Interactive didactical utility for dynamic routing using LP solve

Tjaša Jereb, Matej Rabzelj, Aljaž Martinčič, Andrej Košir

Faculty of Electrical Engineering
University of Ljubljana
Tržaška cesta 25, 1000 Ljubljana
E-mail: tjasa.jereb@svet.fe.uni-lj.si, matej@rabzelj.si, aljaz.martincic@ltfe.org, andrej.kosir@fe.uni-lj.si

Abstract

A good visual representation is often the key element to understand a linear optimisation problem. There are some good options for solving linear problems, but they lack a good user interface that would enable users to better understand what they're calculating. This paper presents a didactical utility for making formulation of optimization problems easier and less complex through a dynamic graphical interface. It describes the problem of linear optimization and its solution using LP_Solve solver [3], which is one of the tools for linear program solving. Furthermore it explains the logic of utility itself and it's use.

1 Introduction

Today we can find a software to help us solve a wide variety of problem types. The difficulty is that sometimes the most versatile tools require pre-knowledge and a certain set of skills for their use and don't provide a graphical representation, either for easier input and/or easier understanding of the output. This can cause problems to various groups of users, ranging from professionals to those who are new in the field. The teaching process of linear programming is nowadays based on explaining mathematical foundations with drawing examples by hand. Those sketches can quickly become over saturated with information, and drawing new graphs over and over again consumes a lot of valuable time during lectures. Students will often use one of the tools like LP Solve, which doesn't provide adequate demonstration of the network for studying. For this purpose - especially with the maximum flow problem [7], which is one of the most used in telecommunication networks and optimization - visualisation and ability to solve more examples in less time, is very beneficial. Hereby both, students and professors could use a tool that would ensure optimization of teaching and learning. The goal of this paper is to present the utility for making a formulation of optimization problem easier and less complex through a dynamic graphical interface as a supportive tool for hands-on learning.

2 Theoretical background

In this section the problem and method of solving maximum flow in telecommunications are presented. In theory, maximum flow problem means finding a feasible maximum flow through a flow network. The best approach to solving this problem is using a linear programming method called linear optimization.

2.1 Linear programming

Linear optimization is a linear programming method with the purpose of finding the best possible solution for a given mathematical model, whose cost (objective) function and requirements are linear [4]. Best possible solution is defined by lowest or highest cost (depends on the task being solved), shortest path or any other chosen metric between two nodes.

For solving a linear optimization problem, two key inputs are needed. The first one is the objective function, which is the function that needs to be minimized or maximized. The second one is a set of constraints representing limitations within which the solution to the problem can be found. Like the objective function, constraints also have to be linear and can be given in the form of equalities or inequalities.

Linear problem's feasible region - that is a set of points, that suits all constraints - has a shape of a convex polytope, which is formed as an intersection of half-spaces [6]. All linear problems can be expressed in canonical form:

$$\begin{aligned} & \text{argmax } \mathbf{c}^T.\mathbf{x} \\ s.t. \quad & \\ & \mathbf{A}.\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned} \tag{1}$$

where $\mathbf{c}^T.\mathbf{x}$ represents the objective function (function to be minimized or maximized [1]), $\mathbf{x}$ is vector of variables, that are to be determined, $\mathbf{b}$ and $\mathbf{c}$ are vectors of known coefficients and $\mathbf{A}$ is a matrix of known coefficients.

Aforementioned arguments with non-negative variables also specified are enough for standard form of a linear problem. Additionally, a standard form can be converted into an augmented form, where slack variables are added in constraints to replace inequalities with equalities. By incorporating slack variables into the model, the following rule applies:

$$x < a \quad \Leftrightarrow \quad x + x_s = a \;\&\; x_s > 0. \tag{2}$$

With this formulation linear problems can be written in block matrix form:

$$\begin{bmatrix} 1 & -\mathbf{c}^T & 0 \\ 0 & \mathbf{A} & \mathbf{I} \end{bmatrix} \cdot \begin{bmatrix} z \\ \mathbf{x} \\ \mathbf{s} \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{b} \end{bmatrix} \tag{3}$$

where additional vector **s** contains all slack variables and a variable z, that is to be maximized or minimized.

2.2 Dynamic routing and maximum flow problem in telecommunications

Dynamic routing is a process of providing optimal route through the given network of nodes based on existing conditions within the given system. Optimal route can be defined by maximum capacity, fastest path or other given parameters specified by the programmer [5]. Dynamic route can change for every other request while static routing route remains the same for all the requests. In telecommunications, dynamic routing is most frequently associated with routing protocols such as RIP, OSPF, BGP etc.

To define the problem of calculating maximum flow based on an input parameters and limitations, we conducted tests using various network models in the LP_solve environment assigning them either costs, capacities or both. With that done, we can create the objective function and then the set of constraints, that contains Kirchhoff equations for every node in the network model.

However, when typing linear equations of a model into the IDE, the bigger the model in terms of numbers of connections and nodes, the more time consuming it gets while rising the chance for an input error resulting in false results.

Another problem that occurs when calculating different models is the complexity of correcting a network model. Just removing or adding one node or one connection can require rewriting the entire model, as all connections are intertwined in Kirchhoff equations for all nodes.

To make the creation of a bigger network model and correction of an existing one easier, we created an application (utility), that accepts a graph of a network and calculates the maximum network flow depending on bandwidths of connections and capacities of the nodes. The application returns the result in the form of a colored graph, to clearly indicate the result.

2.3 Hands-on learning

Hands-on learning is a form of experiential learning, which is defined as a process of learning through experience. An important aspect of this experience is given - quoted: "Learning is best conceived as a process, not in terms of outcomes. To improve learning in higher education, the primary focus should be on engaging students in a process that best enhances their learning – a process that includes feedback on the effectiveness of their learning efforts.", see [2]. As such, it complements the theoretical background students can learn through classic learning forms. However, high quality hands-on learning requires additional teaching resources in terms of quantity and in terms of resource types. Among others, it requires a significant amount of teachers time invested in preparation of materials and tools supporting this type of experience learning. In this regard, a small case construction and visualization tool such as presented in this paper, can enable higher quality of hands-on learning in the area of optimization and telecommunication networks.

3 Didactic utility

In this section the principle, working and use of a didactic utility is explained. The use of classical LP solve tools is rather complex for a beginner programmer or a student. It requires the knowledge of syntax and underlying mathematics while providing no graphical cue. To make the process of learning easier and less complicated, didactical utility provides an intuitive graphical interface, which makes the solving of first LP problems easy and beneficial.

3.1 LP solve

Independently of the problem's form, several tools can be used to get the needed results. One such tool is the LP_solve, a free linear programming solver [3]. It is based on the Dantzig's simplex method, and it solves pure linear, mixed integer or binary, semi-continuous and special ordered sets models [6]. Several library implementations exist for use with numerous programming libraries. Additionally, as it is written in ANSI C, it can be compiled on different platforms between Linux and Windows. There are three ways to pass data to the library:

- through API calls,
- in form of input files,
- by use of IDE.

LP_solve tool API is a set of routines that can be called from several programming language including Python and can be used to build the model in memory, solve it and return the result. When using input files, any of the supported types can be used with MPS (a standardized linear program formulation file format) being the most known format. Since MPS is not available in human readable form, LP_format is used which tends to be more user-friendly and therefore easier to work with. The last option is an integrated development environment which provides an abstract tool for the user to solve linear problems without extensive knowledge of the underlying reasoning.

3.2 Utility integrated back-end

Using Python, there are two main ways of interacting with LP_solve. First option is the use of python to create a text file and then pipe it into a standalone LP_solve program. The second option, that we decided to go with, was using a dedicated lpsolve55 library in a Python script. Input parameters, that are collected from the graphical user interface are as follows:

- list of connections, made of start node, end node and its price: *Apath*
- list of nodes, made of node names and their capacities: *Anode*

- start node: *Start*
- end node: *Stop*
- payload: *Load*

Functions for parameter calculation and making vectors and matrices to be inserted into a LP model extract the needed data from input parameters listed above. The function for finding all parameters needed to create an objective function and a set of constraints extracts information from input lists. The second function uses calculated parameters to create vectors and matrices. When all of the vectors and matrices are computed, the function, which makes LP model, calculates it and returns the result, is the following:

```
def ComputeLoadDistribution(Start, Stop, Load, Apath, Anode):
    Parameters(Apath, Anode)
    lp = lpsolve('make_lp', 0, NumOfPaths)
    lpsolve('set_verbose', lp, 'IMPORTANT')
    lpsolve('set_obj_fn', lp, PathPrice)
    lpsolve('set_minim', lp)
    PathFlow(Nodes,Start,Stop,NumOfPaths,Apath,Paths,Load,lp)
    NodeFlow(Nodes,Start,Stop,NodeCapacity,NumOfPaths,Apath,Paths,lp)
    lpsolve('write_lp', lp, 'dynamicnetwork.lp')
    lpsolve('solve', lp)
    vars = lpsolve('get_variables', lp)[0]
    if (vars == list):
        vars = [round(x) for x in vars]
    elif (vars == float):
        vars = round(vars)
    return vars
```

Figure 1: Function 'ComputeLoadDistribution' is the main function. It takes input parameters, calls previously described functions and uses lpsolve55 library to create and calculate a linear problem and return it's result.

3.3 Utility integrated frontend

Figure 2: Grapical user interface of didactical utility, which has options for creating a new graph, exporting it in JSON format, importing an existing graph and changing the viewing options of the solution.

The user interface enables simple input of nodes, connections, and their capacities. On top of that, it allows importing, saving, and exporting network graphs in JSON or PNG format.

While designing the user interface, we focused on clear structure and ease of use (Figure 2). By choosing the appropriate function in the top part of the left column, the user can quickly draw the network graph. Next step is to decide on the start and end node, along with the amount of payload that has to be distributed over the network. Those parameters can be set in the top right corner. The last step is to run the program encapsulating the network into data structure appropriate for processing and forwarding it into Python script as explained above. The script then calculates results that are finally visualized on the graph.

The entire application is a standalone executable file, based on three key modules - Chromium Embedded Framework (CEF), LP_Solve 5.5 and GUI. All three modules are connected by Python, but the application itself is based on interlacing of different technologies.

First of all, the combination of HTML, CSS and Javascript provides the interface. Aforementioned web technologies along with numerous predefined libraries allow developer a great deal of flexibility in the field of user interface design. Today, the rendering of web graphics is hardware accelerated, which ensures the scalability of the desirable network. Technologies used in this project are cytoscape.js, jQuery, Semantic UI and FontAwesome.

The second module actually connects the GUI described above with Python functions that provide dynamic calculations of the network. CEF opens immediately after calling the main Python program and plots graphical interface in a standalone window. In the case of this application, data is transmitted using a custom data structure in Python and Javascript bindings.

The last of the three modules is a separate Python script with a set of network conversion functions (lpsolve55), that was already described in the previous section.

The entire program is accessible through a single Python script that connects all of the modules. Selected technologies allow for portability of the program between POSIX and NT platforms, with only minor changes. The final, executable file can be created using Python compilers such as Py2Exe and Py2App, which encapsulates the entire project into one executable file.

4 Proposed utility demonstration case

In order to verify the program functions, a network model as seen on Figure 3 was created. We started by choosing 'Node' in 'Edit mode' in the top left corner and placing all nodes on the canvas, then switching to 'Link' mode and connecting nodes together. As we are operating with oriented graphs, the connection between two nodes starts in the node that is clicked first and ends in the node that is clicked second. If any mistakes are made during the model plotting, there is always a chance to delete selected components with the 'Delete' option. Finally, in the top right corner, we set the A as a start and H as a stop node, and the amount of traffic we want to send through the network is set to 5 units. At any point in time during the build of the model, current configuration can be exported in JSON format, so it can be imported and modified next time, instead of configuring everything from scratch. In the end, by clicking on 'Run LPSolve' button, the pro-

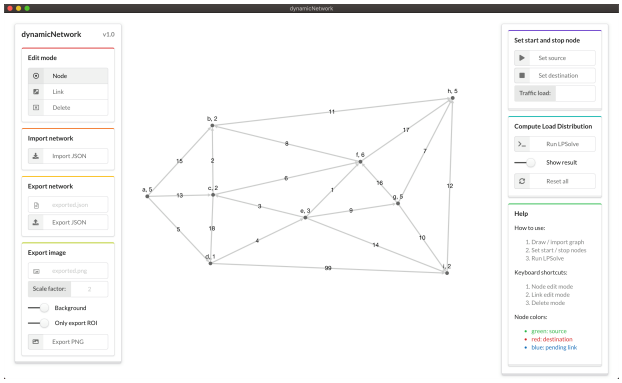

Figure 3: User-made network model in graph shape

gram returns the results, which are seen in Figure 4.

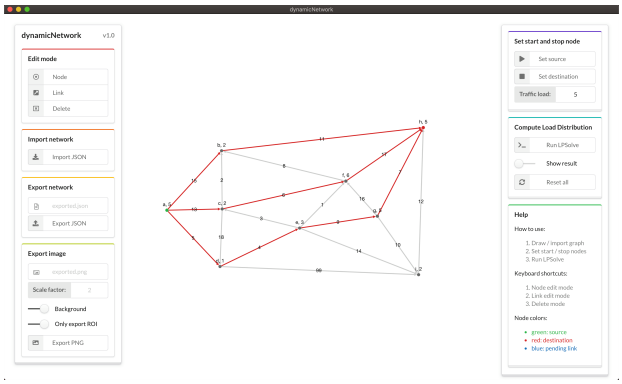

Figure 4: Result with selected connections coloured in red

The program colours the paths that will distributively transport the payload from one selected node to another, in red. Simultaneously, the source node will turn green and the destination node red while the unused paths will remain gray. If the setting 'Show result' stays off, then the graph will still display the maximum available capacity of the path after the calculation. Switching the 'Show result' on, the graph will display, how much of the path capacity has been used (Figure 5).

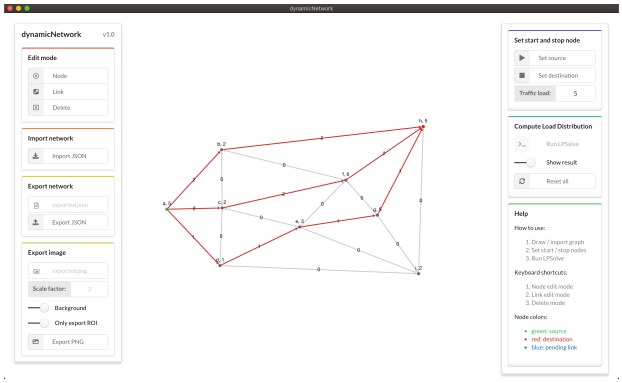

Figure 5: Result with selected connections coloured in red and used connection bandwidth written next to it

5 Conclusion

The use of a good didactical tool in the process of teaching presents a key to practical understanding of a given problem. Didactical utility created in this project enables university and university level teachers to explain the basic concepts of linear programming on simple dynamic graphical examples before going into the details of underlying math and logic. Furthermore, it enables students to compare results of their own LP solutions making it a useful add-on to classes.

Acknowledgement. This research was supported by the project *P2-0246 ICT4QoL - Information and Communications Technologies for Quality of Life*.

References

[1] Thomas S. Ferguson. Linear Programming: A Concise Introduction . `https://www.math.ucla.edu/~tom/LP.pdf`, 2019. [Online; accessed 10-July-2019].

[2] Alice Kolb and David Kolb. Experiential learning theory: A dynamic, holistic approach to management learning, education and development. *In Armstrong, S. J. & Fukami, C. (Eds.) Handbook of management learning, education and development*, 04 2011.

[3] Open source community. Lpsolve tool. `https://sourceforge.net/projects/lpsolve/`, 2019. [Online; accessed 10-July-2019].

[4] Ubalt.edu. Deterministic Modeling: Linear Optimization with Applications. `http://home.ubalt.edu/ntsbarsh/opre640a/partVIII.htm`, 2019. [Online; accessed 10-July-2019].

[5] Wikipedia. Dynamic Routing. `https://en.wikipedia.org/wiki/Dynamic_routing`, 2019. [Online; accessed 10-July-2019].

[6] Wikipedia. Linear Programming. `https://en.wikipedia.org/wiki/Linear\_programming`, 2019. [Online; accessed 10-July-2019].

[7] Wikipedia. Maximal flow problem. `https://en.wikipedia.org/wiki/Maximum_flow_problem`, 2019. [Online; accessed 10-July-2019].