Nonlinear Symbolic Transformations for Interval Optimization Problems

Tibor Csendes[1*] and Elvira D. Antal[2]

[1*]Institue of Informatics, University of Szeged, Árpád tér 2, Szeged, H-6720, Hungary.
[2]Department, John von Neumann University, Street, Kecskemét, H-6000, Hungary.

*Corresponding author(s). E-mail(s): csendes@inf.szte.hu;
Contributing authors: antale@inf.szte.hu;

Abstract

With the spreading of the symbolic algebra systems it is more and more obvious that they can be utilized to transform optimization problems automatically. Such simplifications can have some advantages. At one hand the related functions can be calculated by less operations, and hence quicker. Then we can recognize substructures in the computation tree that are redundant – which remain usually hidden. Finally the possible dimension reduction can result in the decrease of the necessary number of iterations of the optimization method applied. Due to the automatic simplification, our procedure does not require human overhead and larger complex problems can be solved.

In the present paper we give an overview of our work done in this subfield, and provide new computational results for an interval arithmetic based global optimization algorithm in terms of precision and efficiency. Closely 18% of the standard global optimization problems could be simplified automatically, and we could achieve dramatic improvements of many orders of magnitude in precision of the place or value of the minima.

Keywords: Symbolic manipulation, Global optimization, Interval arithmetic, Verification of neural networks

MSC Classification: 65K05 , 90C30

1 Introduction

Symbolic tools have a long history in the solution of optimization problems. For example, such are the symbolic preprocession in linear programming [17], or such transformations as the "presolving" mechanism of the AMPL [10, 11]. A recent example is the Reformulation-Optimization Software Engine by Liberti and coworkers for mixed integer optimization [15].

The papers of Csendes and Rapcsák [8, 20] demonstrated that it is possible to transscribe unconstrained nonlinear optimization problems by symbolic tools in such a way that a bijection exists between the extrema of the two optimization problems. This method is capable to get rid of redundant variables, and to simplify the problem in other way too.

The original motivating problem was a bit complex, related to parameter estimation of the function for respiratory system modelling [5, 14]:

$$F\left(R_{aw}, I_{aw}, B, \tau\right) = \left[\frac{1}{m}\sum_{i=1}^{m}\left|Z_L(\omega_i) - Z'_L(\omega_i)\right|^2\right]^{1/2}.$$

This function is to be minimized, where $Z_L(\omega_i) \in \mathbb{C}$ is the measured impedance value, and $Z'_L(\omega_i)$ is the model function of the impedance for the ω_i frequency values $(i = 1, 2, \ldots, m)$. The parameters to be determined are R_{aw}, I_{aw}, B, and τ. The original nonlinear model function is with its physical parameters:

$$Z'_L(\omega) = R_{aw} + \frac{B\pi}{4.6\omega} - \imath\left(I_{aw}\omega + \frac{B\log(\gamma\tau\omega)}{\omega}\right),$$

where $\gamma = 10^{1/4}$ and $\imath$ is the imaginary unit.

The development of the symbolic algorithm was motivated by the simplified model function, which is obviously linear in the new model parameters:

$$Z'_L(\omega) = R_{aw} + \frac{B\pi}{4.6\omega} - \imath\left(I_{aw}\omega + \frac{A + 0.25B + B\log(\omega)}{\omega}\right).$$

The hard to find clever substitution is $A = B\log(\tau)$. The number of model parameters remained the same. The substantial merit of the transformation is that the linear model function can be fitted in the least-squares sense directly, no optimization algorithm is needed.

In the second section we introduce the idea and give the found supporting theoretical results. The third section will provide the details on the advantages obtained by using the suggested procedure.

2 The simplification method, idea, and theoretical results

Consider the unconstrained nonlinear optimization problem in the form

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}), \tag{1}$$

where $f(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$ is a smooth function that is also known in expression form. Here we understand expression as the series of symbols (constant, variable, operation sign, function sign and brackets) written in a closed form, in a syntactically correct form. This expression can be given in a directed acyclic graph [21] form in a symbolic algebra system (such as Mathematica). The availability of the symbolic form is a natural precondition, although the objective function of an optimization problem is often given only as a calculation procedure, or in the form of an approximation scheme or e.g. available through some simulation.

The simplification procedure will decide whether (1) can be transformed into such an equivalent form, that is preferable according to some features, e.g. the evaluation of the new function requires less arithmetic operations, the dimension of the problem is smaller, or it is easier to solve for some reason. The equivalent form means here that a bijection exists between the extrema of the original and transformed problems. Csendes and Rapcsák showed [8] that an objective function $g(\boldsymbol{y})$ is equivalent with $f(\boldsymbol{x})$, if $g(\boldsymbol{y})$ can be provided by the following transformation:

- apply a substitution for $f(\boldsymbol{x})$-re:

$$y_i := h(\boldsymbol{x}) \,, \quad 1 \le i \le n,$$

  where $h(\boldsymbol{x})$ is a smooth function, the range of which is $\mathbb{R}$, and it is strictly monotonous at least in one x_i variable,
- rename the remaining variables:

$$y_j := x_j \,, \quad j = 1, \ldots, i-1, i+1, \ldots, n,$$

  and
- delete those y_i variables, that do not appear in the resulting objective function.

We call a $y_i = h(\boldsymbol{x})$ substitution to be a *suitable substitution*, if it fulfills the conditions:

- $h(\boldsymbol{x})$ is smooth, monotonous at least in one variable x_i, and its range is $\mathbb{R}$,
- at least for a variable x_i the expression $h(\boldsymbol{x})$ represents all occurrences of the variable x_i, i.e. the variable x_i will disappear from the computation tree of $f(\boldsymbol{x})$-ből, if we substitute $h(\boldsymbol{x})$ everywhere by y_i, and
- $y_i = h(\boldsymbol{x})$ is not a simple renaming, in other words $h(\boldsymbol{x}) \neq x_i$, $i = 1, \ldots, n$.

After the $y_i = h(\boldsymbol{x})$ substitution the number of $\boldsymbol{y}$ variables is at most so much as the dimension of $\boldsymbol{x}$. The redundant variables will be deleted, when $h(\boldsymbol{x})$ gives all the occurrences of some variables. In other words, we can recognize that the model can be formulated with less unknowns. The given conditions for the substitution are satisfactory, but not always necessary. In this way, our proposed algorithm is just one of the possible techniques, but other procedures may also exist.

Consider for example the minimization of the function $f(x_1, x_2) = (x_1 + x_2)^2$. This is obviously equivalent with finding the minimum of $g(y_1) = y_1^2$, and the respective optimal values of x_1 and x_2 can be determined by the equation $y_1 = x_1 + x_2$, that gives a proper substitution. In this way we can handle an infinity of minimizer points, that would not be possible by traditional computational techniques. This is one of the main aims of our method: by symbolic substitution to recognize redundancy, and to get rid of it in our function by the proper substitution.

The following two theorems proved in [8] gives conditions for the applied transformations to have an applicable way to calculate the extrema of the simplified function from those of the original one, and vice versa.

Theorem 1 *If $h(x)$ is smooth and strictly monotonic in x_i, then the corresponding transformation simplifies the function in the sense that each occurrence of $h(x)$ in the expression of $f(x)$ is padded by a variable in the transformed function $g(y)$, while every local minimizer (or maximizer) point of $f(x)$ is transformed to a local minimizer (maximizer) point of the function $g(y)$.*

Theorem 2 *If $h(x)$ is smooth, strictly monotonic as a function of x_i, and its range is equal to $\mathbb{R}$, then for every local minimizer (or maximizer) point y^* of the transformed function $g(y)$ there exists an x^* such that y^* is the transform of x^*, and x^* is a local minimizer (maximizer) point of $f(x)$.*

The same paper suggested a method to find the suitable substitution expressions in the Assertions 1 and 2 of [8]. This method requires the $\partial f(\boldsymbol{x})/\partial x_i$ partial derivatives, it will factorize them, and find the suitable substitution expressions on the basis of the factors.

Assertion 1 If a variable x_i appears everywhere in the expression of a smooth function $f(x)$ in a term $h(x)$, then the partial derivative $\partial f(x)/\partial x_i$ can be written in the form $(\partial h(x)/\partial x_i)\, p(x)$, where $p(x)$ is continuously differentiable.

Assertion 2 If the variables x_i and x_j appear everywhere in the expression of a smooth function $f(x)$ in a term $h(x)$, then the partial derivatives $\partial f(x)/\partial x_i$ and $\partial f(x)/\partial x_j$ can be factorized in the forms $(\partial h(x)/\partial x_i)\, p(x)$ and $\left(\partial h(x)/\partial x_j\right) q(x)$, respectively, and $p(x) = q(x)$.

In the case when $\partial f(\boldsymbol{x})/\partial x_i$ cannot be factorized, then only such proper substitutions can be applied for which the expression is linear in x_i.

On the basis of the mentioned theoretical results, we can write a computational program that is capable to automatically find proper substitutions for the simplification of unconstrained optimization problems. This implementation has the following steps.

1. Determine the gradient of the objective function.
2. Factorize the partial derivatives.
3. Collect the subexpressions of the proper substitutions for x_i in the list l_i:
   (a) Initialize l_i by the empty set.
   (b) If the factorization of $\partial f(\boldsymbol{x})/\partial x_i$ was successful, then add to the list l_i the integrals of the factors.
   (c) Complete the list l_i with the subexpressions of $f(\boldsymbol{x})$ that are linear in x_i.
   (d) Delete those elements of l_i, that do not fulfil the necessary conditions of proper substitutions (the expressions in l_i have to be monotonous in x_i).
4. Compose a list S of proper substitutions of $f(\boldsymbol{x})$ $S = \bigcup l_i$, $i = 1, \ldots, n$.
5. Select that element from S, which gives the least computational complexity simplified objective function.
6. Solve the simplified objective function minimization problem (if possible).
7. Determine the solution of the original problem by applying the inverse transformation.

The majority of the steps required by the algorithm (partial differentiation, factorization, symbolic integration and substitution) are directly available in computer algebra systems. On the other hand, our first implementation in Maple showed that even market leading computer algebra systems can be error prone regarding their substitution and calculation with infinite intervals capabilities [4].

The monotonicity of a function can easily be checked by interval calculations. An $f : \mathbb{R}^n \to \mathbb{R}$ function is monotonous exactly when for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$, $\boldsymbol{x} \leq \boldsymbol{y}$ imply that $f(\boldsymbol{x}) \leq f(\boldsymbol{y})$ ($f(\boldsymbol{x}) \geq f(\boldsymbol{y})$) holds. This can be checked by calculating the range of the derivative of f: if it does not contains zero, then the function f is monotonous. In our algorithm we have to check whether it holds for the substitution expression $h_i(\boldsymbol{x})$ the partial derivative $\partial h_i(\boldsymbol{x})/\partial x_i$ cannot have the zero value. If the derivative of our objective function is monotonous in one of its variables, then the objective function of the related unconstrained optimization problem is unimodal, in other words it can only have a single extremum.

We mention that the recognition of the monotonicity of a function can be hardened by the overestimation property of interval calculations, since the obtained lower and upper bounds are typically not sharp.

3 Implementation and computational results

The Maple implementation introduced in [4] had some weaknesses with the substitution, so we have reimplemented the procedure in Mathematica [3]. Mathematica has some advantages [25]. First, the substitutions required by our procedure work better, since the programming in Mathematica is based on term rewriting [16]. More precisely, the substitution routine of Mathematica can be controlled by regular expressions, and what is more, these rules, defined by the users, have higher degree of precedence compared to the system built in tools. [12].

Our special substitution subroutine is a complex procedure of ca. 50 lines, what tells a lot for those who know the elegant, expressive language of Mathematica. A dozen delayed rules were introduced, that are evaluated in four different way, involving the simplified, expanded, and factorized forms of our expression. Probably it is the most important part of our algorithm, since it is called several times in different phases, and in this way, that result of the simplification procedure can be improved.

Mathematica has also better interval arithmetic implementation: this was crucial for quick and reliable range calculation on the expressions to be substituted. Naive interval inclusion for the enclosure of the ranges have been realized with the standard range arithmetic of Mathematica.

The new version supports the listing of all the possible substitutions too, and the selection of the best fitting one for the Steps 4 and 5 in our algorithm. The running time is compatible with the earlier simple Maple version that selected in a greedy way the first proper substitution. This is due to the functional programming paradigm offered by both computer algebra systems, but this feature is more useful again in Mathematica [13, 24]. Mathematica has a parallelization scheme for the list manipulation. Still, a branch-and-bound procedure applied on the search space of all possible proper substitutions could speed up the algorithm.

3.1 Improvements compared to the previous version

The new version was first tested on our own test set, especially on those which could not be solved correctly by the earlier version. The results of the new version for these problem cases are comprised in Table 1. The results for all other problems were the same for the algorithm versions. For the sake of better readability, we have not listed the simple renaming ($y_j := x_j$).

Notations: Id: identifier of the problem, PT: problem type, RT: result type. The meaning of the last two qualifiers: we define the type of the problem A if simplifying transformations can be given in accordance with the introduced theory. The problem type is D if we do not expect to have any useful transformations. The type of the result shows whether our procedure provided a correct result (1), it has not suggested any substitutions (2). This categorization was more detailed in our earlier publications on the subject [2–4].

Table 1 *The results of the new, Mathematica based algorithm for those problem instances which were not correctly solved earlier. Abbreviations: Id identifier of the problem, PT problem type, RT result type.*

Id	Function f	Function g	Substitutions	PT	RT
Sin2	$2x_3 \cdot \sin(2x_1 + x_2)$	$2x_3 \sin(y_1)$	$y_1 = 2x_1 + x_2$	A	1
Exp1	$e^{x_1+x_2}$	e^{y_1}	$y_1 = x_1 + x_2$	A	1
Exp2	$2e^{x_1+x_2}$	$2e^{y_1}$	$y_1 = x_1 + x_2$	A	1
Sq1	$x_1^2 x_2^2$	nothing	nothing	D	2
Sq2	$(x_1x_2 + x_3)^2$	y_1^2	$y_1 = x_1x_2 + x_3$	A	1
SqCos1	$(x_1x_2 + x_3)^2 - \cos(x_1x_2)$	$y_1^2 - \cos(x_1x_2)$	$y_1 = x_1x_2 + x_3$	A	1
SqExp2	$(x_1 + x_2)^2 + 2e^1 e^{x_1+x_2}$	$y_1^2 + 2e^{1+y_1}$	$y_1 = x_1 + x_2$	A	1
SqExp3	$(x_1 + x_2)^2 + 2e^{1+x_1+x_2}$	$y_1^2 + 2e^{1+y_1}$	$y_1 = x_1 + x_2$	A	1

Because of the anomalies caused by the interval arithmetic of Maple we have implemented a heuristic in the earlier implementation for the estimation of the range bounds. The found other errors were caused by the weak substitution routine. In the following, we discuss the details on the differences between the implementations. For the *Sin2* problem the new algorithm has found a proper substitution, while the old method found a more complex, but not monotonous substitution expression.

For the function *Exp2* the range of the expression $e^{x_1+x_2}$ is not the full set of real numbers, still the heuristic range estimation routine of our earlier implementation found that it is a proper substitution. Our new method's range bounding routine worked better.

In the case of the *Sq1* problem the earlier implementation has not recognized in $x_1^2x_2^2$ the x_1x_2 proper substitution expression, although for *Sq2* it has found $x_1x_2 + x_3$ in the squared term. Since the latter is in the upper level of addition type in stead of multiplication, the representation is different. For the Mathematica implementation the case is similar, still the special subroutine that makes use of unique substitution rules was successful for this problem instance. On the other hand, x_1x_2 is not monotonous neither in the variable x_1, nor in x_2 for the whole search space (considered to be $\mathbb{R}$), this is why it was not considered to be a proper substitution expression,

In a similar way for the test problem *SqCos1* the new, Mathematica based procedure recognized correctly that $y_1 = x_1x_2$ is not monotonous, and hence it has removed this expression from the list of possible proper substitution expressions.

In the case of *SqExp2-3* again the sample fitting capabilities of the Maple based algorithm were weak, since x_1+x_2 was recognized as proper substitution expression in $e^1e^{x_1+x_2}$, but not in $e^{1+x_1+x_2}$. The recent algorithm does not produce this error.

Summarizing our finding, we can state that the new, Mathematica based algorithm showed substantial improvements compared to the old one, and it is functioning more or less as good as it can be expected.

3.2 Computational results on global optimization test problems

Here we considered the Mathematica implementation on a slightly extended test set compared to the publication [4] that was discussed in detail in [3]. Most of the obtained results were the same as those obtained with the first implementation. The two differing cases were those on the two dimensional versions of the test problems *Schwefel-227 (Sch227)* and *Schwefel-32 (Sch32)*. For the *Schwefel-227* problem Maple version gave the $y_1 = x_1^2 + x_2^2 - 2x_1$ substitution. This substitution obviously covers the variable x_2, but is not monotonous in either variable. This is why the Mathematica version has not suggested it for substitution. The *Schwefel-32* problem is similar to the Rosenbrock function, and our new algorithm was able to find a proper substitution, while the old one could not do that.

We have measured also the computation time necessary for the transformation. The running time was limited to half an hour. The numerical tests were completed by using Mathematica 9.0, on a 8 GB RAM computer with 64 bit operations system.

Most of the running times were necessary for the transformation with the symbolic algorithm. Although the preprocessing for the problems in Table 1 required below 0.2 seconds, the computation time needed for the standard global optimization test problems was much higher. Out of the 45 cases 24 required less than a second. Further 10 cases were simplified within a minute, but 7 cases would needed more than half an hour.

Out of the 45 studied global optimization problems our Mathematica based algorithm has found equivalent transformations that simplified the given problem in 8 cases. In other words, in 18% of the cases we obtained a better formulation. Since there exists no similar technique to the authors knowledge that would be capable for such automatic simplifications, these results can be regarded as remarkable [22].

We have studied what were the effect of the transformations for a classic multistart global optimization method. On average, the number of objective function evaluations decreased by 32.0%. In the case of our designed test problems, this indicator was batter, 51.8%, while for the standard problems it was slightly less, just 14.7%. For the average of the running time the simplified form allowed a quicker completion for the GLOBAL algorithm [7]. For the whole test set, the running time improvement was 31.5%, while for our designed problems we obtained 56.9% less running times, and for the standard global optimization problems 9.3%. For all the test results please keep in mind that the test problems are typically quick to evaluate, and simple expressions, so for real live problems we can anticipated better improvements in function calls and running times.

3.3 Improvement of interval inclusion functions

Interval arithmetic based methods [1] are popular for problems when the reliable solution of a global optimization problem is needed. The computational tools like Intlab, the interval package of Matlab, or high level programming languages having extensions to calculate both the interval inclusion functions and the derivatives by automatic differentiation (e.g. C-XSC) are capable to widen the set of possible applications. Still, drawbacks of interval calculations, such as the overestimation due to the dependency problem can increase the computation times for finding the optimal solutions.

An often used frightening example for the overestimation is the inclusion function of $f(x) = x^2 - x$ obtained by naive interval extension. Consider the interval $X = [0, 1]$. The range of $f(x)$ on this interval is obviously $f(X) = [-0.25, 0]$, since the zero and 1 are the zeros of the function. The minimum of $f(x)$ is attained in 0.5, and its value is -0.25. However, the inclusion function of $f(x)$ is $F(X) = X \cdot X - X$. If we evaluate it on the interval [0,1], we obtain $[0, 1] \cdot [0, 1] - [0, 1] = [0, 1] - [0, 1] = [-1, 1]$. This interval is eight times wider than the range of $f(x)$ on the interval [0,1]! We have just two operations in the function, and obviously for more complex functions we can expect worse bounding of the respective ranges.

Without going into details we can explain it shortly that the problem is caused by our assumption that the argument intervals of an operation are independent in the sense that all possible values of both argument intervals should be contained in the result interval. If the argument intervals are not independent, i.e. with the selection of a real number in the first interval we also determine the only real in the second interval that should be considered, then the calculation rules of interval operations (based on the lower and upper bounds of the intervals) are not precise any more in mathematical sense. In our example above, the two argument intervals of the subtraction are not independent. According to the rule of thumb for interval calculations, our result will be sharp mostly if we have a SUE type expression (single use expression), which uses every variable only once during calculation. In practical problems we usually cannot assume that the functions to be evaluated are of SUE type. For example large scale modelling of process network synthesis problems [9] lead to complex computation trees having redundant subexpressions, far from the SUE requirements.

The method described in our present article is aiming basically to decrease the number of operations necessary for the calculation of the objective function, and if possible to recognize the redundancies on the original expression. It is not exactly what is necessary to decrease the overestimation of interval evaluations of the respective inclusion functions [6, 23]. Still, as a first step to improve the computational burden involved in interval based optimization, we study the effect of nonlinear simplifying transformations on the solution of global optimization problems by interval methods. The tested algorithm

[18, 19] is a sophisticated branch-and-bound method extended by the interval Newton step. The stopping criterion required at least two precise decimal digits in the solution.

Consider first the results obtained for the well known Rosenbrock (or banana) function. Its objective function is $(1-x)^2 + 100(x^2-y)^2$, and the single minimum point is obviously $(1,1)^T$ with zero objective function value. The simplification should have spared just a negligible amount of computation, since the difference between the two expressions is not substantial in terms of required number of operations. We obtained for the original function:

```
Function name: ros2

The set of global minimizers is located in the union of the following boxes:

c1: [0.99487304687500, 1.00235210730573] [0.98876953125000, 1.00708007812500]

c2: [1.00271012643331, 1.00488165028086] [1.00097656250000, 1.01318359375000]

The global minimum is enclosed in: [0.000000000000000000, 0.000022260135021864]

Statistics:

   Iter    Feval    Geval    Heval     MLL  Time(sec)

    36      251      174       15      10       0.94
```

We can state that the result is correct, the interval c1 contains the minimizer point, and the obtained relative precision is good too: 3-4 digits regarding the place and similar in the objective function value as well. Let us see the output obtained for the simplified problem:

```
Function name: ros2v

The set of global minimizers is located in the union of the following boxes:

c1: [-0.00000000000000, 0.00000000000000] [-0.00000000000000, 0.00000000000000]

The global minimum is enclosed in:

[0.000000000000000000, 0.000000000000000000]

Statistics:

   Iter    Feval    Geval    Heval     MLL  Time(sec)

      9       42       31        1       1       0.11
```

Note that you can see here the solution of the transformed problem, hence the minimizer point found at zero is correct together with the minimum value. The obtained precision value is so good that it cannot even be compared to that of the original problem. Knowing how Matlab print the results, we can state that although the given intervals differ from [0, 0], still the difference is so small that it cannot be printed even with the longer printing format.

Any way, our result obtained for the transformed problem is about ten order of magnitude more precise. What is more, we have reached this result with less computational efforts, since every efficiency indicator is now better: in stead of 36 iterations, now only 9 was enough (iter). Each of the number of function evaluation (Feval), gradient calls (Geval), and also the Hesse matrix evaluations (Heval) decreased: from 251 to 42, from 174 to 31, and from 15 to 1, respectively. It is very telling that the maximal number of subintervals in the working list (MLL) decreased as well: from 10 to 1. This is indicating that the optimizer algorithm has found the simplified problem the easiest possible. The decrease in the necessary CPU time is in accordance with the above, but it has less importance, since for real life problems better the earlier efficiency indicators count. Summarizing the obtained results, the simplification brought more than substantial improvements in all efficiency factors.

Let us see now a second example standard global optimization problem in detail, the Levy-10.

```
Function name: L10

The set of global minimizers is located in the union of the following boxes:
c1: [ 0.99999993564181, 1.00000006433416 ][ 0.96591507795486, 1.04004632763272 ]
    [ 0.99366780273703, 1.12761120490455 ][ 0.93750000000000, 1.19587646550836 ]
    [ 0.89381419541709, 1.00189285440539 ]

The global minimum is enclosed in:
[ 0.000000000000000000, 0.000567456706858098 ]


Statistics:
   Iter   Feval   Geval   Heval     MLL  Time(sec)
     17     127      85       6      17       3.20
```

The result obtained for the simplified version of the objective function:

```
Function name: L10v

The set of global minimizers is located in the union of the following boxes:
c1: [ 0.00000000000000, 0.00077514614094 ][ -0.01031504005235, 0.02046607537265 ]
    [ -0.04137133829842, 0.01705999539191 ][ -0.04080756533673, 0.04296875000000 ]
    [ -0.02127534259525, 0.01985113485206 ]

The global minimum is enclosed in:
[ 0.000000000000000000, 0.000000000000000000 ]


Statistics:
   Iter   Feval   Geval   Heval     MLL  Time(sec)
    258    2130    1651     133      32      51.73
```

This time all the efficiency indicators become worse for the transformed problem, we used one-two order of magnitude more computational resources. In spite of this, we can be satisfied, since although the stopping criterion was the same, we obtained much more precise result. If we calculate with the achieved precision, the number of objective function evaluations etc. decreased

Table 2 *Computational results for the interval global optimization procedure ([19]), average of* 10 *runs.*

Id.	Rel. size of min. point (%)	Rel. size of min. (%)	Run time (s)
Br	$6{,}411\,4 \cdot 10^{-004}$	$1{,}405\,9 \cdot 10^{-011}$	1,530 4
Brv	$0{,}000\,0 \cdot 10^{+000}$	$8{,}734\,9 \cdot 10^{-008}$	0,424 3
L8	$6{,}368\,0 \cdot 10^{-010}$	$6{,}457\,8 \cdot 10^{-008}$	1,049 9
L8v	$1{,}204\,9 \cdot 10^{-005}$	$1{,}929\,3 \cdot 10^{-031}$	4,279 1
L9	$1{,}441\,4 \cdot 10^{-009}$	$5{,}304\,9 \cdot 10^{-008}$	1,848 6
L9v	$1{,}357\,0 \cdot 10^{-007}$	$3{,}507\,8 \cdot 10^{-032}$	16,662 0
L10	$1{,}115\,1 \cdot 10^{-015}$	$1{,}773\,3 \cdot 10^{-008}$	3,007 7
L10v	$9{,}544\,3 \cdot 10^{-011}$	$6{,}377\,8 \cdot 10^{-033}$	51,223 0
L11	$3{,}711\,0 \cdot 10^{-030}$	$5{,}552\,2 \cdot 10^{-013}$	7,307 1
L11v	$4{,}044\,0 \cdot 10^{-015}$	$3{,}833\,4 \cdot 10^{-035}$	1 025,000 0
Rb2	$7{,}362\,8 \cdot 10^{-007}$	$5{,}565\,0 \cdot 10^{-008}$	0,922 0
Rb2v	$0{,}000\,0 \cdot 10^{+000}$	$3{,}867\,0 \cdot 10^{-033}$	0,112 3
Rb5	$1{,}162\,2 \cdot 10^{-011}$	$5{,}690\,1 \cdot 10^{-006}$	60,372 0
Rb5v	$6{,}221\,4 \cdot 10^{-091}$	$2{,}263\,7 \cdot 10^{-007}$	36,840 0
Sch3.2	$9{,}327\,4 \cdot 10^{-003}$	$4{,}442\,0 \cdot 10^{-004}$	0,321 4
Sch3.2v	$0{,}000\,0 \cdot 10^{+000}$	$0{,}000\,0 \cdot 10^{+000}$	0,101 4

substantially. Probably the reason behind is that the simplified expression was preferable for the interval Newton step.

This finding is also supported by the results on the other, here not detailed test problems, see Table 2. The interval Newton method is the most critical part of the interval global, it can be very efficient together with other acceleration tools such as the monotonicity test. On the other hand it can also require too much computation resources that can be unjustified by the uncertainty decrease achieved.

Table 2 contains the computational test results obtained for all those standard global optimization problems that could be simplified. Here Id. stands for the identifier of the test problem, Rel. size of min. point (%) for the relative size of the interval box we have obtained for the minimizer point (in percentages), Rel. size of min. (%) indicates the relative size of the interval bounding the minimum value in percentages, and Run time (s) stands for the measured running time in seconds. The latter has been measured ten times, and the average of these values can be found in the last column.

The overall picture is similar to that detailed for the two examples before. Evaluating the results in Table 2 we can state, that in every cases when the simplification transformation was found at least one of the three major qualifying factors: the precision of the place, the value of the minimum, and the computation time was improved. In many cases all the three indicators were

better after the symbolic transformation. It should be noted that the improvements in the place or the value of the minima were not only tens of percentages, but as a rule many order of magnitudes.

We could identify some patterns for the achieved acceleration: For the Levy functions the improvement comes from the relocation of the solution inside the search box to the center of it. The Rosenbrock test functions have been transformed to a quadratic function, hence the interval Newton step could be extremely efficient. The underlying reason behind the other cases is still unclear.

4 Conclusion

On the basis of our present work we can draw the cautious conclusion that the symbolic simplification transformation showed a very promising efficiency improvements on standard global optimization problems. Closely 18% of the standard global optimization problems could be simplified automatically, and we could achieve dramatic improvements of many orders of magnitude in precision of the place or value of the minima. At present, the main problem with these improvements are that they are less controllable, i.e. we cannot really economize on the huge precision improvements. Further developments need extensive testing, and the deeper understanding of the causes behind the better bounds.

Declarations

The authors declare that they have no conflict of interest and no competing interests related to this work. We declare that we have not received any payment or support in kind for any aspect of the submitted project.

References

[1] G. Alefeld, J. Herzberger, Introduction to Interval Computation, Academic Press, New York, 1983.

[2] E.D. Antal, A matematikai modellezés hatása nemlineáris optimalizálási feladatok megoldásának hatékonyságára, PhD Dissertation, University of Szeged, Szeged, 2017.

[3] E.D. Antal, T. Csendes, Nonlinear Symbolic Transformations for Simplifying Optimization Problems, Acta Cybernetica 22(2016) 715–733.

[4] E.D. Antal, T. Csendes, J. Virágh, Nonlinear Transformations for the Simplification of Unconstrained Nonlinear Optimization Problems, Central European J. Operations Research 21(2013) 665–684.

[5] G. Avanzolini, P. Barbini, Comment on "Estimating Respiratory Mechanical Parameters in Parallel Compartment Models", IEEE Trans. Biomedical Engineering 29(1982) 772–774.

[6] T. Csendes and E.D. Antal, Nonlinear Symbolic Transformations for Optimization Problems (In Hungarian), Szigma 48(2017) 33-46.

[7] T. Csendes, L. Pál, J.O.H. Sendín, J.R. Banga, The GLOBAL Optimization Method Revisited, Optimization Letters 2(2008) 445–454.

[8] T. Csendes, T. Rapcsák, Nonlinear Coordinate Transformations for Unconstrained Optimization. I. Basic Transformations, J. Global Optimization 3(1993) 213–221.

[9] T. Farkas, E. Rév, and Z. Lelkes, Process flowsheet superstructures: Structural multiplicity and redundancy: Part I: Basic GDP and MINLP representations, Computers & Chemical Engineering, 29(2005) 2180–2197.

[10] R. Fourer, and D.M. Gay, Experience with a Primal Presolve Algorithm. In W.W. Hager, D.W. Hearn, and P.M. Pardalos, editors, Large Scale Optimization: State of the Art, pages 135–154. Kluwer Academic Publishers, Dordrecht, 1994.

[11] D.M. Gay, Symbolic-Algebraic Computations in a Modeling Language for Mathematical Programming, In G. Alefeld, J. Rohn, and T. Yamamoto, editors. Symbolic Algebraic Methods and Verification Methods, pages 99–106. Springer-Verlag, Berlin, 2001.

[12] R.J. Gaylord, C. Kamin, S.N. Wellin, and R. Paul, An Introduction to Programming with Mathematica, Springer New York, 1996, 141–143.

[13] A. Heck, Introduction to computer algebra, In Introduction to Maple, Springer New York, 2003, 11.

[14] Z. Hantos, B. Daróczy, T. Csendes, B. Suki, and S. Nagy, Modeling of Low-frequency Pulmonary Impedance in the Dog, J. of Applied Physiology 68(1990) 849–860.

[15] L. Liberti, S. Cafieri, D. Savourey, The Reformulation-Optimization Software Engine, Mathematical Software – ICMS 2010, LNCS 6327, 303–314, 2010.

[16] R.E. Maeder, The Mathematica Programmer, Academic Press, 1994, 8.

[17] Cs. Mészáros, U.H. Suhl, Advanced preprocessing techniques for linear and quadratic programming, OR Spectrum 25(2003) 575–595.

[18] L. Pál, Global optimization algorithms for bound constrained problems. Ph.D. dissertation, University of Szeged, 2011.

[19] L. Pál and T. Csendes, INTLAB implementation of an interval global optimization algorithm, Optimization Methods and Software 24(2009) 749–759.

[20] T. Rapcsák and T. Csendes, Nonlinear Coordinate Transformations for Unconstrained Optimization. II. Theoretical Background, J. Global Optimization 3(1993) 359–375.

[21] H. Schichl, A. Neumaier, Interval Analysis on Directed Acyclic Graphs for Global Optimization, J. Global Optimization 33(2005) 541–562.

[22] D.R. Stoutemyer, Ten commandments for good default expression simplification, J. Symbolic Computation 46(2011) 859–887.

[23] B. Tóth, T. Csendes, Empirical investigation of the convergence speed of inclusion functions, Reliable Computing 11(2005) 253–273.

[24] P. Wellin, Functional programming. In Programming with Mathematica: An Introduction, Cambridge University Press, 2014, 115–188.

[25] Wolfram Mathematica 9 Documentation Center, Mathematica Tutorial: Basic Internal Architecture, available at: https://reference.wolfram.com/mathematica/tutorial/BasicInternalArchitecture.html