Available online at www.sciencedirect.com
ScienceDirect
Computers & Operations Research 35 (2008) 1412–1435
computers & operations research
www.elsevier.com/locate/cor

Efficient interval partitioning—Local search collaboration for constraint satisfaction

Chandra Sekhar Pedamallu[a], Linet Ozdamar[b,*], Martine Ceberio[c]

[a]*Mechanical & Production Engineering, Division of Systems and Engineering Management, Nanyang Technological University, Singapore*
[b]*Department of Logistical Management, Mithatpasa Cad. 937/10, Guzelyali, Izmir Economy University, Izmir, Turkey*
[c]*Computer Science Department, University of Texas at El Paso (UTEP), El Paso, TX, USA*

Available online 4 October 2006

Abstract

In this article, a cooperative solution methodology that integrates interval partitioning (*IP*) algorithms with a local search, feasible sequential quadratic programming (*FSQP*), is presented as a technique to enhance the solving of continuous constraint satisfaction problems (continuous CSP). *FSQP* is invoked using a special search tree management system developed to increase search efficiency in finding feasible solutions.

In this framework, we introduce a new symbolic method 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 published results of established symbolic-numeric methods for solving CSP on a number of state-of-the-art benchmarks.

Keywords: Interval partitioning algorithms; Subdivision direction selection; Tree management; Cooperative local search; Feasible sequential quadratic programming

1. Introduction

Continuous constraint satisfaction problems (CSP), (also known as numeric CSP) are a very well known framework for expressing and solving problems. CSP, and specifically continuous CSP, are very useful in many real-world applications, ranging for instance from chemical engineering to aircraft design.

A CSP is totally defined by

- a set of variables, $V = \{x_1, \ldots, x_n\}$;
- a set of constraints, $C = \{c_1, \ldots, c_r\}$, over the variables x_i of the problem: numeric constraints are linear or nonlinear equations or inequalities. Let us represent them as follows:

$$g_i(x_1, \ldots, x_n) \leqslant 0, \quad i = 1, \ldots, k,$$

* Corresponding author.
E-mail address: linetozdamar@lycos.com (L. Ozdamar).

0305-0548/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2006.08.003

$$h_i(x_1, \ldots, x_n) = 0, \quad i = k+1, \ldots, r,$$

variable domains $D_i = [\underline{D_i}, \overline{D_i}]$ for x_i, $i = 1, \ldots, n$.

A solution of a CSP is an element (x^*) of the search space ($X = D_1 \times \cdots \times D_n$), that meets all the constraints, i.e., that satisfies all equations/inequalities simultaneously.

CSP problems are difficult to solve because, unless we deal with a specific CSP whose constraints can be solved with a dedicated algebraic method, the only way parts of the search space can be by proving that they do not contain a feasible solution. It is hard to tackle general nonlinear CSP with computer algebra systems, and in general, traditional numeric algorithms cannot guarantee completeness in the sense that some solutions may be missed and/or the search might result with an infeasible result (despite the fact that a solution exists).[1]

As a result, we distinguish complete from incomplete solvers/solving techniques, complete solvers being those that guarantee to provide the user with all the solutions of the original CSP.[2]

Neumaier et al. [1] compare the performance of major complete/global and incomplete/local solvers using an extensive set of constrained optimization problems and CSP. In particular, interval methods (see [2] for a complete description of intervals, and [3–5] for a description of interval solving methods) are complete because they never discard a subdomain that has a potential to contain feasible solutions, thanks to the inherent feasibility cut-off test carried out by using interval arithmetic (IA). Conventionally, interval and non-interval partitioning methods use local search procedures to identify feasible solutions. By principle, an interval partitioning (IP) method continues to subdivide a given box (sub-domain) until either it becomes infeasible and discarded by the cutoff test, or it becomes small enclosure with the potential to contain a feasible solution. During the partitioning process, an increasing number of distinct feasible solutions are identified within promising boxes. A box is subjected to local search only once, because after that it gets re-partitioned again and feasible solutions are searched in its child boxes. Therefore, it is not important whether or not a box contains multiple solutions, because the partitioning will continue until the cutoff test verifies that a child box does not contain any solutions.

Lebbah [6] describes an interval method, ICOS, that is praised for its reliability in finding solutions of CSPs. However, a note is made on its slow convergence [1]. Furthermore, results for ICOS are reported for the identification of a first feasible solution rather than for all feasible solutions.

An important issue in enhancing the convergence rate of an interval method is the selection of the subdomain to explore. This does not only involve the box ranking decision in the search tree. An equally significant decision is how to sub-divide a given box, that is, the selection of the variables to bisect so as to concentrate in a given area of a promising box. In this article, we are interested in designing a good algorithm of choice for domain splitting as well as box ranking.

This paper introduces a cooperative strategy for solving continuous CSP, which integrates:

- an IP algorithm that conducts systematic parallel variable domain bisection; with
- a dynamic stage-wise tree management system that activates calls to the local solver, feasible sequential quadratic programming (*FSQP*) [7–9];

in order to determine *all* solutions of the continuous CSP. The basic domain reduction technique used in *IP* is interval evaluation and bisection.

Our interval approach focuses on how to bisect boxes (the sequence of variables to be split in a given box) rather than on the implementation of consistency techniques. We introduce a new symbolic subdivision direction selection approach (Symbolic Interval Inference, *SII*) that guides *IP* in identifying the best sequence of variables to be partitioned. *SII* identifies major impact variables in a constraint expression, so that the degree of uncertainty of the infeasibility of the constraint is reduced in the immediate child boxes. The latter speeds up *IP*s convergence rate significantly as compared to well-established variable selection rules. The minimum number of variables that are partitioned in parallel is two for *SII* and the other rules tested here. We also introduce a simple parallelization scheme that prevents both *SII* and other selection rules from bisecting too many or too few variables. In this approach, every variable whose calculated weight is more than the average weight of all candidate variables is bisected. This partitioning

[1] An infeasible result is a result that does not meet the constraints. A solution of a CSP is sometimes called a feasible element.

[2] In addition to all solutions, complete solvers also return some noise, that is, infeasible solutions.

scheme seems to be quite effective in reducing CPU times and makes *IP* a viable method for identifying *all* solutions of the CSP. The latter is very important in certain fields. For instance, identifying all configurations of a parallel manipulator in real time maximizes workspace coverage. To evaluate the proposed *IP*, we conduct tests on a number of CSP benchmarks published in COPRIN (http://www-sop.inria.fr/coprin/logiciels/ALIAS/Benches/) and COCONUT (COCOS) project web pages (http://www.mat.univie.ac.at/~neum/glopt/coconut/benchmark.html). We compare *SII* with established variable selection rules, and the adaptive tree management strategy with two others.

The COPRIN project web page publishes results obtained by using ALIAS, a comprehensive set of C++ libraries that include 2B/3B box and hull consistency variants, linearization [10], the interval Newton, unicity operators, and numerical and interval root approximations for univariate polynomials (http://www-sop.inria.fr/coprin/logiciels/ALIAS/ALIAS-C++/ALIAS-C++.html). On the other hand, COCOS proposes an advanced *IP* algorithm cooperating with hull consistency and linear programming filtering techniques. In the COCONUT project, an extensive testing has been carried out for box-constrained and constrained optimization problems as well as CSPs.

Besides the results of the proposed *IP*, published results of ALIAS, COCOS, ICOS [6] and QUAD (for two problems reported in [11]) are also provided for the selected test bed as additional information. Among the latter methods, ALIAS and QUAD results are provided for identifying all solutions whereas COCOS and ICOS results are reported for identifying a single solution.

The paper is organized as follows. An initial introduction to *IP* is followed by the description of the proposed *IP* with adaptive tree management approach that also coordinates *FSQP* calls. A new subdivision direction selection rule that improves the efficiency of the approach is then explained. Implementation and comparison of the proposed approach is detailed in the numerical results section. The paper ends with conclusions and areas of future research.

2. Preliminary notions

2.1. Continuous constraints

Definition 1. A *continuous real constraint* is an atomic formula made of expressions and a relation symbol, i.e., an equality or an inequality.

In the following, when there is no risk of ambiguity, we will refer to such continuous real constraints as constraints.

Definition 2. Let c be a real continuous constraint defined over D included in $\mathbb{R}^n$. Let ρ_c be the subset of D satisfying c. Then any element s of D 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 D into two distinct subsets: the subset of consistent elements and the subset of inconsistent elements ($D \backslash \rho_c$).

Definition 3. A *constraint system* is the conjunction of a set of constraints $C = \{c_1, \ldots, c_r\}$ defined over a set of variables $V = \{x_1, \ldots, x_p\}$, each variable x_i defined over a given domain D_i of $\mathbb{R}$. A constraint system is denoted by $S = (V, C, D)$, where $D = D_1 \times \cdots \times D_p$, and the corresponding problem to solve is called a constraint satisfaction problem, usually referred to as CSP. Let us note that, when this is not ambiguous, we may refer to a CSP, $S = (V, C, D)$, simply as C.

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

Our objective is, given a constraint system, to solve it in a complete manner. This means that we want our solving technique to isolate all solutions of the constraint system, and provide us with all of them. However, using classical floating-point arithmetic generates rounding-error, and therefore using such arithmetic's may return solutions that are inconsistent (incorrectness), and may not even return all the solutions (incompleteness). In order to tackle this problem, inherent to solving (in a complete manner) continuous constraint systems, we choose to use IA instead of floating-point arithmetic (refer to [12,13] for floating-point arithmetic, to [14] for inexact computations).

2.2. Intervals

Definition 4. IA (Refs. [2,15,16]) is an arithmetic defined on convex sets of real numbers, called *intervals*.

The *set of intervals* is denoted by $\mathbb{I} := \{[a, b] | a \leqslant b, a, b \in \mathbb{I}. \text{ or } \mathbb{I}.$

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

For every $a \in \mathbb{I}$, 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 $w(X) = \overline{X} - \underline{X}$. Given two real intervals X and Y, X is said to be *tighter than* Y if $w(X) \leqslant w(Y)$.

Elements of $\mathbb{I}^n$ also define boxes. Given $(X_1, \ldots, X_n)^{\mathrm{T}} \in \mathbb{I}^n$, the corresponding *box* is the Cartesian product of intervals, $\boldsymbol{X} = X_1 \times \cdots \times X_n$, where $\boldsymbol{X} \in \mathbb{I}^n$. A subset of $\boldsymbol{X}$, $\boldsymbol{Y} \subseteq \boldsymbol{X}$, is a sub-box of $\boldsymbol{X}$. By misuse of notation, the same symbol is used for vectors and boxes. The notion of *width* is defined as follows:

$$w(X_1 \times \cdots \times X_n) = \max_{1 \leqslant i \leqslant n} w(X_i).$$

IA operations are set theoretic extensions of the corresponding real operations. Given $X, Y \in \mathbb{I}$, and an operation $\diamond \in \{+, -, \times, \div\}$, we have: $X \diamond Y = \{\text{Hull } x \diamond y \text{ where } (x, y) \in X \times Y\}$.

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 law and the commutative law are preserved over $\mathbb{I}$. However, the distributive law does not hold. In general, only a weaker law is verified, called semi-distributivity.

We observe in particular that equivalent expressions over the real numbers are no longer equivalent when handling intervals: different symbolic expressions may lead to different interval evaluations.

For instance, consider the following three expressions equivalent over the real numbers: $x^2 - x$, $x(x - 1)$, and $(x - \frac{1}{2})^2 - \frac{1}{4}$. When evaluated over the intervals, with $X = [0, 1]$, we obtain the following results: $X^2 - X = [-1, 1]$, $X(X-1) = [-1, 0]$, $(X - \frac{1}{2})^2 - \frac{1}{4} = [-\frac{1}{4}, 0]$. Expressions formerly equivalent over the real numbers are not necessarily equivalent when *extended to* the intervals. This problem is known as the dependency problem of IA, and a fundamental open problem in IA consists in finding expressions that lead to tight interval computations.

2.3. Interval extensions of functions and constraints

IA is particularly appropriate to represent outer approximations of real quantities. In particular, the range of a real function f over a domain D, denoted by $\boldsymbol{f}^u(D)$, can be computed by interval extensions.

Definition 5 (*Interval extension*). An *interval extension* of a real function $f: D_f \subset \mathbb{R}^n \to \mathbb{R}$ is a function $\vartheta : \mathbb{R}^n \to \mathbb{R}$ such that

$$\forall \boldsymbol{X} \in \mathbb{I}^n, \quad (\boldsymbol{X} \in D_f \Rightarrow \boldsymbol{f}^u(\boldsymbol{X}) = \{f(x) | x \in \boldsymbol{X}\} \subseteq \boldsymbol{\vartheta}(\boldsymbol{X})).$$

This inclusion formula is the basis of what is called the Fundamental Theorem of IA. In brief, interval extensions always enclose the range of the corresponding real function. As a result, suppose that, for instance, you are looking for a zero of a real function f over domain D. If the evaluation of an interval extension of f over D does not contain zero, it means that zero was not part of the range f over D in first place.

Interval extensions are also called *interval forms* or *inclusion functions*.

Let us note that 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 $X \to \text{Hull}\boldsymbol{f}^u(X)$. Both satisfy the inclusion formula.

The most common extension is known as the *natural extension*. Natural extensions are obtained by replacing each arithmetic operation in an expression of a real function with an enclosing interval operation. As a result, natural extensions are *inclusion monotonic* (this property follows from the monotonicity of interval operations), which means that, given a real function f, (whose natural extension is denoted F), and two intervals $\boldsymbol{x}$ and $\boldsymbol{y}$ such that $X \subset Y$, we have: $F(X) \subset F(Y)$.

Since natural extensions are defined by the syntax of real expressions, two equivalent expressions of a given real function f generally lead to different natural interval extensions. As we saw the pervious section, two symbolic expressions, equivalent on the real numbers, may not be equivalent on intervals. There is then necessarily at least one of the two that overestimates the targeted real quantity.

The overestimation problem, known as the *dependency problem of IA*, is due to the decorrelation of the occurrences of a variable during interval evaluation. For instance, given $X = [a, b]$ with $a \neq b$, we have:

$$X - X = [a - b, b - a] \underset{\neq}{\supset} 0.$$

An important result is Moore's theorem known as the *theorem of single occurrences* [15].

Theorem 1. *Let t be a symbolic expression, and let f: $D_f \subset \mathbb{R}^n \to \mathbb{R}$, $\{x_1, \ldots, x_n\} \to t\{x_1, \ldots, x_n\}$ be a real function interpreting t. If each variable x_i, $1 \leqslant i \leqslant n$, occurs only once in t, then the interval evaluation of the range of f using the natural extension of f based on t is exact* (*i.e.*, *not overestimated*). *In particular, we have* $\forall \boldsymbol{X} \in \mathbb{R}^n$, $(\boldsymbol{X} \subseteq D_f \Rightarrow \boldsymbol{f}^u(\boldsymbol{X}) = f(\boldsymbol{X}))$.

In other words, there is no overestimation if all variables occur only once in the given expression.

2.4. Interval constraints

Definition 6. An *interval constraint* is built from an atomic interval formula (interval function) and relation symbols, whose semantics is extended to intervals as well.

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

Similarly to a real constraint, an interval constraint c divides the search space into two disjoint sets: the set of intervals that satisfy c (consistent intervals), and the set of inconsistent intervals.

Definition 7. Let c be a real constraint defined by $f \diamondsuit g$, where $\diamondsuit$ is the relation symbol (either $=$ or $\leqslant$). Let F be the natural interval extension of f, and G be the natural interval extension of g. Then the natural interval extension of c is the constraint C such that the relation it defines is as follows:

- If $\diamondsuit$ is $=$ then
  $\boldsymbol{\rho}_c = \{\{X_1, \ldots, X_p\} \in \mathbb{I}^p | F(X_1, \ldots, X_p) \cap G(X_1, \ldots, X_p) \neq \phi\}$.
- If $\diamondsuit$ is $\leqslant$ then
  $\boldsymbol{\rho}_c = \{\{X_1, \ldots, X_p\} \in \mathbb{I}^p | \underline{F}(X_1, \ldots, X_p) \leqslant \overline{G}(X_1, \ldots, X_p)\}$.

CSPs are extended in a similar way, where each constraint of the system is replaced by its natural extension.

The main guarantee of interval constraint is that if its solution set is empty, it has no solution; it follows that if the solution set of a CSP is empty, and then the original CSP is guaranteed not to have a solution.

Example (*Eliminating boxes*). Let us consider constraint c defined by: $x^2 - x = y + 3$. c has no solution in box $[0, 1] \times [-1, 1]$. Indeed on this box, we have: $([0, 1]^2 - [0, 1]) \cap ([-1, 1] + 3) = [-1, 1] \cap [2, 4] = \phi$. As a consequence, if $[0, 1] \times [-1, 1]$ is a part of the search space, then this box can be discarded.

Unfortunately, as pointed out before, IA is often too weak and generates evaluation overestimation. As a consequence, it does not happen as often as it is actually the case, that such an intersection as shown in the above example is empty. Therefore, parts of the search space that do not contain solutions are not discarded and kept for further exploration (for lack of better knowledge about their inconsistency), while they should be disregarded. This slows down the overall process by forcing to perform unnecessary operations.

One way to limit the overestimation of evaluation is to modify the expressions of the functions symbolically, therefore obtaining different and hopefully more efficient interval extensions of these functions. Indeed, input symbolic forms can be converted into more manageable/solvable expressions by simplication/factorization [17], by addition of redundant constraints, by using Grobner bases [18], by linearization [10,19], and by conversion of constraints to univariate constraints. All these techniques may have a strong impact on both preventing overestimation in interval computations, and enhancing the solving of the original problem. However, in some cases (such as for Grobner bases), such symbolic manipulations may lead to the inclusion of too many new variables, generating thus an extra-computational cost.

2.5. Interval constraint solving

The general scheme for interval constraint solving is as follows. Interval techniques for solving CSP are based on branch and prune/splitting and filtering approaches:

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

Splitting is generally carried out by bisecting the domain(s) of the selected variable(s).[3] 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 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 [21]).

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 $\boldsymbol{X}$. If the interval evaluation of f over $\boldsymbol{X}$ does not contain 0, then you can immediately conclude that c is inconsistent (i.e., is not feasible) over $\boldsymbol{X}$, and you can discard $\boldsymbol{X}$. Convergence of this method is very slow for it totally relies on interval evaluations that are known to overestimate ranges of functions.

In this article, we focus on splitting and box selection strategies of IP algorithms.

3. IP methods

3.1. Basic IP algorithm

IP is a systematic variable domain partitioning algorithm that conducts interval evaluation over sub-domains (boxes) and eliminates inconsistent boxes from the exploration space. Boxes are sub-divided until either they are found to be consistent (and stored) or their size falls below a given tolerance (stored as potential solution enclosures). The basic *IP* algorithm is given below.

Procedure IP

Step 0: Set first box $\boldsymbol{Y} = \boldsymbol{X}$. Set list boxes, $\mathscr{B} = \{\boldsymbol{Y}\}$.

[3] Splitting 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 [20].

Step 1: If $\mathscr{B} = \phi$, (or if known, the theoretical number of solutions are found) stop and report all feasible solutions found. Else, pick first box $\boldsymbol{Y}$ in $\mathscr{B}$.

1.1. If $\boldsymbol{Y}$ is infeasible, remove $\boldsymbol{Y}$ from $\mathscr{B}$ and repeat Step 1.
1.2. If $\boldsymbol{Y}$ is sufficiently small, store it as a possible solution enclosure, remove $\boldsymbol{Y}$ from $\mathscr{B}$ and repeat Step 1.
1.3. Else, go to Step 2.

Step 2. Select variable(s) to partition (use any subdivision direction selection rule). Set υ=number of variables to partition.

Step 3. Partition $\boldsymbol{Y}$ into 2^{υ} non-overlapping child boxes.

Step 4. Remove $\boldsymbol{Y}$ from $\mathscr{B}$ and add 2^{υ} boxes to $\mathscr{B}$. Go to Step 1.

IP partitions variable domains until either a box is feasible, or it is sufficiently small and still has a potential of enclosing a feasible solution $\boldsymbol{x}^*$. Otherwise, when the box is guaranteed to be infeasible, *IP* discards it. Below we provide the criteria for box assessment.

1. If $\overline{G}_i(\boldsymbol{Y}) \leqslant 0$, and $H_i(\boldsymbol{Y}) = 0$, for $\forall i$, then box $\boldsymbol{Y}$ is a *feasible* box and it is stored. (In our implementation, it is sufficient to satisfy all equality constraints with a precision of $\varepsilon = 10^{-15}$, that is, $H_i(\boldsymbol{Y}) \in [-\varepsilon, \varepsilon]$.)
2. If $\underline{G}_i(\boldsymbol{Y}) > 0$ or $0 \notin H_i(\boldsymbol{Y})$ for any i, then box $\boldsymbol{Y}$ is called an *infeasible* box and it is discarded.
3. If Y cannot be discarded or stored, that is, if $(\underline{G}_i(\boldsymbol{Y}) < 0$ AND $\overline{G}_i(\boldsymbol{Y}) > 0)$ OR $(0 \in H_i(\boldsymbol{Y}) \neq 0)$ for any i, implying that indeterminate constraints exist, then $\boldsymbol{Y}$ is called an *indeterminate* box and it holds the potential of containing both $\boldsymbol{x}^*$ and $\boldsymbol{X} \backslash \boldsymbol{x}^*$.

Definition 8. The degree of uncertainty, $PG^i_{\boldsymbol{Y}}$ ($PH^i_{\boldsymbol{Y}}$) of an indeterminate inequality (equality) is defined by Eqs. (1)–(2), respectively,

$$PG^i_Y = \overline{G}_i(\boldsymbol{Y}), \tag{1}$$

$$PH^i_Y = \overline{H_i}(\boldsymbol{Y}) + |\underline{H}_i(\boldsymbol{Y})|. \tag{2}$$

Definition 9. The total uncertainty degree of a box, IF_Y, is the sum of uncertainty degrees of equalities and inequalities that are indeterminate over $\boldsymbol{Y}$.

IP is a reliable convergent algorithm that sub-divides an indeterminate box to reduce PG^i_Y and PH^i_Y of all constraints and it drives IF_Y to zero in the limit by nested partitioning. The contraction and α-convergence properties enable this. In the end, all feasible solutions are identified within sufficiently small boxes; however, convergence rate might be very slow. An exception may occur when all constraints of the CSP are equalities. In such a case, the theoretical number of solutions might be known. If such information exists, then IP might be stopped when it encloses all distinct feasible solutions.

3.2. IP algorithm with adaptive tree management approach

The tree management system in the proposed *IP* maintains a stage-wise branching scheme that is conceptually similar to the iterative deepening approach [22]. 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. That is, a box is selected among the children of the same parent according maximum IF_Y criterion (worst-first, opposite of best-first, hence, we name it as best-first) 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 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. As mentioned previously, whether or not *FSQP* fails to find a solution or when there are multiple solutions in

a box, *IP* will continue to partition the box since it passes the cutoff test as long as it has potential to contain a solution. Finally, the algorithm encloses potential solutions in sufficiently small boxes where local search can identify them. In *IP*, *FSQP* acts as a catalyst that occasionally scans larger boxes to identify solutions at earlier stages of the search.

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 only of $\boldsymbol{X}$ and B_{s+1} is empty. As child boxes are added to a selected parent box, they are ordered in descending order of IF_Y. Boxes are selected in descending order of IF_Y, because the algorithm aims to delete as many boxes as possible in the early stages of the search. 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 possible. 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 when there are no boxes remain in B_s and B_{s+1}. As mentioned earlier, if a procedure exists to find the theoretical number of solutions, the performance of the algorithm might then be the amount of CPU time taken to identify them all. Otherwise, the number of solutions found within a limited CPU time would then indicate the success of the algorithm. The proposed *IP* algorithm is described below.

IP with adaptive tree management

Step 0: Set tree stage, $s=1$. Set 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 $\boldsymbol{X}$, $B_s=\{\boldsymbol{X}\}$, and $B_{s+1}=\emptyset$.

Step 1: If $B_s=\emptyset$ and $B_{s+1}\neq\emptyset$, then set $s\leftarrow s+1$, set $r\leftarrow s$, and repeat Step 1. If both $B_s=\emptyset$ and $B_{s+1}=\emptyset$, then stop.

Else pick first box $\boldsymbol{Y}$ in B_s and continue.

1.1. If $\boldsymbol{Y}$ is infeasible, discard $\boldsymbol{Y}$, and go to Step 1.
1.2. If $\boldsymbol{Y}$ is sufficiently small, store it as a solution enclosure, remove $\boldsymbol{Y}$ from B_s, and go to Step 1.
1.3. Else, go to Step 2.

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

Step 3: Partition $\boldsymbol{Y}$ 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 $\boldsymbol{Y}$ 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 Fig. 1 on a small tree where node labels indicate the order of nodes visited.

4. A novel subdivision direction selection rule for IP algorithm

4.1. General overview

The order in which variable domains are partitioned has an impact on the convergence rate 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

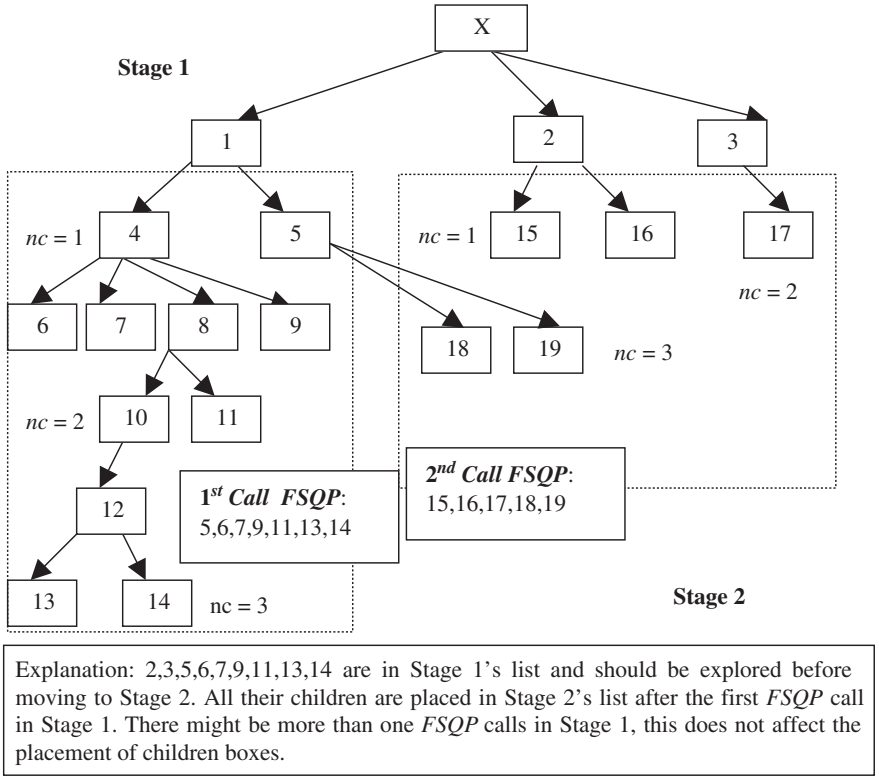

Fig. 1. Implementation of the adaptive iterative deepening procedure.

a new symbolic subdivision direction selection rule, *SII*, to improve *IP*s convergence rate by partitioning in parallel, those variable domains that reduce IF_Y in immediate child boxes. Hence, new boxes are formed with an appropriate partitioning sequence resulting in diminished uncertainty caused by the overestimation in constraint ranges. Before *SII* is applied, each constraint is interpreted as a binary tree that represents all recursive sub-expressions hierarchically. Such a binary tree enables interval propagation over all sub-expressions of the constraint [23]. Interval propagation and function trees are used by Kearfott [24] in improving interval Newton approach by decomposition and variable expansion, by Smith and Pantelides [25] in automated problem reformulation, by Sahinidis [26] and Talawarmani and Sahinidis [27] where feasibility based range reduction is achieved by tightening variable bounds.

After interval propagation is carried out over the sub-expressions in the binary tree, *SII* traverses the tree to label its nodes so as to identify the pair of variables (source variables) that are most influential on the constraint's uncertainty degree. These two maximum impact variables are identified for each constraint and placed in the pool of variables whose domains are possibly partitioned in the next iteration. This pool is screened and reduced by allocating symbolic weights to variables and re-assessing them.

4.2. Symbolic subdivision direction selection rule

4.2.1. Interval propagation over a binary tree

Before the labeling process *SII_Tree* can be applied on a constraint expression, it has to be parsed and intervals have to be propagated through all sub-expression levels. This is achieved by calling an *Interval Library* at each (molecular) sub-expression level of the binary tree from bottom to top 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 "Θ" is an arithmetic operator ($\cdot, +, -, /$) 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

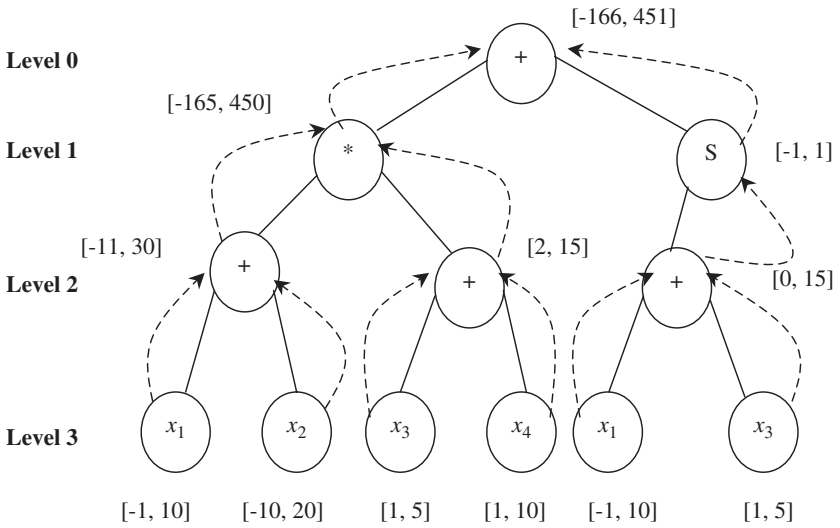

Fig. 2. Interval propagation for "$((x_1 + x_2) * (x_3 + x_4)) + \sin(x_1 + x_3)$". (Legend: *S*: sin function).

below is illustrated in Fig. 2.

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

Variable intervals in the box are $X_1 = [-1.0, 10.0]$, $X_2 = [-10.0, 20.0]$, $X_3 = [1.0, 5.0]$, and $X_4 = [1.0, 10.0]$. In this expression, the partial expression "$\sin(x_1 + x_3)$" contains one unary operator (sin) that always branches out to its left, however, the addition operator within the *sin* operator is a binary operator connecting x_1 and x_3. In Fig. 2, 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 whole constraint is reached resulting in the constraint range of $[-166, 451]$.

4.2.2. SII_Tree labeling procedure

Suppose a binary tree is constructed for a constraint and its source variables have to be identified over a given box Y. A labeling procedure called *SII_Tree* accomplishes this by tree traversal. We shall assume that the expression depicted in Fig. 2 represents an inequality constraint. In Fig. 3, the path constructed by *SII_Tree* is illustrated graphically. Straight lines in the figure indicate the propagation tree, dashed arrows indicate binary decisions, and bold arrows indicate the path constructed by *SII_Tree*.

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

Q^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 Q^k at level $k + 1$
$[\underline{Q}^k, \overline{Q}^k]$	interval bounds of parent sub-expression Q^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
$\mathscr{L}^k$	labeled bound at level k

The tree traversal procedure *SII_Tree* starts applying *SII* by targeting $\overline{Q}^0$. The target is $\overline{G}_i(Y))$ for inequalities so as to reduce PG_Y^i, and in equalities the target is $\max\{|\underline{Q}^0|, \overline{Q}^0\}$ (i.e., $\max\{|\underline{H}_i(Y)|, \overline{H}_i(Y)\}$).

Suppose we have the inequality example in Fig. 3 with the expression interval $[-166, 451]$. Then, "451" is selected as labeled bound $\mathscr{L}^0$ at the root node. Next, we determine which pair of interval bounds ($\{\underline{L}^1 + \underline{R}^1\} \vee \{\underline{L}^1 + \overline{R}^1\} \vee \{\overline{L}^1 + \underline{R}^1\} \vee \{\overline{L}^1 + \overline{R}^1\}$) result exactly in $\overline{Q}^0$. The pair of interval bounds that provides 451 is "450+1 = 451". Hence, $\overline{L}^1 \Theta \overline{R}^1 = \overline{Q}^0$. We then compare the absolute values of individual bounds in this pair and take their maximum as the label at level $k + 1$. $\Lambda^{k+1} = \max\{\overline{L}^1, \overline{R}^1\} = \overline{L}^1 = 450$.

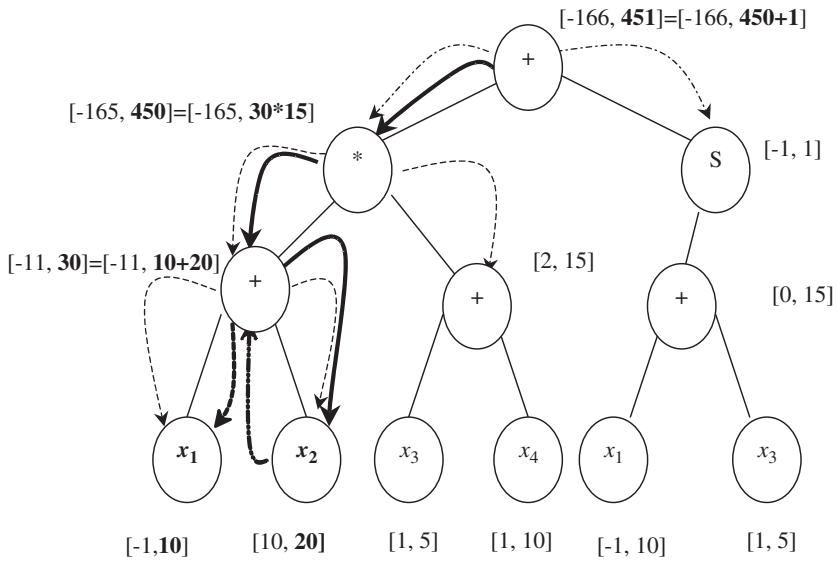

Fig. 3. Implementation of*SII_Tree* over the binary tree for "$((x_1 + x_2) * (x_3 + x_4)) + \mathrm{Sin}(x_1 + x_3) <= 0$".

Node_Type *SII (Node_Type Node)* {
if (node_level $k = 0$), $bnd = \bar{G}(\mathrm{Y})$; /* if equality $bnd = \max\{|\underline{H}_i(\mathrm{Y})|, \bar{H}_i(\mathrm{Y})\}$*/
else $bnd = \Lambda^k$;
Identify the pair a Θ b =
$\{\{\underline{L}^{k+1}\Theta\underline{R}^{k+1}\} \vee \{\underline{L}^{k+1}\Theta\bar{R}^{k+1}\} \vee \{\bar{L}^{k+1}\Theta\underline{R}^{k+1}\} \vee \{\bar{L}^{k+1}\Theta\bar{R}^{k+1}\}\}$: a Θ b = *bnd*;
$\Lambda^{k+1} = \mathrm{MAX}\{|a|, |b|\}$;
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*;
}

Fig. 4. Pseudocode for *SII* (input: node at level k; output: labeled node at level $k + 1$).

A formal description of *SII* rule is given in a pseudocode in Fig. 4.

SII_Tree applies *SII* at each level of the tree starting at the root node. The procedure is applied recursively from top to bottom, each time searching for the bound pair resulting in the labeled bound $\mathscr{L}^{k+1}$ for the upper level 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. *SII_Tree*'s pseudocode of is given in Fig. 5. Steps taken in the example are illustrated below:

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

$$\mathscr{L}^0 = \overline{Q}^0.$$

$$a\Theta b = \{(-165 + 1) or (450 + 1) or (-165 - 1) or (450 - 1)\} = \mathbf{451}.$$

Hence, $a\Theta b = \overline{L}^1 + \overline{R}^1$.

$$\mathscr{L}^1 = \max\{|\overline{L}^1|, |\overline{R}^1|\} = \max\{|450|, |1|\} = \mathbf{450} = \overline{L}^1.$$

```
Node_Type SII _Tree (Node_Type Start_Node) {
 If ((Count>2) OR (All leaves are "Closed")) exit;
 Select_Node = SII (Start_Node);     /*calls procedure SII */
 If (Select_Node. Status = "Open Node")
    Start_Node = SII_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 = SII_Tree (Next_Up(Select_Node));   /*backtrack to identify second source*/
     }
    Else Start_Node = SII_Tree (Next_Up(Select_Node)); /*backtrack to identify second source*/
 Return Start_Node;
}
```

Fig. 5. Procedure *SII _Tree*: recursive tree traversal of *SII*. (Input: root node; output: pair of source leaves-variables).

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

$$a\Theta b = \{(-11*2) or (30*2) or (-11*15) or (30*15)\} = \mathbf{450}.$$
$$\Rightarrow a\Theta b = \overline{L}^2 * \overline{R}^2$$
$$\mathscr{L}^2 = \max\{|\overline{L}^2|, |\overline{R}^2|\} = \max\{|30|, |15|\} = \mathbf{30} \Rightarrow \overline{L}^2.$$

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

$$a\Theta b = \{(-1+-10) \text{ or } (-1+20) \text{ or } (10+20) \text{ or } (10-10)\} = \mathbf{30}.$$
$$\Rightarrow a\Theta b = \overline{L}^3 + \overline{R}^3$$
$$\mathscr{L}^3 = \max\{|\overline{L}^3|, |\overline{R}^3|\} = \max\{|10|, |20|\} = \mathbf{20} \Rightarrow \overline{R}^3.$$

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

4.3. Restricting degree of variable partitioning parallelism by symbolic priority allocation

When the number of constraints is large, there might be a large set of variables resulting from the local application of *SII_Tree* to each constraint. Here, we develop a symbolic priority allocation scheme to narrow down the set of variables (selected by *SII*) to be partitioned in parallel. In this approach, all variable pairs identified by *SII_Tree* are merged into a single set *Z*. Then, a weight w_j is assigned to each variable $x_j \in Z$ and the average $\overline{w}$ calculated. The final set of variables to be partitioned is composed of all $x_j \in Z$ with $w_j > \overline{w}$.

w_j is defined as a function of several criteria: PG_Y^i (PH_Y^i) of constraint g_i for which x_j is identified as a source variable, the number of times x_j exists in g_i, total number of multiplicative terms in which x_j is involved within g_i. Furthermore, the existence of x_j in a trigonometric and/or 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 total w_j defined below.

$$w_j = \sum_{i \in IC_j} [PH_Y^i/PH_{\max} + PG_Y^i/PG_{\max} + e_{ji}/e_{j,\max} + a_{ji}/a_{j,\max} + t_{ji} + p_{ji}]/5,$$

where IC_j is the set of indeterminate constraints (over Y) where x_j is a source variable; TIC the total set of indeterminate constraints; $PH_{\max}$ the $\max_{i \in TIC}\{PH_Y^i\}$; $PG_{\max}$ the $\max_{i \in TIC}\{PG_Y^i\}$; e_{ji} the number of times x_j exists in constraint $i \in IC_j$; $e_{j,\max}$ the $\max_{i \in ICj}\{e_{ji}\}$; a_{ji} the number of multiplicative terms x_j is involved in constraint $i \in IC_j$; $a_{j,\max}$ the $\max_{i \in ICj}\{a_{ji}\}$; t_{ji} the binary parameter indicating that x_j exists in a trigonometric expression in constraint $i \in IC_j$; and p_{ji} the binary parameter indicating that x_j exists in an even power or *abs* expression in constraint $i \in IC_j$.

5. Numerical experiments

Comparison between SII and other subdivision direction selection rules: We compare the performance of the symbolic subdivision method, *SII,* 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 *SII*, only variables that are identified by *SII_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, *SII*, *Rule A* and *Smear*, with pure depth-first and pure best-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 ([28], http://www-sop.inria.fr/coprin/logiciels/ALIAS/Benches/), QUAD [11], ICOS [6], and COCOS wherever results are available [1]. 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 authors 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 [29]. ICOS is reported to be the most reliable method for the CSP among those compared by Neumaier et al. [1].

Comparison between stand-alone multi-start FSQP and IP-guided FSQP: In partitioning algorithms, a conventional technique utilized to identify roots or stationary points to a problem is to use a numerical method that has convergence guarantee if it starts at a location nearby the root. This is done because it is not realistic to expect a partitioning algorithm to split a box until it encloses all solutions within a reasonable computation time. In this comparison we run *FSQP* as a stand-alone local search method with multiple random start points in X. The aim is to show the impact of using *FSQP* under *IP*s guidance rather than as a stand-alone multi-start local search technique.

The benchmarks: Most of the benchmarks originate from the European IST project COCONUT [30] and the COPRIN web page [31]. Twenty-three test problems (some quite challenging to solve) are used in the comparison (including 7 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 1, all test problems are listed with their details (number of dimensions, nonlinear and linear equations, linear inequalities, the number of feasible solutions) 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 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 solved

Table 1
Characteristics of the test problems

Problem	(Dim, # Nonlin. Eq., # Lin. Eq., #Lin. Ineq.)	Category	Description	# of sol.	Solved by
Kin2	8,8,0,0	Quadratic	4 quadratic ladder type equations, 4 highly dependent quadratic equations	10	QUAD,ICOS, COCOS
Kin1-Modified	6,6,0,0	Trigonom.	Trigonometric, highly nonlinear, high constraint dependency	16	ALIAS
KinCox	4,4,0,0	Quadratic	Quadratic, 2 constraints independent	2	ALIAS, ICOS, COCOS
Direct Kinematics	11,11,0,0	Trigonom.	8 trigonometric, 3 quadratic high constraint dependency	2	ALIAS
Stewart-Gough	9,9,0,0	Quadratic	6 quadratic, 3 linear constraints, 3 quadratic constraints are independent	2	QUAD
FredTest	6,5,1,2	Polynom.	2 ladder type linear constraints, 1 linear covering remaining vars., up to 5th order polynomials, high constraint dependency	1	ALIAS, ICOS, COCOS
Eco9	8,7,1,0	Quadratic	Linear constraint contains all vars., ladder like quadratic constraints, constraints have many multiplicative terms	16	ALIAS, ICOS, COCOS
Redeco8	8,6,2,0	Quadratic	Similar to Eco9	8	ALIAS, ICOS, COCOS
Puma	8,7,1,0	Quadratic	4 quadratic ladder type equations, 4 highly dependent quadratic equations	16	ALIAS, ICOS, COCOS
Chemequ	5,5,0,0	Polynom.	3rd degree polynomial, high constraint dependency	4	ICOS, COCOS
Counter concurrent reactors 2	6,6,0,0	Polynom.	Polynomial of 2nd degree, each constraint has a single occurrence variable	2	ALIAS
Cyclic5	5,5,0,0	Polynom.	Geometric terms of increasing length (up to 5th degree), very high constraint dependency, 1 linear constraint including all variables	1	ICOS COCOS
Dietmaier	12,12,0,0	Quadratic	All quadratic equations, high constraint dependency	40	ALIAS
Heart	8, 6,2,0	Polynom.	Many complex multiplicative terms with higher order polynomial variables, 2 independent linear constraints impacting 4 variables	2	ICOS, COCOS
Neuro	6, 6,0,0	Polynom.	Many complex multiplicative terms with higher order polynomial variables	1	ICOS, COCOS
Quadfor2	4,3,1,0	Polynom.	Less complex multiplicative terms	1	ICOS, COCOS
Wright	5,5,0,0	Quadratic	5 constraints, each with a single quadratic variable and linear summation of all variables	5	ICOS, COCOS
Solotarev	4,4,0,0	Polynom.	Less complex multiplicative terms, mostly linear terms	6	ICOS, COCOS
S9_1	8,4,4,0	Polynom.	Mostly second order multiplicative terms	4	ICOS, COCOS
Butcher	7,7,0,0	Polynom.	Many recurrences of same variable within constraints, higher order complex multiplicative terms	7	ICOS, COCOS
Trinks	6,4,2,0	Polynom.	Simple multiplicative terms	2	ICOS, COCOS
Lorentz	4,4,0,0	Polynom.	Two-variable multiplicative terms (2 in each constraint) and an additive term' each constraint includes every variable	3	ICOS, COCOS
Remier5	6,5,0,0	Polynom.	Constraints of increasing order polynomial terms, non-multiplicative additive terms including every variable in each constraint	2	ICOS, COCOS

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 three totally independent constraints that might make constraint propagation based filtering methods (such as 2B and Box) ineffective (as verified by Lebbah et al. [11]). 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 numbers 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 [32] represents the forward kinematics equations of a parallel robot and it is quite famous in robotics for its guaranteed 40 real solutions. Heart and Neurophysics problems involve complex expressions built by multiplicative terms involving higher order polynomials. The remaining problems (last eight) have simpler forms.

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 IA library [33] and CFSQP [9]. 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.

Test results are given in terms of standard time units, STU [34] 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 STUs by taking into consideration the processing speed of machines reported.

All *Rules* are limited to run for at most to 1.31 STUs. If the theoretical number of feasible solutions are found before the time limit, then the *IP* is stopped and the CPU time taken is reported. When a method cannot find all solutions in less than 1.31 STUs, we simply indicate 1.31 as its STU. Though STU limits are reported to be higher for ALIAS, QUAD, ICOS, and COCOS, we replace them with 1.31 STU for conformity when they fail to converge.

For each problem, the multi-start *FSQP* starts with a number of solutions that is equivalent to three times the number of *FSQP* calls invoked by the *IP* configuration *SII*-adaptive tree management. The number of *FSQP* iterations is also doubled as compared to that of *IP*'s. This is because we wish to provide *FSQP* with ample time to see how many feasible solutions it identifies within the allowed CPU time limit.

Results: Table 2

provides detailed and summarized results obtained by all three subdivision selection rules and three branching strategies covering all 23 problems as well as ALIAS (nine problems), QUAD (two problems), ICOS (18 problems) and COCOS (18 problems) results. Multi-start *FSQP* results are also indicated.

Before we discuss the results in Table 2, 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 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 difficulty degrees of the benchmarks.

We provide the following information in Table 2: the STUs taken (to identify all solutions—if a run takes more than 1.31 STUs, then all solutions have not been found), the number of tree levels the solutions are found in (only for adaptive branching strategy); number of solutions found and total number of solutions; 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*). In the summarized results, we give the averages of these measures as well as the average percentage of solutions found by each method per problem with respect to total number of feasible solutions. We also provide the number of best solutions found with regard to maximum number of solutions found, number of problems that could not be solved during the given time limit (not converged) and the number of problems where ICOS, COCOS

Table 2
Numerical results

Name	Performance	Adaptive			Best first			Depth first			Multi-start FSQP	(All sol) ALIAS QUAD	(One sol) ICOS	(All sol) COCONUT
		SII rule	Rule A	Smear	SII rule	Rule A	Smear	SII rule	Rule A	Smear				
Kin2	CPU (STU)	0.218	0.847	0.151	0.934	1.310	1.310	1.310	1.310	1.133	1.310	0.160	1.310	0.0416
	No. of stages	2	2	2									(=4.0)	
	No. of solutions (Total no. of sols.)	10(10)	10(10)	10(10)	10(10)	8(10)	8(10)	5(10)	1(10)	10(10)	3(10)			
	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	107280	6577	7298	6208	6369	1196312	1139747	23062				
Kin1 Mod.	CPU (STU)	0.105	0.084	1.310	0.169	0.337	1.310	1.310	1.310	1.310	1.310	0.679	1.310	NA
	No. of stages	4	3	4									(=2.6)	
	No. of solutions (Total no. of sols.)	16(16)	16(16)	13(16)	16(16)	16(16)	8(16)	4(16)	5(16)	11(16)	1(16)			
	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)	3.000(5,3)	5.64(6,2)	5.1(6.4)				
	No. of function calls	17749	12266	121788	6907	4491	16270	402234	571317	74713				
KinCox	CPU (STU)	0.000	0.000	0.012	0.001	0.002	0.001	1.310	0.016	0.002	0.181	0.000	0.100	0.0154
	No. of stages	1	1	1										
	No. of solutions (Total no. of sols.)	2(2)	2(2)	2(2)	2(2)	2(2)	2(2)	1(2)	2(2)	2(2)	0(2)			
	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)	0.016	0.064	1.31	0.035	0.856	0.187	1.310	1.310	0.116	1.310	1.310	NA	NA
	No. of stages	1	2	3		N						=12.6		
	No. of solutions (Total no. of sols.)	2(2)	2(2)	2(2)	2(2)	2(2)	2(2)	1(2)	1(2)	2(2)	0(2)			
	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				
Stewart Gough	CPU (STU)	0.003	0.003	0.004	0.009	0.006	0.008	0.005	0.010	0.004	1.026	0.471	NA	NA
	No. of stages	1	1	1										
	No. of solutions (Total no. of sols.)	2(2)	2(2)	2(2)	2(2)	2(2)	2(2)	2(2)	2(2)	2(2)	1(2)			
	No. of SQP calls	13	13	12	13	13	12	10	21	8				

Table 2 (*contd.*)

Name	Performance	Adaptive			Best first			Depth first			Multi-start FSQP	(All sol) ALIAS QUAD	(One sol) ICOS	(All sol) COCONUT
		SII rule	Rule A	Smear	SII rule	Rule A	Smear	SII rule	Rule A	Smear				
	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				
FredTest	CPU (STU)*	0.002	0.002	0.005	0.004	0.007	0.011	0.004	1.310	0.029	0.458	1.310	1.310	0.0896
	No. of stages	1	1	1								(> 5.25)	(> 170)	
	No. of solutions (Total no. of sols.)	1(1)	1(1)	1(1)	1(1)	1(1)	1(1)	0(1)	1(1)	0(1)				
	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				
ECO9	CPU (STU)*	0.311	0.060	1.310	1.310	0.160	1.310	1.310	0.177	1.310	1.310	0.000	1.310	1.31
	No. of stages	2	2	2									(=2.3)	(=2.392)
	No. of solutions (Total no. of sols.)	16(16)	16(16)	11(16)	15(16)	16(16)	9(16)	10(16)	16(16)	11(16)	1(16)			
	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				
Redeco8	CPU (STU)	0.699	1.310	1.310	1.310	1.310	1.310	1.310	1.310	1.310	0.872	0.173	1.310	1.31
	No. of stages	2	3	3									(=3.6)	(=2.2146)
	No. of solutions (Total no. of sols.)	8(8)	7(8)	6(8)	7(8)	6(8)	5(8)	7(8)	8(8)	6(8)	8(8)			
	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				
PUMA	CPU (STU)*	0.006	0.013	1.310	0.016	0.026	1.310	0.034	0.032	1.310	1.310	0.001	0.100	0.0376
	No. of stages	1	1	2										
	No. of solutions (Total no. of sols.)	16(16)	16(16)	14(16)	16(16)	16(16)	8(16)	16(16)	16(16)	4(16)	0(16)			
	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)	0.270	1.310	0.010	1.310	1.310	0.014	1.310	1.310	0.015	1.310	NA	1.310	1.31
	No. of stages	4	5	1									(=1.6)	(=4.3266)
	No. of solutions (Total no. of sols.)	4(4)	3(4)	4(4)	1(4)	3(4)	4(4)	2(4)	1(4)	4(4)	0(4)			

	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)	0.110	0.080	1.310	0.254	0.015	1.310	0.002	0.005	1.310	1.310	0.087	NA	NA
	No. of stages	1	2	2										
	No. of solutions (Total no. of sols.)	2(2)	2(2)	0(2)	2(2)	2(2)	1(2)	2(2)	2(2)	1(2)	0(2)			
	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)	0.010	1.310	1.310	1.310	1.310	1.310	1.310	1.310	1.310	0.149	NA	1.310	1.1224
	No. of stages	2	5	3										
	No. of solutions (Total no. of sols.)	1(1)	0(1)	0(1)	0(1)	0(1)	0(1)	0(1)	0(1)	0(1)	1(1)			
	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				
Dietmaier	CPU (STU)	1.310	1.310	0.711	1.310	1.310	1.005	1.310	1.310	0.883	1.310	0.067	NA	NA
	No. of stages	3	2	2										
	No. of solutions (Total no. of sols.)	24(40)	5(40)	40(40)	24(40)	10(40)	40(40)	3(40)	1(40)	40(40)	1(40)			
	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)	1.310	1.310	1.310	1.310	1.310	1.310	1.310	1.310	0.520	1.310	NA	1.310 (> 170)	1.31 (=3.285)
	No. of stages	3	3	3										
	No. of solutions (Total no. of sols.)	1(2)	0(2)	1(2)	1(2)	0(2)	1(2)	1(2)	0(2)	2(2)	0(2)			
	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)	0.074	0.015	0.153	0.081	0.006	1.241	0.113	0.010	0.273	0.026	NA	1.310 (=4.40)	0.0722
	No. of stages	1	1	2										
	No. of solutions (Total no. of sols.)	1(1)	1(1)	1(1)	1(1)	1(1)	1(1)	1(1)	1(1)	1(1)	1(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				

Table 2 (*contd.*)

Name	Performance	Adaptive			Best first			Depth first			Multi-start FSQP	(All sol) ALIAS QUAD	(One sol) ICOS	(All sol) COCONUT
		SII rule	Rule A	Smear	SII rule	Rule A	Smear	SII rule	Rule A	Smear				
Butcher	CPU (STU)	0.234	0.071	0.459	0.083	0.044	1.310	1.310	1.31	0.133	1.310	NA	1.310	1.31
	No. of stages	2	1	2									(> 170)	(=3.0134)
	No. of solutions (Total no. of sols.)	7(7)	7(7)	7(7)	7(7)	7(7)	0(7)	3(7)	1(7)	7(7)	0(7)			
	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				
Lorentz	CPU (STU)	0.012	0.008	1.310	0.007	0.005	1.310	0.021	1.310	1.310	1.310	NA	1.310	0.2054
	No. of stages	2	2	3									(> 170)	
	No. of solutions (Total no. of sols.)	3(3)	3(3)	2(3)	3(3)	3(3)	2(3)	3(3)	2(3)	2(3)	1(3)			
	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)	0.012	0.013	0.001	0.012	0.013	0.001	1.310	1.310	0.001	0.113	NA	1.310	0.5808
	No. of stages	1	1	1									(> 170)	
	No. of solutions (Total no. of sols.)	1(1)	1(1)	1(1)	1(1)	1(1)	1(1)	0(1)	0(1)	1(1)	1(1)			
	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)	0.089	0.611	1.310	1.310	1.310	1.310	1.310	1.310	0.136	1.310	NA	1.310	1.31
	No. of stages	2	3	4									(> 170)	(=2.2148)
	No. of solutions (Total no. of sols.)	2(2)	2(2)	1(2)	1(2)	1(2)	1(2)	1(2)	1(2)	2(2)	0(2)			
	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)	0.109	0.126	1.310	1.310	0.118	1.310	1.310	1.310	0.130	1.310	NA	1.310	1.31
	No. of stages	1	1	3									(=2.3)	(=1.5854)
	No. of solutions (Total no. of sols.)	4(4)	4(4)	3(4)	3(4)	4(4)	3(4)	3(4)	3(4)	3(4)	1(4)			
	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				

Solotarev	CPU (STU)	0.007	0.035	0.055	0.010	0.007	1.310	1.310	1.31	0.052	1.310	NA	1.900	0.0462
	No. of stages	2	3	4										
	No. of solutions (Total no. of sols.)	6(6)	6(6)	6(6)	6(6)	6(6)	5(6)	4(6)	5(6)	6(6)	1(6)			
	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)	0.005	0.732	0.053	0.006	1.310	0.003	0.003	0.004	0.056	0.090	NA	1.310	0.2142
	No. of stages	1	4	1									(> 170)	
	No. of solutions (Total no. of sols.)	2(2)	2(2)	2(2)	2(2)	1(2)	2(2)	2(2)	2(2)	2(2)	2(2)			
	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	0.315	NA	0.100	1.31
	No. of stages	1	1	1										(=2.185)
	No. of solutions (Total no. of sols.)	5(5)	5(5)	5(5)	5(5)	5(5)	5(5)	5(5)	5(5)	5(5)	5(5)			
	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				

Summary statistics

	Adaptive			Best first			Depth first			Multi-start FSQP	ALIAS QUAD	ICOS	COCOS
	SII	Rule A	Smear	SII	Rule A	Smear	SII	Rule A	Smear				
Avg. CPU (STU)	0.21	0.41	0.70	0.53	0.53	0.85	0.86	0.87	0.60	0.94	0.39	1.15	0.72
Avg. no. of stages	1.78	2.17	2.22	–	–	–	–	–	–	–	–	–	–
Avg. no. of SQP calls	858.22	3093.09	3772.17	356.91	369.52	595.74	7412.17	9254.43	2538.35	–	–	–	–
Avg. % of sol. found	96.09	85.87	78.44	84.40	80.65	72.77	63.78	56.87	83.88	32.94	–	–	–
Avg. no. of function calls	16356.09	38218.39	52959.26	4535.22	4417.13	6662.04	603921.39	755506.30	30379.74	–	–	–	–
No. of problems not covered	2	5	11	8	8	13	15	15	8	17	2	7	8
No. of problems with max. sol found	21	18	12	15	15	10	8	8	15	6	–	–	–
Date not available	0	0	0	0	0	0	0	0	0	0	12	14	5
	Avg. percentage of solutions found												
Stage 1	72.50	64.55	61.16										
Stage 2	17.39	14.62	14.20										
Stage 3	4.57	4.53	0.82										
Stage 4	1.63	2.17	2.26										
Stage 5	–	0.00	–										

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 for 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 slow 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 *SII*, *Rule A* and *Smear* are compared under adaptive tree management scheme *SII* 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 50% less time than *Rule A* in solving CSPs. *SII* 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 *SII*, it tends to need lesser or similar STUs 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 *SII* whereas it is 4,849,203 under *Rule A*, but *SII* takes 0.699 STUs to converge while *Rule A* takes more than 1.31 STUs. In general, the difficult problems for all rules under adaptive tree management scheme are Redeco8 (except for *SII*), Chemeq, Cyclic5 (except for *SII*), 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 best-first branching scheme, *Rule A* and *SII* 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 *SII* and *Rule A* as compared with the adaptive and best-first approaches. When we compare the three branching strategies, the adaptive approach is observed to produce the best overall results with *SII* and *Rule A*. One should also note that *SII*-adaptive approach manages to find 96% of all feasible solutions within the given time limit and this is followed by *Rule A*-adaptive approach.

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, *SII* 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 2) 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 *SII* 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, *SII* converges in 0.109 STUs with average parallelism scale of 7.8 whereas *Rule A* converges in 0.126 STUs with a parallelism scale of 4.28, and *Smear* cannot converge within 1.31 STU limit with a scale of 5.0. 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. A good example of such self-sustained parallelism is Direct Kinematics where both *SII* 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 among ALIAS/QUAD and ICOS/ COCOS, one should remember that in the nine 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 STUs 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 19 problems that it attempts, it is able to converge only in three problems before the STU limit is reached whereas COCOS can converge in 10 problems. 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 average STUs, 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.

In the comparison between *IP* + *FSQP* and multi-start *FSQP* we observe that the multi-start *FSQP* is given sufficient CPU time since it takes more time on the average than the slowest *IP* configuration. Yet, the average percentage of solutions found is very inferior to that of other methods. It does not converge in 17 out of 23 problems. This is expected because incomplete solvers have little chance of success when used for this objective.

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 *SII* and *Rule A* whereas Redeco8 and Dietmaier are serious obstacles for the *Rules*. Redeco8 is also a problem for ICOS/COCOS. In Redeco8, only *SII* is able to identify all solutions among the *Rules*, but it takes longer time than ALIAS. Multi-start *FSQP* is also able to solve this problem. In Kin2 that has appropriate constraint structure for filtering methods, *SII* 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 CSPs, 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 STUs at 1/1000 level in almost all branching schemes. COCOS converges in this problem but it is slower than the *Rules*. Cyclic5 can be solved by *SII* 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 and faster than the latter by multi-start *FSQP*. 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 *SII* (under adaptive branching approach), though the latter is much slower. In Reactor 2, *Rule A* and *SII* are compatible with and better than (in best-first and depth-first approaches) ALIAS in terms of STUs.

In the last 10 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 best-first tree management strategies perform better than ICOS/COCOS. In the more difficult benchmarks such as reimer5 and S9_1, *SII* and *Rule A* outperform COCOS only with the adaptive tree management approach but not with the other two. Out of these 10 problems, multi-start *FSQP* can solve the easier problems (Neuro, Quadfor2, Trinks and Wright).

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 *SII* and adaptive tree management that invokes *FSQP* is a viable approach that can be adopted in CSPs. Testing performance of *IP*-*SII* 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, *SII*, outperforms *Rule A* and *Smear* in the adaptive tree management approach that produces the best overall results when compared with best-first and depth-first strategies. COPRIN group explain 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 an a priori number. Similar parallel bisection methods are also available in unconstrained 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.

6. Conclusion and future work

A generic collaborative methodology that integrates basic IP with local search (feasible sequential quadratic programming—-*FSQP*) is developed here to solve the CSP. The implementation is illustrated on some moderate and difficult CSP benchmarks.

The proposed *IP* is enhanced by a symbolic interval inference (*SII*) method that is used to organize the sequence of variables to be partitioned. A simple variable bisection parallelization scheme is also introduced for established

partitioning rules and *SII*. *SII* improves convergence in *IP* because it selects variables to be subdivided by targeting a reduction in constraint uncertainty degrees.

The call for *FSQP* in *IP* is organized by an adaptive tree management system developed here. The latter system involves a stage-wise search tree where after the box to be partitioned is selected, the search operates only within the sub-tree under the selected parent box until it is decided that the infeasible area discarded does not improve in a number of consecutive iterations. At that point, all boxes under the sub-tree are sent as a batch to *FSQP*. The search then goes back to unexplored boxes in the current stage. In this sense, the depth of the sub-tree is bounded by area removal performance criterion and its width is restricted by the number of parallel boxes in the sub-tree that grows under the selected box.

Implementation on a number of CSP benchmarks shows that the collaborative method is able to converge fast in many instances without using sophisticated filtering tools. The proposed subdivision rule *SII* is able to economize on *FSQP* calls and thus improves *IP*s performance significantly as compared to other subdivision direction selection rules. Furthermore, the adaptive tree management strategy outperforms best-first and depth-first strategies. Comparisons are made among all rules embedded in the collaborative *IP* and among all mentioned tree management approaches.

The proposed approach is also applicable in solving both unconstrained and constrained optimization problems. Such an extension would turn our current approach into a generic solver. This work is currently in progress. In the future, we also wish to embed local and global filtering methods in our solver by adapting an efficient constraint representation tool, the directed acyclic graph (DAG), proposed by Schichl and Neumaier [35]. The latter will also enable us to perform more intensive testing of our approach.

Acknowledgments

We wish to thank Professor Andre Tits (Electrical Engineering and the Institute for Systems Research, University of Maryland, USA) for providing the source code of CFSQP. Also, we wish to thank Dr. Tamas Vinko (Research Group on artificial intelligence of Hungarian Academy of Sciences and University of Szeged) for providing the COCONUT results.

References

[1] Neumaier A, Shcherbina O, Huyer W, Vinko T. A comparison of complete global optimization solvers. Mathematical Programming 2005;103: 335–56.
[2] Alefeld G, Herzberger J. Introduction to interval computations. New York, USA: Academic Press Inc.; 1983.
[3] Neumaier A. Interval methods for systems of equations encyclopedia of mathematics and its applications Cambridge: Cambridge University Press; vol. 37, 1990.
[4] Hansen E. Global optimization using interval analysis. New York: Marcel Dekker Inc; 1992.
[5] Ratschek H, Rokne J. Interval methods. In: Horst R, Pardalos PM, editors. Handbook of global optimization. Netherlands: Kluwer Academic Publisher; 1995. p. 751–828.
[6] Lebbah Y. ICOS (interval constraints solver). WWW-document, 2003. ⟨http://www-sop.inria.fr/coprin/ylebbah/icos⟩.
[7] Panier ER, Tits AL. On combining feasibility, descent and superlinear convergence in inequality constrained optimization. Mathematical Programming 1993;59:261–76.
[8] Zhou JL, Tits AL. An SQP algorithm for finely discretized continuous minimax problems and other minimax problems with many objective functions. SIAM Journal on Optimization 1996;6:461–87.
[9] Lawrence CT, Zhou JL, Tits AL. User's guide for CFSQP version 2.5: a code for solving (large scale) constrained nonlinear (minimax) optimization problems, generating iterates satisfying all inequality constraints. Institute for Systems Research, University of Maryland, College Park, MD, 1997.
[10] Yamamura K, Kawata H, Tokue A. Interval solution of nonlinear equations using linear programming. BIT Numerical Mathematics 1998;38: 186–99.
[11] Lebbah Y, Rueher M, Michel C. A global filtering algorithm for handling systems of quadratic equations and inequalities. CP2003, Lecture notes on computer science, vol. 2470. 2003. p. 109–213.
[12] Stevenson D. IEEE standard for binary floating point arithmetic. IEEE/ANSI 754-1985, Floating Point Working Group, Microprocessor Standards Sub-Committee, 1985.
[13] Goldberg DE. Genetic algorithms in search, optimization and machine learning. Reading, MA: Addison-Wesley; 1989.
[14] Jeavons P, Cohen D, Cooper M. Constraint, consistency and closure. Artificial Intelligence 1998;101:251–65.
[15] Moore RE. Interval analysis. Englewood Cliffs, NJ: Prentice-Hall; 1966.
[16] Hickey TJ. CLIP: A CLP intervals dialect for metalevel constraint solving. Proceedings of PADL '2000 international workshop on practical aspects of declarative languages, Lecture notes on computer science, vol. 1753. Boston: Springer; 2000. p. 200–14.

[17] Ceberio M, Granvilliers L. Solving nonlinear equations by abstraction, Gaussian elimination, and interval methods. In: Alessandro A, editor. Proceedings of frontiers of combining systems, fourth international workshop, FroCoS 2002. Lecture notes on artificial intelligence, vol. 2309. Italy: Springer; 2002. p. 117–31.
[18] Buchberger B. Grobner bases: an algorithmic method in polynomial ideal theory. In: Bose NK, editor. Mutidimensional systems theory. Dordrecht: D. Reidel Publishing Co; 1985. p. 184–232.
[19] Rump SM. On the solution of interval linear systems. Computing 1992;47:337–53.
[20] Chabert G, Trombettoni G, Neveu B. Box-set consistency for interval based constraint problems. Symposium on applied computing SAC 2005. In: Proceedings of the 2005 ACM symposium on applied computing. New York, USA: ACM Press; 2005. p. 1439–43.
[21] Kearfott RB, Manuel NIII. INTBIS: a portable interval Newton/bisection package. ACM Transactions on Mathematical Software 1990;16: 152–7.
[22] Korf RE. Depth-first iterative deepening: an optimal admissible tree search. Artificial Intelligence 1985;27:97–109.
[23] Benhamou F, McAllester D, Van Hentenryck P. CLP (intervals) revisited. Proceedings of ILPS'94, international logic programming symposium, 1994. p. 124–38.
[24] Kearfott RB. Decomposition of arithmetic expressions to improve the behaviour of interval iteration for nonlinear systems. Computing 1991;47:169–91.
[25] Smith EMB, Pantelides CC. Symbolic reformulation/spatial branch and bound algorithm for the global optimization of nonconvex MINLPs. Computers and Chemical Engineering 2001;25:1399–401.
[26] Sahinidis NV. Global optimization and constraint satisfaction: the branch and reduce approach. Leture notes on computer science, vol. 2861. 2003. p. 1–16.
[27] Tawarmalani M, Sahinidis NV. Global optimization of mixed-integer nonlinear programs: a theoretical and computational study. Mathematical Programming 2004;99:563–91.
[28] Merlet J-P. ALIAS: an interval analysis based library for solving and analyzing system of equations. Séminaire Systèmes et équations algébriques, Toulouse, Automation, 2000. p. 1964–1969.
[29] Van-Hentenryck P, Michel L, Deville Y. Numerica: a modeling language for global optimization. London, England: MIT Press; 1997.
[30] COCONUT ⟨ http://www.mat.univie.ac.at/~neum/glopt/coconut/benchmark.html⟩, 2005.
[31] COPRIN ⟨http://www-sop.inria.fr/coprin/logiciels/ALIAS/Benches/⟩, 2004.
[32] Dietmaier P. The Stewart–Gough platform of general geometry can have 40 real postures. Proceedings of ARK, Strobl, Austria; 1998. p. 7–16.
[33] Knuppel O. PROFIL/BIAS—a fast interval library. Computing 1994;53:277–87.
[34] Shcherbina O, Neumaier A, Sam-Haroud D, Vu X-H, Nguyen T-V. 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; 2002.
[35] Schichl H, Neumaier A. Interval analysis on directed acyclic graphs for global optimization. Institut for Mathematik; Universitat Wien, Austria: 2003.