A complete nonlinear system solver using affine arithmetic

Ali Baharev and Endre Rév *

Budapest University of Technology and Economics
Department of Chemical and Environmental Process Engineering
1521 Budapest, Pf. 91, Hungary
ali.baharev@gmail.com
http://reliablecomputing.eu

Abstract. A general framework is presented for finding all solutions to systems of nonlinear equations, or proving that there is no solution to the problem. Components of this framework are interval arithmetic, affine arithmetic, constraint propagation based on directed acyclic graph (DAG) representation of the problem, a pruning technique based on linear programming and a local approximate optimizer. The proposed method is implemented in C++ and will be released as an open source standalone solver. An important feature of the solver is that it is interfaced with the AMPL® modeling system. Numerical results on difficult benchmarks and comparisons to other solvers are given. The favorable effect of exploiting common subexpressions is also discussed. Real-world problems from chemical engineering, having multiple solutions, are successfully solved.

1 Introduction

Solving systems of nonlinear equations constitute an important part of the everyday practice in chemical engineering. Interval analysis has been applied to a wide variety of problems in chemical engineering, an overview of the pioneering work of Stadtherr and colleagues are given in [38]. Reliably computing multiple steady states of distillation columns is crucial to their design, simulation, control, and operation. Computation of these multiple steady states requires solving large-scale systems of nonlinear equations. Even professional simulators dedicated to these problems return a single solution at a time without indicating possible existence of other solutions [23, 57]. Consequently, finding multiple solutions / proving uniqueness of a solution requires making a troublesome case

* The authors are grateful to the unknown reviewer, Yahia Lebbah, Jean-Pierre Merlet, Gilles Trombettoni, Ignacio Araya, Arnold Neumaier, Ralph Beaker Kearfott, Lubomir Kolev, Andrew Makhorin, Stefan Vigerske, Andreas Waechter, Peter Spellucci, Renata Silva, Luis Nunes Vicente, Iain Duff and John K. Reid (packages MA27 and MC19) for their help, valuable comments and suggestions. The research was funded by the Hungarian Scientific Research Foundation (OTKA); Grant Number: K062099.

study. Continuation methods seem to be the only general methods capable of finding the above multiple solutions to those large-scale systems of equations [34, 35, 57]. The goal of the authors' research is to develop such an alternative to continuation methods that is guaranteed to find all solutions and can efficiently handle the above-mentioned large-scale systems of nonlinear equations.

Several general purpose solvers exist that provide completeness; the reader is referred to the survey of Neumaier [44] for an exhaustive overview. The paper of Belotti *et al.* [8] gives a thorough review of the latest results achieved since the paper of Neumaier. Those components of the authors' solver that are considered to be new and / or different compared to the other solvers are the linearization technique based on the mixed affine arithmetic and interval arithmetic model [30, 54], the efficient implementation of the pruning technique based on linear programming, and the local search heuristic.

2 Implementation of the proposed method

Outline of the implementation is given in *Figure 1* on the following page. The user processes the model of the problem with AMPL [12]. The output of the AMPL, a plain ASCII text file, is passed to the solver. Roughly speaking, this text file contains the directed acyclic graph (DAG) representation of the problem. Function Main coordinates the iterative search procedure. Two DAGs are built using the DAG template library in function Main: one using the interval data type of the interval arithmetic library C-XSC [17] (interval DAG), and another one using the affine data type of the affine library (affine DAG). The DAG template library and the affine library are implemented in C++ from scratch by the first author.

The interval DAG is used for constraint propagation as described in *Section 5* of the paper of Schichl and Neumaier [49]. Forward and backward propagation are repeated as long as the box is 'sufficiently' reduced. According to the authors' best knowledge, this kind of propagation was first proposed by Kearfott [25]. The powerful consistency technique "HC4 is in essence Kearfott's [25] Algorithm 3.1 with particular ordering" [15].

The affine DAG is used for evaluating the equations. This evaluation yields a linear enclosure since affine arithmetic is a linearization technique as well. This linearized system is used for pruning: it is made hull consistent [9] with respect to the current box by applying LP pruning as discussed in subection 3.3. The heuristic of Tobias Achterberg makes this pruning efficient. The LP solver GNU GLPK [14] is used through its C interface.

If the box cannot be 'sufficiently' reduced further, the local optimizer IPOPT [60] is started from the midpoint of the box. The local optimizer either finds a solution or returns a local minimizer for the constraint violation (L^1 norm). Solutions are displayed to the user as soon as they are found. IPOPT is used through its C++ interface; the `AmplTNLP` class provides almost all that the implementation needs.

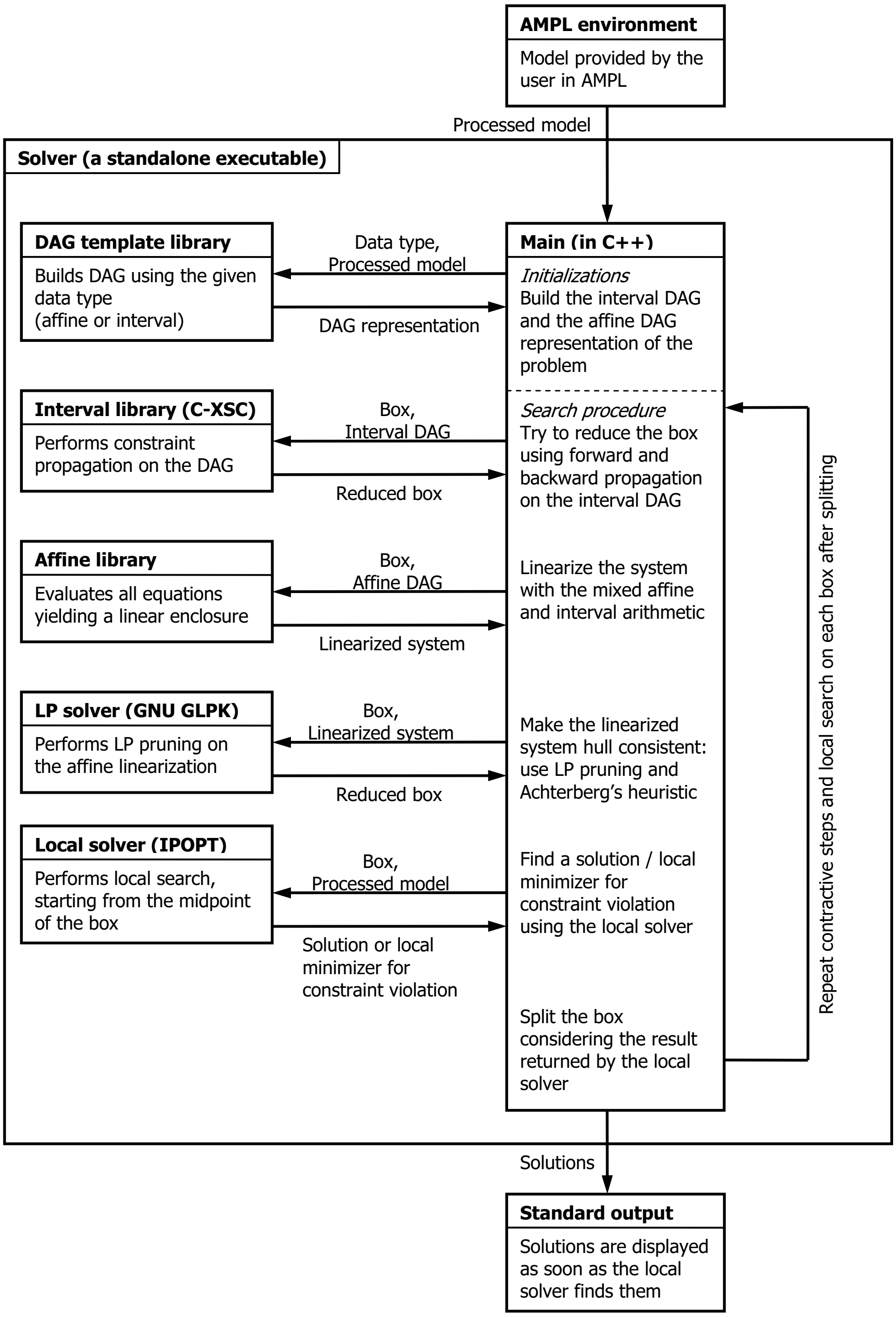

Fig. 1. Outline of the implementation.

If the width of the box is above a certain threshold, the box is split into smaller parts after the local search. A simple splitting rule is proposed at section Local search heuristic, which takes into consideration the result of the local search. All of the above procedures (starting with constraint propagation on the interval DAG) are repeated on each smaller boxes iteratively. The search terminates with a list of (not verified) solutions.

3 Characteristic components

3.1 The AMPL modeling system as interface

The output of the AMPL modeling system is basically a DAG representation of the user's model, in a custom format. The AMPL/Solver Interface Library (ASL, [13]) provides routines to handle this output, which makes it relatively easy to attach (local) solvers to AMPL. Quite unfortunately, the ASL uses the `double` data type. A complete rewrite is needed to make ASL work with abstract data types like the affine data type. The first author implemented the necessary C++ function templates to build the corresponding DAG of arbitrary type from the output of the AMPL. The DAG class templates implement the evaluation of the equations and the forward-backward propagation. These templates are referred to as the DAG template library in *Figure 1*. Simplification / transformation of the DAG is a feature to be implemented later.

The COCONUT environment [44, 47, 50] also provides tools to read the output of AMPL and to convert it into many other formats, *e.g.* C source, GAMS, LGO and GlobSol format (latter in `FORTRAN 90`), through the custom DAG file format of COCONUT. The COCONUT environment also implements evaluation of, constraint propagation on, and transformation of DAGs.

3.2 Linearization technique / linear relaxation

Numerical evidence published in the literature, *e.g.* [29, 30, 42, 43, 62], seem to indicate superiority of the linear enclosure

$$\boldsymbol{L}(x) = Ax + \boldsymbol{b}, \qquad x \in \boldsymbol{X},\ A \text{ is a } \textit{real} \text{ matrix}, \tag{1}$$

proposed by Kolev in a number of publications, compared to the traditional one

$$\boldsymbol{L}(x) = \boldsymbol{A}(x - z) + f(z), \qquad x \in \boldsymbol{X},\ \boldsymbol{A} \text{ is an } \textit{interval} \text{ matrix}, \tag{2}$$

such as the interval Newton method. The former has the following advantages over the latter when applied to solving systems of nonlinear equations.

1. The solution set of $Ax+\boldsymbol{b}=0$ has a much simpler form, the hull solution is straightforward: $\boldsymbol{X} \cap A^{-1}\boldsymbol{b}$.
2. Linear programming is directly applicable to prune the current box, as the enclosure is convex.

3. Keeping track of the correlation between the computed partial results and the original variables is automatically achieved during the computation of (1) with affine arithmetic, yielding tighter enclosures.

There is no significant difference between (1) and (2) with respect to the computational efforts [4, 5]. The authors empirical comparison [4, 5] of the linearization techniques (1) and (2) is in good agreement with the above cited results [29, 30, 42, 43, 62]. The properties and the implementation of linearization technique (1) are further discussed in [3].

3.3 Efficient implementation of the LP pruning

The LP pruning makes the linearized system hull consistent with respect to the current box by solving the following LP subproblems for all i-s:

$$\begin{aligned}
&\min/\max\ x_i \\
&\text{subject to} \\
&\qquad b_L \le Ax \le b_U \\
&\qquad x_L \le x \le x_U .
\end{aligned} \tag{3}$$

In (3), the constraints are directly yielded by the evaluation of the original nonlinear function with the mixed affine and interval arithmetic model, and the bounds of the box. Application of this or similar variants of LP pruning has been proposed by many authors, *e.g.* [8, 24, 32, 33, 37, 39, 40, 61, 63].

It seems as if there were $2n$ LP subproblems to be solved (where n denotes the number of variables). This is not the case. The minimization / maximization subproblem i is skipped if x_i equals its lower / upper bound, respectively, in any of the primal feasible solution vectors obtained during the pruning procedure. The gain is evident.

Manipulating the objective does not influence primal feasibility, therefore only the first LP subproblem has to be solved from scratch. All the other subproblems use the optimal solution of the preceding subproblem as the initial primal feasible basis and skip Phase I of the primal simplex algorithm.

The number of simplex iterations in Phase II is reduced by choosing the sequence of subproblems as discussed below. The naïve sequence [4] to process the LP subproblems would be min x_1, max x_1, min x_2, max x_2, ... *etc.* The subproblems min x_1 and max x_1, however, are likely to produce completely different solutions. This can result in a lot of simplex iterations when using the optimal solution of subproblem min x_1 as the initial basis for subproblem max x_1.

The following simple heuristic is used for selecting the next subproblem. Find that variable which is the closest to its lower / upper bound and the corresponding subproblem, *i.e.* min x_i if x_i is close to its lower bound or max x_i if x_i is close to its upper bound, has not yet been considered in the pruning step. This heuristic is based on the assumption that the current basis is not far from the optimal solution of the chosen subproblem. Numerical evidence [3] shows that this is a reasonable assumption in practice. This is referred to as Achterberg's heuristic.

3.4 Local search

Local search is integrated into the solver to test two expectations.

1. The local solver is expected to find solutions in a far earlier stage of the iteration (*i.e.* in wide boxes) than the interval solver itself would find them.
2. The found solutions may help to find the component to split.

3.4.1. Requirements. A plain implementation of a local nonlinear system solver, *e.g.* direct application of Newton's method for nonlinear equations with unit step lengths, is inadequate to study the above expectations. The following requisites are the key to success.

1. Given an appropriately chosen merit function (*e.g.* L^1 or L^2 norm of constraint violation), the local solver should (almost always) converge to a solution or to a local minimizer to constraint violation *in the current box*. In other words, it has to tolerate poor initial points because the midpoint of the current box is chosen as initial point in every iteration. Both line search and trust-region methods meet this requirement [45].
2. In connection with the previous point, bound constraints on the variables have to be handled by the local solver. This ensures that only the current box is searched, prevents the method from 'jumping out' of the domain of interest and / or converging to solutions that are not physically meaningful.
3. A robust scaling method has to be implemented in the local solver. Many practical problems are badly scaled. Even problems that are globally well-scaled can become badly scaled in certain boxes, *i.e.* they can become badly scaled locally.

All of the above requirements are satisfied by the restoration phase [59, 60] in IPOPT. As it is shown in subsection 4.4, IPOPT is able to converge to the so-called unstable steady state solution of `homoazeo40` from a poor starting point. This solutions is missed by state-of-the-art methods dedicated to this problem even when supplied with a good starting point [23]. This fact corroborate that above properties indeed are the key to success.

3.4.2. Discovering solutions in early stages. The trivial benefit of getting solutions quickly is that they can be analyzed while the usually time consuming computations are still in progress. Even if the interval solver fails to deliver any result in acceptable time because the problem is too complicated, the local solver may still find solutions. This is exactly the case with benchmark `homoazeo40` to be discussed in subsection 4.4.

If the search procedure is abandoned due to time limitation, the proposed method degrades to a 'smart' multistart method. It is smart because the interval solver acts as an accelerator by discarding parts of the box that cannot contain any solution. As a consequence, the local solver has to search just in those parts that may contain solutions. The boxes become smaller and smaller during the branch and prune procedure and the likelihood of finding a solution increases. One may go even further, and perform local search in every box from several initial points generated by an appropriate heuristic algorithm. If the search was not abandoned, the method would find all solutions.

The sooner the solutions are obtained, the more likely they can be exploited in the splitting step. This is discussed further in the next subsection.

3.4.3. Splitting heuristic based on local search. Several authors have proposed incorporating some sort of local search method to accelerate the overall efficiency of interval methods, *e.g.* [11, 20, 22, 26, 28, 31, 48]. A particular proposal (back-boxing) is the following. If a solution found by the local approximate solver can be verified unique after appropriate inflation, then it is cut out from the box by splitting the box into (at most) $2n$ subboxes, where n denotes the number of variables. One of the resulting boxes contains the verified solution exclusively.

The authors have no doubts that this is a reasonable approach, especially with respect to the parametric slopes as proposed by Kolev [31]. However, forming $2n$ new boxes does not seem appealing in case of those problems that are of interest to the authors. The real-world problem `extr22`, also computed in [3], has 418 variables. Suppose the local solver finds (the otherwise unique) solution before the very first splitting and this solution can be verified. Then, splitting according to the above proposal would result in $2 \times 418 = 836$ boxes, and these boxes are likely to be split further. However, this problem can be solved by examining no more than 3 boxes.

After this observation, another splitting rule is tried here. If there is no known solution in the current box then apply bisection. If the box is known to contain a solution then split the widest component of the box into two, and assure that the solution is 'not close' to the bounds of the new box not containing the solution. In other words, try to avoid creating a new box that is (expected to be) difficult to discard because of the nearby solution. The 'not close' is quantified as follows. Let w denote the width of the component to be split, and let x_S denote the corresponding component of the solution. Compute the following two points $x_S - pw$ and $x_S + pw$ with an appropriate positive parameter p (*e.g.* $p = 0.20$), and split the component at the one closer to the midpoint of the component.

There are many straightforward alterations to this very simple splitting rule, *e.g.* the component to be split may be chosen by other measures than the width (for example MaxSmear [6]); split the component into three if x_S is sufficiently close to the midpoint of the component; separate found solutions by splitting; consider local minimizers for the constraint violation as well if the maximal constraint violation is smaller than a pre-defined threshold ('almost solutions') etc. A major issue with local minimizers to constraint violation is that they are likely to change with the norm used and / or with scaling. Nevertheless, they may indicate that a particular region in the box is difficult to discard because it contains an 'almost solution.' Linearization techniques of type $\boldsymbol{A}(x - z) + f(z)$, using the point of expansion z when computing the linear enclosure, may benefit from local minimizers for constraint violation [18] (11.4 An inner iteration, pp. 244–246).

4 Numerical results

All the difficult benchmarks from [37] are taken on which `QuadSolver` outperformed the other solvers because the other solvers either failed to solve the problem within the time limit or showed poor performance compared to `QuadSolver`, and the system of equations can be evaluated with the functions implemented so far. (*E.g.* evaluation of x^3 with affine arithmetic is not yet implemented). Those benchmarks that have an `_rev` suffix, which stands for revised, are a transformed form of the corresponding benchmark. The applied transformation is discussed in subsection 4.3 *Exploiting common subexpressions.* The problem `eco9` is taken from the COCONUT benchmarks [10].

`Jacobsen91`, `homoazeo20`, `homoazeo40`, `extr22`, `extr30`, and `LLE3` are new benchmarks proposed by the authors and are made publicly available on the website of the first author: http://reliablecomputing.eu. Reliably computing multiple steady states of distillation columns is crucial to their design, simulation, control, and operation. Four of the five solutions to `Jacobsen91` correspond to the steady states of a distillation column (ideal two-product distillation) [21]; the fifth solution is infeasible in practice. The `homoazeo40` has three solutions; they correspond to the steady states of a distillation column (homogeneous azeotropic distillation, ternary system). The `homoazeo20` has one solution and differs from `homoazeo40` in the `N`, `N_F` and `D` values (number of stages, feed stage location, distillation flow rate, respectively); `D` can be chosen as bifurcation parameter. The `extr22` and `extr30` are the models of the extractive distillation column computed in [3] with 22 and 30 theoretical stages, respectively.

The benchmark `LLE3` contains the necessary condition for liquid-liquid equilibrium of a ternary mixture discussed as Example 2 in [41] and Problem 1 in [55]. In these publications, the necessary and sufficient conditions forming a global optimization problem are solved, this formulation is superior to `LLE3`. The liquid phase split (ternary mixture) problem in [3–5] is similar to `LLE3`, but with different parameters and / or different formulation.

The proposed method is guaranteed to find all solutions assuming exact arithmetic. The current implementation does not use directed rounding for various reasons. This can results in loosing solutions due to the rounding errors but there is no evidence of lost solutions in the current study. All computations have been carried out on an Intel Pentium 4 machine at 2.6 GHz, running Kubuntu Linux 8.04 (kernel 2.6.24-23-generic), gcc 4.2.4 with flag -O3 (target: i486-linux-gnu).

4.1 The effect of local search

Table 1 shows that the local solver finds many / all of the solutions before the interval method itself would find the first solution. This confirms the expectations discussed in 3.4.2. *Discovering solutions in early stages.*

Table 1. Comparing the number of boxes examined before finding the first solution with and without local search. The local solver finds many / all of the solutions before the interval method itself would find the first solution.

	First solution found by the local solver in box	First solution found by the interval solver in box	Solutions found by local search before the first sol. of the interval solver
LLE3	62	451	1/1
didrit	1	133	4/4
jacobsen91	1	33	5/5
kinema	1	246	8/8
yamamura196	1	14	2/2
eco9	1	11190	13/16
camera1s	2	235793	16/16
camera1s_rev	1	811	12/16
geneig[1]	2	51981	10/10
geneig_rev[1]	1	3597	10/10
lee	29	72501	2/2
lee_rev	3	374	2/2
katsura7	1	2065	38/44
stewgou40	1	44348	40/40
homoazeo20[2]	3	124	1/1
homoazeo40[2]	1	-	3/3[3]
extr22[2]	1	2	1/1
extr30[2]	1	5	1/1

The results of the local search are ignored in the splitting step.
There is no evidence of lost solutions (due to rounding errors).
(1) With initial intervals [-10.0, 10.0] due to lack of information.
(2) Problem specific splitting rules are applied.
(3) Completing the search is not possible yet in reasonable time.

The bottleneck of the algorithm in the previous implementation [3] was the LP pruning procedure. It was assumed that incorporating a local solver would not change that. As it is shown in *Table 2*, this assumption does not hold in case of the studied examples with the current implementation. The entire implementation of the proposed method uses sparse data structures, including the local solver IPOPT. The studied examples are small and their Jacobian is dense, whereas IPOPT is meant for solving large and sparse NLP problems. As a result, this is not surprising that the local search considerably slowed down the procedure.

To back up this guess, some tests have been carried out. The subroutine `DMNHB` from the `PORT` Mathematical Subroutine Library [46] satisfies all the requirements listed in subsection 3.4: it uses a model/trust technique, handles variable bounds and has a built-in scaling algorithm. All the subroutines in the `PORT` library are meant to be used for small and dense problems. The benchmarks `eco9`, `lee_rev` and `geneig_rev` were involved in the test, on which IPOPT imposed a significant overhead. They are solved with less than 10% overhead if `DMNHB` is used as a local solver. The issue with DMNHB is that the model, at the moment, has to be manually transformed to an optimization problem (minimizing the L^2 norm) in the AMPL modeling language. In contrast, IPOPT minimizes the L^1 norm automatically. Switching to other local solvers (*e.g.* DONLP2 [52, 53] uses dense linear algebra; ipfilter [51, 56]) is an option to be investigated.

Table 2. Comparing computational efforts without and with local search. The table also shows the favorable effect of exploiting common subexpression (benchmarks with the _rev suffix); n denotes the number of variables.

			Without local search			With local search	
	n	Sols[a]	Number of boxes not discarded	Splits	Time (s)	Solutions found by the local solver	Time (s)
LLE3	49	1	4	261	21	1	31
didrit	9	4	4	232	2	4	8
jacobsen91	19	5	5	85	2	5	3
kinema	9	8	8	317	3	8	10
yamamura196	30	2[b]	2	21	1	2	2
eco9	8	16	26	15009	76	16	451
camera1s_rev	15	16	20	833	19	16	38
geneig[c]	6	10	11	98122	389	10	3149
geneig_rev[c]	17	10	11	2510	69	10	126
lee	9	2	31	87172	979	2	3751
lee_rev	27	2	3	231	19	2	36
katsura7	8	44[b]	48	23503	213	44	744
stewgou40	9	40	603	34166	1136	40	2128

The results of the local search are ignored in the splitting step.
There is no evidence of lost solutions (due to rounding errors).
(a) Number of solutions is known from external sources.
(b) Not all solutions are verified.
(c) With initial intervals [-10.0, 10.0] due to lack of information.

4.2 Splitting based on the results of the local search

Unfortunately, the proposed splitting rule does not seem to make use of the found solutions, as it is demonstrated in *Table 3.* The reason may be that the splitting rule does not significantly differ from bisection. Other splitting strategies are to be studied, for example separating solutions by splitting.

4.3 Exploiting common subexpressions

As demonstrated in *Table 2*, the difficult benchmarks camera1s, geneig and lee become easier to solve after a straightforward transformation. A new variable (and equation) is introduced and this variable replaces all the simple nonlinear

Table 3. Comparing the computational efforts with and without considering the results of local search in the splitting step.

	Without considering the results of the local search in the splitting step		With considering the results of the local search in the splitting step	
	Splits	Time (s)	Splits	Time (s)
LLE3	261	31	263	31
didrit	232	8	244	6
jacobsen91	85	3	80	4
kinema	317	10	321	10
yamamura196	21	2	31	2
eco9	15009	451	15103	427
camera1s	–	(929)[a]	161511	4798
camera1s_rev	833	38	707	32
geneig	98122	3149	76025	2397
geneig_rev	2510	126	2225	116
lee	87172	3751	83026	3753
lee_rev	231	36	235	36
katsura7	23503	744	24581	761
katsura7_rev	17458	1020	17814	1009
stewgou40	34166	2128	33238	2026

There is no evidence of lost solutions (due to rounding errors).
(a) Function evaluation fails later due to an implementation design flaw. However, all solutions are found by local search in 929 s.

terms appearing multiple times in different equations. For example in the lee benchmark, the expression `xb1*xb2` appears in equations 4, 7, 8, 9. The following equation is appended to the original system: `xb1*xb2 - v1 = 0`, and variable `v1` replaces all other occurrences of `xb1*xb2`; *etc.*

For other transformations and tricks, which make difficult benchmarks easy to solve, the reader is referred to [1]. Exploiting common subexpressions is almost always fruitful; it can reduce computational effort by several orders of magnitude and rarely makes the solution procedure slower. A thorough review on this topic, also presenting a new algorithm for exploiting common subexpressions, is [2].

It remains for a further study to analyze whether it is worth to branch and / or apply LP pruning on the original variables only, or the auxiliary variables introduced for the common subexpression should also be considered. In the computations presented here, all the variables are considered. In connection with this idea, see also the next subsection. In a different context, Kearfott and Hongthong [27] proposed a splitting strategy based on the structure of the DAG. In particular, they discuss strategies determining subspaces in which to branch.

4.4 Problem specific splitting rules

Based on the splitting rule discussed in [3], `extr22` and `extr30` can be easily solved (64 seconds / 1 split and 266 seconds / 3 splits, resp.) if variable `x[3,22]` and `x[3,30]`, resp., is split exclusively. It would be beneficial to find an algorithm that is able to recognize this property of the problem by analyzing the structure of the DAG. The above splitting rule is quite reasonable from engineering point of view.

In case of benchmark `homoazeo20`, only the variables `x[i,20] (i=1..C)`, `T[20]` are split as long as their width is above 0.05 (relative width for `T`), then all the `x` and `T` variables are appended to this set of variables to be split. This splitting strategy is inspired by the advice of Professor Kolev (personal communication, 2005). As cited above in the previous subsection, it would be interesting to see how the proposals of Kearfott and Hongthong [27] relates to these problem specific splitting rules.

The `homoazeo20` with `D = 0.42` (bifurcation parameter) can be solved in 107 s. In case of the `homoazeo40`, all the three solutions are found by the local solver in 9 s but completing the search procedure (proving that there are no more solutions) is not yet possible in reasonable time. IPOPT is able to converge to the solution corresponding to the unstable steady state from a poor starting point. This demonstrates the power of the applied local search method: this solution is missed by state-of-the-art methods dedicated to these problems even when supplied with a good starting point [23].

4.5 Comparisons to other solvers

The authors believe that the difference between the machine and software environment used in the current study and the one that was used in [37] (both PCs with Intel Pentium 4 at 2.6 GHz running Linux) is insignificant, thus the computation time data can be compared in a fair way. *Table 4* and *5* give the computational efforts required to solve the benchmarks with (*i*) the proposed method, (*ii* to *iv*) the latest version of `ICOS` [36] with / without `Quad` filtering (`search-quad.cfg`, and `search.cfg` from [19], resp.) and the reported data in [37], (*v*) a Branch and Prune solver based on the ILOG commercial implementation of box-consistency (data from [37]) and (*vi*) RealPaver [16] (data from [37]).

The authors should know more about the internal algorithms of the other solvers to draw further conclusions. This black-box comparison of solvers does not help to identify which component(s) of a particular solver is (are) responsible for its efficiency / inefficiency on a particular benchmark. Even if the internal algorithms of the solver are known, it is not trivial to find the reasons. As written in the summary of [26]:
"An effective algorithm involves several acceleration techniques that interact. Assessment of a particular technique would change depending on what other techniques are present and when they are applied."

Table 4. Comparison to ICOS.

		The proposed method without local search			The proposed method with local search		ICOS with QUAD filtering			ICOS without QUAD filtering			ICOS with QUAD filtering as reported in [37]		
Benchmark	True sols	Number of boxes not discarded	Splits	Time (s)	Solutions found by the local solver	Time (s)	Sols	Splits	Time (s)	Sols	Splits	Time (s)	Sols	Thousand splits	Time (s)
didrit	4	4	232	**2**	4	**8**	4(4)	59	**12**	4(4)	250160	**348**	4(4)	0.1	**14.7**
kinema	8	8	317	**3**	8	**10**	8(8)	48	**13**	15(12)[a]	3338822	**3466**	8(8)	0.2	**19.9**
yamamura196	2[b]	2	21	**1**	2	**2**	17(0)	314	**29**	0(0)	4264166	**–**[c]	2(1)	0.0	**6.7**
eco9	16	26	15009	**76**	16	**451**	5(4)	237	**–**[c]	17(2)	103206	**104**			*n.t.*[d]
camera1s	16			**(e)**	16	**(929)**[e]	16(16)	328	**24**	0(0)	10044631	**–**	16(16)	1.0	**28.9**
camera1s_rev	16	20	833	**19**	16	**38**	0(0)	91737	**–**	0(0)	5617139	**–**			*n.t.*
geneig[f]	10	11	98122	**389**	10	**3149**	10(10)	453	**30**	10(10)	2250602	**4116**	10(10)	0.8	**39.1**
geneig_rev[f]	10	11	2510	**69**	10	**126**	2(2)	628	**–**	0(0)	5394590	**–**			*n.t.*
lee	2	31	87172	**979**	2	**3751**	2(2)	56	**20**	0(0)	2050523	**–**	2(2)	0.3	**27.1**
lee_rev	2	3	231	**19**	2	**36**	0(0)	299	**–**	0(0)	1566166	**–**			*n.t.*
katsura7	44[b]	48	23503	**213**	44	**744**	7(6)	722	**–**	214(77)[a]	4579224	**–**	58(42)	1.7	**686.9**
katsura7_rev	44[b]	56	17458	**510**	44	**1020**	12(12)	771	**–**	117(50)[a]	1554010	**–**			*n.t.*
stewgou40	40	603	34166	**1136**	40	**2128**	2(2)	61	**–**	3(3)	928192		40(40)	1.6	**924**

Sols stands for the number of solutions. The numbers in brackets in the sols columns give the number of verified unique solutions.
The data for the proposed method are taken from Table 2; the results of the local search (if applied) are ignored in the splitting step.
(a) Conflicting values. They may be caused by some compiler optimization or a bug in ICOS.
(b) Not all solutions are verified.
(c) The '–' sign denotes time out at 7200 seconds.
(d) Not tested in [37].
(e) Function evaluation fails later due to an implementation design flaw. However, all solutions are found by local search in 929 s
(f) With initial intervals [–10.0, 10.0] due to lack of information, likely to be different from [37].

Table 5. `ICOS` compared to different solvers as reported in [37]. The proposed method is compared to `ICOS` in *Table 4*.

	ICOS with Quad filtering			Box consistency (ILOG)			Realpaver	
	Sols	Ksplits	Time (s)	Sols	Ksplits	Time (s)	Sols	Time (s)
didrit	4(4)	0.1	14.7	4(4)	51.3	132.9	4	94.6
kinema	8(8)	0.2	19.9	15(7)	244	572.4	8	268.4
yamamura196	2(1)	0.0	6.7	0(0)	816.7	–[a]	0	–
cameras	16(16)	1.0	28.9	2(2)	11820.3	–	0	–
geneig	10(10)	0.8	39.1	10(10)	290.7	868.6	10	475.6
katsura7	58(42)	1.7	686.9	231(42)	1858.5	11104.1	44	4671.1
stewgou40	40(40)	1.6	924	11(11)	3128.6	–	8	–

Ksplits stands for thousand splits.
(a) The '–' sign indicates time out at 8 hours.

5 Conclusions and future work

In spite of the fact that the current implementation is still just in an initial stage, difficult benchmarks are successfully solved in reasonable time. There is still plenty of room for improvement. Integrating other consistency techniques and state-of-the-art propagation methods [7, 58] into the solver is expected to significantly increase both efficiency and robustness. The numerical results available on the website of ALIAS [1] indicate that symbolic preprocessing and *automatic generation* of redundant constraints to prune the domain of variables can speed up the search procedure by several orders of magnitude in time.

Splitting rules exploiting solutions found by local search will be studied further, *e.g.* separating already found solutions by splitting one component. Gaps obtained with consistency techniques may also help. Kearfott and Hongthong [27] proposed a splitting strategy based on the structure of the DAG; details of this strategy will be examined and compared to the 4.4 *Problem specific splitting rules.*

Pre- and post-solve phases are to be implemented, *e.g.* (pre-solve) appropriate symbolic transformations, such as collecting common subexpression, and aggressive consistency techniques on the root node, automatic generation of redundant constraints for propagation as in ALIAS; (post-solve) procedure for verifying solutions (such as Algorithm 13.3 `VerificationStep` in C-XSC [17], p. 299.) and for discarding the cluster boxes. The latter post-solve procedure could also make use of solutions found by local search. Making the solver rigorous by introducing directed rounding is also planned.

References

1. ALIAS library, the COPRIN project, difficult problems
http://www-sop.inria.fr/coprin/logiciels/ALIAS/Benches/node6.html
2. I. Araya, B. Neveu, G. Trombettoni; *Exploiting Common Subexpressions in Numerical CSPs*; In Proceedings of the 14$^{\text{th}}$ International Conference on the Principles

and Practice of Constraint Programming (CP), Sydney, Australia; LNCS 5202, Springer, 2008, 342–357

3. A. Baharev, T. Achterberg, E. Rév; *Computation of an extractive distillation column with affine arithmetic*; AIChE Journal, 2009, **55** (7), 1695–1704
4. A. Baharev, E. Rév; *Reliable Computation of Equilibrium Cascades with Affine Arithmetic*; AIChE Journal, 2008, **54** (7), 1782–1797
5. A. Baharev, E. Rév; *Comparing inclusion techniques on chemical engineering problems*; 13$^{\text{th}}$ GAMM - IMACS International Symposium on Scientific Computing, Computer Arithmetic, and Verified Numerical Computations SCAN'2008; El Paso, Texas, USA, Sept 29 - Oct 3, 2008
6. T. Beelitz, C. H. Bischof, B. Lang; *A Hybrid Subdivision Strategy for Result-Verifying Nonlinear Solvers*; PAMM, 2004, **4** (1), 632–633
7. T. Beelitz, A. Frommer, B. Lang, P. Willems; *Symbolic-Numeric Techniques for Solving Nonlinear Systems*; PAMM, 2005, **5** (1), 705–708
8. P. Belotti, J. Lee, L. Liberti, F. Margot, A. Waechter; *Branching and Bounds Tightening Techniques for Non-Convex MINLP*; IBM Research Report, RC24620, 2008
9. F. Benhamou, F. Goualard, L. Granvilliers, J.-F. Puget; *Revising hull and box consistency*; In Proceedings of the International Conference on Logic Programming, ICLP'99, 230–244; Las Cruces, New Mexico: MIT Press, 1999
10. COCONUT benchmark, http://www.mat.univie.ac.at/coconut
11. L. H. de Figueiredo, R. Van Iwaarden, J. Stolfi; *Fast interval branch-and-bound methods for unconstrained global optimization with affine arithmetic*; Technical report IC-97-08, Institute of Computing, Univ. of Campinas; June 1997
12. R. Fourer, D. M. Gay, B. W. Kernighan; *AMPL: A Modeling Language for Mathematical Programming*; Second edition, Brooks/Cole USA, 2003
13. D. M. Gay; *Hooking Your Solver to AMPL*; Technical report, Bell Laboratories, Murray Hill, NJ 1993 (revised 1994, 1997)
14. GNU Linear Programming Kit (GLPK); http://www.gnu.org/software/glpk/
15. F. Goualard; *Interval Multivalued Inverse Functions: Relational Interval Arithmetic and its Use*; Invited talk, SCAN'08, El Paso, TX, 2008
16. L. Granvilliers, F. Benhamou; *Algorithm 852: Realpaver: an interval solver using constraint satisfaction techniques*; ACM Transactions on Mathematical Software (TOMS), 2006, **32** (1), 138–156
17. R. Hammer, M. Hocks, U. Kulisch, D. Ratz; *C++ Toolbox for Verified Computing I, Basic Numerical Problems*; Springer-Verlag, Berlin, 1995
18. E. R. Hansen, G. W. Walster; *Global Optimization Using Interval Analysis*; Marcel Dekker, Inc., 2004
19. Interval COnstraints Solver, http://ylebbah.googlepages.com/icos
20. R. J. van Iwaarden; *An improved unconstrained global optimization algorithm*; PhD. Thesis, University of Colorado at Denver, Denver, CO, 1996
21. E. W. Jacobsen, S. Skogestad; *Multiple steady states in ideal two-product distillation*; AIChE Journal, 1991, **37** (4), 499–511
22. C. Jansson; *On self-validating methods for optimization problems*; pp. 381–438 in: Topics in validated computation (J. Herzberger, ed.), Elsevier, Amsterdam 1994
23. A. Kannan, M. R. Joshi, G. R. Reddy, D. M. Shah; *Multiple-Steady-States Identification in Homogeneous Azeotropic Distillation Using a Process Simulator*; Ind. Eng. Chem. Res. 2005, **44**, 4386–4399
24. R. B. Kearfott; *Preconditioners for the Interval Gauss-Seidel Method*; SIAM J. Numer. Anal., 1990, **27** (3), 804–822

25. R. B. Kearfott; *Decomposition of Arithmetic Expressions to Improve the Behavior of Interval Iteration for Nonlinear Systems*; Computing 1991, **47** (2), 169–191
26. R. B. Kearfott; *Empirical Evaluation of Innovations in Interval Branch and Bound Algorithms for Nonlinear Systems*; SIAM Journal on Scientific Computing, 1997, **18** (2), 574–594
27. R. B. Kearfott, S. Hongthong; *Validated Linear Relaxations and Preprocessing: Some Experiments*; SIAM Journal on Optimization, 2005, **16** (2), 418–433
28. R. B. Kearfott, M. Novoa; *Algorithm 681: INTBIS, a Portable Interval Newton/Bisection Package*; ACM Trans. Math. Software, 1990, **16** (2), 152–157
29. L. V. Kolev; *A general interval method for global nonlinear dc analysis*; Proc. of the 1997 European Conf. on Circuit Theory and Design, ECCD'97, Technical University of Budapest, 30th August - 3rd Sept., 1997, Budapest, Hungary, volume 3, pp. 1460–1462
30. L. V. Kolev; *An improved interval linearization for solving non-linear problems*; Numerical Algorithms, 2004, **37**, 213–224
31. L. V. Kolev; *A novel approach to globally solving a class of nonlinear systems*; submitted to Reliable Computing, 2008
32. L. V. Kolev, V. M. Mladenov; *A linear programming implementation of an interval method for global non-linear DC analysis*; IEEE International Conference on Electronics, Circuits and Systems, 1998, **1**, 75–78
33. L. V. Kolev, I. Nenov; *A combined interval method for global solution of nonlinear systems*; Proc. of the XXIII International Conference on Fundamentals of Electronics and Circuit Theory - SPETO 2000, Gliwice, Poland, 24-27 May 2000, pp. 365–368
34. J. W. Kovach, W. D. Seider; *Heterogeneous Azeotropic Distillation: Homotopy-Continuation Methods*; Comput. Chem. Eng. 1987, **11** (6), 593–605
35. J. W. Kovach, W. D. Seider; *Heterogeneous Azeotropic Distillation: Experimental and Simulation Results*; AIChE Journal, 1987, **33** (8), 1300–1314
36. Y. Lebbah; *ICOS: a branch and bound based solver for rigorous global optimization*; Optimization Methods and Software, 2009, **24** (4), 709–726
37. Y. Lebbah, C. Michel, M. Rueher, D. Daney, J.-P. Merlet; *Efficient and Safe Global Constraints for Handling Numerical Constraint Systems*; SIAM J. Numer. Anal., 2004, **42** (5), 2076–2097
38. Y. Lin, C. R. Gwaltney, M. A. Stadtherr; *Reliable Modeling and Optimization for Chemical Engineering Applications: Interval Analysis Approach*; Reliable Computing, 2006, **12**, 427–450
39. Y. Lin, M. A. Stadtherr; *LP Strategy for Interval-Newton Method in Deterministic Global Optimization*; Ind. Eng. Chem. Res., 2004, **43**, 3741–3749
40. Y. Lin, M. A. Stadtherr; *Advances in Interval Methods for Deterministic Global Optimization in Chemical Engineering*; J. Global Optimization, 2004, **29**, 281–296
41. C. M. McDonald, C. A. Floudas; *Decomposition Based and Branch and Bound Global Optimization Approaches for the Phase Equilibrium Problem*; Journal of Global Optimization, 1994, **5** (3), 205–251
42. S. Miyajima, M. Kashiwagi; *Existence test for solution of nonlinear systems applying affine arithmetic*; J. Comput. Appl. Math. 2007, **199**, 304–309
43. I. P. Nenov, D. H. Fylstra; *Interval Methods for Accelerated Global Search in the Microsoft Excel Solver*; Reliable Computing, 2003, **9**, 143–159
44. A. Neumaier; *Complete Search in Continuous Global Optimization and Constraint Satisfaction*; pp. 271–369 in: Acta Numerica 2004 (A. Iserles, ed.), Cambridge University Press 2004

45. J. Nocedal, S. J. Wright; *Numerical Optimization*; Springer, New York, 1999
46. PORT Mathematical Subroutine Library, http://www1.bell-labs.com/topic/swdist/
47. H. Schichl; *Mathematical Modeling and Global Optimization*; Habilitation thesis, Faculty of Mathematics, University of Vienna, Autralia, November (2003)
48. H. Schichl, A. Neumaier; *Exclusion regions for systems of equations*; SIAM J. Numer. Anal., 2004, **42**, 383–408
49. H. Schichl, A. Neumaier; *Interval Analysis on Directed Acyclic Graphs for Global Optimization*; J. Global Optimization, 2005, **33**, 541–562
50. H. Schichl, O. Shcherbina; *External converters*; Technical Report in "Set of Combination Algorithms for State of the Art Modules", Deliverable D14, COCONUT project, July 2003.
51. R. Silva, M. Ulbrich, S. Ulbrich, L. N. Vicente; *A globally convergent primal-dual interior-point filter method for nonlinear programming: new filter optimality measures and computational results*; pre-print 08-49, Department of Mathematics, University of Coimbra
52. P. Spellucci; *An SQP method for general nonlinear programs using only equality constrained subproblems*; Mathematical Programming, 1998, **82** (3), 413–448
53. P. Spellucci; *A new technique for inconsistent QP problems in the SQP method*; Mathematical Methods of Operations Research, 1998, **47** (3), 355–400
54. J. Stolfi, L. H. Figueiredo; *Self-Validated Numerical Methods and Applications*; Monograph for the 21st Brazilian Mathematics Colloquium (CBM'97), IMPA, Rio de Janeiro, Brazil; 1997
55. S. R. Tessier, J. F. Brennecke, M. A. Stadtherr; *Reliable Phase Stability Analysis for Excess Gibbs Energy Models*; Chem. Eng. Sci. 2000, **55**, 1785–1796
56. M. Ulbrich, S. Ulbrich, L N. Vicente; *A globally convergent primal-dual interior-point filter method for nonlinear programming*; Mathematical Programming, 2004, **100**, 379–410
57. A. Vadapalli, J. D. Seader; *A generalized framework for computing bifurcation diagrams using process simulation programs*; Computers and Chemical Engineering, 2001, **25**, 445–464
58. X.-H. Vu, H. Schichl, D. Sam-Haroud; *Interval propagation and search on directed acyclic graphs for numerical constraint solving*; Journal of Global Optimization, DOI 10.1007/s10898-008-9386-7
59. A. Wächter, L. T. Biegler; *Line Search Filter Methods for Nonlinear Programming: Motivation and Global Convergence*; SIAM Journal on Optimization, 2005, **16** (1), 1–31
60. A. Wächter, L. T. Biegler; *On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming*; Mathematical Programming, 2006, **106** (1), 25–57
61. K. Yamamura; *Interval solution of nonlinear equations using linear programming*; Proceedings of IEEE 1997 International Symposium on Circuits and Systems, June 1997, 837–840
62. K. Yamamura, T. Kumakura, Y. Inoue; *Finding All Solutions of Nonlinear Equations Using Inverses of Approximate Jacobian Matrices*; IEICE Trans. Fundamentals, 2001, **E84-A**(11), 2950–2952
63. K. Yamamura, K. Suda; *An Efficient and Practical Algorithm for Finding All DC Solutions of Nonlinear Circuits*; Communications, Circuits and Systems, 2007. ICCCAS 2007. 1111–1115