Initial Plan: Linear Programming When There is No Feasible Solution

Author: Iwan Munro, Supervisor: Dr Richard Booth, Moderator: Jose Camacho Collados

Module: CM3203 Final Year Project – 40 credits

Project Description

Linear programming is a method to achieve the best outcome, be it maximising profits or minimising costs, in a mathematical model whose requirements take the form of constraints, which are linear relations.

Many real-life industrial problems can be expressed as linear programs (LPs) (sometimes with huge numbers of decision variables and constraints). It is usually assumed that there is at least one feasible solution, but there are situations, where constraints might have conflicting interests or manual errors occur, in which there might be no feasible solution. In this case, we need a way to pinpoint what is causing this infeasibility (i.e. finding a minimal set of infeasible constraints) automatically.

In this project I will look at ways to pinpoint the errors previously discussed and possibly repair them. My project will comprise of the theoretical side to finding a good algorithm and then the practical side for implementing said algorithm. I hope to produce a program which can take in a LP and decide whether it has a feasible solution. Upon finding this out it can either: 1. if it has a feasible solution then output the LP as is to the user; or 2. if it doesn't have a feasible solution then output the set of constraints causing the infeasibility. If time allows, I will proceed to compute how the LP could be minimally reduced in order to achieve a feasible region.

I am going to need a large sample set in order to test my program which will need to be of different sizes. For this I will either need to accrue a dataset from the internet of large sized LPs, or if I cannot find a suitable dataset then I will have to find a way to create the LPs myself so that I can test my program. Additionally, there are libraries such as, python libraries or dedicated software like Gurobi, which can solve the LP for me so I will be using them for that part. My programme will deal will the output of these pieces of software.

Project Aims and Objectives

My project is going to have 2 aspects to it. The first being the theoretical side. Most of this side revolves around the algorithm for calculating the corrections needed to make an inconsistent LP, consistent. This algorithm needs to be admissible, meaning that it doesn't return a local optimum as a solution but re-evaluates until it finds the globally optimum solution. With regards to this, a good algorithm needs not only to produce the correct result, but it also needs to be time-efficient, producing it in a good time frame. If the algorithm is taking exponentially longer and longer to solve the LP as it grows, then for larger LPs it will be of no use.

The other side is the practical side. Once the theoretical side is finished then I can start on the practical side of the project which is going to have multiple iterations, some of which might not be implemented. I have considered the MoSCoW method when ranking these.

Something not mentioned in the MoSCoW list is testing because it is intrinsic to working code. The testing that I intend to carry out will need a dataset, which I will be getting from the internet or

otherwise I will have to create it myself. This is to ensure that my program produces the global optimum. I will also need to make sure that it is time efficient as the LPs scale in size, meaning that I will need lots of different sized LPs from small ones with 2/3 decision variables and a couple of constraints to the hundreds of each. If I move into the COULD HAVEs section, I will also have to test the UI out to make sure that it is user friendly and intuitive.

I plan for the project to have the theoretical side and a physical implementation of this to be considered "done" in my opinion, but to answer the proposal only the theoretical side is needed. The rest of the parts are predicated on that.

Must

Theoretical side

- Write down the problem to be addressed.
- Write a method/algorithm as follows:
    - Takes an input and decides whether it has a feasible solution.
    - I.e. if the LP is already consistent then return it, if not identify which constraints are conflicting.
    - Has good time complexity.

Should

Practical side

- Find online dataset or generate them myself.
- Implement algorithms, maybe using Gurobi.
- Return whether a feasible solution exists, and if not, returns sets of constraints that are responsible for the infeasibility.

Could

Practical side extras

- Return automatically a new linear program that has a feasible solution, minimally changed from the input.
- In this case, this will occur by completely removing constraints involved in the inconsistency.

- **More advanced**: slightly modify the constraints involved in the inconsistency so that consistency is achieved (e.g. change $x + y \leq 4$ to $x + y \leq 5$).
    - This could include some sort of protection to certain constraints, or even a level of importance attributed to each one. In belief revision, typically the most recently added constraint would be protected but given the constraints are added arbitrarily to the LP then this doesn't help.

- **Graphical**:
    - If there are only 2 decision variables, then have the option of presenting graphical form to the user via Desmos-like picture in order to explain the process occurring in the background.
    - User interface which can be used for this dialogue between the user and the program about which constraints are more important than others.

Work Plan

I intend to use Agile in order to plan my project. The Agile Methodology uses an iterative cycle to continuously improve a solution and always make sure that one is doing the minimal work to achieve the current MVP, hence not wasting time. In the waterfall method, it is expected that nothing about the project will ever change but I can't accurately calculate how much time each part of my project will take – some parts less time than anticipated and some parts more. Therefore, Agile gives a great mix of not wasting a lot of time in areas that aren't fruitful and iteratively working upon a 'complete' solution. I will be using the Scrum framework for my work as I like the format of planning what I'm going to manage this week, keeping my progress up to date with an online scrum board (Trello) and iteratively reflecting on the effectiveness of my work and improving where necessary. I may also use a Kanban board to track the progress of the project holistically.

As discussed in the previous section, there are aims and objectives to the project. These are going to shape the deliverables for my project. Each week I will have a meeting with my supervisor to discuss progress and make any necessary adjustments for the following sprint. Meetings will take place every Thursday as it's the end/start of the sprint. Sprint 1 starts on Thursday 30th January with the main sprint goal of delivering this Initial report on the 3rd of February and ends on 6th February. Then the weekly sprints will start and end every Thursday until the last sprint (Sprint 14) ends on Thursday 7th May which is when the final report is due. I will then reconsider my velocity each week at the end of the sprint and then pull in the appropriate amount of work into the next sprint on top of anything that might have not been completed in the previous sprint. The sprint backlog will be ordered based on the MoSCoW list provided in the previous section.

I plan for roughly the first 3/4 sprints to be the MUST HAVEs of the list, hence working on the theoretical side, researching and building the algorithm for the project. Therefore, by the end of sprint/week 3/4 I hope to have completed the theoretical side of the project. After sprint 4, to make sure I am working iteratively I will start working on my final report, writing about my work to date. This could take 1/2 sprints, meaning that I will hopefully have an MVP in place by sprint 6 at the latest. Next, I will move into the practical section and into the SHOULD HAVEs. This starts by implementing the algorithm that I will have worked on before that and creating a physical representation of it. Gathering the data could prove to take a long time to complete. If I manage to find a dataset online then I can just use that for testing with my program but if I can't find a suitable dataset then I will have to spend time working on a piece of software to generate the data set for me. If I must create my dataset then I will have to give at least a sprint to writing the piece of software that manages that and then test my program. I foresee the writing of the program, dataset and other testing to take roughly 4 sprints which will put me at week/sprint 10. The remaining time will then be devoted to writing the final report ready for submission and the end of sprint 14. If these timings end up as an overestimation and I have sprints to spare, then I can start trying to implement some of my COULD HAVEs section into the project.