NOTICE WARNING CONCERNING COPYRIGHT RESTRICTIONS:
The copyright law of the United States (title 17, U.S. Code) governs the making of photocopies or other reproductions of copyrighted material. Any copying of this document without permission of its author may be prohibited by law.

An Intelligent Modelling Interface for Design Optimization

by

Regis J. Amarger, Larry T. Biegler, Ignacio E. Grossmann

EDRC 06-95-90

AN INTELLIGENT MODELLING INTERFACE FOR DESIGN OPTIMIZATION

by

Regis J. Amarger
Ignacio E. Grossmann
Lorenz T. Biegler

Engineering Design Research Center
and Department of Chemical Engineering
Carnegie Mellon University
Pittsburgh, PA 15213

November 1990

University Libraries
Carnegie MGHOK University
Pittsburgh PA 15213-3890

ABSTRACT :

Optimization of engineering problems relies on the use of modern softwares for solving algebraic models, including NLP, MILP and MINLP models. Although a number of powerful optimization packages are currently available, it is still far from being trivial for general design engineers to properly formulate optimization models. Since a qualitative knowledge base by experts in optimization does exist for modelling, the objective of our work is to develop an interface written in LISP to encode this knowledge to reformulate the model so as to increase the robustness and the efficiency of algorithms for nonlinear and integer problems. In this work, models specified in GAMS input files are automatically preprocessed by the interface which generates a new well-formulated model. Capabilities include constraint reformulation to induce linearity and convexity, tightening of bounds, scaling as well as fixing 0-1 variables a priori. Computational results are presented to illustrate how the proposed interface can reformulate poor models to increase both the robustness and the efficiency of the optimization methods.

INTRODUCTION :

Mathematical programming techniques are increasingly being used for design optimization. The extensive research conducted over the past forty years has led to the development of algorithms that are capable of solving progressively larger and more difficult problems. This task has been facilitated by the increased power and speed of computers.

Most of the research in mathematical programming has been devoted to linear programming (LP), largely for two reasons. Firstly, linear models arise frequently in problems of manufacturing, banking, management, and other decision making domains. Secondly, they constitute the class of mathematical programming problems which is simplest to solve. The different implementations of the simplex algorithm and the new interior point methods are quite efficient and the packages available on the market are robust.

More recently, research has focused on methods for solving mixed integer linear programming (MILP) and nonlinear programming (NLP) problems. Algorithms developed for the solution of MILP models are usually based on branch-and-bound procedures. Robustness of these methods is good in general but efficiency may vary. New techniques are still under development, in particular cutting planes techniques. NLP methods such as reduced gradient [Wolfe 1962] (e.g. MINOS) and SQP [9] [20] have made good progress but their robustness and efficiency are not always satisfactory.

One of the most challenging research areas in mathematical programming today is mixed integer nonlinear programming (MINLP). Software packages such as GAMS/DICOPT++ are already emerging but reliability and efficiency are still important limitations.

From the above discussion one can draw the general conclusion that nonlinear and integer mathematical programming techniques still lack the degree of robustness and efficiency that has been achieved for linear programming. Although algorithmic improvements can be expected in the future to circumvent some of these problems, another equally important aspect to be considered is problem formulation. Currently, many of the failures experienced by design engineers with nonlinear and integer optimization techniques can be attributed to the fact that engineers are often not experts in mathematical programming and that the methods are still quite sensitive to the form of the model. Therefore the motivation of this work was to develop a tool that can help engineers to automatically reformulate "well-behaved" optimization models by making use of common rules and heuristics that are used by experts in mathematical programming in order to increase the efficiency and robustness of a method.

IMPACT OF FORMULATION :

The manner in which NLP, MILP and MINLP problems are formulated can have a great impact on the robustness and performance of the algorithm used for solving. Model designers know that in practice it is likely that they will need to modify the formulation of their model several times before successfully solving it. In this they are aided by various rules used by experts in optimization which constitute a qualitative knowledge base for the formulation of mathematical programming problems.

The scope of our work is therefore to analyze what rules and heuristics should be applied to obtain "good** models and then to develop an interface that will automatically preprocess a given design optimization model and reformulate it according to these rules. We define a ^{H}good^{H} model as one where the formulation of the equations increases the robustness of the algorithm and possibly the efficiency of finding the solution.

The optimization platform used in our work is GAMS (General Algebraic Modeling System) [3] which is in fact a language that can interact with different solvers. The solvers that can be interfaced with GAMS include ZOOM, SCICONIC and MPSX for LP and MILP; MINOS and SQP for NLP; and DICOPT++ [12] [23] for MINLP. GAMS is one of the most advanced commercial packages for NLP/MILP/MINLP models in equation form.

Our interface is a front-end that parses a model specified in a GAMS input file, processes it and returns a new improved input file for GAMS with the reformulated model. The interface is written in COMMON LISP which allows symbolic manipulation of the equations and data.

Typically, the solution of a mathematical programming model consists of minimizing or maximizing an objective function

$$z := c^{T}y + f(x) \qquad (1)$$

subject to a set of constraints of the general form

$$\begin{aligned} Ay + h(x) &= 0 \\ By + g(x) &\leq 0 \qquad (2) \\ Cy + Dx &\leq 0 \end{aligned}$$

where x are continuous variables and y are binary variables

$$x \in X = \{x \in R^{n} : x^{L} \leq x \leq x^{U}\} \qquad (3)$$
$$y \in Y = \{0,1\}^{m}$$

As will be explained in this paper, many of the heuristics for the transformation or reformulation of a model may not necessarily improve the robustness and efficiency of a model. The interface must be sufficiently flexible to accomodate different decisions, possibly allowing for user intervention rather than following a strict step-by-step procedure. The interface interacts with the model designer to decide which transformations should be attempted for a particular model. Of course, a default procedure exists that will apply the standard formulation enhancement rules to the model.

The interface can be viewed as a tool to apply selected encoded transformation rules. It allows the user to generate different formulations of the same optimization model.

The next section will describe how to reformulate constraints so as to enhance the formulation. It will be followed by a section explaining a method for reducing the domain of search based on tightening of the bounds. The next section describes what can be done with mixed-integer or pure integer constraints. Finally, the last section presents selected computational results using the interface to perform various model transformations.

CONSTRAINT FORMULATION :

Most of the problems associated with poor constraint formulation can usually be avoided by following the rules stated below :

- avoid potentially undefined functions
- reduce the degree of non-linearity
- try to convexity the model

* avoid poor scaling of variables and equations

These simple rules are well known by expert designers. The novel aspect in our work is to implement them within an interface which can identify badly formulated equations and rewrite them in a more desirable form, using a fully symbolic and automated procedure.

A detailed discussion of the above rules is presented in the ensuing subsections. In particular, we show why they are necessary and how they are to be implemented.

Avoiding potentially undefined functions :

A problem frequently encountered by designers, e.g. when they run an optimization model for the first time, occurs when the algorithm attempts a function evaluation at a point where the function is not defined. This immediately causes a numerical error and the algorithm fails to converge.

The most common undefined mathematical functions in engineering design problems are :

- divisions by zero

- logarithms of non positive expressions

For the treatment of logarithms one can specify bounds on the variables so that the function is always defined. For example if the expression log(x) is used, x is restricted to being strictly positive. The difficulty with this approach however, is that it is not always clear how to specify the lower bounds. For example, if the function is given by log (x-y), x-y needs to be positive but this will depend on the individual range of values for x and y. To avoid this difficulty, our interface uses an alternative approach ; the log terms are removed by applying exponential functions.

To circumvent the problem of divisions by zero our interface automatically multiplies each equation by the denominator(s) of the division(s). Using slack variables may be necessary in case the sign of the expression multiplying the inequality for reformulation is not fixed a priori by the domain of the variables. The use of slack variables is, however, usually not advised since it increases the degree of complexity of the model.

The following example illustrates the method :

Given log (x / (y-z)) <.d, this inequality is transformed into x - (y-z) * exp (d) £ 0.

In this way the constraint is defined for any value of x, y and z, and is linear when d is constant.

There are of course a number of other potentially undefined functions. For instance, consider the Logarithmic Mean Temperature Differences (LMTDs) used in chemical engineering problems for heat exchangers:

LMTD = (ATi - AT_2) / log (ATVAT2)

It is to be noted that in the case where AT1 - AT2 this expression leads to an undefined value for the LMTD. When this happens a more suitable approximation should be used. Two alternatives are :

Chen's approximation : LMTD «[(AT1) (AT2) (AT1 + AT2) / 2 J$\frac{1}{3}$

Patterson's approximation : LMTD - $\frac{-i}{6}$ * [AT1 + AT2] + $\frac{£}{3}$ $^{\#}$ [AT1 * AT2]^{os}

which are well defined for non-negative values of AT«| and AT_2.

Reducing the degree of nonlinearity :

It is well-known in nonlinear programming that problems are far easier to solve if the constraints are mostly linear Consequently it is clear that reducing the nonlinearities in the constraints is a way to improve the efficiency and possibly also the robustness of the algorithm.

Being able to symbolically linearize the constraints is not always possible, but it can be done in a few cases. A very common case is when a bound over a ratio is specified in the following manner :

$f(X) / g(X) \leq a$ where a is a numerical constant, X is a vector, and f and g are continuous functions.

The equivalent expression $f(X) - a * g(X) \leq 0$ is clearly preferable, especially when f(X) and g(X) are linear functions.

When monomial expressions containing ratios are systematically multiplied by the denominators, and when monomial logarithms are removed from expressions by taking the exponential, these expressions are transformed into a linear form. Therefore this simple algebraic device is included in our interface.

An additional technique has been implemented as part of our interface in an attempt to reduce the degree of nonlinearity of the model. It is very frequent in engineering design optimization models that several constraints contain the same nonlinear term that often expresses a variable as a function or a correlation of some other variables. Replacing such a term by a new variable can hence reduce the overall degree of nonlinearity of the model.

In our preprocessing strategy the equations are first scanned to identify the nonlinear terms. An automatic substitution is then performed on some of these terms when this replacement can reduce the number of nonlinear terms in the model by more than one half. One half is arbitrarily selected as a criterion for improvement. In the procedure, all the nonlinear terms are considered ; if the number of globally different terms is more than half the total, new variables are introduced. The limitation of our implementation is that it does not try to partially match the expressions ; for instance, it fails to recognize that the expressions $x * (x^2 - 3xy)$ and $y * (x^2 - 3xy)$ contain similar terms.

Convexifying models :

The convergence of a nonlinear optimization algorithm to the global optimum can only be guaranteed if the model is convex, i.e. when the objective function and all the constraints of the model are convex. Furthermore, convex constraints avoids the problem of infeasible or inconsistent linearizations.

In engineering design geometric programming problems that consist of posynomial expressions are quite common. This well-known class of problems can be convexified with exponential transformations. For example, the inequality

$$x^{0.6} / y + 1 / z \leq a$$

can be convexified in terms of the new variables X, Y and Z, using the transformations $x = \exp(X)$, $y = \exp(Y)$ and $z = \exp(Z)$. This leads to the convex inequality

$$\exp(0.6 X - Y) + \exp(- Z) \leq a$$

The proposed interface can automatically perform this reformulation and it also recognizes the inherent linearity of monomials, that is

$$xy/z \leq a$$

is transformed as

$$X + Y - Z \leq bg(a)$$

It should be noted that in the GAMS input file the original variables are reported to the user only through the display command after the solution of the NLP has been obtained.

In general, transforming constraints by systematically using exponential functions instead of logarithmic functions has another advantage besides helping to avoid potential numerical errors as described earlier. It can also help to convexify the constraint, completely or only partially if other non-convex functions are present.

As an example consider the inequality,

$\log(x-b) < a$ which is not convex

but $(x - b) < \exp(a)$ is convex.

Similarly, $\log(x - a) + \log(y - b) < c$

is not convex but can be advantageously replaced by the set of constraints

$$X + Y < c$$
$$(x-a) = \exp(X)$$
$$(y-b) = \exp(Y)$$

which are convex if the last two inequalities relax as

$$(x - a) \geq \exp(X)$$
$$(y-b) \geq \exp(Y)$$

<u>Using well scaled expressions :</u>

Although it is well-known that scaling has a great impact on the robustness and efficiency of an algorithm, no ideal a priori scaling strategy exists. Most mathematical programming codes have an internal scaling procedure, usually based on the values of the gradient coefficients.

An expresston is considered to be well scaled when all its elements are neither too big nor too small. As a heuristic, one can consider an expression as being reasonably scaled when lying between 0.01 and 100. Too big or too small expressions or coefficients lead to ill-conditioning in the jacobian and hessian matrices, and thus to poor step predictions.

In the interface we have implemented the following procedure that attempts to rescale both the variables and the constraints. We do not claim our scaling strategy is ideal but it seems to be reasonably acceptable and to work well. The procedure can be described by the following rules :

- scaling of the variables :

If a variable has an upper bound greater than 100, it will be rescaled using a new proportional variable whose upper bound will be 100. In a similar manner a variable with a lower bound lower than .01 will be rescaled such that the lower bound of the new variable is .01.

• scaling of the equations :

A "mean" value of the left hand side of each constraint is evaluated corresponding to a "mean" value of the vector of variables. When this mean value of the left hand side is too big (arbitrarily over 100) or too small (arbitrarily under .01) the equation is then multiplied by a scaling factor such that the new mean value of the left hand side lies between .01 and 100.

The main problem of this method is to determine a good "mean** value of a variable. Here, a "mean" value is defined by :

- if a variable has both a lower and an upper bound, take the mean value of the bounds
- if the variable has only a lower bound take 1.5 times its lower bound as a mean value
- if the variable has only an upper bound take .66 times its upper bound as a mean value
- if the variable has no bounds take 1 as a mean value

When rescaling the equation by a certain factor it is important to apply this factor as deep in the expression tree as possible and not to just globally multiply the member by this factor. The reason is that applying the factor to the coefficients of the variables induces numerical simplifications and scale changes in the intermediate function and derivative calculations, while global multiplication does not.

For example, 200*x + 400*y < 500 with x *100 and y *100
should be rescaled as 0.5*x + y < 1.25 rather than (200*x + 400*y) / 400 < 1.25

TIGHTENING OF THE BOUNDS :

An effective way to reduce the domain of search in optimization problems is to tighten the upper and tower bounds of the variables. The impact of such a space reduction is most significant when dealing with large nonlinear and mixed-integer nonlinear models. To achieve this space reduction as part of our interface, we have implemented an algorithm based on the monotonicity of the constraints. The basic idea is as follows :

For all the constraints, we select the subset of constraints which are monotonic in the variables (i.e. partial derivatives along the coordinate directions are one signed). We then determine for the variables of such constraints the "worst" value, given that the other one signed variables are placed at one of their bounds. The "worst" computed values are then updated as bounds and this information is incorporated in the analysis of the subsequent equations. A recursive cycle of iterations is performed until the change in the bounds is small or zero.

This section describes a procedure to obtain tight bounds. Further, details of some of the steps (e.g. how to analyze the monotonicity of an equation, and how to determine the critical bounds) are presented in Appendix A. It is to be noted that the procedure described in this section is not related to the one developed by Papalambros [18] which uses the monotonicity of the constraints to relax the equalities and eliminate the inactive inequalities.

General procedure :

The procedure for tightening the bounds can be discribed in three major steps : definition and construction of lists, recursive algorithm for bounds tightening and the updating of the bounds.

Step 0 : Definition of lists

Define the following lists or sets for the equations and variables :

. list_all_var
- contains all the variables in the model.

. list_eqn
- contains the monotonic equations : eqn_1, eqn_2,... eqn_k,... eqn_n.

. var_in_eqn_k
- n sublists that contain the variables appearing in each eqn_k.

. list_monotonicity
- contains n sublists monotonicity_in_eqn_k.

. monotonicity_in_eqn_k
- contains type of monotonicity (+/-) for each variable in eqn_k.

. critical_bounds_in_eqn_k
- contains critical bound type (up/lo) for each variable in eqn_k. For a $\leq$ ($\geq$) inequality, a critical bound for a variable is defined as the variable bound for which the left hand side of the inequality is minimized (maximized). Assigning all but one of the variables to their aitical bound values will allow a monotonic inequality to determine the critical bound value for the remaining variable. See part 2 of Appendix A, for bound value determination.

. list_upperjbounds
- contains values for variable upper bounds.

. listjower_bounds
- contains values for variable lower bounds.

. updated_LO
- contains variables for which lower bounds were specified at the end of the last cycle.
Initialized to list_alLvar.

. updatedJJP
- contains variables for which upper bounds were specified at the end of the last cycle
Initialized to list_alLvar.

. updated_lo
- contains variables for which lower bonds are specified (at the current cycle).
Initialized to NIL.

. updaled_up
contains varaibles for which upper bonds are specified (at the current Cycle).
Initialized to NIL.

Step 1 : Construction of lists

Using the method described in part 1 of Appendix A, process the constraints and obtain the following information:

- construct list_eqn by identifying all the monotonic constraints.
(Note that all equalities have been substitued by two inequalities)
Discard the remaining constraints and the objective function.

- construct the lists varjn_eqn_k and monotonfcity_in_eqn_k.
Initialize list_upperjx>unds and listjowerjx>unds with specified bound values.

- determine critical bounds for each variable in each eqn_k
and construct the list criticaljx>undsjor_eqn_k.

- initialize updated_UP, updated_LO, updated_up and updated_lo as described above.

Step 2 : Tightening of the bounds

```
for eqn_k,
        - determine N, the number of unspecified critical bounds for that equation.

        - if N < 2, then

                if at least one element of critical_bounds_for_eqn_k appears in updated_LO
                or updated_UP, derive new critical bounds for all variables in var_in_eqn_k,
                as described in part 3 of Appendix A.

                if the new lower (upper) bounds are significantly better than the current values,
                replace values in list_lower_bounds (list_upper_bounds)
                and enter the corresponding variables in updated_lo (updated_up).

        - proceed to next eqn_k
```

Step 3 : Updating and termination

If updated_lo or updated_up are not empty, then set updated_LO to updated_lo and updated_UP to updated_up. Go to step 2.

Else, no further bound updating is possible. Report values in list_upper_bounds and list_lower_bounds as new upper and lower variables bounds.

Furthermore using these tighter values of the bounds it is possible to derive a starting point inside the domain limited by these bounds. For inequality constrained problems, the searchspace limited by these problems is smaller, a starting point chosen inside these bounds is more likely to yield a feasible point that one chosen with the initial bounds. The interface will automatically generate a starting point by taking as an initial value of the variables the mean value of the bounds ; in case not both upper and lower bounds are available for a variable the rules presented in section B.4 for constraint scaling are used for evaluating a mean value.

Having a starting point located nearby the center of the search domain is not always a good choice as far as efficiency is concerned but again it is more likely to be inside the feasible region.

We note that when no bounds are specified for a given variable, the interface arbitrarily uses 0 and 50000 as the default lower and upper bounds for the variables respectively.

Example :

Model:

eqn_1:	y-2x-3£0
eqn_2:	x^2-4y+i£0
eqn_3:	4-xy≤0
eqn_4:	1.5x + y-10≤0
eqn_5 :	x^2-6x + y≤0
eqn_6:	profit = y

where x and y are positive continuous variables

step 1a:

eqn_5 is not monotonic ; it is hence discarded when applying the algorithm.

eqn_6 represents the objective function to be maximized ; it is also discarded.

BsLaljvar: (x y)
varjn_eqn_1 : (x y)
varjh_eqn_2: (x y)
var_in_eqn_3: (x y)
varJn_eqn_4: (x y)
nx>notonicityjn_eqn_1 : (- +)
ironotonicityjh_eqn_2 : (+ -)
monotonicity_in_eqn_3 : (- -)
monotonicityjh_eqn_4 : (+ +)

step-1b:

listjower_bounds : (0 0)
list_upper_bounds : (nil nil)

```
step 1c:
        critical_boundsjor_eqnj :  (up  lo)
        criticaLboundsJor_eqn_2 :  (lo  up)
        critical_boundsJor_eqnJ3:  (up  up)
        crtfcal_boundsjor_eqn_4:   (lo  lo)

step 2a:
        updated_LO:   (x y)
        updatedJJP:   (x y)
        updated Jo: nil
        updated_up:  nil

step 2b:
loop 1 :
        eqn_1 :  N = 1  ( N is the number of unspecified critical bounds.  See Appendix A)
                    x.up?    solve   0-2x-3≤0    ->    x≥-3  which is not interesting

        eqn_2 :  N = 1
                    y.up ?    solve   0^2 - 4y +1≤ 0    ->    y ≥ .25
                                                                  y.lo = .25
                                                                  updated_up = nil
                                                                  updated Jo = (y)

        eqn__3 :  N = 2, Too many unspecified critical bounds. Nothing further can be determined.

        eqn_4 :  N = 0
                    if   x.up?    solve   .25 + 1.5x-10≤0    ->    x≤6.5
                    if  y.up?    solve   y + 0-10≤0    ~>    y≤10
                                                                  x.up = 6.5
                                                                  y.up = 10
                                                                  updated_up = (x  y)
                                                                  updatedjo = (y)

        end of loop 1 :
                    0 ≤ x ≤ 6.5
                    .25≤y≤10
                    updatedJJP = (x  y)
                    updatedJ-0 = (y)


loop 2:
        eqnjl :  N = 0
                    if   x.up?    solve   .25-2x-3≤0    ~>    x≥-1.375   which is not interesting
                    if  y.b?    solve   y-2*6.5-3£0    ~>    y≤16  which is not as good as 10

        eqn_2:   N*0
                    if   x.lo ?    ~>    x ≤ V 39   = 6.25 which is better than 6.5
                    if   y.up?    ->    y ≥ .25

        eqn_3 :  N = 0
                    if   x.up?    ->    x^.4 which is better than 0
                    if  y.up?    ->    y£ 4/6.25 = .64   which is better than .25

        eqn_4 :  N » 0
                    if   x.to?    solve   .64 + 1.5x≤10    ->    x≤6.24
                    if  y.lo?    solve   y + 1.5\4≤10    ->    y≤9.4

        end of loop 2 :
                    .40 ≤ x £ 6.24
                    .64 ≤ y ^ 9.40
                    updatedJJP = (x  y)
                    updated_LO «(x  y)


loop 3 :
        eqnjl :  N « 0
                    if   x.up?    ->    x^-_   which is not interesting
                    if  y.lo?     ->    y^ 15.48   which is not as good as 9.4

        eqn_2 :  N « 0
                    if   x.lo ?    ->    x ≤  6.05 better than 6.24
                    if   y.up?    -->    y £ .29 not as good as .64

        eqn_3 :  N = 0
                    if   x.up?    -->    x≥.42 better than .40
```

```
            if  y.up ?    -->    y ≥ .66 better than .64

    eqn_4 :  N = 0
            if  x.lo ?    -->    x ≤ 6.23 not as good as 6.05
            if  y.lo ?    -->    y ≤ 9.37 better than 9.40

    end of loop 3 :
            .42 ≤ x ≤ 6.05
            .66 ≤ y ≤ 9.37
            updated_UP = ( x  y )
            updated_LO = ( x  y )
loop 4 :
```

the same procedure could be applied one more time without giving better values (with a .01 precision) of the bounds.

step 3 :

The final bounds are then given by

x.lo = 0.42

y.lo = 0.66

x.up = 6.04

y.up = 9.37

This simple example with two variables shows how the domain can be narrowed at each loop. *Table 1* shows the progress of the bonds at each loop.

	initial	loop 1	loop 2	loop 3	loop 4
x.lo	0	0	0.40	0.42	0.42
x.up	∞	6.5	6.24	6.05	6.04
y.lo	0	0.25	0.64	0.66	0.66
y.up	∞	10	9.40	9.37	9.37

Table 1 : Values of the bounds after each loop

Figure 1 shows in the two-dimensional space how the bounds that we have obtained for the example reduce the domain of search (rectangle). The initial search space was the entire positive quadran since neither x nor y had upper bounds. The actual feasible region is the nonshaded area. The constraints are represented by the lines with the corresponding indices.

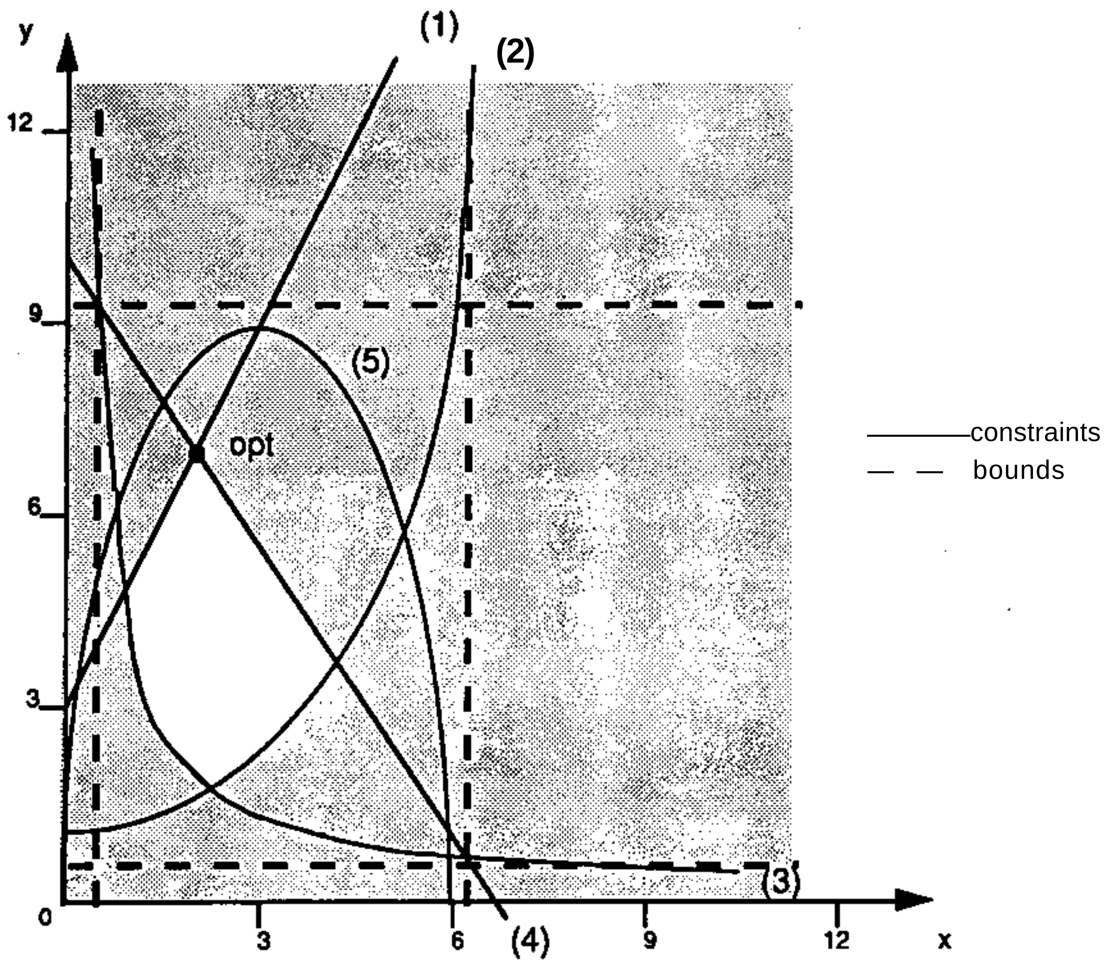


***Figure 1:* plan representation of the model**

Note that in this example our proposed method did not find the absolute smallest search space. However, finding a smaller search space would require the solution of several nonlinear programs. It is also interesting to note that when solving this problem with GAMS/MINOS and without specifying any bounds it takes 23 iterations to obtain the optimum with profit = 7. Solving the same model with the four bounds obtained above requires only 16 iterations.

Efficiency of the bounds tightening procedure :

It is to be stressed that using tighter bounds is a way to improve the robustness of the model but not necessarily the efficiency of the solving procedure. Smaller domains of search implies better robustness because the NLP algorithms used are always a variation of the Newton method [15] which is valid only nearby the optimum.

In fact after having tested numerous test problems we can draw the following conclusions :

- for linear models and mixed-integer linear models it is generally not an advantage to tighten the bounds (unless binary variables can thus be eliminated). Introducing tighter bounds on the variables is like adding new constraints to the problem. Therefore the number of vertices in the feasible region increases and hence the number of pivots required by the simplex method increases as well. Furthermore, notice that with our procedure for tightening the bounds, any new bound corresponds to the intersection of two constraints and we see that a bound applied directly at a vertex point is not only unnecessary but leads to degeneracies when solving the LP. - Here, simplex methods have difficulty in determining unique active sets and shadow prices at these points.

- On the other hand, for nonlinear models and mixed-integer nonlinear models, having tight bounds presents a great advantage as far as the robustness is concerned. When reducing the search domain by tightening the bounds the NLP algorithm can indeed find more easily a feasible starting point and it generally requires less steps to converge.

- for mixed-integer linear models and mixed-integer nonlinear models, bound tightening preprocessing can be advantageous if the algorithm is able to fix some of the binary variables. Here, the number of nodes in the trees used for the branch-and-bound procedure of the IP algorithm is exponentially increasing with the number of free binary variables. As will be explained later in this paper having an upper bound on the continuous variable of a mixed-integer constraint can allow reduction of the coefficient value of that constraint Therefore a better approximation of the corresponding binary variable when solving the relaxed problem can be found. This leads to better choices for the parent node of the tree used in the branch-and-bound procedure of the IP algorithm.

MODELS WITH BINARY VARIABLES :

Mixed-integer linear and nonlinear programming problems have the feature that it is often possible to develop markedly different formulations for the same problem, which in turn may mean the difference between making the problem solvable or not.

Some rules exists to obtain well formulated mixed-integer and pure integer constraints [10], [11], [19]. The next section presents these rules and describe how they can be implemented in our interface. It also presents the application of the bound tightening procedure for 0-1 variables, possibly fixing some of them.

Mixed-Integer and pure integer constraints :

When dealing with some pure integer constraints it is desirable to make ad hoc improvements to formulation of the model applying the few rules developed below. These rules tend to eliminate inactive constraints, to rewrite other equations in a simple and logical linear form and to reduce the coefficients used in the equations. It has to be noted at this point that unlike the LP problems the addition of new constraints to a MILP or a MINLP problem will in general reduce the computational effort, especially when the new constraints are purely integer variables.

In the case of a type one special ordered set, if we have the constraint

2 >yj E_ ak yk ≤ a0 0)

where N+ and N- are two disjoint sets of indices and where all the a_j and a_k are strictly positive constants, then the following preprocessing operations can be performed [16] (they are not yet implemented in our interface)

if - £ a< > ao then (1) is infeasible : error in modelling
N+

I £ 3 <-ao then (1) is inactive and can be removed from the model

if 3 > ao+X * then M can be *ixed t0 o and dropped from the model
ft

if - an > ao + £ 3 then yk can be fixed to 1 and dropped from the model

Another treatment can be performed over the pure integer constraints known as the coefficient subset reduction which replaces the equation X a^ <, a_Q with an equivalent equation X a'jX; £ a_Q where the a'j s are smaller than the initial a; s. These coefficient reductions are more important than it may seem since they provide with a tighter constraint when solving the LP or NLP subproblems. Despite the interest that this method presents, it has not been incorporated in our preprocessing package.

A particular treatment can applied to the mixed-integer equations of the form

x - A * y ≤ 0 where x is a continuous variable, y is a binary variable and A is an upper bound.

One should try to chose A as small as possible, but still maintaining it as a valid upper bound for x. The procedure in the interlace is implemented as follows. After we attempt to tighten the bounds, the equation of the type mentioned above are selected and rewritten of the form x - A1 * y £ 0 where At is the new upper bound of x, if it exists.

The procedure has also been extended to the case where x is not a variable but a linear combination of continuous variables. For example x1 - x2 - A * y ≤ 0 is replaced by x1 - x2 - (min[A, x1 .up - x2.lo]) * y ≤ 0.

Preprocessing of the binary variables :

As for the possible fixing of 0-1 variables, this is done when the interface tries to tighten the bounds as it includes the binary variables to the set of continuous variables having 0 and 1 as lower and upper bounds respectively. If the new bounds are tighter than 0 and 1, the value of the corresponding binary variable can be fixed. For instance, if the bound tightening procedure gives y1.lo=0.2 and y2.up=0.9 then y1 can be fixed to 1 and y2 to 0. This method can therefore reduce the number of 0-1 variables and thus reduce the branch and bounds enumeration effort.

COMPUTATIONAL EXPERIMENTS :

Our interface is still under development. In particular it does not yet handle indexed equations in GAMS, which makes it difficult to test it on large models. However, its application to small test problems has produced some interesting results as will be shown below.

In this section three types of examples will be presented : one illustrating the constraint reformulation and scaling procedure, one presenting the exponential transformation of a geometric model, one concerning bounds tightening and binary fixing for MILP models, one showing bounds tightening on MINLP models.

Alkylation problem :

This problem was first presented by Bracken and McCormick (1968). The design optimization of an alkylation unit can be modelled as follows :

$$\begin{aligned}
\text{profit} &= 0.063x_4 x_7 - 5.04x_1 - 0.035x_2 - 10x_3 - 3.36x_5 \\
x_4 &= x_1 * (1.12 + 0.13167x_8 - 0.00667x_8^2) \\
x_7 &= 86.35 + 1.098x_8 - 0.038x_8^2 + 0.325 * (x_6 - 89) \\
x_9 &= 35.82 - 0.222x_{10} \\
x_{10} &= 3x_7 - 133 \\
x_8 &= (x_2 + x_5) / x_1 \qquad (\#) \\
x_5 &= 1.22x_4 - x_1 \\
x_6 &= 98000x_3 / (x_4 x_9 + 1000x_3) \qquad (*)
\end{aligned}$$

where $x_1, x_2, \dots x_{10}$ are continuous variables and profit is the variable to be maximized.

According to physical considerations the variables bounds are as follows :

Lower bounds		Upper bounds	
x_1	: 0	x_1	: 2000
x_2	: 0	x_2	: 16000
x_3	: 0	x_3	: 120
x_4	: 0	x_4	: 5000
x_5	: 0	x_5	: -
x_6	: 85	x_6	: 93
x_7	: 90	x_7	: 95
x_8	: 3	x_8	: 12
x_9	: 1.2	x_9	: 4
x_{10}	: 145	x_{10}	: 162

This problem has its global optimum at the point:

Solution point	
x_1	: 2000
x_2	: 16000
x_3	: 118.15
x_4	: 3532.8
x_5	: 2310.0
x_6	: 89.6
x_7	: 93.4
x_8	: 9.15
x_9	: 3.131
x_{10}	: 147.25

1) When trying to solve the model (M) in its original formulation presented above a numerical failure of the GAMS/MINOS solver is experienced.

2) **If the model (M)** is preprocessed through our interface for an attempt to reformulate the constraints, then the two constraints that have been marked when presenting the initial model are automatically reformulated as follows:

$$x_1 * x_1 - x_2 + x_5 \qquad (\#\#)$$

$$x_6 * (x_4 x_1 + 1000 x_3) = 98000 x_3 \qquad ('')$$

When trying to solve the new model (M') that includes the constraints (##) and (**) instead of the constraints (#) and (*) the solution is reached after 389 iterations. The reformulation done during the preprocessing has hence permitted the model that could not be solved initially, to be solved.

3) Furthermore if the user attempts also to improve the scaling of the model when doing the preprocessing, several variables and constraints are then rescaled. When trying to solve the new model, the solution is obtained after only 46 iterations. Our scaling procedure appears to be very efficient with this example.

This example illustrate that our preprocessing interface can greatly improve the robustness of NLP algorithms. It can also increase the efficiency of the algorithm by reducing the number of necessary iterations. However, one should note that as far as the global CPU time is concerned the time saved when solving a model may not in general be important enough to balance the time required by the preprocessing procedure.

Fleet problem :

This problem has been presented by P.Y.Papalambros [18]. In its initial formulation it can not be solved by GAMS/MINOS and GAMS/SQP. When preprocessed through the interface, the model is found to be geometric and a transformed model using the exponential transformation is obtained that can easily be solved by GAMS/MINOS and GAMS/SQP. The GAMS input files of both the initial and the reformulated models are presented in Appendix C.

MILP problems :

A few small to medium sized mixed-integer linear problems have been tested with our interface. The results obtained with four selected models are summarized in *Table* 2.

In the top section of *Table 2* the size of the models are listed. The next section of the table shows how many bounds (either upper bounds or lower bounds) were specified for the continuous variables initially and after having applied the tightening procedure. The last row of that section shows how many of the binary variables can be fixed from the results obtained when doing the tightening procedure. The two sections at the **bottom** of the table present the computational results obtained using GAMS/ZOOM. It shows first how many nodes have been evaluated by ZOOM during the branch and bounds procedure with the initial model and with the reformulated model. Finally, it compares at last the total number of necessary iterations for solving the initial model and the reformulated model.

It is clear that the preprocessing procedure has produced a large number of bounds and that in several cases it permitted fixing some of the binary variables. The impact of fixing some of the binary variables is that it reduces the number of nodes required by the branch and bounds algorithm used in ZOOM. This number is reduced to zero in three out of four examples making the branch and bound procedure unecessary and drastically reduces the total number of iterations.

	7sp3	GS	maxmi2	pshex
cont. var.	74	9	13	80
bin. var.	14	4	4	21
equations	60	12	19	87
non-lin.	0	0	0	0
bounds before	0	0	0	0
bounds after	142	12	10	118
bin.fixed	8	4	0	13
nodes before	42	16	16	47
nodes after	0	0	16	0
iter. before	284	46	103	555
iter. after	39	7	102	83

on Sun3/60 workstation using GAMS/ZOOM

Table 2: Computational results with selected MILP models

In the maxmi2 example no improvement upon the number of required iterations is achieved although 10 out of 26 possible bounds have been derived. This illustrates the point that unless binary variables are eliminated by this procedure, bound tightening may not help.

MINLP examples :

Two test examples are presented in this section : one for which the bounds tightening procedure has a beneficial impact, and the other for which the procedure does not lead to any improvement. The data and results concerning these two models are summarized in *Table 3.* The structure of *Table 3* is similar to the one of *Table 2* presented earlier.

In the preprocessing of both examples we obtain tighter bounds on the continuous variables but none of the binary variables can be fixed before solving the models. However, the impact on the total number of iterations is quite different..

In the batch problem (a fairly large and nonlinear model for optimizing the number of units, their volumes and their cycle times for a low cost production), the total number of iterations can be reduced to one

	duran	batch
cont. var.	25	24
bin. var.	8	24
equations	32	74
non-lin.	5	22
bounds before	9	18
bounds after	24	27
bin.fixed	0	0
nodes before	36	64
nodes after	37	66
iter. before	801	2051
iter. after	905	694

on Sun3/60 workstation using GAMS/ZOOM/MINOS/DICOPT++

Table 3 : Computational results with selected MINLP models

third. It should also be noted that in the reformulated model many of the coefficients of the mixed-integer constraints have been reduced.

The example duran [4] is presented to stress the fact that, as with the reformulation procedure, tightening the bounds does not always lead to faster convergence. The topology of the feasibility region, the degree of nonlinearity of the model and the starting point are three factors that are difficult to control and whose impact on the efficiency of the solution can camouflage the effect of having tight bounds.

CONCLUSIONS :

In this paper we have presented how some of the principles and techniques that help in formulating well-behaved models have been encoded in an interface that automates the process. This interface can be used by model designers as a preprocessing tool to automatically obtain alternative formulations that would provide him with a better model. Since very few transformations can be guaranteed to improve any given model, there is no standard way of formulating a problem to ensure a more robust formulation, so the model proposed by the interface is not guaranteed to be more efficient for all problems.

Various features has been encoded within the interface that allows one to tacke the problem of model reformulation at various levels. Firstly, when preprocessing a set of constraints, the interface is able to automatically identify troublesome equations and to symbolically manipulate them to produce well-formulated expressions. In some cases global manipulations of the set of constraints are performed in order to convexity the model. Besides, the use of new variables and new equations can lessen its degree of nonlinearity. Moreover it is to be noted that the preprocessing pays special attention to numerical concerns by avoiding potential problems of locally undefined functions and also by applying an empirical scaling procedure.

Secondly, a strategy has been developed and implemented for reducing the domain of search and thereby increasing the robustness of the model and often speeding up the convergence. The algorithm takes advantage of monotonic constraints and through a recursive procedure over the set of these constraints derives tighter bounds for the continuous variables. Test problems have shown that this proposed algorithm can greatly reduce the domain of search. It can also allow for the fixing of the values of some of the binary variables even before starting the solution procedure and thus may decrease the number of node evaluations required during the branch-and-bound search.

The developed interface is a good tool for automatically deriving alternative formulations and well-behaved models. As shown by the examples, the strengthening of the model robustness that can be performed through our interface is worth the extra effort .

REFERENCES :

[1] Benders, J.F., *"Partitioning Procedures for Solving Mixed-variables Programming Problems"*, Numerische Mathematik 4, 238-252.

[2] Biegler, L.T., *"Simultaneous Modular Simulation and Optimization"*, Proceedings 2nd Intl. Conf. FOCAPD, Eds. Westerberg and Chien, CACHE, 369-409 (1984).

[3] Brooke, A., D.Kendrick and A.Meeraus, *"GAMS - A Users Guide"*, Scientific Press (1988).

[4] Duran, M.A. and I.E.Grossmann, *"An Outer-Approximation Algorithm fora Special Class of Mixed Integer Nonlinear Programs"*, Mathematical Programming 36.307 (1986).

[5] Edgar, T.F. and D.M.Himmelblau, *"Optimization of Chemical Processes"*, McGraw Hill (1988).

[6] Geoffrion, A.M., *"Generalized Benders Decomposition"*, Journal of Optimization Theory and Applications, 10(4), 237-260.

[7] Gill, P.E., W. Murray and M. H.Wright, *"Practical Optimization"*, Academic Press, London (1981).

[8] Grossmann, I.E., *"Mixed-Integer Nonlinear Programming Techniques for the Synthesis of Engineering Systems"*, Research in Engineering Design. 1, 205 (1990).

[9] Han, S.P., *"Superlinearly Convergent Variable Metric Algorithms for General Nonlinear Programming Problems"*, Math. Progr.. Vol. 11, 263-282,1976.

[10] IBM General Information Manual, *"An Introduction to Modeling Using Mixed Integer Programming"*, Amsterdam, 1972.

[11] Johnson,. E.L. and U.H. Suhl, *"Experiments in Integer Programming"*, Discrete Applied Mathematics, 2, 39-55 (1980).

[12] Kocis, G.R. and I.E.Grossmann *"Computational Experience with DICOPTsolving MINLP*

Problems in Process Synthesis Engineering", Computers and Chemical Engineering, 13, 307-315 (1989).

[13] Kocis, G.R. and I.E.Grossmann *"A Modeling and Decomposition Strategy for the MINLP Optimization of Process Flowsheets"*, Computers and Chemical Engineering, 13, 7, 797-819 (1989).

[14] Liebman, J., L.Ladson, L.Schrage and A.Warren, *"Modelling and Optimization with GINO"*, Scientific Press (1986).

[15] Murtagh, B.A. and M.A.Saunders, *"A Projected Lagrangian Algorithm and its Implementation for Sparse Nonlinear Constraints"*, Mathematical Programming Study, Vol. 16,84-117(1982).

[16] Nemhauser, G.L. and LA.Wolsey, "Integer and Combinatorial Optimization", John Wiley (1988).

[17] Padberg, M., H.Crowder and E.L.Johnson, *"Solving Large-Scale Zero-One Linear Programming Problems"*, Operations Research. Vol. 31, No. 5, Sept/Oct 1983.

[18] Papalambros, P.Y. and DJ. Wilde, *"Principles of Optimal Design - Modeling and Computation"*, Cambridge University Press (1988).

[19] Plane, D.R. and C.McMillan Jr, *"Discete Optimization"*, Prentice-Hall, Englewood Cliffs, NJ, 1971.

[20] Powell, M.J.D., *"A Fast Algorithm for Nonlinearly Constrained Optimization Calculations"*, In Numerical Analysis, Dundee 1977. G.A. Watson (ed.) Lecture Notes in Mathematics 630, Springer-Verlag, Berlin (1978).

[21] Reklaitis, G.V., A.Ravindran and K.M.Ragsdell, *"Engineering Optimization : methods and applications"*, John Wiley (1983).

[22] Van Roy, T.J. and L.A. Wolsey, *"Solving Mixed Integer Programming Problems Using Automatic Reformulation*, Operations Research. 35, 45-57 (1987).

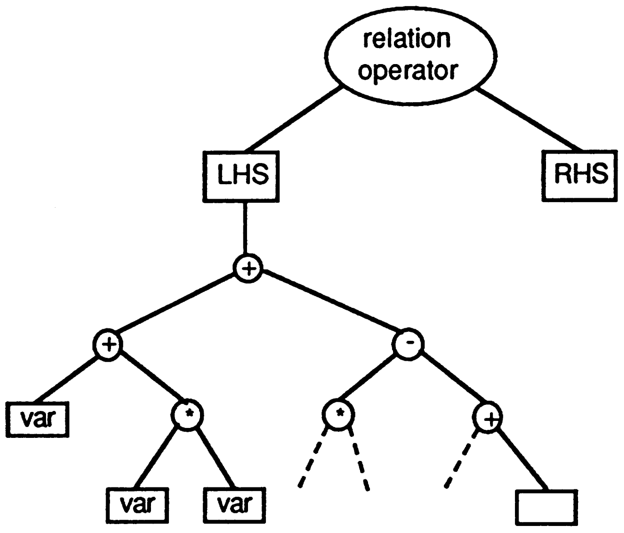
[23] Viswanathan, J. and I.E.Grossmann, *"A Combined Penalty Function and Outer-Approximation Method for MINLP Optimization"*, Computers and Chemical Engineering. 14, 7, 769-782 (1990).

Appendix A :

1) Monotonicity of the equations :

An equation is defined as being monotonic when its left hand side (the right hand side being a constant) is monotonically increasing or decreasing with respect to each variable. In the context of this definition, all the variables are assumed to be non negative. To determine whether an expression is increasing or decreasing with respect to a given variable, one can check the sign of the partial derivative of the expression. An alternative approach is to build the tree representing the expression and then to propagate information concerning the monotonicity of the variables along that tree.

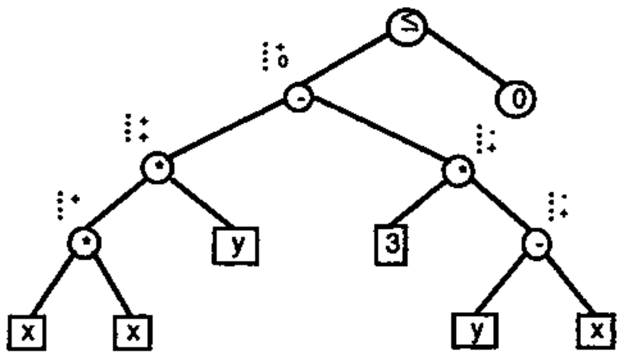
Building the tree representing an equation consistsof drawing the following schematic for the equation



Once the tree has been built, using the rules of precedence and associativity, we can evaluate the monotonicity at each node given the partial expression beneath the node. This is done by considering the arithmetic operator involved at the node as well as the monotonicity of the arguments of the operator. Each variable is considered in turn, starting at the roots of the tree. Note that no restriction is placed on the linearity of the expression. This method is easy to implement once the equations have been written in the reverse polish notation used in our LISP implementation.

The example below is intended to clarify the steps involved in the method.
Suppose we want to analyze the monotonicity of the equation $x^2y - 3 * (y-x) \leq 0$. By directly applying

the method described above we obtain the figure shown below. Next to each node are two signs (+, - or 0); the top one represents the monotonicity of the subtree below the node with respect to the variable x ; the bottom one represents the monotonicity with respect to y. A V means that the expression is increasing with the variable, a"-" means it is decreasing and a "0" means that the expression is not monotonic with respect to the variable.



From the "+" next to the "≤" sign we can conclude that the expression $x^2y - 3 * (y-x) \leq 0$ is monotonic (and increasing) with respect to the varaible x. From the "0" next to "≤" we conclude that $x^2y - 3 * (y-x) \leq 0$ is not monotonic with respect to the variable y.

2) **Obtaining the required critical bounds :**

Since each equality constraints is converted into two inequalities, the model consists only of inequalities. Then for each monotonic inequality eqn_k, we dispose of the following information : a list var_in_eqnj< of the variables that occur in the inequality; and a list monotonicityJn_eqn_k, consisting of V and "-", describing the monotonicity of eqn_k with respect to the corresponding variables of varjn_eqn_k.

If the sign in monotonicity_in_eqn_k corresponding to variable XJ of var_jn_eqn_n is "+" and the relational operator is V, then the lower bound of XJ is called the critical bound; when the relational operator is ">" the critical bound is given by the upper bound on XJ. Similarily, when the sign in monotonicity_in__eqn_k is "-", the critical bound on XJ is its upper bound if the relational operator is V\ or its lower bound if the relational operator is V.

It is thus possible to construct a list of "low" and "up" describing which bound is critical for the each of the variables in var_in_eqn_k.

In the equation $x - y < 2$, for instance, the critical bounds are the lower bound for x and the upper bound for y.

3) Deriving tight bounds :

For each equation eqn_k, we are given first varjn_eqn_k, second the corresponding lists monotonicityjn_eqn_k and criticaljboundsjor_eqn_k, third the lists of the names of the bounds that have been updated in the previous loop (updated_LO and updated_UP) and so far in the current loop (updatedJo and updated_up). We then count the number N of unknown bounds in critical__boundsjor_eqn_k.
Note : If the default values (0 and 50000) are used for the unknown bounds then process directly to the case N=0.

if N>=2 then tightening the bounds is not possible using this equation at the current loop.

if N=1 then
let us call x* the variable for which a new bound is to be derived, then fix all the variables except x* to their critical bound numerical values and solve the equation for x\ If the numerical value of x* obtained thus is in the feasible range (between 0 and 50000), take it as the new value for the critical bound of x*; otherwise discard it.
Remark about the way the equation is solved in our interface : when x* occurs only once in the expression, the equation is solved for x* and the exact solution is obtained ; when x* occurs more than once a bisection method is applied (recalling that the expression is either monotically increasing or decreasing with x*).

if N=0 then
for each variable of varjn_eqn_k in turn, assume that no critical bound is known for that variable and apply the process described for the case where N=1. In case a bound is derived from that processing, compare it to the existing one and record the best one. Proceed then with the next variable.

It is interesting to be noted that one can easily predict the nature of the bound that will be derived at each calculation depending upon the type of inequality constraint and the type of monotonicity for the equation whose bound is seeked for. Any monotonic constraint can be casted in one of the two cases below where the V represents an operation that keeps the monotonicity such as an addition or a log and the "-" represents an operation that reverses the monotonicity such as a substractjon or a division.

Assuming that
- X is a vector of continuous variables x
- Y is a vector of continuous variables y

- X and Y are disjoint
- f is a function monotonically increasing with respect with all the x
- g is a function monotonically increasing with respect with all the y
- Y.lo means that all the variables y of the vector Y have a lower bound y.lo and Y.up, X.lo and X.up are defined in a similar manner,

the ensuing rules stand :

when $f(X) + g(Y) \leq a$

if Y.lo is known some information about the x.up might be derived,

if Y.up is known no information can be derived.

when $f(X) - g(Y) \leq a$

if Y.lo is known no information can be derived,

if Y.up is known some information about the x.up might be derived,

if X.lo is known some information about the y.lo might be derived,

if X.up is known no information can be derived.

Appendix B :

In this appendix we will present an example of GAMS input file preprocessing for a NLP model, the alkylation problem for which results have been presented earlier in this paper. The variations between the initial input file and the reformulated input file are marked in bold style for better identification.

1) Initial GAMS input file :

```
$TITLE : IMPACT OF LINEAR EQUATIONS ON THE ALKYLATION PROBLEM

$OFFUPPER
$OFFSYMXREF
$OFFSYMLIST

VARIABLES
   PROFIT ;

POSITIVE VARIABLES
   X1, X2, X3, X4, X5, X6, X7, X8, X9, X10 ;

EQUATIONS
   EQ1, EQ2, EQ3, EQ4, EQ5, EQ6, EQ7, EQ8 ;

EQ1.. X1*(1.12 + 0.13167*X8 - 0.00667*X8**2) - X4 =E= 0 ;
EQ2.. 86.35 + 1.098*X8 - 0.038*X8**2 + 0.325*(X6 - 89) - X7 =E= 0 ;
EQ3.. 35.82 - 0.222*X10 - X9 =E= 0 ;
EQ4.. X10 - 3*X7 + 133 =E= 0 ;
EQ5.. X8*X1 - X2 - X5 =E= 0 ;
EQ6.. X5 - 1.22*X4 + X1 =E= 0 ;
EQ7.. X6 * (X4*X9 + 1000*X3) - 98000 * X3 =E= 0 ;
EQ8.. PROFIT =E= 0.063*X4*X7 - 5.04*X1 - 0.035*X2 - 10*X3 - 3.36*X5;

 X1.LO=10;  X1.UP=2000;
 X2.LO=0;   X2.UP=16000;
 X3.LO=0;   X3.UP=120;
 X4.LO=0;   X4.UP=5000;
```

```
  X5.LO=0;
  X6.LO=85;   X6.UP=93;
  X7.LO=90;   X7.UP-95;
  X8.LO-3;    X8.UP=12;
  X9.LO=1.2;  X9.UP=4;
  X10.LO-145;  X10.UP=162;


OPTION NLP=MINOS5 ;
OPTION ITERLIM = 1000;

MODEL PB/ALL/;

SOLVE PB USING NLP MAXIMIZING PROFIT;
```

1) Reformulated GAMS Input file :

```
$TITLE : IMPACT OF LINEAR EQUATIONS ON THE ALKYLATION PROBLEM

SOFFUPPER
$OFFSYMXREF
$OFFSYMLIST


 VARIABLES
    SX1, SX2, SX3, SX4, SX5,  X6, X7, X8, X9, SX10, PROFIT  ;

 PARAMETERS
    XI,  X2, X3, X4, X5, X10  ;


EQUATIONS
   EQ1, EQ2, EQ3, EQ4, EQ5, EQ6, EQ7, EQ8 ;
```

```
EQ1..  199.0 * SX1 * (1.12 + 0.13167 * X8 - 0.00667 * X8 " 2) - 500.0 * SX4 =E= 0;

EQ2..  86.35 + 1.098 * X8 - 0.038 * X8 " 2 + 0.325 * (X6 - 89) - X7 =E= 0;

EQ3..  35.82 - 0.222 * 2.0 * SX10 - X9 =E= 0;

EQ4..  2.0 * SX10 - 3 * X7 + 133 =E= 0;

EQ5..  X8 * 199.0 * SX1 /1000 - 1.6 * SX2 - SX5 =E= 0.0;

EQ6..  0.5 * SX5 - 1.22 * 500.0 * SX4 / 2000 + 199.0 • SX1 / 2000 =E= 0.0;

EQ7..  X6 * (500.0 * SX4 * X9 / 2418000.0 + 1000 * 12.0 * SX3 / 2418000.0)
          - 98000 * 12.0 * SX3 / 2418000.0  =E= 0.0;

EQ8..  PROFIT =E= 0.063*X4*X7 - 5.04*X1 - 0.035*X2 - 10*X3 - 3.36*X5 ;


 SX1.LO=0.05   ;
 SX2.LO=0  ;
 SX3.LO=0  ;
 SX4.LO=0  ;
 SX5.LO=0  ;
 X6.LO=85;
 X7.LO=90;
 X8.LO=3;
 X9.LO=1.2;
 SX10.LO=72.5   ;

 SX1.UP=10.06   ;
  SX2.UP=10.0  ;
  SX3.UP=10.0  ;
  SX4.UP=10.0  ;
 X6.UP=93;
 X7.UP=95;
 X8.UP=12;
 X9.UP=4;
```

```
   SX10.UP=81.0   ;

   SX1.L=5.03;
    SX2.L=5.0   ;
     SX3.L=5.0  ;
     SX4.L=5.0  ;
     SX5.L=1.0  ;
    X6.L=89  ;
    X7.L=92.5  ;
    X8.L=7.5  ;
    X9.L=2.6  ;
     SX10.L=76.75  ;


MODEL PB/ALL/;

OPTION  NLP=MINOS5;
OPTION  ITERLIM = 1000 ;

SOLVE PB USING NLP MAXIMIZING PROFIT;


  X1  = SX1 * 199.0 ;
  X2  = SX2 * 1600.0 ;
  X3  = SX3 * 12.0 ;
  X4  = SX4 * 500.0 ;
  X5  = SX5 * 1000.0 ;
  X10  = SX10 * 2.0 ;

DISPLAY
   XI, X2,   X3, X4, X5, X10   ;
```

Appendix C :

In this appendix we will present an example of GAMS input file preprocessing for a NLP model which is identified by the interface as being geometric and which is then reformulated using the exponential transformations. Computational results with this model have been presented earlier in this paper under the fleet model appellation.

1) Initial GAMS input file :

```
VARIABLES
n, l, b, h, t, v, d, di, u, cost ;

EQUATIONS
eq1, eq2, eq3, eq4, eq5, eq6, eq7, eq8, eq9 ;

eq1.. 2E-6*d*u*v/3 + u =e= 1 ;

eq2.. 0.4*((l*b*h)**0.9)*(di**0.52) + d*(di**0.52)
      + 0.02*(v**2.16)*di - di*(di**0.52) =e= 0 ;

eq3.. 0.93*dil*(l**0.5) + 0.93*v*l*b*t - (l**0.5)*l*b*t =e= 0 ;

eq4.. 2.75E6 + 2.1E-3*v*v*(di**(2/3))*n*u*v
      - n*d*u*v =l= 0 ;

eq5.. 1.9E-3*(t**1.43) + t - h =l= 0 ;

eq6.. 2*t - b =l= 0 ;

eq7.. 0.07*l - h =l= 0 ;

eq8.. 3*d - l*b*h =l= 0 ;

eq9.. cost =e= 160*n*((l*b*h)**0.9) + (2+0.8*u)*n*(v**2.16)/(di**0.48) ;

n.lo=1 ;
```

```
l.lo=5;
b.lo=5 ;
h.lo=3 ;
t.lo=2 ;
v.lo=3 ;
d.lo*10000 ;
di.lo=50000 ;
u.lo=0.6 ;

n.up=50 ;
u.up=1 ;

MODEL fleet/ALL/ ;

OPTION LIMCOL=0;
OPTION LIMROW=0 ;
OPTION NLP=MINOS;

SOLVE fleet USING nip MINIMIZING cost;
```

1) Reformulated GAMS Input file :

```
VARIABLES
n, l, b, h, t, v, d, di, u, cost;

PARAMETERS
on, ol, ob, oh.o t, ov, od, odi. ou ;

EQUATIONS
eq1, eq2, eq3, eq4, eq5, eq6, eq7, eq8, eq9 ;

eq1.. exp( d + v -14.220976 + u) + exp( u) =e= 1 ;

eq2.. exp( 0.9*(l+b+h) - 0.916291 - di) + exp( d - di)
      + exp( 2.16*v - 3.912023 - 0.52*di) =e= 1 ;

eq3.. exp( di - 0.07257 -1 - b -1) + exp( v - 0.072571 - 0.5*1) =e= 1 ;
```

```
eq4..  exp( 14.82711 -n-d-u-v)

        + exp( 2*v + 0.666667*di - 6.165818 - d) =l= 1  ;

eq5..  exp( 1.43M - h - 6.265901) + exp( t - h) =l= 1  ;

eq6..  t - b + 0.69315=1=0;

eq7..  I - h - 2.65926 =1= 0 ;

eq8..  d -1 - b - h +1.0986123 =l= 0 ;

eq9..  cost =e= exp( n + 0.9*(l+b+h) + 5.0714)                    ;
                + exp( log( 2+0.8*exp(u)) + n + 2.16*v - 0.48*di)


n.lo=1  ;
l.lo=1 ;
b.lo=1  ;
h.lo=1  ;
t.lo=1 ;
v.lo=1 ;
d.lo=1  ;
di.lo=1 ;

u.up=O  ;

MODEL fleet /ALL/  ;

OPTION LIMCOL=0;
OPTION LIMROW=0;
OPTION SOLPRINT=OFF;

SOLVE fleet USING nip MINIMIZING cost;


on= exp(n.l);
DISPLAY 'n= ', on ;
```

```
ol= exp(l.l);
DISPLAY W . o l ;

ob= exp(b.l);
DISPLAY 'b=\ ob;

oh= exp(h.l);
DISPLAY 'h-1, oh;

ot=exp(t.l);
DISPLAY t=\ot;

ov= exp(v.l);
DISPLAY V=', ov ;

od= exp(d.l);
DISPLAY 'd= *, od ;

odi= exp(di.l);
DISPLAY *di=', odi;

ou= exp(u.l);
DISPLAY 'u=', ou ;
```

Appendix D :

In this appendix we will present an example of GAMS input file preprocessing for a MINLP model, the model called duran earlier in the paper. The variations between the initial input file and the reformulated input file are marked in bold style for better identification.

1) Initial GAMS Input file :

```
$TITLE MARCO DURAN'S EXAMPLE #3 PROBLEM FROM PH.D. THESIS
$OFFSYMXREF
$OFFSYMLIST
* SELECT OPTIMAL PROCESS FROM WITHIN GIVEN SUPERSTRUCTURE.
*
* REFERENCE: MARCO DURAN , PH.D. THESIS , 1984.
*         CARNEGIE-MELLON UNIVERSITY , PITTSBURGH , PA.

* MASS BALANCES ARE REFERENCE ACCORDING TO THEIR POSITION IN
* THE SUPERSTRUCTURE. I.E. MASS BALANCE #1 OCCURS BETWEEN
* UNITS #4 , 6 , AND 7.


 POSITIVE VARIABLES
        x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16,
       x17,x18,x19,x20,x21,x22,x23,x24,x25 ;

 VARIABLES
       profit ;

 BINARY VARIABLES
        y1,y2,y3,y4,y5,y6,y7,y8 ;


 EQUATIONS
       INOUT1    INPUT-OUTPUT RELATIONS FOR PROCESS UNIT 1
       INOUT2    INPUT-OUTPUT RELATIONS FOR PROCESS UNIT 2
       INOUT3    INPUT-OUTPUT RELATIONS FOR PROCESS UNIT 3
 IN0UT4      INPUT-OUTPUT RELATIONS FOR PROCESS UNIT 4
 INOUT5      INPUT-OUTPUT RELATIONS FOR PROCESS UNIT 5
 INOUT6      INPUT-OUTPUT RELATIONS FOR PROCESS UNIT 6
 INOUT7      INPUT-OUTPUT RELATIONS FOR PROCESS UNIT 7
 INOUT8      INPUT-OUTPUT RELATIONS FOR PROCESS UNIT 8
 MASSBAL1    MASS BALANCE #1
 MASSBAL2    MASS BALANCE #2
 MASSBAL3    MASS BALANCE #3
 MASSBAL4    MASS BALANCE #4
 MASSBAL5    MASS BALANCE #5
 MASSBAL6    MASS BALANCE #6
 MASSBAL7    MASS BALANCE #7
 SPECS1      DESIGN SPECSIFICATION 1
 SPECS2      DESIGN SPECSIFICATION 2
 SPECS3      DESIGN SPECSIFICATION 3
 SPECS4      DESIGN SPECSIFICATION 4
 LOGICAL!    CONSTRAINTS WHICH ALLOW FLOW IFF UNIT 1 EXISTS
 LOGICAL2    CONSTRAINTS WHICH ALLOW FLOW IFF UNIT 2 EXISTS
 LOGICAL3    CONSTRAINTS WHICH ALLOW FLOW IFF UNIT 3 EXISTS
 LOGICAL4    CONSTRAINTS WHICH ALLOW FLOW IFF UNIT 4 EXISTS
 LOGJCAL5    CONSTRAINTS WHICH ALLOW FLOW IFF UNIT 5 EXISTS
 LOGICAL6    CONSTRAINTS WHICH ALLOW FLOW IFF UNIT 6 EXISTS
 LOGICAL7    CONSTRAINTS WHICH ALLOW FLOW IFF UNIT 7 EXISTS
 LOGICAL8    CONSTRAINTS WHICH ALLOW FLOW IFF UNIT 8 EXISTS
 PUREINT1    PURELY INTEGER CONSTRAINT #1
 PUREINT2    PURELY INTEGER CONSTRAINT #2
 PUREINT3    PURELY INTEGER CONSTRAINT #3
 PUREINT4    PURELY INTEGER CONSTRAINT #4

 addoonstr

 OBJ       OBJECTIVE FUNCTION DEFINITION;


INOUT1..    EXP(X3)    -1. =E= X2          ;
INOUT2..    EXP(X5/1.2) -1. =E= X4          ;
INOUT3..     1.5 * X9 + X10 =E= X8          ;
INOUT4..     1.25 * (X12+X14) =E= X13       ;
```

```
INOUT5..     X15          =E= 2.*X16        ;
iNOirra.     EXP(X20/1.5)-1. =E= X19             ;
INOUT7..     EXP(X22)  -1. =E= X21            ;
INOUT8..     EXP(X18)  -1. =E= X10 + X17    ;

MASSBAL1..     X13       =E= X19 + X21       ;
MASSBAL2..     X17       =E= X9 + X16 + X25;
MASSBAL3..     X11       =E=X12 + X15        ;
MASSBAL4..     X3 + X5=E== X6 + X11         ;
MASSBAL5..     X6        =E= X7 + X8        ;
MASSBAL6..     X23       =E=X20 + X22        ;
MASSBAL7..     X23       =E= X14 + X24       ;

SPECS1..     X10 =L= 0.8* X17               ;
SPECS2..     X10 =G= 0.4"X17                ;
SPECS3..     X12 =L= 5.0* X14               ;
SPECS4..     X12 =G= 2.0"X14                ;

LOGICAL1..    X2 =L= 50. * Y1               ;
LOGICAL2..    X4 =L= 50. * Y2               ;
LOGICAL3..    X9  =U50.* Y3                 ;
LOGICAL4..    X12 + X14 =L=50.*Y4           ;
LOGICAL5..    X15 =L= 50. *Y5               ;
LOGICAL6..    X19 =L= 50. *Y6               ;
LOGICAL7..    X21 =L= 50. *Y7               ;
LOGICAL8..    X10 + X17 =L= 5O.*Y8          ;

PUREINT1..    Y1+Y2=E= 1.                   ;
PUREINT2..    Y4 + Y5 =L= 1.                ;
PUREINT3..    Y6 + Y7-Y4 =E=0.              ;
PUREINT4..    Y3-Y8 =L= 0.                  ;

addconstr.. :<7 + x18 + x24'=g= 0.1 ;

OBJ.. PROFIT =E= - 5*y1 - 8*y2 - 6*y3 - 10*y4 - 6*y5 - 7*y6 - 4*y7 - 5*y8
         - x2 + 10*x3 - x4 + 15*x5 + 40*x9 - 15*x10 - 15*x14 - 80*x17
         + 65*x18 -25*x19 + 60*x20 -35*x21 + 80*x22 + 35*x25 -122 ;
```

```
MODEL EXAMPLE3  / ALL /;

      OPTIONS LIMCOL = 0                    ;
option optcr=0;

SOLVE EXAMPLE3 USING MIDNLP MAXIMIZING PROFIT  ;
```

1) Reformulated GAMS input file :

```
$TITLE MARCO DURAN'S EXAMPLE #3 PROBLEM FROM PH.D. THESIS
$OFFSYMXREF
$OFFSYMLIST

BINARY VARIABLES
  y1, y2, y3, y4, y5, y6, y7, y8 ;

POSITIVE VARIABLES
  x2, x3, x4, x5, x6, x7, x8, x9, x10, x11, x12, x13, x14, x15, x16, x17,
   x18, x19, x20, x21, x22, x23, x24, x25 ;

VARIABLES
  profit;

EQUATIONS
   inouti, inout2, inout3, inout4, inout5, inout6, inout7, inout8, massbali,
   massbal2, massbal3, massbaW, massbal5, massbal6, massbal7, specsi,
   specs2, specs3, specs4, logicaU, logical2, logical3, logicaW, logical5,
   logical6, logical7, logical8, pureinti, pureint2, pureint3, purein(4,
   addoonstr, obj;


inouti..  exp (x3) -1 - x2 =e= 0 ;
inout2..  exp (x5 /1.2) -1 - x4 =e= 0 ;
inout3..  1.5 * x9 + x10 - x8 =e= 0 ;
inout4..  1.25*(x12 + x14)-x13=e=0;
```

```
inout5..  x15 - 2 * x16 =e= 0 ;
inout6..  exp (x20 / 1.5) - 1 - x19 =e= 0 ;
inout7..  exp (x22) - 1 - x21 =e= 0 ;
inout8..  exp (x18) - 1 - x10 - x17 =e= 0 ;
massbal1..  x13 - x19 - x21 =e= 0 ;
massbal2..  x17 - x9 - x16 - x25 =e= 0 ;
massbal3..  x11 - x12 - x15 =e= 0 ;
massbal4..  x3 + x5 - x6 - x11 =e= 0 ;
massbal5..  x6 - x7 - x8 =e= 0 ;
massbal6..  x23 - x20 - x22 =e= 0 ;
massbal7..  x23 - x14 - x24 =e= 0 ;
specs1..  x10 - 0.8 * x17 =l= 0 ;
specs2..  x10 - 0.4 * x17 =g= 0 ;
specs3..  x12 - 5.0 * x14 =l= 0 ;
specs4..  x12 - 2.0 * x14 =g= 0 ;
logical1..  x2 - 50 * y1 =l= 0 ;
logical2..  x4 - 50 * y2 =l= 0 ;
logical3..  x9 - 5.7668 * y3 =l= 0 ;
logical4..  x12 + x14 - 12.9753 * y4 =l= 0 ;
logical5..  x15 - 8.6502 * y5 =l= 0 ;
logical6..  x19 - 16.2192 * y6 =l= 0 ;
logical7..  x21 - 16.2192 * y7 =l= 0 ;
logical8..  x10 + x17 - 30.2757 * y8 =l= 0 ;
pureint1..  y1 + y2 =e= 1 ;
pureint2..  y4 + y5 =l= 1 ;
pureint3..  y6 + y7 - y4 =e= 0 ;
pureint4..  y3 - y8 =l= 0 ;
addconstr..  x7 + x18 + x24 =g= 0.1 ;

obj.. profit =e= 0 - 5 * y1 - 8 * y2 - 6 * y3 - 10 * y4 - 6 * y5 - 7 * y6
      - 4 * y7 - 5 * y8 - x2 + 10 * x3 - x4 + 15 * x5 + 40 * x9 - 15 * x10 - 15
      * x14 - 80 * x17 + 65 * x18 - 25 * x19 + 60 * x20 - 35 * x21 + 80 * x22 +
      35 * x25 - 122 ;


   x2.up=50.0  ;
   x3.up=3.9319  ;
   x4.up=50.0  ;
```

```
X5.up=4.7182 ;
x6.up=8.6502 ;
x7.up=8.6502 ;
x8.up=8.6502 ;
x9.up=5.7668 ;
    x10.up=8.6502 ;
Xi1.up=8.6502 ;
x12.up=8.6502 ;
x13.up=16.2192 ;
x14.up=4.3251 ;
x15.up=8.6502 ;
x16.up=4.3251 ;
x17.up=21.6255 ;
x18.up=3.4429 ;
x19.ups16.2192 ;
x20.up=4.2691 ;
x21.up=16.2192 ;
X22.up=2.8461 ;
x23.up=7.1152 ;
X24.up=7.1152 ;
X25.up=21.6255 ;


x2.l=25.0;
x3.l=1.96595;
x4.l=25.0;
x5.l=2.3591 ;
x6.l=4.3251 i
x7.l=4.3251 ;
x8.l=4.3251 ;
x9.l=2.8834 ;
x10.1=4.3251 ;
x11.1=4.3251 ;
x12.1=4.3251 ;
x13.l=8.1096 ;
x 14.1=2.16255 ;
x15.l=4.3251 ;
X16.1=2.16255 ;
```

```
  x17.l=10.81275;
 x18.l=1.72145;
 x19.1=8.1096;
 x20.l=2.13455;
 x21.1=8.1096;
 x22.l=1.42305;
        x23.l=3.5576  ;
 x24.l=3.5576;
 x25.l=10.81275  ;

MODEL EXAMPLE3 / ALL / ;

OPTION  LIMCOL = 0 ;
OPTION  OPTCR=0;

SOLVE EXAMPLE3 USING MIDNLP MAXIMIZING PROFIT;
```