Linear Programming Meets Block-based Languages

Hugo da Gião
University of Minho & HASLab/INESC TEC
Portugal
hugo.a.giao@inesctec.pt

Jácome Cunha
University of Minho & HASLab/INESC TEC
Portugal
jacome@di.uminho.pt

Rui Pereira
HASLab/INESC TEC
Portugal
rui.a.pereira@inesctec.pt

***Abstract*—Linear programming is a mathematical optimization technique used in numerous fields including mathematics, economics, and computer science, with numerous industrial contexts, including solving optimization problems such as planning routes, allocating resources, and creating schedules. As a result of its wide breadth of applications, a considerable amount of its user base is lacking in terms of programming knowledge and experience and thus often resorts to using graphical software such as Microsoft Excel. However, despite its popularity amongst less technical users, the methodologies used by these tools are often *ad-hoc* and prone to errors. To counteract this problem we propose creating a block-based language that allows users to create linear programming models using data contained inside spreadsheets. This language will guide the users to write syntactically and semantically correct programs and thus aid them in a way that current languages do not.**

***Index Terms*—linear programming, spreadsheets, block-based languages, end-user programming**

I. Introduction

The versatility of linear programming in specifying all sorts of problems lends itself useful in many industrial contexts since many of its users have little to no programming or technical knowledge. Thus, visual software such as Microsoft Excel is often the preferred tool when it comes to specifying and solving this type of problem [5].

However the typical methodologies used when solving those types of problems using spreadsheet software often come with underlying problems such as relying on an imprecise process to feed data from the spreadsheet to the solver as well as the difficulties in visualizing as a whole the models. During our research, we found that other tools commonly used by professionals working with linear programming such as MATLAB [9] and GAMS [4] either require considerable programming knowledge or use *ad-hoc* and error-prone methodologies.

Some projects have used visual languages to tackle aspects of linear programming, however, the majority of them focus on the educational and teaching of mathematical aspects of linear programming [14, 6], and the few existing projects focusing on the applied side of linear programming tend to be several decades old and have dated and unappealing interfaces and do not make use of recent advances in the field of visual languages and human-centered computing [7, 15].

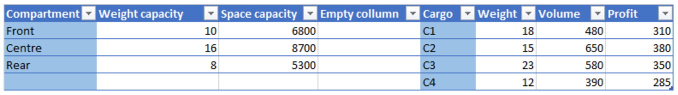
This work is supported by the national funds through the Portuguese Funding Agency (FCT - Fundação para a Ciência e a Tecnologia, within project UIDB/50014/2020).

Numerous projects have applied visual languages to various areas of computing with the focus on increasing accessibility to novice and non-technical users as well as teaching. A considerable amount of these languages use the Blockly [2] framework for their implementation. Examples include BlockPy, a web-based platform that lets the user write and run Python code using a block-based language [1], and Scratch, a block-based visual programming language and educational tool targeted at children [8].

Given the potential of Blockly to improve the usability and practice of linear programming and the extensive study of block-based languages and their practices [3, 13], we aim to build upon the work previously done in this field to the create a visual language and tool capable of expressing linear programming models in a user-friendly manner.

II. A block-based language for linear programming

In this section, we introduce our proposed language using an example featured in a Master of business administration exam (MBA) [10]. The example problem aims to increase the profit of delivery airplanes. The problem statement provides values for the weight and space capacity of three different compartments (front, rear and center) and maximum values for the weight, volume and profit for four different cargoes (C1, C2, C3 and C4) as seen in Figure 1.



Compartment	Weight capacity	Space capacity	Empty collumn	Cargo	Weight	Volume	Profit
Front	10	6800		C1	18	480	310
Centre	16	8700		C2	15	650	380
Rear	8	5300		C3	23	580	350
				C4	12	390	285

Fig. 1. Input from our example problem in the specified format

A. Input data

Our solution requires the input data to follow a predefined structure. This structure allows for the definition of index columns (as seen highlighted in blue in Figure 1), to reference values and iterate over the data columns (seen in white in the same figure). To distinguish between the two we assume that the data columns addressed by a given index column appear in the spreadsheet immediately after the said column, and that the sets of index and data columns are separated by an empty column as can be seen in the figure. In this case there are two sets, the first being for the three compartments and the second for the four types of cargo.

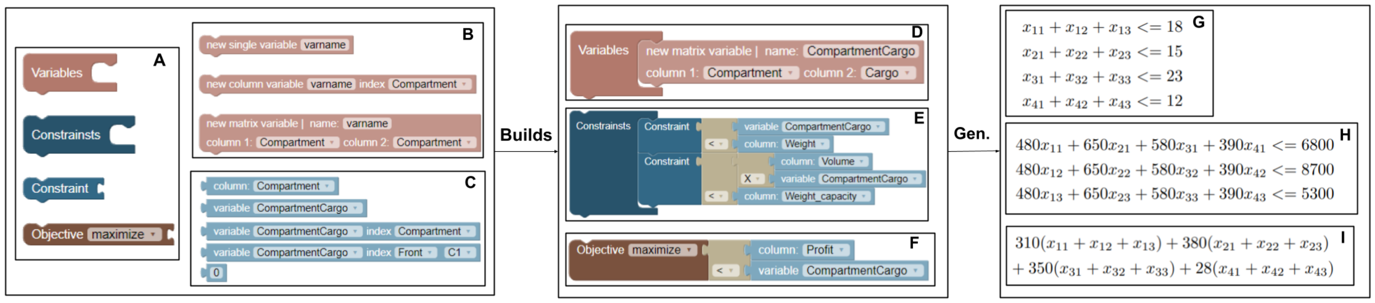

A
Variables
Constraints
Constraint
Objective maximize
B
new single variable varname
new column variable varname index Compartment
new matrix variable name: varname column 1: Compartment column 2: Compartment
C
column: Compartment
variable CompartmentCargo
variable CompartmentCargo index Compartment
variable CompartmentCargo index Front C1
0
Builds
D
Variables new matrix variable name: CompartmentCargo column 1: Compartment column 2: Cargo
E
Constraints
Constraint
variable CompartmentCargo
column: Weight
Constraint
column: Volume
variable CompartmentCargo
column: Weight_capacity
F
Objective maximize
column: Profit
variable CompartmentCargo
Gen.
G
$x_{11}+x_{12}+x_{13}<=18$
$x_{21}+x_{22}+x_{23}<=15$
$x_{31}+x_{32}+x_{33}<=23$
$x_{41}+x_{42}+x_{43}<=12$
H
$480x_{11}+650x_{21}+580x_{31}+390x_{41}<=6800$
$480x_{12}+650x_{22}+580x_{32}+390x_{42}<=8700$
$480x_{13}+650x_{23}+580x_{33}+390x_{43}<=5300$
I
$310(x_{11}+x_{12}+x_{13})+380(x_{21}+x_{22}+x_{23})$
$+350(x_{31}+x_{32}+x_{33})+28(x_{41}+x_{42}+x_{43})$

Fig. 2. The language constructs are used to **build** an example model which is used to **generate** the mathematical format

B. *Main language building blocks*

The building blocks of a linear programming model seen in Figure 2.A are the variables, the constraints, and the objective function. In our language we include two nesting blocks for the variables and constraints, an objective block with the option to minimize or maximize a given objective and an intermediate block for the individual constraints since constructing the constraints requires the use of several blocks. To further facilitate this process for novice and inexperienced users, when building a new linear programming model, the variables, constraints and objective block are connected when creating a fresh solution.

C. *Defining variables*

Users have several options to define variables (Figure 2.B):

- a single variable through its name;
- a column variable defining its name and an index column for which the variable will be iterated and accessed;
- a matrix variable that take a name and two index columns for which it can be iterated and the values accessed.

These can then be used through the variable blocks (Figure 2.B), in different ways. In the example seen in Figure 2.D we use a matrix variable block to create a new $N \times M$ matrix variable named `CompartmentCargo` with N being the length of the column `Compartment` and M the length of the column `Cargo`.

D. *Defining constraints*

Each constraint is defined by dragging a constraint block inside the constraints block (second and third blocks from the top in 2.A) and then using the value blocks (blocks in 2.C) and operation blocks (blocks following the Constraint Block in 2.E) to express the constraints. In our language, operation blocks are used to join value and other operation blocks. This blocks can express several operations including arithmetic operations and inequalities. The value blocks can represent columns, previously defined variables, and numbers. The variables can be accessed using different blocks and options which influence how the constraints will be generated. As an example a user can access a matrix variable with a single slot variable block in Figure 2.E to generate multiple constraints or use the three slot variable blocks to access a particular value of the given variable.

The first constraint in Figure 2.E, expressed in natural language as “one cannot pack more of each of the four cargoes than one has available” is defined in our language by using a $<=$ operation block, a variable block with the option `CompartmentCargo` and a column block with the option `Weight`. Since the constraints block only appears after the variables block the compiler knows the index values for both the column and variable used and thus can generate the correct constraints which in this case are expressed in Figure 2.G.

The second constraint can be expressed in natural language as “the volume (space) capacity of each compartment must be respected”. This constraint (the second in 2.E) uses `X` and $<=$ operation blocks and value blocks to compile the more complex constraint. For these constraints, our compiler generates the linear programming constraints featured in Figure 2.H.

E. *Defining the objective function*

To define the objective function users must fit the objective block into the constraints block and use several value and operation blocks to define the function.

In the example seen in Figure 2.G the objective function is created by using an operation block with value $<=$, a column block with option `Profit`, and a variable block with the option `CompartmentCargo`. The objective function generated by this statement is the one featured in 2.I.

F. *User interaction*

This language will extend previous work exploring the creation of spreadsheets through systematic processes [11, 12] thus increasing the number of operations available. The user will use the spreadsheet with the linear programming data as input to a spreadsheet creation process and define the linear programming model using our language which will use the spreadsheet as input.

III. Concluding Remarks

In this work we present a language to aid end users defining linear programming models. With our language, users are forced to create correct programs as the constructs are based on the inputs of the problems. However, it is still possible to build models with problems and thus we intend to include error-handling in the support tool. We will also design and run empirical evaluations to assess the usability of the language.

References

[1] Austin Cory Bart et al. “BlockPy: An Open Access Data-Science Environment for Introductory Programmers”. In: *Computer* 50.5 (2017), pp. 18–26. DOI: 10.1109/MC.2017.132.

[2] Blockly. URL: https://developers.google.com/blockly.

[3] Neil Fraser. “Ten things we’ve learned from Blockly”. In: *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*. 2015, pp. 49–50. DOI: 10.1109/BLOCKS.2015.7369000.

[4] GAMS. URL: https://www.gams.com.

[5] Hector Guerrero. *Excel Data Analysis: Modeling and Simulation*. Springer, 2010. URL: https://www.springer.com/gp/book/9783642108341.

[6] Vassilios Lazaridis et al. “Visual LinProg: A web-based educational software for linear programming”. In: *Comput. Appl. Eng. Educ.* 15.1 (2007), pp. 1–14. DOI: 10.1002/cae.20084.

[7] Pai-Chun Ma, Frederic H. Murphy, and Edward A. Stohr. “A Graphics Interface for Linear Programming”. In: *Commun. ACM* 32.8 (Aug. 1989), pp. 996–1012. ISSN: 0001-0782. DOI: 10.1145/65971.65978.

[8] John Maloney et al. “The Scratch Programming Language and Environment”. In: *ACM Trans. Comput. Educ.* 10.4 (Nov. 2010). DOI: 10.1145/1868358.1868363.

[9] MATLAB. URL: https://www.mathworks.com/help/optim/ug/linprog.html.

[10] MBA. URL: http://people.brunel.ac.uk/~mastjjb/jeb/jeb.html.

[11] Jorge Mendes et al. “Systematic spreadsheet construction processes”. In: *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 2017, pp. 123–127. DOI: 10.1109/VLHCC.2017.8103459.

[12] Jorge Mendes et al. “Towards systematic spreadsheet construction processes”. In: *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. 2017, pp. 356–358. DOI: 10.1109/ICSE-C.2017.141.

[13] Erik Pasternak, Rachel Fenichel, and Andrew N. Marshall. “Tips for creating a block language with blockly”. In: *2017 IEEE Blocks and Beyond Workshop (B B)*. 2017, pp. 21–24. DOI: 10.1109/BLOCKS.2017.8120404.

[14] José Pereira and Susana Fernandes. “Two-variable Linear Programming: A Graphical Tool with Mathematica”. In: *SYMCOMP 2013 - 1st International Conference on Algebraic and Symbolic Computation*. Sept. 2013, pp. 159–173.

[15] E.L.F. Senne, C. Lucas, and S. Taylor. “Towards an Intelligent Graphical Interface for Linear Programming Modelling”. In: *Journal of Intelligent Systems* 6.1 (1996), pp. 63–94. DOI: doi:10.1515/JISYS.1996.6.1.63.