Some Interval Approximation Techniques for MINLP

Nicolas Berger and **Laurent Granvilliers**
Université de Nantes – CNRS – France
2, rue de la Houssinière – BP 92208 – F-44322 Nantes Cedex 3
nicolas.berger, laurent.granvilliers@univ-nantes.fr

Abstract

MINLP problems are hard constrained optimization problems, with nonlinear constraints and mixed discrete continuous variables. They can be solved using a Branch-and-Bound scheme combining several methods, such as linear programming, interval analysis, and cutting methods. Our goal is to integrate constraint programming techniques in this framework. Firstly, global constraints can be introduced to reformulate MINLP problems thus leading to clean models and more precise computations. Secondly, interval-based approximation techniques for nonlinear constraints can be improved by taking into account the integrality of variables early. These methods have been implemented in an interval solver and we present experimental results from a set of MINLP instances.

Introduction

Mixed Integer Nonlinear Programming (MINLP) problems are constrained optimization problems with both discrete and continuous variables, where the objective function may be nonconvex, and the constraints may be nonlinear (Grossmann 2002; Tawarmalani & Sahinidis 2004). They can be found in many application domains such as engineering design, computational biology, and portfolio optimization.

The integrality of variables and the nonconvexity of functions make MINLP problems difficult to solve. The general Branch-and-Bound scheme is a method of choice, provided bound constraints on the variables. The main principle is to divide the initial problem into a set of subproblems by enumerating discrete domains or bisecting continuous domains. Bounding techniques are used to compute lower and upper bounds on the objective function, thus allowing the rejection of subproblems proved to be non optimal.

Two kinds of bounding techniques may be combined. Interval methods (Moore 1966) can derive improved bounds on domains from relaxed NLP problems, locally considering discrete variables as continuous. Upper bounds on the objective function can also be calculated, possibly using a local optimizer to find feasible points (Neumaier 2004). Linear methods can compute global optima over relaxed LP problems, requiring nonlinearities to be reformulated and linearized (Sherali & Adams 1999). This way, lower bounds on the objective function can be obtained. It turns out that these techniques complement each other. Reducing the domain bounds leads to tighter linear relaxations, and then to more precise lower bounds. Several solvers integrate these techniques, such as Couenne (Belotti *et al.* 2008), which is part of the COIN-OR infrastructure.

We focus on improving state-of-the-art interval-based bounding techniques for MINLP problems. Our starting point is the interval constraint propagation framework (Cleary 1987; Lhomme 1993; Van Hentenryck, Michel, & Deville 1997). Constraint propagation is a general bounding algorithm where contracting operators are applied on domains until they are not refined enough. These operators are based on interval arithmetic, numerical methods for differentiable problems, and projection techniques to tackle nonlinear constraints. They require functions and constraints to be expressed as combinations of arithmetic operations and elementary functions.

In this paper, we propose two kinds of improvements. Firstly, the integrality constraints can be taken into account during constraint propagation to reinforce domain contractions, as done by Apt and Zoeteweij (Apt & Zoeteweij 2007) for pure integer problems. Secondly, MINLP problems may be reformulated by means of so-called global constraints such as the famous `alldifferent` constraint stating that a set of integer variables must all have different values. These constraints give a better expressiveness at modeling time. Moreover, they are associated with specialized domain contraction algorithms. In fact, we investigate the well-known approach of mixing constraints and optimization techniques (Hooker 2007) for MINLP problems.

The remaining of this paper is organized as follows. The following section introduces MINLP problems. Reformulations using global constraints are illustrated for two non trivial problems. Interval approximation techniques and proposed improvements are described in the third section. The next section reviews several ways of processing the `alldifferent` constraint. An experimental study and a conclusion follow.

MINLP and Reformulations

The notion of MINLP problem is introduced in this section and we show how to reformulate two real-world problems using the `alldifferent` global constraint.

Definition

Consider the MINLP problem

$$\min_{x\in\Omega} f(x)$$

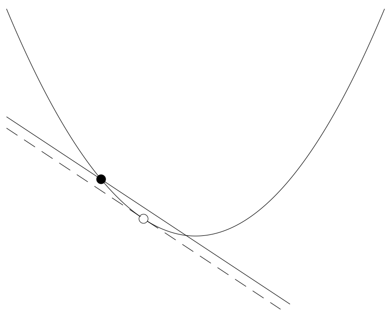
such that Ω is the feasible region defined by

$$(\mathbf{P})\quad \begin{array}{ll} g_j(x) = 0 & \forall j \in \mathcal{J} \\ x_i^l \leqslant x_i \leqslant x_i^u & \forall i \in \mathcal{I} \\ x_k \in \mathbb{Z} & \forall k \in \mathcal{K} \subseteq \mathcal{I} \end{array}$$

where f and the g_j are possibly nonconvex functions and x is the vector of variables such that the x_k are integer variables for all $k \in \mathcal{K}$. The goal is to find the feasible points x minimizing f.

Example 1 *Consider the constraint* $x_2 - x_1^2 + 1 = 0$ *and let the objective be* $\min(x_2+1.1(x_1-1))$. *Solving the first-order optimality conditions leads to the global minimum* $(-0.55, -0.6975)$ *with objective* -2.4025 *(white point).*

Now suppose that x *is an integer variable. The global minimum becomes* $(-1, 0)$ *with objective* -2.2 *(black point).*



Considering integrality constraints make problems more complex to solve. In particular, the optimality conditions do not hold at local optima.

Reformulations

We first introduce Montemanni's mixed-integer programming formulation for the total flow time single machine robust scheduling with interval data (Montemanni 2007), where an interval of equally possible processing times is considered for each job. Optimization is carried out according to a *robustness* criterion: we want to find the schedule that performs the best under the worst case scenario.

Each of some N tasks must be assigned a distinct processing order, which models in MINLP using binary variables: x_{ik} is true when task i is processed in k^{th} position under the constraints that each job is given exactly one position and each position is given to exactly one job:

$$\sum_{i=1}^{N} x_{ik} = 1,$$
$$\sum_{k=1}^{N} x_{ik} = 1.$$

This scheme perfectly fits with the purpose of the `alldifferent` global constraint. Given a set of integer variables t_i ranging in $\{1, \ldots, N\}$, the two above constraints can be reformulated into an `alldifferent` global constraint:

$$\text{alldifferent}([t_1, \ldots, t_N]).$$

The model we give here for the robust scheduling problem results from a well-known technique that exploits the unimodularity property of the nested maximization problem.

$$\min \sum_{i=1}^{N} \eta_i + \sum_{k=1}^{N} \tau_k$$

$$\text{subject to :}$$

$$\begin{array}{lrcl}
(1) & & & \text{alldifferent}(t_1, \ldots, t_N), \\
(2) & t_i & = & \sum_{k=1}^{N} x_{i,k} \cdot k, \\
(3) & \sum_{k=1}^{N} x_{ik} & = & 1, \\
(4) & \eta_i + \tau_k & \geqslant & \overline{p_i} \sum_{j=1}^{k} (k-j) x_{ij} \\
& & & + \ \underline{p_i} \sum_{j=k}^{N} (k-j) x_{ij}, \\
& x_{ik} & \in & \{0, 1\}, \\
& t_i & \in & \{1, \ldots, N\}, \\
& \eta_i & \in & \mathbb{R}, \\
& \tau_k & \in & \mathbb{R}.
\end{array}$$

Constraint *(1)* expresses that each task must have a distinct processing order; *(2)* channels this value to binary variables as needed by constraint *(4)*. Constraint *(3)* forbids this channeling to imply more than one binary variable set to true. Finally, *(4)* simply links the variables we maximize the sum of with the arithmetic expression of the regret cost of the worst case scenario for this schedule.

We now introduce a part of the simplified yet realistic model for the nuclear reactor core optimal reload pattern problem (Quist *et al.* 1999) that can be found in the *MINLPLib* collection of MINLP problems (Bussieck, Drud, & Meeraus 2003).

The nuclear reactor reloading problem consists in finding the optimal fuel reloading pattern for a nuclear reactor core. In the model we consider here, the objective is to maximize the end-of-cycle ratio of neutron production rate over neutron loss rate —what is called the *reactivity* of the core— while satisfying some security constraints, given that the position of fuel bundles at the end of a cycle determines their initial properties for the next cycle.

The trajectory of the bundles in the core from cycle to cycle can be represented by a matrix of the following kind:

$$\begin{array}{ccccccc} 4 & \rightarrow & 7 & \rightarrow & 3 & \rightarrow & 1 \\ 2 & \rightarrow & 8 & \rightarrow & 10 & \rightarrow & 12 \\ 5 & \rightarrow & 11 & \rightarrow & 6 & \rightarrow & 9 \end{array}$$

We see here a possible 4 cycles long trajectory, for a core of 12 nodes in which 3 bundles —a quarter of the bundles— are discharged at each end-of-cycle. Each number refers to a different node of the core. According to this matrix, the bundles located at nodes 1, 12 and 9 are discharged at each reloading while fresh bundles are inserted into nodes 4, 2 and 5. The bundle in node 7 is moved to node 3, the bundle in node 10 is moved to node 12, etc.

Here again, the classical MINLP approach to modeling the assignment of a node to a cell of the matrix is to use binary variables: $x_{i,l,m}$ is true when the bundle in node i is affected to column l and row m in the trajectory matrix. We then have to constrain these variables so as exactly one will be set to true for a given i —each node must appear exactly once in the matrix. Reciprocally, exactly one variable must be set to true for a given position l, m in the matrix, as each cell of the matrix must be assigned to exactly one node. Thus we have:

$$\sum_{i=1}^{N} x_{i,l,m} = 1,$$
$$\sum_{l=1}^{L} \sum_{m=1}^{M} x_{i,l,m} = 1.$$

It turns out that the problem of assigning each node a position in the trajectory matrix can be expressed by means of the `alldifferent` global constraint. Given the following indexing of the cells:

$$\begin{array}{cccc} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{array}$$

each node i must be assigned a distinct position number n_i, ranging in $\{1, \ldots, 12\}$, which fits perfectly with the purpose of this global constraint. Then the trajectory matrix assignment constraint models simply as following:

$$\text{alldifferent}([n_1, \ldots, n_N]).$$

Interval Approximation Techniques

Interval constraint propagation can be applied on NLP problems defined from MINLP problems by relaxing the integrality constraints. Some interval solvers are able to tackle these constraints by regularly truncating the bounds of integer domains. In this section, we propose to take them into account in the early phases of interval calculations, leading to stronger domain contractions.

Interval arithmetic

Interval arithmetic is an arithmetic over real intervals. For instance we have:

$$\begin{array}{lcl} [a, b] + [c, d] & = & [a + c, b + d] \\ [a, b] - [c, d] & = & [a - d, b - c] \\ [a, b]^3 & = & [a^3, b^3] \end{array}$$

These operations can be used to enclose the range of a real function over an interval domain (Moore 1966). For instance, let $f(x) = x^3 + 1$ and let $x \in D_x = [-2, 2]$. Then we have:

$$f(x) \in F(D_x) = D_x^3 + [1, 1] := [-7, 9]$$

where F is an interval extension of f.

Constraint propagation

Interval arithmetic can be used to implement contracting operators of domains of variables occurring in linear or nonlinear constraints. For instance let $x + y = 1$ with $x \in [0, 2]$ and $y \in [0, 2]$. It is immediate to rewrite the constraint as $x = f(y) = 1 - y$. Then the domain of x can be contracted as follows:

$$D_x := \text{hull}(D_x \cap F(D_y)) := [0, 1]$$

where hull computes the interval enclosure of a set of real numbers. This operation is necessary in some cases, when F results in a union of intervals, as we will see later.

This operator computes the projection of constraint $x + y = 1$ over x within the given interval domain using a constraint inversion method. A very similar operator can be associated to the projection over y. This idea originates from (Cleary 1987) and it has been extended to efficiently cope with arbitrary complex constraints (Benhamou *et al.* 1999).

Given a constraint system, a set of contracting operators can be defined from the set of constraint projections. They have to be iteratively applied to reach a stable state where no domain can be reduced enough, in a so-called constraint propagation algorithm.

Handling integrality constraints

As said before, some interval solvers are able to apply integrality constraints during constraint propagation. The common way consists in implementing, for every integer variable x, a contracting operator truncating the domain bounds, as follows:

$$D_x := \text{hull}(\text{int}(D_x))$$

where int takes the integral part of a set of real numbers.

This operator is regularly applied during constraint propagation, allowing to maintain integer bounds for the integer variables. In fact, it is possible to perform truncation directly in contracting operators (next paragraph) and to improve interval arithmetic to cope with integer data (last paragraph).

Early truncation Let $x = f(y)$ be a projection constraint and let x be an integer variable. Then the domain of x can be reduced as follows:

$$D_x \leftarrow \text{hull}(\text{int}(D_x \cap F(D_y))) \qquad (1)$$

where the integral part of the set $D_x \cap F(D_y)$ happens before the hull. This operator may replace the classical implementation of integrality constraints, which is equivalent to:

$$\begin{array}{lcl} D_x & := & \text{hull}(D_x \cap F(D_y)) \\ D_x & := & \text{hull}(\text{int}(D_x)) \end{array} \qquad (2)$$

where the processing of the integrality constraint is done after the domain contraction wrt. the constraint.

We have the following proposition.

Proposition 1 *Let D_x^1 be obtained from equation (1) and let D_x^2 be obtained from equation (2). Then we have $D_x^1 \subseteq D_x^2$.*

As a consequence, every contracting operator for the domain of an integer variable must compute integral sets before returning interval hulls.

Example 2 *Let $y = x^2$ such that $y \in [2,3]$ is a real variable and $x \in [-2,2]$ is an integer variable. The domain of x is computed by means of the inverse of the square operation applied on y. It follows that x must belong to $[-\sqrt{3}, -\sqrt{2}] \cup [\sqrt{2}, \sqrt{3}]$. Then we have:*

$$\begin{array}{lcl} \text{hull}(\text{int}(\text{hull}([-\sqrt{3}, -\sqrt{2}] \cup [\sqrt{2}, \sqrt{3}])) & = & [-1,1] \\ \text{hull}(\text{int}([-\sqrt{3}, -\sqrt{2}] \cup [\sqrt{2}, \sqrt{3}]) & = & \varnothing \end{array}$$

which shows that the integrality constraint must be considered as soon as possible during interval computations.

Improved arithmetic Interval arithmetic can be improved by means of integrality constraints. For instance, let x be an integer variable and suppose that D_x strictly contains 0. Then we have:

$$1/x \in [-1, 1/\min(D_x)] \cup [1/\max(D_x), 1] \qquad (3)$$

This new operation may be compared with the original definition of the division when the integrality constraint is relaxed, as follows:

$$1/x \in [-\infty, 1/\min(D_x)] \cup [1/\max(D_x), +\infty] \qquad (4)$$

It is clear that equation (3) leads to a more precise interval wrt. equation (4), by tightening the extremal bounds. In fact, there is no real number within $[-\infty, -1) \cup (1, +\infty]$ obtained as $1/x$ with x integer.

The purpose here is not to present the full interval arithmetic extended to mixed-integer problems. However, let us remark that the division is much useful, in particular for polynomial problems, as illustrated in the following example.

Example 3 *Let $x \in [-10, 10]$ be an integer variable and let $y \in [-10, 10]$ be a real variable. Consider the system $xy = 1, x + y = 2$.*

1. *By using Algorithm (4) of the interval division the first equation becomes useless. Only the second equation is useful to contract the domains, as follows:*

$$\begin{array}{lclcl} D_x & := & \text{hull}(D_x \cap (2 - D_y)) & := & [-8, 10] \\ D_y & := & \text{hull}(D_y \cap (2 - D_x)) & := & [-8, 10] \end{array}$$

2. *Considering now the integrality of x, Algorithm (3) can be enforced. As a consequence, the first equation leads to contract the domain of y and to propagate through the system, eventually computing the only solution, as follows:*

$$\begin{array}{lclcl} D_y & := & \text{hull}(D_y \cap (1/D_x)) & := & [-1, 1] \\ D_x & := & \text{hull}(\text{int}(D_x \cap (2 - D_y))) & := & [1, 3] \\ D_y & := & \text{hull}(D_y \cap (1/D_x)) & := & [\frac{1}{3}, 1] \\ D_x & := & \text{hull}(\text{int}(D_x \cap (2 - D_y))) & := & 1 \\ D_y & := & \text{hull}(D_y \cap (1/D_x)) & := & 1 \end{array}$$

Taking integrality of variables into account early during arithmetic evaluation not only leads to stronger domain reductions as shown above, but also can help achieving exponential solving time gains.

Example 4 *Let x be an n-dimensional real vector and let y be an n-dimensional integer vector. Suppose that each variable lies in $[-10^8, 10^8]$ and consider the following system:*

$$\left\{ \begin{array}{lcl} x_i y_i & = & 1, \quad i = 1, \ldots, n \\ \displaystyle\sum_{i=1}^{n} x_i^2 & = & 2n \end{array} \right.$$

If Alg. (3) is implemented then the constraint propagation algorithm alone is able to prove inconsistency. The new division method allows domains to be contracted with respect to the constraint $x_i y_i = 1$.

Only weak contractions are obtained by means of Alg. (4). To prove inconsistency, constraint propagation must be combined with a bisection technique. However, it requires an exponential number of bisections, exactly 2^{n-1}.

Example 5 *Let x be a n-dimensional real vector and let y be a n-dimensional integer vector. Suppose that each variable lies in $[-10^8, 10^8]$ and consider the following system:*

$$\left\{ \begin{array}{lcl} (x_i - 1) y_i & = & 1, \quad i = 1, \ldots, n \\ \log(\displaystyle\prod_{i=1}^{n} y_i) & = & 1 \end{array} \right.$$

The following table shows the number of bisections required to prove inconsistency, using the two different algorithms of the division.

n	Alg. (3)	Alg. (4)	Ratio
3	7	16	2.3
4	23	71	3.1
5	63	288	4.6
6	159	1099	6.9
7	383	4016	10.5
8	895	14223	15.9
9	2047	49216	24.0

Once again, the new algorithm is shown to be more efficient. We obtain an exponential gain in the number of bisections, where the ratio can be expressed as the function $f(n) = 1.4^n$.

Processing Global Constraints

In this section, we present several ways of processing the `alldifferent` constraint, either transforming it into a set of constraints supported by the solver, or implementing specialized algorithms.

Techniques

Clique of disequalities A naive implementation of the `alldifferent` constraint for n variables uses a clique of disequalities on pairs of variables, thereby adding in the background $\mathcal{O}(n^2)$ new constraints to the model. Many opportunities for filtering the domains of the variables are lost during such a translation; on the other hand, it only requires that the solver provides a disequality constraint.

Inter-distance constraints When the solver does not implement a disequality constraint, we can model each disequality between two integer variables as a greater or equal to one distance constraint:

$$\{|x_i - x_j| \geqslant 1 \mid 1 \leqslant i < j \leqslant n\}.$$

where the absolute can also be replaced with a square $(x_i - x_j)^2$. This way, we still have the ability to express some model making use of the `alldifferent` constraint in a solver that does not implement a disequality constraint.

Sum constraint We can strengthen the clique modelind above with the help of the convex hull relaxation of the `alldifferent` constraint as defined in (Hooker 2002) for integer interval domains:

$$\begin{cases} \displaystyle\sum_{j=1}^{n} x_j = \frac{n(n+1)}{2} \\ \displaystyle\sum_{j \in J} x_j \geqslant \frac{|J|(|J|+1)}{2},\ \forall J\ \subset \{1,\ldots,n\}\ with\ |J| < n \end{cases}$$

provided that $x_j \in [1, n]$. If we have $x_j \in [a, b]$ with $a \neq 1$, it suffices to consider the translation $y_j = x_j - a + 1 \in [1, b - a + 1]$.

However, the number of constraints in the relaxation constraints set described by the formula above grows exponentially with the number of variables in use in the model, and it easily becomes impractical to use this set as is. Still, our experiments showed that the fastest minimal subset of the convex hull relaxation was the one consisting only of the equality constraint summing all variables.

Domain cardinality We introduce here a domain cardinality operator # applying on a union of integer intervals and returning the number of distinct values in it. Now suppose that a `alldifferent` constraint on a set of variables $x_1, \ldots, x_n$ holds true. Then we have

$$\#\,(D_{x_1} \cup \cdots \cup D_{x_n}) \geqslant n$$

where D_{x_i} is the domain of x_i for every i. This is in fact the well-known necessary condition based on the Hall interval associated with the whole set of variables.

The following specialized algorithm is implemented. First, if the domain cardinality operator calculates an integer smaller than n then the constraint fails. Second, when a variable is fixed to some value, this value is removed from the domains of all the other variables for which it is a bound value. This later technique corresponds to the classical processing of disequality constraints: given $x \neq y$ and $x = a$ then remove a from the domain of y.

Bound consistency We can also implement a dedicated, state-of-the-art bound consistency filtering algorithm such as the one given in (López-Ortiz *et al.* 2003), or a classical matching algorithm (Régin 1994).

Experiments

In this section are presented our experimental results on both generated and real-world problems.

Generated problems We designed three problems with different features. The first one `int-frac` is a purely discrete problem:

$$\begin{cases} \text{alldifferent}(x_1, \ldots, x_n), \\ \dfrac{x_1}{x_2} + \ldots + \dfrac{x_{n-1}}{x_n} = \dfrac{1}{2} + \ldots + \dfrac{n-1}{n}, \\ x_i + 1 \leqslant x_{i+2},\ 1 \leqslant i \leqslant n,\ i \text{ odd}, \\ x_i \in \{1, \ldots, n\},\ 1 \leqslant i \leqslant n. \end{cases}$$

The second one `mixed-sin` is a mixed-integer problem with linear and non-linear constraints:

$$\begin{cases} \text{alldifferent}(p_1, \ldots, p_n), \\ x_i p_i\ +\ \displaystyle\sum_{j \neq i} x_j = i,\ 1 \leqslant i \leqslant n, \\ \sin(\pi x_i) = 0,\ 1 \leqslant i \leqslant n, \\ p_i \in \{1, \ldots, n\},\ 1 \leqslant i \leqslant n, \\ x_i \in [-100, 100],\ 1 \leqslant i \leqslant n. \end{cases}$$

The third one `hard-mixed-sin` is a mixed-integer problem generator inspired from `mixed-sin` but with harder non-linear constraints:

$$\begin{cases} \text{alldifferent}(p_1, \ldots, p_n), \\ \sin(x_i p_i)\ +\ \displaystyle\sum_{j \neq i} x_j = i,\ 1 \leqslant i \leqslant n, \\ \sin(\pi x_i) = 0,\ 1 \leqslant i \leqslant n, \\ p_i \in \{1, \ldots, n\},\ 1 \leqslant i \leqslant n, \\ x_i \in [-10, 10],\ 1 \leqslant i \leqslant n. \end{cases}$$

The experimental results are presented in Table 1. All the experiments were conducted on an Intel Xeon at 3 GHz with 16 GB of RAM, using the interval solver `Realpaver 1.0` (Granvilliers & Benhamou 2006). For every problem, `var` gives the number of discrete variables and the number of continuous variables on the constraint model with `alldifferent` constraints and `sol` stands for the number of solutions. The last four columns correspond to the different methods: `MILP` for the MILP problem formulation; $\neq$ for the problem formulation with disequality constraints; `SUM` $\neq$ for the problem formulation with disequality constraints and the convex hull relaxation; `CARD` for the problem formulation with cardinality constraints; `ALLDIF` for the problem formulation with the dedicated, state-of-the-art `alldifferent` global constraint filtering. The solving times in seconds are given in the first row and the number of branching steps in thousands are given in the second row. A question mark stands for a method that cannot solve the problem in reasonable time.

First of all, it is clear that the MILP formulation is not efficient for those problems. For instance, let us consider the constraint $b + \sum_i b_i = 1$ stating that exactly one variable among a set of Boolean variables must be true —this is one of the Boolean constraints used to simulate the `alldifferent` constraint. Suppose that the domain of b is split. If $b = 1$ it is immediate to derive that each b_i is equal to 0. However, if $b = 0$ nothing can be deduced for the other variables. That may explain the great number of splitting steps in the `MILP` column.

The $\neq$ formulation is more efficient than the `MILP` formulation but only for small problem instances: in case of larger problems, it does not permit to solve the problem in reasonable time. On the contrary, the `SUM` $\neq$ formulation is able to solve all the instances in a significantly lower time. However, we can see that the harder the problem, the smaller the improvement wrt. the $\neq$ formulation.

The `CARD` formulation is up to five times faster than the `SUM` $\neq$ formulation, especially on instances of `int-frac` while in other cases, solving times are only reduced in a factor up to a third. The `ALLDIF` formulation performs as good as `CARD` on instances of `mixed-sin` and `hard-mixed-sin`, but is a bit less than twice faster on instances of `int-frac`.

Real-world problems We solved several instances of the problems we described in Section 2: three of `Scheduling` with respectively 9, 10, 11 tasks; two of `Nuclear`, one with a trajectory matrix of size 3x3 with each cycle divided into 3 time steps —`NuclearA`— and one with a trajectory matrix of size 4x3 with each cycle divided into 2 time steps —`NuclearB`. Experimental settings are the same than introduced in the previous subsection and the results are presented in Table 2.

The `CARD` formulation is always the fastest. Nonetheless, for those instances the $\neq$ formulation remains competitive which was clearly not the case for the generated problems in the previous subsection. An explanation for this can be that in these real-world problems, the `alldifferent` constraint yields not many domain reductions wrt. the other hard, many continuous constraints. In such a situation, the simpler the filtering, the faster the resolution. This should also explain that the `MILP` formulation for `Nuclear` instances surprisingly performs quite as well as the best formulation.

Another surprising result is the behavior of the `ALLDIF` formulation for all the tested instances. The `MILP` formulation on `Schedule` instances excepted, it is always slightly slower than the other formulations. An explanation could be that the dedicated filtering algorithm does not yield sufficiently many domain reductions wrt. its time cost, which makes the other formulations competitive.

To conclude, it seems that the `CARD` formulation is a good approach to implement the `alldifferent` constraint in continuous constraint solvers. Its implementation, which simply requires to compute domain unions and to remove values at domain bounds, is very easy. Furthermore, the resulting domain contractions can be much better than the other formulations.

Conclusion

Interval methods are designed to solve NLP problems using a branch-and-bound algorithm. In this paper, we have proposed several improvements for solving MINLP problems. We have shown that interval arithmetic can be extended to process problems with discrete and continuous variables. The new operations handle the integrality constraints, resulting in tighter interval computations for mixed nonlinear expressions.

We have also proposed to reformulate combinatorial parts of MINLP problems using global constraints. Stronger domain contractions are obtained by means of specialized algorithms for processing these global constraints. Several algorithms for the `alldifferent` constraint have been compared. It reveals that simple techniques (MILP models and sum relaxations) are enough for small integer subproblems. However, bound consistency algorithms behave better in general. We believe that they must be implemented in efficient mixed constraint solvers.

This work will be extended in several directions. A systematic interval arithmetic will be defined to process mixed variables. We plan to implement new relaxations of the `alldifferent` constraint, in particular for exploiting the inequality constraints related to the sum-based technique, as suggested by the reviewers. Finally, the next step will be to fully implement a mixed interval solver and to integrate it in a MINLP solver like Couenne. That will then allow us to make new experiments from sets of standard benchmarks.

Acknowledgments

We are grateful to Frédéric Goualard for interesting discussions on these topics. We also thank the anonymous reviewers for their valuable comments on a preliminary version of this article.

References

Apt, K., and Zoeteweij, P. 2007. An Analysis of Arithmetic Constraints on Integer Intervals. *Constraints* 12(4):429–468.

Belotti, P.; Lee, J.; Liberti, L.; Margot, F.; and Wacher, A. 2008. Branching and Bounds Tightening Techniques for Nonconvex MINLP. Submitted.

Benhamou, F.; Goualard, F.; Granvilliers, L.; and Puget, J. 1999. Revising Hull and Box Consistency. In *ICLP*.

Bussieck, M. R.; Drud, A. S.; and Meeraus, A. 2003. Minlplib - a collection of test models for mixed-integer nonlinear programming. *INFORMS Journal on Computing* 15(1).

Cleary, J. 1987. Logical Arithmetic. *Future Computing Systems* 2(2):125–149.

Granvilliers, L., and Benhamou, F. 2006. Algorithm 852: Realpaver: an interval solver using constraint satisfaction techniques. *ACM Trans. Math. Softw.* 32(1):138–156.

Grossmann, I. 2002. Review of Nonlinear and Mixed Integer and Disjunctive Programming Techniques. *Optimization and Engineering* 3(3):227–252.

Hooker, J. N. 2002. Logic, optimization and constraint programming. *INFORMS Journal on Computing* 14:295–321.

Hooker, J. 2007. A Framework for Integrating Optimization and Constraint Programming. In *SARA*.

Lhomme, O. 1993. Consistency Techniques for Numeric CSPs. In *IJCAI*.

López-Ortiz, A.; Quimper, C.-G.; Tromp, J.; and van Beek, P. 2003. A fast and simple algorithm for bounds consistency of the alldifferent constraint. In *IJCAI*.

Montemanni, R. 2007. A Mixed Integer Programming Formulation for the Total Flow Time Single Machine Robust Scheduling Problem with Interval Data. *Journal of Mathematical Modelling and Algorithms* 6(2):287–296.

Moore, R. 1966. *Interval Analysis.* NJ: Prentice-Hall, Englewood Cliffs.

Neumaier, A. 2004. Complete Search in Continuous Global Optimization and Constraint Satisfaction. *Acta Numerica* 13:271–369.

Quist, A.; Geemert, R. V.; Hoogenboom, J.; Illest, T.; Klerk, E. D.; Roos, C.; and Terkaly, T. 1999. Finding Optimal Nuclear Reactor Core Reload Patterns Using Nonlinear Optimization and Search Heuristics. *Engineering Optimization* 32(2):143–176.

Régin, J.-C. 1994. A filtering algorithm for constraints of difference in csps. In *AAAI*, pp 362–367.

Sherali, H., and Adams, W. 1999. *A Reformulation-Linearization Technique for Solving Discrete and Continuous Nonconvex Problems.* Kluwer Academic Publishers.

Tawarmalani, M., and Sahinidis, N. 2004. Global Optimization of Mixed Integer Nonlinear Programs: A Theoretical and Computational Study. *Mathematical Programming* 99(3):563–591.

Van Hentenryck, P.; Michel, L.; and Deville, Y. 1997. *Numerica: a Modeling Language for Global Optimization.* MIT Press.

Problem	var	sol	MILP	$\neq$	SUM $\neq$	CARD	ALLDIF
`int-frac`	12/0	24	751.4 394.6	26.2 51.3	7.43 14.7	1.24 8.72	**0.8** 5.9
	14/0	49	?	780.7 1160.2	179.3 263.5	25.8 149.1	**15.4** 102.0
	16/0	1068	?	?	4845.3 5607.6	719.8 3024.5	**362.3** 2082.7
`mixed-sin`	5/5	121	140.9 271.4	8.9 19.2	7.7 16.2	6.0 14.7	**5.8** 13.6
	6/6	746	5587.2 8143.9	187.0 307.4	133.0 202.3	101.6 177.7	**97.1** 164.2
	7/7	5041	?	?	7642.3 10260.3	5532.0 8914.7	**5389.0** 8350.3
`hard-mixed-sin`	4/4	24	3.7 9.7	1.0 1.8	0.9 1.6	**0.8** 1.6	**0.8** 1.5
	5/5	832	484.0 956.7	90.2 151.5	89.3 120.9	71.2 117.1	**70.9** 114.9
	6/6	18228	?	?	?	?	?

Table 1: Solving time in seconds (first row) and number of branchings in thousands.

Problem	var	sol	MILP	$\neq$	SUM $\neq$	CARD	ALLDIF
`Schedule9`	90/19	1	750.3 169.7	9.0 1.9	8.5 1.9	**8.0** 1.9	9.8 1.9
`Schedule10`	110/21	1	1 303.9 229.0	38.0 6.6	38.5 6.6	**35.2** 6.6	37.9 6.6
`Schedule11`	132/23	1	?	205.0 25.5	197.6 25.5	**186.3** 25.5	228.6 25.5
`NuclearA`	90/57	1	78.7 6.4	79.2 6.4	79.0 6.4	**78.0** 6.4	81.0 6.4
`NuclearB`	156/50	1	659.7 12.7	660.8 12.7	671.3 12.7	**658.2** 12.7	693.0 12.7

Table 2: Solving time in seconds (first row) and number of branchings in thousands.