval arithmetic library.

5.1.1. Comparison Basis

The performance of *IIR* is compared with two well established and efficient gradient-based subdivision direction selection rules (*Rules B/C*) from the literature (Ratschek and Rokne 1995, Csendes et al. 2000). These rules have become standard benchmarks because they have been identified as best performing among others after extensive testing. For a fair comparison with our multi-section approach, *Rules B/C* is also converted into multi-section rules by applying *2-best* subdivision strategy (Berner 1996), i.e., the first two variables from the list (sorted according to *Rules B/C*) are partitioned. These rules are briefly described below.

Rule B (Hansen 1992):

Rule B chooses variables according to a maximal index consisting of variable interval width multiplied by the width of its respective first order derivative, $w(F_i'(\mathbf{X}))$, as defined in equation (5.1):

$$\textit{Select } x_k\text{: } C_k := \max_{i=1\ldots n}\{C_i\}, \text{ where } C_i = w(X_i)w(F_i'(\mathbf{X})). \tag{5.1}$$

Rule C (Ratz 1992):

The first order derivative of each variable is multiplied by the difference between its interval and its midpoint, M_i. The variable with the maximum index value defined in equation (5.2) is selected by *Rule C.*

$$\textit{Select } x_k\text{: } C_k := \max_{i=1\ldots n}\{C_i\}, \text{ where } C_i = w(F_i'(\mathbf{X})\,(X_i - M_i)). \tag{5.2}$$

5.1.2. Test Functions

27 well-known test functions from the literature are selected to compare the performance of *IIR* against *Rules B/C* multi-section approach. The number of test instances becomes 34 as some functions such as Levy, Griewank and Schwefel are run with increasing number of dimensions (up to 30). The test functions are provided with their references and features in Table 5.1. The complexities and features of these test functions are discussed in detail in previous comparisons (Özdamar and Demirhan 2000) and they present a balanced portfolio from easy (such as Schwefel 3.1, Box), through moderate (such as Griewank) to difficult (such as Schwefel 3.7) problems with topological properties discussed in many global optimization references.

Problem (Dim.)	Description	Reference
Ackley (4)	Multimodal trigonometric function	Website MATLAB / TEST/Lazauskas
Brownal (10)	Twice differentiable sum of squares.	CUTEr
Box 3D (3)	Singular problem with manifold of solutions	Schwefel (1981)
Cos 4 (4)	Multimodal trigonometric function	Breiman and Cutler (1993)
Dixon3dq (10)	Twice differentiable quadratic function	CUTEr
Djong's Function 2 (8)	Global optimum inside a long, narrow, parabolic shaped flat valley, slow convergence in the valley	De Jong (1975)
Eg1 (3)	Twice differentiable trigonometric function	CUTEr
Exp 6 (6)	Exponential function	Breiman and Cutler (1993)
Extended Kearfott (4)	Polynomial function	Kearfott (1979)
Extrosnb (10)	Twice differentiable Sum of Squares.	CUTEr
Genhumps (5)	Twice differentiable Sum of Squares.	CUTEr
Griewank (5, 10, 20)	Wide spread regularly distributed maxima, trigonometric	http://iridia.ulb.ac.be/langerman/ ICEO.html
Hartman (6)	4 local minima	Törn and Zilinskas (1989)
Hs045 (5)	Twice differentiable geometric function	CUTEr
Levy 14,16,18 (3, 5,7)	2700, 10^5, 10^8 local minima, trigonometric	Levy et al. (1981)
Levy 10,11,12 (5, 8, 10)	10^5 , 10^8 , 10^{10} local minima, trigonometric	Levy et al. (1981)
Michalewicz (5)	Multimodal trigonometric function, function values around narrow peaks give little information	http://iridia.ulb.ac.be/langerman/ ICEO.html
Powell (4)	Singular, Hessian at origin	Moré et al. (1981)
Rastrigin (8)	Highly multimodal trigonometric, regularly distributed local maxima	Website MATLAB / TEST/Lazauskas
Rosenbrock (10)	Long curved only slightly decreasing valley	Rosenbrock (1970)
S271 (6)	Twice differentiable quadratic function	Schittkowski (1987).
S288 (20)	Twice differentiable quadratic function	Schittkowski (1987).
Schwefel 1.2 (4)	Continuous unimodal function	Schwefel (1981)
Schwefel 3.1 (3)	Unimodal function	Schwefel (1981)
Schwefel 3.7 (15, 30)	Singular Hessian at $x^* = 0$	Schwefel (1981)
Shekel (4; m=10)	Multimodal test function	Törn and Žilinskas (1989)
Sphere (7)	Unimodal	http://iridia.ulb.ac.be/largerman/ ICEO.html

Table 5.1. Description and references of the bound constrained optimization test functions.

5.1.3. Computational Results

Performance is measured in terms of the number of function and gradient calls (as indicated by FE and GE, respectively in Table 5.2), the CPU time in seconds, and the absolute deviation from the global optimum value. Positive absolute deviations occur in cases where methods fail to converge within 300 CPU seconds. The latter test instances are indicated at the end of Table 5.2. In *IIR,* runs FE does not include calls at subexpression levels because they are partial expression calls, and the latter are assumed as computational overhead. FE indicated for *IIR* is equal to the number of tree traversals. *Rules B* and *C* are supported by the monotonicity test since it does not require additional gradient calls. Finally, all methods use the cut-off test.

A run is completed when for all non-discarded pending boxes the difference of the function upper bound over the box to the current lower bound is less than 1×10^{-13}.

In the last five rows of Table 5.2, we can observe that *Rules B* and *C* were not able to converge on four test functions within the CPU time limit imposed, but they are able to converge for the 5th one in 0.141 seconds. Similarly, the *IIR* rule does not converge for the first three functions, but it was able to converge in the 4th and 5th functions within 6.153 seconds, and 0.282 seconds respectively. However, *IIR_Widths* do not converge for first three test functions and the 5th test function, but it was able to converge in 4th one within 6.374 seconds. The performance of *IIR* is notable in the function S288 where *Rules B/C* end up very far from the global optimum.

Considering all 34 test functions, the results obtained by *Rules B* and *C* are not significantly different. When the first part of Table 5.2 is analyzed, we observe that the average number of function calls for *IIR* is larger than those of *Rules B* and *C* (including their gradient calls). Despite this fact the average CPU time required for

IIR is almost half of those of *Rules B* and *C*. That of *IIR_Widths* is almost one-fourth of *Rules B/C*. The tree traversal overhead in *IIR* that can be compared with the task of calculating the gradient in the other rules. The number of best solutions obtained by *IIR_Widths* compares very well with others. Hence, we can conclude that *IIR*'s symbolic methodology of selecting the maximum impact variables is more efficient than that of the function rate of change based rules.

In Table 5.3, we provide a summary of total CPU times taken by all rules for functions with less than five dimensions and for those greater than five dimensions. In the first part of Table 5.3, we observe that *IIR*'s performance is inferior in test functions up to five dimensions. In problems with larger dimensions, its performance is significantly superior as compared to *Rules B* and *C*. When the outlier CPU time (Griewank 20D) was removed from this set, we have found the difference in performance statistically significant (at a 5% significance level). In Table 5.3, the total CPU time needed by all three methods is given for the first 29 test problems (split into less than or greater than five dimensions) where all methods converge. This outcome is expected because the sequence of variables to be partitioned gains more importance in larger dimensional problems. Both *Rules B* and *C* are affected by the width of variable domains, and this tends to push the selected variable sequence into a more balanced manner in terms of box size. However, the size of variable domains has a more implicit impact on the choice of variables in *IIR*.

Function (Dimension)	*IIR_Bounds*		*IIR_Widths*		*Rule B*			*Rule C*		
	FE	**CPU**	**FE**	**CPU**	**FE**	**GE**	**CPU**	**FE**	**GE**	**CPU**
Box 3D(3D)	180	0.407	164	**0.296**	**140**	141	0.313	**140**	141	0.312
Eg1(3D)	414	0.203	292	0.125	**12**	13	**0.016**	**12**	13	**0.016**
Levy14(3D)	324	0.142	156	**0.095**	**164**	165	0.172	**164**	165	0.188
Powell(4D)	1224	0.875	1224	**0.859**	**1000**	1004	1.141	1060	1061	1.297
Ackley(4D)[$]	4260	2.950	416	**0.312**	**-**	-	-	-	-	-
Cos4(4D)	5460	6.771	4664	4.263	**276**	277	**0.391**	**276**	277	**0.391**
Extended Kearfott(4D)	**296**	**0.125**	**296**	0.156	432	433	0.421	432	433	0.421
Schwefel 1.2 (4D)	1488	5.500	260	**0.125**	**200**	201	0.140	**200**	201	0.140
hs045(5D)	320	0.375	512	0.530	**20**	21	**0.015**	**20**	21	0.047
Griewank(5D)	316	0.359	252	**0.234**	**240**	241	0.390	**240**	241	0.391
Levy10(5D)	316	0.453	304	**0.423**	**236**	237	0.719	**236**	237	0.874
Levy16(5D)	416	0.469	340	**0.155**	**268**	269	0.594	**268**	269	0.594
Genhumps(5D)	10556	11.384	496	**0.593**	**416**	417	1.155	**416**	417	1.155
Exp6(6D)	624	0.671	572	0.514	**28**	29	**0.062**	**28**	29	**0.062**
S271(6D)	932	0.719	524	**0.344**	**520**	521	0.781	**520**	521	0.781
Sphere(7D)	384	0.281	384	0.281	**108**	109	**0.203**	**108**	109	**0.203**
Levy18(7D)	416	**0.500**	532	0.765	**364**	365	1.624	**364**	365	1.624
Rastrigin(8D)	538	1.187	492	**0.765**	**488**	489	2.140	**488**	489	2.140
Levy8(8D)	568	**1.311**	580	1.483	**380**	381	3.156	**380**	381	3.156
Djong's Function s(8D)	488	**0.717**	488	0.750	**484**	485	2.219	488	489	2.220
Rosenbrock(10D)	**652**	1.410	**652**	**1.389**	720	721	6.156	708	709	6.047
Griewank(10D)	640	2.017	572	**1.110**	488	489	3.578	**484**	485	3.484
Extrosnb(10D)	572	**1.141**	572	1.145	552	553	4.437	**544**	545	4.338
Dixon3dq(10D)	616	**1.297**	616	1.329	644	655	3.859	588	589	3.687
Levy12(10D)	672	1.915	564	**1.529**	**472**	473	6.422	**472**	473	6.422
Brownal(10D)	648	3.393	608	**2.184**	**484**	485	5.422	**484**	485	5.516
Schwefel3.7(15D)	128	**0.172**	128	0.203	**124**	125	1.313	**124**	125	1.313
Griewank(20D)	1332	13.590	1120	**5.891**	**960**	961	39.482	972	973	39.733
Schwefel3.7(30D)	252	**0.812**	252	0.830	**244**	245	17.187	**244**	245	17.124
Average	1208	2.108	622	0.989	374	375	3.697	455	456	3.703
Std. dev.	2141	3.283	815	1.266	255	256	7.832	259	259	7.865
No. of Best Sol.	-	8	-	16	-	-	6	-	-	5
% of Best Sol.		27.5		55.17			20.68			17.24
	Problems not Converged within 300 CPU Seconds									
Function (Dimension)	*IIR_Bounds*		*IIR_Widths*		*Rule B*			*Rule C*		
	FE	**Abs. Dev.**	**FE**	**Abs. Dev.**	**FE**	**GE**	**Abs. Dev.**	**FE**	**GE**	**Abs. Dev.**
Shekel (4D, m=10)	10712	0.008	10656	**0.000**	17182	17183	**0.000**	17182	17183	**0.000**
Michalewicz (5D)	10112	2.884	14292	**0.000**	17506	17507	**0.000**	17274	17275	**0.000**
Hartman(6D)	9636	0.163	12144	0.002	9484	9485	**0.000**	9484	9485	**0.000**
S288(20D)	1356	**0.000**	135	**0.000**	6196	6197	3000	10576	10577	3000
Schwefel 3.1(3D)	308	**0.000**	21072	**0.000**	220	221	**0.000**	220	221	**0.000**

$: indicates problem in computing Gradient value

Table 5.2. Comparison of numerical results for bound constrained optimization problems.

Dimension	*IIR_Bounds*	*IIR_Widths*	*Rule B*	*Rule C*
n < 5	16.973	6.231	2.594	2.765
n >= 5	44.173	22.447	100.914	100.911

Table 5.3. Total CPU times in seconds for small and large size *BCOP*

5.2. Continuous Constraint Satisfaction Problems

The numerical experiments are conducted on well-known test problems from the literature in order to assess the performance of different tree management of *IIR* against established subdivision direction selection rules and some established software such as *ALIAS, QUAD*, *ICOS,* and *COCOS*.

Testing environment:

All our runs are executed on a PC with 256 MB RAM, 1.7 GHz P4 Intel CPU, on Windows platform. Our codes are developed with Visual C++ 6.0 interfaced with *PROFIL* interval arithmetic library (Knuppel 1994) and *CFSQP* (Lawrence et al. 1997). CPU times for *QUAD* are reported on a 1.0 GHz PC Notebook whereas *ALIAS* runs are reported for a DELL400 1.7 GHz machine.

5.2.1. Comparison Basis

Comparison between IIR and other subdivision direction selection rules:

We compare the performance of the symbolic subdivision method, *IIR,* with two established subdivision direction selection rules: largest width (commonly known as *Rule A*), and maximum interval rate of change (S*mear*), a rule mentioned previously. All three rules are embedded in the same *IP* algorithm with adaptive tree management, restricted parallelism approach and *FSQP*. In *Rule A* and *Smear*, w_j are taken as variable interval width or maximum Jacobian interval bound and they are divided by the largest width or maximum Jacobian bound. Let us re-emphasize that,

in *IIR*, only variables that are identified by *IIR_Tree* are considered and assigned weights, while in *Rule A* and *Smear*, all variables are candidates for partitioning. We also carried out experiments using the *Round Robin* variable selection strategy. However, in most cases it could not converge, and we therefore omit it from the presentation.

Comparison between adaptive tree management approach and other strategies:

The impact of the adaptive tree management approach is measured by running the three rules, *IIR*, *Rule A* and *Smear*, with pure depth-first and pure worst-first branching strategies that are usually utilized in *IP*.

Comparison basis with other interval methods:

In the presentation of results, we also provide the published results of the following symbolic-interval methodologies: *ALIAS* (Merlet 2000, http://www-sop.inria.fr/coprin/logiciels/ALIAS/Benches/), *QUAD* (Lebbah et al. 2003), ICOS (Lebbah 2003), and *COCOS* wherever results are available (Neumaier et al. 2005). As mentioned previously, *ALIAS* is an extensive interval-symbolic software library where many of the local and global interval and symbolic filtering methods co-exist with special tools for univariate polynomials. *COCOS* involves an advanced *IP* algorithm with hull consistency and linear programming techniques. *QUAD* is designed for filtering quadratic systems, its first stage involves linearization, and the second stage uses simplex algorithm to narrow down the bounds of variables in the resulting linear program. The developers show for their two illustrative examples (also included here) that *QUAD* is more efficient than 2B and 3B consistency techniques and compare their method with *Numerica* (Van-Hentenryck et al. 1997). *ICOS* is reported to be the most reliable method for the *CCSP* among those compared by Neumaier et al. (2005).

5.2.2. Test Functions

The benchmarks:

Most of the benchmarks originate from the European IST project *COCONUT* (http://www.mat.univie.ac.at/~neum/glopt/coconut/benchmark.html) and the *COPRIN* web page (http://www-sop.inria.fr/coprin/logiciels/ALIAS/Benches/). Twenty-three test problems (some quite challenging to solve) are used in the comparison (including seven problems from kinematics-robotics fields, one from chemistry, a reactor problem, two economic models and others), three of which are announced as difficult problems (Direct Kinematics, Countercurrent Reactors 2, Fredtest) by the *COPRIN* group. Three other difficult problems, Cyclic5, Heart, Neurophysics, come from the *COCONUT* group. *ALIAS* results are available mostly among the first thirteen problems (except for Cyclic5, Chemequ, Kin2 and Stewart-Gough) whereas *ICOS* and *COCOS* results are available for most of the total set of 23 problems (in 18 problems). Two quadratic problems (Kin2 and Stewart-Gough) are solved by *QUAD*. In Table 5.4, all test problems are listed with their details (number of dimensions, nonlinear and linear equations, linear inequalities) and the software they have been solved with. *ALIAS* usually solves these problems with Gradient_Solve+Hull Consistency+3B or Hessain_Solve+3B+StrongHullConsistency method combinations. Gradient_Solve and Hessian_Solve algorithms obtain sharper function and Jacobian bounds, respectively. We discuss some of the difficult problems below.

Direct Kinematics has two close solutions that are hard to isolate. This problem determines the pose parameters of a parallel robot platform and involves 8 difficult highly non-linear and inter-dependent trigonometric equations with 3 independent

Problem	D,# NE, #LE, #LIE	Category	# of Sol.	Solved by
Kin2	8,8,0,0	Quadratic	10	QUAD, ICOS
Kin1-Modified	6,6,0,0	Trigonom.	16	ALIAS
KinCox	4,4,0,0,	Quadratic	2	ALIAS, ICOS
Direct Kinematics	11,11,0,0	Trigonom.	2	ALIAS
Stewart-Gough	9,9,0,0	Quadratic	2	QUAD
FredTest	6,5,1,2	Polynom.	1	ALIAS, ICOS
Eco9	8,7,1,0	Quadratic	16	ALIAS, ICOS
Redeco8	8,6,2,0	Quadratic	8	ALIAS, ICOS
Puma	8,7,1,0	Quadratic	16	ALIAS, ICOS
Chemequ	5,5,0,0	Polynom.	4	ICOS
Counter concurrent reactors 2	6,6,0,0	Polynom.	2	ALIAS
Cyclic5	5,5,0,0	Polynom.	1	ICOS
Dietmaier	12,12,0,0	Quadratic	40	ALIAS
Heart	8, 6,2,0	Polynom.	2	ICOS
Neuro	6, 6,0,0	Polynom.	1	ICOS
Quadfor2	4,3,1,0	Polynom.	1	ICOS
Wright	5,5,0,0	Quadratic	5	ICOS
Solotarev	4,4,0,0	Polynom.	6	ICOS
S9_1	8,4,4,0	Polynom	4	ICOS
Butcher	7,7,0,0	Polynom	7	ICOS
Trinks	6,4,2,0	Polynom	2	ICOS
Lorentz	4,4,0,0	Polynom	3	ICOS
Remier5	6,5,0,0	Polynom	2	ICOS

Table 5.4. List of *CCSP* benchmarks.

(Note: D: Dimension, NE: Nonlinear Equations, LE: Linear Equations, LIE: Linear Inequailities)

quadratic equations. Other difficult kinematics problems included here and not covered by *COPRIN* group but solved by *QUAD* are: 6R (Kin2) and Stewart Gough. Kin2 is a quadratic problem with 10 real solutions solved by *QUAD* and also tested by *COCONUT* group. Kin2 describes the inverse position problem for six-revolute-joint, and the Stewart Gough involves a manipulator configuration problem. The Stewart Gough has 3 totally independent constraints that might make constraint propagation based filtering methods (such as 2B and Box) ineffective (as verified by Lebbah et al.

2003). *Numerica* (where Box consistency technique is included) makes more than 10,000,000 narrowing iterations to solve this problem. Another benchmark is the trigonometric Kin1-Modified that describes the inverse kinematics of an elbow manipulator.

Puma represents the inverse kinematics of a 3R robot whereas the KinCox is the simple inverse position problem. Kinematics problems specifically require the identification of all configurations (feasible solutions) because missing some solutions result in uncovered workspace. The benchmark, Fredtest, also takes place in the difficult problem category of the *COPRIN* group. Eco9 and ReDeco8 are quadratic economic modeling problems.

Chemequ is 3rd degree chemical equilibrium of hydrocarbon combustion with high constraint dependency. Counterconcurrent Reactors 2 is a 2nd degree sparse and partially separable polynomial. Cyclic5 involves constraints that have geometric terms with increasing number of variables. This problem is also ladder type in the sense that an additional variable is added to the multiplicative terms in each consecutive constraint. Hence, all constraints have strong inter-relations.

Dietmaier benchmark represents the forward kinematics equations of a parallel robot and it is quite famous in robotics for its guaranteed 40 real solutions (Dietmaier 1998). Heart and Neurophysics problems involve complex expressions built by multiplicative terms involving higher order polynomials. The remaining problems (last eight) have simpler forms.

5.2.3. Computational Results

Test results are given in terms of Standard Time Units, STU (Shcherbina et al. 2002) taken by each software to identify all solutions (*ALIAS/QUAD*, *IP*) or a single solution

(*COCOS/ICOS*). All CPU times reported for *ALIAS, QUAD, ICOS, COCOS*, and proposed *IP* are converted into STU's by taking into consideration the processing speed of machines reported.

Results:

Table 5.5 provides a summary of results obtained by all three subdivision selection rules and three branching strategies covering all 23 problems as well as *ALIAS* (9 problems), *QUAD* (2 problems), *ICOS* (18 problems) and *COCOS* (18 problems). The detailed results for all 23 problems are omitted due to lack of space and difficulty in following large tables.

Before discussing the results in Table 5.5, we would like to mention that none of the methods in comparison includes all available symbolic-interval tools. For instance, *COCOS* lacks some numerical and consistency techniques that are available in *ALIAS*, *QUAD* is very different from other methods, and the *Rules* in *IP* use only interval evaluation as filtering technique and they do not yet include numerical tools. Furthermore, the test bed includes interesting and hard benchmarks, but it is not an exhaustive test bed, and all method's results are not published in the literature for all available benchmarks. In order to provide such a full comparison, all codes pertaining to the methods mentioned here should be made publicly executable with common data structures. Therefore, our tests and comparisons would only be fair among the three *Rules* and *IP* tree management schemes proposed here. The results of other methods should be viewed as informative and they are included for providing some insight on the degrees of difficulty of the benchmarks. More detailed results are given in Appendix-A.

The following information are provided in Table 5.5: the average STU's taken (all methods), the number of tree levels the solutions are found in (only for adaptive branching strategy); number of *FSQP* calls (all *Rules*); average number of variables partitioned in parallel for *IP* (all *Rules*), the number of function calls outside *FSQP* (all *Rules*). We also provide the number of best solutions found (all methods), number of problems that could not be solved during the given time limit (all methods) and the number of problems where *ICOS*, *COCOS* or *ALIAS*/*QUAD* results are not available. We do not display the number of Jacobian calculations used by *Smear* rule. We also summarize the average percentage of solutions found in each tree level of the adaptive *IP*. In three of the problems (Fredtest, ECO9 and PUMA) simple consistency check has been applied to linear constraints. The convergence rate of the rules slows down significantly when we apply this check in Redeco8, because the linear equality has too many variables and in this case, the reduction in variable domains is not worth the consistency check overhead.

Comparison among the Rules and IP tree management approaches:

When *IIR*, *Rule A* and *Smear* are compared under adaptive tree management scheme *IIR* is distinctly superior as compared to *Rule A* and *Smear*, with respect to average STU, number of unsolved problems and best solutions found. On the average, it consumes 46.177% less time than *Rule A* in solving *CCSP*'s. *IIR* reduces both the number of function calls outside *FSQP* and the number of *FSQP* calls significantly. Sometimes, despite the fact that the number of *FSQP* calls or function evaluations is higher in *IIR*, it tends to need lesser or similar STU's to converge as compared to *Rule A*. The reason is that due to different partitioning sequences, in such problems, *FSQP* takes much longer time in *Rule A* in its search of feasible solutions in a given box. For instance, in Redeco8, the total number of function calls within *FSQP* is 2,501,033

under *IIR* whereas it is 4,849,203 under *Rule A*, but *IIR* takes 0.411 STU's to converge while *Rule A* takes more than 0.771 STU's. In general, the difficult problems for all rules under adaptive tree management scheme are Redeco8 (except for *IIR*), Chemeq, Cyclic5 (except for *IIR*), Dietmaier, and Heart. It is interesting to note that the *Smear* rule works well in Dietmaier as compared to other rules, however, in this problem, ALIAS produces the best overall result. Similarly, *Smear* rule under depth-first approach is the only method that converges in the Heart within the given STU limit.

In the worst-first branching scheme, *Rule A* and *IIR* have similar performance, and *Smear* is still the worst performing rule. In addition to the previously mentioned complex problems, in this branching scheme, Kine2, ECO9 and Reimer5 cannot be solved within the given time limit. In the depth-first approach, the two rules are still similar but *Smear* becomes the best performing one. In this tree management approach, the number of unsolved problems becomes quite large for *IIR* and *Rule A* as compared with the adaptive and worst-first approaches. When the three branching strategies are compared, the adaptive approach is observed to produce the best overall results with *IIR* and *Rule A*.

Many of the complex benchmarks have constraints that are formed of long expressions involving multiplicative terms. The latter property leads to slower convergence and higher tree levels in the adaptive approach. In the adaptive tree management scheme, for Direct, Redeco8, Cyclic5, Reactor 2, Reimer5 and Solotarev, *IIR* finds all solutions in lesser number of tree levels whereas in Dietmaier and Kin1_modified, it requires more levels. The average percentage of solutions found (indicated in Table 5.5) in each level illustrates the differences in the performance of the *Rules*. From this information, we can gather that *Rule A* and

Smear have difficulty in identifying solutions after the second level whereas *IIR* identifies more solutions in Level 1 on the average and it is able to converge in every problem except Dietmaier and Heart.

Although the average scale of variable partitioning parallelism seems to be similar among the *Rules*, we observe that all *Rules* use different scales when this information is viewed on individual problem basis. A larger scale does not imply better performance or vice versa, e.g., in S9_1, *IIR* converges in 0.045 STU's with average parallelism scale of 7.8 whereas *Rule A* converges in 0.052 STU's with a parallelism scale of 4.28, and *Smear* cannot converge within 0.771 STU limit with a scale of 5.0. Every rule adjusts its own scale of adaptive parallelism that may change from two (minimum number of variables to be partitioned in all *Rules*) to eight. A good example of such self-sustained parallelism is Direct Kinematics where both *IIR* and *Rule A* prefer to maintain a lower parallel profile whereas in Fredtest a high profile results in good performance. However, we can generally say that the scale of parallelism is higher in more difficult problems.

Other methods:

In the comparison between *ALIAS/QUAD* and *ICOS/ COCOS*, one should remember that in the 9 problems of the first 13 benchmarks that include quite difficult ones, *ALIAS* results are available. For the last 10 problems, the first two of which are complex, *ALIAS/QUAD* results are not available. On the other hand, results for *ICOS/COCOS* do not include STU's for three difficult kinematics problems, Direct, Stewart-Gough, Dietmaier, and also for Reactor 2. We observe that *ICOS* is quite slow in converging and out of 18 problems that it attempts, it is able to converge only

in 3 problems before the STU limit is reached whereas *COCOS* can converge in 10 problems.

	***IP* - Adaptive Tree management**			***IP* - Worst-First Tree management**		
	***IIR* Rule**	***Rule A***	***Smear***	***IIR Rule***	***Rule A***	***Smear***
Avg. CPU (STU)	0.122	0.226	0.395	0.308	0.308	0.490
Avg. no. of stages	1.739	2.174	2.217	-	-	-
Avg. no. of SQP calls	858.217	3093.09	3772.17	356.913	369.522	595.739
Avg. no. of function calls	16356.09	34020.48	52959.26	4535.22	4417.13	6662.04
No. of unsolved problems	2	5	11	8	8	13
% of unsolved problems	8.69	21.7	47.83	34.78	34.78	56.52
No. of best solutions found	9	4	2	1	4	2
% of best solutions found	39.13	17.39	8.69	4.34	17.39	8.69
Data not available	0	0	0	0	0	0
Avg. percentage of Solutions found						
Stage 1	72.50	64.55	56.81	-	-	-
Stage 2	17.39	10.27	11.49	-	-	-
Stage 3	5.07	1.63	0.50	-	-	-
Stage 4	1.63	0.00	0.50	-	-	-
Stage 5	-	0.00	-	-	-	-
Total	96.59	76.45	69.3	84.40	80.65	72.77
	***IP* - Depth-First Tree management**			***ALIAS***	***ICOS***	***COCOS***
	IIR Rule	***Rule A***	***Smear***	***QUAD***		
Avg. CPU (STU)	0.506	0.509	0.345	0.234	0.667	0.448
Avg. no.of stages						
Avg. no. of SQP calls	7412.17	9254.43	2538.35	-	-	-
Avg. no. of function calls	603921.3	755506.3	30379.74	-	-	-
No. of unsolved problems	12	15	8	2	7	-
% of unsolved problems	52.17	65.21	34.72	18.18*	36.84*	
No. of best solutions found	2	1	3	4	0	
% of best solutions found	8.69	4.34	13.04	36.36*	0*	
Data not available	0	0	0	12	14	-
Avg. percentage of Solutions found						
Stage 1	-	-	-	-	-	-
Stage 2	-	-	-	-	-	-
Stage 3	-	-	-	-	-	-
Stage 4	-	-	-	-	-	-
Stage 5	-	-	-	-	-	-
Total	63.78	56.87	83.88			

Table 5.5. Summary of results for *CCSP* benchmarks

* Percentage is calculated using the actual number of problems (Actual number of problems = total no. of problems – no. of problems for which data is not available)

ICOS converges in Wright, Kincox and PUMA that are relatively simple as compared to others. Meanwhile, *ALIAS* is able to converge fastest among all methods in Kincox, ECO9, Redeco8, PUMA and Dietmaier, some of which are hard benchmarks. This is quite an achievement because *ALIAS* results are for finding all solutions rather than one. Yet, when we look at the average STU's, we should remember that *ALIAS* results are not available for three time consuming problems, Cyclic5, Chemeq, and Heart whereas *COCOS* and *ICOS* results are available for these problems.

Some detailed observations.

It is interesting to see that problems that are difficult for filtering oriented *ALIAS-QUAD* methods (Fredtest, Direct, Kin1-Modified) are easier to solve for *IIR* and *Rule A* whereas Redeco8 and Dietmaier are serious obstacles for the *Rules*. Redeco8 is also a problem for *ICOS/COCOS*. In Redeco8, only *IIR* is able to identify all solutions among the *Rules*, but it takes longer time than *ALIAS*. In Kin2 that has appropriate constraint structure for filtering methods, *IIR* and *Smear* results are compatible with that of *QUAD*. *ICOS* does not converge and COCOS produces the best overall result, which is remarkably superior to others. In trigonometric *CCSP*'s, Kin1-Modified and Direct Kinematics, filtering techniques in *ALIAS* seem to be much less effective as compared to the *Rules*. In the Stewart Gough, the linear filtering technique in *QUAD* performs worse than the *Rules*, possibly due to the overhead of Simplex method. In Fredtest, *ALIAS* and *ICOS* cannot converge whereas the *Rules* have STU's at 1/1000 level in almost all branching schemes. *COCOS* converges in this problem but it is somewhat slower than the *Rules*. Cyclic5 can be solved by *IIR* under adaptive branching scheme but not by other *Rules* or tree management approaches. It can however be solved by *COCOS* though nearly using total allowed time. ECO9 and Redeco8 are problems where filtering techniques are very successful. They both have

ladder type of nonlinear equation structures where constraint propagation is expected to be effective. Chemeq is successfully solved by *Smear* (under all branching schemes) and *IIR* (under adaptive branching approach), though the latter is much slower. In Reactor 2, *Rule A* and *IIR* are compatible with and better than (in worst-first and depth-first approaches) *ALIAS* in terms of STU's.

In the last ten problems where all method results are available except for *ALIAS/QUAD*, it is observed that except for the Heart, in most problems (Butcher, Lorentz, Quadfor2, Trinks and Wright) all *Rules* under adaptive and worst-first tree management strategies perform better than *ICOS/COCOS*. In the more difficult benchmarks such as reimer5 and S9_1, *IIR* and *Rule A* outperform *COCOS* only with the adaptive tree management approach but not with the other two. More detailed results are illustrated in Appendix A.

Final comments:

As final comments on these numerical results, we would like to add that these tests are only preliminary and the target is to show that basic *IP* that is enhanced by *IIR* and adaptive tree management that invokes *FSQP* is a viable and superior approach that can be adopted in *CCSP's*. Without *FSQP* (or any other local solver), *IP* would have no chance to converge. Testing performance of *IP*- *IIR* with other local solvers or interval Newton constitutes another line of investigation. Integrating advanced consistency techniques within the latter method combination is also a topic of future research.

The novel subdivision direction selection rule, *IIR*, outperforms *Rule A* and *Smear* in the adaptive tree management approach that produces the best overall results when compared with worst-first and depth-first strategies. *COPRIN* group explains that the

tree management system in *ALIAS* swaps adaptively from worst-first/best-first to depth first according to memory usage of the program. In other methods, such as interval Newton, basic best-first strategy is used for tree management.

Finally, we know from the manual of *ALIAS*, that parallel variable bisection can be carried out, though the user has to fix the number of variables to a priori number. Similar parallel bisection methods are also available in bound constrained optimization literature. It would be interesting to run such codes with our simple parallelization scheme. In preliminary experimentation that is not displayed here for lack of space, we found that convergence is slow when the degree of parallelism is fixed to a certain a priori number.

5.3. Constrained Optimization

The numerical experiments are conducted on well-known test problems from the literature in order to assess the performance of different tree management of *IIR* against established subdivision direction selection rules and some established commercial software such as *FSQP, Baron*, *Minos, Conopt, LGO* and *Snopt*.

Testing environment:

All runs are executed on a PC with 256 MB RAM, 1.7 GHz P4 Intel CPU, on Windows platform. The *IP* code is developed with Visual C++ 6.0 interfaced with *PROFIL* interval arithmetic library (Knuppel 1994) and *FSQP*.

5.3.1. Comparison Basis

The *IP* results are compared with five different solvers that are linked to the commercial software *GAMS* (www.gams.com) and with *FSQP* (Zhou and Tits 1996, Lawrence et al. 1997) whose code has been provided by *AEM*

(www.aemdesign.com/FSQPmany obj.htm). The solvers used in this comparison are *Baron 7.0* (Sahinidis 2003), *Conopt 3.0* (Drud 1996), *LGO 1.0* (Pinter 1997), *Minos 5.5* (Murtagh and Saunders 1987), *Snopt 5.3.4* (Gill et al. 1997) and *FSQP*. Every solver is allowed to complete its run without imposing additional stopping criteria except the maximum CPU time.

In order to illustrate the individual impacts of *IP*'s features (the box selection criterion, the adaptive tree search scheme and the branching rule *IIR*), we include the following *IP* variants in the comparison.

i) *IP* with Widest Variable Rule (*Rule A*, Csendes and Ratz 1996, 1997); *IP* with *Smear Rule* (variable with the largest rate of change*$w(x_i)$ by Kearfott and Manuel 1990); and *IP* with *IIR*;

ii) *IP* with depth-first tree search approach; *IP* with best-first tree search approach where box ranking is the same as that of adaptive tree approach; and *IP* with adaptive tree management approach;

iii) *IP* with box ranking according to maximum objective bound augmented with penalty, $\bar{F}(\mathbf{Y})$-INF_Y^2 (Yenjay 2005);

iv) *IP* with *max* INF_Y /*max* $\bar{F}(\mathbf{Y})$ swapping box ranking approach (as described earlier).

The first set of *IP* variants listed above enable us to measure the impact of the branching rule, *IIR* against rules established in the interval literature. The second set of variants enable the comparison of the proposed adaptive tree search management approach against classical tree management approaches. The third and fourth variants enable the comparison of the *max* INF_Y /*max* $\bar{F}(\mathbf{Y})$ swapping box ranking criterion

against the static penalty approach. All three branching rules are run both with augmented $\bar{F}(\mathbf{Y})$ box selection approach and the proposed swapping criterion approach as well as all three tree search management schemes. Since the depth-first tree management approach does not need a box ranking criterion, we have a total number of 15 *IP* runs for each test problem.

The performance of each *IP* variant is measured on all benchmarks with the following performance criteria: the average absolute deviation from the global optimum over all 55 and 5 benchmarks, the average CPU time in STU's, the average number of tree stages where *IP* stops, the average number of times *FSQP* is invoked, the average number of function calls invoked outside *FSQP*, the number of optimal solutions obtained and the number of problems where a feasible solution could not be obtained. We provide two summaries of results: one for 55 problems and one for 5 trigonometric problems, the reason being that *Baron* is not enabled to solve trigonometric models (http://www.gams.com/dd/docs/solvers/baron.pdf).

5.3.2. Test Functions

Numerical experiments have been conducted on a set of 60 *COP* benchmarks, five of them involving trigonometric functions. Most of these test problems are extracted from the *COCONUT* benchmark library (http://www.mat.univie.ac.at/~neum/glopt/coconut/benchmark.html) and Princetonlib **(**http://www.gamsworld.org/performance/princetonlib/princetonlib.htm). These problems are listed in Table 5.6 with the number of dimensions, number of linear and nonlinear inequalities and equalities.

While executing *IP*, we allow at most 2000*(number of variables + number of constraints) function calls carried out outside *FSQP* calls. The maximum iteration number allowed for *FSQP* is limited to 100. We also restrict *IP*'s run time by 2.827

(i.e., 900 seconds) Standard Time Units (STU defined by Scherbina et al. 2002). One STU is equivalent to 318.369 seconds on our machine.

Problem	D, # NE, # LE, # NIE, #LIE	Reference	Problem	D, # NE, # LE, # NIE, #LIE	Reference
Aircraftb	18, 5, 5, 0, 0	Coconut	**Hs053**	5, 0, 3, 0, 0	Coconut
Avgasb	8, 0, 0, 10, 0	Princetonlib	**Hs056**	7, 4, 0, 0, 0	Coconut
Alkyl	14, 6, 1, 0, 0	Coconut	**Hs407**	5, 3, 0, 0, 0	Coconut
Bt4	3, 1, 1, 0, 0	Coconut	**Hs108**	9, 0, 0, 12, 0	Coconut
Bt8	5, 2, 0, 0, 0	Coconut	**Hs080**	5, 3, 0, 0, 0	Coconut
Bt12	5, 3, 0, 0, 0	Coconut	**Hs043**	4, 0, 0, 3, 0	Coconut
Bt11	5, 2, 1, 0, 0	Coconut	**Hs116**	13, 0, 0, 10, 5	Coconut
Bt7	5, 3, 0, 0, 0	Coconut	**Himmel11**	9, 3, 0, 0, 1	Coconut
Dispatch	4, 1, 0, 0, 1	Coconut	**Immum**	21, 0, 7, 0, 0	Coconut
Dipigri	7, 0, 0, 4, 0	Coconut	**Lootsma**	3, 0, 0, 2, 0	Coconut
Degenlpa	20, 0, 14, 0, 0	Coconut	**Lewispol**	6, 6, 3, 0, 0	Coconut
Degenlpb	20, 0, 15, 0, 0	Coconut	**Mwright**	5, 3, 0, 0, 0	Coconut
Eigminc	22, 22, 0, 0, 0	Coconut	**Mhw4d**	5, 3, 0, 0, 0	Coconut
Ex5_2_4	7, 0, 1, 3, 2	Coconut	**Madsen**	3, 0, 0, 6, 0	Coconut
Ex9_1_4	10, 4, 5, 0, 0	Coconut	**Minmaxrb**	3, 0, 0, 2, 2	Coconut
Ex8_4_2	24, 10, 0, 0, 0	Coconut	**Median_scop_vareps**	5, 0, 0, 3, 0	Coconut
Ex9_2_5	7, 3, 4, 0, 0, 0	Coconut	**Matrix2**	6, 0, 0, 2, 0	Coconut
Ex14_1_5	6, 0, 4, 2, 0	Coconut	**Mistake**	9, 0, 0, 12, 0	Coconut
Ex9_2_6	16, 6, 6, 0, 0	Coconut	**O32**	5, 0, 0, 6, 0	Coconut
Ex9_2_7	10, 4, 5, 0, 0	Coconut	**Pgon**	12, 0, 0, 15, 5	Coconut
Ex9_1_2	10, 4, 5, 0, 0	Coconut	**Robot**	14, 2, 0, 0, 0	Coconut
Ex2_1_9	10, 0, 1, 0, 0	Coconut	**Rk23**	17, 7, 4, 0, 0	Coconut
Ex2_1_3	13, 0, 0, 0, 9	Coconut	**S381**	13, 0, 1, 0, 3	Princetonlib
Ex8_4_1	22, 10, 0, 0, 0	Coconut	**S355**	8, 5, 0, 0, 0	Princetonlib
Ex8_4_5	15, 11, 0, 0, 0	Coconut	**S336**	3, 1, 1, 0, 0	Princetonlib
F_e	7, 0, 0, 3, 4	Epperly (1995)	**S262**	4, 0, 1, 0, 3	Princetonlib
Fermat_scop_vareps	5, 0, 0, 3, 0	Princetonlib	**S203**	5, 3, 0, 0, 0	Princetonlib
Fp_2_1	6, 0, 0, 1, 1	Epperly (1995)	**Springs_nonconvex**	32, 0, 0, 10, 0	Princetonlib
Genhs28	10, 0, 8, 0, 0	Coconut	**Steifold**	4, 3, 0, 0, 0	Balogh and Toth (2005)
Hs087	11, 4, 2, 0, 0	Coconut	**Sample**	4, 0, 0, 2, 0	Princetonlib
Hs108	9, 0, 0, 12, 0	Coconut			
Hs080	5, 3, 0, 0, 0	Coconut			

Table 5.6. List of *COP* benchmarks used in experiments

(Note: D: Dimension, NE: Nonlinear Equations, LE: Linear Equations, NIE: Nonlinear Inequality Equations, LIE: Linear Inequalities)

5.3.3. Computational Results

The numerical results are provided in Tables 5.7 and 5.8 for non-trigonometric and trigonometric problems, respectively. When we compare the three tree management schemes for *IP* in Table 5.7, we observe that the overall best deviations from the optimum are obtained by *IIR*_adaptive tree management scheme under the box ranking rule that does not involve augmented objective function bounds (non-penalty). This observation is confirmed by the fact that all three rules under this configuration have the lowest number of problems where *IP* does not converge to a feasible solution. The performance of widest variable rule (*Rule A*) is close to that of *IIR* under both adaptive and best-first tree management schemes. *Rule A* performs best only under best-first/non-penalty box ranking configuration. In other configurations *Rule A* is inferior to *IIR*. *Smear* is usually the worst performing rule under all configurations except for the depth-first approach where it is close to *IIR*. *IIR* is the best performing rule in all configurations except for best-first/non-penalty box ranking configuration.

An advantage of the adaptive tree management scheme is that it minimizes CPU times due to relieved memory requirements and reduced computation times due to less box sorting operations. Further, it is effective in sending the correct boxes (boxes that contain feasible solutions) to *FSQP* that converges to feasible solutions in a lesser number or iterations as compared to other tree management schemes. As expected, the best-first approach is the slowest one among all three tree management schemes (due to maintaining lengthy sorted box lists) and the fastest one is the depth-first. However, the fastest approach is significantly inferior in solution quality when using *Rule A* and *IIR*. In this approach, the performance of *Smear* and *IIR* are not significantly different and that of *Rule A* is quite inferior.

A final observation is that the non-penalty box ranking method is generally better than the penalty one in both best-first and adaptive tree management approaches with respect to the sum of all three partitioning rules' average deviation from the global optimum.

When we compare solvers other than the *IP*, we observe that the two complete solvers *Baron* and *LGO* are best performing. The performance of *Baron* is better than the best *IP* configuration (*IIR*_adaptive_non-penalty) both in terms of average deviation from the optimum and CPU time. However, this difference is not statistically significant. *LGO's* performance is quite inferior to *Baron's* and the third best non-IP solver, stand-alone *FSQP*, is much worse than *LGO*. The difference in performance between *FSQP* and best *IP* configuration that uses *FSQP* as a local solver illustrates the strength of *IP* as a complete solver.

In Table 5.8, for trigonometric problems, we observe that the relative performance of different *IP* configurations is quite similar to our findings in Table 5.7. It is noted that the zero deviation of *Smear* is due to its incapability of solving one problem whereas the other two rules converge in this problem. Solvers other than *IP* are significantly inferior as compared to all *IP* configurations.

Non-Trigonometric (55 Problems)	IP Adaptive tree management								
	Non-Penalty Box Ranking			Penalty Box Ranking					
	IIR	*Rule A*	*Smear*	*IIR*	*Rule A*	*Smear*			
Summary									
Avg. deviation from optimum	0.550	0.613	8.443	0.923	2.269	7.703			
Avg. CPU time (in STU's)	0.317	0.341	0.564	0.421	0.458	0.607			
Avg. no. of stages	4.84	5.82	4.17	4.98	5.89	4.32			
Avg. no. of FSQP calls	1502.04	1311.69	3132.22	1382.72	1271.82	2498.48			
Avg. no. of func. calls	28703.00	27541.96	28146.26	25831.33	29824.40	27895.85			
No. of best solutions	44	43	33	41	39	35			
% of best solutions	80.00	78.18	60.00	74.55	70.91	63.64			
No. of unsolved problems	0	0	6	2	1	5			
% of unsolved problems	0.00	0.00	10.91	3.64	1.82	9.09			
	IP Best-first tree management								
	Non-Penalty Box Ranking			Penalty Box Ranking					
	IIR	*Rule A*	*Smear*	*IIR*	*Rule A*	*Smear*			
Summary									
Avg. deviation from optimum	1.830	0.578	3.869	0.787	2.268	7.312			
Avg. CPU time (in STU's)	1.656	1.791	2.467	1.847	1.928	2.827			
Avg. no. of FSQP calls	860.60	894.29	985.31	854.07	1055.64	982.28			
Avg. no. of func. calls	19402.44	22604.29	11589.37	16654.42	19678.07	11708.91			
No. of best solutions	39	41	30	38	40	31			
% of best solutions	70.91	74.55	54.55	69.09	72.73	56.36			
No. of unsolved problems	3	2	12	5	1	9			
% of unsolved problems	5.45	3.64	21.82	9.09	1.82	16.36			
	IP Depth -first tree management								
	IIR	*Rule A*	*Smear*	*FSQP*	*Baron*	*Conopt*	*LGO*	*Minos*	*Snopt*
Summary									
Avg. deviation from optimum	1.387	9.399	1.497	5.537	0.434	14.541	1.899	14.660	16.357
Avg. CPU time (in STU's)	0.122	0.109	0.422	0.000	0.067	0.000	0.079	0.000	0.000
Avg. no. of FSQP calls	506.69	498.24	1625.85						
Avg. no. of func. calls	27802.65	29450.95	28851.00						
No. of best solutions	33	33	31	31	45	35	40	35	34
% of best solutions	60.00	60.00	56.36	56.36	81.82	63.64	72.73	63.64	61.82
No. of unsolved problems	11	7	12	13	0	9	8	9	9
% of unsolved problems	20.00	12.73	21.82	23.64	0.00	16.36	14.55	16.36	16.36

Table 5.7. Summary of results for non-trigonometric *COP*.

Trigonometric (5 Problems)	IP Adaptive tree management					
	Non-Penalty Box Ranking			Penalty Box Ranking		
	IIR	*Rule A*	*Smear*	*IIR*	*Rule A*	*Smear*
Summary						
Avg. deviation from optimum	0.015	0.026	0.000	0.024	0.026	0.000
Avg. CPU time (in STU's)	0.185	0.286	0.295	0.251	0.269	0.284
Avg. no. of stages	4.00	3.60	3.40	5.40	3.80	3.60
Avg. no. of FSQP calls	1233.40	1475.60	2126.40	1214.00	1330.00	1818.40
Avg. no. of func. calls	27610.00	27718.20	27684.40	932.78	27718.20	27684.40
No. of best solutions	4	4	4	4	4	4
% of best solutions	80.00	80.00	80.00	80.00	80.00	80.00
No. of unsolved problems	0	0	1	0	0	1
% of unsolved problems	0.00	0.00	20.00	0.00	0.00	20.00

	IP Best-first tree management					
	Non-Penalty Box Ranking			Penalty Box Ranking		
	IIR	*Rule A*	*Smear*	*IIR*	*Rule A*	*Smear*
Summary						
Avg. deviation from optimum	0.048	0.029	0.000	0.109	0.039	0.000
Avg. CPU time (in STU's)	1.739	1.781	1.311	1.711	1.708	2.268
Avg. no. of FSQP calls	880.00	800.20	876.20	863.20	950.80	918.80
Avg. no. of func. calls	22457.60	14044.00	13490.20	21505.60	16172.60	10336.60
No. of best solutions	4	4	4	4	4	4
% of best solutions	80.00	80.00	80.00	80.00	80.00	80.00
No. of unsolved problems	0	0	1	0	0	1
% of unsolved problems	0.00	0.00	20.00	0.00	0.00	20.00

	IP Depth -first tree management							
	IIR	*Rule A*	*Smear*	*FSQP*	*Conopt*	*LGO*	*Minos*	*Snopt*
Summary								
Avg. deviation from optimum	0.155	0.101	0.000	1.170	40.781	2.230	104.143	38.082
Avg. CPU time (in STU's)	0.041	0.032	0.128	0.000	0.000	0.003	0.000	0.000
Avg. no. of FSQP calls	386.80	400.60	1322.00					
Avg. no. of func. calls	27626.80	27682.20	27718.20					
No. of best solutions	4	4	4	2	1	1	0	1
% of best solutions	80.00	80.00	80.00	40.00	20.00	20.00	0.00	20.00
No. of unsolved problems	0	0	1	1	0	1	0	0
% of unsolved problems	0.00	0.00	20.00	20.00	0.00	20.00	0.00	0.00

Table 5.8. Summary of results for trigonometric *COP*.

5.4. Summary of results

Table 5.9 illustrates the performance of the best strategies recommended for solving *BCOP*, *CCSP* and *COP* problems. It also illustrates the maximum size of the problems tested here.

In the case of *BCOP*, we observe that *IIR_Widths* method is the best *IIR* configuration and its performance is superior when compared to the best of other rules i.e., *Rule B*. Moreover, the average CPU time required for *IIR_Widths* is almost of one-fourth of

Rule B's. Similarly, in the *CCSP*, we observe that *IIR*_adaptive tree management. method is the best *IIR* configuration versus *ALIAS*/*QUAD*. The average CPU time required for *IIR*_adaptive tree management is almost half of *ALIAS / QUAD*.

Problem Class		**Recommended Strategy**	**Performance measure**	**Performance value**	**Maximum problem dimension**
BCOP	Best of *IIR* configuration	*IIR_Widths*	CPU time	0.989	30
	Best of other rules	*Rule B*	CPU time	3.697	30
CCSP	Best of *IIR* configuration	*IIR*_adaptive tree management	CPU time	0.122	11
	Best of other methods /solvers	*ALIAS / QUAD*	CPU time	0.234	12
COP (Non-Trig. Func.)	Best of *IIR* configuration	*IIR*_adaptive_non-penalty	Avg. dev. from optimality	0.55	32
	Best of other methods / solvers	*Baron*	Avg. dev. from optimality	0.434	32
COP (Trig. Func.)	Best of *IIR* configuration	*IIR*_adaptive_non-penalty	Avg. dev. from optimality	0.015	14
	Best of other methods / solvers	*Snopt*	Avg. dev. from optimality	38.082	14

Table 5.9 Summary of computational results

For the *COP* (non-trigonometric functions), we observe that *IIR*_adaptive_non-penalty is the best *IIR* configuration but this time its performance is inferior when compared to the best performer of other solvers i.e., *Baron*. The average deviation from the optimal solution for *Baron* is close to that of *IIR*_adaptive tree management with non-penalty box ranking. However, the difference in performance is not statistically significant at a 5% significance level. For trigonometric *COP* problems, we observe that *IIR*_adaptive_non-penalty method is significantly superior over the best of other solvers i.e., *Snopt*. These results show that the overall performance of *IIR*

in global optimization problems satisfactory in terms of the performance criteria stated here.

Chapter 6

Applications

6.1. Applications – Continuous Constraint Satisfaction Problem

The Continuous Constraint Satisfaction Problem (*CCSP*) is a core topic in many real-world engineering applications including kinematic analysis. Kinematics is fundamental in the design and control of robot manipulators (used in contact analysis, assembly planning, position analysis, path planning) since performance is achieved through the movement of links/legs whose geometry is crucial. Geometric kinematics calculates the state of a robot from measurements (direct kinematics) or poses (inverse kinematics), and answers associated questions of accuracy and singularities. These problems require the identification of all object positions and orientations that satisfy a coupled nonlinear system of equations (Tsai and Morgan 1985, Dietmaier 1998).

Testing environment

All the *IPA* runs are executed on a PC with 256 MB RAM, 1.7 GHz P4 Intel CPU, on Windows platform. All codes are developed with Visual C++ 6.0 interfaced with *PROFIL* interval arithmetic library (Knuppel 1994) and *CFSQP* (Lawrence et al. 1997).

6.1.1. Description of the Problem

A brief introduction of the inverse position problem is provided for a 6-Revolute-joint problem in mechanics. It is also referred to as test problem Kin2 and is illustrated in Figure 6.1.

A 6R manipulator has six moving links, numbered sequentially from 2, 3, 4, 5 and 6, as shown in Figure 6.1. Link 1 is designated as the base (fixed to ground) and link 6 as the hand or the manipulator. Every two neighboring links are connected by a joint that is associated with a joint axis Z_i, i = 1 to 6. Let Z_i, and Z_{i+1} be two adjacent joint axes and H_iO_{i+1} be the directed common normal between Z_i, and Z_{i+1}. H_i is the intersection of H_iO_{i+1} and Z_i, and O_{i+1} is the intersection of H_iO_{i+1} and Z_{i+1}. Then one can define the following link parameters shown in Figure 6.2.

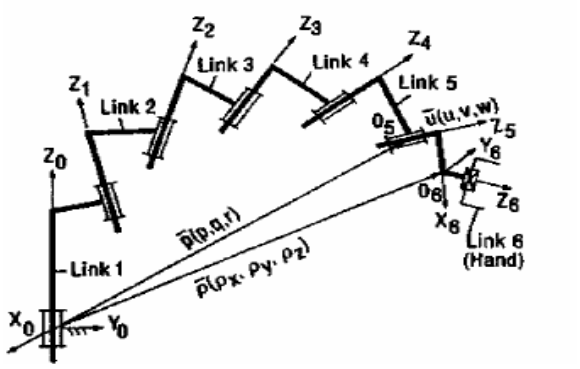

Figure 6.1. A general 6-R Manipulator

Source: Tsai and Morgan (1985)

a_i = the offset distance from the common normal H_iO_{i+1}.

α_i = the angle to rotate the axis Z_i about the common normal H_iO_{i+1} so that Z_i is parallel to Z_{i+1}. The sign of rotation is given by the right hand screw rule with the screw taken along the normal H_iO_{i+1}.

d_i = the distance between the two normals $H_{i-1}O_i$ and H_iO_{i+1} measured from Z_i. The sign of d_i is positive if O_iH_i points to the positive Z_i direction. Otherwise, d_i is negative.

θ_i = the angle to rotate extended line of $H_{i-1}O_i$ about Z_i so that the extended line $H_{i-1}O_i$ is parallel to H_iO_{i+1}. The sign of rotation is given by the right hand screw with the screw pointing along the positive Z_i–axis.

If the i^{th} joint is revolute, then a_i, d_i, and α_i are constant while θ_i is variable. If the i^{th} joint is prismatic, then a_i, α_i, and θ_i are constant while d_i is a variable.

A coordinate system (X_i, Y_i, Z_i) is attached to each link of the manipulator as shown in Figure 6.2. In each coordinate system, the Z_i – axis is defined to align with the i^{th} joint axis, the X_i-axis is the one along the extended line of $H_{i-1}O_i$; and the Y_i- axis is defined according to the right-hand screw rule. The first coordinate system is fixed to ground. Since the common normal H_0O_1 does not exist, the X_1-axis is chosen perpendicular to Z_1, in an arbitrary manner. Also, a seventh coordinate system is attached to the free-end to specify the position of the hand. Z_7-axis lies in the direction from which the hand would approach an object as shown in Figure 6.1. X_7-axis is defined by the common normal between Z_6 and Z_7 axes, and Y_7-axis is defined according to the right-hand screw rule.

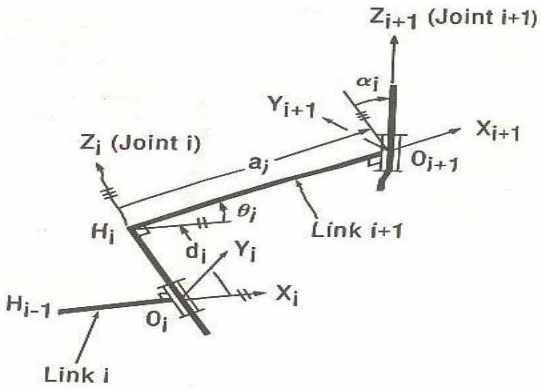

Figure 6.2. The basic notation of 6-R Manipulator Source: Tsai and Morgan (1985)

The equations representing the 6R problem are derived by first defining the coordinates of a point P in the i^{th} and $(i+1)^{st}$ coordinate systems as (p_{xi}, p_{yi}, p_{zi}) and

($p_{xi+1,}\ p_{yi+1,}\ p_{zi+1}$), respectively. These two vectors are related to the hand position and orientation vectors by equations of the form: $\boldsymbol{p_i} = \boldsymbol{A}_i\ \boldsymbol{p}_{i+1}$, where $\boldsymbol{A}_i$ is a matrix whose elements are $c_i = \cos\theta_i$, $s_i = \sin\theta_i$, $\lambda_i = \cos\alpha_i$, and $\mu_i = \sin\alpha_i$. The inverse transformation is written as: $\boldsymbol{p}_{i+1} = \boldsymbol{A}_i^{-1}\boldsymbol{p}_i$. By applying matrix transformation to each pair of coordinate systems between two successive links and proceeding from link 7 to link 1, the following equation is obtained: $\boldsymbol{p}_1 = \boldsymbol{A}_1\ \boldsymbol{A}_2\ \boldsymbol{A}_3\ \boldsymbol{A}_4\ \boldsymbol{A}_5\ \boldsymbol{A}_6\ \boldsymbol{p}_7$. Since an equivalent transformation matrix defines the relationship between the coordinates of any point in the seventh system $\boldsymbol{p}_7$, and that of the same point expressed in the first system, $\boldsymbol{p}_1$, the matrix $\boldsymbol{A}_{eq} = \boldsymbol{A}_1\ \boldsymbol{A}_2\ \boldsymbol{A}_3\ \boldsymbol{A}_4\ \boldsymbol{A}_5\ \boldsymbol{A}_6$ is known when the position and orientation of the hand is specified. Let $\boldsymbol{\rho}$ ($\rho_x,\ \rho_y,\ \rho_z$) be the position vector from the origin of the first system to the origin of the seventh system as shown in Figure 6.1; and $\boldsymbol{l}$ ($l_x,\ l_y,\ l_z$), $\boldsymbol{m}$ ($m_x,\ m_y,\ m_z$) and $\boldsymbol{n}$ ($n_x,\ n_y,\ n_z$) be three mutually perpendicular unit vectors aligned with X_7, Y_7, and Z_7 axes, respectively. Further, when $\boldsymbol{\rho}$, $\boldsymbol{l}$, $\boldsymbol{m}$ and $\boldsymbol{n}$ are given in the first system, the equivalent $\boldsymbol{A}$ matrix consists of elements $\boldsymbol{\rho}$, $\boldsymbol{l}$, $\boldsymbol{m}$ and $\boldsymbol{n}$.

By applying coordinate transformation and variable elimination, one can arrive at a system of eight nonlinear equations with eight unknowns expressed in the system of equations given below (Tsai and Morgan 1985) where $1 \le i \le 4$.

$$
\left.
\begin{aligned}
& x^2_i + x^2_{i+1} - 1 = 0 \\
& a_{1i}x_1x_3 + a_{2i}x_1x_4 + a_{3i}x_2x_3 + a_{4i}x_2x_4 + a_{5i}x_5x_7 \\
& + a_{6i}x_5x_8 + a_{7i}x_6x_7 + a_{8i}x_6x_8 + a_{9i}x_1 + a_{10i}x_2 \\
& + a_{11i}x_3 + a_{12i}x_4 + a_{13i}x_5 + a_{14i}x_6 + a_{15i}x_7 \\
& + a_{16i}x_8 + a_{17i}x_8 = 0 \\
& -1 \le x_i \le 1
\end{aligned}
\right\} \qquad (6.1)
$$

The variables x_i and x_{i+1} represent the cosine (c) and sine (s) of the angle of the i^{th} revolute joint to rotate (extended line of $H_{i-1}O_i$ about Z_i so that the extended line $H_{i-1}O_i$ is parallel to H_iO_{i+1}) as illustrated in Figure 6.2. The variables $x_1, x_2, x_3, x_4, x_5, x_6,$, x_7 and x_8 actually represent the c_1, s_1, c_2, s_2, c_4, s_4, c_5 and, s_5 respectively. In equation (6.1), the coefficients a_{ki} are defined as the manipulator parameters. The details are available in Morgan (1987).

6.1.2. Overview of Solution Methods for Kinematics Problems

Three major categories of solution methods have been proposed in kinematics applications (Nielsen and Roth 1997). These are symbolic methods (Elimination, Gröbner basis), approximate numerical methods (continuation method for polynomials) and interval-based techniques. Elimination methods use variable elimination in order to reduce the initial system to a univariate polynomial. The roots of this polynomial are substituted into other equations yielding all solutions of the original system. Elimination method was applied in the 6R inverse kinematics problem (Manocha and Canny 1994) and direct kinematics of Stewart Gough platform (Husty 1996, Innocenti 2001, Lee and Shim 2001). The elimination method requires taking the inverse of a large size coefficient matrix that may lead to numerical instability. The second symbolic method utilized in parallel manipulators (Faugere and Lazard 1995) is the use of the Grobner basis where solutions are searched in another triangular system of equations. A general risk of using symbolic methods is that there might be an explosion in complexity and extraneous roots may be introduced.

Continuation methods (Allgower and Georg 1980) begin with an initial system whose solutions are known, and then transform it gradually to the original system whose solutions are sought, while tracking all solution paths along the way. Researchers

have worked on the improvement of continuation methods and dealt with parametric difficulties related to the adjustment of continuation weights (Wampler et al. 1990, Wampler and Morgan 1991, Recio and Gonzalex-Lopez 1994). An interval approach for achieving reliability in continuation is given by Kearfott and Xing (1994). Some applications of the continuation method are related to inverse 6R manipulator kinematics, direct kinematics of the Stewart-Gough platform and nine-point path synthesis problems for four-bar linkages (Tsai and Morgan 1985, Wampler and Morgan 1991, Nielsen and Roth 1997). A discussion on the progress of continuation methods is given by Sommese et al. (2002).

Interval-based methods (Neumaier 1990, Hansen 1992) are complete and numerically stable algorithms where equations can be entered in their original form without the need of intuition-guided symbolic reductions. Two main classes of interval-based methods have been applied in the robotics field: Interval Newton (Rao et al. 1998, Didrit et al. 1998, Merlet 2001) and Box Consistency techniques (Van-Hentenryck et al. 1997a, Van-Hentenryck et al. 1997b, Merlet 2004). Both methods are used within the basic Interval Partitioning Algorithm (*IP*).

Interval methods can verify reliably that there is no solution in a given sub-set either by interval evaluation of the search space, by an interval Newton method (Hansen 1992) or by local consistency methods (Benhamou et al. 1994). Interval evaluation is the simplest way to declare that a box does not have feasible roots, since it calculates a constraint interval in a given box. Otherwise, if the interval does not contain a function root (the zero), it discards the box. Convergence of this method can be slow due to the overestimation of inclusion function ranges that lead to repetitive bisection of boxes until it is reduced significantly such that the box is discarded. On the other hand, the Interval Newton method has quadratic convergence in finding a single root,

and it is based on narrowing variable intervals in a box using the iterative Newton step that stems from an equation derived from the mean value theorem. If the box has a unique root, Interval Newton method converges to enclose it in a sufficiently small box. This method also detects that the box is infeasible by observing that the resultant variable domain does not intersect with that of the box. The computational burden of this step involves the calculation of the Jacobian and the Gauss-Seidel implementation that is used to solve the resulting set of linear equations. Researchers have applied Interval Newton method to solve problems such as 6R inverse kinematics, direct kinematics of Stewart Gough platform, general single-loop inverse kinematics and singularity analysis and mechanism design of parallel manipulators (Castellet 1998, Didrit et al. 1998, Merlet 2001).

The second interval approach used in kinematics is the local consistency method that is based on narrowing down variable domains by constraint propagation. There are two types of consistency methods: hull and box consistencies. Hull consistency (Benhamou et al. 1994) is basically constraint inversion that uses relational interval arithmetic. It is applicable to more simple constraints due to the dependency problem that takes place in the case of multiple occurrences of the same variable in a constraint expression. The dependency problem is partially eliminated by the box consistency technique (Benhamou et al. 1994, Van-Hentenryck et al. 1997a) that checks the consistency of gradually expanding sub-domains near box boundaries using interval evaluation. In both consistency techniques, reduced variable domains are fed into each constraint and the procedure is repeated resulting in cyclic constraint propagation. If a variable domain is not reduced substantially, it is bisected to result in two new sibling boxes that are inserted into the list of pending boxes waiting to be assessed. Hull and box consistency methods may work in conjunction with each other

and with interval Newton method to gain efficiency. Box consistency implementations in kinematics are found in 6R inverse kinematics (Van-Hentenryck et al. 1997a, Van-Hentenryck et al. 1997b), and Gough type parallel manipulator problem (Merlet 2004), where a comprehensive algorithm that includes 2B and 3B box consistency, linearization, interval Newton, and unicity operators are used. This algorithm uses an available ALIAS library which is an advanced tool containing symbolic, interval, and numerical techniques that deal with root finding in nonlinear systems of equations and inequalities (Merlet 2000).

6.1.3. Numerical Results

Seven challenging kinematics-robotics problems are used in the comparison, one of which is considered as difficult problem (Direct Kinematics) by the *COPRIN* group (*COPRIN*). Here, some of their features are discussed. Direct Kinematics has two close solutions that are hard to isolate. This problem determines the pose parameters of a parallel robot platform and involves eight difficult highly non-linear and inter-dependent trigonometric equations with three independent quadratic equations. Other difficult kinematics problems included here and not covered by *COPRIN* group but solved by *QUAD* are 6R (Kin2) and Stewart Gough. Kin2 is a quadratic problem with 10 real solutions and it describes the inverse position problem for six-revolute-joint. The Stewart Gough involves a manipulator configuration problem that has three totally independent constraints that might make constraint propagation based filtering methods such as 2B ineffective (as verified by Lebbah et al. (2003)). *Numerica* (where Box consistency technique is included) makes more than 10,000,000 narrowing iterations to solve this problem. Another benchmark is the trigonometric Kin1-Modified that describes the inverse kinematics of an elbow manipulator. Puma represents the inverse kinematics of a 3R robot whereas the KinCox is the simple

inverse position problem. Finally, Dietmaier (Dietmaier 1998) is direct kinematics of a general Gough platform with 40 real solutions. In Table 6.1, all test problems are listed with their details (number of dimensions, nonlinear and linear equations) and their source references.

Tables 6.2 and 6.3 illustrate the detailed comparison of results and summary of results obtained with *IIR*, *Rule A*, *Smear* under three tree management approaches and CPU times reported for *ALIAS/QUAD* respectively.

CPU times are indicated in terms of STU's (Scherbina et al. 2002). One STU is equivalent to 388.438 seconds on our machine. All *Rules* are limited to run for at most 0.771 STU's. When a method cannot find all real solutions within this limit, then it is simply indicated as 0.771 in the corresponding CPU time row. The original STU reported by *ALIAS*, *QUAD* and *ICOS* is overwritten.

Table 6.3 provides the average STU's taken (all methods), the number of tree stages the solutions are found in (only for adaptive branching strategy); number of *FSQP* calls; average number of variables partitioned in parallel for *IPA*, the number of function calls outside *FSQP*. It also provide the number of best solutions found (all methods), number of problems that could not be solved during the given time limit (all methods) and the number of problems where *ICOS* or *ALIAS/QUAD* results is not available. The number of Jacobian calculations used by *Smear* rule is displayed in the table. It also illustrates the average percentage of solutions found in each tree level of the adaptive *IPA*. These indicators are summarized as averages in the last block of rows in Table 6.3.

Let us first compare the performance of the three rules in *IPA* under three tree management schemes. On the average, in the adaptive and best-first tree management

approaches *IIR*'s performance is better than *Rule A* and *Smear*. In the depth-first tree management approach, the ranking of the rules are reversed. However, the best results of all rules are obtained under the adaptive scheme. It is also observed that under the adaptive scheme the average percentage of feasible solutions found is higher in early stages of the search tree as compared to *Rule A* and *Smear*. The number of problems where all feasible solutions are not found within the given time limit is smaller in the adaptive scheme as compared to best-first and depth-first approaches.

A general observation on the implementation of the adaptive method is that the CPU times are faster in this approach as compared to other tree management schemes even when the number of *FSQP* calls and the number of function calls outside *FSQP* are larger. There are two main reasons for this: although the maximum number of iterations allowed for *FSQP* is limited to 100 for all versions of *IPA*, the number of iterations taken by the *FSQP* under the adaptive scheme is smaller because it is invoked in the right boxes where feasible solutions actually exist. A second reason is that the adaptive scheme works with a smaller list of boxes to rank due to its stagewise movement in the tree. Therefore, even in cases where the number of functions and *FSQP* calls are the same, the adaptive scheme works faster than the other two tree management methods.

When we observe results in the adaptive scheme on an individual basis, *Rule A* performs best in Kin1-Modified and its performance is equivalent to *IIR*'s in Kincox and Stewart_Gough. In Dietmaier, *IIR* is able to identify all 40 solutions in 1.259 STU's whereas *Smear* finds them much earlier. *Smear* does not work well in trigonometric expressions probably because the Jacobian is less discriminating in this type of functions. In the trigonometric problem Direct, *IIR* requires much less function calls outside *FSQP* than *Rule A*. Furthermore, in the problems where ladder

type of constraints exist (Kin2 and Puma), *IIR* works better than *Rule A*. The reason is that every constraint removes one variable from the previous one and adds a new variable. Thus, *IIR* produces a larger set of variables to re-partition, the degree of parallelism increases, resulting in reduced overestimation in sibling boxes. This puts *IIR* at an advantage over other rules. However, this is not valid in all test problems. The scale of parallelism differs among all *Rules* and a larger scale does not necessarily imply better performance or vice versa. Every rule adjusts its own scale of adaptive parallelism that may change from 2 (minimum number of variables to be partitioned in all *Rules*) to 8.

When we compare the best *IPA* results (*IIR*-adaptive) with filtering/local consistency methods, it is observed that in the ladder type constraints, (PUMA, and KIN2) *ALIAS/QUAD* performs quite well. A good reason for the latter might be that these constraints are very suitable for constraint propagation, each time reducing the domain of one variable. In trigonometric expressions, filtering methods do not produce good results. It is interesting that *QUAD*, which is particularly developed for quadratic problems, is not as successful as *IPA* in the quadratic Stewart Gough. *ICOS* is slower than *ALIAS/QUAD* in the three problems where they may be compared.

# Prob.	Name of the Problem	D, # NE, # LE	Category	Description	# of Sol.	Source
1	**Kin2**	8,8,0	Quadratic	4 quadratic ladder type equations, 4 highly dependent quadratic equations	10	(Lebbah et al. 2003, Van-Hentenryck 1997a)
2	**Kin1-Modified**	6,6,0	Trigonom.	Trigonometric, highly non-linear, high constraint dependency	16	(COPRIN)
3	**KinCox**	4,4,0	Quadratic	Quadratic, 2 constraints independent	2	(COPRIN)
4	**Direct Kinematics**	11,11 ,0	Trigonom.	8 trigonometric, 3 quadratic high constraint dependency	2	(COPRIN)
5	**Stewart-Gough**	9,6,3	Quadratic	6 quadratic, 3 linear constraints, 3 quadratic constraints are independent	2	(Lebbah et al. 2003)
6	**Puma**	8,7,1	Quadratic	4 quadratic ladder type equations, 4 highly dependent quadratic equations	16	(COPRIN)
7	**Dietmaier**	12,12 ,0	Quadratic	12 quadratic equations, high constraint dependency	40	(COPRIN)

Table 6.1 Characteristics of the *CCSP* Applications.

(Note: D: Dimension, NE: Nonlinear Equations, LE: Linear Equations, NIE: Nonlinear Inequality Equations, LIE: Linear Inequalities)

Observing these results, one can say (within the scope of the *CCSP's* tested here) that without advanced symbolic consistency techniques and substitution methods, the proposed adaptive *IPA* (with *IIR* or *Rule A*) is as successful as *ALIAS/QUAD*. *IIR* looks quite promising despite the fact it cannot guarantee immediate reduction of IF_Y in sibling boxes when quadratic and trigonometric expressions exist in the *CCSP* (refer to remarks 4.1 and 4.2 in Chapter 4). One can observe in Table 6.1, that all tested *CCSP's* have quadratic and trigonometric expressions.

# Prob.		Iterative deepening			Worst-first			Depth-first			*ALIAS*	
		IIR Rule	***Rule A***	***Smear***	***IIR Rule***	***Rule A***	***Smear***	***IIR* Rule**	***Rule A***	***Smear***	***QUAD***	***ICOS***
1	**CPU (STU)**	0.128	0.498	**0.089**	0.549	0.771	0.771	0.771	0.771	0.666	0.16	0.771
	No. of stages	2	2	2								(=4.0)
	No. of SQP calls	993	4583	445	730	643	720	8977	6660	1748		
	Avg. no. vars. in parallel(max. ,min.)	5.14(6,4)	4.52(6,2)	3.23(4,3)	5.2(6,5)	4.36(6,2)	3.32(4,3)	3(5,3)	4.18(6,2)	3.27(4,3)		
	No. of function calls	11717	10728	6577	7298	6208	6369	1196312	1139747	23062		
2	**CPU (STU)**	0.062	**0.050**	0.771	0.099	0.198	0.771	0.771	0.771	0.771	0.399	0.771
	No. of stages	4	3	4								(=1.53)
	No. of SQP calls	299	317	5746	254	327	863	5453	2709	5387		
	Avg. no. vars. in parallel(max. ,min.)	4.64(6,3)	3.18(6,2)	5.08(6,4)	4.74(6,3)	3.4(4,2)	5.23(6,4)	3(5,3)	5.64(6,2)	5.1(6,4)		
	No. of function calls	17749	12266	121788	6907	4491	16270	402234	571317	74713		
3	**CPU (STU)**	**0.000**	**0.000**	0.007	0.001	0.001	0.001	0.771	0.009	0.001	**0.000**	0.100
	No. of stages	1	1	1								
	No. of SQP calls	13	3	11	9	11	11	27966	141	13		
	Avg. no. vars. in parallel(max., min.)	2(2,2)	2(2,2)	2(2,2)	2(2,2)	2(2,2)	2(2,2)	2(2,2)	2(2,2)	2(2,2)		
	No. of function calls	65	149	145	65	143	145	619599	3104	193		

# Prob.		Iterative deepening			Worst-first			Depth-first			*ALIAS*	
		IIR Rule	***Rule A***	***Smear***	***IIR Rule***	***Rule A***	***Smear***	***IIR Rule***	***Rule A***	***Smear***	***QUAD***	***ICOS***
4	**CPU (STU)**	**0.009**	0.038	0.771	0.021	0.504	0.110	0.771	0.771	0.068	0.77	NA
	No. of stages	1	2	3							(=12.6)	
	No. of SQP calls	20	126	1305	20	380	116	2319	2493	66		
	Avg. no. vars. in parallel(max., min.)	2.5(3,2)	2.79(4,2)	3(3,3)	2.39(3,2)	2.78(3,2)	3(3,3)	2.7(3,2)	3.09(5,2)	3(3,3)		
	No. of function calls	265	2545	15665	265	7033	1409	87001	182916	881		
5	**CPU (STU)**	**0.002**	**0.002**	**0.002**	0.005	0.004	0.005	0.003	0.006	**0.002**	0.277	NA
	No. of stages	1	1	1								
	No. of SQP calls	13	13	12	13	13	12	10	21	8		
	Avg. no. vars. in parallel(max. ,min.)	2(2,2)	2(2,2)	2(2,2)	2(2,2)	2(2,2)	2(2,2)	2(2,2)	2.5(3,2)	2(2,2)		
	No. of function calls	145	145	145	145	145	145	145	217	145		
6	**CPU (STU)**	0.003	0.008	0.771	0.009	0.015	0.771	0.020	0.019	0.771	**0.001**	0.059
	No. of stages	1	1	2								
	No. of SQP calls	28	75	1589	145	77	860	99	105	900		
	Avg. no. vars. in parallel(max., min.)	6.07(7,6)	3.1(6,2)	5.5(7,2)	6.2(7,6)	3.23(6,2)	5.3(7,2)	6.6(8,6)	2.34(6,2)	5.1(7,2)		
	No. of function calls	2741	857	15109	2741	933	8297	2866	2350	8121		

Contd…

# Prob.		Iterative deepening			Worst-first			Depth-first			*ALIAS*	
		IIR Rule	***Rule A***	***Smear***	***IIR Rule***	***Rule A***	***Smear***	***IIR Rule***	***Rule A***	***Smear***	***QUAD***	***ICOS***
7	**CPU (STU)**	0.771	0.771	0.418	0.771	0.771	0.591	0.771	0.771	0.519	**0.039**	NA
	No. of stages	3	2	2								
	No. of SQP calls	4680	1182	1195	757	1429	414	2448	2178	553		
	Avg. no. vars. in parallel(max. ,min.)	4.31(8,3)	4.4(10,2)	4.81(6,4)	4.23(6,3)	4.23(10,2)	2(2,2)	4.51(6,3)	2.5(10,2)	4.94(6,4)		
	No. of function calls	51568	25131	68226	12777	23935	5882	553308	791854	15549		

Table 6.2. Comparison of results for *CCSP* Applications

Contd...

	***IP* - Adaptive tree management**			***IP* - Worst-First tree management**		
	IIR Rule	***Rule A***	***Smear***	***IIR Rule***	***Rule A***	***Smear***
Avg. CPU (STU)	0.139	0.195	0.404	0.208	0.323	0.431
Avg. no. of stages	1.86	1.71	2.14	-	-	-
Avg. no. of SQP calls	863.71	899.86	1471.86	258.71	411.43	428.00
Avg. no. of function calls	12035.71	7403.00	3522.14	4314.00	6126.86	5502.43
No. of unsolved problems	1	1	3	1	2	2
% of unsolved problems	14.28	14.28	42.86	14.28	28.58	28.58
No. of best solutions found	3	3	2	0	0	0
% of best solutions found	42.86	42.86	28.58	0	0	0
Data not available	0	0	0	0	0	0
Avg. percentage of solutions found						
Stage 1	71.96	65.35	57.50	-	-	-
Stage 2	12.68	16.78	25.54	-	-	-
Stage 3	9.52	5.35	1.65	-	-	-
Stage 4	1.79	-	1.65	-	-	-
Total	95.59	87.49	86.33	94.29	86.43	82.86

	***IP* - Depth-First tree management**			***ALIAS***	***ICOS***
	IIR Rule	***Rule A***	***Smear***	***QUAD***	
Avg. CPU (STU)	0.554	0.445	0.400	0.235	0.425
Avg. no. of stages	-	-	-	-	-
Avg. no. of SQP calls	6753.14	2043.86	1239.29	-	-
Avg. no. of function calls	48780.7	384500.7	17253.43	-	-
No. of unsolved problems	5	4	2	1	2
% of unsolved problems	71.42	57.15	28.58	14.28	50*
No. of best solutions found	0	0	1	3	0
% of best solutions found	0	0	14.28	42.86	0
Data not available	0	0	0	0	3
Avg. percentage of solutions found					
Stage 1	-	-	-	-	-
Stage 2	-	-	-	-	-
Stage 3	-	-	-	-	-
Stage 4	-	-	-	-	-
Total	54.64	56.25	84.82		

Table 6.3. Summary of Results for Kinematics benchmarks

* Percentage is calculated using the actual number of problems (Actual number of problems = total no. of problems – no. of problems for which data is not available.)

6.2. Applications – Constrained Optimization Problem

Many important real world problems can be expressed in terms of a set of nonlinear constraints that restrict the domain over which a given performance criterion is optimized, that is, as a Constrained Optimization Problem (*COP*). In the general *COP* with a non-convex objective function, discovering the location of the global optimum

is NP-hard. Hence, locating feasible solutions in a non-convex feasible space is also NP-hard. Solution approaches using derivatives developed for solving the *COP* might often be trapped in infeasible and/or sub-optimal sub-spaces if the combined topology of the constraints is too rugged. The same problem exists in the discovery of global optima in non-convex bound-constrained global optimization problems. The *COP* has augmented complexity as compared to bound-constrained problems due to the restrictions imposed by highly non-linear relationships among variables.

Testing environment

All runs are executed on a PC with 256 MB RAM, 2.4 GHz P4 Intel CPU, on Windows platform. The *IP* code is developed with Visual C++ 6.0 interfaced with *PROFIL* interval arithmetic library (Knuppel 1994) and *FSQP*.

6.2.1. Description of the Problem

The following applications have been selected to test the performance of the proposed *IP*.

1. Planar truss design (Hsu et al. 2003)

Consider the planar truss with parallel chords shown in Figure 6.3 under the action of a uniformly distributed factored load p = 25kN/m, including the dead weight of approximately 1 kN/m. The truss is constructed from bars of square hollowed cross-section made of steel 37. For chord members, limiting tensile stresses are 190 Mpa and for other truss members 165 MPa.

The members are divided into four groups according to the indices shown in Figure 6.3. The objective of this problem is to minimize the volume of the truss, subject to stress and deflection constraints. Substituting material property parameters and the maximum allowable deflection that is 3.77cm, the optimization model can be

simplified. However, the original model is a discrete constrained optimization problem (Hsu et al. 2003), which is converted into a continuous optimization problem described below.

Maximize $f = 0-(600*a_1+2910.4*a_2+750*a_3+1747.9*a_4)$ (cm^3)

Subject to:

$a_1 \geq 30.0cm^2$

$a_2 \geq 24.0cm^2$

$a_3 \geq 14.4cm^2$

$a_4 \geq 11.2cm^2$

} Stress constraints

$313920/A_1+497245/A_2+22500/A_3+67326/A_4 \leq 25200$(kN-cm) (deflection constraint)

Search space: A_1=[30, 1000]; A_2=[24, 1000]; A_3=[14.4, 1000]; A_4=[11.2, 1000].

Here, a_i is the area in cm^2 with indices i=1, 2, 3, 4. The objective function is a simple linear function, but the deflection constraint turns the feasible domain into a non-convex one.

Hsu et al. (2003) report an optimal design point for the original discrete model as a = (55, 37.5, 15, 15), and the minimum volume of the truss for this solution is 179608.5. However, for a continuous model, we find a minimum volume of the truss as 176645.373 using the *IP* and other solvers used in the comparison.

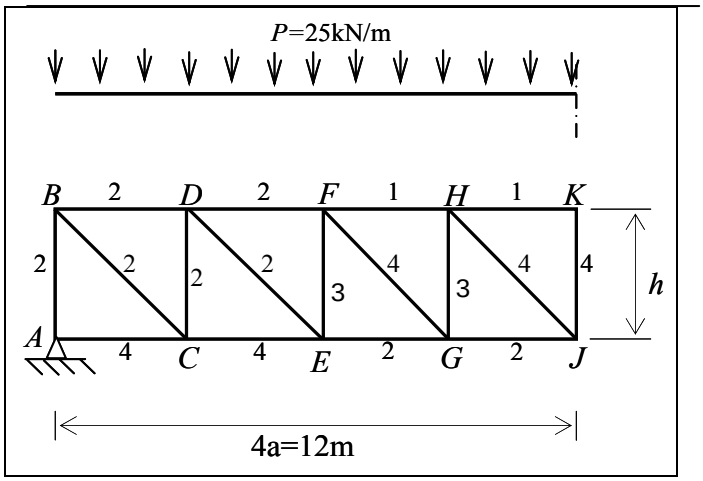

Figure 6.3. Optimal design of a planar truss with parallel chords

Source: Hsu et al. 2003

2. Pressure Vessel Design (Hsu et al. 2003)

Figure 6.4 shows a cylindrical pressure vessel capped at both ends by hemispherical heads. This compressed air tank has a working pressure of 3,000 psi and a minimum volume of 750 feet^3. The design variables are the thickness of the shell and head, and the inner radius and length of the cylindrical section. These variables are denoted by x_1, x_2, x_3, and x_4, respectively. The objective function to be minimized is the total cost, including the material and forming costs expressed in the first two terms, and the welding cost in the last two terms. The first constraint restricts the minimum shell thickness and the second one, the spherical head thickness. The 3rd and 4th constraints represent minimum volume required and the maximum shell length of the tank respectively. However, the last constraint is redundant due to the given search domain. The original model for this application is again a discrete constrained optimization problem. The continuous model is provided below.

Maximize $f = 0-(0.6224*x_1*x_3*x_4 + 1.7781*x_1*x_3^2 + 3.1661*x_1^2*x_4 + 19.84*x_1^2*x_3)$

Subject to:

$-x_1 + 0.0193*x_3 \leq 0$

$-x_2 + 0.00954*x_3 \leq 0$

$(-\pi*x_3^2*x_4 - (4\pi*x_3^3)/3)/1296000 + 1 \leq 0$

$x_4 - 240 \leq 0$

Search Space: $X_1 = [1.125, 2]$; $X_2 = [0.625, 2]$; $X_3 = [40, 60]$; $X_4 = [40, 120]$

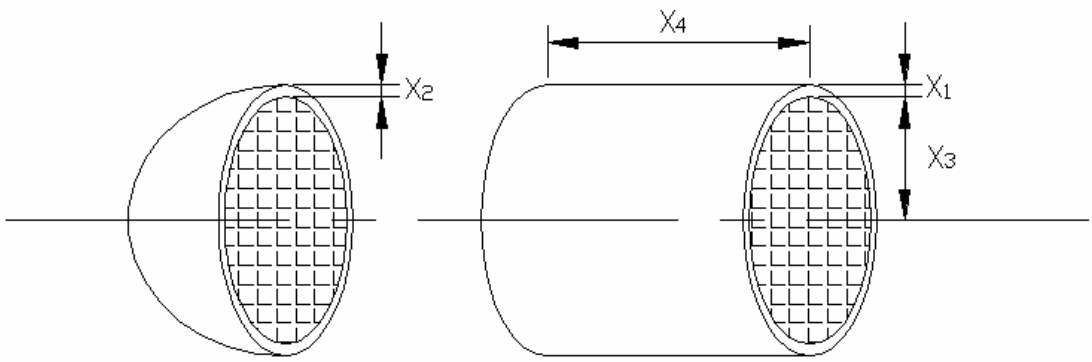

Figure 6.4. Pressure vessel design.

Source: Hsu et al. 2003

Hsu et al. (2003) list the reported optimal costs obtained by different formulations as given in Table 6.4.

The least cost reported by *IIR* for designing a pressure vessel subjected to the given constraints is 7198.006. Other solvers report the same objective function value.

Problem Formulation	Reported optimal solution	Reference
Continuous	-7198.01	Hsu et al. (2003)
Discrete	-7442.02	Hsu et al. (2003)
	-8129.14	Sandgren (1988)
Mixed discrete	-7198.04	Kannan and Kramer (1994)

Table 6.4. Reported optimal costs obtained by different formulations for pressure vessel design (Hsu et al. 2003)

3. Simplified Alkylation Process (Berna et al. 1980, PrincetonLib)

This model describes a simplified alkylation process. The nonlinearities are bounded in a narrow range and introduce no additional computational burden.

The design variables for the simplified alkylation process are olefins feed, isobutane recycle, acid feed, alkylate yield, isobutane makeup, acid strength, octane number, iC4 olefin ratio, acid dilution factor, F4 performance number, alkyate error, octane error, acid strength error, and F4 performance number error. The objective function maximizes profit per day. The constraints represent alkylate volumetric shrinkage equation, acid material balance, isobutane component balance, alkylate definition, octane number definition, acid dilution factor definition, and F4 performance number definition. The model is provided below.

Maximize $f = 5.04*x_3*x_4 + 0.35*x_{13} + 3.36*x_{14} - 6.3*x_1*x_2$

Subject to:

$x_1 - 0.81971\, x_3 - 0.81967\, x_{14} = 0$

$-3x_2 + x_8*x_{12} = -1.33$

$22.2*x_8 + x_7* x_{11} = 35.82$ (acid material balance)

$-0.325*x_5 - 0.01098*x_6 + 0.00038*x_6^2 + x_2*x_{10} = 0.57425$

$0.98*x_4 - x_5*(x_4 + 0.01\, x_1*x_7) = 0$

$x_1*x_9 - x_3*(0.13167*x_6 - 0.0067*x_6^2 + 1.12) = 0$

$10*x_{13} + x_{14} - x_3*x_6 = 0$ (isobutane component balance)

Search space:

X_1= [1, 5]; X_2= [0.9, 0.95]; X_3= [0, 2]; X_4= [0, 1.2]; X_5= [0.85, 0.93]; X_6= [3, 12];

X_7= [1.2, 4]; X_8= [1.45, 1.62]; X_9= [0.99, 1.0101]; X_{10}= [0.99, 1.0101];

X_{11}= [0.9, 1.112]; X_{12}= [0.99, 1.0101]; X_{13}= [0, 1.6]; X_{14}= [0, 2].

The optimal solution for alkylation process 1.765 (PrincetonLib).

4. Stratified Sample Design (PrincetonLib)

The problem is to find a sampling plan that minimizes cost and yields variances of the population limited by an upper bound.

Maximize f= $0-((x_1 + x_2)+(x_3 + x_4))$

Subject to

$(0.16/x_1)+(0.36/x_2)+(0.64/x_3)+(0.64/x_4)-0.010085 \leq 0$

$(4/x_1)+(2.25/x_2)+(1/x_3)+(0.25/x_1)-0.0401 \leq 0$

Search space:

X_1= [100, 400000]; X_2= [100, 300000]; X_3= [100, 200000]; X_4= [100, 100000]

The optimal solution for this problem is -725.479 (PrincetonLib).

*5. Robot (*Benhabib et al. 1987, PrincetonLib)

This model is designed for the analytical trajectory optimization of a robot with seven degrees of freedom.

Maximize f = $0-((x_1-x_8)^2+(x_2-x_9)^2+(x_3-x_{10})^2+(x_4-x_{11})^2+(x_5-x_{12})^2+(x_6-x_{13})^2+(x_7-x_{14})^2)$

Subject to

$cos(x_1)+ cos\ (x_2)+ cos\ (x_3)+ cos\ (x_4)+ cos\ (x_5)+ cos\ (x_6)+0.5*cos\ (x_7) =4$

$sin\ (x_1)+ sin\ (x_2)+ sin\ (x_3)+ sin\ (x_4)+ sin\ (x_5)+ sin\ (x_6)+0.5*sin(x_7) = 4$

Search space: $X_i=[-10, 10]$; $i = 1, 2, 3, 4, \ldots, 14$.

The optimal solution for this problem is 0.0(PrincetonLib).

6.2.2. Overview of Solution Methods

Existing global optimization algorithms can be categorized as deterministic and stochastic methods. Extensive surveys on global optimization exist in the literature (Törn and Zilinskas 1989, and recently by Pardalos and Romeijn 2002). Although we cannot cover the *COP* literature in detail within the scope of this paper, we can cite deterministic approaches including Lipschitzian methods (Hansen et al. 1992, Hansen and Jaumard 1995, Pinter 1997), branch and bound methods (Al-Khayyal and Sherali 2000), cutting plane methods (Tuy et al. 1985), outer approximation (Horst et al. 1992), primal–dual method (Floudas and Visweswaran 1993, Ben-Tal et al. 1994), alpha-Branch and Bound approach (Androulakis et al. 1995), reformulation techniques (Smith and Pandelides 1999), interior point methods (Morales et al. 2001, Forsgren et al. 2002) and interval methods (Hansen 1992, Kearfott 1996c, Csendes 1997).

We show, on a test bed of practical applications, that the *IIR* with adaptive tree management is a viable method in solving the general *COP* with equalities and inequalities. The results obtained are compared with commercial software such as *Baron*, *Minos* and other solvers interfaced with *GAMS* (www.gams.com).

6.2.3. Numerical Results

The numerical results are provided in Table 6.4. We compare *IP* results with five different solvers that are linked to the commercial software *GAMS* (www.gams.com) and *FSQP* (Zhou and Tits 1996, Lawrence et al. 1997) code provided by *AEM* (www.aemdesign.com/FSQPmanyobj.htm). The solvers used in this comparison are *Baron* (Sahinidis 2003), *Conopt* (Drud 1996), *LGO* (Pinter 1997), *Minos* (Murtagh and Saunders 1987) and *Snopt* (Gill et al. 1997).

For each application we report the absolute deviation from the global optimum obtained at the end of the run and the CPU time taken for each run in Standard Time Units (STU, Scherbina et al. 2002). One STU is equivalent to 229.819 seconds on our machine. *GAMS* solvers are run until each solver terminates on its own without restricting the CPU time taken or the number of iterations. *FSQP* is run with a maximum number of iterations allowed, that is 100 in this case. However, in these applications *FSQP* never reaches this iteration limit. *IP* is run until no improvement in the *CLB* is obtained as compared with the previous stage of the search tree. However, if a feasible solution has not been found yet, the stopping criterion becomes the least feasibility degree of uncertainty, INF_Y.

In Table 6.5, we report additional information for *IP*. For each application, we report the number of tree stages where *IP* stops, the number of times *FSQP* is invoked, the average number of variables partitioned in parallel for a parent box (the maximum and minimum numbers are also indicated in parenthesis), and the number of function calls invoked outside *FSQP*. We provide two summaries of results obtained excluding and including the robot application. The reason is that *Baron* is not enabled to solve trigonometric models.

While analyzing results, we observe that *Snopt* identifies the optimum solution for four of the applications excluding the robot. However, its performance is inferior for the robot as compared to *IP* and *FSQP*. In the robot application, *FSQP* identifies the global optimum solution in the initial box itself (stage zero in *IP*). That is why *IP* stops at the first stage. The performance of the local optimizers, *Minos* and *Conopt*, is significantly inferior in this problem. In the pressure vessel and planar truss applications, all *GAMS* solvers, *IP* and *FSQP* identify the global optimum taking short CPU times (*IP* and *Baron* take longer CPU time). In Alkyl, *Minos* is stuck at a local stationary point while *BARON* and IP take longer CPU times. In the Sample application, *LGO* does not converge and *FSQP* ends up with a very inferior solution. On the other hand, *IP* runs for 6 tree stages and results in an absolute deviation that is compatible with those of *Baron*, *Conopt* and *Minos*.

When the final results summary is analyzed, we observe that *IP's* performance is as good as *Baron's* (which is a complete and reliable solver) in identifying the global optimum and CPU time. The use of *FSQP* in *IP* rather (rather than the Generalized Reduced Gradient local search procedure available in *Baron*) becomes an advantage for *IP* in the Robot. Furthermore, *IP* does not have any restrictions in dealing with trigonometric functions. The impact of interval partitioning on performance is particularly observed in the Sample application where *FSQP* fails to converge. For these applications, the number of tree stages that *IP* has to run for is quite small (two) except for the Sample. The average number of variables partitioned in parallel in *IP* varies between 2 and 4. The dynamic parallelism imposed by the weighting method seems to be effective as it is observed that different scales of parallelism are adopted for different applications.

Prob.	Dim.	Performance	*IIR*	*FSQP*	*Baron*	*Conopt*	*LGO*	*Minos*	*Snopt*
1	4	Deviation	0.000	0.000	0.000	0.000	0.000	0.000	0.000
		CPU(STU)	0.000	0.000	0.001	0.000	0.000	0.000	0.000
2	4	Deviation	0.000	0.000	0.000	0.000	0.000	0.000	0.000
		CPU(STU)	0.001	0.000	0.001	0.000	0.000	0.000	0.000
3	14	Deviation	0.000	0.000	0.000	0.000	0.000	1.765	0.000
		CPU(STU)	0.263	0.000	0.292	0.000	0.003	0.000	0.000
4	4	Deviation	1.200	26706.5	1.158	1.201	∞	1.168	0.000
		CPU(STU)	0.056	0.000	0.000	0.000	0.002	0.000	0.000
Summary									
Avg. deviation			0.300	6676.63	0.290	0.300	0.000	0.733	0.000
Std. dev. for it			0.600	13353.25	0.579	0.601	0.000	0.881	0.000
Avg. CPU time			0.080	0.000	0.073	0.000	0.001	0.000	0.000
# best solutions			3	3	3	3	3	2	4
% of best solutions			75	75	75	75	75	50	100
# unsolved problems			0	0	0	0	1	0	0
% of unsolved problems			0	0	0	0	25	0	0
5	14	Deviation	0.000	0.000	NA	27.1	5.463	343.022	13.391
		CPU(STU)	0.000	0.000		0.000	0.046	0.001	0.000
Final Summary									
Avg. deviation			0.240	5341.300	0.299	5.659	1.366	69.191	2.678
Std. dev. for it			0.537	11943.510	0.579	11.994	2.732	153.078	5.989
Avg. CPU time			0.064	0.000	0.073	0.000	0.010	0.000	0.000
# best solutions			4	4	3	3	3	2	4
% of best solutions			80	80	60	60	60	40	80
# unsolved problems			0	0	1	0	1	0	0
% of unsolved problems			20	0	20	0	20	0	0

Table 6.5. Comparison of Results for constrained optimization applications.

The problem names (global optimum values) are Pressure Vessel (-7198.006), Planar Truss (-176645.373), Alkyl (1.765), and Sample (-725.479), respectively. For the *IIR* method the number of stages, *FSQP* calls, and the average number of variables in parallel (maximal/minimal), and the number of function calls were (2, 18, 3.04/4.2, 402), (2, 27, 3/4.2, 257), (2, 631, 4.04/5.3, 8798), and (6, 917, 3.56/4.3, 12000), respectively. The summary of the average results for the first 4 problems is: for the number of stages is 3.000, the number of *FSQP* calls is 398.250, and the average number of function calls is 5364.25. For the problem 5, robot optimization, the global optimum was zero, and the efficiency measures (1, 1, 2.25/3.2, 62). The final summary provides the following average figures: the number of stages is 2.600, the

number of *FSQP* calls is 318.800, and the average number of function calls is 4303.800.

6.3. Summary of results

Table 6.6 illustrates the performance of the best strategies recommended for solving *CCSP* and *COP* based applications from the field of kinematics, robotics, engineering design and so on. It also illustrates the maximum size of the problems tested here.

Problem Class		**Recommended Strategy**	**Performance measure**	**Performance value**	**Maximum problem dimension**
CCSP	Best of IP configuration	*IIR*_adaptive tree management	CPU time	0.139	11
	Best of other methods /solvers	*ALIAS / QUAD*	CPU time	0.235	12
COP	Best of IP configuration	*IIR*_adaptive_non-penalty	Dev. From optimality	0.240	14
	Best of other methods / solvers	*Baron*	Dev. From optimality	0.290	14

Table 6.6. Summary of computational results on applications

In case of *CCSP* applications, we observe that *IIR*_adaptive tree management method is the best *IIR* configuration and its performance is superior (almost half) as compared to, that of *ALIAS/QUAD*.

For the *COP* applications, *IIR*_adaptive_non-penalty is the best *IIR* configuration and its performance similar to *Baron's*. The average deviation from the optimal solution for *Baron* is close to *IIR*_adaptive tree management with non-penalty box ranking. The overall performance of *IIR* is superior in solving *CCSP* application problems. However, in case of the *COP* applications, *IIR* is almost as effective as *Baron*.

Chapter 7

Conclusions and Future Recommendations

7.1. Conclusions

Global Optimization Problems are encountered in many scientific fields concerned with industrial applications such as kinematics, chemical process optimization, molecular design, and so on. When non-linear relationships among variables are defined by problem constraints resulting in non-convex feasible spaces the problem of identifying feasible solutions may become very hard. Consequently, finding the location of the global optimum in the problem is more difficult. This research develops a generic methodology that can solve *BCOP*, *CCSP* and *COP*. A new subdivision direction selection rule (*IIR*) has been proposed in this research for bound constrained optimization, continuous constraint satisfaction and constrained optimization problems.

A variant of *IIR*, *IIR_Widths*, has also been proposed for bound constrained optimization problems. The new variant considers interval width as well as sub-expression bounds. In the *BCOP*, the proposed two rules target directly on the uncertainty degree of the objective function with respect to the optimality. Reducing these uncertainties as such results in the reliable detection of sub-optimal boxes, thereby diminishing the number of boxes to be assessed.

The efficiency of the proposed variants is illustrated on well-known bound constrained test functions and compared with established subdivision direction selection methods from the literature.

For constraint satisfaction and constrained optimization problems a new adaptive search tree framework where nodes (boxes defining different variable domains) are explored using a restricted hybrid depth-first and best-first branching strategy has been proposed. This hybrid approach has also been used for activating local search in boxes with the aim of identifying different feasible stationary points. The proposed search tree management approach improves the convergence of the interval partitioning method that is also supported by the new parallel subdivision direction selection rule.

In the *CCSP* and *COP,* the proposed rule targets directly the uncertainty degrees of constraints (with respect to feasibility) and the uncertainty degree of the objective function (with respect to optimality). Reducing these uncertainties as such results in the early and reliable detection of infeasible and sub-optimal boxes, thereby diminishing the number of boxes to be assessed. Consequently, chances of identifying local stationary points during the early stages of the search increase.

For continuous constraint satisfaction problems, the effectiveness of the proposed interval partitioning algorithm has been compared with the published results of established symbolic-numeric methods for solving *CCSP* on a number of state-of-the-art benchmarks. The effectiveness has also been illustrated on several practical applications.

For constrained optimization problems, the effectiveness of the proposed interval partitioning algorithm has been illustrated on several state of the art benchmark problems and also several practical applications and compared with professional commercial local and global solvers. Empirical results have shown that this approach is as good as available *COP* solvers.

The contribution of this research can be briefly summarized as follows:

- A generic approach, *IIR,* has been proposed for *BCOP*, *CCSP*, and *COP*. This approach makes use of information derived from the problem structure and no additional information other than function ranges is needed.
- Numerical tests on a wide range of benchmark problems and commercial optimization software show that *IIR* is successful in solving *BCOP* and *COP* as well as the *CSP*s.
- In case of the *BCOP*, the average CPU time required for *IIR_Widths* is almost one-fourth of *Rule B*. Similarly, in the *CCSP*, the average CPU time required for IIR_adaptive tree management is half of *ALIAS / QUAD's*.
- For the *COP* (non-trigonometric functions), the average deviation from the optimal solution for Baron is close to *IIR*_adaptive tree management with non-penalty box ranking. However, the difference in performance is not statistically significant (at a 5% significance level). For trigonometric *COP* problems, the average deviation from optimal solution for *IIR*_adaptive_non-penalty is almost of the 1/1000 of *Snopt*.
- The adaptive tree management strategy proposed here can also be used in non-interval partitioning algorithms such as *Baron* and *LGO*. It is effective in the sense that it allows going deeper into selected promising parent boxes while providing a larger perspective on how promising a parent box is by comparing it to all other boxes available in the current stage's box list.

7.2. Recommendations for Future Research

Future work that merits further investigation requires the following feature developments.

i. Ranking constraints in a constrained global optimization problem

One of the basic problems that most traditional techniques face in solving a non-convex constrained optimization or constraint satisfaction problem is the slow convergence leading to unacceptable response times for constraint systems. Feasible solutions for constraint systems are found by constraint propagation methods which reduce variable ranges by sequential substitution. The main reason behind the slow convergence of constraint propagation methods is the waste of computations on an inappropriate constraint, which does not result in a good reduction in the search space (Lhomme et al. 1998). This brings about motivation to develop new approaches by which a ranking approach is proposed in solving the constrained optimization problem or constraint satisfaction problems before processing them. The new approach will define a performance index for each constraint based on the pending status and sub-expression complexity of the constraint. Then, constraints will be sorted based on their degree of uncertainty and sub-expression complexity. This prior step will improve the efficiency of any methodology used for solving the *COP* or *CCSP* irrespective of the application field.

ii. Dealing with discrete problems

Vaidyanathan et al. (1996) propose a methodology for solving discrete problems using interval analysis, which is a modification made to the basic constrained optimization problem solving algorithms. However, research in this field is very limited.

The development of new methodologies for solving nonlinear discrete problems will definitely be a good contribution in this field. The discrete domain defined for a given global optimization problem can be assumed as a continuous domain (box). However,

the partitioning of the domain would be modified based on the discrete values defined for a given variable. This new approach may help in faster convergence because of shorter pending lists preserved when compared to a traditional discrete solution algorithm. Also, the new adaptive tree management proposed in this thesis would reduce the limited use of memory and CPU time for large-scale nonlinear discrete problems.

This problem has many industrial applications in different fields such as chemical engineering, power transmission, planning, and so on, which need extensive computations with existing solution techniques.

iii. Analysis of different merit functions for pending box ranking

The current merit function defined in this thesis is a static penalty function. However, it is worthwhile to try different merit functions such as dynamic and adaptive penalty functions defined in the literature (Joines and Houck 1994, Michalewicz and Attia 1994, Carlson et al. 1998, Morales and Quezada 1998, Yeniay 2005). The merit function an importance element that may improve algorithm efficiency in the box selection procedure. It may also improve the convergence of the algorithm through faster deletion of infeasible subspaces.

The new merit function analysis will provide a robust merit function for the methodology defined in this thesis. This will help the algorithm to solve large-scale nonlinear nonconvex problems.

iv. Extension to mixed integer nonlinear programming problems

Vaidyanathan et al. (1996) propose a methodology for solving discrete problems using interval analysis, which is a modification made to the basic constrained

optimization problem solving algorithms. However, research in this field is very limited.

Similar to pure problems described previously, the discrete domain defined for a given problem can be assumed as a continuous domain like any other continuous domain, but partitioning of the discrete variables would be modified while continuous variables would be partitioned as usual.

v. Extension of adaptive tree management strategy to other solvers

Tree management is one of the factors that influences the efficiency of any partitioning algorithm. The new adaptive tree management strategy proposed in this thesis can also be used in non-interval partitioning algorithms such as *Baron* and *LGO*. It is effective in the sense that it allows going deeper into selected promising parent boxes while providing a larger perspective on how promising a parent box is by comparing it to all other boxes available in the current stage's box list.

References

Aarts, E., and Korst, J. (1989). *Simulated annealing and boltzmann machines*. J. Wiley and Sons.

Adjiman, C.S., Dallwig, S., Floudas, C.A., and Neumaier, A. (1998a). A global optimization method: alphaBB-for general twice-differentiable constrained NLPs - I. Theoretical Advances. *Computers and Chemical Engineering*, 22: 1137-1158.

Adjiman, C.S., Androulakis, I.P., and Floudas, C.A. (1998b). A global optimization method, alphaBB, for general twice-differentiable constrained NLP's: II - Implementation and Computational Results. *Computers and Chemical Engineering,* 22: 1159 – 1179.

Adjiman, C.S., and Floudas C.A. (2001). The alphaBB global optimization algorithm for nonconvex problems: an overview. In Migdalas, A., Pardalos, P.M., Varbrand, P., editors, *From local to global optimization*, London, Kluwer Academic, Pages: 155 - 186

Alefeld, G., and Herzberger, J. (1983). *Introduction to interval computations*. Academic Press Inc. New York, USA.

Allgower, E.L., and Georg, K. (1980). Simplicial and continuation methods for approximating fixed points and solutions to systems of equations. *SIAM Review*, 22: 28-85.

Al-Khayyal, F.A., and Sherali, H.D. (2000). On finitely terminating branch-and-bound algorithms for some global optimization problems. *SIAM Journal on Optimization*, 10: 1049-1057.

Anderson, N., and Walsh, G. (1986). A graphical method for a class of branin trajectories. *Journal of Optimization Theory and Applications*, 49: 367-374.

Androulakis, I.P., Maranas, C.D., and Floudas, C.A. (1995). αBB: A global optimization method for general constrained nonconvex problems. *Journal of Global Optimization*, 7: 337-363.

Arminjo, L.(1966). Minimization of functions having continuous partial derivates. *Pacific J. Math.,* 16:1-3.

Baldi, P. (1995). Gradient descent learning algorithm overview: a general dynamical systems perspective. *IEEE Transactions on Neural Networks*, 6: 182-195.

Balogh, J., and Tóth, B. (2005) Global optimization on stiefel manifolds: a computational approach. *Central European Journal of Operations Research*, 13: 213-232.

Baritompa, W. (1993). Customized methods for global optimization – a geometric view point. *Journal of Global Optimization*, 3: 193-212.

Baritompa, W., and Cutler A. (1994). Accelerations for global optimization covering methods. *Journal of Global Optimization*, 4: 329-341.

Bazaraa, M.S., Sherali, H.D., and Shetty, C.M. (1993). *Nonlinear Programming. Theory and Algorithms*, John Wiley & Sons, Inc., New York.

Beckernad, R.W., and Lago, G.W. (1970). A global optimization algorithm. In *Proc. of the 8th Allerton Conf. on Circuits and Systems Theory*, pp. 3-12, Monticello, Illinois.

Benhamou, F., McAllester, D., and Van Hentenryck, P. (1994). CLP (Intervals) revisited. *Proceedings of ILPS'94, International Logic Programming Symposium*, pp. 124-138.

Benhamou, F., and Older, W.J. (1997). Applying interval arithmetic to real, integer and boolean constraints. *Journal of Logic Programming*. 32: 1-24.

Benhamou, F., Goualard, F., Granvilliers, L., and Puget, J-F. (1999). Revising hull and box consistency. *Proc. of 16th Int. Conf. on Logic Programming*, 230-244, Los Cruces, USA.

Benhabib, B., Fenton, R.G., and Goldberg, A.A. (1987). Analytical trajectory optimization of seven degrees of freedom redundant robot. *Transactions of the Canadian Society for Mechanical Engineering*, 11: 197-200.

Ben-Tal, A., Eiger, G., and Gershovitz, V. (1994). Global optimization by reducing the duality gap. *Math. Prog.*, 63: 193-212.

Berna, T., Locke, M., and Westerberg, A.W. (1980). A new approach to optimization of chemical processes. *American Institute of Chemical Engineers Journal*, 26: 37-43.

Berner, S. (1996). New results on verified global optimization. *Computing*, 57: 323-343.

Betro, B., and Schoen, F. (1987). Sequential stopping rules for multistart algorithm in global optimization. *Mathematical Programming*, 38: 271-286.

Betro, B., and Schoen, F. (1992). Optimal and sub-optimal stopping rules for the multistart algorithm in global optimization. *Mathematical Programming*, 57: 445-458.

Boender, C.G.E., Rinnooy kan, A.H.G., Stougie, L., and Timmer, G.T. (1982). A stochastic method for global optimization. *Mathematical programming*, 22: 125-140.

Boender, C.G.E., and Rinnooy kan, A.H.G. (1987). Bayesian stopping rules for multi-start global optimization. *Mathematical programming*, 37: 59 - 80.

Boender, C.G.E., and Rinnooy kan, A.H.G. (1991). On when to stop sampling for the maximum. *Journal of Global Optimization*, 1: 331-340.

Bonnans, J.F., Panier, E., Tits, A.L., and Zhou, J.L. (1992). Avoiding the maratos effect by means of a nonmonotone line search: II. Inequality Problems - Feasible Iterates. *SIAM Journal on Numerical Analysis*, 29: 1187-1202.

Breiman, L. and Cutler, A. (1993). A deterministic algorithm for global optimization. *Mathematical Programming*, 58: 179-199.

Broyden, C.G. (1972). Quasi-newton methods. In Murray, W., eds., *Numerical methods for Unconstrained Optimization*, pp. 87-106, Academic Press, New York.

Buchberger, B. (1985). Grobner bases: an algorithmic method in polynomial ideal theory. In *Mutidimensional Systems Theory*, pp. 184-232, D. Reidel Publishing Co.

Burkard, R.E., Hamacher, H., and Rote, G. (1992). Sandwich approximation of univariate convex functions with an application to separable convex programming. *Naval Research Logistics*, 38: 911-924.

Caprani, O., Godthaab, B., and Madsen, K. (1993). Use of a real-valued local minimum in parallel interval global optimization. *Interval Computations*, 2: 71–82.

Carlson, S., Shonkwiler, R., Babar, S., and Aral, M. (1998). Annealing a genetic algorithm over constraints. *SMC 98 Conference*. (http://www.math.gatech.edu/shenk/body.html.)

Castellet, A. (1998). *Solving inverse kinematics problems using an interval method.* PhD Thesis, Universitat Politechnica de Catalunya, Barcelona, Spain.

Casado, L.G., García, I., and Csendes, T. (2000). A new multisection technique in interval methods for global optimization. *Computing*, 65: 263-269.

Casado, L.G., Martinez, J.A., and Garcia, I. (2001a). Experiments with a new selection criterion in a fast interval optimization algorithm. *J. Global Optimization*, 19: 247-264.

Casado, L.G., Garcia, I., and Csendes, T. (2001b). A heuristic rejection criterion in interval methods for global optimization. *BIT* , 41: 683-692.

Ceberio, M., and Granvilliers, L. (2002). Solving nonlinear equations by abstraction, gaussian elimination, and interval methods. *Proceedings of Frontiers of Combining Systems*, Fourth International Workshop, FroCoS 2002, Alessandro Armando ed. LNAI, Springer, Italy, 2309: 117-131.

Cetin, B.C., Barben, J., and Burdick, J. W. (1993). Terminal repeller unconstrained subenergy tunneling (TRUST) for fast global optimization. *Journal of Optimization Theory and Applications*, 77: 97-126.

Chabert, G., Trombettoni, G., and Neveu, B. (2005). Box-set consistency for interval based constraint problems. *Symposium on Applied Computing, SAC 2005, Proc. Of the 2005 ACM symposium on Applied Computing*, pp. 1439-1443, ACM Press, NY, USA.

Chazan, D., and Miranker, W.L. (1970). A nongradient and parallel algorithm for unconstrained minimization. *SIAM Journal on control*, 8: 207-217.

Cleary, J.G. (1987). Logical arithmetic. *Future Computing Systems*, 2: 125-149.

Coconut: http://www.mat.univie.ac.at/~neum/glopt/coconut/Benchmark/Benchmark.html.

Conn, A. R., Gould, N., and Toint, Ph.L. (1994) A note on exploiting structure when using slack variables. *Math. Prog.*, 67: 89–99.

Corana, A., Marchesi, M., Marini, C., and Ridella, S. (1987). Minimizing multimodal functions of continuous variables with the simulated annealing algorithm. *ACM Trans. of Mathl. Software,* 13: 262-279.

Csallner, A.E., and Csendes, T. (1996). On the convergence speed of interval methods for global optimization. *Computers, Mathematics and Applications*, 31: 173-178.

Csallner, A.E., Csendes, T., and Markót, M.C. (2000a). Multi-section in interval branch and bound methods for global optimization I. Numerical tests. *J. Global Optimization*, 16: 219-228.

Csallner, A.E., Csendes, T., and Markót, M.C. (2000b). Multi-section in interval branch and bound methods for global optimization II. Theoretical results. *J. Global Optimization*, 16: 371-392.

Csendes, T., and Pinter, J. (1993). The impact of accelerating tools on the interval subdivision algorithm for global optimization. *European Journal of Operational Research*, 65: 314-320.

Csendes, T., and Ratz, D. (1996). A review of subdivision selection in interval methods for global optimization. *ZAMM* , 76: 319-322.

Csendes, T., and Ratz, D. (1997). Subdivision direction selection in interval methods for global optimization. *SIAM Journal of Numerical Anal*., 34: 922-938.

Csendes, T., Klatte, R., and Ratz, D. (2000). A posteriori direction selection rules for interval optimization methods. *CEJOR*, 8: 225-236.

Csendes, T. (2001). New subinterval selection criteria for interval global optimization. *Journal of Global Optimization*, 19: 307-327.

CUTEr: A constrained and unconstrained testing environment, revisited. http://cuter.rl.ac.uk/cuter-www/problems.html

Dallwig, S., Neumaier, A., and Schichl, H. (1997). GLOPT - A program for constrained global optimization. In Bomze, I. M., Csendes, T., Horst, R., and Pardalos, P. M., editors, *Developments in Global Optimization*, pp. 19-36, Kluwer, Dordrecht.

Dantzig, G. B. (1963). *Linear programming and extensions.* Princeton University Press, Princeton, New Jersey.

De Jong, K. (1975). *An analysis of the behaviour of a class of genetic adaptive systems*. PhD Thesis, University of Michigan, Ann Arbor.

Dekkers, A. and Aarts, E. (1991). Global optimization and simualted annealing. *Mathematical Programming*, 50: 367-393.

Dennis, J.E., and Schnabel, R.B. (1983). *Numerical methods for unconstrained optimization and nonlinear equations.* Prentice-Hall, Englewood Cliffs, New Jersey.

Dennis, J.E., and Torczon, V. (1991). Direct search methods on parallel computers. *SIAM Journal of Optimization*, 1: 448-474.

Didrit, O., Petitot, M., and Walter, E. (1998). Guaranteed solution of direct kinematic problems for general configurations of parallel manipulator. *IEEE Trans. On Robotics and Automation*, 14: 259-266.

Diener, I., and Schaback, R. (1990). An extended continuous newton method. *Journal of Optimization Theory and Applications*, 67: 57-77.

Dietmaier, P. (1998). The Stewart–Gough platform of general geometry can have 40 real postures, *Proceedings of ARK*, Strobl, Austria, pp. 7–16.

Dijkstra, E.W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1: 269-271.

Dillenburg, J.F., and Nelson, P.C. (1993). Improving the efficiency of depth-first search by cycle elimination. *Information Processing Letters*, 45: 5-10.

Dillenburg, J.F., and Nelson, P.C. (1994). Perimeter search. *Artificial Intelligence*, 65: 165-178.

Doran, J.E., and Michie, D. (1966). Experiments with the graph traverser program. *Proceedings of the Royal Society A*, 294: 235-259.

Drud, A.S. (1996). *Conopt: A system for large scale nonlinear optimization, reference manual for conopt subroutine library*, 69p, ARKI Consulting and Development A/S, Bagsvaerd, Denmark.

Epperly, T.G. (1995). *Global optimization of nonconvex nonlinear programs using parallel branch and bound.* Ph.D dissertation, University of Wisconsin-Madison, USA.

Evtushenko, Y.G., Potapov, M.A., and Korotkich, V.V. (1992). Numerical methods for global optimization. In Floudas, C. A., and Pardalos, P. M. editors, *Recent advances in Global Optimization*. pp. 274-297. Princeton University Press.

Faugere, J.C., and Lazard, D. (1995). The combinatorial classes of parallel manipulators, *Mechanism and Machine Theory*, 30: 765-776.

Fernández, J., and Pelegrin, B. (2001). Using interval analysis for solving planar single facility location problems: new discarding tests, *Journal of Global Optimization*, 19: 61-81.

Floudas, C.A., and Pardalos, P.M. (1992). *Recent advances in global optimization.* Princeton University, USA.

Floudas, C.A., and Visweswaran, V. (1993). A primal-relaxed dual global optimization approach. *J. Opt. Th. Appl.,* 78:187.

Floudas, C.A., and Pardalos, P.M. (1996). *State of the art in global optimization : computational methods and applications*. Kluwer Academic Publishers.

Forsgren, A., Gill, P.E., and Wright M.H. (2002). Interior methods for nonlinear optimization. *SIAM Review*, 44: 525-597.

Freuder, E.C. (1978). Synthesizing constraint expressions. *Communications of the ACM*, 21: 958-966.

Gablonsky, J.M. (2001). *DIRECT version 2.0 userguide.* Technical Report, CRSC-TR01-08,Center for Research in Scientific Computation, North Carolina State University, USA.

Gablonsky, J.M., and Kelley, C.T. (2001). A locally-biased form of the DIRECT algorithm. *Journal of Global Optimization*, 21: 27-37.

Ge, R.P., and Qin, Y.F. (1987). A class of filled functions for finding global minimizers of a function of several variables. *Journal of Optimization Theory and Applications*, 54: 241-252.

Gill, P.E., Murray, W., and Saunders, M.A. (1997). *Snopt: An SQP algorithm for large-scale constrained optimization*, Numerical Analysis Report 97-2, Department of Mathematics, University of California, San Diego, La Jolla, CA.

Glover, F. (1980). Tabu Search – Part I. *ORSA J. Computing*, 1: 190-206.

Goldberg, D.E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley.

Gourdin, E., Hansen, P., and Jaumard, B. (1994). *Global optimization of multi-variate Lipschitz functions: a survey and computational comparison*. Les Cahiers du GERAD, McGill University, Montreal.

Granvilliers, L. (1998). *Consistances locales et transformations symboliques de contraintes d'intervalles*. Phd Thesis, Laboratoire d'Informatique Fondamentale d'Orléans, Université d'Orléans.

Granvilliers, L. (2001). On the combination of interval solvers. *Reliable Computing*, 7: 467-483.

Granvilliers, L., Monfroy, E., and Benhamou, F. (2001). Symbolic-interval cooperation in constraint programming. *International Conference on Symbolic and Algebraic Computation, Proceedings of the 2001 international symposium on Symbolic and algebraic computation*, pp. 150-166. Ontario, Canada.

Granvilliers, L. (2004). An interval component for continuous constraints. *Journal of Computational and Applied Mathematics*, 162: 79–92.

Granvilliers, L., and Benhamou, F. (2006). RealPaver: An interval solver using constraint satisfaction techniques. *ACM Transactions on Mathematical Software*, 32 : 138-156.

Grossmann, I.E. (1996). *Global optimization in engineering design.* Kluwer Academic Publishers.

Hammer, R., Hocks, M., Kulish, U., and Ratz, D. (1993). *Numerical toolbox for verified computing I.* Springer-Verlag, Berlin.

Hansen, E. (1968). On solving systems of equations using interval analysis. *Math. Comput.*, 22: 374-384.

Hansen, E., and Sengupta, S. (1980). Global constrained optimization using interval analysis. In Nickel, K. L., *Interval Mathematics*, Academic Press, New York, pp. 25-47.

Hansen, E. (1992). *Global optimization using interval analysis.* Marcel Dekker Inc.

Hansen, P., Jaumard, B., and Lu, S. (1992). Global optimization of univariate Lipschitz functions: I. Survey and Properties. *Math. Prog.*, 55: 251.

Hansen, E., and Walster, G.W. (1993). Bounds for lagrange multipliers and optimal points. *Comput. Math. Appl.*, 25: 59.

Hansen, P., and Jaumard, B. (1995). Lipschitz optimization. In Horst, R., and Pardalos, P. M., editors, *Handbook of Global Optimization*, pp. 407-493, Kluwer Academic Publishers, Dordrecht.

Hart, P.E., Nilsson, N.J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4: 100-107.

He, L., and Polak, E. (1993). Multistart method with estimation scheme for global satisfying problems. *Journal of Global Optimization*, 3: 139-156.

He, J., Watson, L. T., Ramakrishnan, N., Schaffer, C.A., Verstak, A., Jiang, J., Bae, K., and Tranter, W. (2002). Dynamic data structures for a direct search algorithms. *Computational Optimization and Applications*, 23: 5-25.

Hestenes, M. R. (1980). *Conjugate direction methods in global optimization.* Springer-verlag.

Horst, R., Thoai, N.V., and De Vries, J. (1992). A new simplicial cover technique in constrained global optimization. *J. Global Opt.*, 2: 1-19.

Horst, R., Pardalos, P.M. (1995). *Handbook of global optimization.* Kluwer Academic Publishers, Dordrecht, The Netherlands.

Horst, R., Pardalos, P.M., and Thoai, N.V. (2000). *Introduction to global optimization.* Kluwer Academic Publishers, Dordrecht, The Netherlands.

Horst, R., and Tuy, H. (2003). *Global optimization: Deterministic approaches.* Springer-Verlag, Berlin.

Hsu, Y-L., Wang, S-G., and Yu, C-C. (2003). A sequential approximation method using neural networks for engineering design optimization problems. *Engineering Optimization*, 35: 489-511.

Husty, M. L. (1996). An algorithm for solving the direct kinematics of Stewart Gough platform. *Mechanism and Machine Theory*, 31: 365-380.

Huyer, W., and Neumaier, A. (1999). Global optimization by multilevel coordinate search. *Journal of Global Optimization*, 14: 331-355.

Hyvönen, E. (2001). Interval input and output. *Scientific computing, validated numerics, interval methods*. In Krämer and Gudenberg, W. V., editors, pp. 41-51. Kluwer Academic Publishers, New York.

Ilog. (2001). *Ilog optimization suite*. white paper. Available via http://www.ilog.fr.

Ingber, L. (1994). Simulated annealing: Practice versus theory. *J. Mathl. Comput. Modelling*, 18: 29-57.

Ingber, L. (1996). Adaptive simulated annealing (ASA): Lessons learned. *Control and Cybernetics*, 25: 33-54.

Innocenti, C. (2001). Forward kinematics in polynomial form of the general Stewart Gough platform. *ASME J. of Mechanical Design*, 123: 254-260.

Jansson, C., and Knüppel, O. (1992). *A global minimization method: The multi-dimensional case*. Technical Report 92.1, Forschungsschwerpunkt Informations- und Kommunikationstechnik, TU Hamburg-Harburg.

Jansson, C. (1994). On self validating methods for optimization methods. In Herzberger, J., editor, *Topics in Validated Computations — Studies in Computational Mathematics*, pp. 381-438, North-Holland, Amsterdam.

Jaulin, L., Kieffer, M., Didrit, O., and Walter, E. (2001). *Applied interval analysis*. Springer-Verlag, Berlin.

Joines, J. and Houck, C. (1994). On the use of non-stationary penalty functions to solve non-linear constrained optimization problems with GAs. *Proceedings of the First IEEE International Conference on Evolutionary Computation*, *IEEE Press*, 579-584.

Jones, D.R., Perttunen, C.D., and Struckman, B.E. (1993). Lipschitzian optimization without Lipschitz constant. *Journal of Optimization Theory and Applications*, 79: 157-181.

Kahan, W.M. (1968). A more complete interval arithmetic. *Lecture notes for summer course at the University of Michigan*.

Kannan, B.K., and Kramer, S.N. (1994). An augmented Lagrange multiplier based method for mixed integer discrete continuous optimization and its applications to mechanical design. *Journal of Mechanical Design*, 116: 405-411.

Kearfott, R.B. (1979). An efficient degree-computation method for a generalized method of bisection. *Numerical Mathematics*, 32: 109-127, 1979.

Kearfott, R.B. (1990). Interval newton / generalized bisection when there are singularities near roots. *Annals of Operations Research*, 25: 181-196.

Kearfott R.B., and Manuel, N.III. (1990). INTBIS: A portable interval newton/bisection package. *ACM Trans. on Mathematical Software*, 16: 152-157, 1990.

Kearfott, R.B. (1991). Decompostion of arithmetic expressions to improve the behaviour of interval iteration for nonlinear systems. *Computing*, 47: 169-191, 1991.

Kearfott, R.B. (1992). An interval branch and bound algorithm for bound constrained optimization problems. *Journal of Global Optimization*, 2: 259–280.

Kearfott, R.B., and Xing, Z. (1994). An interval step control for continuation methods. *SIAM J. Num. Anal.*, 31: 892-914, 1994.

Kearfott, R.B., and Kreinovich, V. (1996). *Applications of interval computations.* Applied Optimization, Kluwer, Dordrecht, Netherlands.

Kearfott, R.B. (1996a). A review of techniques in the verified solution of constrained global optimization problems. In Kearfott, R. B. and Kreinovich, V. editors, *Applications of Interval Computations*, Kluwer, Dordrecht, Netherlands, pp. 23-60.

Kearfott, R.B. (1996b). Test results for an interval branch and bound algorithm for equality-constrained optimization, In Floudas, C., and Pardalos, P., editors, *State of the Art in Global Optimization: Computational Methods and Applications*, Kluwer, Dordrecht, pp. 181-200.

Kearfott, R.B. (1996c). *Rigorous global search: Continuous problems*, Kluwer Academic Publishers, Netherlands.

Kearfott, R.B. (1996d). *On verifying feasibility in equality constrained optimization problems.* preprint. http://interval.louisiana.edu/preprints/big_constrai.pdf

Kearfott, R.B. (1997). Empirical evaluation of innovations in interval branch and bound algorithms for nonlinear algebraic systems. *SIAM Journal on Scientific Computing*, 18: 574-594.

Kearfott, R.B., and Walster, G.W. (2000). On stopping criteria in verified non-linear systems or optimization algorithms. *ACM TOMS*, 26: 373–389.

Kearfott, R.B., Nakao, M.T., Neumaier, A., Rump, S.M., Shary, S.P., and van Hentenryck, P. (2002). *Standardized notation in interval analysis.* preprint. http://www.mat.univie.ac.at/~neum/software/int/notation.ps.gz

Kearfott, R.B. (2003). *An overview of the GlobSol package for verified global optimization.* Talk given for the Department of Computing and Software, McMaster University.

Kearfott, R.B. (2005). Improved and simplified validation of feasible points: Inequality and equality constrained problems. *submitted to Mathematical Programming*. http://interval.louisiana.edu/preprints/2005_simplified_feasible_point_verification.ps

Kearfott, R.B. (2006). Discussion and empirical comparisons of linear relaxations and alternate techniques in validated deterministic global optimization. *Optimization Methods and Software,* 21:715-731.

Kinsella, J.A. (1992). Comparison and evaluation of variants of the conjugate gradient methods for efficient learning in feed-forward neural networks with backward propagation. *Neural Networks*, 3: 27-35.

Kirkpatrick, A., Gelatt Jr., C.D., and Vechi, M.P. (1983). Optimization by simulated annealing. *Science*, 220: 671-680.

Klatte, R., Kulisch, U., Wiethoff, A., Lawo, C., and Rauch, M. (1993). *C-XSC - A C++ Class Library for Extended Scientific Computing.* Springer-Verlag, Heidelberg-New York.

Knuppel, O. (1994). PROFIL/BIAS – A fast interval library. *Computing*, 53: 277-287.

Körf, R.E. (1985) Depth iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27: 97-109.

Körf, R.E. (1998). *Artificial intelligence search algorithms*. Algorithms and Theory of Computation Handbook, CRC Press, 1998.

Lawrence, C.T., Zhou, J.L., and Tits, A.L. (1997). *User's guide for CFSQP version 2.5: A C code for solving (Large scale) constrained nonlinear (Minimax) optimization*

problems, generating iterates satisfying all inequality constraints. Institute for Systems Research, University of Maryland, Technical Report TR-94-16r1, College Park, MD 20742.

Lawrence, C.T., and Tits, A.L. (2001). A computationally efficient feasible sequential quadratic programming algorithm. *SIAM J. Optimization*, 11: 1092-1118.

Lebbah, Y., and Lhomme, O. (2002). Accelerating filtering techniques for numeric CSPs. *Artificial Intelligence*, 139: 109-132.

Lebbah, Y., Michel, C., and Rueher, M. (2003). Global filtering algorithms based on linear relaxations. *2nd International Workshop on Global Constrained Optimization and Constraint Satisfaction (Cocos'03)*, Lausanne, Switzerland, pp.18-21.

Lebbah, Y. *ICOS (Interval constraints solver)*, WWW-document (2003). http://www-sop.inria.fr/coprin/ylebbah/icos.

Lee, T.-Y., and Shim, J.-K. (2001). Elimination based solution method for the forward kinematics of the general Stewart Gough platform. *Proceedings of the 2nd workshop on computational kinematics*, Seoul, Korea, pp. 259-267.

Le Grand, S.M., and Merz Jr., K.M. (1993). The application of the genetic algorithm to the minimization of potential energy functions. *Journal of Global Optimization*, 3: 49-66.

Levy, A.V., Montalvo, A., Gomez, S., and Calderon, A. (1981). Topics in global optimisation. *Lecture Notes in Mathematics*, 909: 18-33.

Lewis, R.M., Torczon, V., and Trosset, M.W. (2000). Direct search methods: Then and now. *Journal of Computational and Applied Mathematics*,124: 191-207.

Leyffer, S., Lopez-Calva, G., and Nocedal, J. (2004). *Interior methods for mathematical programs with complementarily constraints*. ANL/MCS-P1211-1204, Argonne National Laboratory, Mathematics and Computer Science Division.

Leyffer, S. (2005). The penalty interior point method fails to converge. *Optimization Methods and Software*. 20: 559-568.

Lhomme, O. (1993). Consistency techniques for numeric CSPs'. In Bajcsy, R., editor, *Proceedings of the 13th IJCAI*, pp. 232-238, IEEE Computer Society Press.

Lhomme, O., Gotlieb, A., and Rueher, M. (1998). Dynamic optimization of interval narrowing algorithms. *Journal of Logic Programming*, 37: 165-183.

Liu, M.D., and Tits, A.L. (1997). *User's guide for ADIFFSQP version0.9: A utility program that allows the user of the FFSQP constrained nonlinear optimization routines to conveniently invoke the computational differentiation preprocessor ADIFOR2.0.* Institute for Systems Research, University of Maryland, College Park, MD 20742.

Lüthi, J., and Lladó, C.M. (2003). Splitting techniques for interval parameters and their application to performance models. *An International Journal for Performance Evaluation*, 51: 47-74.

Manocha, D., and Canny, J.F. (1994). Efficient inverse kinematics for general 6R manipulators. *IEEE Journal on Robotics and Automation*,10: 648-657.

Markót, M.C. (2003). *Reliable global optimization methods for constrained problems and their application for solving circle packing problems.* PhD dissertation. University of Szeged, Hungary.

Markót, M.C., Fernández, J., Casado, L.G., and Csendes, T. (2006). New interval methods for constrained global optimization. *Mathematical Programming*, 106: 287-318.

Mayne, D.Q., and Polak, E. (1976). Feasible direction algorithms for optimization problems with equality and inequality constraints. *Mathematical Programming*, 11: 67-80.

Meewella, C.C., and Mayne, D.Q. (1989). An algorithm for global optimization of Lipschitz functions. *Journal of Optimization Theory and Applications*, 61: 247-270.

Meier, M., and Schimpf, J. (1993). *ECLiPSe user manual*. Tech. Rep. ECRC-93-6, ECRC (European Computer-industry Research Centre), Munich, Germany.

Merlet, J-P. (1998). Determination of the presence of singularities in 6D workspace of a Gough parallel manipulator. In *ARK*, pp. 39-48, Strobl, 29 Juin-4 Juillet.

Merlet, J.-P. (2000). *ALIAS: An interval analysis based library for solving and analyzing system of equations*. Séminaire Systèmes et équations algébriques, Toulouse, Automation pp. 1964–1969.

Merlet, J.-P. (2001). An improved design algorithm based on interval analysis for parallel manipulator with specified workspace. *Proc. of the IEEE Int. Conf. on Robotics and Automation*, Seoul, South Korea, pp. 1289–1294, 2001.

Merlet, J.-P. (2004). Solving the forward kinematics of a Gough type parallel manipulator with interval analysis. *The Int. J. of Robotics Research*, 23: 221-235.

Michalewicz, Z. (1994). *Genetic Algorithms+Data Structures=Evolution programs.* Springer Verlag, Berlin.

Michalewicz, Z. and Attia, N. (1994). Evolutionary optimization of constrained problems. *Proceedings of the Third Annual Conference on Evolutionary Programming*, 98-108.

Miranda, C. (1940). Un' osservazione su un teorema di Brouwer. *Bol. Un. Mat. Ital*., 2: 5-7.

Mockus, J. (1989). *Bayesian approach to global optimization.* Kluwer Academic Publishers, Dordrecht-London-Boston.

Mockus, J. (1994). Application of bayesian approach to numerical methods of global and stochastic optimization. *Journal of Global Optimizatin*, 4: 347-365.

Moore, R. E. (1966). *Interval analysis*. Prentice-Hall, Englewood Cliffs, NJ.

Moore, R., Hansen, E., and Leclerc, A. (1992). Rigorous methods in global optimization. In Floudas, C. A., and Pardalos, P. M., editors, *Recent Advances in Global Optimization*, pp. 321-342, Princeton University Press.

Morales, K. A. and Quezada, C.C. (1998). A universal eclectic genetic algorithm for constrained optimization. *Proceedings 6th European Congress on Intelligent Techniques & Soft Computing,* 518-522.

Morales, J.L., Nocedal, J., Waltz, R., Liu, G., and Goux, J.P. (2001). *Assessing the potential of interior methods for nonlinear optimization.* Report OTC 2001/6, Optimization Technology Center.

Moré, J.J., Garbow, B.S., and Hillstrom, K.E. (1981). Testing unconstrained optimization software. *ACM Trans. Mathematical Software*, 7: 17-41.

Mozart Consortium. (1999). *The Mozart programming system.* Available from http://www.mozart-oz.org.

Murray, W. (1997). Sequential quadratic programming methods for large-scale problems. computational optimization and applications, 7: 127-142.

Murtagh, B.A., and Saunders, M.A. (1978). Large-scale linearly constrained optimization. *Mathematical Programming*,14: 41–72.

Nataraj, P.S.V., and Kotecha, K. (2002). An algorithm for global optimization using the Taylor Bersten form as inclusion function. *Journal of Global Optimization*, 24: 417-436.

Nelder, J.A., and Mead, R. (1965). A simplex method for function minimization. *Computer Journal*, 7: 308-313.

Neumaier, A. (1982). Overestimation in linear interval equations. *SIAM Journal of Numerical Anals*, 24: 207-214.

Neumaier, A. (1990). *Interval methods for systems of equations encyclopedia of mathematics and its applications 37*, Cambridge University Press, Cambridge.

Neumaier, A. (2004). Complete search in continuous global optimization and constraint satisfaction. In Iserles, A., editor, *Acta Numerica 2004*, Cambridge University Press,.

Neumaier, A., Shcherbina, O., Huyer W. and Vinko, T. (2005). A comparison of complete global optimization solvers. *Math. Programming*, 103: 335-356.

Nielsen, J., and Roth, B. (1997). Formulation and solution for the direct and inverse kinematics problems for mechanisms and mechatronic systems. In: Angeles, J., and Zakhariev, E., editors, *Proceedings of the NATO Advanced Study Institute on Computational Methods in Mechanisms*, pp. 233–252.

Özdamar, L., and Demirhan, M.B. (2000). Experiments with new stochastic global optimization search techniques. *Computers and OR*, 27: 841-865.

Özdamar, L., and Demirhan, M.B. (2001). Comparison of partition evaluation measures in an adaptive partitioning algorithm for global optimization. *Fuzzy Sets and Systems*, 117: 47-60.

Panier, E.R., Tits, A.L., and Herskovits, J.N. (1988). A QP-free, globally convergent, locally superlinearly convergent algorithm for inequality constrained optimization. *SIAM J. on Control and Optimization*, 26: 788—811.

Pardalos, P.M., and Rosen, J.B. (1987). Constrained global optimization: Algorithms and applications. *Lecture Notes in Computer Science*, 268: 143-143.

Pardalos, P.M. (1993). *Complexity in numerical optimization.* World Scientific, Singapore and River Edge, New Jersey.

Pardalos, P.M., and Romeijn, H.E. (2002). *Handbook of global optimization Volume 2 (Nonconvex optimization and its applications).* Springer, Boston/Dordrecht/London.

Petrov, E., and Benhamou, F. (2002). Improved interval constraint propagation for constraints on partial derivatives. *Lecture Notes in Computer Science,* 2330: 1097-1105.

Pinter, J.D. (1986) Extended univariate algorithm for *n*-dimensional global optimization. *Computing*, 36: 91-103.

Pinter, J.D. (1988). Branch and bound algorithms for solving global optimization problems with Lipschitzian structure. *Optimization*, 19:101-110.

Pinter, J.D. (1992). Convergence qualification of adaptive partition algorithms in global optimization. *Mathematical programming,* 56: 343-360.

Pinter, J.D. (1996). *Global optimization in action*. Kluwer Academic Publishers, Dordrecht.

Pinter, J.D. (1997). LGO- A program system for continuous and Lipschitz global optimization. In Bomze, I. M., Csendes, T., Horst, R., and Pardalos, P. M., editors, *Developments in Global Optimization*, pp.183-197.Kluwer Academic Publishers, Dordrecht / Boston / London.

Pohl, I. (1971). Bi-directional search. In Meltzer, B., and Michie, D., editors, *Machine Intelligence*, pp. 127-140, American Elsevier, New York.

Popova, E.D. (1996). Interval operations involving NaNs. *Reliable computing*, 2: 161-166.

Popova, E.D. (2001). Multiplication distributivity of proper and improved intervals. *Reliable Computing*, 7: 129 – 140.

Powell, M.J.D. (1964). An efficient method for finding the minimum of a function of several variables without calculating the derivatives. *Computer Journal*, 7: 155-162.

PrincetonLib, *Princeton library of nonlinear programming models.* http://www.gamsworld.org/performance/ princetonlib/princetonlib.htm.

PrologIA. (1994). *Prolog IV: Reference manual and user's guide*, Tech. Report.

Puget, J.F., and Van Hentenryck, P. (1998). A constraint satisfaction approach to a circuit design problem. *Journal of Global Optimization*, 13: 75-93.

Rao, R.S., Asaithambi, A., and Agrawal, S.K. (1998). Inverse kinematic solution of robot manipulators using interval analysis. *ASME Journal of Mechanical Design,* 120: 147–150.

Ratschek, H., and Rokne, J. (1988). *New computer methods for global optimization.* Ellis Horwood, Chichester.

Ratschek, H., and Rokne, J. (1995). Interval methods. In Horst, R., and Pardalos, P. M., editors, *Handbook of Global Optimziation*, pp. 751-828, Kluwer Academic publisher, Netherlands.

Ratz, D. (1992). An inclusion algorithm for global optimization in a portable PASCAL-XSC implementation. In Atanassova, L., and Herzberger, J., editors, *Computer Arithmetic and Enclosure Methods*, pp. 329-338, North-Holland, Elsevier, Amsterdam.

Ratz, D. (1994). Box splitting strategies for the interval Gauss-Seidel step in a global optimization method. *Computing*, 53: 337-353.

Ratz, D., and Csendes, T. (1995). On the selection of subdivision directions in interval branch-and-bound methods for global optimization. *Journal of Global Optimization*, 7:183-207.

Ratz, D. (1996). On the branching rules in second order branch and bound methods in global optimization. In Alefeld, G., Frommer, A., and Kang, B., editors, *Scientific Computing and Validated Numerics,* pp. 221-227, Akademie-Verlag, Berlin.

Ratz, D. (1997). New results on gap-treating techniques in extended interval newton Gauss Seidel steps in global optimization. *Developments in Global Optimization*, pp. 55-72. Kluwer Academic Publishers.

Recio, T., and Gonzalex-Lopez, M.J. (1994). On the symbolic insimplification of the general 6R-manipulator kinematic equations, In *Proceedings of the international symposium on Symbolic and algebraic computation*, pp. 354 – 358.

Robinson, S.M. (1973). Computable error bounds for nonlinear programming. *Math. Programming*, 5:235-242.

Romeijn, H.E., and Smith, R.L. (1994). Simulated annealing for constrained global optimization. *Journal of Global Optimization*, 5:101-126.

Rosenbrock, H.H. (1970), *State space and multivariable theory*. Wiley Interscience Division, New York.

Rump, S.M. (1992). On the solution of interval linear systems. *Computing*, 47: 337-353.

Ryoo, H.S., and Sahinidis, N.V. (1995). Global optimization of nonconvex NLPs and MINLPs with applications in process design. *Computers & Chemical Engineering*, 19: 551-566.

Ryoo, H.S., and Sahinidis, N.V. (1996). A branch-and-reduce approach to global optimization. *Journal of Global Optimization*, 8: 107-139.

Sahinidis, N.V. (1996). *Baron*: A general purpose global optimization software package. *Journal of Global Optimization*, 8: 201-205.

Sahinidis, N.V. (2003). Global optimization and constraint satisfaction: The branch-and-reduce approach. In Bliek, C., Jermann, C., and Neumaier, A., editors, *COCOS 2002*, LNCS 2861, pp. 1–16.

Sam-Haroud, D., and Faltings, B. (1996). Consistency techniques for continuous constraints. *Constraints*, 1: 85–118.

Sandgren, E. (1988). Nonlinear integer and discrete programming in mechanical design. *Proceeding of the ASME Design Technology Conference*, Kissimmee, FL, pp. 95-105.

Sarma, M.S. (1990). On the convergence of the Baba and Dorea random optimization methods. *Journal of Optimization Theory and Applications*, 66: 337-343.

Schaffler, S., and Warsitz, H. (1990). A trajectory-following method for unconstrained optimization. *Journal of Optimization Theory and Applications*, 67:133-140.

Scherbina, O., Neumaier, A., Sam-Haroud, D., Vu, X.-H., and Nguyen, T.-V. (2002). Benchmarking global optimization and constraint satisfaction codes. *Global Optimization and Constraint Satisfaction: First International Workshop on Global Constraint Optimization and Constraint Satisfaction*, *COCOS 2002*, Valbonne-Sophia Antipolis, France.

Schichl, H. (2003). *Mathematical modeling and global optimization.* Ph.D dissertation, University of Vienna, Austria.

Schichl, H. (2004). Global optimization in the COCONUT project. In *Proceedings of the Dagstuhl Seminar Numerical Software with Result Verification*, Springer Lecture Notes in Computer Science 2991, Springer, Berlin.

Schichl, H., and Neumaier, A. (2004). Exclusion regions for systems of equations. *SIAM J. Numer. Anal.*, 42: 383-408.

Schichl, H., and Neumaier, A. (2005). Interval analysis on directed acyclic graphs for global optimization. *Journal of Global Optimization*, 33: 541-562.

Schittkowski, K. (1987). *More test examples for nonlinear programming codes.* Lecture Notes in Economics and Mathematical Systems 282, Springer-Verlag, Berlin.

Schoen, F. (1991). Stochastic techniques for global optimization: A survey on recent advances. *Journal of Global Optimization*, 1: 207-228.

Schwartz, D.I. (1999). *Deterministic interval uncertainty methods for structural analysis.* Ph.D Dissertation. State University of New York at Buffalo.

Schwefel, H.P. (1981). *Numerical optimization of computer models.* Wiley & Sons, Chichester.

Shang, Y. (1997). *Global search methods for solving nonlinear optimization problems*. Ph.d dissertation, University of Illinios, Urbana-Champaign, 1997.

Shanno, D.F., and Phua, K.H. (1989). Numerical experience with sequential quadratic programming algorithms for equality constrained nonlinear programming. *ACM Transactions on Mathematical Software*, 15: 49-63.

Sherali, H. D., and Tuncbilek, C. H. (1992). A global optimization algorithm for polynomial programming problems using a reformulation-linearization technique, *Journal of Global Optimization*, 2:101-112.

Sherali, H. D., and Wang, H. (2001). Global optimization of nonconvex factorable programming problems, *Mathematical Programming*, 89:459-478.

Skelboe, S. (1974). Computation of rational interval functions. *BIT*, 14: 87-95.

Smolka, G. (1995). The OZ programming model. In van Leeuwen, J., editor, *Lecture Notes in Computer Science*, 1000: 324-343. Springer-Verlag.

Smith, E.M.B. (1996). *On the optimal design of continuous processes*. Ph.D thesis, Imperial College, London.

Smith, E.M.B., and Pantelides, C.C. (1996). Global optimization of general process models. In Grossmann, I. E., editor, *Global Optimization in Engineering design*, Kluwer Publishers, pp. 355-386.

Smith, E.M.B., and Pantelides, C.C. (1999). A symbolic reformulation/spatial branch and bound algorithm for the global optimization of nonconvex MINLP's. *Computers and Chemical Eng.*, 23: 457-478.

Snyman, J.A., and Fatti, L.P. (1987). A multi-start global minimization algorithm with dynamic search trajectories. *Journal of Optimization Theory and Applications*, 54: 121-141.

Sommese, A.J., Verschelde, J., and Wampler, C.W. (2002). Advances in polynomial continuation for solving problems in kinematics. In *Proc. ASME Design Engineering Technical Conf. (CDROM)*, Montreal, Quebec.

Spaans, R., and Luus, R. (1992). Importance of search-domain reduction in random optimization. *Journal of Optimization Theory and Applications*, 75: 635-638.

Spellucci, P. (2002). *Solving QP problems by penalization and smoothing*. TUD Dept. of Math. preprint 2242. http://wwwbib.mathematik.tu-darmstadt.de/Math-Net/Preprints/Listen/files/2242.ps.gz

Stahl, V. (1995). *Interval methods for bounding the range of polynomials and solving systems of nonlinear equations*. Ph.D dissertation. Universitat Linz.

Stahl, V. (1997). A sufficient condition for non-overestimation in interval arithmetic. *Computing*, 59: 349-363.

Strongin, R.G. (1978). *Numerical methods for multiexternal Problems*. Nauka, Moscow.

Stickel, M.E., and Tyson, W.M. (1985). An analysis of consecutively bounded depth-first search with applications in automated deduction. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-85)*, Los Angeles, Ca., pp. 1073-1075,.

Sturua, E.G., and Zavriev, S.K. (1991). A trajectory algorithm based on the gradient method I. the search on the quasioptimal trajectories. *Journal of Global Optimization*, 1991: 375-388.

Taylor, L., and Kröf, R.E. (1993). Pruning duplicate nodes in depth-first search. *Proceedings of National Conference on Artificial Intelligence (AAAI-93)*, Washington D.C., pp. 756-761.

Tawarmalani, M., and Sahinidis, N.V. (2002). Convex extensions and envelopes of lower semi-continuous functions. *Mathematical Programming*, 93: 247-263.

Tawarmalani, M., and Sahinidis, N.V. (2004). Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. *Mathematical Programming*, 99: 563-591.

Törn, A. (1973). Global optimization as a combination of global and local search. *Gothenburg Business Adm. Studies*, 17:191-206.

Törn, A. (1977). Cluster analysis using seed points and density determined hyperspheres as an aid to global optimization. *IEEE Trans. Syst. Men and Cybernetics*, 7: 610-616.

Törn, A., and Zilinskas, A. (1989). Global optimization. *Lecture Notes in Computer Science*, No. 350, Springer-Verlag.

Törn, A., and Viitanen, S. (1992). Topographical global optimization. In Floudas, C. A., and Pardalos, P. M., editors, *Recent Advances in Global Optimization*, pp. 385-398, Princeton University Press.

Törn, A., Ali, M.M., and Viitanen, S. (1999). Stochastic global optimization: Problem classes and solution techniques. *Journal of Global Optimization*, 14: 437-447.

Tóth, B. (2002). Empirical investigation of the convergence speed of inclusion functions. *10th GAMM - IMACS International Symposium on Scientific Computing*, Computer Arithmetic, and Validated Numerics (SCAN2002), Paris, France.

Tung, J. (2001). *Interval analysis and its applications to optimization in behavioural ecology.* Independent Research Report, University of Cornell, USA.

Tsai, L.W., and Morgan, A.P. (1985). Solving the kinematics of the most general six and five-degree-of-freedom manipulators by continuation methods. *Journal of Mechanisms*, Transmissions, and Automation in Design 107 189-200.

Tuy, H., Thieu, T.V., and Thai. N.Q. (1985). A conical algorithm for globally minimizing a concave function over a closed convex set. *Mathematics of Operations Research*, 10: 498.

Vaidyanathan, R., and Halwagi, M.E.L. (1994). Global optimization of nonconvex nonlinear programs via interval analysis. *Computers Chemical Engineering*, 18: 889-897.

Vaidyanathan, R., and Halwagi, M.E.L. (1996). Global optimization of nonconvex MINLP by interval analysis. *Global Optimization in Engineering Design,* pp. 175-193, Kluwer Academic Publishers.

Van-Hentenryck, P., Michel, L., and Deville, Y. (1997a). *Numerica: A modeling language for global optimization.* MIT press, London, England.

Van-Hentenryck, P., Mc Allester, D., and Kapur, D. (1997b). Solving polynomial systems using branch and prune approach. *SIAM Journal on Numerical Analysis,* 34: 797-827.

Verdonk, B., Vervloet, J., and Cuyt, A. (2002). *Blending interval and set arithmetic for maximal reliability*. Technical report (http:// www.win.ua.ac.be /~verdonk/publications .html)

Vincent, T.L., Goh, B.S., and Teo, K.L. (1992). Trajectory-following algorithms for min-max optimization problems. *Journal of Optimization Theory and Applications*, 75: 501-519.

Vinkó, T., Lagouanelle, J.-L., and Csendes, T. (2002). Kite: A new inclusion function for optimization. *Validated Computing,* Extended Abstracts, pp. 179-181, Toronto, Canada.

Vu, X.-H., Sam-Haroud, D., and Silaghi, M.-C. (2003). Numerical constraint satisfaction problems with non-isolated solutions. *Global Optimization and Constraint Satisfaction: First International Workshop on Global Constraint Optimization and Constraint Satisfaction, COCOS 2002*, Revised Selected Papers, LNCS 2861, pp. 194-210, Springer-Verlag.

Vu, X.-H., Schichl, H., and Sam-Haroud, D. (2004). Using directed acyclic graphs to coordinate propagation and search for numerical constraint satisfaction problems. In *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004)*, pp. 72-81, Florida, USA.

Vu, X-H. (2005). *Rigorous solution techniques for numerical constraint satisfaction problems*. Ph.D dissertation, Swiss Federal Institute of Technology, Lausanne, Switzerland.

Wah, B. W., and Wang, T. (1999). Simulated annealing with asymptotic convergence for nonlinear constrained global optimization. *Lecture Notes In Computer Science,* 1713: 461-475.

Wampler, C., Morgan, A., and Sommese, A. (1990). Numerical continuation methods for solving polynomial systems arising in kinematics. *ASME Journal of Mechanical Design,* 112: 59–68.

Wampler, C., and Morgan, A. (1991). Solving the 6R inverse position problem using a generic-case solution methodology. *Mech. Mach. Theory*, 26: 91-106.

Watson, L.T., and Baker, C.A. (2000). *A fully distributed parallel global search algorithm.* Technical report TR-00-06, Dept. of Computer Science, Virginia Tech.

Wolfe, P. (1962). *The reduced-gradient method.* unpublished manuscript, RAND Corporation.

Wolfe, P. (1967). Methods of nonlinear programming. In Abadie, J., editor, *Nonlinear programming*, pp. 97-131, John Wiley, NewYork.

Wolfe, M.A. (1994). An interval algorithm for constrained global optimization. *J. Comput. Appl. Math.*, 50: 605–612.

Yamamura, K., Kawata, H. and Tokue, A. (1998). Interval solution of nonlinear equations using linear programming. *BIT*, 38: 186-199.

Yang, Y-F., Li, D-H., and Qi, L. (2003). A feasible sequential linear equation method for inequality constrained optimization. *SIAM J. Optim.*, 13: 1222-1244.

Yeniay, O. (2005). Penalty function methods for constrained optimization with genetic algorithms. *Mathematical and Computational Applications*, 10: 45-56.

Zabinsky, Z.B., and Smith, R.L. (1992). Pure adaptive search in global optimization. *Mathematical Programming*, 53:323-338.

Zacharias, C.R., Lemes, M.R., and Pino, A.D. (1998). Combining genetic algorithms and simulated annealing: A molecular geometry optimization study. *THEOCHEM - Journal of Molecular Structure*, 430: 29-39.

Zhou, J.L., and Tits, A.L. (1996). An SQP algorithm for finely discretized continuous minimax problems and other minimax problems with many objective functions. *SIAM J. on Optimization*,. 6: 461-487.

Zhou, J.L., Tits, A.L., and Lawrence, C.T. (1997). *User's guide for FFSQP version 3.7 : A fortran code for solving optimization programs, possibly minimax, with general inequality constraints and linear equality constraints, generating feasible iterates.* Institute for Systems Research, University of Maryland,Technical Report SRC-TR-92-107r5, College Park, MD 20742, 1997.

Zilinskas, A. (1992). A review of statistical models for global optimization. *Journal of Global Optimization*, 2: 145-153.

Appendix A

Detailed computational results on Continuous Constraint Satisfaction Problems

Name	Performance	Iterative Deepening			Worst First			Depth First			ALIAS	ICOS	Others	COCONUT
		SII Rule	Rule A	Smear	SII Rule	Rule A	Smear	SII Rule	Rule A	Smear	QUAD		First feasible Sol.	
Kin2	CPU (STU)	0.128	0.498	0.089	0.549	0.771	0.771	0.771	0.771	0.666	0.16	0.848	Baron (0.002)	0.0416
	No. of Stages	2	2	2								(=4.0)	LINGO 0.1	
	No. of SQP Calls	993	4583	445	730	643	720	8977	6660	1748				
	Avg No. Vars. in Parallel (max, min)	5.14 (6,4)	4.52 (6,2)	3.23 (4,3)	5.2(6, 5)	4.36(6, 2)	3.32(4, 3)	3(5, 3)	4.18(6, 2)	3.27(4, 3)				
	No. of Function Calls	11717	10728	6577	7298	6208	6369	1196312	1139747	23062				
Kin1 - Mod.	CPU (STU)	0.062	0.050	0.771	0.099	0.198	0.771	0.771	0.771	0.771	0.399	0.771	Baron-NA	NA
	No. of Stages	4	3	4								(=1.53)	LINGO 0.0	
	No. of SQP Calls	299	317	5746	254	327	863	5453	2709	5387				
	Avg No. Vars. in Parallel (max, min)	4.64 (6,3)	3.18 (6,2)	5.08 (6,4)	4.74 (6,3)	3.44(, 2)	5.23(6,4)	3.000(5, 3)	5.64(6, 2)	5.1(6, 4)				
	No. of Function Calls	17749	12266	121788	6907	4491	16270	402224	571317	74713				
KinCox	CPU (STU)	0.000	0.000	0.007	0.001	0.001	0.001	0.771	0.009	0.001	0.000	0.100	Baron 0.0004	0.0154
	No. of Stages	1	1	1									LINGO 0.30	
	No. of SQP Calls	13	3	11	9	11	11	27966	141	13				
	Avg No. Vars. in Parallel (max, min)	2 (2,2)	2 (2,2)	2 (2,2)	2 (2,2)	2 (2,2)	2 (2,2)	2(2, 2)	2(2, 2)	2(2,2)				
	No. of Function Calls	65	149	145	65	143	145	619599	3104	193				
Direct	CPU (STU)	0.009	0.038	0.771	0.021	0.504	0.110	0.771	0.771	0.068	0.771	NA	Baron-NA	NA
	No. of Stages	1	2	3							(=12.6)		LINGO 0.0	
	No. of SQP Calls	20	126	1305	20	380	116	2319	2493	66				
	Avg No. Vars. in Parallel (max, min)	2.5 (3,2)	2.79 (4,2)	3 (3,3)	2.39 (3,2)	2.78(3, 2)	3(3, 3)	2.7(3, 2)	3.09 (5, 2)	3(3, 3)				
	No. of Function Calls	265	2545	15665	265	7033	1409	87001	182916	881				

Name	Performance	Iterative Deepening			Worst First			Depth First			ALIAS QUAD	ICOS	Others First feasible Sol.	COCONUT
		SII Rule	Rule A	Smear	SII Rule	Rule A	Smear	SII Rule	Rule A	Smear				
	CPU (STU)	0.002	0.002	0.002	0.005	0.004	0.005	0.003	0.006	0.002	0.277	NA	Baron (0.0009)	NA
	No. of Stages	1	1	1									LINGO 0.0	
Stewart	No. of SQP Calls	13	13	12	13	13	12	10	21	8				
Gough	Avg No. Vars. in Parallel (max, min)	2 (2,2)	2 (2,2)	2 (2,2)	2 (2,2)	2 (2,2)	2 (2,2)	2(2, 2)	2.5(3, 2)	2(2, 2)				
	No. of Function Calls	145	145	145	145	145	145	145	217	145				
	CPU (STU)*	0.001	0.001	0.003	0.002	0.004	0.007	0.003	0.771	0.017	0.771	0.771	Baron (0.016)	0.0896
	No. of Stages	1	1	1							(>5.25)	(>170)	LINGO (infeas)	
FredTest	No. of SQP Calls	12	6	64	13	6	65	42	8817	166				
	Avg No. Vars. in Parallel (max, min)	4.8 (5,4)	6 (6,6)	6 (6,6)	4.8(5, 4)	6 (6,6)	6 (6,6)	4.85 (6, 3)	2.57(4, 2)	6(6, 6)				
	No. of Function Calls	1117	1763	952	1117	2170	1152	1066	469693	3662				
	CPU (STU)*	0.183	0.035	0.771	0.771	0.094	0.771	0.771	0.104	0.771	0.000	0.771	Baron 0.0002	0.771
	No. of Stages	2	2	2								(=2.3)	LINGO 0.1	2.392
ECO9	No. of SQP Calls	1454	237	7336	1178	242	1485	11193	564	10550				
	Avg No. Vars. in Parallel (max, min)	5.75 (8,4)	4 (6,2)	5.76 (8,5)	5.8(7,4)	4(6, 2)	5.8(8, 5)	4.26(8, 4)	4.22(6, 2)	6.02(8, 5)				
	No. of Function Calls	36125	2177	68464	18287	2177	13039	1751749	58572	107612				
	CPU (STU)	0.411	0.771	0.771	0.771	0.771	0.771	0.771	0.771	0.771	0.102	0.771	Baron (0.0002)	0.771
	No. of Stages	2	3	3								(=3.6)	LINGO 0.20	2.2146
Redeco8	No. of SQP Calls	2810	2892	5867	939	674	1275	4658	6682	7873				
	Avg No. Vars. in Parallel (max, min)	6.15 (8,3)	4.67(6,2)	7 (7,7)	5.23(8, 5)	4.12(6, 2)	7(7,7)	6.18(8, 5)	4.3(6, 2)	7(7, 7)				
	No. of Function Calls	148074	120542	168750	19536	5575	28446	3287011	1199768	217447				

Name	Performance	Iterative Deepening			Worst First			Depth First			ALIAS QUAD	ICOS	Others First feasible Sol.	COCONUT
		SII Rule	Rule A	Smear	SII Rule	Rule A	Smear	SII Rule	Rule A	Smear				
Dietmaier	CPU (STU)	0.771	0.771	0.418	0.771	0.771	0.591	0.771	0.771	0.519	0.039	NA	Baron (0.0009)	NA
	No. of Stages	3	2	2									LINGO (infeas)	
	No. of SQP Calls	4680	1182	1195	757	1429	414	2448	2178	553				
	Avg No. Vars. in Parallel (max, min)	4.31 (8,3)	4.4 (10,2)	4.81 (6,4)	4.23(6, 3)	4.23(10, 2)	2.0 (2,2)	4.51(6, 3)	2.5(10, 2)	4.94(6, 4)				
	No. of Function Calls	51568	25131	68226	12777	23935	5882	553308	791854	15549				
Heart	CPU (STU)	0.771	0.771	0.771	0.771	0.771	0.771	0.771	0.771	0.217	NA	0.771	Baron (0.03)	0.771
	No. of Stages	3	3	3								(>170)	LINGO(0.7)	3.285
	No. of SQP Calls	6211	4579	9189	785	1148	869	6237	8915	1089				
	Avg No. Vars. in Parallel (max, min)	3.46(4, 2)	3.7(6,2)	3.7 (4, 3)	3.23(4,3)	3.65(6, 2)	4(4, 4)	3.55(4, 3)	3.09(6, 2)	3.76(4, 3)				
	No. of Function Calls	75689	89441	80255	6973	16242	7105	938299	1352035	8865				
Neuro	CPU (STU)	0.031	0.006	0.064	0.034	0.002	0.518	0.047	0.004	0.114	NA	0.771	Baron (0.0035)	0.0722
	No. of Stages	1	1	2								(=4.40)	LINGO(0.1)	
	No. of SQP Calls	7	9	39	7	5	84	16	16	40				
	Avg No. Vars. in Parallel (max, min)	4.375(5, 3)	2.88(4,2)	4.15 (6, 4)	4.35(5, 3)	2.76(4, 2)	4.97(6, 4)	5(5, 5)	2.85(4, 2)	4.15(6, 4)				
	No. of Function Calls	1669	369	1845	1669	369	4109	1033	241	1845				
Butcher	CPU (STU)	0.098	0.030	0.192	0.035	0.018	0.771	0.771	0.771	0.056	NA	0.771	Baron (0.003)	0.771
	No. of Stages	2	1	2								(>170)	LINGO(Infeas)	3.0134
	No. of SQP Calls	108	33	896	49	13	878	4955	9385	150				
	Avg No. Vars. in Parallel (max, min)	2.78(3, 2)	3.33(5,2)	3(3, 3)	2.68(3, 2)	3.2(5, 2)	3(3, 3)	2.31(3,2)	4.22 (5, 2)	3(3, 3)				
	No. of Function Calls	953	887	8227	449	887	7547	178761	1890486	1279				

Name	Performance	Iterative Deepening			Worst First			Depth First			ALIAS	ICOS	Others	COCONUT
		SII Rule	Rule A	Smear	SII Rule	Rule A	Smear	SII Rule	Rule A	Smear	QUAD		First feasible Sol.	
PUMA	CPU (STU)*	0.003	0.008	0.771	0.009	0.015	0.771	0.020	0.019	0.771	0.001	0.059	Baron (0.0013)	0.0376
	No. of Stages	1	1	2									LINGO 0.0	
	No. of SQP Calls	28	75	1589	28	77	860	99	105	900				
	Avg No. Vars. in Parallel (max, min)	6.07 (7,6)	3.14 (6,2)	5.45 (7,2)	6.17(7, 6)	3.23(6, 2)	5.26(7, 2)	6.6(8, 6)	2.34(6, 2)	5.04(7, 2)				
	No. of Function Calls	2741	857	15109	2741	933	8297	2866	2350	8121				
Chemeq	CPU (STU)	0.159	0.771	0.006	0.771	0.771	0.008	0.771	0.771	0.009	NA	0.771	Baron (0.0004)	0.771
	No. of Stages	4	5	1								(=1.6)	LINGO 0.20	4.3266
	No. of SQP Calls	1368	5369	12	792	783	12	12179	26974	12				
	Avg No. Vars. in Parallel (max, min)	2.0 (2,2)	2.35 (3,2)	2.0 (2,2)	2.0 (2,2)	2.25(3, 2)	2.0 (2,2)	2 (2,2)	2.33(3, 2)	2(2, 2)				
	No. of Function Calls	11899	67460	173	4821	7074	173	303977	1198335	173				
Counter current reactors2	CPU (STU)	0.065	0.047	0.771	0.149	0.009	0.771	0.001	0.003	0.771	0.051	NA	Baron (0.0004)	NA
	No. of Stages	1	2	2									LINGO (infeas)	
	No. of SQP Calls	95	103	4418	95	31	809	20	17	6620				
	Avg No. Vars. in Parallel (max, min)	5.8 (6,5)	3.28 (4,2)	4.93 (6,4)	5.75 (6,5)	3.4(4, 2)	4.27(5, 4)	5.09(6, 5)	2.5(4, 2)	5.16(6, 4)				
	No. of Function Calls	1144	1202	173	1144	436	8177	1310	339	58506				
Cyclic5	CPU (STU)	0.006	0.771	0.771	0.771	0.771	0.771	0.771	0.771	0.771	NA	0.771	Baron (0.03)	0.771
	No. of Stages	1	5	3									LINGO (infeas)	1.1224
	No. of SQP Calls	804	11399	14231	683	809	853	7954	11918	9317				
	Avg No. Vars. in Parallel (max, min)	2.24 (3,2)	2.56 (3,2)	4.49 (5,2)	2(2, 2)	2.76(3, 2)	2.0 (2,2)	2.97(3, 2)	4.83(5, 2)	4.72(5, 2)				
	No. of Function Calls	4989	273376	100040	4161	6093	4301	45674	1469220	54836				

Name	Performance	Iterative Deepening			Worst First			Depth First			ALIAS QUAD	ICOS	Others First feasible Sol.	COCONUT
		SII Rule	Rule A	Smear	SII Rule	Rule A	Smear	SII Rule	Rule A	Smear				
Lorentz	CPU (STU)	0.005	0.003	0.547	0.003	0.002	0.771	0.009	0.771	0.771	NA	0.771	Baron (0.007)	0.2054
	No. of Stages	2	2	3								(>170)	LINGO(0.10)	
	No. of SQP Calls	138	85	1195	63	50	836	145	30458	6015				
	Avg No. Vars. in Parallel (max, min)	2.81(3, 2)	2.5(3, 2)	2.76(3, 2)	2.78(3, 2)	2.63(3, 2)	2.009(3, 2)	2.96 (3,2)	2.62(3, 2)	2.85(3, 2)				
	No. of Function Calls	976	706	7675	426	341	3553	808	758068	26840				
Quadfor2	CPU (STU)	0.005	0.006	0.000	0.005	0.006	0.000	0.771	0.771	0.000	NA	0.771	Baron (0.0006)	0.5808
	No. of Stages	1	1	1								(>170)	LINGO(Infeas)	
	No. of SQP Calls	28	32	3	46	24	3	16320	29728	3				
	Avg No. Vars. in Parallel (max, min)	3.36(4, 3)	3.75(4,2)	4(4, 4)	3.46(4, 3)	3.65(4, 2)	4(4, 4)	3.003(4, 3)	3.99(4,2)	4(4, 4)				
	No. of Function Calls	421	387	129	421	387	129	962610	1932908	129				
reimer5	CPU (STU)	0.037	0.255	0.771	0.771	0.771	0.771	0.771	0.771	0.057	NA	0.771	Baron (0.03)	0.771
	No. of Stages	2	3	4								(>170)	LINGO(0.9)	2.2148
	No. of SQP Calls	375	2933	10792	858	805	893	11919	30873	538				
	Avg No. Vars. in Parallel (max, min)	3.03(5, 2)	2.58(3,2)	3.44 (4,2)	2.25(3, 2)	2.78(3, 2)	3.76(4, 2)	2.02(3, 2)	2.51(3, 2)	3.4 (4, 2)				
	No. of Function Calls	2524	26387	67625	4571	4721	4749	323538	1154878	2757				
S9_1	CPU (STU)	0.045	0.052	0.771	0.771	0.049	0.771	0.771	0.771	0.771	NA	0.771	Baron (0.005)	0.771
	No. of Stages	1	1	3								(=2.3)	LINGO(0.1)	1.5854
	No. of SQP Calls	164	166	21805	783	185	829	16476	17138	6840				
	Avg No. Vars. in Parallel (max, min)	7.8(8, 7)	4.28(6,2)	5(5, 5)	3.16(4, 3)	4.38(6, 2)	5(5, 5)	3(3, 3)	4.28 (6, 2)	5(5, 5)				
	No. of Function Calls	4785	2077	474219	9423	2069	12741	1442068	2739504	87952				

Name	Performance	Iterative Deepening			Worst First			Depth First			ALIAS QUAD	ICOS	Others First feasible Sol.	COCONUT
		SII Rule	Rule A	Smear	SII Rule	Rule A	Smear	SII Rule	Rule A	Smear				
Solotarev	CPU (STU)	0.003	0.015	0.023	0.004	0.003	0.771	0.771	0.771	0.022	NA	0.771	Baron (0.005)	0.0462
	No. of Stages	2	3	4									LINGO(0.5)	
	No. of SQP Calls	75	450	592	63	55	1797	31071	17040	478				
	Avg No. Vars. in Parallel (max, min)	2.74(3, 2)	2.61(3,2)	2.011(3, 2)	2.74(3, 2)	2.72(3, 2)	2.006(3, 2)	2.99 (3,2)	2.53(3,2)	2.03(3,2)				
	No. of Function Calls	1054	4295	11553	593	381	19161	1790450	460680	3893				
Trinks	CPU (STU)	0.002	0.306	0.022	0.002	0.771	0.001	0.001	0.001	0.024	NA	0.771	Baron (0.001)	0.2142
	No. of Stages	1	4	1								(>170)	LINGO(0.0)	
	No. of SQP Calls	39	36544	13	39	784	13	15	15	11				
	Avg No. Vars. in Parallel (max, min)	2.81(3, 2)	3.23(4, 2)	2(2 ,2)	2.13(3, 2)	3.42(4, 2)	2(2, 2)	3 (3, 3)	3(4, 2)	2(2, 2)				
	No. of Function Calls	355	139440	97	355	9643	97	236	259	97				
Wright	CPU (STU)	0.001	0.002	0.001	0.001	0.002	0.001	0.001	0.002	0.001	NA	0.100	Baron (0.005)	1.31
	No. of Stages	1	1	1									LINGO(0.1)	2.185
	No. of SQP Calls	5	5	5	5	5	5	8	5	5				
	Avg No. Vars. in Parallel (max, min)	2.44(3, 2)	2.75(3,2)	3.44 (4, 2)	2.44(3, 2)	2.55(3, 2)	3.44(4, 2)	2.14 (3, 2)	2.5(3,2)	3.11(4, 2)				
	No. of Function Calls	166	141	231	166	141	231	137	154	177				

** indicates simple consistency check is applied linear equations*

Appendix – B

List of Constrained Optimization Problems

PROBLEM	(Dim, # Nonlin. Eq., # Lin. Eq., # Nonlin. Ineq., #Lin. Ineq.)	Description	Reference
Aircraftb	18, 5, 5, 0, 0	Objective function is a second degree polynomial function 5 quadratic equations 5 linear equation	Coconut
Avgasb	8, 0, 0, 10, 0	Objective function is a quadratic function 10 linear inequality constraints	Princetonlib
Alkyl	14, 6, 1, 0, 0	Objective function is a quadratic function 2 highly dependent quadratic equations 4 quadratic equations 1 linear equation	Coconut
Bt4	3, 1, 1, 0, 0	Objective function is a quadratic function 1 linear equation 1 quadratic equation	Coconut
Bt8	5, 2, 0, 0, 0	Objective function is a quadratic function 2 quadratic equation	Coconut
Bt12	5, 3, 0, 0, 0	Objective function is a quadratic function 3 quadratic equation	Coconut
Bt11	5, 2, 1, 0, 0	Objective function is a quadratic function 2 quadratic equation 1 linear equation	Coconut
Bt7	5, 3, 0, 0, 0	Objective function is a quadratic function 1 quadratic equation 2 second degree polynomial equation	Coconut
Dispatch	4, 1, 0, 0, 1	Objective function is a second degree function 1 quadratic equation 1 linear equality	Coconut
Dipigri	7, 0, 0, 4, 0	Objective function is a quadratic function 1 quadratic inequality 3 nonlinear second degree polynomial inequalities	Coconut
Degenlpa	20, 0, 14, 0, 0	Objective function is a linear function 14 linear equations	Coconut
Degenlpb	20, 0, 15, 0, 0	Objective function is a linear function 15 linear equations	Coconut
Eigminc	22, 22, 0, 0, 0	Objective function is a linear function 1 second degree polynomial equation 21 quadratic equations	Coconut
Ex5_2_4	7, 0, 1, 3, 2	Objective function is a quadratic function 1 linear equation 2 linear inequalities 3 nonlinear quadratic inequalities	Coconut
Ex9_1_4	10, 4, 5, 0, 0	Objective function is a linear function 5 linear equation 4 quadratic equations	Coconut
Ex8_4_2	24, 10, 0, 0, 0	Objective function is a second degree	Coconut

PROBLEM	(Dim, # Nonlin. Eq., # Lin. Eq., # Nonlin. Ineq., #Lin. Ineq.)	Description	Reference
		polynomial function 10 quadratic equation	
Ex9_2_5	7, 3, 4, 0, 0, 0	Objective function is a second degree polynomial function 3 quadratic equation 4 linear equation	Coconut
Ex14_1_5	6, 0, 4, 2, 0	Objective function is a linear function 2 highly interactive quadratic equations 4 linear equations	Coconut
Ex9_2_6	16, 6, 6, 0, 0	Objective function is a second degree polynomial function 6 quadratic equations 6 linear equations	Coconut
Ex9_2_7	10, 4, 5, 0, 0	Objective function is a second degree polynomial function 4 quadratic equations 5 linear equations	Coconut
Ex9_1_2	10, 4, 5, 0, 0	Objective function is a linear function 4 quadratic equations 5 linear equations	Coconut
Ex2_1_9	10, 0, 1, 0, 0	Objective function is a quadratic function 1 linear equations	Coconut
Ex2_1_3	13, 0, 0, 0, 9	Objective function is a second degree polynomial function 9 linear inequality	Coconut
Ex8_4_1	22, 10, 0, 0, 0	Objective function is a second degree polynomial function 10 quadratic equations	Coconut
Ex8_4_5	15, 11, 0, 0, 0	Objective function is a second degree polynomial function 11 quadratic equations	Coconut
F_e	7, 0, 0, 3, 4	Objective function is a linear function 1 second degree polynomial inequalities 2 quadratic inequalities 4 linear inequalities	Epperly
Fermat_scop_vareps	5, 0, 0, 3, 0	Objective function is a linear function 3 nonlinear inequalities	Princetonlib
Fp_2_1	6, 0, 0, 1, 1	objective function is a linear function 1 nonlinear second degree polynomial inequality 1 linear inequality	Epperly
Genhs28	10, 0, 8, 0, 0	objective function is a quadratic function 8 linear equations	Coconut
Hs087	11, 4, 2, 0, 0	objective function is a linear function 4 nonlinear trigonometric equations 2 linear equations	Coconut

PROBLEM	(Dim, # Nonlin. Eq., # Lin. Eq., # Nonlin. Ineq., #Lin. Ineq.)	Description	Reference
Hs053	5, 0, 3, 0, 0	objective function is a quadratic function 3 linear equations	Coconut
Hs056	7, 4, 0, 0, 0	objective function is a interactive quadratic function 4 nonlinear trigonometric equations	Coconut
Hs407	5, 3, 0, 0, 0	objective function is a highly interactive quadratic function 1 third degree polynomial equation 1 second degree polynomial equation 1 quadratic equation	Coconut
Hs108	9, 0, 0, 12, 0	objective function is a quadratic function 1 second degree polynomial equation 11 quadratic equation	Coconut
Hs080	5, 3, 0, 0, 0	objective function is a exponential function 1 second degree polynomial equation 1 quadratic equation 1 third degree polynomial equation	Coconut
Hs043	4, 0, 0, 3, 0	objective function is a second degree polynomial function 3 second degree polynomial inequalities	Coconut
Hs116	13, 0, 0, 10, 5	objective function is a linear function 10 quadratic inequalities 5 linear inequalities	Coconut
Himmel11	9, 3, 0, 0, 1	objective function is a quadratic function 1 linear inequality 3 quadratic equation	Coconut
Immum	21, 0, 7, 0, 0	objective function is a second degree polynomial function 7 linear equations	Coconut
Lootsma	3, 0, 0, 2, 0	objective function is a three degree polynomial function 2 second degree polynomial inequalities	Coconut
Lewispol	6, 6, 3, 0, 0	objective function is a second degree polynomial function 6 third degree polynomial equations 3 linear equations	Coconut
Mwright	5, 3, 0, 0, 0	objective function is a quadratic function 1 quadratic equations 2 second degree polynomial equations	Coconut
Mhw4d	5, 3, 0, 0, 0	objective function is a quadratic function 1 quadratic equations 1 second degree polynomial equations 1 third degree polynomial equations	Coconut
Madsen	3, 0, 0, 6, 0	Objective function is a linear function 4 trigonometric inequalities 2 quadratic inequalities	Coconut
Minmaxrb	3, 0, 0, 2, 2	Objective function is a linear function 2 linear inequalities 2 second degree polynomial inequalities	Coconut
Median_scop_v	5, 0, 0, 3, 0	Objective function is a linear function	Coconut

PROBLEM	(Dim, # Nonlin. Eq., # Lin. Eq., # Nonlin. Ineq., #Lin. Ineq.)	Description	Reference
areps		3 nonlinear inequalities	
Matrix2	6, 0, 0, 2, 0	Objective function is a quadratic function 2 quadratic inequalities	Coconut
Mistake	9, 0, 0, 12, 0	Objective function is a quadratic function 10 quadratic inequalities 2 second degree polynomial inequalities	Coconut
O32	5, 0, 0, 6, 0	Objective function is a quadratic function 6 quadratic inequalities	Coconut
Pgon	12, 0, 0, 15, 5	Objective function is a trigonometric function 15 nonlinear trigonometric inequalities 5 linear inequalities	Coconut
Robot	14, 2, 0, 0, 0	Objective function is a quadratic function 2 nonlinear trigonometric equations	Coconut
Rk23	17, 7, 4, 0, 0	Objective function is a linear function 4 quadratic equations 3 highly interactive quadratic equations 4 linear equations	Coconut
S381	13, 0, 1, 0, 3	Objective function is a linear function 1 linear equations 3 linear inequalities	Princetonlib
S355	8, 5, 0, 0, 0	Objective function is a second degree polynomial function 4 quadratic equations 1 second degree polynomial equation	Princetonlib
S336	3, 1, 1, 0, 0	Objective function is a linear function 1 linear equation 1 second degree polynomial equation	Princetonlib
S262	4, 0, 1, 0, 3	Objective function is a linear function 1 linear equation 3 linear inequalities	Princetonlib
S203	5, 3, 0, 0, 0	Objective function is a Second degree polynomial function 3 quadratic equation	Princetonlib
Springs_nonconvex	32, 0, 0, 10, 0	Objective function is a Second degree polynomial function 9 quadratic inequalities 1 second degree polynomial inequality	Princetonlib
Steifold	4, 3, 0, 0, 0	Objective function is a Second degree polynomial function 1 quadratic equation 2 second degree polynomial equations	Balogh and Toth
Sample	4, 0, 0, 2, 0	Objective function is a linear function 2 quadratic inequalities	Princetonlib