Solving the Conditional IF Problem with a MILP Model: from One Interval to *N* Non Overlapping Intervals

Jose Barahona da Fonseca

Department of Electrical Engineering and Computer Science
Faculty of Sciences and Technology
New University of Lisbon
Monte de Caparica, 2829-516 Caparica
Portugal
Email: jbfo@fct.unl.pt

***Abstract*—Although it is very easy to solve the conditional IF problem with a procedural programming language, it is very difficult to solve it with mathematical programming, especially if the argument of the IF it is an AND or an OR logical condition, based on the variables of the Mixed Integer Liner Program (MILP) model. In this work we propose three very elegant and simple MILP models to solve this problem, and we show with three examples that they are very efficient in terms of runtime and memory usage. Next we generalize the first MILP model to *N* non overlapping intervals. Finally we present the outputs of the solution of these MILP models using the Cplex solver and the GAMS modeling software.**

Keywords–Mixed Integer Linear Programming; Linearization of a Nonlinear Problem; Solving any Nonlinear Problem with a MILP Model; Conditional IF Problem; Solution of the Conditonal IF Problem with a MILP Model.

I. Introduction

In procedural programming languages it is very simple to solve the condiotional IF problem. Nevertheless, in mathematical programming, this problem turns into a very nonlinear problem. In this work we apply some linearization techniques [1] to solve the conditional IF problem with a linear MILP model. To our knowledge there are no previous proposals published in the literature to solve the conditional IF problem with mathematical programming. This paper is organized as follows. In section II we make a small introduction to the basic concepts of mathematical programming and MILP model. In section III we describe the first MILP model that solves the conditional IF problem with one interval with an AND argument. In section IV we describe the second MILP model that solves the conditional IF problem with two intervals with an OR argument. In section V we describe the third MILP model that solves the conditional IF problem with *N* non overlapping intervals and in section VI we present the conclusions and possible vectors of evolution of this work.

II. What is a Mathematical Program? What is a MILP Model?

A mathematical program is a set of constraints, each one with a set of variables and an objective variable, defined by an equality constraint, that will be maximized or minimized by an optimization algorithm, the solver. The constraints can be inequalities (greater or equal or less or equal) or equalities.

A Mixed Integer Linear Program (MILP) model is a mathematical program with integer and binary variables and where all constraints are linear. In this work we will use the Cplex solver [2] to obtain the solution of our three MILP models. To implement the MILP model we use a modeling software, the GAMS software [3], which allows to translate each constraint into a line of code, and call the Cplex algorithm with another line of code. This way we used a very complex optimization algorithm to solve our MILP models without knowing its details and parameters.

III. Design and Implementation of a MILP Model to Solve the Conditonal IF Problem with One Interval with an AND Argument

The conditional IF problem with one interval with an AND argument is formalized in Algorithm 1. Next we will explain how we solved this very nonlinear problem with a linear MILP model.

Algorithm 1 Conditonal IF Problem with One Interval with an AND Argument

```
for $i = 1\ to\ Nx$ do
    $x(i) \leftarrow x\_par(i)$
end for
for $i = 1\ to\ Nx$ do
    if $x(i) \geq x\_min\ AND\ x(i) \leq x\_max$ then
        $TA(i) \leftarrow value0$
    else
        $TA(i) \leftarrow value1$
    end if
end for
```

We begin by initializing the array variable $x(i)$ with the array parameter $x\_par(i)$ through the set of Nx equalities (1).

$$\forall i,\ x(i) = x\_par(i) \tag{1}$$

In the set of equalities (1), the array $x\_par(i)$ is a predifined parameter and $x(i)$ an array variable. We convert the parameter array in a variable array to show that the MILP model works well over an array variable. In the next two MILP models we also make this conversion, so we will not make reference to it anymore. The set of equalities (1) is implemented in GAMS syntax by the following line of code:

calc_x(*i*).. *x(i)*=**e=***x_par(i)***;**.

Since the model is linear, we cannnot make the logical AND of $x(i) \geq x\_min$ and $x(i) \leq x\_max$. So we divide the two inequalities in two separate constraints and introduce an auxiliary indexed binary variable, $relax(i)$, that will have the value 1 when the AND condition is false. To prevent the trivial solution of all 1s in $relax(i)$, later on we will minimize it. This way, we formulate the solution of the IF problem with one interval as an optimization problem. This works because we can consider two constraints of a mathematical program as the logical AND of them. Then we multiply $relax(i)$ by a parameter, max_min, and add this product to $x(i)$. This paramenter must ensure that, for all $x(i)$, when $relax(i)$ is 1, then (2) is always true.

$$\forall i,\ x(i) + max\_min\ relax(i) \geq x\_min \tag{2}$$

The set of inequalities (2) is implemented in GAMS syntax by the following line of code:

calc_relax(*i*).. *x*(*i*) + *max_min* * *relax*(*i*) **=g=** *x_min***;**.

The other set of Nx inequalities is expressed by (3).

$$\forall i,\ x(i) \leq x\_max + max\_min\ relax(i) \tag{3}$$

The set of inequalities (3) is implemented in GAMS syntax by the following line of code:

calc_relax2(*i*).. *x(i)* **=l=** *x_max* + *max_min* * *relax(i)***;**.

Now that we have *calculated* $relax(i)$, it is straighforward to choose between the two values, the *in* value, $value0$, associated to a 0 in $relax(i)$, and the *out* value, $value1$, associated to a 1 in $relax(i)$. This is expressed by the set of Nx equations (4).

$$\forall i,\ TA(i) = (1 - relax(i))\ value0\ +\ relax(i)\ value1 \tag{4}$$

Note that $1 - relax(i)$ is the negation of $relax(i)$, and when $relax(i)$ is 0, this means that $x(i)$ is between x_min and x_max. In other words, we can say that (4) is *equivalent* to Algorithm 2.

Algorithm 2 Algorithm *Implemented* by (4)

```
for $i = 1\ to\ Nx$ do
    if $relax(i) = 0$ then
        $TA(i) \leftarrow value0$
    else
        $TA(i) \leftarrow value1$
    end if
end for
```

The set of Nx equalities (4) is implemented in GAMS syntax by the following line of code:

calc_TA(*i*).. *TA(i)* **=e=***(1-relax(i))*value0* + *relax(i)* * *value1***;**.

To minimize $relax(i)$ and prevent the trivial solution of all 1s in this binary variable, we sum all its elements and attribute this sum to the objective variable, obj, through (5), and then minimize obj.

$$obj = \sum_{i=1}^{Nx} relax(i) \tag{5}$$

Equation (5) is implemented in GAMS syntax by:

calc_obj.. *obj*=**e=sum***(i, relax(i))***;**.

The minimization of obj is done in GAMS syntax by the following two lines of code:

Model *CondIF* / **all** /**;**
Solve *CondIF* **minimizing** *obj* **using MIP;**.

In Appendix A, we present the complete GAMS code that implements this first MILP model with $x\_min = 5$, $x\_max = 10$, $max\_min = 100$, $value0 = 50$ and $value1 = -5$, and in Appendix B we present the output of a run of this model using the Cplex solver, with a set of $x\_par(i)$ with $Nx = 50$ values.

IV. Design and Implementation of a MILP Model to Solve the Conditonal IF Problem with Two Intervals with an OR Argument

The conditional IF problem with one interval with an OR argument is formalized in Algorithm 3. Next we will explain how we solved this problem with a MILP model.

Algorithm 3 Conditonal IF Problem with Two Intervals with an OR Argument

```
for $i = 1\ to\ Nx$ do
    $x(i) \leftarrow x\_par(i)$
end for
for $i = 1\ to\ Nx$ do
    if $x(i) \leq x\_min$ then
        $TA(i) \leftarrow value0$
    else
        if $x(i) \geq x\_max$ then
            $TA(i) \leftarrow value1$
        else
            $TA(i) \leftarrow value2$
        end if
    end if
end for
```

Since now we have two disjoint inequalities, we must have two auxiliary indexed binary variables, $relax(i)$ and $relax2(i)$, that must be simultaneously minimized, to prevent the trivial solution of all 1s in these two indexed binary variables.

The first set of inequalities is *implemented* by (6) which is similar to (3).

$$\forall i,\ x(i)\ \leq\ x\_min\ +\ max\_min\ relax(i) \quad (6)$$

The set of inequalities (6) is implemented in GAMS syntax by the following line of code:

calc_relax(*i***)..** *x(i)* **=l=** *x_min + max_min * relax(i)***;**.

The second set of inequalities is *implemented* by (7), where $relax(i)$ is substitued by $relax2(i)$.

$$\forall i,\ x(i)\ +\ max\_min\ relax2(i)\ \geq\ x\_max \quad (7)$$

The set of inequalities (7) is implemented in GAMS syntax by the following line of code:

calc_relax2(*i***)..** *x(i) + max_min * relax2(i)* **=g=** *x_max* **;**.

Now, we need another indexed binary variable, $else\_if(i)$, that will implement the logical AND of $relax(i)$ and $relax2(i)$, i.e., $else\_if(i) = 1$ when $x(i)$ does not belong to both intervals. The logical AND of these two indexed binary variables is *implemented* by (8). To prevent the trivial solution of all 0s in $else\_if(i)$, we will maximize it. This way, $else\_if(i)$ is 1 only when $relax(i)$ and $relax2(i)$ are 1.

$$\forall i,\ 2\ else\_if(i)\ \leq\ relax(i)\ +\ relax2(i) \quad (8)$$

The set of inequalities (8) is implemented in GAMS syntax by the following line of code:

calc_else_if(*i***)..** 2**else_if(i)***=l=***relax(i)+relax2(i)***;**.

Since the two disjoint inequalities of the OR expression are true when $relax(i) = 0$ or $relax2(i) = 0$, we must negate them and then multiply them by the respective associated values. By the contrary, when $x(i)$ is outside of the two intervals, then $else\_if(i) = 1$, so we just multiply it by $value2$. The attribution of values to $TA(i)$ is *implemented* by (9).

$$\begin{aligned}\forall i,\ TA(i) = (1 - relax(i))\ value1 \\ +\ (1 - relax2(i))\ value0 \\ +\ else\_if(i)\ value2\end{aligned} \quad (9)$$

The set of equations (9) is implemented in GAMS syntax by the following line of code:

calc_TA(*i***)..** *TA(i)***=e=***(1-relax(i))*value1 + (1-relax2(i)) * value0 + else_if(i) * value2***;**.

To simultaneously minimize $relax(i)$ and $relax2(i)$ and maximize $else\_if(i)$, we must add $relax(i)$ and $relax2(i)$ and subtract $else\_if(i)$ and then minimize the objective variable, obj. This is *implemented* by (10). Since $else\_if(i)$ appears with a minus signal, the minimization is *converted* in maximization.

$$obj = \sum_{i=1}^{Nx} relax(i) + relax2(i) - else\_if(i) \quad (10)$$

Equation (10) is implemented in GAMS syntax by the following line of code:

calc_obj.. *obj***=e=sum(***i, relax(i)***)+sum(***i, relax2(i)***)-sum(***i, else_if(i)***);**.

Finally in GAMS syntax we minimize obj and tell to the system to use the Cplex solver:

Model *CondIF* **/all/;**
Solve *CondIF* **minimizing** *obj* **using MIP;**.

In Appendix C we show the complete GAMS code that implements this second MILP model and in Appendix D we show an output of a run of this model using using the Cplex solver, with $Nx = 50$ elements, $x\_min = 3$, $x\_max = 5$, $value0 = 50$, $value1 = -5$, $value2 = -20$ and $max\_min = 100$.

V. Design and Implementation of a MILP Model to Solve the Conditonal IF Problem with *N* Non Overlapping Intervals

The solution of the conditional IF problem with *N* non overlapping intervals is formalized by Algorithm 4. Next we will describe the MILP model that solves this problem.

Algorithm 4 Conditonal IF Problem with *N* Non Overlapping Intervals

```
for $i = 1\ to\ Nx$ do
  $x(i) \leftarrow x\_par(i)$
end for
for $i = 1\ to\ Nx$ do
  $flag \leftarrow 0$
  for $k = 1\ to\ N$ do
    if $x(i) \geq x\_min(k)\ AND\ x(i) \leq x\_max(k)$ then
      $flag \leftarrow 1$
      $TA(i) \leftarrow value(k)$
    end if
  end for
  if $flag = 0$ then
    $TA(i) \leftarrow value0$
  end if
end for
```

Now we have *N* intervals, so we will have *N* values of *x_min(j)* and *N* values of *x_max(j)* and the indexed binary auxiliary relaxation variable will have two indexes, $relax(i, j)$. When $relax(i, j) = 1$ this means that $x(i)$ does not belong to the interval $[x\_min(j)\ x\_max(j)]$ and so (11) and (12) are relaxed. When $relax(i, j) = 0$ this means that $x(i)$ belongs to the interval $[x\_min(j)\ x\_max(j)]$ and so (11) and (12) are not relaxed. This works well because max_min is big enough and $relax(i, j)$ will be minimized to prevent the trivial solution of all 1s in this binary variable.

$$\forall i,j,\ x(i)\ +\ max\_min\ relax(i,j)\ \geq\ x\_min(j) \quad (11)$$

The set of inequalities (11) is implemented by the following line of GAMS code:

calc_relax(*i,j***)..** *x(i) + max_min * relax(i,j)* **=g=** *x_min(j)***;**.

Similarly, we must impose that when $x(i) \leq x\_max(j)$, then $relax(i,j) = 0$. This is *implemented* by (12).

$$\forall i,\ x(i)\ \leq\ x\_max(j)\ +\ max\_min\ relax(i,j) \quad (12)$$

The set of inequalities (12) is implemented by the following line of GAMS code:

calc_relax2(*i,j***)..** *x(i)* **=l=** *x_max(j) + max_min * relax(i,j)***;**.

To identify the elements of $x(i)$ that do not belong to any of the *N* intervals we create an auxiliary indexed variable, *else_if(i)*, that is 1 when all $relax(i,j) = 1$, j=1..*N*. This is *implemented* by (13) and by the maximization of $else\_if(i)$, to prevent the trivial solution of all 0s in this binary variable.

$$\forall i,\ N\ else\_if(i)\ \leq\ \sum_{j=1}^{N} relax(i,j) \quad (13)$$

We can say that (13) implements the logical AND of all $relax(i,j)$ for a given i and varying j between 1 and N. This is expressed by (14).

$$\forall i,\ else\_if(i) = \prod_{j=1}^{N} relax(i,j) \quad (14)$$

In the set of equations (14) the multiplication means the logical AND. The set of inequalities (13) is implemented by the following line of GAMS code:

calc_else_if(*i***)..** *N * else_if(i)* **=l= sum***(j, relax(i,j))***;**.

Since when $x(i)$ belongs to interval j, $relax(i,j) = 0$ and all the remaining values of this binary variable, for this value of i, are 1, the $value(j)$ must be attributed to $TA(i)$, and when $else\_if(i) = 1$ all the $relax(i,j) = 1$ and the value $value\_out$ must be attributed to $TA(i)$. This is *implemented* by (15).

$$\forall i,\ TA(i) = else\_if(i)\ value\_out\ + \sum_{j=1}^{N}(1 - relax(i,j))\ value(j) \quad (15)$$

The set of equalities (15) is implemented by the following line of GAMS code:

calc_TA(*i***)..** *TA(i)***=e= sum(***j, (1-relax(i,j))*value(j))* + *else_if(i) * value_out***;**.

Finally we must solve this MILP model with the Cplex solver and simultaneously minimize $relax(i,j)$ and maximize $else\_if(i)$. This is *implemented* by (16), (17) and (18).

$$obj1 = \sum_{i,j} relax(i,j) \quad (16)$$

Equation (16) is implemented by the following line of GAMS code:

calc_obj1.. *obj1***=e=sum(**(*i,j*), *relax(i,j)***);**.

$$obj2 = \sum_{i} else\_if(i) \quad (17)$$

Equation (17) is implemented by the following line of GAMS code:

calc_obj2.. *obj2***=e=sum(***i, else_if(i)***);**.

$$obj\ =\ obj1\ -\ obj2 \quad (18)$$

Equation (18) is implemented by the following line of GAMS code:

calc_obj.. *obj***=e=***obj1 - obj2* **;**.

Finally we obtain the solution of our MILP model with the two following lines of GAMS code:

Model *CondIF* **/all/;**
Solve *CondIF* **minimizing** *obj* **using MIP;**

In Appendix E we present the complete GAMS code that implements this MILP model and in Appendix F we present the output of a run of this MILP model with $Nx = 50$ elements of $x(i)$, $N = 7$ intervals defined by $[x\_min(1) = 5\ x\_max(1) = 10]$, $value(1) = 50$, $[x\_min(2) = 15\ x\_max(2) = 20]$, $value(2) = -50$, $[x\_min(3) = 30\ x\_max(2) = 40]$, $value(3) = 20$, $[x\_min(4) = 50\ x\_max(4) = 60]$, $value(4) = 40$, $[x\_min(5) = 70\ x\_max(5) = 80]$, $value(5) = 80$, $[x\_min(6) = 90\ x\_max(6) = 100]$, $value(6) = 90$, $[x\_min(7) = 105\ x\_max(7) = 115]$, $value(7) = 99$, $value\_out = -5$ and $max\_min = 200$.

VI. Conclusions and Future Work

We designed and implemented the solution of the conditional IF problem with one argument interval with a linear MILP model and then we generalize it for N non overlapping intervals. Although we ran the third MILP model with a small set of *x(i)*, $Nx = 50$ elements, and with $N = 7$ intervals, in the near future we will run this MILP model with greater Nx and N to test the scalability of our MILP model.

References

[1] J. Barahona da Fonseca, ”Solving Nonlinear Problems with MILP Models,” in Proceedings of Escape-19 Conference, June 14-17, 2009, Cracow, Poland, 2009, pp. 647-652

[2] ILOG, Cplex Users Manual. ILOG, 2003.

[3] A. Meeraus, ”Toward a General Algebraic Modelling System,” in Proceedings of IX. International Symposium on Mathematical Programming. Budapest, Hungary. Mathematical Programming Society, 1976, pp. 185-186.

Appendix A
Implementation of the First MILP Model with GAMS Syntax

```
set i /1*50/;

parameter x_par(i);

x_par(i)=3;



x_par(’1’)=5;
x_par(’3’)=6;
x_par(’4’)=7;
x_par(’6’)=6;
x_par(’8’)=10;
x_par(’9’)=4;

scalar x_min /5/
x_max /10/
value0 /50/
value1 /-5/
max_min /100/;

variable obj, TA(i), x(i);

binary variable relax(i);

equations

calc_x(i)
calc_relax(i)
calc_relax2(i)

calc_obj

calc_TA(i)
;

calc_x(i).. x(i)=e=x_par(i);

calc_relax(i).. x(i) + max_min * relax(i) =g= x_min;

calc_relax2(i).. x(i) =l= x_max + max_min * relax(i);

calc_TA(i).. TA(i)=e=(1-relax(i))*value0 + relax(i) * value1;

calc_obj.. obj=e=sum(i, relax(i));

Model CondIF /all/;

Solve CondIF minimizing obj using MIP;

display  TA.l, obj.l, x.l, x_min, x_max, max_min, relax.l;
```

Appendix B
Output of a Run of the First MILP Model

```
GAMS Rev 229  WIN-VIS 22.9.2 x86/MS Windows           03/24/18 13:23:34 Page 6
G e n e r a l   A l g e b r a i c   M o d e l i n g   S y s t e m
E x e c u t i o n


----     51 VARIABLE TA.L

1  50.000,    2  -5.000,    3  50.000,    4  50.000,    5  -5.000,    6  50.000
7  -5.000,    8  50.000,    9  -5.000,    10 -5.000,    11 -5.000,    12 -5.000
13 -5.000,    14 -5.000,    15 -5.000,    16 -5.000,    17 -5.000,    18 -5.000
19 -5.000,    20 -5.000,    21 -5.000,    22 -5.000,    23 -5.000,    24 -5.000
25 -5.000,    26 -5.000,    27 -5.000,    28 -5.000,    29 -5.000,    30 -5.000
31 -5.000,    32 -5.000,    33 -5.000,    34 -5.000,    35 -5.000,    36 -5.000
37 -5.000,    38 -5.000,    39 -5.000,    40 -5.000,    41 -5.000,    42 -5.000
43 -5.000,    44 -5.000,    45 -5.000,    46 -5.000,    47 -5.000,    48 -5.000
49 -5.000,    50 -5.000


----     51 VARIABLE obj.L                 =       45.000

----     51 VARIABLE x.L

1   5.000,    2   3.000,    3   6.000,    4   7.000,    5   3.000,    6   6.000
7   3.000,    8  10.000,    9   4.000,    10  3.000,    11  3.000,    12  3.000
13  3.000,    14  3.000,    15  3.000,    16  3.000,    17  3.000,    18  3.000
19  3.000,    20  3.000,    21  3.000,    22  3.000,    23  3.000,    24  3.000
25  3.000,    26  3.000,    27  3.000,    28  3.000,    29  3.000,    30  3.000
31  3.000,    32  3.000,    33  3.000,    34  3.000,    35  3.000,    36  3.000
37  3.000,    38  3.000,    39  3.000,    40  3.000,    41  3.000,    42  3.000
43  3.000,    44  3.000,    45  3.000,    46  3.000,    47  3.000,    48  3.000
49  3.000,    50  3.000

----     51 PARAMETER x_min                =        5.000
PARAMETER x_max                =       10.000
PARAMETER max_min              =      100.000

----     51 VARIABLE relax.L

2  1.000,    5  1.000,    7  1.000,    9  1.000,    10 1.000,    11 1.000
12 1.000,    13 1.000,    14 1.000,    15 1.000,    16 1.000,    17 1.000
18 1.000,    19 1.000,    20 1.000,    21 1.000,    22 1.000,    23 1.000
24 1.000,    25 1.000,    26 1.000,    27 1.000,    28 1.000,    29 1.000
30 1.000,    31 1.000,    32 1.000,    33 1.000,    34 1.000,    35 1.000
36 1.000,    37 1.000,    38 1.000,    39 1.000,    40 1.000,    41 1.000
42 1.000,    43 1.000,    44 1.000,    45 1.000,    46 1.000,    47 1.000
48 1.000,    49 1.000,    50 1.000
```

Appendix C
Implementation of the Second MILP Model with GAMS Syntax

```
set i /1*50/;
parameter x_par(i);
x_par(i)=2; x_par(’1’)=5; x_par(’2’)=4; x_par(’3’)=6; x_par(’4’)=7; x_par(’5’)=4;
x_par(’6’)=6; x_par(’7’)=4; x_par(’8’)=10; x_par(’9’)=4; x_par(’10’)=4; x_par(’11’)=4;
x_par(’12’)=4;
scalar x_min /3/
x_max /5/
value0 /50/
value1 /-5/
value2 /-20/
max_min /100/;
variable obj, TA(i), x(i);
binary variable relax(i), relax2(i), else_if(i);
calc_x(i).. x(i)=e=x_par(i);
calc_relax(i).. x(i) =l= x_min + max_min * relax(i);
calc_relax2(i).. x(i) + max_min * relax2(i) =g= x_max ;
calc_else_if(i).. 2*else_if(i)=l=relax(i)+relax2(i);
calc_TA(i).. TA(i)=e=(1-relax(i))*value1 + (1-relax2(i)) * value0 + else_if(i) * value2;
* if value0=value1 this will be equivalent to
* IF x(i) ge x_max OR x(i) le x_min THEN TA(i)=value1
calc_obj.. obj=e=sum(i, relax(i))+sum(i, relax2(i))-sum(i, else_if(i));
Model CondIF /all/;
Solve CondIF minimizing obj using MIP;
display  TA.l, obj.l, x.l, x_min, x_max, max_min, relax.l, relax2.l;
```

Appendix D
Output of a Run of the Second MILP Model

```
GAMS Rev 229  WIN-VIS 22.9.2 x86/MS Windows           02/20/18 20:00:34 Page 6
G e n e r a l   A l g e b r a i c   M o d e l i n g   S y s t e m
E x e c u t i o n


----     71 VARIABLE TA.L

1   -5.000,    2  -20.000,    3   -5.000,    4   -5.000,    5  -20.000
6   -5.000,    7  -20.000,    8   -5.000,    9  -20.000,    10 -20.000
11 -20.000,    12 -20.000,    13  50.000,    14  50.000,    15  50.000
16  50.000


----     71 VARIABLE obj.L                 =       16.000

----     71 VARIABLE x.L

1   5.000,    2   4.000,    3   6.000,    4   7.000,    5   4.000,    6   6.000
7   4.000,    8  10.000,    9   4.000,    10  4.000,    11  4.000,    12  4.000
13  2.000,    14  2.000,    15  2.000,    16  2.000


----     71 PARAMETER x_min                =        3.000
PARAMETER x_max                =        5.000
PARAMETER max_min              =      100.000

----     71 VARIABLE relax.L

1  1.000,    2  1.000,    3  1.000,    4  1.000,    5  1.000,    6  1.000
7  1.000,    8  1.000,    9  1.000,    10 1.000,    11 1.000,    12 1.000


----     71 VARIABLE relax2.L

2  1.000,    5  1.000,    7  1.000,    9  1.000,    10 1.000,    11 1.000
12 1.000,    13 1.000,    14 1.000,    15 1.000,    16 1.000


----     71 VARIABLE else_if.L

2  1.000,    5  1.000,    7  1.000,    9  1.000,    10 1.000,    11 1.000
12 1.000


----     71 VARIABLE TA.L

1   -5.000,    2  -20.000,    3   -5.000,    4   -5.000,    5  -20.000
6   -5.000,    7  -20.000,    8   -5.000,    9  -20.000,    10 -20.000
11 -20.000,    12 -20.000,    13  50.000,    14  50.000,    15  50.000
16  50.000
```

Appendix E
MILP Model for the Solution of the Conditional IF Problem with *N* Intervals

```
scalar N /7/;

set i /1*50/
```

```
j /1*7/;
* We must have N=card(j)

parameter x_par(i), x_min(j), x_max(j), value(j);

x_min(j)=0;
x_max(j)=0;
value(j)=0;

x_min('1')=5;
x_max('1')=10;

x_min('2')=15;
x_max('2')=20;

x_min('3')=30;
x_max('3')=40;

x_min('4')=50;
x_max('4')=60;

x_min('5')=70;
x_max('5')=80;

x_min('6')=90;
x_max('6')=100;

x_min('7')=105;
x_max('7')=115;


value('1')=50;
value('2')=-50;
value('3')=20;
value('4')=40;
value('5')=80;
value('6')=90;
value('7')=99;


x_par(i)=3;



x_par('1')=2;
x_par('3')=6;
x_par('4')=7;
x_par('5')=8;
x_par('6')=6;
x_par('8')=2;
x_par('9')=4;
x_par('10')=16;
x_par('11')=17;
x_par('12')=18;
x_par('13')=19;
x_par('14')=20;

x_par('15')=3;
x_par('16')=31;
x_par('17')=32;
x_par('18')=33;
x_par('19')=34;
x_par('20')=35;

x_par('21')=3;
x_par('22')=55;
x_par('23')=57;
x_par('24')=58;
x_par('25')=60;

x_par('26')=3;
x_par('27')=71;
x_par('28')=72;
x_par('29')=73;
x_par('30')=74;
x_par('31')=75;
x_par('32')=76;
x_par('33')=77;

x_par('34')=3;
x_par('35')=91;
x_par('36')=92;
x_par('37')=93;
x_par('38')=94;
x_par('39')=95;
x_par('40')=96;
x_par('41')=97;
x_par('42')=100;

x_par('43')=3;
x_par('44')=107;
x_par('45')=109;
x_par('46')=111;
x_par('47')=113;
x_par('48')=115;


*x_par('49')=18;

scalar value_out /-5/
max_min /200/;

variable obj, obj1, obj2, TA(i), x(i);

binary variable relax(i,j), else_if(i);

equations

calc_x(i)
calc_relax(i,j)
calc_relax2(i,j)
calc_obj1
calc_obj2
calc_obj

calc_TA(i)

calc_else_if(i)

;

calc_x(i).. x(i)=e=x_par(i);

calc_relax(i,j).. (x(i) + max_min * relax(i,j)) =g= x_min(j);

calc_relax2(i,j).. x(i) =l= (x_max(j) + max_min * relax(i,j));

calc_else_if(i).. N * else_if(i) =l= sum(j, relax(i,j));

calc_TA(i).. TA(i)=e= sum(j, (1-relax(i,j))*value(j)) + else_if(i) * value_out;

calc_obj1.. obj1=e=sum((i,j), relax(i,j));

calc_obj2.. obj2=e=sum(i, else_if(i));

calc_obj.. obj=e=obj1 - obj2 ;

Model CondIF /all/;

Solve CondIF minimizing obj using MIP;

display   obj.l, obj1.l, obj2.l, x_min, x_max, value, value_out, max_min, TA.l, x.l;
```

Appendix F
Output of the Run of the MILP Model for the Solution of the Conditional IF Problem with *N* Intervals with GAMS Software

```
GAMS Rev 229  WIN-VIS 22.9.2 x86/MS Windows             03/12/18 15:20:56 Page 6
G e n e r a l   A l g e b r a i c   M o d e l i n g   S y s t e m
E x e c u t i o n


----    146 VARIABLE obj.L                 =      300.000
VARIABLE obj1.L                =      312.000
VARIABLE obj2.L                =       12.000

----    146 PARAMETER x_min

1   5.000,    2  15.000,    3  30.000,    4  50.000,    5  70.000,    6  90.000
7 105.000


----    146 PARAMETER x_max

1  10.000,    2  20.000,    3  40.000,    4  60.000,    5  80.000,    6 100.000
7 115.000


----    146 PARAMETER value

1  50.000,    2 -50.000,    3  20.000,    4  40.000,    5  80.000,    6  90.000
7  99.000


----    146 PARAMETER value_out            =       -5.000
PARAMETER max_min              =      200.000

----    146 VARIABLE TA.L

1   -5.000,    2   -5.000,    3   50.000,    4   50.000,    5   50.000
6   50.000,    7   -5.000,    8   -5.000,    9   -5.000,    10 -50.000
11 -50.000,    12 -50.000,    13 -50.000,    14 -50.000,    15  -5.000
16  20.000,    17  20.000,    18  20.000,    19  20.000,    20  20.000
21  -5.000,    22  40.000,    23  40.000,    24  40.000,    25  40.000
26  -5.000,    27  80.000,    28  80.000,    29  80.000,    30  80.000
31  80.000,    32  80.000,    33  80.000,    34  -5.000,    35  90.000
36  90.000,    37  90.000,    38  90.000,    39  90.000,    40  90.000
41  90.000,    42  90.000,    43  -5.000,    44  99.000,    45  99.000
46  99.000,    47  99.000,    48  99.000,    49  -5.000,    50  -5.000


----    146 VARIABLE x.L

1    2.000,    2    3.000,    3    6.000,    4    7.000,    5    8.000
6    6.000,    7    3.000,    8    2.000,    9    4.000,    10  16.000
11  17.000,    12  18.000,    13  19.000,    14  20.000,    15   3.000
16  31.000,    17  32.000,    18  33.000,    19  34.000,    20  35.000
21   3.000,    22  55.000,    23  57.000,    24  58.000,    25  60.000
26   3.000,    27  71.000,    28  72.000,    29  73.000,    30  74.000
31  75.000,    32  76.000,    33  77.000,    34   3.000,    35  91.000
36  92.000,    37  93.000,    38  94.000,    39  95.000,    40  96.000
41  97.000,    42 100.000,    43   3.000,    44 107.000,    45 109.000
46 111.000,    47 113.000,    48 115.000,    49   3.000,    50   3.000
```