Universidade do Minho
Escola de Engenharia

Hugo Afonso da Gião

LPBlocks - A Block-based Language for Linear Programming

November, 2021

Universidade do Minho
Escola de Engenharia

Hugo Afonso da Gião

LPBlocks - A Block-based Language for Linear Programming

Master's Dissertation
Master's in Informatics Engineering

Work supervised by
Jácome Cunha
Rui Pereira

November, 2021

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the Universidade do Minho.

Barcelos, 22/11/2021

(Location) (Date)

[illegible]

(Hugo Afonso da Gião)

Acknowledgements

I will like to thank the Univesity of Minho and HASLab/INESC TEC, my advisors Jácome Cunha and Rui Pereira for offering their guidance and experience, as well as all the volunteers that participated in our study and every person that helped in this project.

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.

Block-based languages have been successfully used to aid novice programmers and even children in programming. Thus, we created a block-based programming language termed LPBlocks that allows users to create linear programming models using data contained inside spreadsheets. This language guides the users to write syntactically and semantically correct programs and thus aid them in a way that current languages do not.

To assess the validity of LPBlocks we used it to express numerous and varied linear programming problems. Further, we implemented LPBlocks and contacted possible participants of different backgrounds, designed and ran a study to collect data referent to their experience with our tool and language. We then used those insights to evaluate LPBlocks and to use it as a marker for possible improvements.

Keywords: linear programming, operations research, optimization, block-based languages, visual languages, Blockly...

Resumo

Programação linear é um conjunto de técnicas de otimização matemática utilizada em várias áreas estas incluem matemática, economia, ciências da computação e usos em contextos industriais, incluindo planear rotas, alocar recursos e planear horários. Como resulta das suas aplicações variadas uma grande quantidade dos seus utilizadores não possuem conhecimentos de programação e por isso utilizam software gráfico como o Microsoft Excel. Apesar da sua popularidade este software utiliza metodologias *ad-hoc* e propicias a erros.

As linguagem de programação por blocos tem surgido nos últimos anos com o intuito de ajudar programadores iniciantes, tendo mesmo aplicações no ensino de crianças. Sendo assim nos criamos uma linguagem de programação pro blocos que utiliza dados contidos em folhas de calculo para criar modelos de programação linear chamada LPBlocks. Esta linguagem guia utilizadores na criação de modelos semanticamente e sintaticamente corretos.

Para avaliar a validade de LPBlocks nos implementamos vários problemas utilizando a mesma. Posteriormente implementamos esta linguagem e utilizamo-la num estudo com utilizadores de vários níveis de experiência. Depois utilizamos a informação recolhida durante o estudo para avaliar LPBlocks e propor melhorias.

Palavras-chave: programação linear, investigação operacional, linguagens de blocos, linguagens visuais, Blockly ...

Contents

List of Figures

List of Tables

Listings

List of Algorithms

Acronyms

LP	linear programming

Symbols

Chapter 1

Introduction

This chapter presents some background information about the field of LP, its uses as well as some of its underlying problems and approaches used in different fields to tackle those problems this is done in Subsection 1.1. In Subsection 1.2 we describe some of our scientific and technical contributions. Further in Subsection 1.3 we pose several questions that we intend on answering within this work and in Subsection 1.4 we describe the organization of this document.

1.1 Motivation

LP is a mathematical optimization technique used to find the best possible outcomes in problems specified by linear specification constraints. It originated as a discipline during the 1940s as an effort to tackle complex problems associated with wartime operations. These methods have since found uses in many areas in fields such as mathematics, computer science, business, economics, and engineering and problems such as planning, routing, scheduling, assignment, and design [13].

There are many steps involved in the LP workflow, the first of them is understanding the problem at hand, what value does the user want to know, what does this value depends upon. The next steps are defining the decision variables according to the problem, writing an objective function, defining the constraints, writing the constraints in terms of decision variables, and adding nonnegativity to the constraints [1].Currently, many tools support this process, these tools cater to both advanced users and less technical users. The versatility of LP in formulating all sorts of problems lends itself useful in many industrial contexts from schedule optimization to route planning. Since many of its users have little to no programming or technical knowledge, visual software such as Microsoft Excel is often the preferred tool when it comes to specifying and solving this type of problem [10].

Despite its widespread use end-users still face some obstacles when trying to leverage these tools since existing libraries and tooling often require some knowledge of programming to achieve desired outcomes (as seen in tools like MATLAB [1] and SAS/OR [2] and various libraries such as Googles OR-Tools [3] and PuLP [4] and more visual tools often use spreadsheet or spreadsheet-like software with the methodology and problems underlaying its use, learning LP also comes with some difficulties that have been previously mitigated with the use of technology [6].

Spreadsheet software such as Excel allows users to formulate and solve optimization and linear and nonlinear optimization problems using algorithms such as simplex, generalized reduced gradient, and evolutionary algorithms [23] [24]. One of the reasons motivating the use of spreadsheets for LP and optimization tasks stands in the prevalence of spreadsheet software in business settings and its use in tasks related to the LP and optimization process. These tasks include data gathering, transformation, and analysis. Their use of the aforementioned software allows users to use maintain their habits and previous knowledge and reduce the learning curve for such tasks. However similarly to other Excel tasks this methodology is also *ad-hoc* and could cause problems to novice users due to input errors at the moment of inputting the variables as well as other errors in the spreadsheet building process due to its methodology. Some of the underlying problems include adding constraints and optimization function from the spreadsheet to the solver, the difficulty in visualizing the complete model, not being able to define constraints by its columns, the inability to call variables by name instead of its cell position, and lack of an interactive model and building process.

Spreadsheets are widely used across the enterprise market and spreadsheet software such as Excel is used for numerous tasks such as data loading, transformation, and analysis and in fields such as Finance and Engineering. However despite that Excel, the most widespread of spreadsheet software, being popular to the point that Excel formulas are the most used programming language in the enterprise market [17], possesses a methodology that is *ad-hoc* and lacking when it comes to robustness, re-usability, and other features available for users of more standard programming languages. Some of the problems it encompasses are the missing concept of types and types checking which can lead to an inappropriate data flow between computation steps [11] and another being lack of expressiveness which leads to harder traceability and spreadsheet maintenance [11].

Studies show that the use of programming languages, block-based or not, often endears students and represents one of the main obstacles when learning computer science [19]. Taking the precedent into account it is paramount to make visual languages the most appealing, intuitive, and easy to use as possible. Some projects have used visual languages to tackle aspects of LP, however, the majority of them focus on the educational and teaching of mathematical aspects of LP [12, 22]. The few existing projects focusing on the applied side of LP 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 [14, 25].

Numerous projects have applied visual languages to various areas of computing generally focused

[1] `https://www.mathworks.com/products/matlab.html`
[2] `https://www.sas.com/en_us/software/or.html`
[3] `https://developers.google.com/optimization`
[4] `https://pypi.org/project/PuLP/`

on increasing accessibility of novice and non-technical users as well as teaching. A considerable amount of these languages use the *Blockly* framework for their implementation [20]. These languages include *BlockPy* [3], a web-based platform that lets the user write and run *Python* code using a block-based language, and Scratch [16], a block-based visual programming language and educational tool mostly targeted at children.

1.2 Contributions

In this section, we list our contributions in both scientific and technological terms a well as scientific publications. Our contributions from this thesis focused mainly on the creation of a visual, block-based language capable of expressing LP problems and a web application that allows users to build, debug and run models using our language. The scientific publications associated with our work are the following:

- Linear Programming Meets Block-based Languages Hugo Gião, Rui Pereira, Jácome Cunha IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'21) I
- Towards a Block-based Language for Linear Programming? Hugo Gião, Rui Pereira, Jácome Cunha 12th National Symposium of Informatics (INForum'21) II
- LPBlocks implementation: source code [5], tool [6]

1.3 Research questions

Considering our motivation and goals our thesis aims to answer the following questions:

- **RQ1** - *Can we use block-based languages to represent LP formulations.* - Currently the tools used in both education and industry to create LP models either require pre-existing programming knowledge, use *ad-hoc* methodologies, or can not be used for formulating general models. Since LP has several uses improving its process for less technical users could be beneficial in improving their productivity and capabilities.
- **RQ2** - *Will using a block-based language provide users an easier environment for LP.* - Even if such a language is possible, the language wouldn't be inherently better than the existing solutions, to evaluate the created language we need to test it with users with no experience with the said language.

[5]https://github.com/h4g0/blockly-spaces
[6]https://lpblocks.herokuapp.com/

1.4 Document organization

To present our work, the document is organized as follows:

- Chapter 2 showcases the state of the art when it comes to end-user tooling for LP, visual and block-based languages, and projects that bridge visual languages and LP.
- Chapter 3 In this Chapter we showcase LPBlocks, a block-based language capable of expressing LP models. We explain how the data is received, the different blocks, and how to use them.
- Chapter 4 In this Chapter we elaborate on the implementation details and compilation process for our language. We also showcase our interface and its features.
- Chapter 5 In this Chapter to showcase the power and wide applicability of our language by using it to solve a selection of problems taken from class notes and operations research textbooks.
- Chapter 6 To not only prove that our language can be used to formulate LP problems we executed a user study in which the users after a brief tutorial solve several LP problems using our tool and then answer a survey where they share their experiences.
- Chapter 7 Here we analyze the results of our work and purpose some possible improvements and continuation to this project.

Chapter 2

State of the art

In this chapter, we expose a variety of tools related to our work. To do so we started by researching capable and some of the more user-friendly tools for operations research and LP purposes this is the work done in Section 2.1. In the same Section after exposition the different solutions and some of their methodologies, these tools are not necessarily the most popular and widely used in industry for the purposes above stated and are not necessarily the most widely used since the more programmatic methodology is often preferred. We proceed to do a comparison and comment on which would be the most appropriate for different user bases according to different aspects such as cost, price, and features and then discuss some of their shortcomings and possible solutions.

In Section 2.2 we present some existing visual languages projects, these projects tackle various development and computing areas such as data science, application creation, and spreadsheet manipulating with applications in both enterprise and education markets.

Then in Section 2.3 we present some works that apply visual languages and human-centered computing to the field of LP. The projects approached in this Section focus on both educational and applied aspects of LP. Some of them in helping novice users use and learn LP and others for more experienced users.

2.1 Operations research tooling for end users

2.1.1 Excel

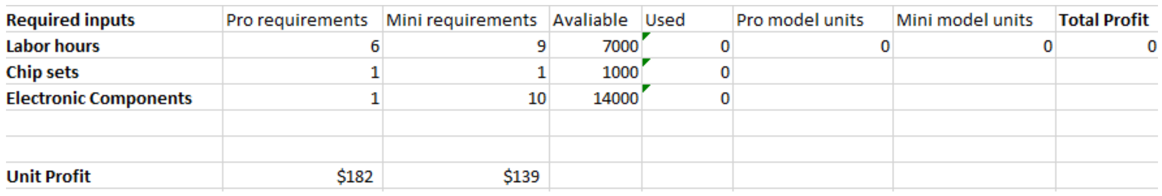
Microsoft Excel is a spreadsheet software developed by Microsoft with support for various platforms such as Windows, macOS, Android, and iOS it boasts many features related to manipulating and analyzing spreadsheets as well as various features related to business, engineering, and data analytics. Excel is closed source and paid for all users except students[1].

Excel[2] allows for formulating and successive solving of LP problems using its add-in solver. Using excel for this task comes with some caveat, one of them being that the methodology used for adding the constraints, variables, and the objective is *ad-hoc*, its interface not being user friendly, not having predefined templates for specific use cases and the solver not being native to Excel.

To be able to solve LP problems in Excel first we need to download the Solver Add-in to do so first go to **File > Options** click **Add-Ins** then **Manage** and **Add-ins**,click **Go** and select the **Excel solver add-in**.

Having the solver add-in installed we can move to the next step, for demonstration purposes we will be using a tablet manufacturing example, this problem states that a tablet manufacturer has to optimize the production of two tablet models to maximize profits. This example was sourced from youtube [15].

To solve this problem we first define the problem variables, constraints, and optimization function inside an Excel spreadsheet as seen in Figure 2.1. In this case, we define the requirements in terms of **Labour hours**, **Chips sets** and **Electronic components** for each of the models as well as their availability. We then define two variable one called **Pro Model units** and another called **Mini Model units** indicating the number of units of each model to be produced. After we define the number of units for each of the requirements as the requirements for each of the model multiplied by the number of produced units, we then define the constant **Unit Profit** for both models and the objective function **Total profit** defined as being the *sumproduct* of each of the units with its profit.



Required inputs	Pro requirements	Mini requirements	Avaliable	Used	Pro model units	Mini model units	Total Profit
Labor hours	6	9	7000	0	0	0	0
Chip sets	1	1	1000	0			
Electronic Components	1	10	14000	0			
Unit Profit	$182	$139					

Figure 2.1: Definition of variables and constraints in the spreadsheet

After defining the problem inside the Excel spreadsheet we open the solver add-in by clicking in **Data > Sovler** inside the Excel menu as seen in figure 2.2. We then are prompted with a window in which we add the different variables, constants, objective function and choose from one of the three available solvers as seen in figure2.3.

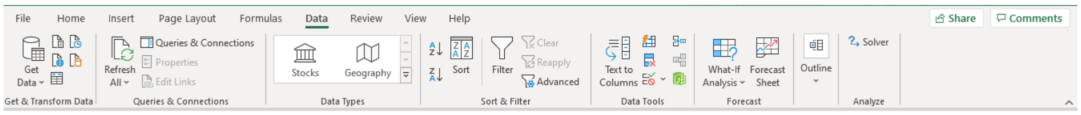
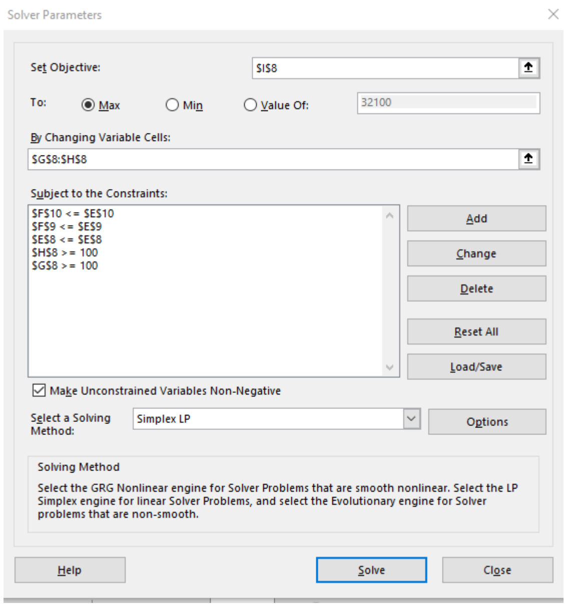


Figure 2.2: Find the solver option in excel

After solving our problem the Excel solver add-in outputs both the solutions seen in figure 2.4 and a report detailing the problem specification as well as some of the steps and details about the algorithm used, seen in 2.5.

[1]https://www.microsoft.com/en-us/microsoft-365/p/excel/cfq7ttc0k7dx?activetab=pivot%3aoverviewtab

[2]https://office.live.com/start/excel.aspx



Set Objective: $I$8

To: [✓ Max] Min Value Of: 32100

By Changing Variable Cells: $G$8:$H$8

Subject to the Constraints:

$F$10 <= $E$10
$F$9 <= $E$9
$E$8 <= $E$8
$H$8 >= 100
$G$8 >= 100

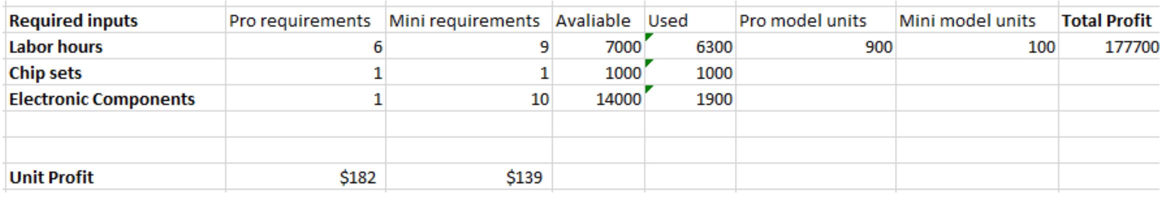
[✓] Make Unconstrained Variables Non-Negative

Select a Solving Method: Simplex LP

Solving Method

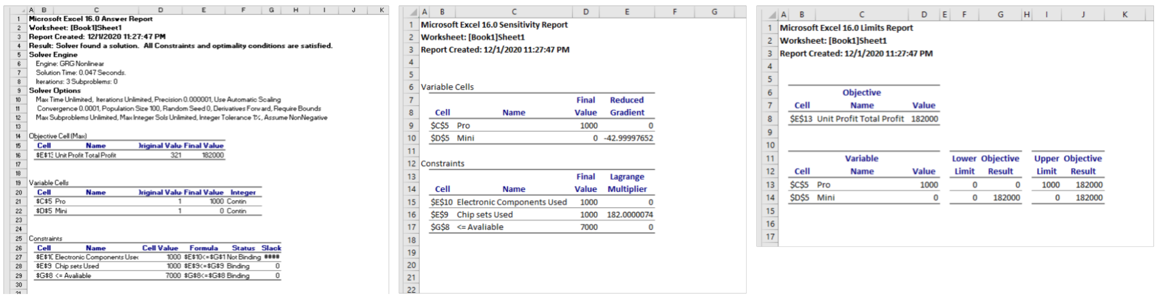
Select the GRG Nonlinear engine for Solver Problems that are smooth nonlinear. Select the LP Simplex engine for linear Solver Problems, and select the Evolutionary engine for Solver problems that are non-smooth.

Figure 2.3: Adding variables,constraints and defining the objective in the excel solver.



Required inputs	Pro requirements	Mini requirements	Avaliable	Used	Pro model units	Mini model units	**Total Profit**
Labor hours	6	9	7000	6300	900	100	177700
Chip sets	1	1	1000	1000			
Electronic Components	1	10	14000	1900			
Unit Profit	$182	$139					

Figure 2.4: Results outputted from the solver into the spreadsheet.



Microsoft Excel 16.0 Answer Report
Worksheet: [Book1]Sheet1
Report Created: 12/1/2020 11:27:47 PM
Result: Solver found a solution. All Constraints and optimality conditions are satisfied.
Solver Engine
Engine: GRG Nonlinear
Solution Time: 0.047 Seconds.
Iterations: 3 Subproblems: 0
Solver Options
Max Time Unlimited, Iterations Unlimited, Precision 0.000001, Use Automatic Scaling
Convergence 0.0001, Population Size 100, Random Seed 0, Derivatives Forward, Require Bounds
Max Subproblems Unlimited, Max Integer Sols Unlimited, Integer Tolerance 1%, Assume NonNegative

Objective Cell (Max)

Cell	Name	Original Valu	Final Value
$E$13	Unit Profit Total Profit	321	182000

Variable Cells

Cell	Name	Original Valu	Final Value	Integer
$C$5	Pro	1	1000	Contin
$D$5	Mini	1	0	Contin

Constraints

Cell	Name	Cell Value	Formula	Status	Slack
$E$10	Electronic Components Usec	1000	$E$10<=$G$1	Not Binding	####
$E$9	Chip sets Used	1000	$E$9<=$G$9	Binding	0
$G$8	<= Avaliable	7000	$G$8<=$G$8	Binding	0

Microsoft Excel 16.0 Sensitivity Report
Worksheet: [Book1]Sheet1
Report Created: 12/1/2020 11:27:47 PM

Variable Cells

		Final	Reduced
Cell	Name	Value	Gradient
$C$5	Pro	1000	0
$D$5	Mini	0	-42.99997652

Constraints

		Final	Lagrange
Cell	Name	Value	Multiplier
$E$10	Electronic Components Used	1000	0
$E$9	Chip sets Used	1000	182.0000074
$G$8	<= Avaliable	7000	0

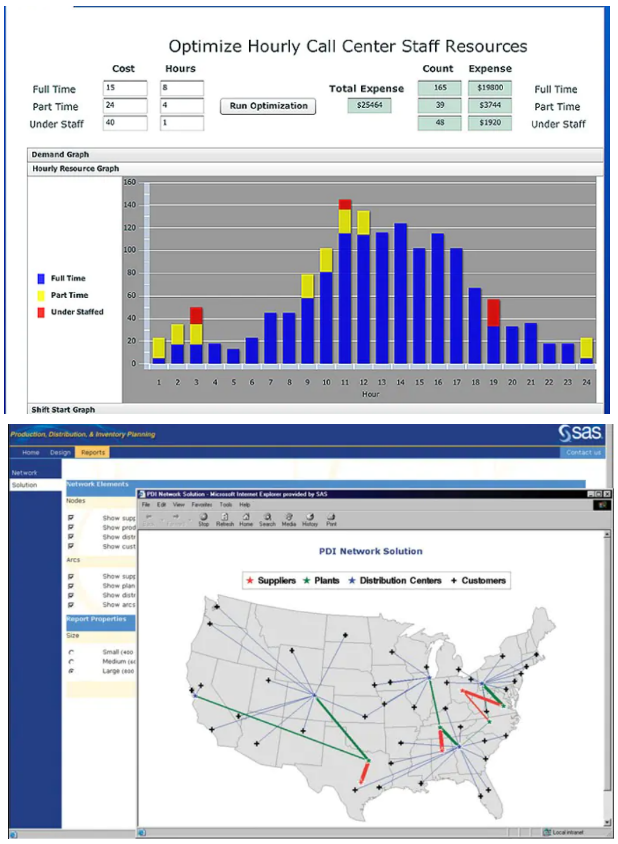
Microsoft Excel 16.0 Limits Report
Worksheet: [Book1]Sheet1
Report Created: 12/1/2020 11:27:47 PM

	Objective	
Cell	Name	Value
$E$13	Unit Profit Total Profit	182000

	Variable		Lower	Objective	Upper	Objective
Cell	Name	Value	Limit	Result	Limit	Result
$C$5	Pro	1000	0	0	1000	182000
$D$5	Mini	0	0	182000	0	182000

Figure 2.5: Excel LP report

2.1.2 SAS/OR


Optimize Hourly Call Center Staff Resources

	Cost	Hours
Full Time	15	8
Part Time	24	4
Under Staff	40	1

Total Expense $25464

Count	Expense	
165	$19800	Full Time
39	$3744	Part Time
48	$1920	Under Staff

Figure 2.6: SAS/OR graphical modelling capabilities.

The SAS/OR software allows its users to use various state of the art optimization, simulation, and scheduling algorithms and software. User interfaces and functionalities vary greatly for the different SAS/OR features this are:

- **Mathematical optimization** Users can make use of this feature using an intuitive algebraic language[3]. On a technical aspect, the SAS/OR software includes a diverse array of features this includes:
  - Aggressive presolvers capable of reducing problem sizes[3].
  - Multithreading and other technologies for improved performance[3].
  - Decomposition algorithm for linear and mixed-integer programming[3].
  - Multiple network diagnostics and optimization algorithms[3].
  - Parallel hybrid global/local search optimization, including multi-objective optimization[3].
  - Constraint programming capabilities with scheduling and resource features[3].
- **Discrete event simulation** For this feature SAS/OR provides it's users friendly interfaces.When using this feature users get:

[3]https://www.sas.com/pt_pt/software/or.html#m=features

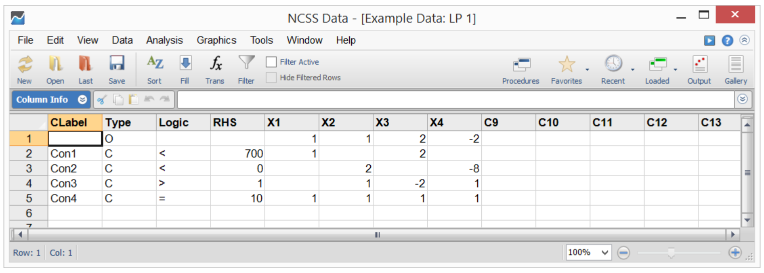
  - Graphical modelling capabilitites[3] as seen in figure 2.6.
  - Support for large models and experiments.
  - Hierarchical modeling: compound blocks and submodel blocks[3].
  - Search facility enables the search of all blocks in the model[3].
  - Integration with SAS and JMP[3].

- SAS/OR users have as well access to an environment to **track, manage and plan project and resource schedules**[3]. When using this environment users have access to:
  - Critical Path Method and CPM-based resource-constrained scheduling[3].
  - Versatile reporting, customizable Gantt charts and project network diagrams[3].
  - Calendars, work shifts, and holidays for determining resource availability and schedules[3].
  - Decision analysis[3].
  - Bill of material (BOM) processing[3].

This software is used across various industries and companies such as Nestlé, Volvo, and Honda. SAS/OR is paid and closed source with a free trying period and discounted or free for educational purposes[4].

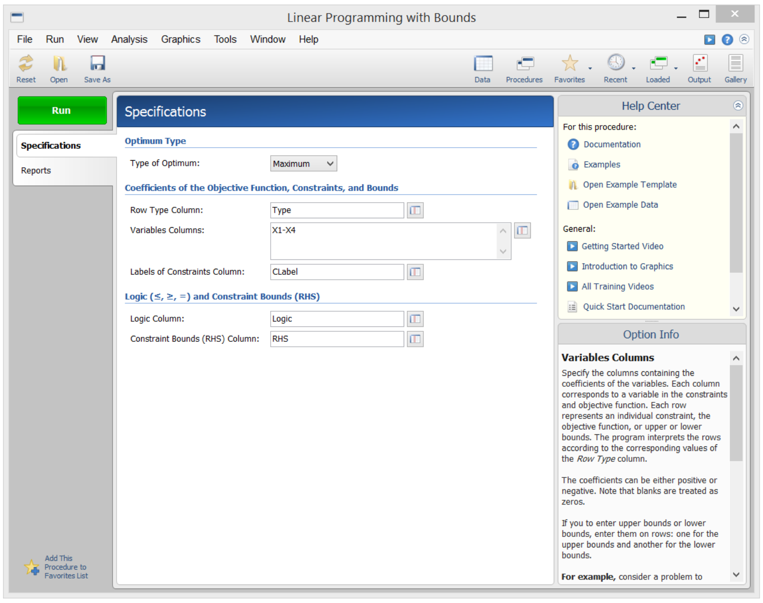
[4] https://www.sas.com/pt_pt/software/how-to-buy/request-price-quote.html

2.1.3 NCSS


NCSS Data - [Example Data: LP 1]
File Edit View Data Analysis Graphics Tools Window Help
New Open Last Save Sort Fill Trans Filter
Filter Active
Hide Filtered Rows
Procedures Favorites Recent Loaded Output Gallery
Column Info
Row: 1 Col: 1
100%

	CLabel	Type	Logic	RHS	X1	X2	X3	X4	C9	C10	C11	C12	C13
1		O			1	1	2	-2					
2	Con1	C	<	700	1		2						
3	Con2	C	<	0		2		-8					
4	Con3	C	>	1		1	-2	1					
5	Con4	C	=	10	1	1	1	1					
6													

(a) Problem construction inside the spreadsheet.



Specifications

Optimum Type

Type of Optimum: Maximum

Coefficients of the Objective Function, Constraints, and Bounds

Row Type Column: Type

Variables Columns: X1-X4

Labels of Constraints Column: CLabel

Logic (≤, ≥, =) and Constraint Bounds (RHS)

Logic Column: Logic

Constraint Bounds (RHS) Column: RHS

Variables Columns

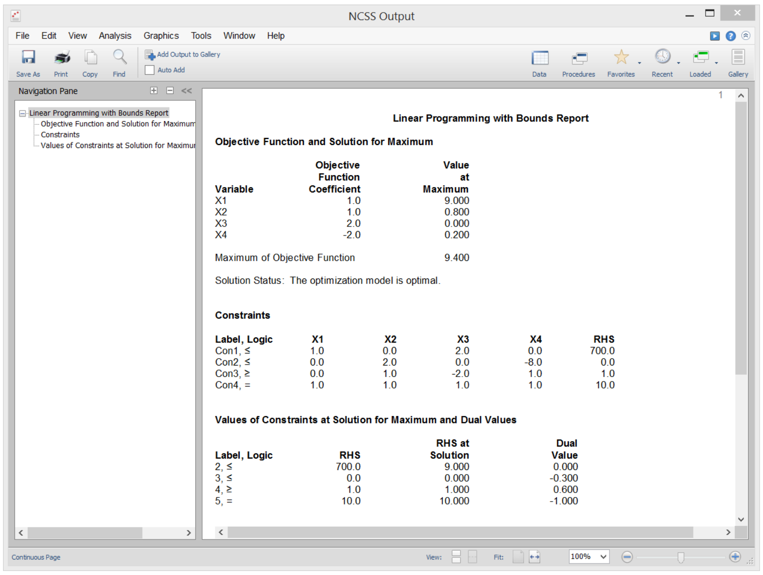
Specify the columns containing the coefficients of the variables. Each column corresponds to a variable in the constraints and objective function. Each row represents an individual constraint, the objective function, or upper or lower bounds. The program interprets the rows according to the corresponding values of the *Row Type* column.

The coefficients can be either positive or negative. Note that blanks are treated as zeros.

If you to enter upper bounds or lower bounds, enter them on rows: one for the upper bounds and another for the lower bounds.

For example, consider a problem to

(b) Adding the problem to the solver.


NCSS Output
File Edit View Analysis Graphics Tools Window Help
Save As Print Copy Find
Add Output to Gallery
Auto Add
Data Procedures Favorites Recent Loaded Gallery
Navigation Pane
Linear Programming with Bounds Report
Objective Function and Solution for Maximum
Constraints
Values of Constraints at Solution for Maximum
Continuous Page
View: Fit: 100%

```
                 Linear Programming with Bounds Report

Objective Function and Solution for Maximum

                 Objective             Value
                  Function                at
Variable       Coefficient           Maximum
X1                     1.0             9.000
X2                     1.0             0.800
X3                     2.0             0.000
X4                    -2.0             0.200

Maximum of Objective Function          9.400

Solution Status:  The optimization model is optimal.


Constraints

Label, Logic     X1         X2         X3         X4        RHS
Con1, ≤         1.0        0.0        2.0        0.0      700.0
Con2, ≤         0.0        2.0        0.0       -8.0        0.0
Con3, ≥         0.0        1.0       -2.0        1.0        1.0
Con4, =         1.0        1.0        1.0        1.0       10.0


Values of Constraints at Solution for Maximum and Dual Values

                                      RHS at              Dual
Label, Logic           RHS          Solution             Value
2, ≤                 700.0             9.000             0.000
3, ≤                   0.0             0.000            -0.300
4, ≥                   1.0             1.000             0.600
5, =                  10.0            10.000            -1.000
```

(c) Solver output.

Figure 2.7: LP problem solving in NCSS

NCSS is a statistical and graphical program with linear and mixed integer programming capabilities, it uses Tableau or Bounds for data visualization and analysis[5].

The NCCSS software has many use-cases and features for operations research this includes:

- **LP** - NCSS allows for the use of LP for the maximization or minimization of a linear objective function subject to several constraints[5].
- **Mixed Integer Programming** - This technique builds upon LP adding the condition that at least one of the variables can only take on integer values[5].
- **Quadratic Programming** - This technique is widely used in operations research and optimization problems. It differs from LP since it optimizes for a quadratic optimization function[5].
- **Assignment** - The purpose of the assignment algorithm is to assign objects to the same number of jobs while minimizing the total cost. This problem is solved using the Mixed Integer programming algorithm[5].
- **Maximum Flow** - The Maximum Flow problem involves for a given network finding the highest feasible flow from the Source, a vertex with no incoming edges to Sink, a vertex with no outgoing edges. In this problem, the flow of a given edge represents a non-negative weight.NCSS uses LP to solve this problem[5].
- and others such as **Minimum Cost Capacitated Flow**,**Shortest Route**, **Transportation** and **Transhipment**.

To solve LP problems using the NCSS software users must define the variables in a spreadsheet as seen in figure 2.7a, and subsequently indicate to the solver which are the columns represent the variables, constraints, and logic as seen in 2.7b. When compared with excel NCSS uses a more structured approach seeing that all variables logic columns and constraints must be defined in specific columns and doesn´t use a point and drag method. The reports seen in 2.7c generated by the solver are similar to the ones generated by excel.

Despite improving on the Excel methodology, the NCSS methodology could be improved with a more interactive and user-friendly approach to the problem definition.

Pricing wise **NCSS** is paid costing for all users but possesses a free trial[6].This software is also closed source.

2.1.4 MATLAB

MATLAB or matrix laboratory is a software tool supporting a visual environment created for iterative analysis and design process and a programming language that expresses matrices and arrays directly as well as

[5]https://www.ncss.com/software/ncss/operations-research-in-ncss/#Technical
[6]https://www.ncss.com/online-store/

a live editor and the capability to output text, graphics, and formatted notebooks [7]. The main appeal behind MATLAB stands from it requiring lower configuration and setup compared to other programming environments, other factors distinguishing MATLAB from the other alternatives stand from its performance, state of the art algorithms and scalability, its ease of integration, and industry-focused features such as it's support for MODEL-based design and easy deployment with business systems. One of it's biggest setbacks stands from its pricing at time of writing it does not have any free tier, MATLAB is closed source as well[7].

This software has various uses for operations research some of them with of the shelf graphical capabilities, some examples of operations research tasks accomplished using MATLAB are:

- Aircraft Boarding Process Flow as sen in figure 2.8, this image illustrates some of the graphical modelling capabilities available to MATLAB users, it shows that users can model complex tasks using highly intuitive and expressive visual language.
- Optimization of Shared Resources in a Batch Production Process.
- Modeling a Kanban Production System.
- Modeling Machine Failure.
- Job Scheduling and Resource Estimation for a Manufacturing Plant.
- and many more.

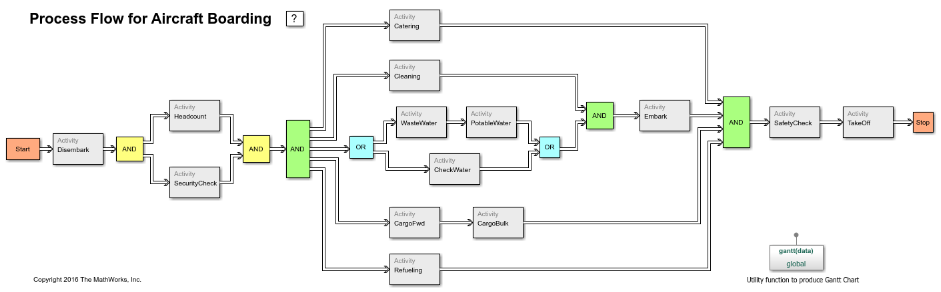


Figure 2.8: Aircraft boarding process flow using MATLAB modelling capabilities

Users can use MATLAB to model and solve LP problems using the *linprog* function.Users can use this function by expressing both variables and constants in a matrix form.

As an example the following inequalities constraint:

[7] https://www.mathworks.com/products/matlab.html

$$x(1) + x(3) \leq 2 \tag{2.1}$$

$$x(1) + x(2)/4 \leq 1 \tag{2.2}$$

$$x(1) - x(2) \leq 2 \tag{2.3}$$

$$-x(1)/4 - x(2) \leq 1 \tag{2.4}$$

$$-x(1) - x(2) \leq -1 \tag{2.5}$$

$$-x(1) + x(2) \leq 2 \tag{2.6}$$

Can be expressed in MATLAB in the following way:

```
A = [1 1
   1 1/4
   1 -1
   -1/4 -1
   -1 -1
   -1 1];

b = [2 1 2 1 -1 2];
```

After defining the inequalities constraints the next step is to define the objective function, for this example, the following function will be used:

$$-x(1) - x(2)/3 \tag{2.7}$$

Defined in MATLAB:

```
f = [-1 -1/3];
```

After defining the inequalities and objective function the next step is to call with $linprog$ function:

```
x = linprog(f,A,b)
```

We then get the optimal solution found by the solver:

```
x = 2 x 1
   0.6667
   1.3333
```

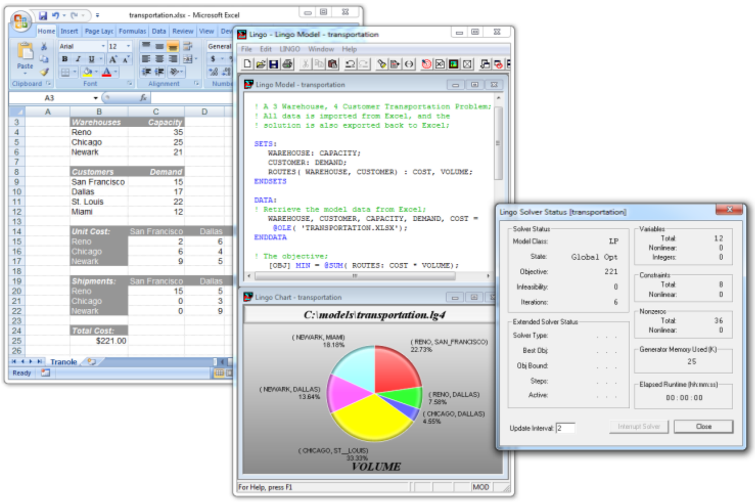
The $linprog$ function acceps equality constraints as an example the $x(1)+x(2)/4 = 1/2$ constraint can be added to our model:

```
Aeq = [1 1/4];
beq = 1/2;

x = linprog(f,A,b,Aeq,beq)
```

The *linprog* function can accept as well lower and upper bound constraints and different solver algorithms and parameters.MATLAB contains more advanced features such as modeling problems that can be formulated in a problem-based approach and more complete outputs.

2.1.5 LINGO and What'sBest!


transportation.xlsx - Microsoft Excel
Home
Insert
Page Layout
Formulas
Data
Review
View
Lingo - Lingo Model - transportation
File
Edit
LINGO
Window
Help
Lingo Model - transportation
Lingo Chart - transportation
C:\models\transportation.lg4
(NEWARK, MIAMI) 18.18%
(RENO, SAN_FRANCISCO) 22.73%
(NEWARK, DALLAS) 13.64%
(RENO, DALLAS) 7.58%
(CHICAGO, DALLAS) 4.55%
(CHICAGO, ST__LOUIS) 33.33%
VOLUME
For Help, press F1
MOD
Lingo Solver Status [transportation]
Interrupt Solver
Close
Tranole
Ready

	A	B	C	D
3		Warehouses	Capacity	
4		Reno	35	
5		Chicago	25	
6		Newark	21	
7				
8		Customers	Demand	
9		San Francisco	15	
10		Dallas	17	
11		St. Louis	22	
12		Miami	12	
13				
14		Unit Cost:	San Francisco	Dallas
15		Reno	2	6
16		Chicago	6	4
17		Newark	9	5
18				
19		Shipments:	San Francisco	Dallas
20		Reno	15	5
21		Chicago	0	3
22		Newark	0	9
23				
24		Total Cost:		
25		$221.00		
26				

```
! A 3 Warehouse, 4 Customer Transportation Problem;
! All data is imported from Excel, and the
! solution is also exported back to Excel;

SETS:
   WAREHOUSE: CAPACITY;
   CUSTOMER: DEMAND;
   ROUTES( WAREHOUSE, CUSTOMER) : COST, VOLUME;
ENDSETS

DATA:
! Retrieve the model data from Excel;
   WAREHOUSE, CUSTOMER, CAPACITY, DEMAND, COST =
    @OLE( 'TRANSPORTATION.XLSX');
ENDDATA

! The objective;
   [OBJ] MIN = @SUM( ROUTES: COST * VOLUME);
```

```
Solver Status                    Variables
  Model Class:        LP           Total:        12
        State:  Global Opt         Nonlinear:     0
    Objective:       221           Integers:      0
  Infeasibility:       0         Constraints
   Iterations:         6           Total:         8
                                   Nonlinear:     0
Extended Solver Status           Nonzeros
  Solver Type:       . . .         Total:        36
     Best Obj:       . . .         Nonlinear:     0
    Obj Bound:       . . .       Generator Memory Used (K)
        Steps:       . . .             25
       Active:       . . .       Elapsed Runtime (hh:mm:ss)
                                      00:00:00
Update Interval: 2
```

Figure 2.9: Lingo user interface

LINGO and What'sBest! are two tools created by the LINDO SYSTEMS INC to solve LP using data contained in spreadsheets, these tools constitute two different approaches to solve this problem. While LINGO uses the data in spreadsheets and uses an SQL like language to express the problem and call the solver, What'sBest! defines the problem inside an excel spreadsheet and calls the solver using a GUI.

What'sBest! offers users a form layout within spreadsheets where users can define variables, constraints, and optimization functions, to finalize the problem definition and call the solver. Users then use the GUI provided by the excel addon to define the different variables, constraints, and functions in the spreadsheet cells as well as choosing the optimization solver and different report options[8].

LINGO whose user interface can be seen in 2.9 is an optimization modeling software for linear, nonlinear, and integer programming. Lingo is a comprehensive tool designed to make solving various optimization problems simpler, faster, and more efficient. This tool provides an integrated package comprising a powerful language capable of expressing various optimization models, editing problems, and fast solvers. LINGO allows for reading data from spreadsheet files using its SQL like language[9].

2.1.6 Summary and comparisons

In this Section we summarized this chapter previous findings and compare the different solutions according to five different categories, this being:

[8]https://www.lindo.com/index.php/products/what-sbest-and-excel-optimization
[9]https://www.lindo.com/index.php/products/lingo-and-optimization-modeling

- **Cost:** We compare the different tools according to their pricing for both enterprise, education, and student users.
- **Code availability:** This metric compares the different tooling according to the availability of their code to the general public.
- **User interface:** In this Section we compare the user interfaces available for each of the different tools, the tasks end users can accomplish using them.
- **Ease of use:** This metric is used to compare the ease of use of the solutions.
- **Features:** We use this metric to compare the applications in terms of available features.

Excel	Paid with free trial and free student tier
SAS/OR	Paid with free trial and free student tier
NCSS	Paid with free trial and discounts for education and government use
MATLAB	Paid with heavy discounts for home, education and student users and free trial
Lingo and What'sBest!	Paid with free trial with education discounts

Table 2.1: Different solutions comparison cost

Looking at table 2.1 we can see that all of the tools above carry costs for enterprise users, however all of them have free trials, Excel and SAS/OR are free for students and all of the others are heavily discounted for students.

Excel	Closed source
SAS/OR	Closed source
NCSS	Closed source
MATLAB	Closed source
Lingo and What'sBest!	Closed source

Table 2.2: Different solutions comparison when it comes to code availability

All of the analyzed tools are closed source as seen in 2.2 .

Excel	Yes for LP problems and most tasks
SAS/OR	Possesses a a graphical user interface for some operations research tasks but LP can only be done using their algebraic language
NCSS	Yes similar to excel
MATLAB	Like SAS/OR has some GUI's but not for LP
Lingo and What'sBest!	Yes very similar to excel

Table 2.3: Different solutions comparison when it comes to User interface

Excel	Numerous features with many spreadshet related uses ,lacking in terms of operations research related features when compared with the other solutions,few LP solvers and options-
SAS/OR	Numerous optimization and operations research features
NCSS	Numerous features related to linear and mixed integer programming,various solvers and options.
MATLAB	Numerous operations research features and a a wide array of tools and a general purpose programming language.
Lingo and What'sBest!	Numerous linear and mixed integer tools.

Table 2.4: Different solutions comparison when it comes to fearures

Excel	Relativly easy to use for seasoned excel users,requires the instalation of aditional adons and uses a error prone methodology
SAS/OR	Some optimization tasks area accessible to end users but others such as LP require some knowledge of mathematical notations and some programming
NCSS	Process very similar to excel with a better and less prone to errors interface
MATLAB	Some tasks can be accomplished without programming but similarly to SAS/OR requires programming knowledge for others such as LP, however users with a mathematical background should be familiar with significant portions of the syntax.
Lingo and What'sBest!	The difficulty level of lingo sits between SAS/OR and excel and What'sBest! has a similar workflow to excel.

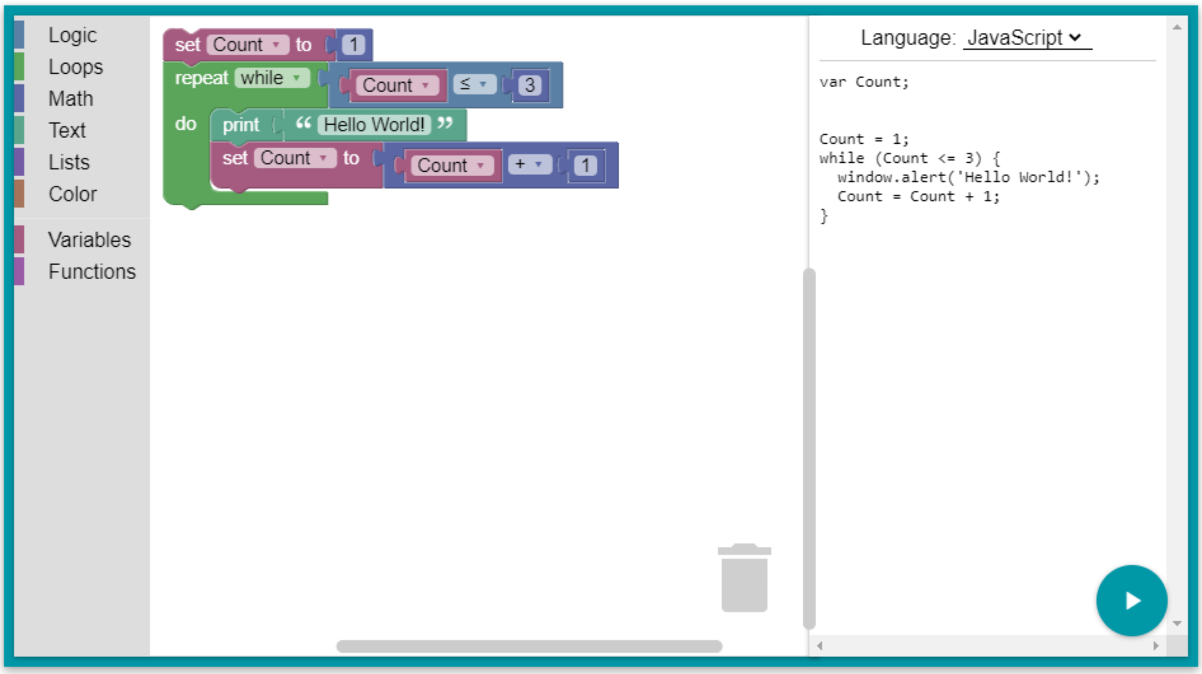
Table 2.5: Different solutions comparison when it comes to ease of use

When it comes to user interfaces users have different options according to their needs we can see in tables 2.3, 2.4 and 2.5 that tools such as Excel, NCSS, and What'sBest! would make a better choice for users with little to no programming experience wanting to solve LP problems. For more experienced users wanting more freedom and capabilities when solving LP problems tools such as MATLAB, Lingo and SAS/OR would be a preferable choice and for users wanting to solve more advanced optimization problems, SAS/OR or MATLAB would be the tooling of choice.

However, despite the array of features offered by the different solutions and the availability of user interfaces for select tasks there are still improvements that could be made in the usability, flexibility, and reliability of these interfaces. More notably various interfaces used for LP tasks such as the ones used in Excel, NCSS, and What'sBest! assume that users know the various steps involved in LP problem solving, the Excel tool has a problematic, *ad-hoc*, and prone to errors methodology involving a considerable amount of pointing, clicking, and dragging. Other tools such as NCSS and What'sBest! improve on the situation by having the references to the cells done manually however none of the previous tools have proactive measures to avoid user errors such as interactively showing the different constraints, variables, and optimization as the model is being constructed. Some of the more useful tools such as Excel lack some more advanced features such as more advanced solvers and options.

2.2 Frameworks and notable Visual Languages projects

2.2.1 Blockly



```
var Count;


Count = 1;
while (Count <= 3) {
  window.alert('Hello World!');
  Count = Count + 1;
}
```

Figure 2.10: A simple Blockly program.

Blockly is a google created library created to build visual applications that output syntactically correct code. This library is written in pure JavaScript, is 100% client-side and without server-side dependencies, can be used with all major browsers, and is highly customizable and extendable[10].

Various projects have been built using this technology,having found success particularly in the educational space, this applications include Blockly Games[11],code.org[12], MIT App Inventor[13] and BlockPy[14].

The creation of this library was greatly influential to the field of visual programming languages not only due to the projects that were build using it but as well as the insights gained into the difficulties and mistakes made when creating visual programming languages some of the mistake noticed by the google team where:

- The google team noticed users found difficulties distinguishing between conditional and loop blocks due to their similar styling. To mitigate this problem the google team moved the conditional block to the logic block.
- Adding borders to blocks is fundamental for their distinction [9].

[10]https://developers.google.com/blockly
[11]https://blockly.games/
[12]https://code.org/
[13]https://appinventor.mit.edu/
[14]https://think.cs.vt.edu/blockpy/

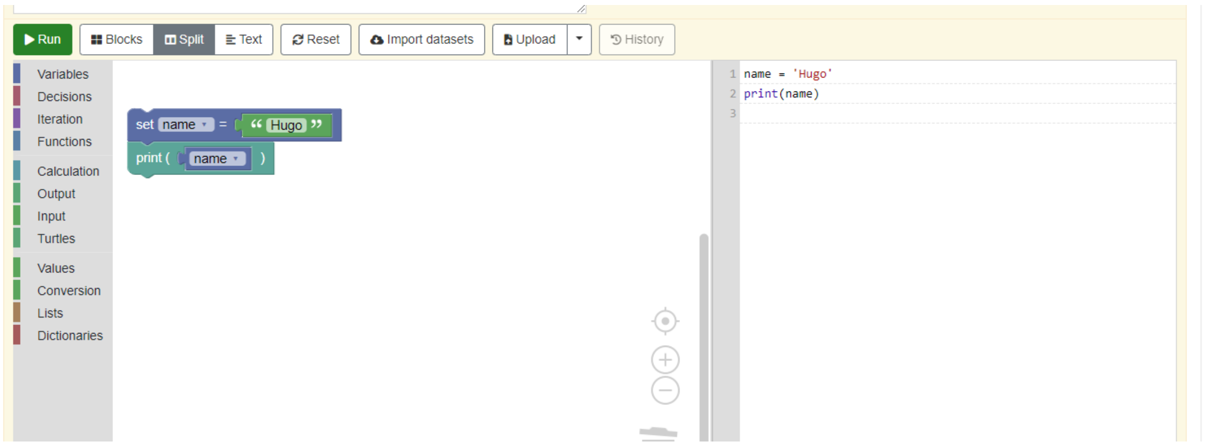
- Avoiding syntax errors is one of the key features of visual programming languages, in earlier Blockly implementations users could forget to connected blocks due to them being close together, this bug is sometimes unnoticeable to the human eye. To solve this kind of problem in successive implementations by connecting unattached connector pairs reasonably close to each other [9].

- Non-intuitive syntax, often the syntax of blocks didn't convey its meaning and utility, an example of this was that some blocks that could be placed inside others don't have a shape that conveys so. Another example of syntactic confusion was the use of a straight arrow and turn arrows to symbolize a character turning right, this proved itself confusing for some users and the problem was mitigated by using a circular arrow.

- Users not reading instructions was also a problem when dealing with students the google team noticed that independently of font-size, color, and placement users continuously closed the pop-ups without reading, to try to solve this problem the google team implemented pop-ups that would close only when the task is completed [9].

Understanding some of the problems found when bringing some of the Blockly projects to end-users will be of use for any project related to the creation, study, or implementation of visual languages.

2.2.2 BlockPy



```python
name = 'Hugo'
print(name)
```

Figure 2.11: Hello World program using the BlockPy interface

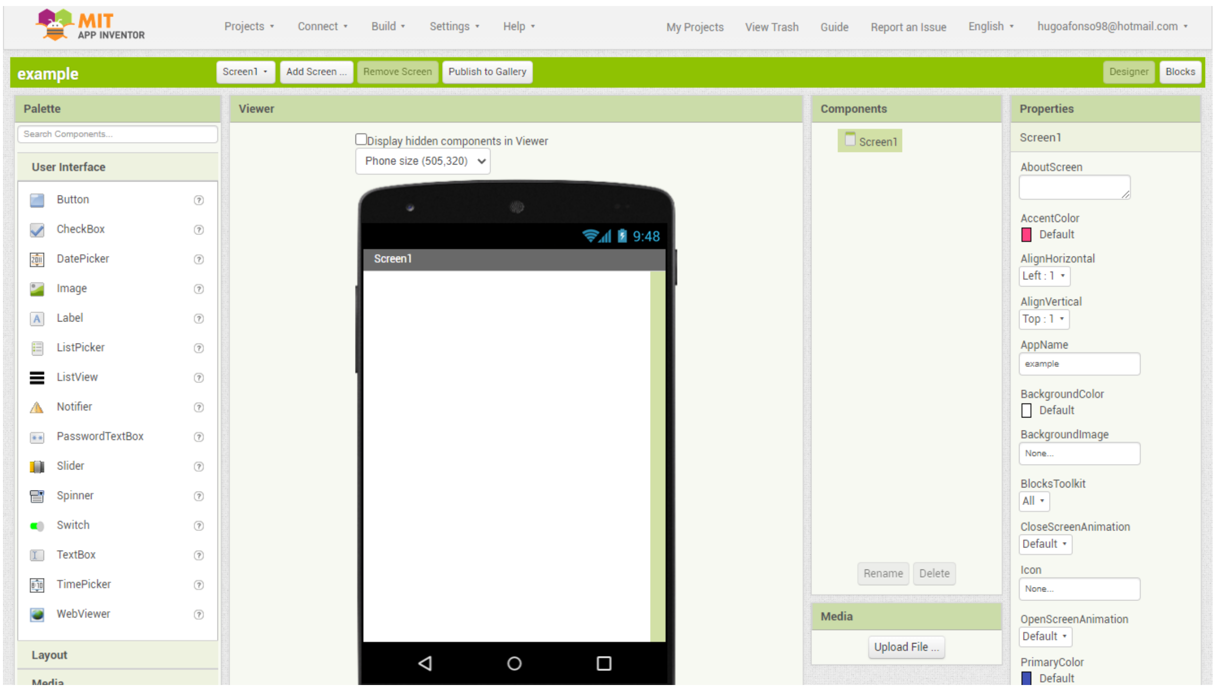
The BlockPy project is a web-based, open-access Python programming environment made for introductory programming and data science education. This project came by due to the increasing need that professionals and non computer science students feel to learn computer science related skills and due to the contextualization of introductory computer science education focusing mostly on game design and media computation thus alienating potential learners due to a perceived pointlessness to their professional activity. Since data science skills such as data processing are widely needed across a variety of fields [3].

The choice of python for this educational tool because of it's explicit syntax, strong support for data science libraries such as pandas, Matplotlib, and scikit-learn, despite its ease of use when compared with other computer languages using an intermediate visual language that allowed for a view of the entire user interface. This was found to improve the usability of the language for novice programmers. One feature that distinguishes BlockPy from other projects allowing users to program in Python using visual languages is the possibility of freely oping between text and visual programming, this feature was found to improve the transition process between the two types of languages [2].

Accessibility is one of BlockPy's project goal for this reason BlockPy is an easily accessible web-based platform, all of its code is open source and leverages an array of open-source libraries. To avoid excessive broadband use BlockPy the python code is run locally, to do so they used a modified instance of the Skulpt JavaScript library capable of parsing and compiling python code to JavaScipt.The visual language portion of this project was done using the Blockly library [3].

Beyond the previously said this project made other contributions to the visual programming field such as adaptative guidance this adds to the user experience by providing more adapted feedback

2.2.3 MIT app Inventor



AboutScreen

AccentColor: Default

AlignHorizontal: Left : 1

AlignVertical: Top : 1

AppName: example

BackgroundColor: Default

BackgroundImage: None...

BlocksToolkit: All

CloseScreenAnimation: Default

Icon: None...

OpenScreenAnimation: Default

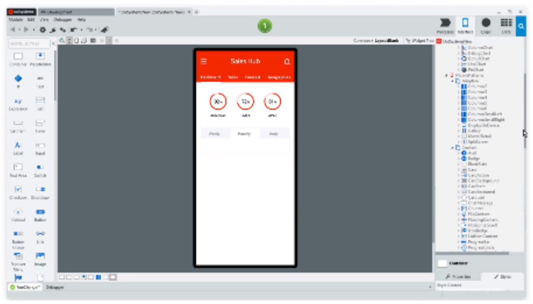
PrimaryColor: Default

Figure 2.12: Mit app Inventor user interface.

The MIT App Inventor whose user interface can be seen in figure 2.12 is an educational platform that uses android app development to teach introductory computer science concepts. This concept was born from the observation that smartphones play an intrinsic role in our everyday lives however the majority don't understand how the technology they use works. This project tries to solve this problem by providing a more user-friendly approach to building smartphone apps. The interface uses a drag and drop approach for different components and the Blockly library for the program logic [21].

This project's contributions to the user interface and visual programming fields stem from the real-world context and applications instead of more traditional programming aspects such as loops, arrays, operators, and conditionals [21].

2.2.4 Outsystems


(a) Outsystems platform GUI creation environment

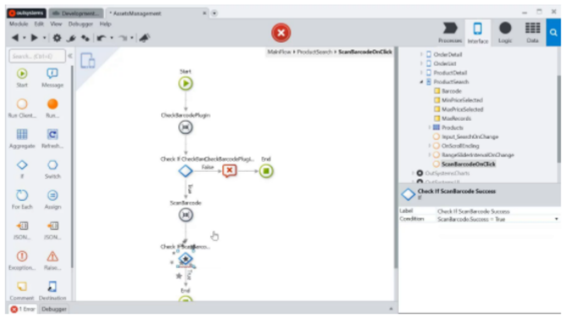

(b) Outsystems platform logic visual language

Figure 2.13: Outsystems platform visual languages

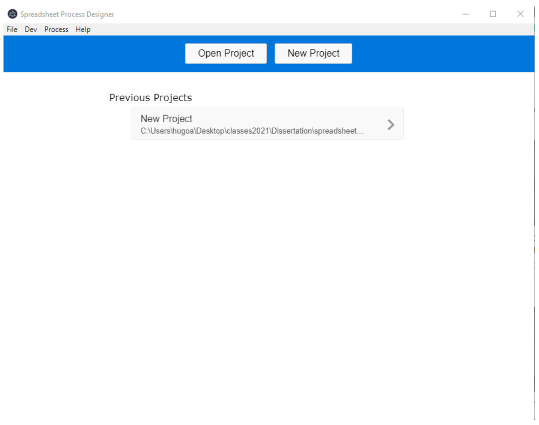
Outsystems was funded in 2001 in Lisbon, Portugal, and is considered one of the few Portuguese and has current headquarters in Boston, Massachusetts with offices in 11 countries as of 2019. Outsystems is a leader when it comes is a low-code platform in the enterprise market being used by companies in fields such as banking, healthcare, insurance, transportation, and energy[15].

The Outsystems platform allows users to develop web applications using an off the shelf visual programming language, this platform diverges from other no-code platforms in its usefulness due to its increased potential and wider reach associated with the possibility of using a hybrid visual and textual development environments in the same project. This approach allows then for faster development by accelerating the critical phases of software development using their drag and drop approach for creating GUI seen in figure 2.13a and flux-based language for some of the logic as seen in figure 2.13b, while preserving flexibility and features associated with traditional text-based software development by allowing its users to complement the visual language[16].

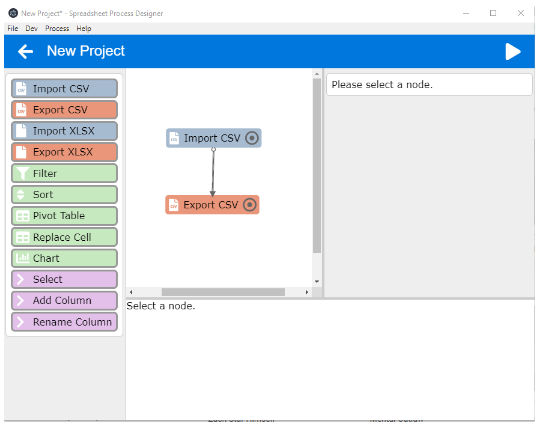
This platform comprises numerous advanced features such as a modern, collaborative ide , integration with more than 275 enterprise systems such as MySql, Google Drive, and SAP, numerous templates and capabilities to create beautiful user interfaces, fast deployment allowing users to deliver reactive web apps both as apps with automatic packaging and as Progressive Web Apps(PWA). The use of this platform allows as well for building enterprise-grade applications, avoiding technical debt due to constant monitorization and enforcement of best practices, automatic security and continuous monitoring, and cost control with the use of real-time data[16].

[15]`https://www.outsystems.com/case-studies/`
[16] `https://www.outsystems.com/`

2.2.5 Visual language for spreadsheet processes


Spreadsheet Process Designer
File Dev Process Help
Open Project
New Project
Previous Projects
New Project
C:\Users\hugoa\Desktop\classes2021\Dissertation\spreadsheet...

(a) Load previous process or create new one



(b) Edit process

Figure 2.14: Edition,creation and loading or spreadsheet processes.

This spreadsheet process creation software is a tool that allows users to define Excel processes using a flow-based language,it came to be from the need to improve the methodology used in the spreadsheet creation process. The methodology behind this tool uses a flux-baxed,seen in figure 2.14b language to express some of the different steps involved in the spreadsheet creation process [18] . It uses Electron, React, Redux, and other web technologies for the front-end and back-end and some back-end tasks such as opening and reading files. The interaction with Excel is done using the Microsoft interop Excel API [17], this API allows for the reading and creation of Excel workbooks suing many of its features such as Graphs, formulas, and operations such as sorting and filtering data.

This tool uses a flow-based language made by different cells each one representing an Excel operation at the moment the tool implements the following operations:

- **Import CSV/Import XLSX -** Loads data from CSV and XLSX files.
- **Export CSV/Export XLSX -** Writes data to CSV adn XLSX files.
- **Filter -** Allows for the filtering of up to three columns.
- **Sort -** Allows for the sorting in ascending or descending order of up to three columns.
- **Pivot table -** Pivots a table.
- **Chart -** Allows for the creation of both bar and pie charts.
- **Select -** Selects a column.
- **Add column -** Creates new column.

[17] `https://docs.microsoft.com/en-us/dotnet/api/microsoft.office.interop.excel.application?view=excel-pia`

- **Rename column -** Renames column.

Currently, it allows for the creation, saving, and loading of processes saved in Json files as seen in figure 2.14a, with some basic features such as loading data from and saving to both CSV and xlsx files, use some Excel features such as Filter, Sort, Pivot table, and Charts. This tool represents excel processes graphically using a flux-based language, to create a process users can drag and drop the wanted processes.

This tool used in an industrial case study to explore the usefulness of visual programming languages as a replacement for standard spreadsheet software in some of its most commonly used tasks, this study was done by the quality department at Bosch Car Multimedia Portugal, this department uses spreadsheets to analyze data exported from an SAP ERP system [17]. These spreadsheets contain data reporting defects claims of the companies produced parts and it is repeated once a week. With the use of this tool, their spreadsheet construction process was made explicit and allowed for greater quality assurance and other benefits such as improved employee productivity [17].

2.3 Projects involving visual languages and LP

2.3.1 A graphics interface for linear programming

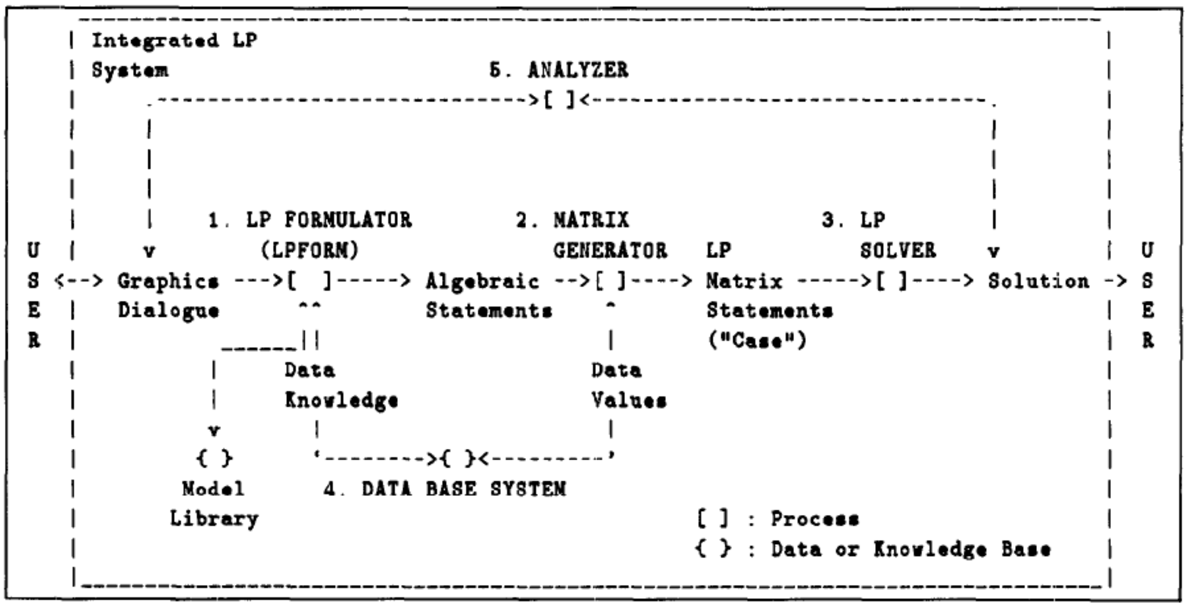


Figure 2.15: Integrated LP System Diagram proposed by the paper

This paper [14] introduces an interface for a software system that guides users when graphically building LP models instead of using a mathematical formulation. Their interface boasts multiple features such as hierarchical decomposition, multiple model representations, alternative formulation approaches, the use of model templates, and database and model management facilities.

The authors of the paper identified the following steps to solve a LP problem, investigation, model formulation, data management, algorithmic solution and report generation and analysis, these steps can

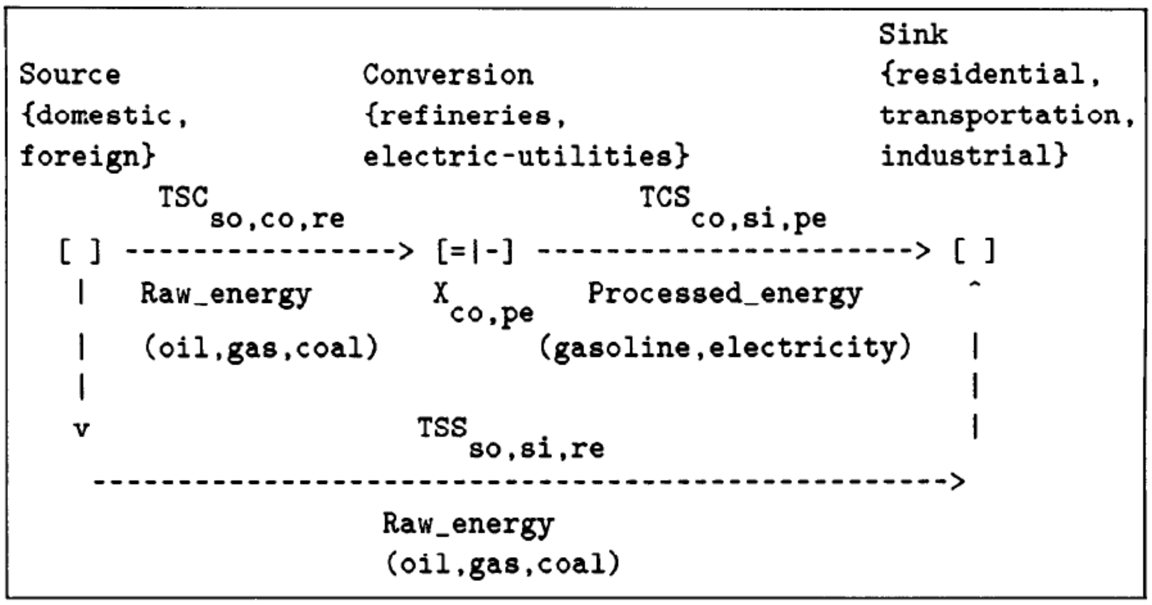


Figure 2.16: Energy problem represented in LPFORM

be seen in the diagram proposed by the authors seen in Figure 2.15. In article [14] only the first four stages were discussed. The first step, problem investigation is due to the nature of the task done manually. In their solution LPFORM the second step being the model formulation in an algebraic language is automated. The third step involves human input, this being necessary to input, store, and retrieve data from a relational database. The fourth step, solving the model is done automatically.

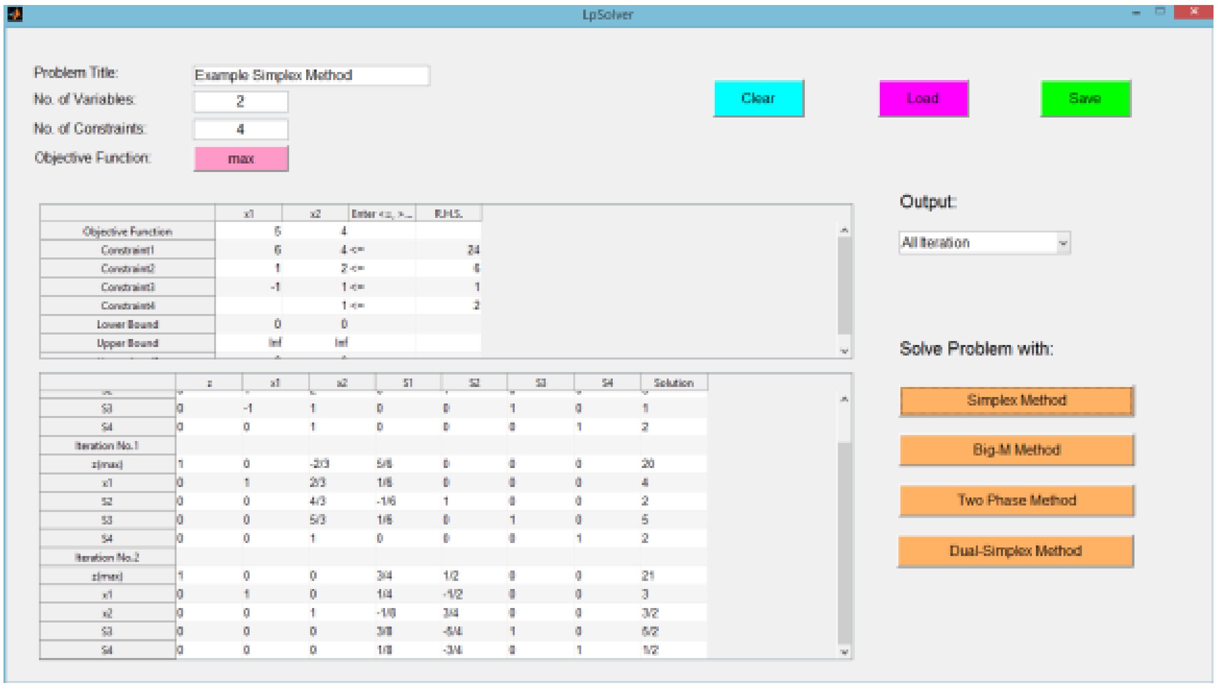
2.3.2 Creating a GUI Solver for Linear Programming Models in MATLAB

In this paper [7], the authors introduce LpSolver a Graphical User Interface (GUI) for LP problems using MATLAB. Their solution was created for classroom-sized problems and boasts features such as computing expressions with symbolic variables and fractions and allowing the users to trace the optimization process.

When building models using the LpSolver users have access to a graphical interface seen in Figures 2.18 where they can input the data manually or load the data from a previously created file, however, users are discouraged to use LpSolver to solve problems with more than 50 constraints or 100 variables. Users can also save the created model and have access to different algorithms for solving the problem, this includes the Simplex Method, the Big-M Method, the Two-Phase Method, and the Dual-Simplex Method.

2.3.3 gLPS: A graphical tool for the definition and manipulation of linear problems

In this paper [8], the authors introduce, gLPS (graphical Linear Programming System), this tool allows users to express linear problems using graphical objects (circles for restrictions, squares for variables, etc.) networked according to specific rules to form a model. The major strength of gLPS is grounded on

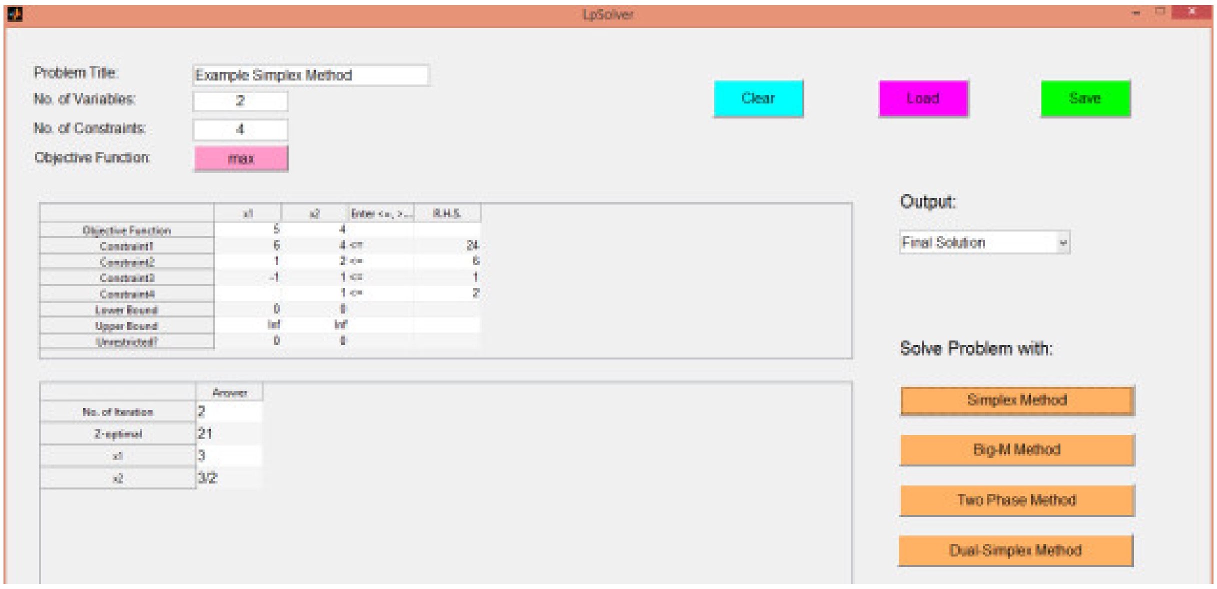


Problem Title: Example Simplex Method
No. of Variables: 2
No. of Constraints: 4
Objective Function: max
Output: All iteration

	x1	x2	Enter <=, >...	R.H.S.
Objective Function	5	4		
Constraint1	6	4	<=	24
Constraint2	1	2	<=	6
Constraint3	-1	1	<=	1
Constraint4		1	<=	2
Lower Bound	0	0		
Upper Bound	Inf	Inf		

	z	x1	x2	S1	S2	S3	S4	Solution
S3	0	-1	1	0	0	1	0	1
S4	0	0	1	0	0	0	1	2
Iteration No.1								
z(max)	1	0	-2/3	5/6	0	0	0	20
x1	0	1	2/3	1/6	0	0	0	4
S2	0	0	4/3	-1/6	1	0	0	2
S3	0	0	5/3	1/6	0	1	0	5
S4	0	0	1	0	0	0	1	2
Iteration No.2								
z(max)	1	0	0	3/4	1/2	0	0	21
x1	0	1	0	1/4	-1/2	0	0	3
x2	0	0	1	-1/8	3/4	0	0	3/2
S3	0	0	0	3/8	-5/4	1	0	5/2
S4	0	0	0	1/8	-3/4	0	1	1/2

Figure 2.17: LpSolver graphical interface create formulation with output all iterations



Problem Title: Example Simplex Method
No. of Variables: 2
No. of Constraints: 4
Objective Function: max
Output: Final Solution

	x1	x2	Enter <=, >...	R.H.S.
Objective Function	5	4		
Constraint1	6	4	<=	24
Constraint2	1	2	<=	6
Constraint3	-1	1	<=	1
Constraint4		1	<=	2
Lower Bound	0	0		
Upper Bound	Inf	Inf		
Unrestricted?	0	0		

	Answer
No. of Iteration	2
Z-optimal	21
x1	3
x2	3/2

Figure 2.18: LpSolver graphical interface with final solution output

it being able to express LP models belonging to a wide array of domains, gLPS is not only a modeling language but also an integrated software system for the creation, modification, and running of LP models.

Contrary to other projects presented in this Subsection gLPS was created for experts capable of formulating LP models algebraically and the author's name LPForm(Subsection 2.3.1) as being one language focused on improving LP for nonexperts. The symbolism used by gLPS is a direct translation of the algebraic notation, a choice whose authors assert restricts the potential users of gLPS. The authors justify this choice by claiming that this notation would be a more natural approach to OR specialists. When creating a restriction using gLPS users are presented with an interface seen in Figure 2.20, using this interface users

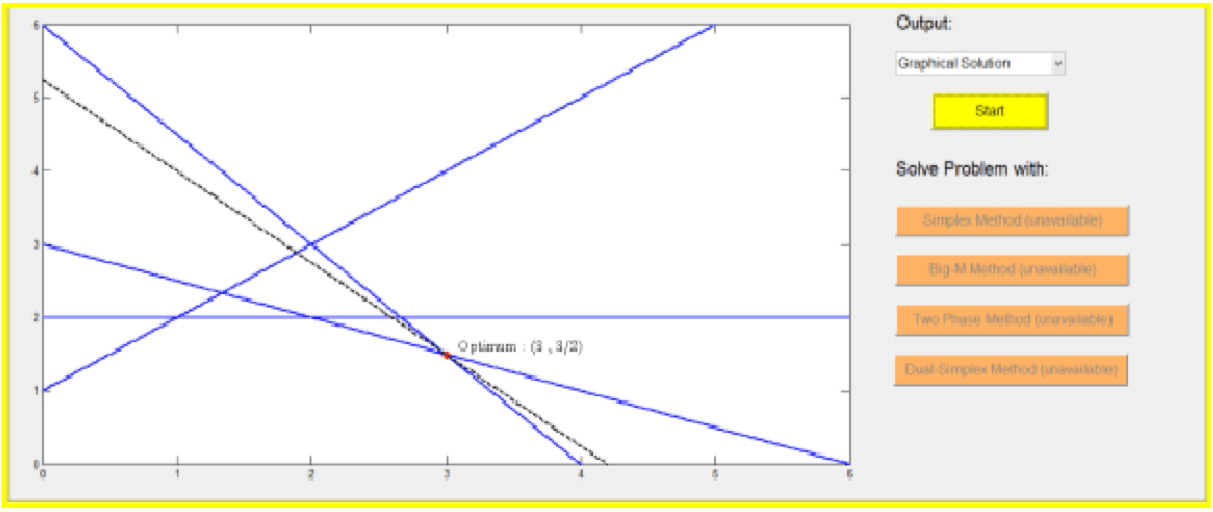

Optimum : (3 , 3/2)
Output:
Graphical Solution
Start
Solve Problem with:
Simplex Method (unavailable)
Big-M Method (unavailable)
Two Phase Method (unavailable)
Dual-Simplex Method (unavailable)

Figure 2.19: LpSolver graphical solution

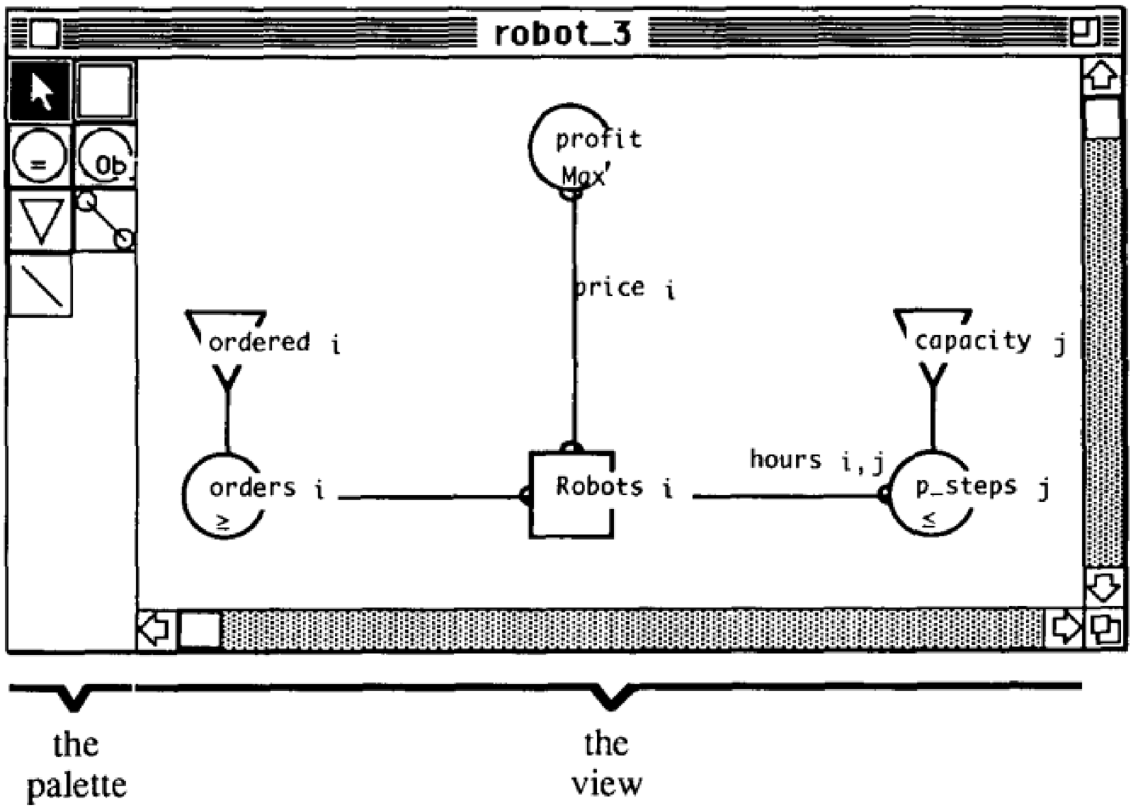


Figure 2.20: gLPS graphical interface

can drag, drop, and connect different components to create a LP model, in this interface the squares designate the variables, the circles the represent either an inequation or an objective and the triangles represent the data.

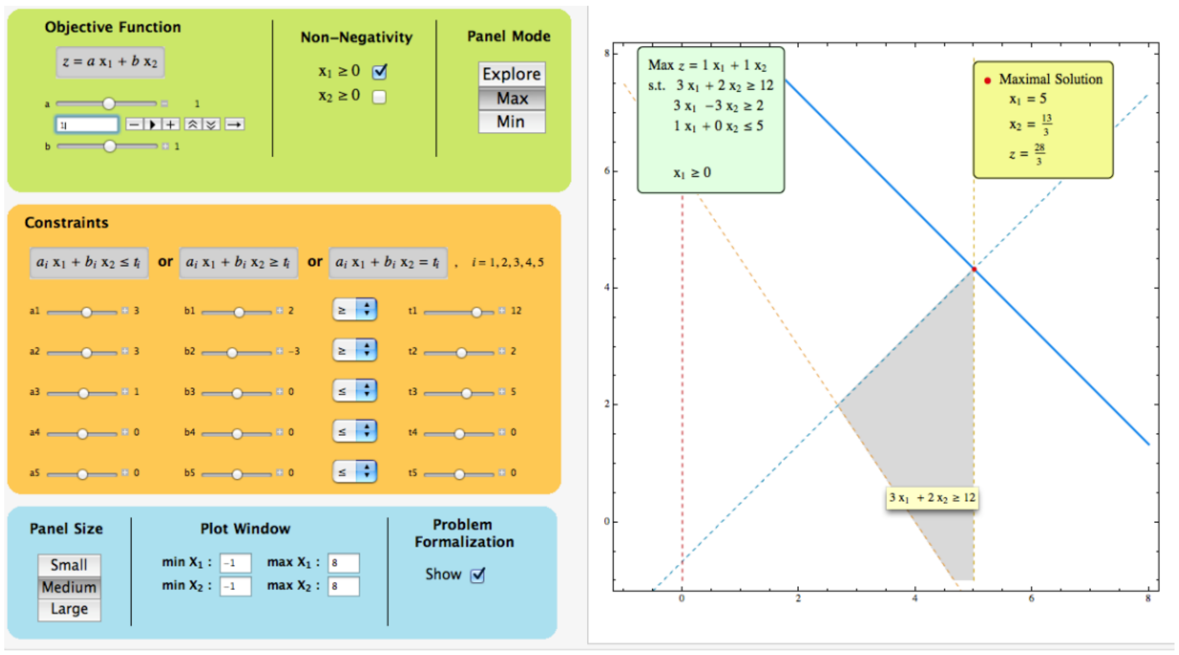


Figure 2.21: GUI LP interface for MATLAB

2.3.4 Two-variable Linear Programming: A Graphical Tool with Mathematica

In this paper [22], the authors introduce GLP-Tool, a MATEMATICA graphical interface designed to help users understand fundamental concepts of LP. The author's goal is to leverage active learning to increase student engagement during the learning process. This tool is a dynamic, interactive, and visual tool that allows solving user-defined LP problems with two variables. In particular, the user can explore different objective functions and constraint sets, obtain graphical and numerical information on optimal solutions and intuitively perform post-optimal and sensitivity analysis. The author's focus is clearly on education and teaching the mathematical component behind LP and the author justifies some of their choices on classroom requirements.

The interface shown in Figure lets the users input the objective and constraints by using selection and sliding inputs, it allows for the definition of nonnegativity by choosing the option in the variable, choosing the values of the constants using a sliding option as well as the variables in the objective and constraints, user can also choose to maximize or minimize the objective. Users can then visualize the feasible area and solution.

2.3.5 Conclusions

Overall we found that the projects referenced in this Section either have a mathematical approach to LP and its methods and do not offer business-focused features such as Excel by allowing users to work with data to create the formulations such as in Excel, this is seen in Subsections 2.21 and 2.3.2. Or despite having a similar focus to us possess outdated and not very intuitive interfaces, this is seen in Subsections

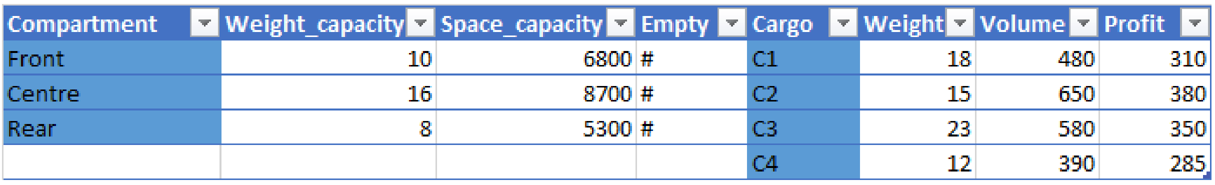
2.3.1 and 2.3.3

Taking into account we believe that it is possible to leverage modern advancements in visual languages and web technologies to create a tool that offers a better approach from an applied perspective than the ones referred to in this Section.

Chapter 3

A block-based language for linear programming

Here, we introduce our proposed language, We will go over how LPBlocks processes the data, which blocks were designed, and how to define the variables, constraints, and LP objective. Additionally, to make it easier for the reader to accompany the language constructs and format, we will be using a running example featured in a Master of Business Administration (MBA) exam [4]. This example problem aims to increase the profit of deliveries by airplanes. The problem statement provides values for the weight and space capacity of three different airplane's 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 3.1.



Compartment	Weight_capacity	Space_capacity	Empty	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

Figure 3.1: Input data for the running example problem

3.1 Data parsing

Our language requires the input data to follow a specific structure. This structure allows for the definition of **index columns** (as seen highlighted in blue in Figure 3.1), this are used to reference values and iterate over the **data columns** (in white the same figure) this being always associated with one index column. 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 **index column**, and that different sets of **index** and **data columns** are separated by an empty column as can be seen in the figure (fourth column). In this case there are two sets, the first being for the three plane compartments and the second for the four types of cargo.

3.2 Blocks

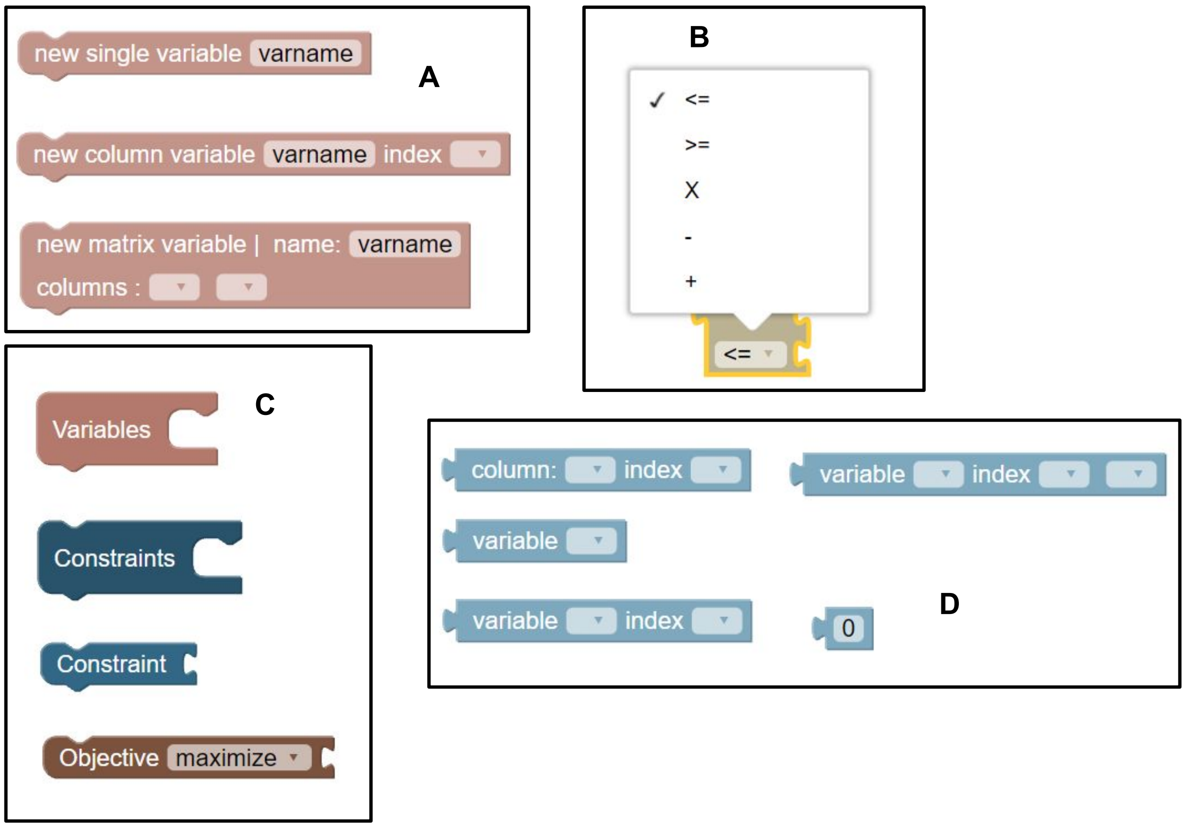


Figure 3.2: Building blocks for LPBlocks

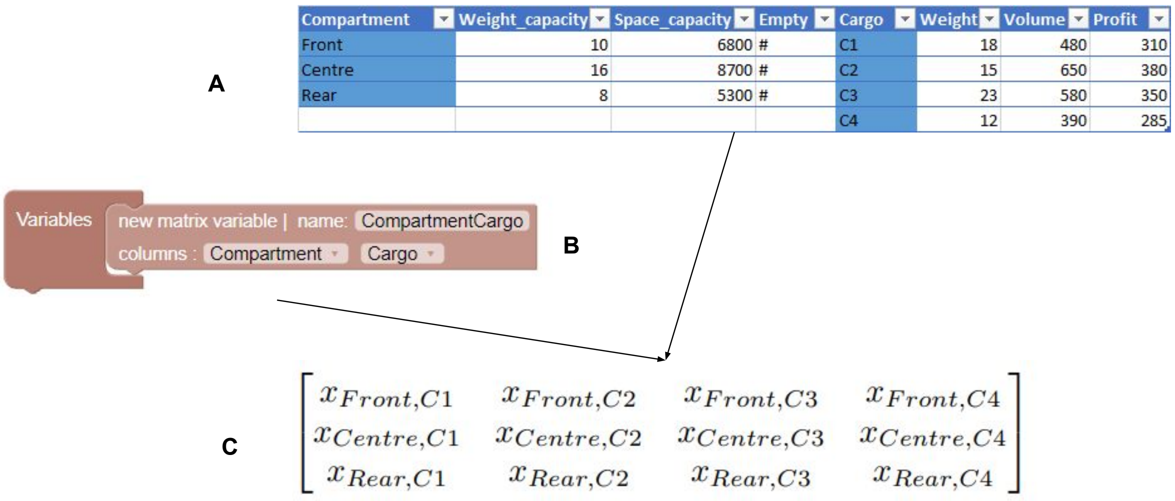
The building blocks of the linear programming language we propose can be seen in Figure 3.2. LP-Blocks includes:

- **Variable blocks** (seen in Figure 3.2.A): Blocks for creating single, column and matrix variables.
- **Operation block** (seen in Figure 3.2.B): A block to construct an individual constraint.
- **Building blocks** (seen in Figure 3.2.C): These blocks include two nesting blocks for the `Variables` and `Constraints` a nesting block to add an individual `Constraint` to a `Constraints` block and an `Objective` block to define the objective function.
- **Value blocks** (seen in Figure 3.2.D): A set of blocks to access the variables created before, that is, `values blocks`.

To further facilitate this process for novice and inexperienced users, when building a new linear programming model, the `variables`, `constraints` and `objective` blocks already appear and are connected in the workplace when creating a fresh solution. In the next sections we detail each of the blocks.

3.3 Defining variables

To define a mathematical linear programming model, considering our running example one would start by creating a set of variables iterating over the the airplane sections and the cargoes as shown in Figure 3.3.C (we refer to the problem's original website for a more common variable naming). Since this is a very common scenario, LPBlocks includes a construct that can be used to define all these variables which we call a matrix. In Figure 3.3.B we use such a construct to create the variables for the running example. In the example, we use a `matrix variable block` to create a new $N \times M$ matrix variable named `CompartmentCargo`, with N being equal to the length of the column `Compartment` and M to the length of the column `Cargo` with these columns serving as its indexes.



Compartment	Weight_capacity	Space_capacity	Empty	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

Variables | new matrix variable | name: CompartmentCargo
columns : Compartment Cargo

$$\begin{bmatrix} x_{Front,C1} & x_{Front,C2} & x_{Front,C3} & x_{Front,C4} \\ x_{Centre,C1} & x_{Centre,C2} & x_{Centre,C3} & x_{Centre,C4} \\ x_{Rear,C1} & x_{Rear,C2} & x_{Rear,C3} & x_{Rear,C4} \end{bmatrix}$$

Figure 3.3: Create variables

LPBlocks offers several options to define new variables, using the blocks seen in Figure 3.2.A respectively:

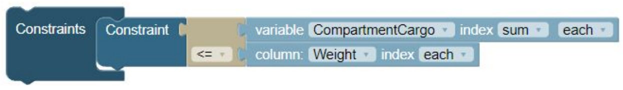
- single variables through its name;
- column variables defining its name and an index column for which the variable will be iterated and accessed;
- matrix variables that take a name and two index columns for which they can be iterated and those values accessed (used in Figure 3.3.B).

The process of generating the model variables is dependent on which `variable blocks` we used:

- For the `single variable block`, a variable is generated with the chosen name.
- For column variable blocks, an array of variables is created.
- For `matrix variables blocks`, a matrix of variables is created(as shown in Figure 3.3.B).

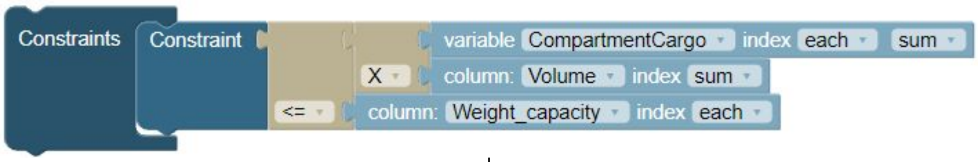
3.4 Defining constraints

The second step to define the mathematical model would be to create a set of constraints, using the variables created before, and encoding the restrictions of the underlying problem. A constraint of the running example is that one “cannot pack more of each of the four cargoes than their available quantity”. The mathematical encoding would be as shown in Figure 3.4. There are four constraints, one for each cargo. For each constraint, the left hand side of the inequality displays the sum of variables referring to the corresponding cargo (e.g. C1 for the first constraint) and for the three different airplane sections. On the right-hand side one would write the cargo weight limit.


Constraints
Constraint
variable CompartmentCargo index sum each
<=
column: Weight index each

$$\begin{aligned}
x_{Front,C1} + x_{Centre,C1} + x_{Rear,C1} &<= 18\\
x_{Front,C2} + x_{Centre,C2} + x_{Rear,C2} &<= 15\\
x_{Front,C3} + x_{Centre,C3} + x_{Rear,C3} &<= 23\\
x_{Front,C4} + x_{Centre,C4} + x_{Rear,C4} &<= 12
\end{aligned}$$

Figure 3.4: Defining constraints for the cargoes weight


Constraints
Constraint
variable CompartmentCargo index each sum
X
column: Volume index sum
<=
column: Weight_capacity index each

$$\begin{aligned}
480x_{Front,C1} + 650x_{Front,C2} + 580x_{Front,C3} + 390x_{Front,C4} &<= 6800\\
480x_{Centre,C1} + 650x_{Centre,C2} + 580x_{Centre,C3} + 390x_{Centre,C4} &<= 8700\\
480x_{Rear,C1} + 650x_{Rear,C2} + 580x_{Rear,C3} + 390x_{Rear,C4} &<= 5300
\end{aligned}$$

Figure 3.5: Defining constraints - second example

In LPBlocks, each constraint is defined by dragging a constraint block inside the constraints block (second and third blocks from the top in Figure 3.2.C) and then using the value blocks (blocks in Figure 3.2.D) and operation blocks (blocks following the constraint block in Figure 3.2.B) to express the constraints. In our language, operation blocks represent relations between blocks and are used to express several operations including arithmetic operations and inequalities. The value blocks can represent:

- Columns;
- Previously defined variables;

- Numbers.

The variables can be accessed using different blocks and options. This as well as the way the columns are used 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 3.4 to generate multiple constraints or use the three slot variable blocks to access a particular value of the given variable.

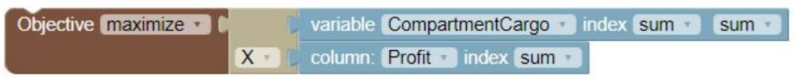
Our solution possesses other features such as using the positioning and index columns of the variables and columns used in the constraint construction automatically deducing summations, sumproducts and sets of linear programming constraints from the high level user defined visual constraints.

The first constraint in Figure 3.4.A is defined in our language by using: *i)* an `operation block` with the inequality sign <=; *ii)* a `variable block` with the option `CompartmentCargo` and; *iii)* 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 3.4.B.

Another example constraint can be expressed in natural language as "the volume (space) capacity of each compartment must be respected". This constraint (in Figure 3.5.A) uses `X` (multiplication) and <= `operation blocks` and `value blocks` to express the more complex constraints. This constraint differs from the previous ones since the use of the `X operation block` leads to the generation of sumproduct constraints instead of sum. For this constraint, our compiler generates the linear programming constraints featured in Figure 3.5.B.

3.5 Defining the objective functions

The final step in a linear programming model is the definition of an objective function. For our running example, one intends to maximize the profit of the airplane usage.



$$310[x_{Front,C1} + x_{Centre,C1} + x_{Rear,C1}] + 380[x_{Front,C2} + x_{Centre,C2} + x_{Rear,C2}] +$$

$$350[x_{Front,C3} + x_{Centre,C3} + x_{Rear,C3}] + 285[x_{Front,C4} + x_{Centre,C4} + x_{Rear,C4}]$$

Figure 3.6: Objective function for the running example

To define the objective function users must connect the `objective block` and the `constraints block` together and use several `value` and `operation blocks`.

In the example seen in Figure 3.6.A 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 3.6 which would be the one written in a mathematical model.

Chapter 4

Implementation and architecture

4.1 Components and architecture

Our implementation possesses three main components these components are the Web application, used to provide the user access to our application and with an interface to select the data, build and run the models. Our application includes a *Rest API* to read spreadsheet files and a *Rest API* to run the model and get a solution. A component diagram with the architecture of our implementation can be seen in Figure4.1.

4.2 Web application

The *Web application* was built using various web technologies such as *React*, *Redux*, *node-blockly*[1] and others, our goal was to create a highly dynamic and interactive way for users to build their models and to do so we believed using the previously referred libraries would be conducive to our goal.

Because of performance issues and to guarantee near-instantaneous feedback for the user when creating the models in the block-based language we tried to do most of the computing in the front-end since this allows for lower communication costs, this was possible to do for the mathematical formulation for the model built by the user since the code generation component of our LPBlocks *Blockly* implementation was built by us. The running of the model by a solver and the reading of data from the spreadsheets is done on the backend, this was done do to better existing libraries in *Node.js* when compared to libraries that run in the browser for the preceding purposes. The communication between the *Web Application* and the services are done using the *Rest* protocol.

[1] https://www.npmjs.com/package/node-blockly

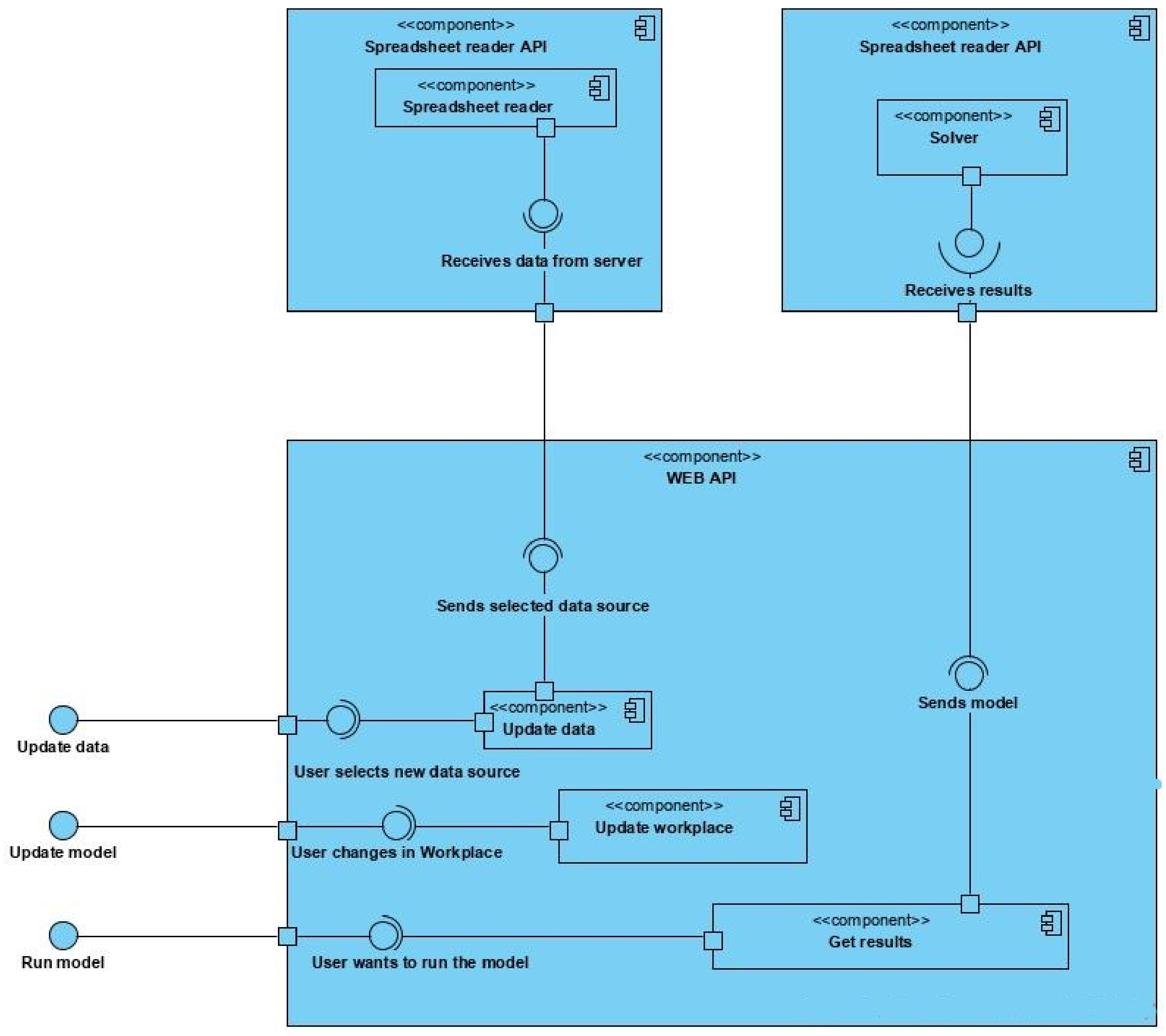


Figure 4.1: Component diagram for our application

4.2.1 Compilation process

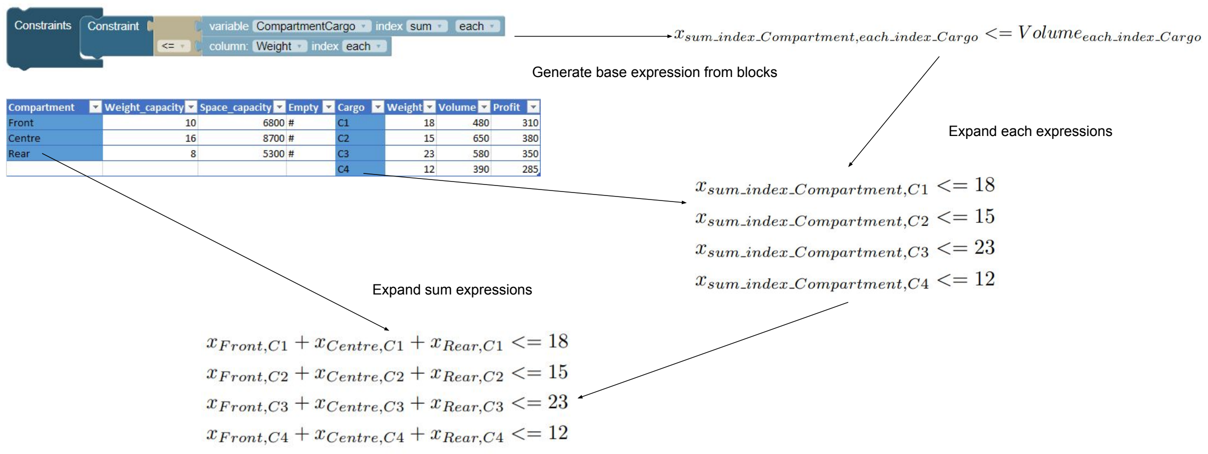
In this Subsection, we elaborate on the different aspects of the compilation process used to generate the LP mathematical formulations from our language. In Subsubsection 4.2.1.1 we explain how the variables are generated from a block statement and the data coming from a spreadsheet we then in Subsubsection 4.2.1.2 explain how we generate the constraints and the objective.

4.2.1.1 Generating variables

To generate the variables into our internal representation when reading a `variable block`, our compilation process consists in reading the name of the given variable and depending on the blocks used we either: *i)* stop there for a `single variable block`, (*ii)*) read the variables index columns. The compiler then sets the variable name as a key to a list with the index columns used to create the variables as its value. The individual variables created with various iterations are computed when necessary.

The process of computing the variables when given their name consists in retrieving the index columns array for the given variable, and calculating all possible permutations for the contents of the columns. This process can be visualize in Figure 3.3, in this Figure the users uses a `Matrix Variable block` to define the variable and our compiler then iterates over both the columns passed as the input to generate the variables.

4.2.1.2 Generating the constraints and objective


Constraints
Constraint
variable CompartmentCargo index sum each
<=
column: Weight index each
Generate base expression from blocks
Expand each expressions
Expand sum expressions

Compartment	Weight_capacity	Space_capacity	Empty	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

$$x_{sum\_index\_Compartment,each\_index\_Cargo} <= Volume_{each\_index\_Cargo}$$

$$x_{sum\_index\_Compartment,C1} <= 18$$
$$x_{sum\_index\_Compartment,C2} <= 15$$
$$x_{sum\_index\_Compartment,C3} <= 23$$
$$x_{sum\_index\_Compartment,C4} <= 12$$

$$x_{Front,C1} + x_{Centre,C1} + x_{Rear,C1} <= 18$$
$$x_{Front,C2} + x_{Centre,C2} + x_{Rear,C2} <= 15$$
$$x_{Front,C3} + x_{Centre,C3} + x_{Rear,C3} <= 23$$
$$x_{Front,C4} + x_{Centre,C4} + x_{Rear,C4} <= 12$$

Figure 4.2: Defining constraints - second example from Chapter 3

Algorithm 4.1 Expand sum expression

```
1:  function expand_sum_expression($expr$)
2:      $Values_{sum} \leftarrow finds\ columns\ associated\ with\ the\ sum\ keywords$
3:      $new_{expr} \leftarrow expr$
4:      for $value\ of\ value_{each}$ do
5:          $col_{value} \leftarrow get\ column\ data\ for\ column\ value$
6:          $sum_{elements} \leftarrow []$
7:          for $cell\ of\ col_{value}$ do
8:              $sum_{elements}.add(cell)$
9:          end for
10:         $new_{expr} \leftarrow new_{expr}.replace("sum" + value, sum_{elements}.join(" + "))$
11:     end for
        return $new_{expr}$
12: end function
```

The process of generating the constraints is not as forward as for generating the variables since creating the constraints in our language involves more blocks and complex operations that support polymorphism such as `operation blocks` with the X option. The first step to compute a constraint from our language to the mathematical formulation is to compute the `value blocks`, the output of the `value blocks` is dependent on the block used and its options and requires that information relative to some of the operations passes from the `value blocks` to the `operation blocks`, this information consists in indications for generating sums and iterations. To do so when accessing a particular value of a given column or variable we represent the value internally as *nameofcolumnorvariable[index_indexcel]*. When the user selects

Algorithm 4.2 Expand each expression

```
1: function expand_each_expression($expr$)
2:     $Values_{each} \leftarrow finds\ columns\ associated\ with\ the\ each\ keywords$
3:     $new_{exprs} \leftarrow []$
4:     for $value\ of\ value_{each}$ do
5:         $col_{value} \leftarrow get\ column\ data\ for\ column\ value$
6:         $new_{exprs}.add(expr.replace("each" + value, cell))$
7:     end for
         return $new_{exprs}$
8: end function
```

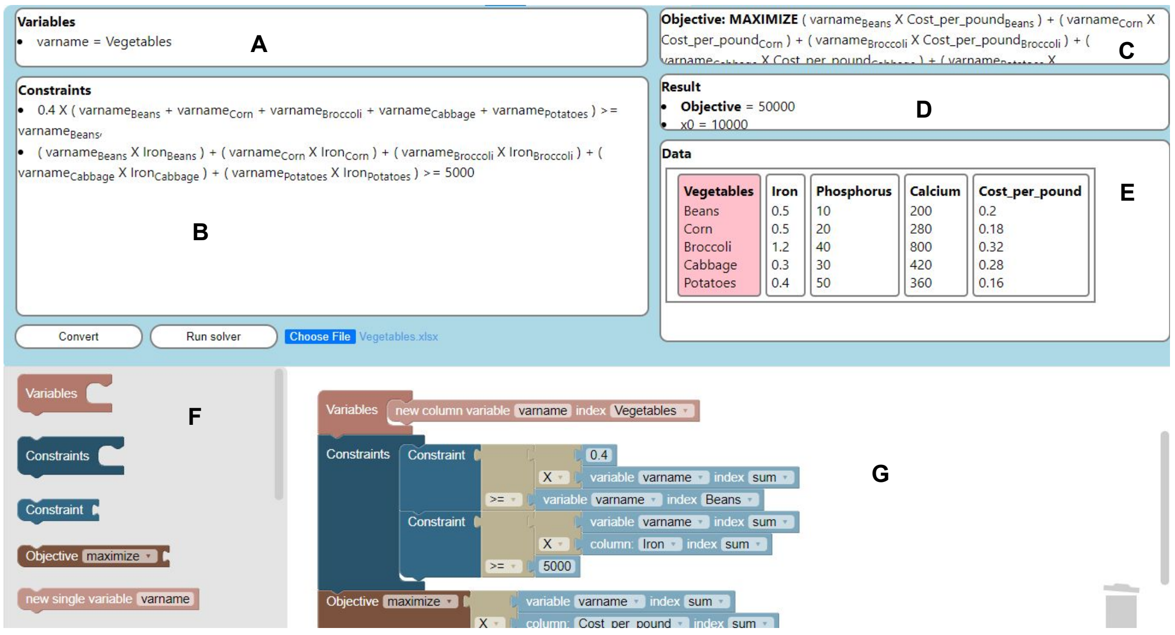
Algorithm 4.3 Generate sumproduct

```
1: function generate_sumproduct($expr$,)
2:     $Values_{each} \leftarrow finds\ columns\ associated\ with\ the\ each\ keywords$
3:     $new_{exprs} \leftarrow []$
4:     for $value\ of\ value_{each}$ do
5:         $col_{value} \leftarrow get\ column\ data\ for\ column\ value$
6:         $new_{exprs}.add(expr.replace("each" + value, cell))$
7:     end for
         return $new_{exprs}$
8: end function
```

`sum` or `each` we represent the value internally as *nameofcolumnorvariable[operation_index_indexcol]*.The purpose of expanding the `each` operation at the moment of parsing the inequation is due to the possibility that the iteration of one column is applied to more than one variable or column. The purpose of expanding the `sum` operation or the objective is due to the fact that this operation can be used in more than one context that changes its meaning such as using it as part of a `operation block` with the option *X* to create a *sumproduct* (as seen in Figure 3.4) of two columns or one column and one variable or standalone to represent the *sum* of a given column or variable.

After compiling the value block to their respective location, sum, or iterations of a column or variable we then compile the different `operation blocks`, the compilation of the `operation blocks` differs forthe given operations, for the inequations we first retrieve the different `each` indication and use them to generate the multiple constraints as seen in Algorithm 4.2 by first treating them as a set and generating constraints for all the permutations of the given columns. When generating the operation performed using the `X` option, the operation computed is dependent on the input given, if we are given either single values or a single value on one side and multiple values on the other, the single value is multiplied by each of the multiple values. When multiplying both values generated with the option `sum` a *sumproduct* as seen in Algorithm 4.3 operation is generated, we opted for this option relatively to doing a multiplication since we considered that it would be the most intuitive for our target audience. For the + and - operation options we use the first input as the left side statement and the second as the right side statement. When a `sum` isn't expanded as a *sumproduct* operation it is replaced by a sum of the different values of the given index column as seen in Algorith 4.1 The way we compile the objective is similar to the constraints, differentiating in not allowing for the use of `each` in `value blocks` and inequations in the `operation blocks`.

4.2.2 Interface overview and organization



Variables
- varname = Vegetables

Constraints
- 0.4 X ($\text{varname}_{\text{Beans}}$ + $\text{varname}_{\text{Corn}}$ + $\text{varname}_{\text{Broccoli}}$ + $\text{varname}_{\text{Cabbage}}$ + $\text{varname}_{\text{Potatoes}}$) >= $\text{varname}_{\text{Beans}}$,
- ($\text{varname}_{\text{Beans}}$ X $\text{Iron}_{\text{Beans}}$) + ($\text{varname}_{\text{Corn}}$ X $\text{Iron}_{\text{Corn}}$) + ($\text{varname}_{\text{Broccoli}}$ X $\text{Iron}_{\text{Broccoli}}$) + ($\text{varname}_{\text{Cabbage}}$ X $\text{Iron}_{\text{Cabbage}}$) + ($\text{varname}_{\text{Potatoes}}$ X $\text{Iron}_{\text{Potatoes}}$) >= 5000

Objective: MAXIMIZE ($\text{varname}_{\text{Beans}}$ X $\text{Cost\_per\_pound}_{\text{Beans}}$) + ($\text{varname}_{\text{Corn}}$ X $\text{Cost\_per\_pound}_{\text{Corn}}$) + ($\text{varname}_{\text{Broccoli}}$ X $\text{Cost\_per\_pound}_{\text{Broccoli}}$) + (

Result
- **Objective** = 50000
- x0 = 10000

Data

Vegetables	Iron	Phosphorus	Calcium	Cost_per_pound
Beans	0.5	10	200	0.2
Corn	0.5	20	280	0.18
Broccoli	1.2	40	800	0.32
Cabbage	0.3	30	420	0.28
Potatoes	0.4	50	360	0.16

A
B
C
D
E
F
G
Convert
Run solver
Choose File
Vegetables.xlsx
Variables
Constraints
Constraint
Objective maximize
new single variable varname
Variables new column variable varname index Vegetables
Constraints
Constraint
0.4
X
variable varname index sum
>=
variable varname index Beans
Constraint
variable varname index sum
X
column: Iron index sum
>=
5000
Objective maximize
variable varname index sum
X
column: Cost_per_pound index sum

Figure 4.3: Our web application interface

For the users to be able to access our language we create a web application that allows users to build LP models using our language seen in Figure4.3.F and Figure4.3.G while accessing the data loaded from a spreadsheet seen in Figure4.3.E and seen the model dynamically compiled into a valid formulation. The users can also run the model and get the results as can be visualized in Figure4.3.D.

The interface was built using *Typescript*, *React*, *Redux* and other web technologies. The blocks and workspace were implemented using the *react-blockly* library[2]. We decided to use *Typescript* for this project since it offers better reliability when compared to *JavaScript* especially when it comes to tasks related to generating the LP specification. We choose *React* and *react-blockly* since it was well suited for the highly dynamic data generated when the users loads data from spreadsheets, create and run the model.

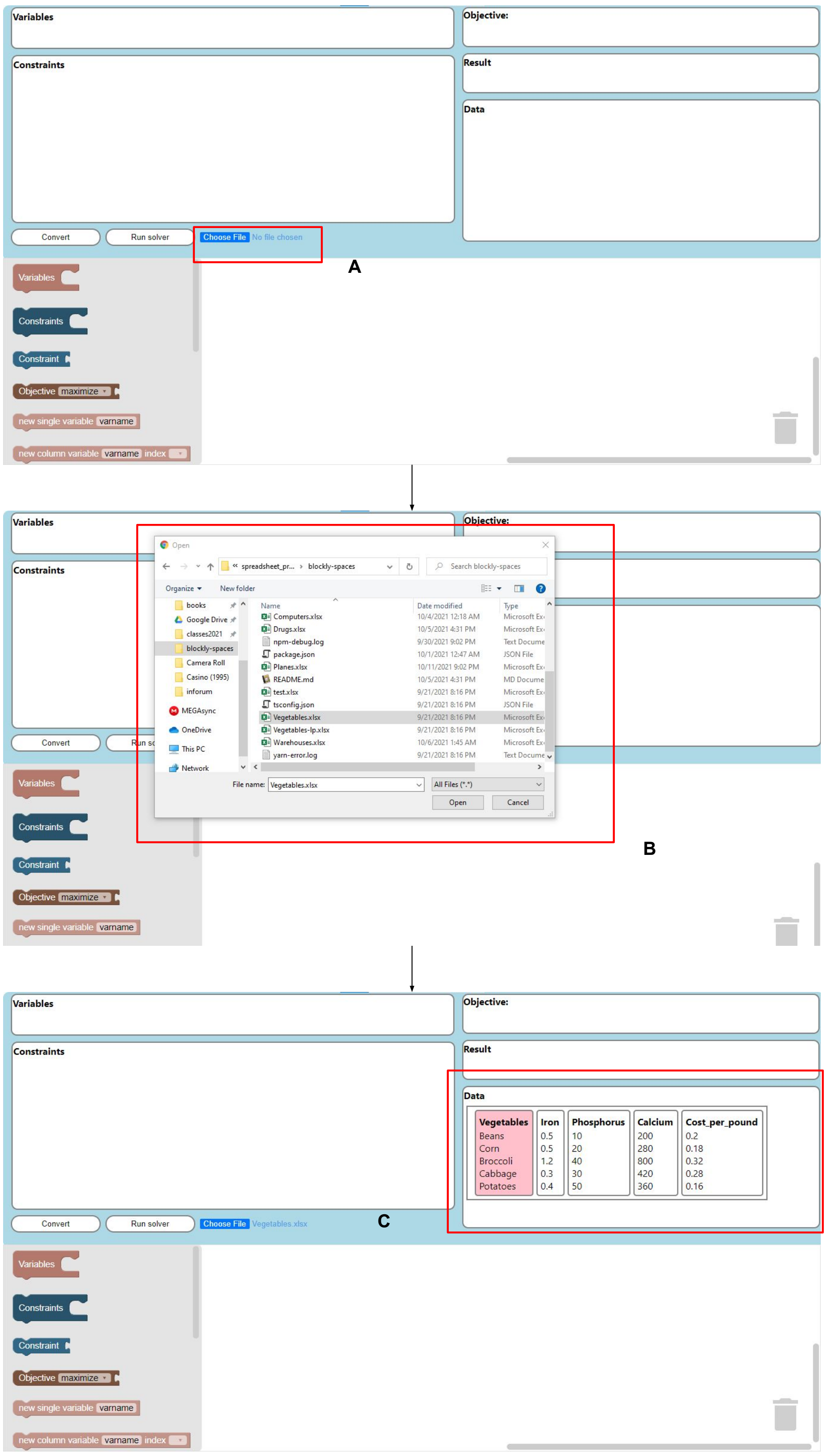
4.2.3 Features

4.2.3.1 Load and visualize data from spreadsheets

One of the core features of our interface is the ability that the users have to use spreadsheets to load the data used to build new models, this feature can be used to reuse the model created with compatible data.

[2]`https://www.npmjs.com/package/react-blockly`

As seen in Section3.1 the data used to build the model is extracted from a compatible spreadsheet the data is then shown in Figure4.3.E and can be used in the workplace when building the model as seen in Figure4.3.G.



Name	Date modified	Type
Computers.xlsx	10/4/2021 12:18 AM	Microsoft Ex
Drugs.xlsx	10/5/2021 4:31 PM	Microsoft Ex
npm-debug.log	9/30/2021 9:02 PM	Text Docume
package.json	10/1/2021 12:47 AM	JSON File
Planes.xlsx	10/11/2021 9:02 PM	Microsoft Ex
README.md	10/5/2021 4:31 PM	MD Docume
test.xlsx	9/21/2021 8:16 PM	Microsoft Ex
tsconfig.json	9/21/2021 8:16 PM	JSON File
Vegetables.xlsx	9/21/2021 8:16 PM	Microsoft Ex
Vegetables-lp.xlsx	9/21/2021 8:16 PM	Microsoft Ex
Warehouses.xlsx	10/6/2021 1:45 AM	Microsoft Ex
yarn-error.log	9/21/2021 8:16 PM	Text Docume

File name: Vegetables.xlsx

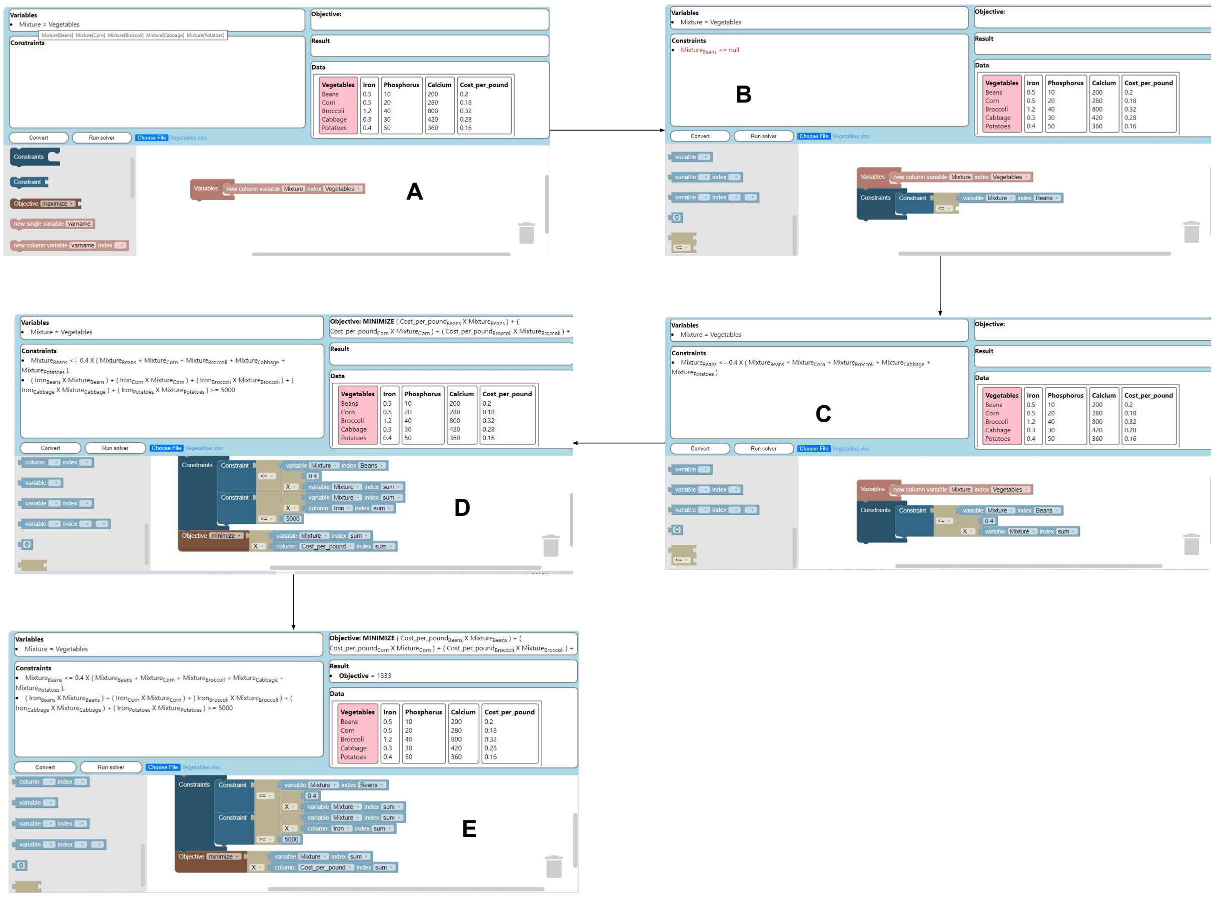
Vegetables	Iron	Phosphorus	Calcium	Cost_per_pound
Beans	0.5	10	200	0.2
Corn	0.5	20	280	0.18
Broccoli	1.2	40	800	0.32
Cabbage	0.3	30	420	0.28
Potatoes	0.4	50	360	0.16

Figure 4.4: Loading data in our application

4.2.3.2 Dynamic model generation

Being able to visualize the generated LP formulation is crucial for users learning how to use our language and to add a layer of safety between the model creation in our language and the execution of the generated model. To further improve this process users can visualize the model being created in the workspace(in Figure4.3.G) using our language in it's mathematical formulation (in Figure4.3.A, Figure4.3.B and Figure4.3.C). This feature is also useful for experimentation since users can observe the impact of each change in the workplace in the resulting model and thus can correct possible mistakes and achieve better efficiency when building their models.

The dynamic generation features can be seen in Figure4.5 in various steps of building the model. In Figure4.5.A the user created a `column variable` and can expand on the variables generated by hovering over the list of generated variables(in the section seen in Figure4.3.A). The user then created the constraints(in Figure4.5.B and Figure4.5.C) and the constraints generated by our tool can be seen in real-time. In Figure4.3.D the user added the objective and in Figure4.5.E the user ran the model and received the results.



Variables
- Mixture = Vegetables

Mixture[Beans] Mixture[Corn] Mixture[Broccoli] Mixture[Cabbage] Mixture[Potatoes]

Constraints
- $Mixture_{Beans}$ <= null

Constraints
- $Mixture_{Beans}$ <= 0.4 X ($Mixture_{Beans}$ + $Mixture_{Corn}$ + $Mixture_{Broccoli}$ + $Mixture_{Cabbage}$ + $Mixture_{Potatoes}$),
- ($Iron_{Beans}$ X $Mixture_{Beans}$) + ($Iron_{Corn}$ X $Mixture_{Corn}$) + ($Iron_{Broccoli}$ X $Mixture_{Broccoli}$) + ($Iron_{Cabbage}$ X $Mixture_{Cabbage}$) + ($Iron_{Potatoes}$ X $Mixture_{Potatoes}$) >= 5000

Objective: MINIMIZE ($Cost\_per\_pound_{Beans}$ X $Mixture_{Beans}$) + ($Cost\_per\_pound_{Corn}$ X $Mixture_{Corn}$) + ($Cost\_per\_pound_{Broccoli}$ X $Mixture_{Broccoli}$) +

Result
- **Objective** = 1333

Data

Vegetables	Iron	Phosphorus	Calcium	Cost_per_pound
Beans	0.5	10	200	0.2
Corn	0.5	20	280	0.18
Broccoli	1.2	40	800	0.32
Cabbage	0.3	30	420	0.28
Potatoes	0.4	50	360	0.16

Figure 4.5: Dynamic model generation in our application

4.2.3.3 Error messages

Another feature of our tool is the addition of dynamic error checking and messages, our goal with this feature is to help novice and more experienced users debug the constructed models as well as helping users learn our language and catch mistakes commonly associated with creating LP models and in using Block-based languages.

LPBlocks offers the following error messages:

- **Column with the given name already exists**: This error is given when the user attempts to create a variable with a name already used for an existing column. When outputting this message the variable generated by our model is colored red and the user can visualize more information about the error by hovering the mouse above the given variable.
- **Missing inequation in constraint**: This error is given when a user attempts to create a constraint without using an `operation block` with an inequation option.
- **More than one inequation in constraint**: This error is generated when the user attempts to create a constraint with two ore more inequations. This error is shown by highlighting the constraints associated with this attempt in red with a more descriptive message appearing when the user hovers the constraints.
- **Null values in expression**: This message is given when an expression contains Null values.
- **Variable multiplication**: This error is generated when the user attempts to multiply two variables together. This message consists of a red highlight of the affected expressions with a more descriptive message consisting of the names of the variables that the user is attempting to multiply.
- **Empty fields**: This error is output when the user did not select the field in a selectable block.
- **Inequation in objective**: Given when the user attempts to use an inequation when creating the objective.
- **Each used in objective**: When the user attempts to use the option `each` in an objective.

4.3 Spreadsheet reading service

Listing 4.1: Spreadsheet data in JSON format

```
{
  [
 {
   Vegetables: 'Beans',
   Iron: 0.5,
   Phosphorus: 10,
    Calcium: 200,
    Cost_per_pound: 0.2
  },
  {
    Vegetables: 'Corn',
    Iron: 0.5,
    Phosphorus: 20,
    Calcium: 280,
    Cost_per_pound: 0.18
  },
  {
    Vegetables: 'Broccoli',
    Iron: 1.2,
    Phosphorus: 40,
    Calcium: 800,
    Cost_per_pound: 0.32
  },
  {
    Vegetables: 'Cabbage',
    Iron: 0.3,
    Phosphorus: 30,
    Calcium: 420,
    Cost_per_pound: 0.28
  },
  {
    Vegetables: 'Potatoes',
    Iron: 0.4,
    Phosphorus: 50,
    Calcium: 360,
    Cost_per_pound: 0.16
  }
]
}
```

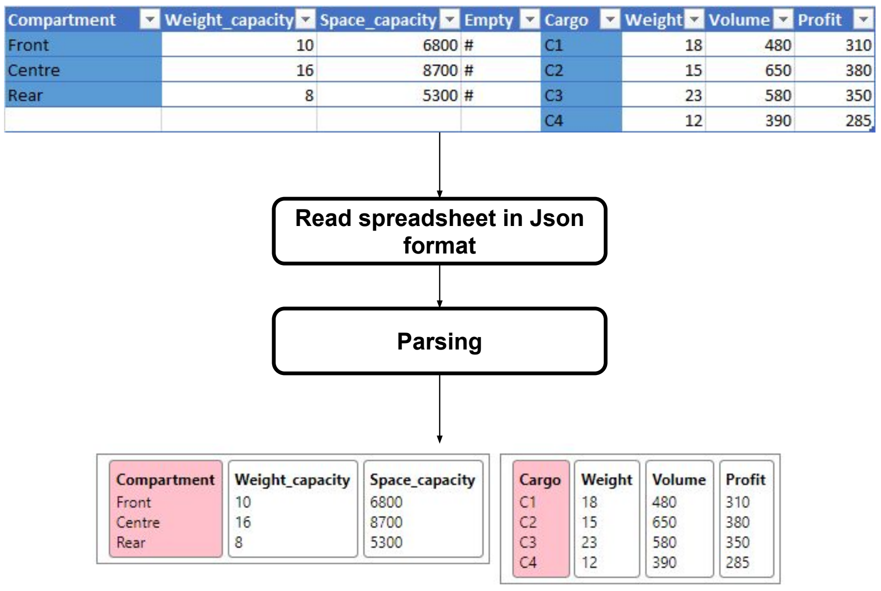
Listing 4.2: Input for the Solver API

```
{
    model: `Maximize
        obj: + 0.6 x1 + 0.5 x2
        Subject To
        cons1: + x1 <= 1
        cons2: + 3 x1 + x2 <= 2
        End`
}
```

This service was created as a *Rest API* using *Express.js*[3] that accepts the upload of a given spreadsheet file and then reads it using *node-xlsx*[4] library and sends back the contents of spreasheet in *JSON* format as seen in Listing 4.1.

To translate the data in the spreadsheet into our internal representation used to generate the constraints(this process can be seen in 4.6 we load the data from the file into memory in the *JSON* format. Subsequently, we parse the *JSON* data into our internal representation, this process comprises the following steps:

- Finding the index columns as seen in Algorithm 4.4, we do this by iterating over the list of columns names and selecting the first column and the first ones after an "Empty"column as seen in Algorithm 4.4
- Finding the indexes for each column as seen in Algorithm 4.5, to do this we iterate over the column names and set each column index column as the previous index column.
- Finding the column data as seen in Algorithm 4.6, to do this task we first iterate over the column names and set the data



Compartment	Weight_capacity	Space_capacity	Empty	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

Compartment	Weight_capacity	Space_capacity
Front	10	6800
Centre	16	8700
Rear	8	5300

Cargo	Weight	Volume	Profit
C1	18	480	310
C2	15	650	380
C3	23	580	350
C4	12	390	285

Figure 4.6: Input data for the running example problem

[3]https://expressjs.com/
[4]https://www.npmjs.com/package/node-xlsx

Algorithm 4.4 Find index columns

```
function find_indexes($cols$)
    $index_{cols} \leftarrow []$
    $index \leftarrow True$
    for col of colnames do
        if $index == True$ then
            $index_{cols} += col$
            $index \leftarrow False$
        end if
        if $col ==$ ”$Empty$” then
            $index \leftarrow True$
        end if
    end for
     return $index_{cols}$
end function
```

Algorithm 4.5 Find column indexes

```
function find_column_indexes($cols$)
    $column_{indexes} \leftarrow Map < string, string > ()$
    $index \leftarrow True$
    $index_{column} \leftarrow$ ””
    for $col\ of\ cols$ do
        if $index == True$ then
            $column_{indexes}[col] = index$
            $index \leftarrow False$
        end if
        if $col ==$ ”$Empty$” then
            $index \leftarrow True$
        end if
    end for
     return $column_{indexes}$
end function
```

Algorithm 4.6 Find column data

```
function find_column_data($cols$)
    $column_{data} \leftarrow Map < string, string > ()$
    for $col\ of\ cols$ do
        $column_{data} \leftarrow cols[col]$
    end for
     return $column_{indexes}$
end function
```

4.4 Optimization service

The optimization service uses *Express.js* once again to create a *Rest API*. This *API* receives a *JSON* structure that includes the model in a format that the solver can run. The *API* then runs the model using the *clp-wasm*[5] library and sends the output back. This library is a port of the *COIN-OR* LP solver to *WebAssembly*. To use this service we also transform the formulation in our web application internal structure to the format seen in Listing 4.2.

[5] `https://www.npmjs.com/package/clp-wasm`

Chapter 5

Language applicability

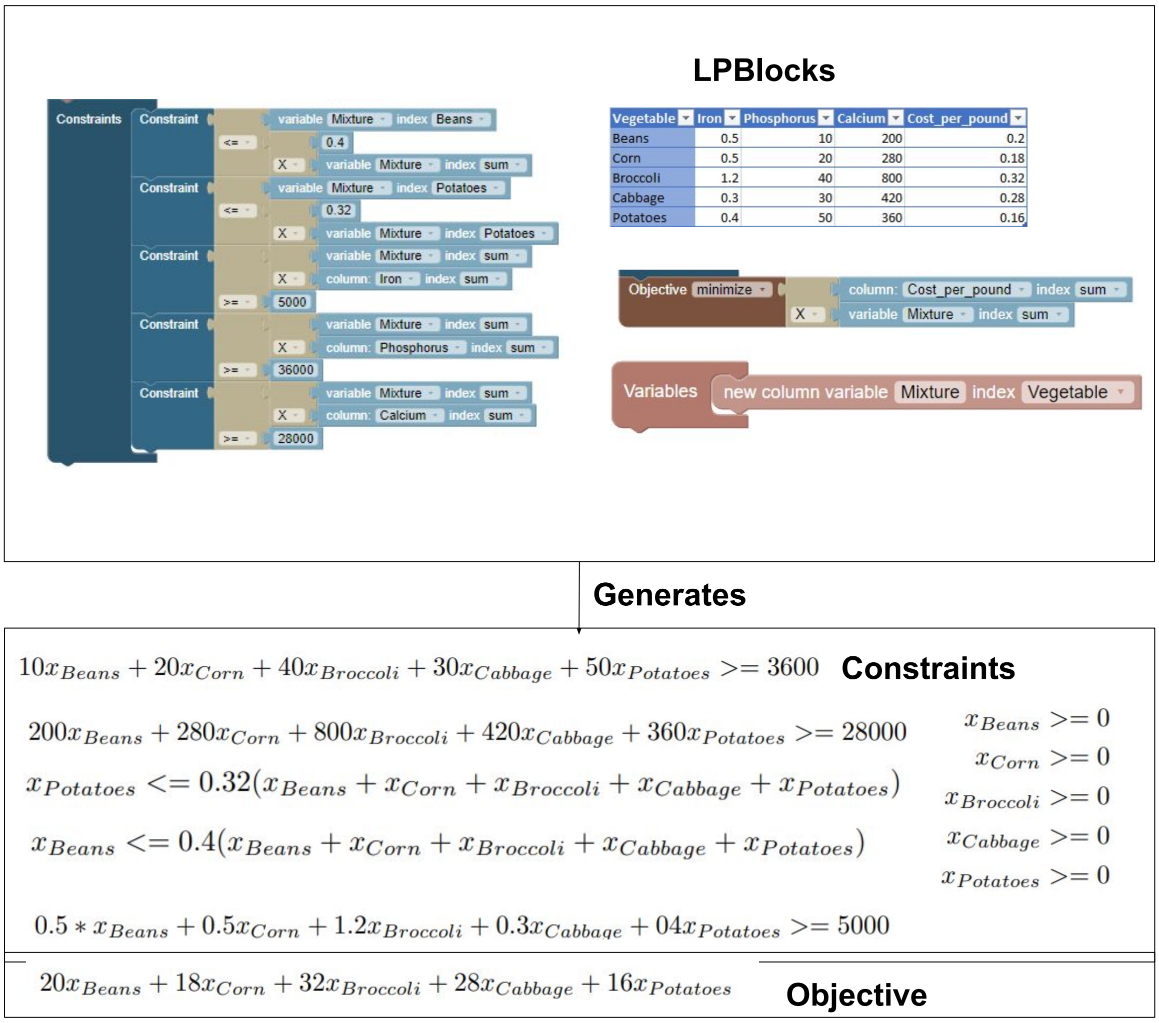
In this section we present a set of LP problems taken from an MBA exam [4] and from an Operations Research textbook, [5] and modeled using LPBlocks. With this, we intend to illustrate the applicability of our language to a broad set of examples since doing so gives us some assurance that our language could be used as a replacement for another LP tool and users would still be able to solve their problems

5.1 Vegetable mixture

This example shown in Figure 5.1 comes from the operations research textbook [5]. In this example, a manufacturer of freeze-dried vegetables aims at reducing production costs while adhering to various nutrition criteria and guidelines. We are given nutritional data for each of the vegetables as well as their cost per pound in the tabular data. We have a maximum percentage for certain vegetables and the lower bounds for certain nutrients.

Since our goal is to find the ratio of each vegetable that goes into the mixture, we created a `column variable` named `Mixture` that takes as its input the column `Vegetable`. For the constraints we start by creating constraints imposing limits of 40% for Beans and 32% for Potatoes. The following three constraints define the lower bounds for the given nutrients and the last adds non-negativity. The objective is to minimize the cost per pound of the mixture. The verbosity of this model could be improved if support for inputing data in the form of matrices in the spreadsheet was added. This would decrease the necessity of using single value blocks since iteration through both indexes would be possible.

When compared with implementing this model using mathematical notation, LPBlocks allows for the creation of a more concise and intuitive model. We feel that our language lends itself to this sort of problem since it allows for the generation of sizable mathematical constraints using more concise business logic.


LPBlocks
Constraints
Constraint
variable Mixture index Beans
<=
0.4
X
variable Mixture index sum
variable Mixture index Potatoes
0.32
variable Mixture index Potatoes
column: Iron index sum
>=
5000
column: Phosphorus index sum
36000
column: Calcium index sum
28000
Objective minimize
column: Cost_per_pound index sum
Variables
new column variable Mixture index Vegetable
Generates
Constraints
Objective

Vegetable	Iron	Phosphorus	Calcium	Cost_per_pound
Beans	0.5	10	200	0.2
Corn	0.5	20	280	0.18
Broccoli	1.2	40	800	0.32
Cabbage	0.3	30	420	0.28
Potatoes	0.4	50	360	0.16

$$10x_{Beans} + 20x_{Corn} + 40x_{Broccoli} + 30x_{Cabbage} + 50x_{Potatoes} >= 3600$$

$$200x_{Beans} + 280x_{Corn} + 800x_{Broccoli} + 420x_{Cabbage} + 360x_{Potatoes} >= 28000$$

$$x_{Potatoes} <= 0.32(x_{Beans} + x_{Corn} + x_{Broccoli} + x_{Cabbage} + x_{Potatoes})$$

$$x_{Beans} <= 0.4(x_{Beans} + x_{Corn} + x_{Broccoli} + x_{Cabbage} + x_{Potatoes})$$

$$0.5 * x_{Beans} + 0.5x_{Corn} + 1.2x_{Broccoli} + 0.3x_{Cabbage} + 04x_{Potatoes} >= 5000$$

$$x_{Beans} >= 0$$
$$x_{Corn} >= 0$$
$$x_{Broccoli} >= 0$$
$$x_{Cabbage} >= 0$$
$$x_{Potatoes} >= 0$$

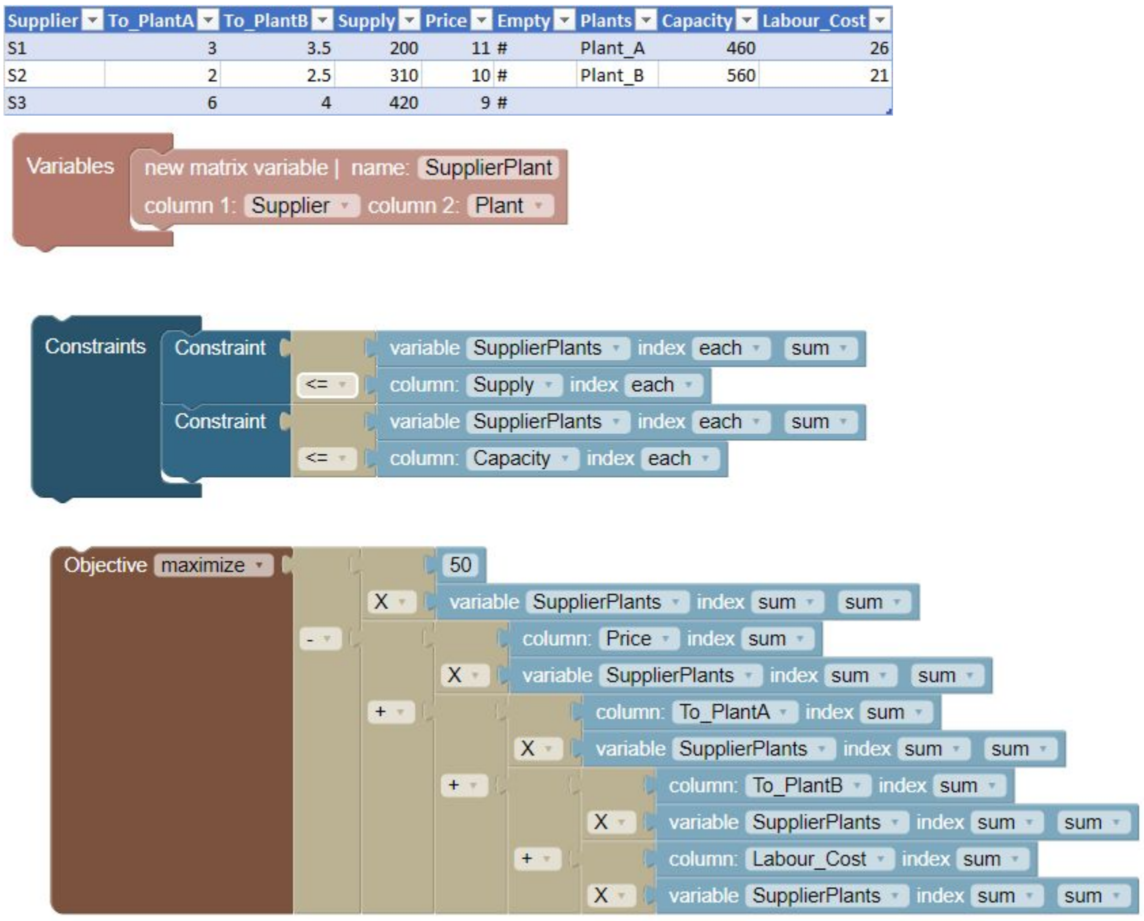
$$20x_{Beans} + 18x_{Corn} + 32x_{Broccoli} + 28x_{Cabbage} + 16x_{Potatoes}$$

Figure 5.1: Definition of the vegetable mixture problem using LPBlocks

5.2 Fruit canning plants

In this example taken from an MBA exam [4] and shown in Figure 5.2 we are given information associated with different suppliers and fruit canning plants with the goal of maximizing its profits. The information includes shipping, labor and operating costs, buying prices and maximum production capacities. Despite not being in the spreadsheet, the problem definition states that the selling price for each tonne is of $50.

To generate the formulation we create a `matrix variable` with indexes `Supplier` and `Plant`. The constraints for this problem are straightforward and can be generically specified, this consisting of the upper bounds for the supply and capacity for each of the plants. The objective function is considerably more complex since it needs to take into account the selling price and all the costs to represent the profit. The objective verbosity could be mitigated by adding the support for defining matrices inside the spreadsheet for the data. To improve the reusability of the created models adding support to define data variables inside the spreadsheet could also be done. To help users better visualize the model we will add support for intermediate values to store portions of more verbose components. More concretely in this model we could create blocks for the different operating costs used in the objective and reference these



Supplier	To_PlantA	To_PlantB	Supply	Price	Empty	Plants	Capacity	Labour_Cost
S1	3	3.5	200	11	#	Plant_A	460	26
S2	2	2.5	310	10	#	Plant_B	560	21
S3	6	4	420	9	#			

```
Variables   new matrix variable | name: SupplierPlant
            column 1: Supplier   column 2: Plant

Constraints Constraint        variable SupplierPlants index each  sum
                       <=     column: Supply index each
            Constraint        variable SupplierPlants index each  sum
                       <=     column: Capacity index each

Objective maximize
            50
      X     variable SupplierPlants index sum  sum
  -         column: Price index sum
        X   variable SupplierPlants index sum  sum
    +       column: To_PlantA index sum
          X variable SupplierPlants index sum  sum
      +     column: To_PlantB index sum
          X variable SupplierPlants index sum  sum
        +   column: Labour_Cost index sum
          X variable SupplierPlants index sum  sum
```

Figure 5.2: Fruit canning plant example modeled using LPBlocks

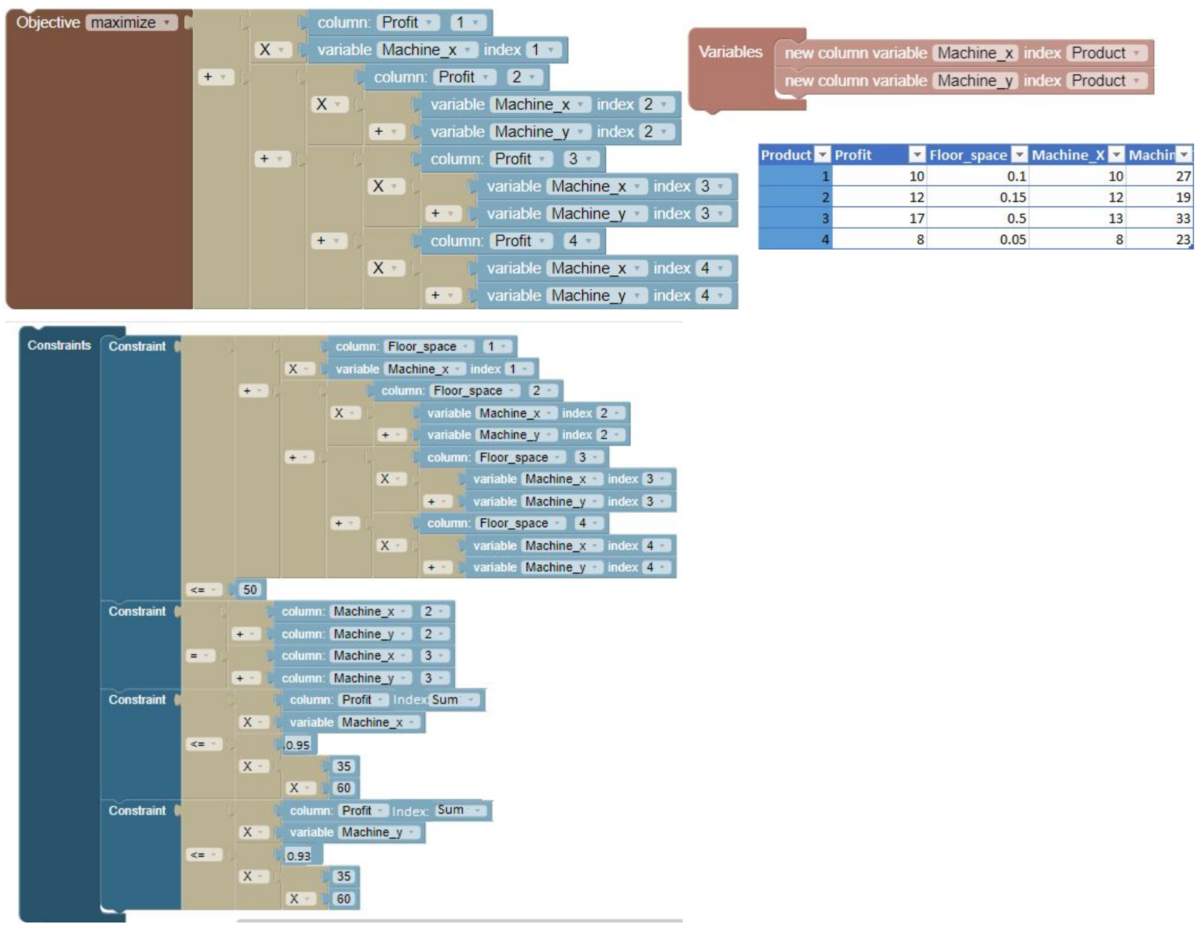
blocks at the moment of its definition.

When comparing with traditional mathematical notation or even other solutions LPBlocks allowed for considerable savings in terms of syntax for the expression of constraints however we found that due to its complexity the objective function specification in LPBlocks was similar to the mathematical form.

5.3 Machine allocation

In the problem shown in Figure 5.3 (taken from [4]) the goal is to maximize a factory's profit by allocating the production of different goods among two machines. In this problem we are given information about each product's profitability, use of floor space and manufacturing time in minutes taken by each machine. We are also given other rolls related to the machines down time, the total floor space of $50m^2$, the time of a work week of 35 hours, the ratios of which some products have to be produced relatively to others and that `Product` 1 can only be manufactured in the second machine.

In this example we create `column variables` for each machine taking the `Proudct` column as the index as opposed to previous examples where we created `matrix variables`. In this we did not create a `matrix variable` as could be assumed do to the fact that the values for the machines are not used as



```
Objective maximize
  column: Profit 1  X  variable Machine_x index 1
  + column: Profit 2  X  ( variable Machine_x index 2  +  variable Machine_y index 2 )
  + column: Profit 3  X  ( variable Machine_x index 3  +  variable Machine_y index 3 )
  + column: Profit 4  X  ( variable Machine_x index 4  +  variable Machine_y index 4 )

Variables
  new column variable Machine_x index Product
  new column variable Machine_y index Product

Constraints
  Constraint
    column: Floor_space 1  X  variable Machine_x index 1
    + column: Floor_space 2  X  ( variable Machine_x index 2  +  variable Machine_y index 2 )
    + column: Floor_space 3  X  ( variable Machine_x index 3  +  variable Machine_y index 3 )
    + column: Floor_space 4  X  ( variable Machine_x index 4  +  variable Machine_y index 4 )
    <=  50
  Constraint
    column: Machine_x 2  +  column: Machine_y 2
    =
    column: Machine_x 3  +  column: Machine_y 3
  Constraint
    column: Profit Index Sum  X  variable Machine_x
    <=
    0.95  X  ( 35  X  60 )
  Constraint
    column: Profit Index: Sum  X  variable Machine_y
    <=
    0.93  X  ( 35  X  60 )
```

Product	Profit	Floor_space	Machine_X	Machin
1	10	0.1	10	27
2	12	0.15	12	19
3	17	0.5	13	33
4	8	0.05	8	23

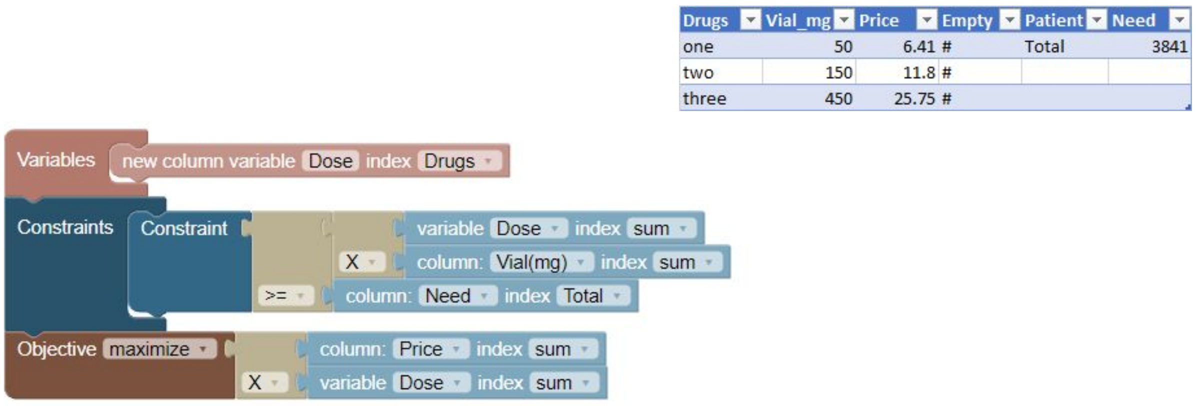
Figure 5.3: Machine allocation problem in LPBlocks

index columns and using them for defining a `matrix variable` would mandate the referencing of one of the values for every use of the variable and would offer nothing in terms of iterability and generalization. In terms of constraints we use the first constraint to express that the maximum floor space use is of $50m^2$. If LPBlocks supported matrices as data input we possibly could express this constraint in a less verbose manner. In the second constraint we express that the production of product 2 is the same as 3. In constraints three and four we take into account the downtime of 5% for machine 1 and 7% for 2 by modeling that the total running time of each machine must be lower or equal to 95% and 93% of the total work week. The objective aims to maximize the profit and takes into account that `Product` 1 can only be manufactured in the second machine. This is the reason we were not able to use a more generic representation for this constraint.

Similarly to the previous example we found some benefits in terms of smaller footprint but since the creation of the model required the use of complex mathematical operations it couldn't make use of most of LPBlocks features.

5.4 Pharmaceutical company

This problem was taken from a *Excel* optimization tutorial[1]. Our goal in this problem is to reduce the manufacturing costs associates with a given drug. The data that we are given consists in three packagings of the same drug. The quantity of drug per vial for each type of the drug is also given, we also get the price per package and total needed in miligrams by the patients. In this model the variables are the quantity of each of the packages, the constraint is that the total produced of the drugs in miligrams must be hiher or equal that the total requested by the patients. The objective is to minize the the production costs of the packagings.



Drugs	Vial_mg	Price	Empty	Patient	Need
one	50	6.41	#	Total	3841
two	150	11.8	#		
three	450	25.75	#		

Figure 5.4: Drug manufacturing problem in LPBlocks

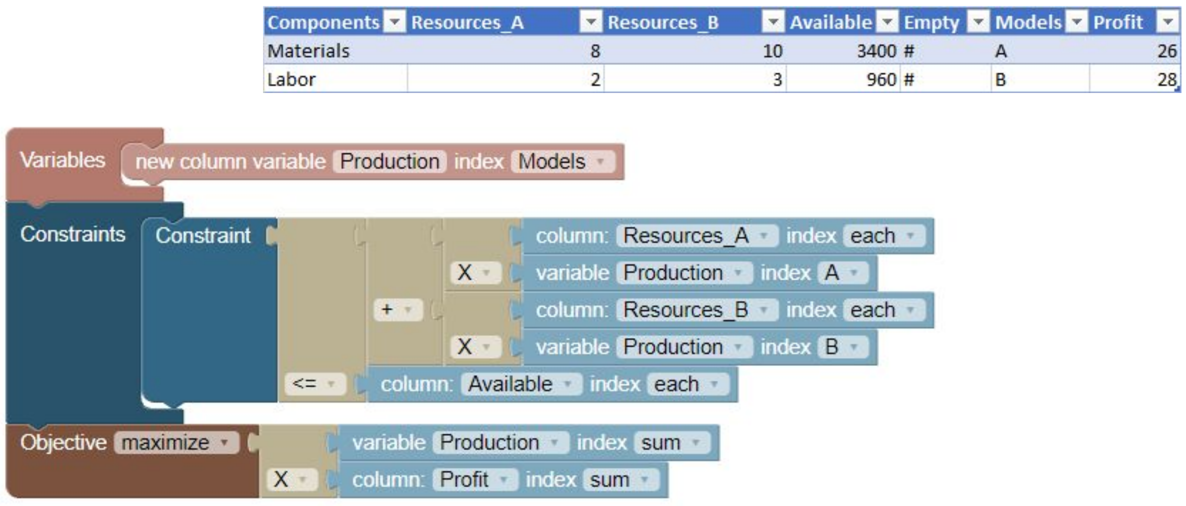
To represent the variables we create a column variable with the column `Drugs` and called this variables `Dose`. To represent our constraint we use a `operation block` with the option $<=$, this blocks first input is another `operation block` with the option X in this block we feet a `variable block` with index sum and a `column block` with the option sum this block represents a sumproduct operation between the variables `Dose` and the column `Vial_mg`. The first `operation block` is also connected to the value *Total* of the column `Need`. The objective was created by doing a sumproduct of column `Price` and the variables `Dose`.

5.5 Computer manufacturing

In this problem taken from the a computer manufacter needs to decide the quanities of each of two computetr models to manufacture. For this we are given two components necesary for the manufacture a computer model this being *Materials* and *Labor*. We also get for each of the components the costs for the manufacture of one model A and for one model B we also get the available quantity of each of this components. We also get the computer models and profit associated with each of them.Our constraint in

[1]`https://www.exceltactics.com/using-solver-to-optimize-solutions-to-costing-problems-in-excel/`

this problem is to guarantee the resources used dont surpass their values. The objective is to maximize profits.



Components	Resources_A	Resources_B	Available	Empty	Models	Profit
Materials	8	10	3400	#	A	26
Labor	2	3	960	#	B	28

```
Variables   new column variable Production index Models
Constraints Constraint
              column: Resources_A index each
            X variable Production index A
          +
              column: Resources_B index each
            X variable Production index B
        <=  column: Available index each
Objective maximize
              variable Production index sum
            X column: Profit index sum
```

Figure 5.5: Computer manufacturing problem in LPBlocks

To represent this problem we first start with the variables, for this we start by creating a `column variable` named `Production` with the column `Models` as its index, doing it allows us to create a variable for each of the two models. To create the constraint we use `opertion blocks` with the X option and the $each$ option as the index of `column Available` and the `variable Production` to multiply the production by the resources for each of the `Components`.

5.6 Satellite launcing

This problem was taken from a Operation Research textbook [5]. In this problem, we did not use spreadsheet data for our formulation. For this we know that a company has two payloads(T1 and T2) and wants to calculate the number of satellites that carry each payload, we also have access to other information such as the success rate of satellites carrying payload T1 and T2 and the profit for each successful payload transportation.

We define two `single variables` in LPBlocks for our variables. The constraints express various launch success rates, launch times, and maximum launches for each of the loads. The objective is to maximize the profit for the loads.

5.7 Cargo allocation

This is the example used for presenting the language in previous chapters. To recapitulate our goal in this problem is to maximize a shipping plane's profit by allocating four types of cargoes amongst three plane sections.

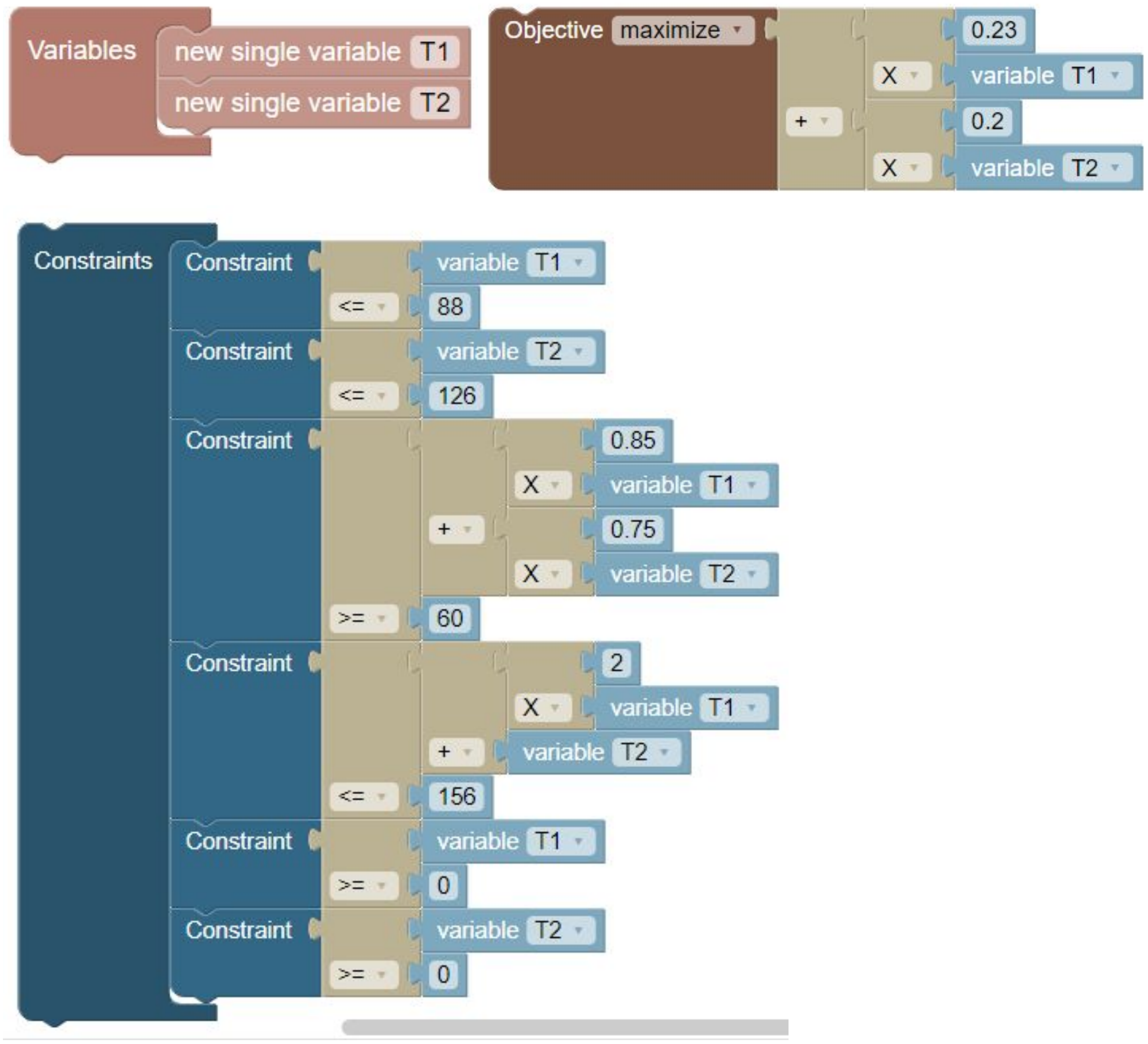
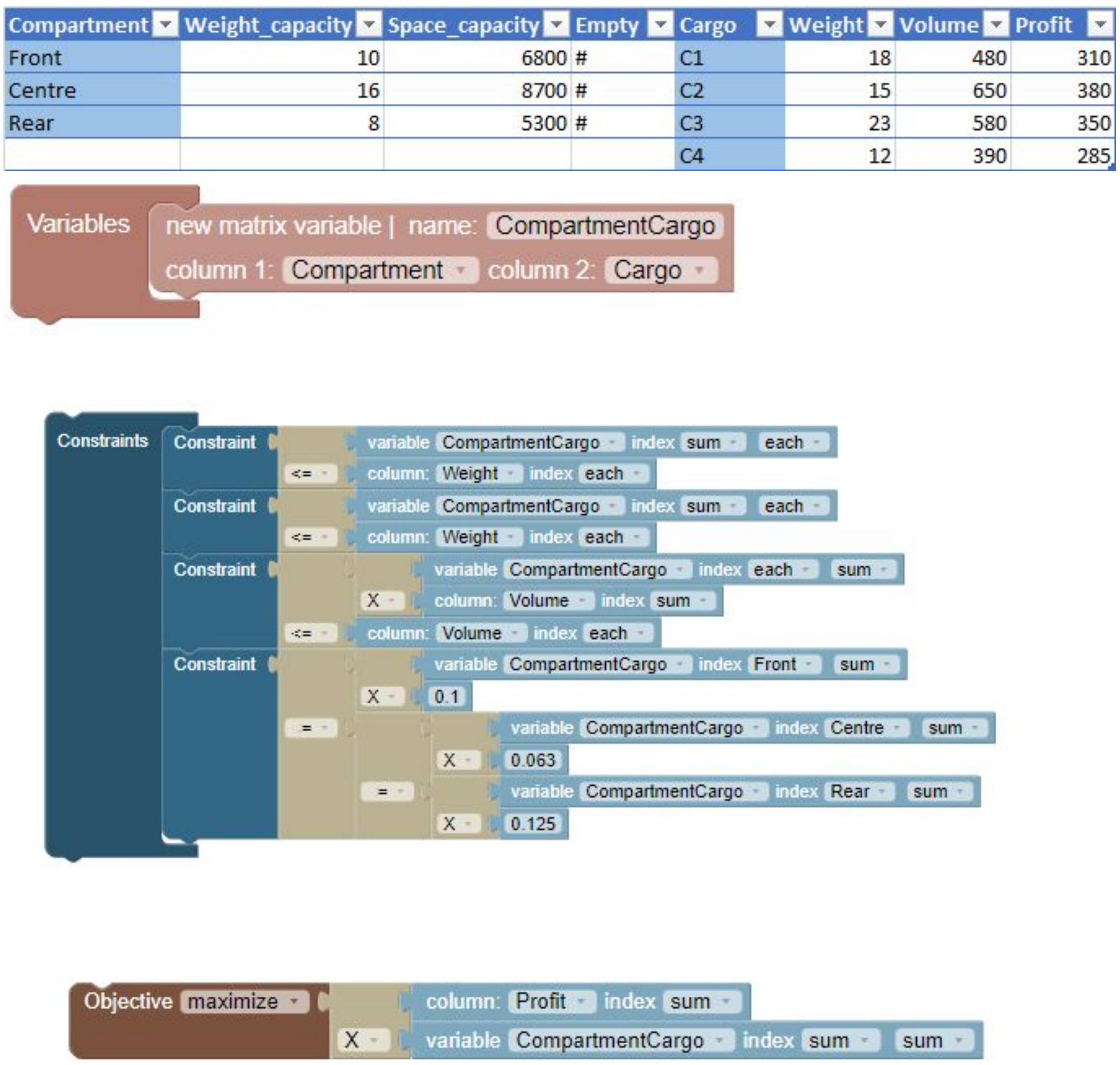


Figure 5.6: Satelite launching problem in LPBlocks

The variable created is a `matrix variable` that takes the `index columns Compartment` and `Cargo` as its input.

For the constraints, the first three relate to space, volume, and weight capacity limitations. LPBlocks shines in these three constraints in the sense that when compared with writing a standard mathematical specification we need to write considerably less specification as doing it in our language allows us to write a generic specification of the constraint we want to represent and the compiler can generate the associated constraints in the mathematical notation. The last constraint related to balancing the plane's cargo in the different sections had to be done manually since LPBlocks does not possess a block capable of representing multiple equalities using a single block. We considered adding this block to our language but found that this specific use case was not prevalent to a level warranting adding the block. The objective function aims to maximize profits by calculating the *sumproduct* of the `Profit` with the sum of our variables elements indexed by the given `Profit` element.



Compartment	Weight_capacity	Space_capacity	Empty	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

```
Variables
  new matrix variable | name: CompartmentCargo
  column 1: Compartment   column 2: Cargo

Constraints
  Constraint
      variable CompartmentCargo index sum   each
    <=
      column: Weight index each
  Constraint
      variable CompartmentCargo index sum   each
    <=
      column: Weight index each
  Constraint
        variable CompartmentCargo index each   sum
      X
        column: Volume index sum
    <=
      column: Volume index each
  Constraint
        variable CompartmentCargo index Front   sum
      X
        0.1
    =
          variable CompartmentCargo index Centre   sum
        X
          0.063
      =
          variable CompartmentCargo index Rear   sum
        X
          0.125

Objective maximize
    column: Profit index sum
  X
    variable CompartmentCargo index sum   sum
```

Figure 5.7: Cargo allocation problem in LPBlocks

Chapter 6

Empirical evaluation

In this chapter, we showcase an empirical study realized with users of different backgrounds, our goal is to assess if our tool has potential applications in both industrial and education settings by observing how users react to its different features and how proficient users from different backgrounds become with our language, tool and the concepts behind them in a relatively short period. In Section 6.1 we showcase the design of our study and the reasons behind it. In Section 6.2 we showcase the problems and datasets used for the study. In Section 6.4 we present the data gathered from forms filled by the users and collected from by us during the interview process. In Section 6.5 we present our conclusions and critical analysis of the data gathered.

6.1 Design

As said previously our aim is to observe the experience that different users have when using our tools to solve a plethora of problems and to collect their reactions and feedback.

To achieve the desired outcome in this study we planned on doing hour-long sessions with each of the participants. During each of the sessions we plan on doing an initial introduction to LP and how our language tackles those types of problems, we then introduce our tool and its features. After doing so we solve a demo exercise in front of the user, followed by a joint exercise where the user solves the problem with extensive input and two individual exercises where the user solves the problem with our input only when necessary.

Beyond solving the problems we created an individual follow-up questionnaire about their experience with our language and tool, prior experience and education, and other basic information such as age and gender. The questionnaire questions can be seen in Appendix A for questions related to the users background and Appending B for user feedback.

Since the goal of this study is to get qualitative feedback from users and to see their difficulties and access if our solution viability with different types of users, we aim to have participants from different backgrounds and with different levels of experience with LP we contacted via email and word of mouth possible participants with diverging academic and professional experiences.

6.2 Instrumentation

During the session, we use several problems taken from the book and class notes referenced in Chapter 5 and whose datasets in a spreadsheet format can be found here [1]. We use a web implementation of our tool that can be found in [2].

We used the same set of problems in all our sessions and those where:

- **Vegetable mixture** - We used a portion of the problem seen in Section 5.1 containing only constraints retaining to the maximum ratio of beans in the mixture and the absolute content of iron in the mixture. We used this problem as the tutorial problem. We decided to use this problem as the introductory problem since it showed a balance of complexity and relatability with the number of our tool features shown during its resolution.
- **Fruit canning plants** - We used this problem as is in Section 5.2 to use as the joint exercise, this exercise is arguably more complex than the ones to thought to be solved more independently by the user but allows the user to experience most of the features needed for solving the future exercises.
- **Pharmaceutical company** - This problem is the first individual problem that the participants are asked to solve, this problem seen in Section 5.4 is notably simpler than the previous problem and its simplicity lets the user focus on some of the core features of our tool and has the added benefit of decreasing the fear and intimidation for users less familiar with LP.
- **Computer manufacturer** - This problem seen in Section 5.5 is the last of the individual problems, this problem is more complex than the first of the individual problems but allows the participants to put into practice various of our language features such as *sumproducts* between variables and columns and the use of the *each* option and requires a more thorough thinking process for the interpretation and expression of the constraints in our language.

6.3 Execution

To find participants for our study we contacted people in our circles as well and student groups of relevant subjects as said in Section 6.1 our goal was to have a diverse group of participants and in our case that entailed increasing the efforts to acquire participants from non-computing backgrounds. Overall we were able to find 7 participants. Of this group 4 of the participants had a higher education degree in a

[1] https://drive.google.com/drive/folders/1SAle03APg4JZefFIleS3-1hYGJ3gURI

[2] https://lpblocks.herokuapp.com/

computing field and were either currently Master or Ph.D. students. The other 3 had no computing degree and worked in the business field, one of them worked in consulting and the other 2 work entailed doing web marketing, customer service, and logistics for e-commerce companies. All participants had at least a bachelor's degree and all participants from the computing background are male and from the business background were female.

The first step when starting the session was to ask users to download the datasets for the session When starting the session we start with a briefing on LP and some of its uses and some of the difficulties associated with its use, after doing so we introduce our solution, explain that the results of the study will be anonymous, and reassure the participants that our goals are to collect qualitative feedback about our tool, that our tool has a learning curve and that we are not passing any judgment on the participants.

After our introduction we start with the tutorial, this tutorial is composed of an explanation and introduction to our language, this includes how the data is loaded from spreadsheets and its representation in the tool, our language blocks ant their uses, we also give an explanation of some aspects of the compilation process and some of the caveats of our tools, we explain to the participant how to navigate our interface and finally do a live demonstration of one LP model creation using our tool. During this process especially during the live demo, we encourage the participants to ask as many questions as they need.

Our next step after the tutorial is to let the user solve one of our problems to do so we ask the participant to change his screen and then after an explanation of the problem and data we guide the user on how to solve the problem using our solution. During this step we allow the user to do some exploration and allow the user if he is capable to solve as much of the problem as he can.

Following the joint exercise, we start the first evaluation exercise, both the evaluation exercises are of a lower difficulty than the explanation exercise, the first one being easier than the second, after explaining the problem and data we allow the user to do as much as he can only intervening when the user asks for help or its necessary to avoid the user going on a wrong path. We then do the second problem similarly.

After solving the problems we ask the user for any feedback and proceed to end the session and direct him to the form.

6.4 Data collection

In this section we showcase tha data collected from the forms, in Subsection 6.4.1 the portion relating to the participants background and in Subsection 6.4.3 the results relating to the feedback given by the users. In Subsection 6.4.2 we describe the individual sections with each user.

6.4.1 Background data

In this subsection, we present the background information about the participants of our study. The questions in the form relating to the subject are available in Appendix A.

Participant	Age	Gender	Gender	Education time	(Education field	LP experience	Optimization tools
1	25-35	Male	Master	8	Computing	Took college classes	Programming tools
2	25-35	Male	Master	8	Computing	Took college classes	Excel
3	25-35	Male	Master	5	Computing	Took college classes	Programming tools, Other visual tools such as GAMS
4	21-24	Male	Bachelor	5	Computing	Took college classes	Programming tools, Excel
5	more than 35	Female	Master	6	Engineering and Business	None	None
6	21-24	Female	Master	5	Business	None	None
7	21-24	Female	Bachelor	3	Business	None	Excel

Table 6.1: Participants background gathered from the survey

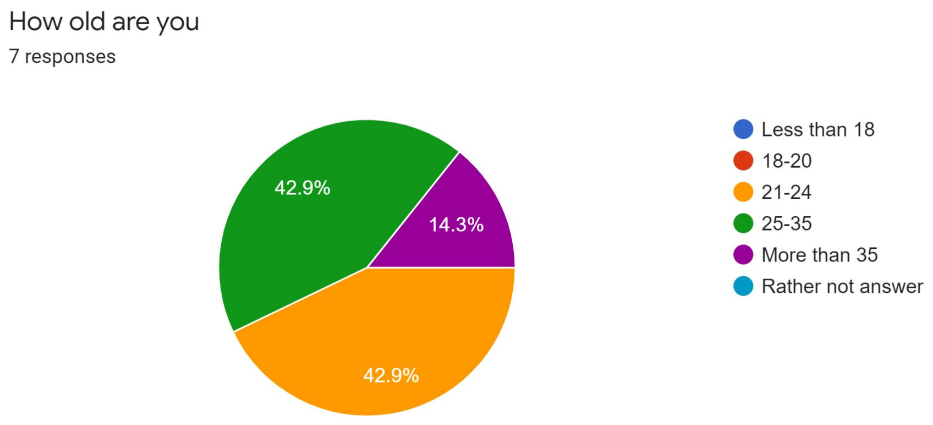


Figure 6.1: Participants age distribution

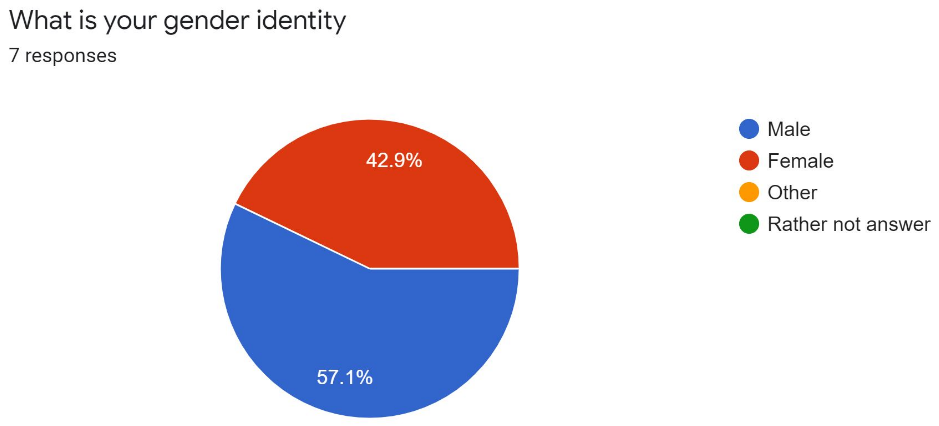


Figure 6.2: Participants gender distribution

From the information in 6.1 and the graphs extracted from it, we were able to observe that our participants have an age distribution predominantly between 21-24 and 25-35, the distribution is split evenly between those two groups, beyond those we had one person over the age of 35 (Figure 6.1). In terms of gender as seen in Figure 6.1 we have a slightly male predominating having 4 males and 1 female. In terms of academic background, we have 4 persons with an exclusively computing background, 2persons with

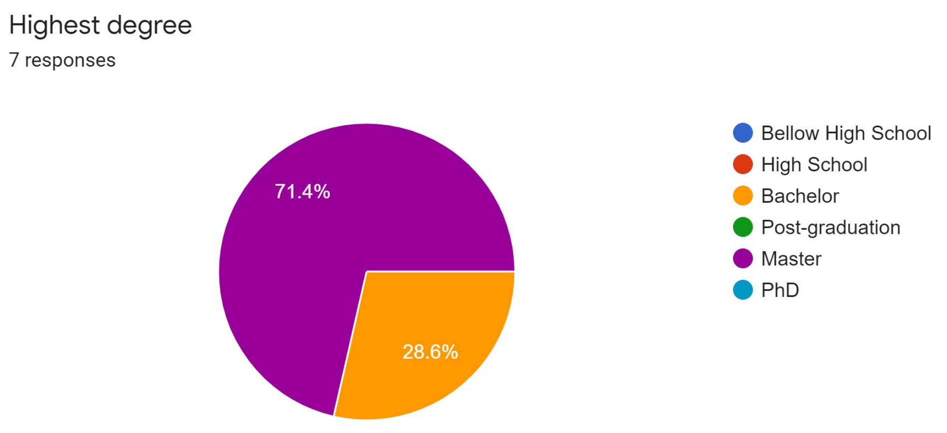


Figure 6.3: Participants highest degree distribution

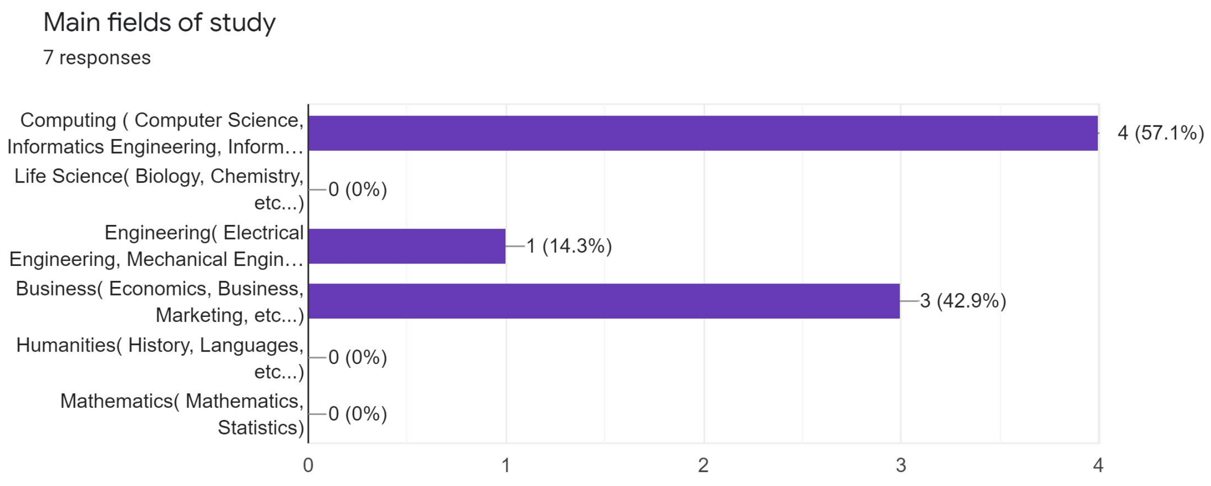


Figure 6.4: Participants main field of education

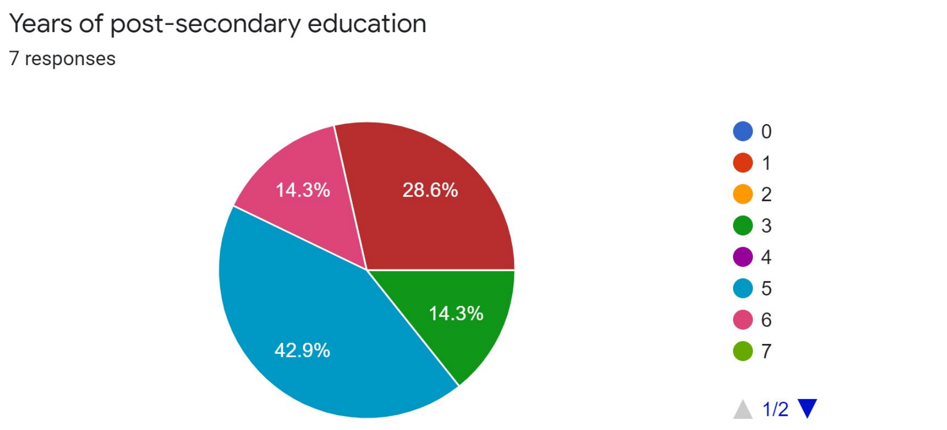


Figure 6.5: Participants years of post secondary education

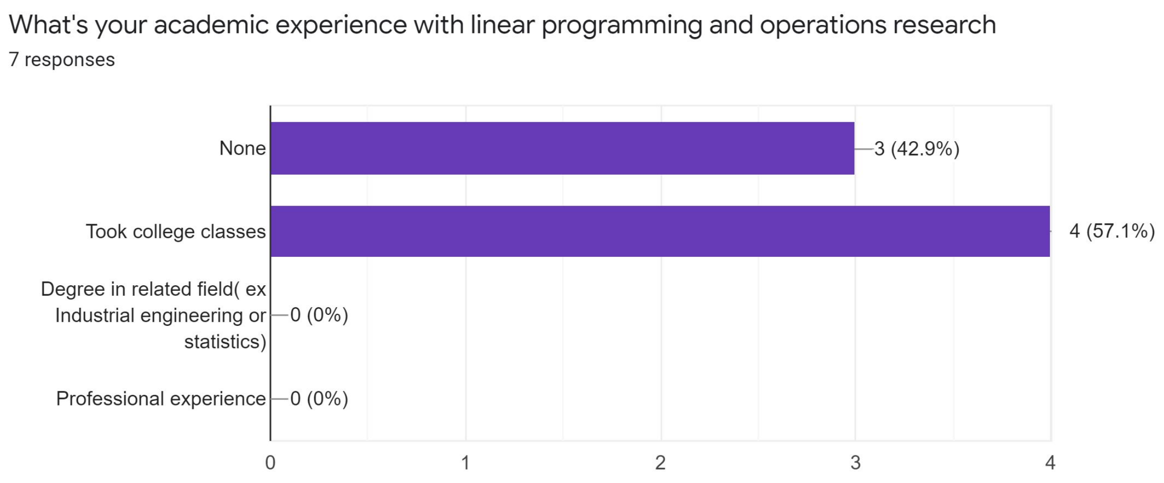


Figure 6.6: Participants prior experience with LP

a business background, and 1 person with both a business and engineering background (Figure 6.4). In terms of degree, all our participants had post-secondary degrees, we had slightly more participants with a Masters than with Bachelors (Figure 6.3) and none with Ph.D., of those some of them had concluded more years beyond their highest completed degree (Figure 6.5). When it comes to prior experience with LP we had 4 persons who took college classes and 3 persons with none (Figure 6.6).

6.4.2 Sessions

In this subsection we describe the one on one session with the different participants.

6.4.2.1 Participant 1

During the tutorial the participant was able to understand the language and our goals, special emphases had to be made when explaining the data and variables. The participant was proactive and was able to solve a considerable portion of the joint exercise. The participant also demonstrated interest in our tool and language.

Relatively to the first individual problem, the participant tried to maximize the objective instead of minimizing, this error could have been made since the user might not have understood that our goal with this problem was not to increase the profit but to decrease production costs since the participant might have been more accustomed to maximization problems. Other than that the user did not need considerable amounts of help.

When solving the second problem the user had some difficulties distinguishing between the option `sum` and `each` in the constraints and objective. This user did demonstrate more difficulty in solving this problem than the previous one especially on how to express the costs in our language but with some help the user was able to get to the correct solution.

Overall the user had a positive reaction to our tool and was able to do a good portion of the exercises and we felt that the user did understand the core of our language and tools and that with some more experience would be able to achieve mastery of our tool.

6.4.2.2 Participant 2

During the language presentation the subject found himself interested in the language and its features, this was demonstrated by the subject asking several questions.

When doing the first demonstration problem the user did have some difficulties on where to place the blocks. After the initial error, he was able to understand the logic behind the variable creation. Other problems subsequent during the demonstration were related to differences between `each` and `sum`, confusion between `sum` values from one column or variable, and using the `operation block` with the value `+`. This was solved. Despite some initial confusion, the participant was able to understand the language.

During the first exercise, the major difficulty was representing the production in milligrams as the *sumproduct* of the vial quantity and the produced vials. The user was able to get the answer and understand the reasoning with some help.

The main difficulty in this problem was using two constraints one for the labor and the other for the materials instead of using an `each` for both. The user ended up understanding the logic behind the model and did the rest of the model by himself.

Overall we found that the users understood the core concepts of our language and with some training could become proficient in the latter.

6.4.2.3 Participant 3

The participant was attentive and asked several questions when watching the tutorial.

When doing the first exercise the user was proactive and was able to solve a significant portion of the latter. Some difficulties involved the use of each and the use of columns and variables.

When doing the second exercise the user was able to solve the problem on his own, the biggest problem was in using the correct block for addressing variables and discerning between `column` and `variables blocks`.

When doing the third exercise the user experienced some confusion between the use of each and sum but was able to solve the exercise.

After our session, manifested interest in our tool and gave some of his opinions related to our tool such as adding more solver options and exporting data to excel files.

6.4.2.4 Participant 4

The participant was interested in the language and tool and asked several questions. During the example exercise, the participant asked questions about block positioning and language features, particularly about the `sumproduct` operation.

During the joint exercises, the user was proactive, the user found some initial difficulties when trying to represent the constraints and objectives in our language but ended up being able to do so with some help. The main problems were related to understanding how the data correlated with the desired goal of the constraints and objective, this might be related to the user not using LP regularly.

In the first exercise, there were some difficulties in creating the variables particularly, in creating 3 variables since the user tried to use 3 `column variable blocks` to create 3 variables, the user was after visualizing the model generated by our tool able to get the correct way to build the model. Understanding how to represent the constraint with the data took some tries but the user ended up with some guidance and visualizing the output getting to the correct answer. For the constraint, the user was able to get there by visualizing the result, error messages and some trial and error.

In the second exercise the participant the main difficulty was in representing the variables, the rest of the model took some trial and error but with minor guidance and output visualization, the user was able to get to the solution. The user did do a manual *sumproduct* instead of using the *sumproduct* operation but that was a result of the trial and error process. With more practice, the user might gain some dexterity with the tool.

6.4.2.5 Participant 5

During the tutorial the participant was able to understand the purpose of our language and tool, special emphases had to be made when explaining the data and variables. Despite not being from a computing background this participant had also an engineering background and understood some of the concepts that our solution tackles, however, the user did demonstrate some skepticism to our language and its complexity for non-technical users.

When doing the tutorial the user did not ask as many questions as the previous ones but did ask some, we tried to explain the concepts and how our tool could be used and the user seem to understand what we were doing. The user did complement the interface and the features but did feel intimidated by our tool and doubted her ability to solve problems such as the tutorial problem without our help, in response that we reassured that after doing an exercise with us and gaining some experience the tool becomes easier and more intuitive.

During the solving of the joint exercise we allow the user to try to understand the problem and ask any questions and give her input, some of the difficulties the user had was in understanding the differences between the blocks for variable access and variable creation, we answered to that and were able to convey their functions. Some of the operations such as *sumproducts* weren't obvious. The user at the end of this exercise expressed some doubts about the virtue of our tool for users without any prior programming experience and admitted that despite her prior programming experience in one or two classes in college she still had difficulties understanding and working with our language.

During the first individual problem the user was able to create the variables, understanding the mapping of our language to the mathematical formulation was something and how to write the expression was something that did not occur immediately but after the initial awkwardness and some further explanation, the user was able to get over those hurdles and once again understanding the differences between variables and columns and how the constraint in the natural language would be expressed in terms of data were only possible with some help and further explanations but the user was able to get there on her own afterward.

We weren't able to do the last problem due to time constraints we had planned on doing a one-hour session but by the time we finished solving the first individual problem the time was almost over and the user could not continue.

6.4.2.6 Participant 6

The user was interested during the presentation and during the demonstration exercise with special emphasis to explaining clearly the uses and workings of LP and optimization, however, the participant did manifest some level of fear related to using our tool. Some of this previous hesitation had to do with the user not having a strong mathematical background. Despite the hesitation, the user did manifest some excitement with our tool and despite not doing any optimization in her line of work, she currently does use excel for tasks related to marketing and business in general, this includes keeping track of stocks and inventory and interacting with logistic software.

We did most of the joint exercise with this participant, but we saw that during the explanation the user was interested in the different features of our language and was able to understand some of the solutions we created and the logic behind it and did show some appreciation for interface and with special relevance for the dynamic features, data visualization, and blocks.

During the first individual problem the user showed some initial difficulty understanding what variable to create but after explaining the purpose of the problem again and some trial and error with different blocks the user was able to get to the correct solution. For creating the constraint understanding where to use the *inequation* was an initial problem but was quickly corrected with some explanation. Creating the *sumproduct* was the harder part and both the blocks and which column and variable blocks and options to use require some extensive explanation but in the end, the user got to the solution and was able to understand the logic behind the use. For creating the objective the user still needed help but was able to get to the solution faster than for the constraint.

When doing the last problem the user faced some difficulties when understanding the problem and creating the variable, this time the blocks weren't an issue. For the constraints, there were still issues of blocks such as trying to use the wrong `variable block` to access a variable and a considerable amount of help was needed for the user to solve the problem but eventually, she did and there was some improvement compared with the first problem especially in understanding the blocks and how they were used. After solving this exercise user express her liking for the interface and the blocks and affirmed her liking for the block-based language and made some comments about possible improvements such as improving the distinction between the constraint and variables value blocks.

6.4.2.7 Participant 7

When introducing the language the participant demonstrated interest in our solution, since she had work experience with excel and found it to have some potential, however since the user did have little to no prior experience with LP the user felt intimidated by it at first. We made a special emphasis on explaining what LP entails and in explaining its components and how they relate to our language. The user during the presentation and tool showcase also expressed her liking for the interface and the language aesthetics and how the data was displayed.

We did most of the joint exercise with this participant, but we saw that during the explanation the user was receptive to the different features of our language and was able to understand the solution we created and the logic behind it.

During the first exercise, there was some difficulty in choosing which variable to use, but after reexplaining, the problem and how variables are created the user was able to get to the correct block, and using our visualization features the user was assured she made the correct choice. There were also some difficulties in representing the value in milligrams from the variable created and the value in milligrams for each vial but the user was able to eventually get there with some help. The user was able to get the value in the spreadsheet to represent the total necessary medication in milligrams. The user was able to get the objective after some trial and help.

During the second problem, the user was able to get the variable and part of the constraints, once again the user needed help in doing the *sumproduct* and there was still some confusion in using the `variable` and `column block` and some help was warranted to achieve the correct solution. Also, some help was necessary to get to the objective but the user was able to solve portions of the problem. Despite the lack of previous experience with LP the user still appreciated the tool and was able to understand some concepts especially using our visualization and dynamic features and gave some advice consisting especially in improving the differentiation between `variable` and `column blocks`. When creating the constraints there was still some confusion on how columns and variables were used, help she was able to get to the correct answer.

6.4.3 Participants feedback

During the survey we alsos strive to get feedback about our tool from the users as well as some ideas and possible improvements that the users might believe would benefiet our tool, to do so we added the questions in Appending B to our survey, the results from this questions can be seen in Table 6.2.

6.5 Analysis and conclusions

In this section, we present some conclusions and critical analysis related to the data gathered from our survey and the sessions and their implication on our work and possible future paths, changes, and applications.

Participant	Other tools	Positive aspects	Negative aspects
1	Better usability.	Usability; Performance;	
2	This tool has a strong visual component. Although Excel is algorithmically ”more complete”	UX is nice. Drag and drop works well. Also nice shortcuts: delete, ctrl+c ctrl+v. Constraints and Objective ”sub-windows”are useful to understand if the inequations match the user’s reasoning.	Took me a while to understand the difference between ”light-blue”variable block and ”light-blue”column block. Each and sum are also a little confusing at first (but the previously mentioned sub-windows help fixing mistakes).
3	Easy to learn interface, good visual aid (in the formula section - we can compare what the blocks translate to in terms of mathematical equations)	Easy to learn interface, good visual aid (in the formula section - we can compare what the blocks translate to in terms of mathematical equations). No installation needed is a plus.	No option export to file. For bigger problems / harder to solve problems, the lack of alternative solvers could result in not being able to solve the problem.
4	Is not drag and drop like this app.	Simpler	Is not intuitive to beginners
5		Easy to use, to grasp and get together.	Terms of the bottons. Depending on the client if he doesn’t have the right knowledge it will be difficult to think in a problem solving logic.
6		Visualy atractive; Well organized; All the information is well explained when building the model.	The operations ordering is confusing.
7		Very intuitive, easy to use, visualy appelative and well organized. We can visualize the model being created	The order of the operations is not very intuitive.

Table 6.2: Participants feedback gathered from the survey

According to the data gathered from our sessions and survey, we can separate our participants into two different groups, one being the users 1 to 4 or the users with a computing background and 5 to 7 or the users with a business background. By coincidence, the computing group was all male and the business group all female, the computing group had on average more years of higher education and probably due to their field of study had prior experience with LP. Age-wise the differences weren’t starling.

From our sessions, we were able to gather that all the participants were interested in our tool and interface and across the board (although possibly due to selection bias) asked various questions this differing according to the user background being slightly more technical in nature from the computing group, we also found that users universally liked the interface and the how the data was accessible, this included the business group where the participants particularly enjoyed having all the data, workspace and mathematical formulation in the same place. however, during the tutorial and introduction to our solution, we found that the business group showed either skepticism for participant 5 or hesitation for participants 6 and 7. This was expected due to them not having as much experience in the field.

In terms of actual dexterity and level of proficiency acquired when using our tool during this session we can conclude that the computing group did considerably better than the business group, this was evident by the considerably higher portion of the problems they were and lesser input needed. However we still found some difficulties that prevailed across the users of this group, we found that the relation between the data and the specification in natural language wasn't a considerable problem for this group except some less obvious cases, we found that distinguishing between *sum* and *each* initially might cause difficulties in some cases but are quickly resolved after creating a couple of expressions using this options, we also found in some cases that the users of this group did not use the language as intended but that quickly resolved itself. Distinguishing between columns and variables was initially a problem but was eventually solved. For this group, we also found that the dynamic compilation to the mathematical formulation was of considerable help in learning our language.

When doing the exercises with the business group we found that the mastery with our tool acquired by the users varied considerably. We found that user 7 did better than user 6 and that user 5 despite having only done the first individual exercise. However, with this group, we found some common problems such as misusing blocks, confusing variables, and columns, difficulties doing the operations such as *sumproducts*, difficulties with block order, and how to construct mathematical operations using the blocks. We also found that these users also had difficulties in interpreting the exercises and in understanding how the data correlated with the goals of the problems, however, we found that as the session progressed their ability to do so improved.

From the feedback gathered from the participants, seen in Table 6.2 we found that for the computing group the negative aspects focus on our tool having some learning curve for beginners, the difficulty in distinguishing between `variable` and `column blocks`, other negative aspects for this group related to the lack of features such as no options to export or import models and the lack of different solvers. The positive aspects for this group were related to our interface features and usability such as the ability to drag and drop, the ability to visualize the mathematical formulation being built in real-time, the ability to visualize the data, and our tool performance and not requiring installation. When compared to other tools this group found that it had better usability, the interface was easy to learn and was especially helpful due to the dynamic compilation and errors allowing users to learn the language as they go along, the drag and drop capabilities were also something that our application did better than the ones previously used by the participants, the participants in this group had previously used programming tools, Excel and one of them had used graphical software.

The business group in the feedback questions expressed that the negative aspects were that the language could be difficult for people who don't have experience in technical fields and that the order in which blocks have to be inserted in the workplace to create an expression differs from the mathematics. This group found also positive aspects in our language such as ease of use and starting building models, the language being visually appealing, being able to see the model being built in real-time and our tool is intuitive and well organized. These participants had no prior experience with other optimization tools.

Overall we found that some features of our tool such as a graphical interface that displays everything needed by the user on a single page, the dynamic compilation, and errors were universally liked, we also found that despite the users in the computing group having done better than the users in the business group both ended our session with some knowledge on how to use our tool. Some of the challenges felt by the participants were for the majority distinguishing between certain blocks such as columns and variables and for users without a computing background, the block order to create expressions was counterintuitive. We feel that for the users with less experience in the field our tool can serve as an introductory experience to LP and operations research and as well and with some added features such as loading and exporting data and more solver options our tool could be used in industry with more experience users or after more training for less experienced ones.

Chapter 7

Conclusions and future work

Here we present our final conclusions and answer to research questions asked in Chapter 1 this is done in Section 7.1 and in Section 7.2 we present our expected future work.

7.1 Conclusions

Current tools for creating LP models often require previous programming knowledge or use *ad-hoc* methodologies and lack some features that would benefit the less technical user, in our work we were able to design a block-based language that can express those problems and create a tool that coupled this language with various features to create an environment that lent itself to users of different technical backgrounds.

We were able to successfully use LPBlocks to express numerous and varied LP problems in Chapter 5. Further, we took our implementation of LPBlocks, contacted possible participants of different backgrounds, designed and ran a study to collect data referent to their experience with our tool and language.

During this study, we were able to gather that some features were universally liked such as the dynamic compilation and errors, drag and drop, the ability to see everything in one window, and the use of a block-based language. We also found that some aspects of LPBlocks cause more problems such as differentiating between `column blocks` and `variable blocks`, some of the logic behind the language and thinking in terms of data to solve problems wasn't obvious for less technical users and some users would have liked to see more features such as data loading and exporting and more solver options.

We now answer the research questions posed in Section 1.3:

- **RQ1** - *Can we use block-based languages to represent LP formulations.* - Yes as seen in Chapter 5 LPBlocks can be used to represent numerous LP formulations.
- **RQ2** - *Will using a block-based language provide users an easier environment for LP.* - This question is not as straightforward from our study we were able to conclude that the users that had previously

used Excel and other optimization software compared ours favorably when it came to the graphical interface, however not all users in our study had experience with those tools and none of the users without a computing background did. Despite the lack of comparison for those users, they did enjoy and benefit from the graphical and block-based features of our tool, features that don't exist on most of the other tools.

7.2 Future work

Our future work aims are to use the data gathered in Chapter 6 to continue improving our tool and language from a user experience perspective and further test our tool using a higher number of participants. Taking the previously said into account we defined the following tasks as possible future work:

1. Improving the differentiation of `variable blocks` and `column blocks` in LPBlocks, this could be done by changing the color scheme used for those blocks and making it more similar to the `variable creation blocks` for the `variable blocks` and to the index columns for the `column blocks`.
2. Adding more features to our tool such as loading and exporting data and mode solver options.
3. Prompting the user with automatic suggestions and templates when building models.
4. Doing further research with a higher number of participants and more time dedicated to teaching the language and some of the processes necessary to create LP models, especially for users with less experience in the field.

Bibliography

[1] D. R. Anderson, D. J. Sweeney, T. A. Williams, J. D. Camm, and J. J. Cochran. *Quantitative Methods for Business*. url: https://www.amazon.com/Quantitative-Methods-Business-David-Anderson/dp/0840062346.

[2] A. C. Bart, E. Tilevich, C. A. Shaffer, and D. Kafura. "Position paper: From interest to usefulness with BlockPy, a block-based, educational environment." In: *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*. 2015, pp. 87–89. doi: 10.1109/BLOCKS.2015.7369009.

[3] A. C. Bart, J. Tibau, E. Tilevich, C. A. Shaffer, and D. Kafura. "BlockPy: An Open Access Data-Science Environment for Introductory Programmers." In: *Computer* 50.5 (2017), pp. 18–26. doi: 10.1109/MC.2017.132.

[4] J. E. Beasley. *Or-notes*. url: http://people.brunel.ac.uk/~mastjjb/jeb/or/lpmore.html.

[5] M. Carter and C. C. Price. *Operations Research: A Practical Introduction*. CRC Press, 2000. isbn: 9780849322563.

[6] M. Chikwature. *Challenges faced by pupils in the learning of linear programming at ordinary level: A case of a secondary school in Umguza District*. 2018. url: http://liboasis.buse.ac.zw:8080/xmlui/bitstream/handle/123456789/10979/chikwature-margaret-curriculum.pdf?sequence=1&isAllowed=y.

[7] L. S. Chong and C. J. Xin. "Creating a GUI Solver for Linear Programming Models in MATLAB." In: *Journal of Science and Technology* 10 (2018).

[8] G. Collaud and J. Pasquier-Boltuck. "gLPS: A graphical tool for the definition and manipulation of linear problems." In: *European Journal of Operational Research* 72.2 (1994), pp. 277–286. issn: 0377-2217. doi: https://doi.org/10.1016/0377-2217(94)90309-3. url: https://www.sciencedirect.com/science/article/pii/0377221794903093.

[9] N. 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.

[10] H. Guerrero. *Excel Data Analysis: Modeling and Simulation*. Springer, 2010. url: https://www.springer.com/gp/book/9783642108341.

[11] F. Hermans. "Analyzing and Visualizing Spreadsheets." In: 2013.

[12] V. Lazaridis, K. Paparrizos, N. Samaras, and A. Sifaleras. "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.

[13] "Linear programming." In: (2017). url: https://www.britannica.com/science/linear-programming-mathematics.

[14] P.-C. Ma, F. H. Murphy, and E. 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.

[15] M. Macarty. *Linear Programming (LP) Optimization with Excel Solver*. url: https://www.youtube.com/watch?v=6xa1x_Iqjzg&ab_channel=MattMacarty.

[16] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond. "The Scratch Programming Language and Environment." In: *ACM Trans. Comput. Educ.* 10.4 (Nov. 2010). doi: 10.1145/1868358.1868363.

[17] J. Mendes, J. Cunha, F. Duarte, G. Engels, J. Saraiva, and S. Sauer. "Systematic spreadsheet construction processes." In: *2017 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2017, Raleigh, NC, USA, October 11-14, 2017*. Ed. by A. Z. Henley, P. Rogers, and A. Sarma. IEEE Computer Society, 2017, pp. 123–127. doi: 10.1109/VLHCC.2017.8103459. url: https://doi.org/10.1109/VLHCC.2017.8103459.

[18] J. Mendes, J. Cunha, F. Duarte, G. Engels, J. Saraiva, and S. Sauer. "Towards systematic spreadsheet construction processes." In: (2017). Ed. by S. Uchitel, A. Orso, and M. P. Robillard, pp. 356–358. doi: 10.1109/ICSE-C.2017.141. url: https://doi.org/10.1109/ICSE-C.2017.141.

[19] J. Parham-Mocello, M. Erwig, and E. Dominguez. "To Code or Not to Code? Programming in Introductory CS Courses." In: *2019 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2019, Memphis, Tennessee, USA, October 14-18, 2019*. Ed. by J. Smith, C. Bogart, J. Good, and S. D. Fleming. IEEE Computer Society, 2019, pp. 187–191. doi: 10.1109/VLHCC.2019.8818909. url: https://doi.org/10.1109/VLHCC.2019.8818909.

[20] E. Pasternak, R. Fenichel, and A. 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.

[21] E. W. Patton, M. Tissenbaum, and F. Harunani. "MIT App Inventor: Objectives, Design, and Development." In: *Computational Thinking Education*. Ed. by S.-C. Kong and H. Abelson. Singapore: Springer Singapore, 2019, pp. 31–49. isbn: 978-981-13-6528-7. doi: 10.1007/978-981-13-6528-7_3.

[22] J. Pereira and S. 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.

[23] D. Saleh and T. Latif. “Solving LProg Problems By Using Excel’s Solver.” In: *Tikrit Journal of Pure Sc.* Vol. 14 (Mar. 2009).

[24] D. Saleh and T. Latif. “Solving LProg Problems By Using Excel’s Solver.” In: *Tikrit Journal of Pure Sc.* Vol. 14 (Mar. 2009).

[25] E. 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`.

Appendix A

LPBlocks study form participant background questions

1. **(Age) How old are you** - In this question we give the participant the option to select one of the following options:
   a) Less than 18
   b) 18-20
   c) 21-24
   d) 25-35
   e) More than 35
   f) Rather not answer
2. **(Gender) What is your gender identity** - In this question we give the participant the option to choose one of the following options:
   a) Male
   b) Female
   c) Other
   d) Rather not answer
3. **(Education degree) Highest degree** - We allow the participant to choose form one of the following options:
   a) Bellow High School
   b) High School

   c) Bachelor
   d) Post-graduation
   e) Master
   f) PhD

4. **(Education time) Years of post-secondary education** - We allow the participant to choose one of the following options:
   a) 0
   b) 1
   c) ...
   d) 9
   e) 10+

5. **(Education field) Main fields of study** - We allow the user to choose multiple of the following options:
   a) Computing (Computer Science, Informatics Engineering, Information systems, Information Technology, etc...)
   b) Life Science(Biology, Chemistry, etc...)
   c) Engineering(Electrical Engineering, Mechanical Engineering, Civil Engineering, etc...)
   d) Business(Economics, Business, Marketing, etc...)
   e) Humanities(History, Languages, etc...)
   f) Mathematics(Mathematics, Statistics)
   g) Other: (fill in option)

6. **(Linear programming experience) What's your academic experience with linear programming and operations research** - We allow the particpants to choose from multiple of the following options:
   a) None
   b) Took college classes
   c) Degree in related field(ex Industrial engineering or statistics)
   d) Professional experience
   e) Other: (fill in otpion)

7. **(Optimization tools) Experience with optimization tools programming and otherwise** - We allow the participants to choose from mutiple of the following options:

a) Programming tools

b) Excel

c) Other visual tools such as GAMS

d) Other: (fill in option)

Appendix B

LPBlocks study form user feedback

1. **(Other tools) If you used any of the tools above how do those compare to ours** - This is a write in question.
2. **(Positive aspects) What were in your opinion the positive aspects of our tool** - This is a write in question.
3. **(Negative aspects) What where in your opinion the negative aspects of our tool** - This is a write in question.

Annex I

Towards a Block-based Language for Linear Programming

Towards a Block-based Language for Linear Programming[⋆]

Hugo da Gião[1,2][0000−0003−3798−0367], Rui Pereira[2][0000−0002−5801−7345], and Jácome Cunha[1,2][0000−0002−4713−3834]

[1] University of Minho
[2] HASLab/INESC TEC
hugo.a.giao@inesctec.pt rui.a.pereira@inesctec.pt jacome@di.uminho.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.
Block-based languages have been successfully used to aid novice programmers and even children in programming. Thus, we propose creating a block-based programming language termed LPBlocks 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. As an initial evaluation we have used LPBlocks to model 7 linear programming problems with success.

Keywords: linear programming, spreadsheets, block-based languages, end-user programming

1 Introduction

The versatility of linear programming in specifying all sorts of problems lends itself useful in many industrial contexts from schedule optimization to route planning. Since many of its users have little to no programming or technical knowledge, visual software such as Microsoft Excel is often the preferred tool when it comes to specifying and solving this type of problem [7].

However, the typical methodologies used when solving those types of problems using spreadsheet software often come with underlying problems such as

[⋆] 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).

relying on an imprecise process to feed data from the spreadsheet to the solver, as well as the difficulties visualizing the models. Many tools commonly used by professionals working with linear programming such as MATLAB[3] and GAMS[4] either require considerable programming knowledge or use *ad-hoc* and error-prone methodologies. Some projects related to the use of GUI's for linear programming in MATLAB are despite the use of graphical interface detached from the business logic and more applied use cases [4].

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 [16,9]. 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 [10,17].

Numerous projects have applied visual languages to various areas of computing generally focused on increasing accessibility of novice and non-technical users as well as teaching. A considerable amount of these languages use the Blockly framework for their implementation [14]. These languages include BlockPy [1], a web-based platform that lets the user write and run Python code using a block-based language, and Scratch [11], a block-based visual programming language and educational tool mostly targeted at children.

Taking into consideration the potential of Blockly to improve the usability and practice of linear programming and the extensive study of block-based languages and their practices [14,5], we aim to build upon the work already done in this field to the create a visual programming language and tool capable of expressing linear programming models in a high level, safe and intuitive manner.

In this work we extend our previous idea of bringing block-based languages and linear programming [6] by presenting in detail LPBlocks, a block-based language written using Blockly that is tailored for the construction of linear programming models. To illustrate the language's applicability we present its use to model several linear programming problems demonstrating its use and fallbacks.

2 Related work

Blockly is a framework that has been used over the last decade in many projects aiming to improve programming accessibility, in diverse areas going from teaching core concepts [11] to data science [1], robotics [8] and app development [15].

One such project, BlockPy, lets users access several data science libraries and features, and generates the associated Python code. Additionally, it runs inside a browser using a Javascript Python interpreter [1]. This project was shown to be particularly useful in helping introductory level computer science students transition into having full-fledged ideas, as well as offering the students a compelling context for learning programming.

[3] `https://www.mathworks.com/products/matlab.html`

[4] `https://www.gams.com/`

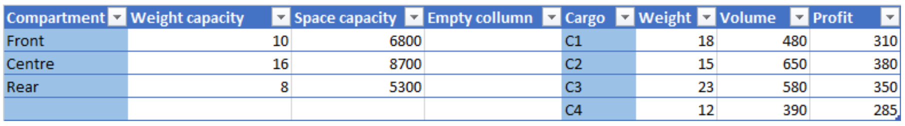
Another such project is the MitApp inventor, a platform that allows users to create full-fledged mobile applications with various graphical interfaces [15]. When using this platform one is able to work on the design of the applications using a graphical user interface, and on the program's logic using a block-based programming language.

Some works tackling various issues associated with the teaching and practice of linear programming using visual programming also exist. One such work presents a tool named GLP-tool which allows users to define two variable linear programming problems using an algebraic language and visualize the solving process graphically [16]. Another relevant work introduces the LPFORM software, which allows managers and operations researchers to formulate large linear programming programs [10]. LPFORM empowers users during the more repetitive steps of building these models by allowing linear programming problems to be represented using objects and relationships, and uses knowledge-based techniques to generate the input given to the solver.

There are many solutions in the market catering to users with different needs and levels of experience although we found that Excel was popular amongst less technical users [7]. The majority of other tooling such as Matlab, GAMS and SAS/OR[5] cater primarily to users with some programming or mathematical knowledge. Some of the problems associated with these more advanced tools relate primarily to the technical complexity of modeling linear programming knowledge, as well as the need to use textual programming languages. When using Excel for modeling users face problems associated with the use of spreadsheet software.

3 A block-based language for linear programming

In this section, we introduce our proposed language, LPBlocks, using an example featured in a Master of Business Administration (MBA) exam [2]. This example problem aims to increase the profit of deliveries by airplanes. The problem statement provides values for the weight and space capacity of three different airplain's 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 data for the running example problem

[5] `https://www.sas.com/en_us/software/or.html`

3.1 Input data specification

Our solution requires the input data to follow a specific structure. This structure allows for the definition of **index columns** (as seen highlighted in blue in Figure 1), this are used to reference values and iterate over the **data columns** (in white the same figure) this being always associated with one index column. 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 **index column**, and that different sets of **index** and **data columns** are separated by an empty column as can be seen in the figure (fourth column). In this case there are two sets, the first being for the three plane compartments and the second for the four types of cargo.

3.2 LPBlocks constructs

The building blocks of the linear programming language we propose can be seen in Figure 2. LPBlocks includes:

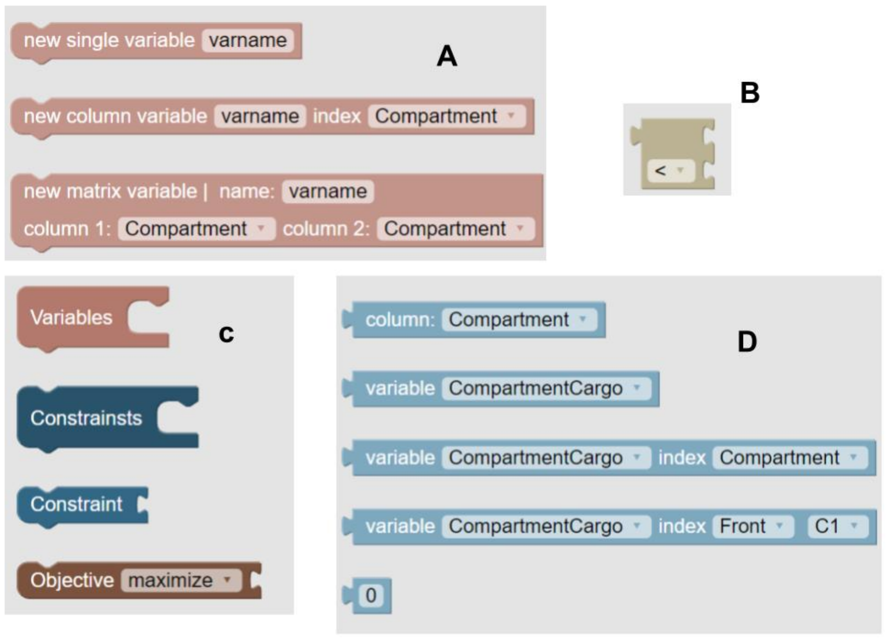


Fig. 2. Building blocks for LPBlocks

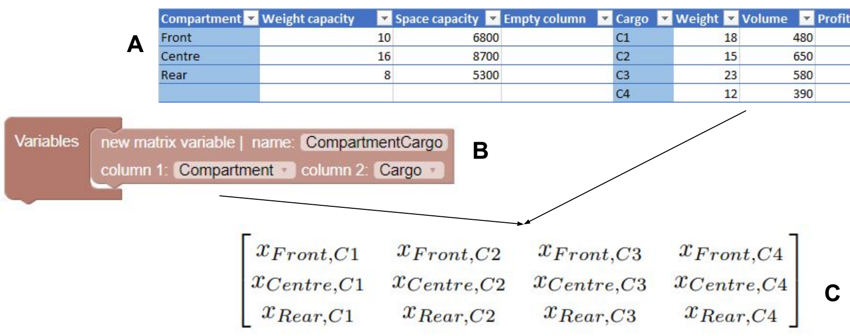
– **Variable blocks** (seen in Figure 2.A): Blocks for creating single, column and matrix variables.

- **Operation block** (seen in Figure 2.B): A block to construct an individual constraint.
- **Building blocks** (seen in Figure 2.C): These blocks include two nesting blocks for the `Variables` and `Constraints` a nesting block to add an individual `Constraint` to a `Constraints` block and an `Objective` block to define the objective function.
- **Value blocks** (seen in Figure 2.D): A set of blocks to access the variables created before, that is, values blocks.

To further facilitate this process for novice and inexperienced users, when building a new linear programming model, the variables, constraints and objective blocks already appear and are connected in the workplace when creating a fresh solution. In the next sections we detail each of the blocks.

3.3 Defining variables

To define a mathematical linear programming model for our running example one would start by creating a set of variables iterating over the the airplane sections and the cargoes as shown in Figure 3.C (we refer to the problem's original website for a more common variable naming). Since this is a very common scenario LPBlocks includes a construct that can be used to define all these variables which we call a matrix. In Figure 3.B we use such a construct to create the variables for the running example. In the example we use a the matrix variable block to create a new $N \times M$ matrix variable named `CompartmentCargo`, with N being equal to the length of the column `Compartment` and M to the length of the column `Cargo` with these columns serving as its indexes.



Compartment	Weight capacity	Space capacity	Empty column	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

Variables
new matrix variable | name: CompartmentCargo
column 1: Compartment column 2: Cargo

$$\begin{bmatrix} x_{Front,C1} & x_{Front,C2} & x_{Front,C3} & x_{Front,C4} \\ x_{Centre,C1} & x_{Centre,C2} & x_{Centre,C3} & x_{Centre,C4} \\ x_{Rear,C1} & x_{Rear,C2} & x_{Rear,C3} & x_{Rear,C4} \end{bmatrix}$$

Fig. 3. Create variables

LPBlocks offers several options to define new variables, using the blocks seen in Figure 2.A:

- single variables through its name;

– column variables defining its name and an index column for which the variable will be iterated and accessed;
– matrix variables that take a name and two index columns for which they can be iterated and those values accessed (used in Figure 3.B).

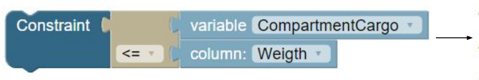
The process of generating the model variables is dependent on the variables block used:

– For the single variable block a variable is generated with the chosen name.
– For column variable blocks an array of variables is created.
– For matrix variables blocks a matrix of variables is created(as shown in Figure 3.B).

3.4 Defining constraints

The second step is to define the mathematical model would be to create a set of constraints, using the variables created before, and encoding the restrictions of the underlying problem. A constraint of the running example is that one "cannot pack more of each of the four cargoes than their available quantity". The mathematical encoding would be as shown in Figure 4.B. There are four constraints, one for each cargo. In each constraint, on the left-hand side of the inequality one should sum the variables referring to the corresponding cargo (e.g. C1 for the first constraint) and for the three different airplane sections. On the right-hand side one would write the cargo weight limit.

A



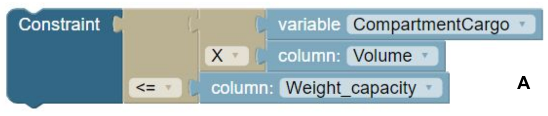
B

$$\begin{aligned}
x_{Front,C1} + x_{Centre,C1} + x_{Rear,C1} &<= 18\\
x_{Front,C2} + x_{Centre,C2} + x_{Rear,C2} &<= 15\\
x_{Front,C3} + x_{Centre,C3} + x_{Rear,C3} &<= 23\\
x_{Front,C4} + x_{Centre,C4} + x_{Rear,C4} &<= 12
\end{aligned}$$

Fig. 4. Defining constraints for the cargoes weight

In LPBlocks, each constraint is defined by dragging a constraint block inside the constraints block (second and third blocks from the top in Figure 2.C) and then using the value blocks (blocks in Figure 2.D) and operation blocks (blocks following the constraint block in Figure 2.B) to express the constraints. In our language, operation blocks represent relations between blocks and are used to express several operations including arithmetic operations and inequalities. The value blocks can represent:

– Columns;
– Previously defined variables;
– Numbers.



A

B
$$480x_{Front,C1} + 650x_{Front,C2} + 580x_{Front,C3} + 390x_{Front,C4} <= 6800$$
$$480x_{Centre,C1} + 650x_{Centre,C2} + 580x_{Centre,C3} + 390x_{Centre,C4} <= 8700$$
$$480x_{Rear,C1} + 650x_{Rear,C2} + 580x_{Rear,C3} + 390x_{Rear,C4} <= 5300$$

Fig. 5. Defining constraints - second example

The variables can be accessed using different blocks and options. This as well as the way the columns are used 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 4.A to generate multiple constraints or use the three slot variable blocks to access a particular value of the given variable.

Our solution possesses other features such as using the positioning and index columns of the variables and columns used in the constraint construction automatically deducing summations, sumproducts and sets of linear programming constraints from the high level user defined visual constraints.

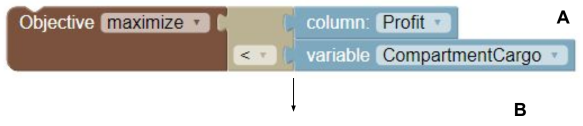
The first constraint in Figure 4.A is defined in our language by using: *i)* an `operation block` with the inequality sign <=; *ii)* a `variable block` with the option `CompartmentCargo` and; *iii)* a column block with the option `Weight`. Since the `constraints block` only appears after the `variables 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 4.B.

Another example constraint can be expressed in natural language as "the volume (space) capacity of each compartment must be respected". This constraint (in Figure 5.A) uses `X` (multiplication) and <= `operation blocks` and `value blocks` to express the more complex constraints. This constraint differs from the previous ones since the use of the `X operation block` leads to the generation of sumproduct constraints instead of sum. For this constraint, our compiler generates the linear programming constraints featured in Figure 5.B.

3.5 Defining the objective function

The final step in a linear programming model is the definition of an objective function. For our running example, one intends to maximize the profit of the airplane usage.

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.



$$310[x_{Front,C1} + x_{Centre,C1} + x_{Rear,C1}] + 380[x_{Front,C2} + x_{Centre,C2} + x_{Rear,C2}] +$$

$$350[x_{Front,C3} + x_{Centre,C3} + x_{Rear,C3}] + 285[x_{Front,C4} + x_{Centre,C4} + x_{Rear,C4}]$$

Fig. 6. Objective function for the running example

In the example seen in Figure 6.A 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 6.B which would be the one written in a mathematical model.

3.6 Implementation

This language is included in a previously existing software prototype developed in the context of a work exploring the creation of systematic spreadsheet processes using a process-like notation [12,13]. This tool currently supports some of the operations associated with creating and editing spreadsheets such as adding columns, sorting, filtering or and creating charts. Our goal is to extend the current software with functionalities that would allow users to build correct and robust linear programming models.

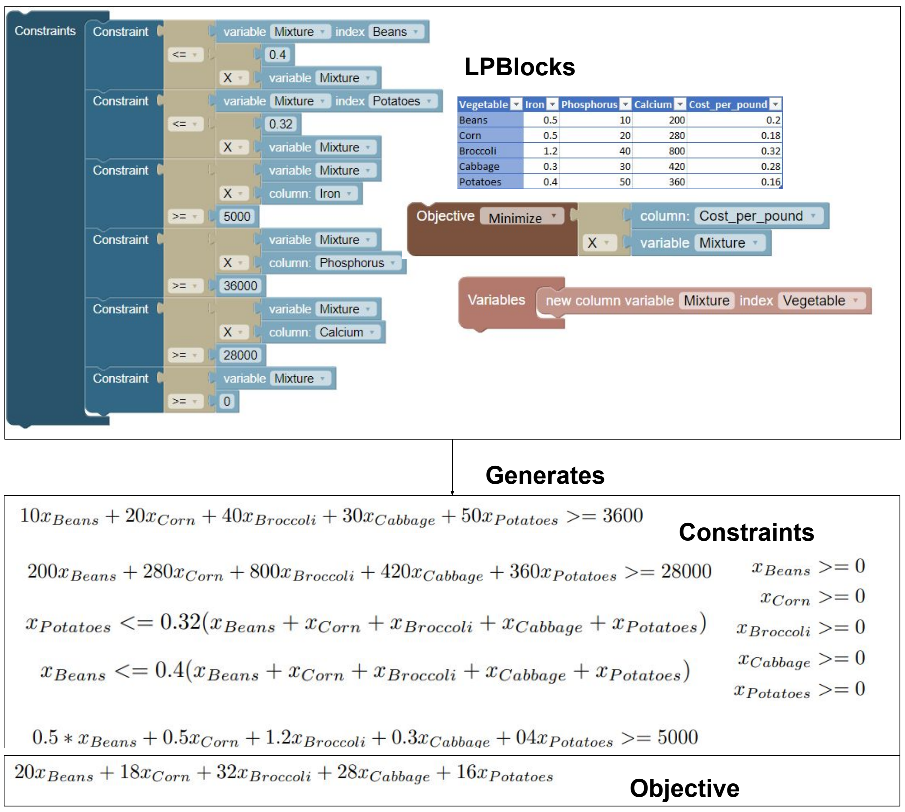
We use the Blockly framework to define our language syntax. This is then compiled into an OR-Tools valid linear programming model.

4 Language applicability

In this section we present a set of linear programming problems taken from an MBA exam [2] and from a Operations Research textbook [3] and modeled using LPBlocks. With this we intend to illustrate the applicability of our language to a broad set of examples. In Appendix A we include the solution in LPBlocks of other problems.

4.1 Vegetable mixture

This example shown in Figure 7 comes from the operations research textbook [3]. In this example a manufacturer of freeze-dried vegetables aims at reducing production costs while adhering to various nutrition criteria and guidelines. We are given nutritional data for each of the vegetables as well as their cost per



```
Constraints
  Constraint  variable Mixture index Beans  <=  0.4  X  variable Mixture
  Constraint  variable Mixture index Potatoes  <=  0.32  X  variable Mixture
  Constraint  variable Mixture  X  column: Iron  >=  5000
  Constraint  variable Mixture  X  column: Phosphorus  >=  36000
  Constraint  variable Mixture  X  column: Calcium  >=  28000
  Constraint  variable Mixture  >=  0

Objective  Minimize  column: Cost_per_pound  X  variable Mixture

Variables  new column variable Mixture index Vegetable
```

Vegetable	Iron	Phosphorus	Calcium	Cost_per_pound
Beans	0.5	10	200	0.2
Corn	0.5	20	280	0.18
Broccoli	1.2	40	800	0.32
Cabbage	0.3	30	420	0.28
Potatoes	0.4	50	360	0.16

$$10x_{Beans} + 20x_{Corn} + 40x_{Broccoli} + 30x_{Cabbage} + 50x_{Potatoes} >= 3600$$

$$200x_{Beans} + 280x_{Corn} + 800x_{Broccoli} + 420x_{Cabbage} + 360x_{Potatoes} >= 28000$$

$$x_{Potatoes} <= 0.32(x_{Beans} + x_{Corn} + x_{Broccoli} + x_{Cabbage} + x_{Potatoes})$$

$$x_{Beans} <= 0.4(x_{Beans} + x_{Corn} + x_{Broccoli} + x_{Cabbage} + x_{Potatoes})$$

$$0.5 * x_{Beans} + 0.5x_{Corn} + 1.2x_{Broccoli} + 0.3x_{Cabbage} + 04x_{Potatoes} >= 5000$$

$$x_{Beans} >= 0 \quad x_{Corn} >= 0 \quad x_{Broccoli} >= 0 \quad x_{Cabbage} >= 0 \quad x_{Potatoes} >= 0$$

$$20x_{Beans} + 18x_{Corn} + 32x_{Broccoli} + 28x_{Cabbage} + 16x_{Potatoes}$$

Fig. 7. Definition of the vegetable mixture problem using LPBlocks

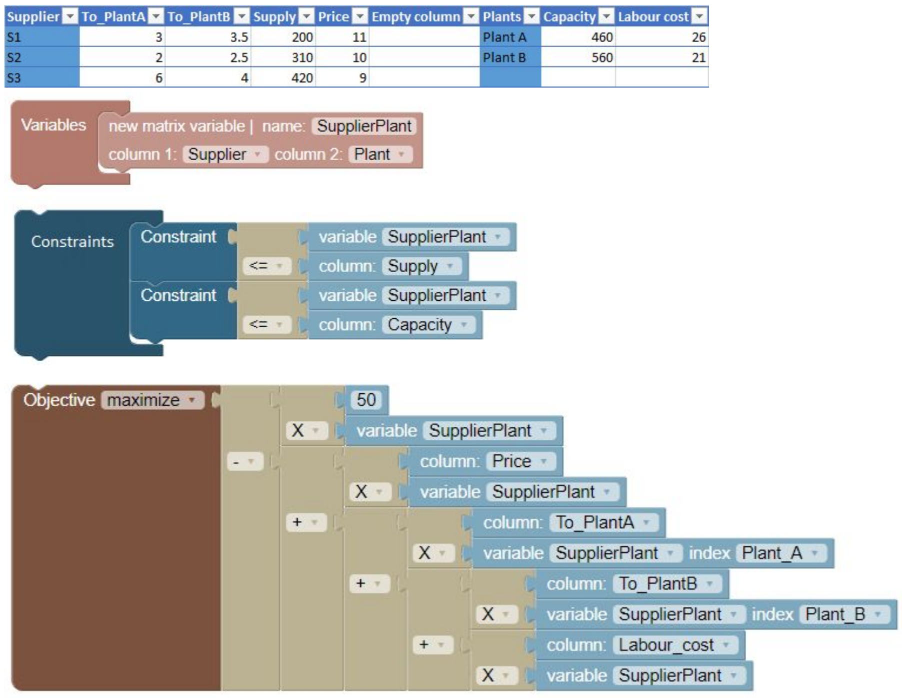
pound in the tabular data. We have a maximum percentage for certain vegetables and the lower bounds for certain nutrients.

Since our goal is to find the ratio of each vegetable that goes into the mixture, we created a `column variable` named `Mixture` that takes as its input the column `Vegetable`. For the constraints we start by creating constraints imposing limits of 40% for Beans and 32% for Potatoes. The following three constraints define the lower bounds for the given nutrients and the last adds non-negativity. The objective is to minimize the cost per pound of the mixture. The verbosity of this model could be improved if support for inputing data in the form of matrices in the spreadsheet was added. This would decrease the necessity of using single value blocks since iteration through both indexes would be possible.

When compared with implementing this model using mathematical notation LPBlocks allows for the creation of a more concise and intuitive model. We feel that our language lends itself to this sort of problem since it allows for the generation of sizable mathematical constraints using more concise business logic.

4.2 Fruit canning plants

In this example taken from an MBA exam [2] and shown in Figure 8 we are given information associated with different suppliers and fruit canning plants with the goal of maximizing its profits. The information includes shipping, labor and operating costs, buying prices and maximum production capacities. Despite not being in the spreadsheet, the problem definition states that the selling price for each tonne is of $50.



Supplier	To_PlantA	To_PlantB	Supply	Price	Empty column	Plants	Capacity	Labour cost
S1	3	3.5	200	11		Plant A	460	26
S2	2	2.5	310	10		Plant B	560	21
S3	6	4	420	9				

Fig. 8. Fruit canning plant example modeled using LPBlocks

To generate the formulation we create a `matrix variable` with indexes `Supplier` and `Plant`. The constraints for this problem are straightforward and can be generically specified, this consisting of the upper bounds for the supply and capacity for each of the plants. The objective function is considerably more complex since it needs to take into account the selling price and all the costs to represent the profit. The objective verbosity could be mitigated by adding the support for defining matrices inside the spreadsheet for the data. To improve the reusability of the created models adding support to define data variables inside the spreadsheet could also be done. To help users better visualize the model we will add support for intermediate values to store portions of more verbose components. More concretely in this model we could create blocks for the different operating costs used in the objective and reference these blocks at the moment of its definition.

When comparing with traditional mathematical notation or even other solutions LPBlocks allowed for considerable savings in terms of syntax for the expression of constraints however we found that due to its complexity the objective function specification in LPBlocks was similar to the mathematical form.

4.3 Machine allocation

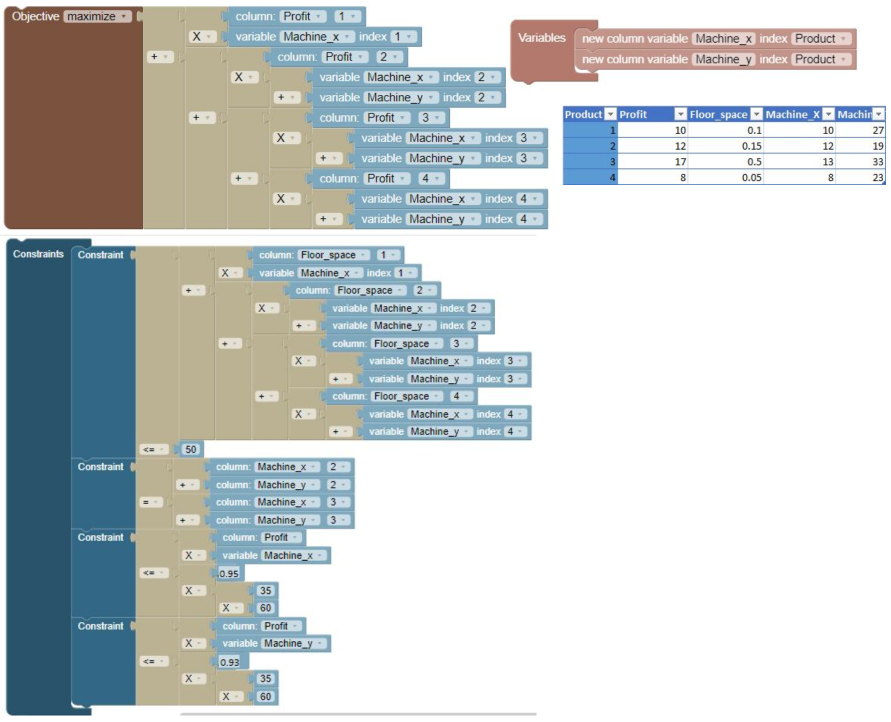
In the problem shown in Figure 9 (taken from [2]) the goal is to maximize a factory's profit by allocating the production of different goods among two machines. In this problem we are given information about each product's profitability, use of floor space and manufacturing time in minutes taken by each machine. We are also given other rolls related to the machines down time, the total floor space of $50m^2$, the time of a work week of 35 hours, the ratios of which some products have to be produced relatively to others and that `Product` 1 can only be manufactured in the second machine.

In this example we create `column variables` for each machine taking the `Proudct` column as the index as opposed to previous examples where we created `matrix variables`. In this we did not create a `matrix variable` as could be assumed do to the fact that the values for the machines are not used as index columns and using them for defining a `matrix variable` would mandate the referencing of one of the values for every use of the variable and would offer nothing in terms of iterability and generalization. In terms of constraints we use the first constraint to express that the maximum floor space use is of $50m^2$. If LPBlocks supported matrices as data input we possibly could express this constraint in a less verbose manner. In the second constraint we express that the production of product 2 is the same as 3. In constraints three and four we take into account the downtime of 5% for machine 1 and 7% for 2 by modeling that the total running time of each machine must be lower or equal to 95% and 93% of the total work week. The objective aims to maximize the profit and takes into account that `Product` 1 can only be manufactured in the second machine. This is the reason we were not able to use a more generic representation for this constraint.

Similarly to the previous example we found some benefits in terms of smaller footprint but since the creation of the model required the use of complex mathematical operations it couldn't make use of most of LPBlocks features.

4.4 Improving LPBlocks

We have specified 7 linear programming models using LPBlocks. Most of these examples were easily encoded using LPBlocks. However, we found that some improvements could be made in terms of usability, this center mainly on providing support for receiving data in the form of matrices and variables. In testing the


Objective maximize
column: Profit 1
X variable Machine_x index 1
+ column: Profit 2
X variable Machine_x index 2
+ variable Machine_y index 2
+ column: Profit 3
X variable Machine_x index 3
+ variable Machine_y index 3
+ column: Profit 4
X variable Machine_x index 4
+ variable Machine_y index 4
Variables
new column variable Machine_x index Product
new column variable Machine_y index Product
Constraints
Constraint
column: Floor_space 1
X variable Machine_x index 1
+ column: Floor_space 2
X variable Machine_x index 2
+ variable Machine_y index 2
+ column: Floor_space 3
X variable Machine_x index 3
+ variable Machine_y index 3
+ column: Floor_space 4
X variable Machine_x index 4
+ variable Machine_y index 4
<= 50
Constraint
column: Machine_x 2
+ column: Machine_y 2
= column: Machine_x 3
+ column: Machine_y 3
Constraint
column: Profit
X variable Machine_x
<= 0.95
X 35
X 60
Constraint
column: Profit
X variable Machine_y
<= 0.93
X 35
X 60

Product	Profit	Floor_space	Machine_X	Machin
1	10	0.1	10	27
2	12	0.15	12	19
3	17	0.5	13	33
4	8	0.05	8	23

Fig. 9. Machine allocation problem in LPBlocks

language applicability the value of these changes was evident. Despite posing some challenges in terms of parsing and changes in the context of our application, this addition would greatly benefit the writing of less verbose and more concise programs in LPBlocks. Other features such as using intermediate blocks could add some benefit to users but come with possible negative impacts when it comes to ease of use and the creation of correct models.

5 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 and it possesses numerous advantages associated with Block-based such as having and forcing strict input, output , next and previous statement data types and having other features such as imposing the selection of various inputs from lists extracted from input spreadsheet. 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.

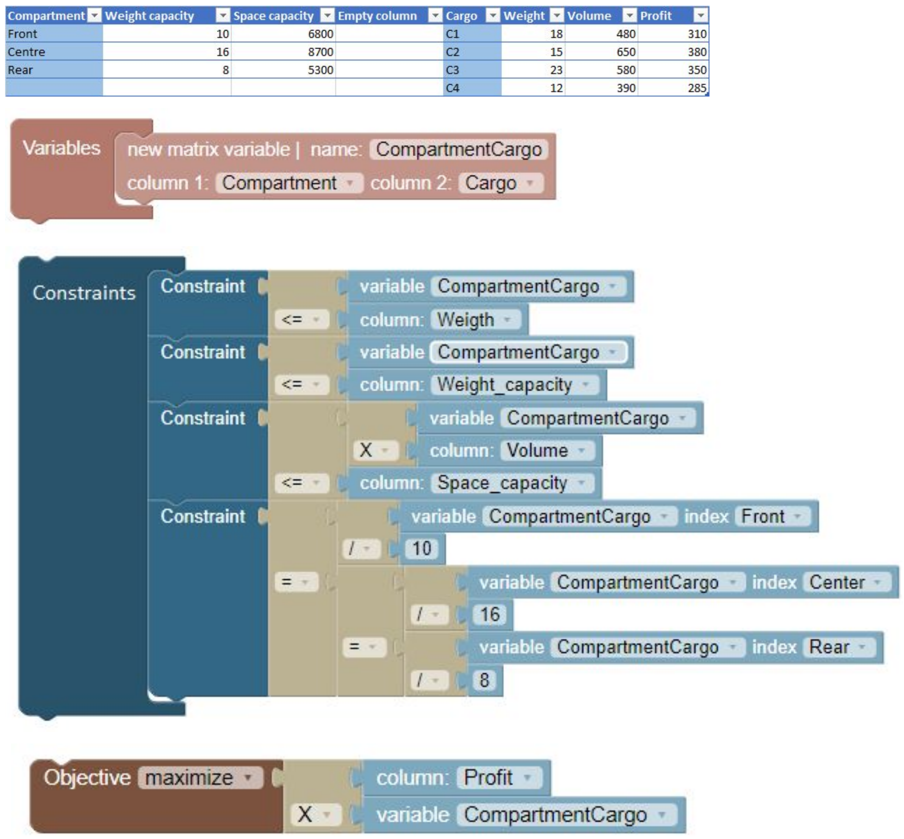
References

1. Bart, A.C., Tibau, J., Tilevich, E., Shaffer, C.A., Kafura, D.: Blockpy: An open access data-science environment for introductory programmers. Computer **50**(5), 18–26 (2017). https://doi.org/10.1109/MC.2017.132
2. Beasley, J.E.: Or-notes, `http://people.brunel.ac.uk/~mastjjb/jeb/or/lpmore.html`
3. Carter, M., Price, C.C.: Operations Research: A Practical Introduction. CRC Press (2000)
4. Chong, L.S., Xin, C.J.: Creating a gui solver for linear programming models in matlab. Journal of Science and Technology **10** (2018)
5. Fraser, N.: Ten things we've learned from blockly. In: 2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond). pp. 49–50 (2015). https://doi.org/10.1109/BLOCKS.2015.7369000
6. da Gião, H., Cunha, J., Pereira, R.: Linear programming meets block-based languages. In: 2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC) (2021), to appear.
7. Guerrero, H.: Excel Data Analysis: Modeling and Simulation. Springer (2010), `https://www.springer.com/gp/book/9783642108341`
8. Krishnamoorthy, S.P., Kapila, V.: Using a visual programming environment and custom robots to learn c programming and k-12 stem concepts. In: Proceedings of the 6th Annual Conference on Creativity and Fabrication in Education. p. 41–48. FabLearn '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/3003397.3003403
9. Lazaridis, V., Paparrizos, K., Samaras, N., Sifaleras, A.: Visual linprog: A web-based educational software for linear programming. Comput. Appl. Eng. Educ. **15**(1), 1–14 (2007). https://doi.org/10.1002/cae.20084
10. Ma, P.C., Murphy, F.H., Stohr, E.A.: A graphics interface for linear programming. Commun. ACM **32**(8), 996–1012 (Aug 1989). https://doi.org/10.1145/65971.65978
11. Maloney, J., Resnick, M., Rusk, N., Silverman, B., Eastmond, E.: The scratch programming language and environment. ACM Trans. Comput. Educ. **10**(4) (Nov 2010). https://doi.org/10.1145/1868358.1868363
12. Mendes, J., Cunha, J., Duarte, F., Engels, G., Saraiva, J., Sauer, S.: Towards systematic spreadsheet construction processes. In: 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C). pp. 356–358 (2017). https://doi.org/10.1109/ICSE-C.2017.141
13. Mendes, J., Cunha, J., Duarte, F., Engels, G., Saraiva, J., Sauer, S.: Systematic spreadsheet construction processes. In: 2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). pp. 123–127 (2017). https://doi.org/10.1109/VLHCC.2017.8103459
14. Pasternak, E., Fenichel, R., Marshall, A.N.: Tips for creating a block language with blockly. In: 2017 IEEE Blocks and Beyond Workshop (B B). pp. 21–24 (2017). https://doi.org/10.1109/BLOCKS.2017.8120404
15. Patton, E.W., Tissenbaum, M., Harunani, F.: MIT App Inventor: Objectives, Design, and Development, pp. 31–49. Springer Singapore, Singapore (2019). https://doi.org/10.1007/978-981-13-6528-7_3
16. Pereira, J., Fernandes, S.: Two-variable linear programming: A graphical tool with mathematica. In: SYMCOMP 2013 - 1st International Conference on Algebraic and Symbolic Computation. pp. 159–173 (09 2013)

17. Senne, E., Lucas, C., Taylor, S.: Towards an intelligent graphical interface for linear programming modelling. Journal of Intelligent Systems **6**(1), 63–94 (1996). https://doi.org/doi:10.1515/JISYS.1996.6.1.63

A Examples of the use of LPBlocks

A.1 Cargo allocation



Compartment	Weight capacity	Space capacity	Empty column	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

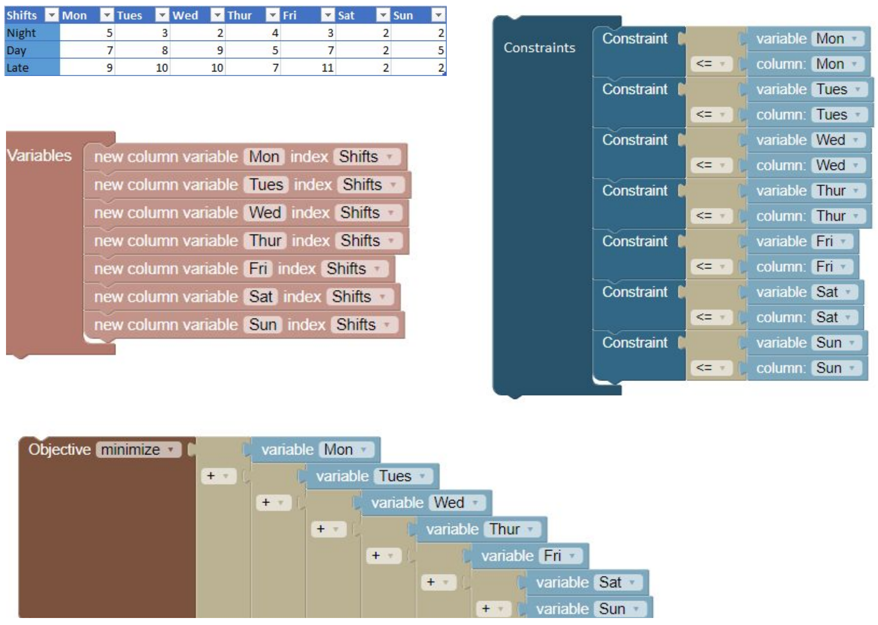
Variables
new matrix variable | name: CompartmentCargo
column 1: Compartment column 2: Cargo

Constraints
Constraint: variable CompartmentCargo <= column: Weigth
Constraint: variable CompartmentCargo <= column: Weight_capacity
Constraint: variable CompartmentCargo X column: Volume <= column: Space_capacity
Constraint: variable CompartmentCargo index Front / 10 = variable CompartmentCargo index Center / 16 = variable CompartmentCargo index Rear / 8

Objective maximize column: Profit X variable CompartmentCargo

Fig. 10. Cargo allocation

A.2 Shift allocation



Shifts	Mon	Tues	Wed	Thur	Fri	Sat	Sun
Night	5	3	2	4	3	2	2
Day	7	8	9	5	7	2	5
Late	9	10	10	7	11	2	2

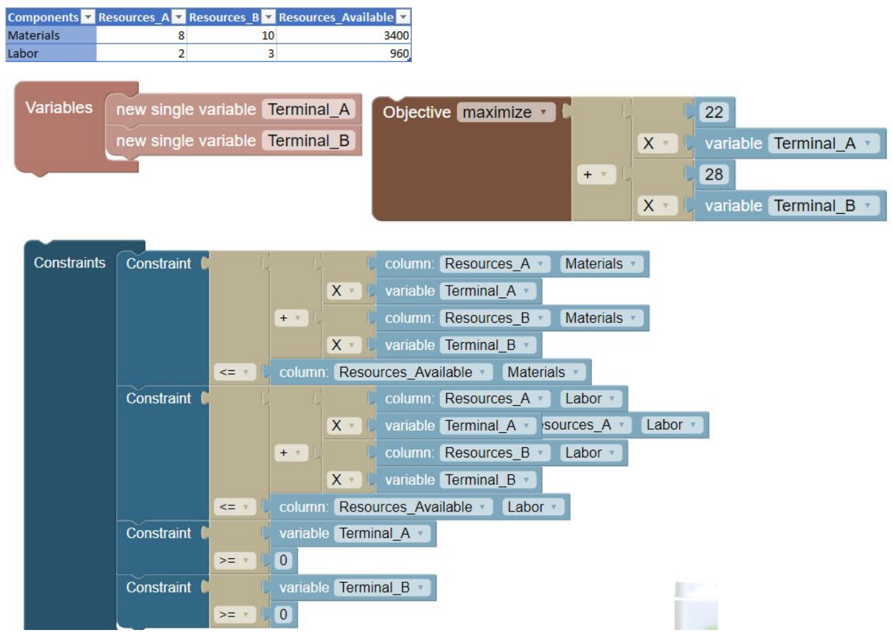
```
Variables
  new column variable Mon index Shifts
  new column variable Tues index Shifts
  new column variable Wed index Shifts
  new column variable Thur index Shifts
  new column variable Fri index Shifts
  new column variable Sat index Shifts
  new column variable Sun index Shifts

Constraints
  Constraint  variable Mon   <=  column: Mon
  Constraint  variable Tues  <=  column: Tues
  Constraint  variable Wed   <=  column: Wed
  Constraint  variable Thur  <=  column: Thur
  Constraint  variable Fri   <=  column: Fri
  Constraint  variable Sat   <=  column: Sat
  Constraint  variable Sun   <=  column: Sun

Objective minimize
  variable Mon
  + variable Tues
    + variable Wed
      + variable Thur
        + variable Fri
          + variable Sat
            + variable Sun
```

Fig. 11. Shift allocation problem

A.3 Terminal manufacture



Components	Resources_A	Resources_B	Resources_Available
Materials	8	10	3400
Labor	2	3	960

```
Variables   new single variable Terminal_A
            new single variable Terminal_B

Objective maximize
      +   22
          X   variable Terminal_A
          28
          X   variable Terminal_B

Constraints
  Constraint
          column: Resources_A Materials
      X   variable Terminal_A
      +   column: Resources_B Materials
      X   variable Terminal_B
      <=  column: Resources_Available Materials
  Constraint
          column: Resources_A Labor
      X   variable Terminal_A   sources_A   Labor
      +   column: Resources_B Labor
      X   variable Terminal_B
      <=  column: Resources_Available Labor
  Constraint
          variable Terminal_A
      >=  0
  Constraint
          variable Terminal_B
      >=  0
```

Fig. 12. Terminal manufacture

A.4 Satellite launching

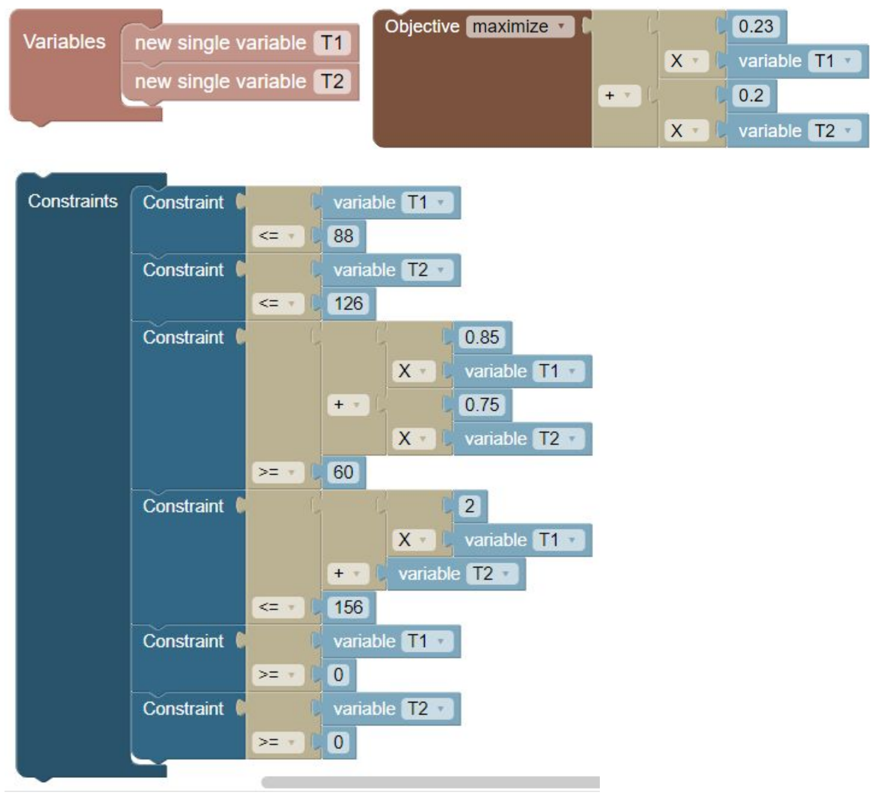


Fig. 13. Satellite launching

Annex II

Linear Programming Meets Block-based Languages

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].

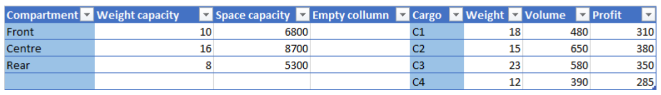
This work is supported by the national funds through the Portuguese Funding Agency (FCT - Fundação para a Ciência e a Tecnologia, within project UIDB/50014/2020).

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.

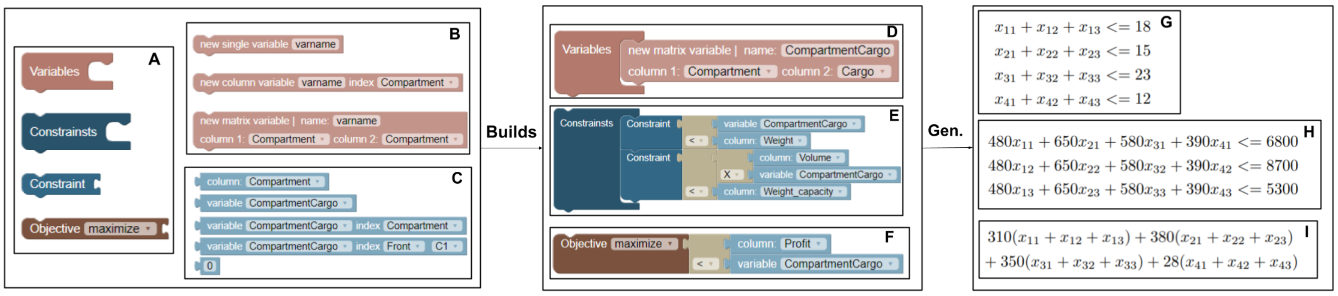

A
Variables
Constrainsts
Constraint
Objective maximize
B
new single variable varname
new column variable varname index Compartment
new matrix variable | name: varname column 1: Compartment column 2: Compartment
C
column: Compartment
variable CompartmentCargo
variable CompartmentCargo index Compartment
variable CompartmentCargo index Front C1
0
Builds
D
Variables new matrix variable | name: CompartmentCargo column 1: Compartment column 2: Cargo
E
Constrainsts Constraint variable CompartmentCargo < column: Weight
Constraint column: Volume X 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.