A NOVEL LINEAR MILP MODEL TO SOLVE KAKURO PUZZLES

José Barahona da Fonseca

Department of Electrical Engineering, Faculty of Sciences and Technology,
New University of Lisbon,
2829-516 Monte de Caparica, Portugal

Abstract: It is shown that a Kakuro puzzle is a NP-Hard problem and very nonlinear since it implies the comparison of segment sums with their desired values, and humans have a lot of difficulty to solve Kakuro puzzles. By the contrary our mixed integer linear programming (MILP) model, using the Cplex solver, solves very hard puzzles in a fraction of a second. We begin to explain why humans have a so great difficulty to solve Kakuro puzzles, even for low level of difficulty ones. Then we review briefly our previous work where we describe *linearization techniques* that allow solving *any* nonlinear problem with a MILP model. Next we describe the constraints and their implementation with the GAMS software and finally we compare the runtimes of our MILP model with previous MILP models using a *black belt* Kakuro puzzle.

Keywords: Artificial intelligence, Operations research, Optimization problems, Mathematical programming, Linear programming.

1. INTRODUCTION

The first problems that Artificial Intelligence solved were *toy problems*, games and more recently *puzzles*. In the eighties there were annual tournaments of chess computer programs and Kasparov was even defeated by one of these chess programs. Recently appeared in Japan the Sudoku and then Kakuro puzzles that were rapidly disseminated through the rest of the world. As an alternative approach to AI in this work we formulate the Kakuro puzzle problem solution as an optimization problem with constraints in the framework of a Mixed Integer Linear Program (MILP) model and then solve it using the Cplex solver with the GAMS software. A Kakuro puzzle consists of a matrix with different length lines and columns, each line and column may have more than one *segment*, and each *segment* must have a specified sum. Moreover each *segment* must have different numbers between 1 and 9. The *runtimes* of the solution of a black belt Kakuro puzzle (Conceptis Puzzles, 2006) using our MILP model are shown to be much smaller than the *runtimes* of previous published MILP models (Simonis, 2008; Davies, 2009).

2. DESCRIPTION OF OUR MILP MODEL TO SOLVE KAKURO PUZZLES

A MILP model is a set of linear constraints, equality and inequality constraints and a linear function, the objective variable that we want to minimize or maximize. Then there is an algorithm, the solver, which integrates all the variables, parameters, constraints and objective variables and searches the optimal solution varying the model variables. We used the GAMS software to implement our MILP model and then we use the Cplex solver in this environment to obtain the optimal solution, i.e. the Kakuro puzzle solution. Although there exist some proposals of MILP models to solve Kakuro puzzle, our proposal differs from the previous in that we use an indexed binary variable to represent the elements of the Kakuro puzzle and the constraints are much more elegant and the runtimes are much smaller. The first and second indexes of the binary variable represent the line and column of the Kakuro matrix element, respectively, and the third index represents the value of the matrix, i.e. there is only value for which the binary variable is one and all the remaining are zero. This way the last index is *translated* into the value of the Kakuro matrix element. This is the main idea to *linearize* this so nonlinear problem. *Empty* elements are *forced* to be zero for all possible values of the third index by *auxiliary constraints*. This way the constraints are much more elegant and simple and the runtimes much smaller than previous MILP models to solve Kakuro puzzles.

The first and fundamental constraint is defined by the following inequality which *logical meaning is Each position of the Matrix have* 1 *or* 0 *values (in the case of being an 'Empty' Position):*

$$\forall_{i,j} \sum_{v\in\{1..9\}} a\_bin(i,j,v) \le 1 \qquad (1)$$

And its implementation with GAMS notation is described by

at_most_one_value(*i,j*).. sum(*v*, a_bin(*i,j,v*))=l=1;

where the operator '=l=' means less or equal.

This latter expression represents a *set of constraints* each one with *all possible instantiations of indexes i and j*. The primitive *sum*(*v*, … means the summation over all possible *instantiations of v* of indexed binary variable a_bin(*i,j,v*)) for a given combination of indexes *i and j*. So *saying* that this sum is less than 1 is equivalent to *say that each position of the Kakuro matrix has at most one value*. This *sounds obvious*, but it must be *declared* to guaranty the logical consistency of the Mathematical Program.

Next we impose the constraint that each line cannot have repetitions. Here we have the problem that each line *l* may has various *segments*, each one identified by the index *seg* and the number of segments identified by the parameter *n_segs_l*(*l*) and the initial column of each segment *seg* being identified by the parameter *ord_min_l*(*l,seg*) and the final column of the segment by the parameter *ord_max_l*(*l,seg*):

$$\forall_{i,v,seg:seg\le n\_segs\_l(i)} \sum_{j:\, j\ge ord\_\min\_l(i,seg)\,\&\, j\le ord\_\max\_l(i,seg)} a\_bin(i,j,v) \le 1 \quad (2)$$

and its implementation with GAMS notation is described by

*all_different_line(i,v,seg)$(***ord***(seg)* **le** *n_segs_l(i))*..
sum(*j*$((**ord**(*j*) **ge** *ord_min_l*(*i,seg*)) **and** (**ord**(*j*) **le** *ord_max_l*(*i,seg*))), *a_bin(i,j,v))***=l=**1;

Note that in this set of constraints the segment order index *seg* is restricted by the dollar operator to be less or equal to the number of *line segments of line i* saved in parameter *n_segs_l(i)*, since the number of segments of each line and column of a Kakuro puzzle varies typically between 1 and 3. For each combination of instantiations *line i and segment seg and value v*, the *sum* operator with the $ operator makes the summation over the columns *j* between *ord_min_l*(*i,seg*) and *ord_max_l*(*i,seg*) that define the segment of order *seg* of *a_bin(i,j,v)* maintaining constant the line index *i* and the value index *v*, and then the *constraint less or equal operator* **=l=** imposes that this sum is less or equal to 1, i.e. it is 0 or 1. If it is 0 means that the value *v* does not exist in the segment *seg* and if it is 1 imposes that value *v* appears only once in segment *seg*.

Next we impose that each segment of each line must has a specified sum saved in the parameter *sum_l(i,seg)*, being each segment defined by the initial column *ord_min_l(i,seg)* and the final column *ord_max_l*(*i,seg)*:

$$\forall i, \forall seg : seg \le n\_segs\_l(i) \quad \sum_{v,j:\, j \ge ord\_min\_l(i,seg) \,\&\, j \le ord\_max\_l(i,seg)} v \;\; a\_bin(i,j,v) = sum\_l(i,seg) \quad (3)$$

which is implemented by the following line of GAMS code:

constraint_sum_l(i,seg)$(ord(seg) le n_segs_l(i))..
sum((*v,j*)$((**ord**(*j*) **ge** *ord_min_l(i,seg))* **and** (**ord**(*j*) **le** *ord_max_l(i,seg))*) , **ord**(*v*)**a_bin(i,j,v)*)=**e**=*sum_l*(*i,seg*);

In the latter expression the *dollar sign* means *sum of argument* **ord**(*v*)**a_bin(i,j,v) restricted to the values of the index j that satisfy the logical condition next to $*. This way for each line *i* and *segment seg* the column index *j* only varies between the two bound limits *ord_min_l(i,seg)* and *ord_max_l(i,seg)*. Since there is only one combination of indexes *i, j* and *v* for which *a_bin(i,j,v)*=1, for each *j*, summing over all possible values of *v* of the argument **ord**(*v*)**a_bin(i,j,v)* will give the value of the Kakuro matrix of line *i* and column *j*. Finally the *equal constraint operator* =**e**= *imposes that the sum of the elements of the segment seg must be equal to the value stored in the indexed parameter sum_l*(*i,seg*). Since the number of segments of each line is variable, in the *beginning of the set of constraints* we use again the $ operator to guarantee that the *segment order index seg* will always be less or equal to the number of segments of line *i* stored in the indexed parameter *n_segs_l(i)*.
Next we impose that each *column segment* cannot has repetitions and then again we impose that each *column segment* must has a sum given by:

$$\forall j, v, \forall seg : seg \le n\_segs\_c(j) \quad \sum_{i \ge ord\_min\_c(j,seg) \,\&\, i \le ord\_max\_c(j,seg)} a\_bin(i,j,v) \le 1 \quad (4)$$

this could be *read* as *for each column j and for each possible value v, this value or does not exist or exist only once*. The implementation of (4) in GAMS code is given by:

all_different_column(j,v,seg)$
(ord(seg) le n_segs_c(j))..
sum(*i*$ ((**ord**(*i*) **ge** *ord_min_c(j,seg))* **and** (**ord**(*i*) **le** *ord_max_c(j,seg)))* , *a_bin*(*i,j,v*))**l**=1;

The dollar operator is the domain restriction operator and *$(ord(seg) le n_segs_c(j))* restricts the domain of variation of *seg* to values less than or equal to *n_segs_c(j)*, i.e. the index *seg* that represents the order of the segment will only assume values between 1 and *n_segs_c(j)*, the number of segments of column *j*.
And again we impose that each *column segment* must has a sum given by the indexed parameter sum_c(*j,seg*):

$$\forall j, seg : seg \le n\_segs\_c(j), \quad \sum_{v,i:\, i \ge ord\_min\_c(j,seg) \,\&\, i \le ord\_max\_c(j,seg)} v \; a\_bin(i,j,v) = sum\_c(j,seg) \quad (5)$$

Finally to prevent *trivial solutions* we must maximize the *objective variable* defined as the number of matrix elements, excluding the *empty positions* which are guaranteed to be zero by a set of auxiliary constraints:

$$obj = \sum_{i \in I,\, j \in J,\, v \in \{1..9\}} a\_bin(i,j,v) \quad (6)$$

which is implemented in GAMS code as:

calc_obj.. *obj*=**e**=**sum**((*i,j,v*), **a_bin**(*i,j,v*));

Next we show a set of auxiliary constraints that guarantee that the *empty position* at line 1 and column 4 be zero, i.e. all the values correspondent to index *v* of the indexed binary variable that defines the Kakuro matrix *a_bin*(‘1’,’4’,*v*)=0:

undefined_value_l1_c1(v).. *a_bin(*'1','4',*v)*=**e**=0*;*

For each of the remaining empty positions we must have a correspondent similar set of auxiliary constraints. In appendix 1 we show the complete GAMS code for the Kakuro puzzle described in the next section where can be found the remaining auxiliary constraints correspondent to the remaining *empty positions*.

3. SOLUTION OF A KAKURO PUZZLE WITH OUR MILP MODEL

In figure 1 we show the Kakuro puzzle taken from (Conceptis Puzzles,2006) that we solved with our MILP model and in figure 2 we show the solution obtained with our MILP model in just few seconds in a PC with 2GHz clock and in appendix 2 we show the output of the GAMS software code. Note that in this *hard* Kakuro puzzles there are two columns with only one segment that turns out the puzzle very difficult due to the combinatorial explosion in the way to fill those columns.

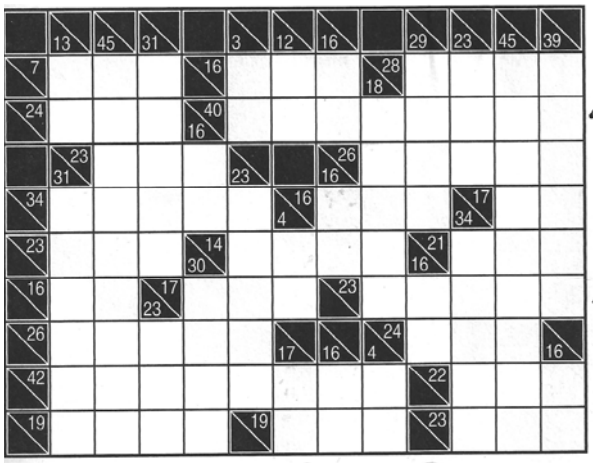

Fig. 1. Hard *black belt* Kakuro puzzle solved by our MILP model. Note the two long columns that contribute to the combinatorial explosion of the number of manners to fill the puzzle.

4. CONCLUSIONS AND FUTURE WORK

We showed that our MILP model to solve Kakuro puzzles is simpler and much more efficient than previous published MILP models. In a near future we plan to expand our MILP model to solve variants of Kakuro puzzles like *Killer Sudoku* and *Kakuro Greater Than* and then adapt them to develop a MILP model to make production planning.

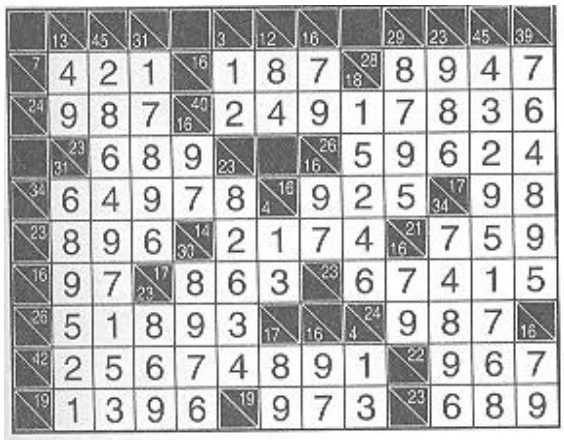

Fig. 2. Solution of the hard *black belt* Kakuro puzzle obtained by our MILP model.

REFERENCES

Conceptis Puzzles (2006). *Black Belt Kakuro*. Sterling Publishing Co., Inc, Toronto, Ontario, Canada.

Davies, R. P. (2009). *An Investigation into the Solution to, and Evaluation of, Kakuro Puzzles*, MSc thesis, University of Glamorgan.

Simonis, H., (2008). Kakuro as a Constraint Problem. In: *Proceedings of Modref Conference*, 2008.

APPENDIX 1. GAMS IMPLEMENTATION OF OUR MILP MODEL TO SOLVE KAKURO PUZZLES

sets *i* /1*9/;
sets *j* /1*12/;
set *seg* /1*3/;
set *v* /1*9/;

binary variable *a_bin(i,j, v);*

Parameter n_segs_l*(i),* n_segs_c*(j),* ord_min_c*(j,seg),* ord_max_c*(j,seg),* ord_min_l*(i,seg),* ord_max_l*(i,seg),* sum_l*(i,seg),* sum_c*(j,seg)*;

****
* Next we will Define a 'Black Belt' Kakuro puzzle 12x9 taken from [??] that GAMS will solve in few seconds and a human takes hours to solve it with a set of parameters
****
****
* First Step: we define the Number of 'Segments' of each Line
****
n_segs_l('1')=3;
n_segs_l('2')=2;
n_segs_l('3')=2;
n_segs_l('4')=3;
n_segs_l('5')=3;
n_segs_l('6')=3;
n_segs_l('7')=2;
n_segs_l('8')=2;
n_segs_l('9')=3;
****
* Second Step: we define the Number of 'Segments' of each Column
****
n_segs_c('1')=2;
n_segs_c('2')=1;
n_segs_c('3')=2;
n_segs_c('4')=2;
n_segs_c('5')=2;
n_segs_c('6')=3;
n_segs_c('7')=3;
n_segs_c('8')=2;
n_segs_c('9')=2;
n_segs_c('10')=2;
n_segs_c('11')=1;
n_segs_c('12')=2;
****
* Third Step: we Define each Segment of Each Line and Its Sum:
****
ord_min_l('1','1')=1;
ord_max_l('1','1')=3;
sum_l('1','1')=7;

*snip: some instructions omitted *

****
* Fourth Step: we Define each Segment of Each Column and Its Sum:
****
ord_min_c('1','1')=1;
ord_max_c('1','1')=2;
sum_c('1','1')=13;

* snip: some instructions omitted*

** Now the Matrix of the Kakuro puzzle is defined...*
** ... But it Remains to 'Define the 'empty positions' with a Set of Auxiliary Constraints*
** undefined_value_li_cj(v)).. a_bin('*li','cj',*v)***=e=**0*;*
****

variable *obj*;

* Each position of the Matrix has 1 or 0 values (in the case of 'Empty' Position):
at_most_one_value(i,j).. *sum(v, a_bin(i,j,v))***=l=**1;

*Each Line Segment (*i,seg*) cannot have repetitions:
all_different_line(i,v,seg)$(ord*(seg)* **le** *n_segs_l(i))..*

sum(*j*$((**ord**(*j*) **ge** *ord_min_l*(*i*,*seg*)) **and** (**ord**(*j*) **le** *ord_max_l*(*i*,*seg*))), *a_bin(i,j,v))*=**l**=1;

**Each Line l, Segment seg, Must has a Sum Equal to sum_l(i,seg):*
constraint_sum_l(i,seg)$(ord(seg) le n_segs_l(i)).\.
sum((*v*,*j*)$((**ord**(*j*) **ge** *ord_min_l(i,seg))* **and** (**ord**(*j*) **le** *ord_max_l(i,seg))*) , **ord**(*v*)**a_bin(i,j,v)*)=**e**=*sum_l*(*i*,*seg*);

**The following constraints define the Empty Positions or Labels of the Kakuro matrix, in this case a 'black belt' Kakuro matrix of very high level of difficulty:*
undefined_value_l1_c1(v).. *a_bin(*'1','4',*v)*=**e**=0*;*
**Position (1,4) is an Empty Position*
undefined_value_l1_c3(v).. *a_bin(*'1','8',*v)*=**e**=0*;*
**Position (1,8) is an Empty Position*
****snip: some auxiliary constraints omitted****

**Each Column Segment (l,seg) cannot have repetitions:*
all_different_column(j,v,seg)$(ord(seg) le n_segs_c(j)).\.
sum(*i*$ ((**ord**(*i*) **ge** *ord_min_c(j,seg))* **and** (**ord**(*i*) **le** *ord_max_c(j,seg)))* , *a_bin*(*i*,*j*,*v*))=**l**=1;

**Each Column c, Segment seg, Must has a Sum Equal to sum_c(c,seg):*
constraint_sum_c(*j*,*seg*)$(ord(*seg*) le n_segs_c(*j*)).\.
sum((*v*,*i*)$((**ord**(*i*) **ge** *ord_min_c*(*j*,*seg*)) **and** (**ord**(*i*) **le** *ord_max_c*(*j*,*seg*))) , ord(*v*) * *a_bin*(*i*,*j*,*v*)) =e= *sum_c*(*j*,*seg*);

** To prevent a Trivial Solution we must maximize the Number of All Matrix Elements*
** that is, the number of ones of the binary variable a_bin(i,j,v)*:
calc_obj.. *obj*=**e**=**sum**((*i*,*j*,*v*), *a_bin*(*i*,*j*,*v*));

model *Kakuro* /**all**/;

solve *Kakuro* **using MIP maximizing** *obj*;

display *a_bin*.l, *obj*.l;

APPENDIX 2. OUTPUT OF THE RUN OF THE MILP MODEL

---- 432 VARIABLE a_bin.L

	1	2	3	4	5	6
1.1				1.000		
1.2		1.000				
1.3	1.000					
1.5	1.000					
1.11				1.000		
2.5		1.000				
2.6				1.000		
2.8	1.000					
2.11			1.000			
2.12						1.000
3.2						1.000
3.8					1.000	
3.10						1.000
3.11		1.000				
3.12				1.000		
4.1						1.000
4.2				1.000		
4.8		1.000				
4.9					1.000	
5.3						1.000
5.5		1.000				
5.6	1.000					
5.8				1.000		
5.11					1.000	
6.5						1.000
6.6			1.000			
6.8						1.000
6.10				1.000		
6.11	1.000					
6.12					1.000	
7.1					1.000	
7.2	1.000					
7.5			1.000			
8.1		1.000				
8.2					1.000	
8.3						1.000
8.5				1.000		
8.8	1.000					
8.11						1.000
9.1	1.000					
9.2			1.000			
9.4						1.000
9.8			1.000			
9.10						1.000

+	7	8	9
1.6		1.000	
1.7	1.000		
1.9		1.000	
1.10			1.000
1.12	1.000		
2.1			1.000
2.2		1.000	
2.3	1.000		
2.7			1.000
2.9	1.000		
2.10		1.000	
3.3		1.000	
3.4			1.000
3.9			1.000
4.3			1.000
4.4	1.000		
4.5		1.000	
4.7			1.000
4.11			1.000
4.12		1.000	
5.1		1.000	
5.2			1.000
5.7	1.000		
5.10	1.000		
5.12			1.000

```
  +         7         8         9

6.1                             1.000
6.2       1.000
6.4                 1.000
6.9       1.000
7.3                 1.000
7.4                             1.000
7.9                             1.000
7.10                1.000
7.11      1.000
8.4       1.000
8.6                 1.000
8.7                             1.000
8.10                            1.000
8.12      1.000
9.3                             1.000
9.6                             1.000
9.7       1.000
9.11                1.000
9.12                            1.000


----    432 VARIABLE obj.L              =       88.000
```

The objective variable *obj*=88 means that this Kakuro puzzle has 88 elements to be find out.