Linear Programming When There is No Feasible Solution

Iwan Munro - C1673143

May 15, 2020

Supervisor: Dr Richard Booth
Moderator: Dr Jose Camacho Collados

1 Abstract

Inconsistent linear programs (LPs) currently provide a dead-end in optimisation problems, forcing manual intervention. Many real-life industrial problems can be expressed as LPs (sometimes with large numbers of decision variables and constraints). It is usually assumed that there is at least one feasible solution, but sometimes due to manual error in formulating the problem or a misunderstanding of the solution space, there might not be a feasible solution. In which case, a way to automatically pinpoint these errors (e.g. finding some minimal set of infeasible constraints) so the industrial partner can correct them, or have an augmented LP returned to them, needs to be formulated. This project will discuss how to pinpoint these errors and augment LPs that have no feasible solutions.

2 Acknowledgements

I would like to thank Dr Richard Booth, my supervisor, for the continued support that he has provided particularly given the complications that Covid-19 has brought.

Contents

List of Figures

3 Introduction

In industry, a common goal of companies is to maximise profits or minimise costs. In order to do this they understand what is limiting these goals and therefore what is the best combination of these limitations to achieve their goal. How do they do this?

Linear programming is an optimisation method used to achieve the best outcome in a mathematical model often expressed in real-life industrial problems in, for example, supply chains in the agricultural, or engineering industries. Here they are used to compare the limitations of different resources (e.g. workforce capacity, materials, etc) and find an optimal combination to maximise or minimise the "objective function" which in this case would be the profit of these items, hence a maximisation. Once the linear program (LP) has been constructed the graph should have a portion of the graph which is considered the feasible space, the part of the graph that represents possible permutations of a problem, see figure 5.

This project will explore how to fix the LP when it doesn't have a feasible space, meaning that there is no solution to the problem. I previously discussed about adding in limitations to the LP, and the problem tends to occur when, for example, production goals are added which can lead to an inconsistent LP, as in figure 6.

3.1 Aims

The aim of this project is to automatically turn infeasible LPs into feasible versions so that the user can not only reach a solution, but also understand the solution space in greater detail and develop an awareness into what created the infeasibility.

In order to reach this goal I will have to segment my work and produce a schedule. Whilst the waterfall method does offer some benefits and provides the opportunity for a Gantt chart to be created, I will be using the agile methodology as I believe although I can produce a rough schedule, there are too many unknowns. Given the nature of agile, breaking work into 2 week stints, it is much more manageable and easier to be more detailed in planning.

For this project, I used the MoSCoW technique to decide the deliverables. My brief was to find a solution to the problem of infeasibility within LPs so I decided that the Minimum viable product (MVP), or what has to be completed in order to consider the brief met is creating the algorithm theoretically. The Should and Could haves are comprised of the practical side with a focus on always working towards an MVP. Which, after creating the algorithm theoretically, is to create the algorithm in code using various libraries. I don't plan to just complete the Must haves because I can't truly estimate how long each process will take and therefore cannot accurately estimate how much work I can do so that's why the Must haves include my MVP. I aim to complete the Must, Should, and some Could haves.

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.
- 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.
- 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.

Won't have

- More advanced: slightly modify the constraints involved in the inconsistency so that consistency is achieved (e.g. change $x + y \leq 4$ to $x + y \leq 5$).

Figure 1: MoSCoW technique for project aims

4 Background

Before I compiled knowledge of the project past and present I needed to consider my approach holistically. There are some constraints on the work that I will be undertaking. Time. The amount of time I have to complete this project is limited, hence there's a need to be efficient with it. A piece of software that I used, Gurobi, reduces some wasted time that I could have spent re-creating founded knowledge but this piece of software does also bring some constraints. Using this software means that I am bound to the infrastructure that it offers which does reduce my freedom with my solution. I talk about this further when discussing IISs. There are also some assumptions that I have made in order to carry out my solution. Since my work is looking at the real-life industrial examples, non-negativity is implied in my solution as well as all values being real numbers as it is not possible to have anything but positive values for real-life situations.

4.1 Similar Research

In order to understand further the length to which my project will be unique, I first need to understand what research is currently available.

Whilst the lack of research in this area allows for an ease into novelty for my solution, it also means that I don't have a huge basis for my own research. I have had to dig far deeper to find research to analyse and see different techniques that can be applied to the solution that I am creating, but also to understand what has already been accomplished by other people so that I can give the necessary credit.

There are some other avenues that individuals have taken, tackling the problem from different angles. Some recent and some older, taking the approach of Irreducible Infeasible Sets, approximations, regularization and duality theories. Erik Dravnieks paper [1] from 1989, although dated, provides a detailed overview of multiple strategies using both the simplex method and IISs, particularly his depth into IIS showing the difference between isolated and overlapped IISs which can increase the complexity of the problem greatly. What Dravnieks segues into is relaxation which is also explored in Roodmans paper [4]. What Roodman describes as "the modeler is usually given no tools for doing further analysis" is still partly a constraint with gurobi today which has the functionality to create an IIS but writes it to a file. In this project my goal is to go a step further and return to the user the optimal feasible region created by removing constraints. Where my research differs with Roodmans is that his research is preliminary to mine. This is accentuated by the age of the paper as we now have tools which can determine infeasibility which gives me the ability to focus my work on the post-analysis of the LP. Roodman does make good headway into describing why the dual simplex method can be useful and how to relax the LP in order to achieve feasibility. From the industrial perspective that I am considering this problem, I think that there isn't a noticeable difference between the two routes (relaxing or removing) for the customer as both return to the customer an solution and both are different from their input.

Other pieces of research considered the problem in it's matrix format [2], [5] and how correction could applied using matrix mathematics. All these pieces have one thing in common. They are a very different route to the one that I am taking as they are heavily theoretical, so whilst they provide a good insight into the mathematical proof behind matrix correction as a solution for this problem they do not venture into implementation. My goal is to, not only, create my algorithm theoretically but to also establish an implementation of it that can be used to solve a problem by someone without the mathematical knowledge necessary to read these papers and understand these solutions. Where they do intersect with my research is in the theory, which has helped me develop my understanding of what is going to make my algorithm different, and also how to justify my choice of algorithm.

4.1.1 Irreducible Infeasible Sets (IIS)

An IIS is a minimal set of constraints which is infeasible and becomes feasible should one constraint in the set be removed. This is the case with one set of infeasibilities for the model. In cases with more than one set of infeasibility then there can be multiple IIS which also have the ability to overlap.

Kellner's paper [3] focusses on the use of IISs in order to formulate a minimum amount of constraints that would relieve the infeasibility from the model. Also what Dravnieks [1] succinctly outlines as well. Although I am not using IISs specifically my solution is very similar in nature to the concept. It has been helpful to see their justifications for use of IISs and how they laid out their research with respect to them. Part of my algorithmic approach, dealt with the different ways to deal with isolated and overlapped IISs, which Dravnieks [1] also explores in his research, but in my approach not using IISs gave me more freedom to create a much simpler solution to the overlapping of inconsistencies because of the constraint that gurobi created as I explained previously.

4.2 Agile Methodologies

The Agile methodology is based around creating and responding to change, especially in an uncertain environment. This project involves a lot of elements that cannot be accurately estimated. Hence the use of Agile. In Agile, the project is broken down repeatedly until we find ourselves at small definable tasks that can be estimated to a greater extent. The methodology encompasses an iterative process which you can see in figure 2. Instead of having all the meeting, then all the planning, so on and so forth, this iterative process allows for small amounts to be done at one time, so a meeting to discuss some early requirements, this then evolves into planning this small amount and then designing, developing, testing and evaluating all whilst focusing on this smaller amount of work. [6]

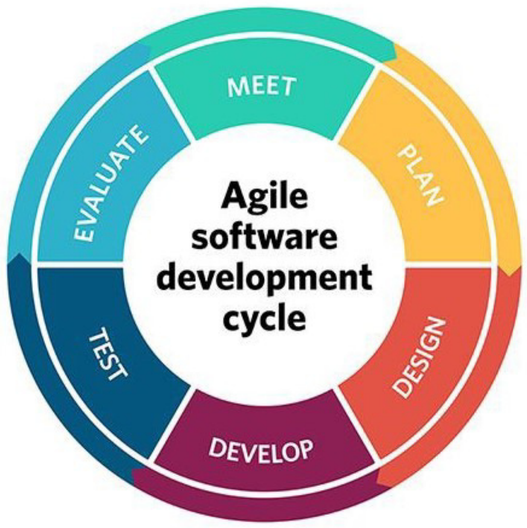
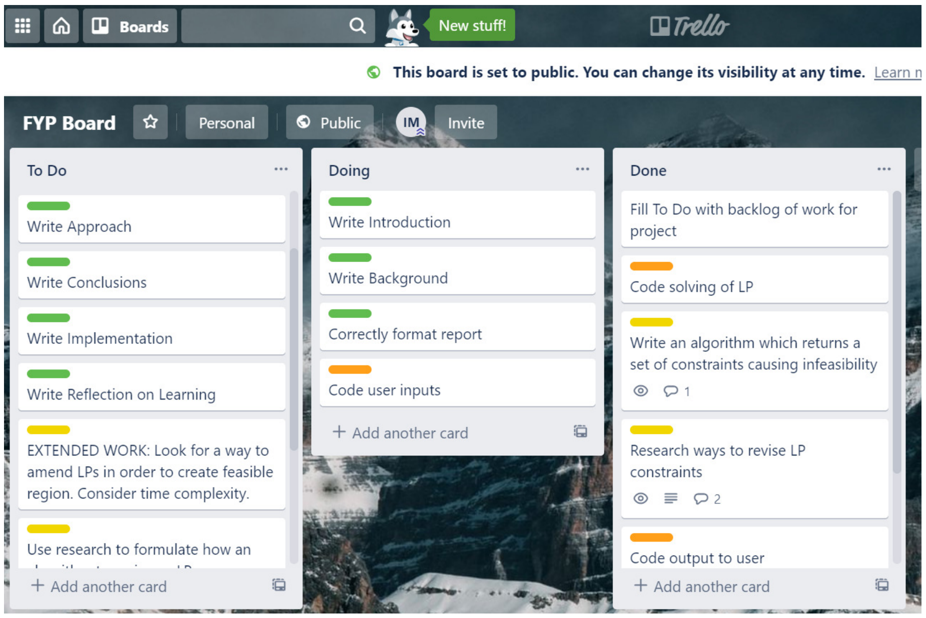


Figure 2: The Agile Software Development Lifecycle

This process minimises time wasted on work that is not included in the final release, therefore increasing the efficiency of my time. It also helps me to adapt and react as the project changes. In other models, such as waterfall, as the flexibility of the work is very limited the amount of wastage increases greatly, because there is a lot of work done before the understanding is gained that the part isn't viable. Going down dead-ends is inevitable in projects as one tries different routes to solve a problem, but by breaking it down this amount of work done before the dead-end is found is greatly reduced. Another benefit of breaking down work, is that the rate of completion is higher which aids as a mental stimulus given you have finished more pieces of work even though they are smaller in size.

Kanban is an agile framework which helps adhere to agile principles in order to get work done. The essence of kanban revolves around visualising your work, with a kanban board (figure 3), and optimising the flow of tasks from “To Do” to “Done”. This board has not only helped me visualise my work, accompanied with the MoSCoW prioritisation to decide which tickets get pulled in, but also to communicate with my supervisor as to where I was in my project without their need to consult with me. Inside each ticket I also have the ability to comment which I used to track progress on different tickets especially when a few days may go between actively working on a ticket which helped me start in the right place each time eliminating unnecessary wasted time on catching myself up. [9]



This board is set to public. You can change its visibility at any time.

FYP Board

To Do
- Write Approach
- Write Conclusions
- Write Implementation
- Write Reflection on Learning
- EXTENDED WORK: Look for a way to amend LPs in order to create feasible region. Consider time complexity.
- Use research to formulate how an
- + Add another card

Doing
- Write Introduction
- Write Background
- Correctly format report
- Code user inputs
- + Add another card

Done
- Fill To Do with backlog of work for project
- Code solving of LP
- Write an algorithm which returns a set of constraints causing infeasibility 1
- Research ways to revise LP constraints 2
- Code output to user
- + Add another card

Figure 3: My Kanban board

4.3 MoSCoW Prioritisation

MoSCoW is a prioritisation technique and a part of the agile framework, DSDM. I am using it to understand the relative importance of separate tasks so I can realistically estimate how work I can manage. MoSCoW is an anagram which stands for Must have, Should have, Could have, and Won't have. One benefit being that delivery to the customer is very clear. The customer can communicate what definitely needs to be delivered and then prioritise the rest of the project according to the other titles which are subjective and can be defined by the customer. The crux of MoSCoW is flexibility in the requirements to be delivered.

Must haves are essential to the project. Put simply, the project is not worth releasing if these requirements are not met. They underpin the main goal of the task therefore everything in Must haves must be a critical task. Whilst the must haves are the essential parts to the project, a team should not plan to just complete the must haves as that would create a skeleton for the project. The aim should be to complete **at least** the must haves and some should haves and could haves.

Should haves and Could haves are quite similar hence the need for subjectivity in the description of each one to create that distinction. It should be possible that the work in Must, Should, and Could have can be completed in the time, but it's not imperative that it must be completed. The slight difference of the tasks in either of the categories are based on their necessity to the final problem. Should haves tend to be important whereas Could haves are more desirable aspects. Could haves will have a lot less impact on the final solution if left out.

The last part, won't haves, are tasks that will not be delivered **in this time frame**. If someone were to pick up the project afterwards, after the remaining should and could haves were completed, then this is where they would start. They show the direction of the project and also help manage the expectations that not all ideas can make it into production. This helps align the customer with the development of the project as they understand that every new idea they have cannot be made first priority as there is already an understanding in place. [7]

4.4 Linear Programming

The requirements of a LP represented through linear relationships e.g. $x_1 + x_2 \leq 8$, as represented by $\mathbf{Ax} \leq \mathbf{b}$ in figure 4, which are made up of decision variables e.g. $x_1 + x_2$. The figure also expresses the objective function ($\mathbf{c}^{\mathrm{T}}\mathbf{x}$) which attributes value to each variable i.e. $2x_1 + x_2$, in this example variable x_1 has greater value. The final part of the expression, $\mathbf{x} \geq 0$, is a non-negative constraint. Simply put, the value of the variables in the solution cannot be below 0.

$$max\{\mathbf{c}^{\mathrm{T}}\mathbf{x} \mid \mathbf{Ax} \leq \mathbf{b} \wedge \mathbf{x} \geq 0\}$$

Figure 4: General Expression for LP in matrix format.

In order to have a solution LPs need to create a feasible region as you can see in figure 5. The objective function is then used to find the optimal combination of the variables with regards to the constraints. **The problem that I will explore is when there is no feasible region.** In this situation the set of constraints provided cannot agree on a region at all which leaves the LP inconsistent, a good example of which is in figure 6. As you can see, if you take any pair of constraints then you can find a feasible solution. It is only when you consider all the constraints that the infeasibility is encountered. Therefore the only way to create a feasible solution from an inconsistent LP is to augment the LP is a way that reaches this.

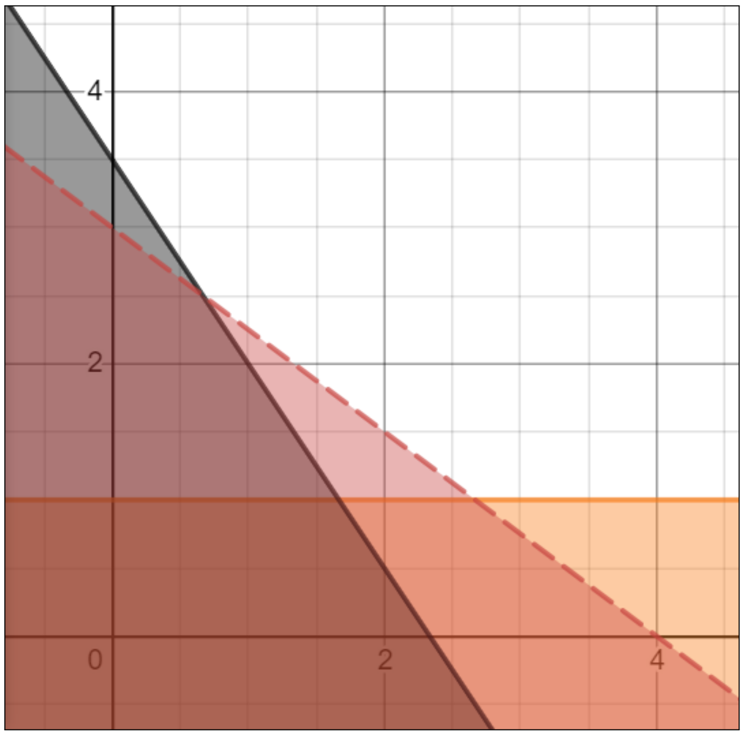


Figure 5: A Consistent Linear Program of $3x + 2y \leq 7, y \leq 1, 3x + 4y < 12$.

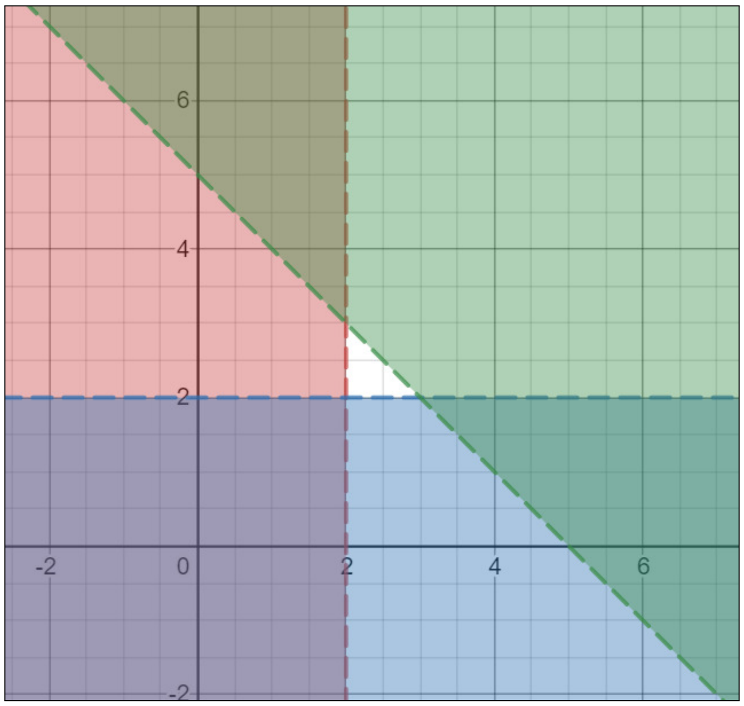


Figure 6: An Inconsistent Linear Program of $x \leq 2, y \leq 2, x + y \geq 5$.

4.5 NumPy

NumPy is a package within python specifically for scientific computing. As outlined earlier, time was a constraint during this project so I used some packages to remove some to the trivial work so that I could focus on the unique parts. Hence I used NumPy to solve simultaneous equations when separating out the constraints.

4.6 Gurobi Optimizer

Gurobi is the best mathematical programming solving software. Instead of creating a framework for dealing with the LPs myself I decided to use the Gurobi Optimizer to handle the objects, and create the base for which I would do my post-processing upon. As is with the NumPy package, this part was trivial and so a better use of my time was to work on implmenting the algorithm I had written. [8]

5 Approach

As I was using a kanban board for this project, I started out by creating tickets for all the work so that I could visualise the life-cycle of the project. I split the project into three sections: Theory; Implementation; and Report writing. Although these are 3 distinct parts, one of the main parts of Agile is to push towards the MVP. Whilst first I needed to theorise my solution, research different techniques, creating an algorithm to meet the goal, if I couldn't implement this then the wasted time would be costly. Hence, I spliced the sections, created loops and I started by theorising solutions with the constraints further down the process in mind. I would also take notes constantly to ensure that when I wrote the report in full I wasn't trying to remember parts of the project that were dated, but I could refer to my notes and explain different challenges and how I overcame them. For example, I wasn't going to be using ISSs as gurobi outputs them to a file which wasn't very accommodating to continuity within the program. Additionally, it gives no freedom in the manipulation of the ISS. I am considering this problem with a case in mind (real-life industrial situations) whereas most other pieces of research using ISSs had been considering a more general format which didn't consider the nuances and pitfalls of different uses. One goal I had in mind was to not only isolate the constraints causing the infeasibility, but creating a feasible region by removing constraints and returning the best solution within this new feasible region to the user.

An important part of implementing my algorithm was testing that it worked. Initially, I had hoped that I would be able to acquire a dataset to use in this example, but the specific nature of the problem lead to me not being able to find a dataset that I could use. Instead I had to create examples to use myself which took time that wasn't intended for it. I will discuss the dataset further when analysing results.

5.1 Alternative Designs and Final Algorithm

In order to create a "good" algorithm I need to define what a "good" algorithm entails. I believe it should satisfy 4 properties:

- Defined inputs and outputs;
- Efficiency;
- Simplicity;
- Effectiveness.

For this problem the I/O's are very simple, the algorithm itself should take an inconsistent LP and then output a consistent LP that has a feasible region. The algorithm must be clearly defined with good regard for detail where necessary. It should also be efficient with regards to the size of the LP. How does the algorithm scale? Does it have a time complexity that exponentially increases the time taken to solve with huge proportions? This also ties into space-efficiency, for the algorithm should not use more memory than is necessary. Is it simple? Does the algorithm take a longer route than is necessary or is this the optimal route given the algorithm. Finally, effectiveness considers the correctness of the algorithm. Does it provide a correct solutions all the time?

After producing a "good" algorithm then I need to consider if this algorithm is achieving optimality or just producing correct solutions. To achieve an optimal solution, I need to consider what the algorithm needs to achieve. In an industry setting, these inequalities all represent different needs within the business, therefore the algorithm needs remove the least amount of constraints in order to produce a feasible region.

I did not reach my final algorithm first, it was an evolution of several ideas. One idea which would have had a large reduction in the overhead of the program was a ranked list of the constraints that was user-determined. What would then happen is an algorithm would simply remove constraints from the bottom until feasibility was reached. I had a few issues with this. For one, even if the list was in the order of individual rankings, how could multiple constraint ranking be considered. Using an example of $constraints = [1, 2, 3, 4, 5, 6]$, by this I mean if the model could be solved by simply removing the constraint at rank 3 then which is more important? Having rank 3 or having ranks 4, 5, and 6? I believe it is a matter of definition which I go on to explain in my next section. My other qualm with this implementation is the feasibility of the solution in the context I am considering. How would such a ranked list be compiled? Would there be a big board meeting to discuss the goals of the company and how it constraint affected them and hence why it was more important than the others. Whilst this is a theory and implementation based project I feel it pertinent to consider the use case.

With regards to inequalities, infeasibility is caused by constraints of opposite inequalities as any possible constraints with the same inequality will produce a feasible region, however small. The problem therein lies with which constraint to remove, so whether the objective function is a maximum or a minimum has a large effect on the removals. In a maximisation function (e.g. MAXIMISE $X + 2Y$), the constricting parts of the LP, those which limit and stop the LP from being unbounded, are those creating a maximum bound (e.g. $X + Y < 3$). Therefore the algorithm should remove the constraints creating a minimum bound first (e.g. $2X + Y > 8$) as the objective function is already telling the algorithm to find the largest values of the variables. Figure 2, provides a good example is which removing one of the maximum bound constraints ($x < 2, y < 2$) would create an unbounded LP which although does produce a feasible region, this region should be finite.

One common industrial example for LPs are supply chains. In this context, minimum bound constraints tend to be sales-orientated goals, for example the minimum amount of a product one needs to produce in order to meet revenue goals, whilst maximum bound constraints tend to be a limit on materials. This example proves that it doesn't help to remove the maximum bounding constraints for they represent real limits which cannot be pushed easily.

```
consts ⇐ list of constraints in model
for i in consts do
    for j in consts do
        Reduce i and j by amount of removed variables
        if i == j then
            skip
        else if either constraint has been removed then
            skip
        else
            solve i = j                                    ▷ Simultaneous Equations
            if solution not in positive quadrant OR re-run = 1 then
                if constraints inequalities are opposite AND both constraints aren't in the blacklist then
                    remove constraint that has opposite inequality to the obj OR constraint that isn't in blacklist    ▷ For a max obj remove ">" constraint
                    optimise model()                       ▷ Looking for solution
                    if constraint removal creates unbounded model then
                        add constraint back to model
                    end if
                    if if solution found then
                        output variable values, obj value and consts removed
                    end if
                end if
            end if
        end if
    end for
    if i at end of loop AND no solution found then
        re-run ⇐ 1
        re-run algorithm()
    end if
end for
```

Figure 7: Pseudocode for final algorithm

In figure 7 you can see the final pseudocode for the solution that I theorised. This is the product of many iterations of improvement, as I added in more functionality but also as I spent time refining it. To break it down, generally, from the pseudocode, the main data structure used in the list of constraints. The use of the doubly nested for loop ensures that every constraint is compared against every other one. What happens next are similar to catch statements. Situations which can potentially break the algorithm and don't add anything in terms of functionality. This is in the form of checking if the constraint is being compared with itself or whether either of the constraints has already been removed from the model, by means of a separate list that I am keeping track of that with. This then moves into the bulk of the algorithm, where the NumPy module gets used to *solve* $i = j$. This is to work out if either of these constraints plays a part in the infeasibility with respect to each other, so I'm finding out if the result of the simultaneous equation (i.e. where the two lines intersect) is in the positive quadrant (i.e. both x and y values are greater than or equal to 0) or not. Then the next if statement decides whether a constraint will be removed, based on the result for the previous check (if the two constraints intersect outside of the positive quadrant) or whether the algorithm is being "re-run" which I will explain last. The next check is another step in deciding the eligibility for removal, this time I am finding out if both constraints have opposite signs (i.e. '<', '>', '=') and whether at least one of the them isn't in my blacklist, another list that I am keeping. The blacklist acts similarly to the catch checks I have at the start of the algorithm, in that if any constraint removal produces a unbounded result then they are added to this blacklist, so that the situation doesn't arise again. Now a constraint is removed but some post-processing needs to be done as gurobi doesn't do any pre-processing to determine feasibility. As explained before, if the constraint removal pushes the model into an unbounded state then it is added to the blacklist and re-add back into the model. Any post-processing step is checking whether the model now has a solution, if so then I can output it to the user and stop the program as it has reached its goal. Otherwise, it carries on to the next pair of constraints. The last part of the algorithm deals with multiple line infeasibility. What this means is when 2 or more lines are creating infeasibility, with one or more then finding this is not as simple as simultaneous equations so I re-run the algorithm again but with a marker. This marker bypasses the need for the two constraints to have to intersect outside of the positive quadrant but still performs all the other checks on the pairing. A good example, is in figure 6 which shows an inconsistent LP where my algorithm would remove the $x + y \geq 5$ constraint as their is a multiple line infeasibility.

6 Implementation

This section will relate the approach to the literal implementation of my solution. The first decision I needed to make was the language that I was going to write my program in. Between Java and Python, whilst Java is more comprehensive and has a better structure, I have a much greater proficiency in python making it easier to manipulate which was of high importance, given my limited knowledge on both gurobi and NumPy. As part of the iterative nature in agile, I decided to get some hands on experience to how the gurobi optimizer works, reading documentation and experimenting with different concepts. This helped me understand what constraints gurobi might have and hence, refactor the way my algorithm works in the first place. The documentation that gurobi provides in quite extensive in some aspects, but it is incredibly hard to traverse which leaves the reader unsure of the full functionality of the software. This coupled with the lack of examples, mean that some issues took a great deal of time to work through a solution to, for example, retrieving the sign that was inside a constraint was a simple process but due to the lack of traversability meant that

I had to use external sources in order to just understand the full flexibility of the software. The specific nature of the problem that I am exploring can also can be considering a constraint. Whilst one way to look at the lack of interest in the area, means that it's easy to create unique work it also means, that there isn't a lot of support online to help troubleshoot issues or give examples with regards to software. Despite these drawbacks, the gurobi software helps builds the structure for which my algorithm will sit on top of. Without this, it would take me lots of valuable time to create this structure, all that I needed to do was familiarise myself with the software. Fortunately, NumPy in a very well established platform that didn't produce any problems and my use of it was very limited. Again, it was just to streamline the process to avoid wasted time on duplicated knowledge.

As the process then began to grow, I made sure that an MVP was always the goal which lead to a few iterations of different parts of my code base. The code was split into two main parts. The first being the construction of the environment using the gurobi optimizer and then the second part would be my work where my algorithm would perform on the output of the gurobi optimizer in the first half. I am going to explore two places in particular: Data structures with regards to efficiency; and how I decided on which constraints should be removed first.

In figure 8 we see the data constructs that I used to build my algorithm around. A point I made earlier was that one element of a good algorithm is efficiency. Efficiency is a relative term, as it is based on the problem that you are solving. In this situation, as a baseline, I need to be able to iterate through the constraints and solve simultaneous equations for pairs of constraints. These are shown by the consts object that I create and the use of A and B matrices for which the simultaneous equations can be completed. By keeping most of the structures inside the loops, most of the structure are not kept past the duration of their use. I also had the blacklist to make sure the algorithm didn't get stuck and removed_lines list for the output of the problem. A complication that then arose was what figure 6 outlines. When there is a multiple line inconsistency, there is no short way to understand the infeasibility apart from solve them as an LP. This is when I had to consider moving some variables into global. As I explained previously, the way I checked for this was by re-running the algorithm with a marker variable which you can see denoted by "re-run". This then tells the algorithm that it doesn't have to seclude itself to finding inconsistency in pairs of constraints that don't cross in the positive quadrant. Re-running the algorithm means, that I needed to add the blacklist and removed_lines lists into global as some constraints could be removed in the first phase and some in the second phase.

Time efficiency combines both efficiency and simplicity. By being the simplest version of the solution then it also satisfies being, subjectively efficient for this problem. By using the agile methodology I have managed to iteratively

```
# globals
re_run = 0
blacklist = []
removed_lines = []

# m denotes the model
def refactor(m):
  global re_run, blacklist, removed_lines

  # initialising some objects
  consts = m.getConstrs()
  A = []
  A1 = []
  numVars = len(m.getVars())
  for o in range(numVars):
    A1.append(0)
 ..........................................

# fill the sim eq's arrays
for k in range(len(m.getVars())):
A[0][k] = m.getCoeff(consts[i], m.getVars()[k])
B[0] = m.getAttr("rhs")[i]
A[1][k] = m.getCoeff(consts[j], m.getVars()[k])
B[1] = m.getAttr("rhs")[j]
```

Figure 8: Algorithm - Constructors and Data Structures

build up the project to its goal and then streamline it to ensure that time and space were not used unnecessarily. The use of commenting has made this process easier. By defining each part of the algorithm I pushed myself to understand its specific function helping me decide whether there was a better way to execute said functionality.

One problem I ran into was to do with accessing removed data. When the model removes constraints, then I need to add in checks which stop my algorithm now try to access that constraint. Not only did I add a line in to make sure that if either of the constraints were previously removed then skip over the comparison of them. Having removed a constraint also augments the indexing used throughout the solution. Therefore, I added a statement at the start of the loops. I subtract the amount of removed lines from each of the enumerators in the for loops. This didn't result in much more memory being used, as I already had the removed_lines list in global so I needed to find the length of that and subtract it. This provides as an additional reason for why removed_lines needs to be in global memory.

```
# set of constraints found or the function is being re-run
if match == 1 or re_run == 1:
 # checks that the inequalities are opposite
  if consts[i].getAttr("sense") != consts[j].getAttr("sense"):
   # randomly pick a constraint to be removed
   if consts[i].ConstrName in blacklist:
    rm_index = j
   elif consts[j].ConstrName in blacklist:
    rm_index = i
   elif (consts[i].ConstrName and consts[j].ConstrName) in blacklist:
    rm_index = "miss"
   else:
    # if modelsense is -1 maximise, if 1 minimise
    if m.getAttr("ModelSense") == -1:
     if consts[i].getAttr("sense") == '<':
      rm_index = j
     else:
      rm_index = i
    elif m.getAttr("ModelSense") == 1:
     if consts[i].getAttr("sense") == '>':
      rm_index = j
     else:
      rm_index = i
```

Figure 9: Algorithm - Decision for constraint removal

Figure 9 is a perfect example of the iterative nature of agile. Building my algorithm from the ground up, I started by relinquishing optimality in order to create an MVP. As my solution grew I implemented how my algorithm decided which constraint would be removed when there's a match in order to increase effectiveness and reach optimality. As explained in my approach, in the case that I am exploring, there tends to be a grouping to constraints based on what inequality they have. The figure shows the path for which the algorithm goes through to decide which constraint should be removed. First I need to make sure that either constraint is not in the blacklist and then if neither is, then I can choose based on what the state of the objective function is. If I'm trying to maximise then constraints such as, $x + y \geq 8$, are going to be of less importance and removed first.

Previously, I discussed the positives and negatives of using the gurobi optimizer. One negative I discussed was the lack of comprehensive documentation that the platform offers. This was highlighted again when trying to introduce a UI into my program. Usability is very important when it comes to writing software, for without it the user can find themselves unable to navigate it rendering the work null. I worked with the objective function and constraints manually input as I understood the system, but communicating this to a user without technical experience could prove very hard. Unfortunately, after trying to get the program to use input from a text file, I encountered error after error and with the support for the software being sparse, I decided to refocus my efforts elsewhere. It was important to learn that some parts of a project will not go to plan and can end up being pushed outside the scope. My use of agile meant, that these are reduced and the time spent on these efforts are reduced to a minimum. One aspect of agile I found very helpful in these situations were “Spikes”. Spikes are tickets in which there isn’t enough information know about an area, hence the individual “spikes” and explores this and understands whether it is feasible and can be within the scope. The result of the spike being whether the ticket regarding the implementation of the idea will go ahead. I used spikes to determine this in this situation and others alike.

An important part of the software life-cycle is testing. Testing is the way to verify the functionality of your program and in my case make sure, that it completes the process that I made it to do. In my case, this is by feeding in inconsistent LPs which should then be returned and written out to cmd as consistent. I planned to find a dataset in order to have a broad spectrum of testing data but unfortunately, again, due to the nature of the problem I could not find a suitable dataset so I had to resort to creating data samples myself, which resulted in samples being small in number and in size. This lead to my program not being as heavily tested as I had hoped, but I still created enough examples to test my program in different ways and increase its resilience.

7 Results and Evaluation

I have used figures 10-12 in order to depict the life-cycle of an LP over the course of my program in a graphical manner. Figure 10 shows a LP by the following configuration.

- Objective Function: MAXIMISE $x + 3y$

  Constraints:

  - $x + 2y \geq 8$, “c0”
  - $2x + y \leq 5$, “c1”
  - $y \leq 2$, “c2”
  - $x \leq 2$, “c3”
  - $2x + 5y \geq 14$, “c4”

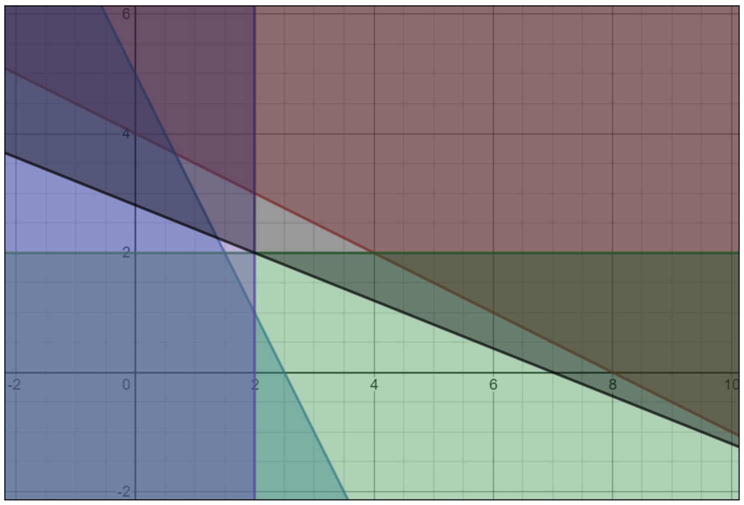


Figure 10: Inconsistent Linear Program Example.

Whatever route we take we are going to have to remove at least 2 constraints to gain a feasible solution for this example. Based on my description earlier of taking the specific use case I am analysing, I would remove both of the greater than or equal to constraints. This would leave behind 3 constraints with a feasible solution. Figure 11 shows the transformed LP with a feasible region. This optimal solution is shown by the blue line intersecting with the region at $x = 1.5, y = 2$.

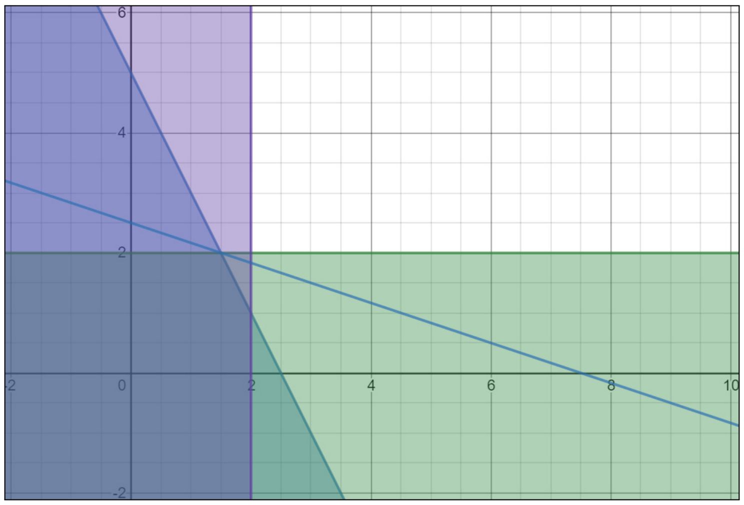
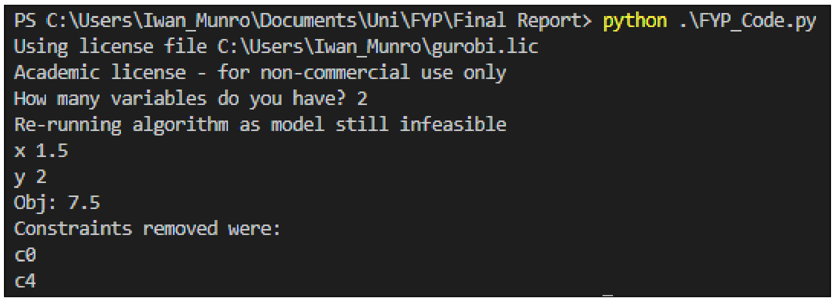


Figure 11: Transformed Consistent Linear Program Example.



```
PS C:\Users\Iwan_Munro\Documents\Uni\FYP\Final Report> python .\FYP_Code.py
Using license file C:\Users\Iwan_Munro\gurobi.lic
Academic license - for non-commercial use only
How many variables do you have? 2
Re-running algorithm as model still infeasible
x 1.5
y 2
Obj: 7.5
Constraints removed were:
c0
c4
```

Figure 12: Cmd output for correct transformation.

Figure 12 shows the output that my program gives to the user after having executed my algorithm on the inconsistent LP. As you can see it has removed both c0 and c4 which are the same, as I removed theoretically and found the optimal solution of $x = 1.5, y = 2$.

This is one of many permutations of constraints that I used to test my solution. These tests proved paramount to the growth of the program and creating a more robust structure to deal with bugs, singing the praises once again of the agile framework, as I could use the tests to integrate solutions whilst I was still building the platform, turning my program into a test-driven environment rather than reactive which can lead to large augmentation of the solution. Testing is a key part of building the credibility of a program. If a program has not been tested, then how can you confidently know that it has met it fundamental aim without bugs and errors. An example of this, was the bug which resulted in the addition of the reduction of the for loop enumerators with respect to the amount of variables removed. If I had not extensively tested this, by increasing the size of the infeasibility, I would have assumed that the program was fully functional given the example I had at the time.

To provide an objective view on whether I met my brief, I must return to the aims and deliverables that I set out in the first place, see figure 1. Whilst I produced an acceptable solution and met many of the goals that I set out to meet, my solution is lacking aesthetically. I did not manage to produce much in the way of a UI and hence my solution is a bit clunky. The manual reconfiguration needed to test different problems, does not lend itself to the average user but one who understands the software and its notation to, at least, a basic level. Whilst the lack of a visual element suggests a lack of work, I am confident in the solution that I have provided, from the algorithm that I have formulated to the implementation of it. Due to the incomprehensive nature of the documentation and support for the gurobi optimizer, even though I tried to integrate this with my solution, I couldn't make it work after having used multiple examples with multiple different formats. Bringing the configuration in from a file is the only option, for using *input*() is not feasible in terms of scalability. The core of the work is strong, but lacks an aesthetically pleasing exterior. Taking into account the time frame in which this took place, also brings light to the complexity of the solution, so with more time the solution could grow to produce a more well-rounded result.

Whilst reviewing the program holistically, I did notice a threat to the optimality of the solution, but it was infeasible to change the course of the project and amend it by this point. It can be argued, as rather trivial as the user would still have a "good solution" and there are many cases where it would still produce an optimal result however given the presence of cases where only a good solution can be found the algorithm cannot be considered optimal. The nature of the removal of constraints when checkout for multiple line inconsistencies, means that there is the opportunity for a constraint that doesn't make the LP infeasible could be removed. In the use case, this could be considered an unnecessary worry given the representation of minimum bound constraints in a maximum objective function and vice versa, but as I outlined previously the optimal solution is one where the least amount of constraints are removed to create a feasible solution. I will discuss an alternate solution in the future work section but the infancy of the idea shows an inability to correctly discern if it is practical.

8 Future Work

Hofstadter's Law of 'Everything takes longer than you think' certainly holds true once again. I went above and beyond the minimum MVP that I could produce but, as discussed previously, the minimalistic UI creates the illusion of a lack of work. Time proved to be a tough element to control and looking to the future of this project, the focus is on the front-facing element. Creating a intuitive and easy-to-use UI, so that the gap between the average and the learned user can be breached for wider use of the system. The two main parts include adding a graphical explanation of what the program has achieved, in order to help the user understand how the feasibility was found, as in figures 10 and 11. Desmos does have a REST-API which I explored in the latter stages of development, but due to the complexity of the idea I had to leave it out of scope. The other was the addition of being able to take the LP configuration from a file. Whilst I explored the use of this and attempted to integrate it into my solution I could not get it to work. This would be a huge step in helping an average user use the program but an important point to make is that this doesn't change the fundamental functionality. It is a functional improvement in order to increase the usability.

With more time I would also be able to delve deeper into the theory of the algorithm, as my base knowledge of the systems now lends me the opportunity to gain more depth in the theory knowing the limitations of the software. What remains as a algorithmic goal inside this project is to not remove any constraints from the LP at all, but augment them to still create a feasible region but without removing any constraints. The heart of this lies at prioritisation which I briefly looked at in my implementation, but decided it didn't deliver enough value given the time constraint. Prioritisation resides within the push to create a more functional UI, so that the user can have a more tailored experience, especially in edge cases where the user might want to flip the decision process for constraint removal. The future of this project would be to take the general case that I have formed and bring specifics in to deal with a far greater amount of use cases.

With regards to improving the current solution that I have in place, focussing on the issue I described within my results I have one other solution theorised. An incremental adding solution by which all of the maximum bound constraints in a maximisation problem are added (i.e. $x + y \leq 5$ in MAXIMISE $3x + y$). As discussed earlier this is not where the infeasibility is encountered, it is when constraints with the opposite inequality are added. Therefore, the incremental part is adding the rest of the constraints one by one and checking the feasibility of the function after each one is added. If it's infeasible then make note and carry on till all constraints are either added or noted. Whilst this way is theoretically sound, the notion of checking the feasibility after every iteration is long and the inefficient time use makes it almost impossible to use it for larger LPs.

9 Conclusions

In conclusion, with respect to the aims set out in the introduction, "to automatically turn infeasible LPs into feasible versions", this project reached its original aim. The program lacks a wide usability pool, but includes all the necessary functionality needed. I managed to complete in full the theoretical portion, creating an algorithm which takes an inconsistent LP and returns one with a feasible region, and whilst some parts of the practical side did not go to plan, it was still a success. I have an implementation of the algorithm that I created and correctly augments LPs and also tells the user which constraints were removed, so they can understand what was creating the infeasibility. I have learnt that a problem can have many different solutions, all of which are valid. I have also explored other areas in order to understand the boundaries of the solution that I have implemented and without the time constraint I could include more of these explorations to give a more rounded user experience.

10 Reflection on Learning

During this project I have gained a more specific knowledge on the subject area and also learnt some transferable skills. Not only has this project expanded my knowledge, but also my soft skills which will prove invaluable in the future. My ability to independently study has grown significantly, from translating the needs of the customer into deliverables, including parts which cannot be within the scope of the project. From the beginning, I have had to be time-conscious and evaluate where time should be best spent to ensure the best possible outcome. Research has also been a focus of this project and I have had to research an area, where I had basic knowledge. I also had to interpret other people's work and use it to reinforce the specialism and need for the research that I was undertaking. A skill in which I have seen significant growth. This has allowed me to coherently translate my physical work into a report format so that it could be understood by others.

Ultimately, I have thoroughly enjoyed the project and the challenge that it gave me from start to finish to signify the culmination of my degree at Cardiff.

References

[1] E. W. Dravnieks, *Identifying minimal sets of inconsistent constraints in linear programs: deletion, squeeze and sensitivity filtering.* PhD thesis, Carleton University, 1989.

[2] V. GORELIK and T. ZOLOTOVA, "Approximation of the improper linear programming problem with restriction on the norm of the correction matrix of the left-hand side of the constraints," *DEStech Transactions on Computer Science and Engineering*, no. optim, 2018.

[3] K. Kellner, M. E. Pfetsch, and T. Theobald, "Irreducible infeasible subsystems of semidefinite systems," *Journal of Optimization Theory and Applications*, vol. 181, no. 3, pp. 727–742, 2019.

[4] G. M. Roodman, "Note—post-infeasibility analysis in linear programming," *Management Science*, vol. 25, no. 9, pp. 916–922, 1979.

[5] A. Vaganov, "Regularization and matrix correction of improper linear programming problems," in *Mathematical Optimization Theory and Operations Research: 18th International Conference, MOTOR 2019, Ekaterinburg, Russia, July 8-12, 2019, Revised Selected Papers*, vol. 1090, p. 283, Springer Nature, 2019.

[6] A. Alliance, "What is agile?." https://www.agilealliance.org/agile101/. Accessed: 2020-05-11.

[7] A. B. Consortium. "Chapter 10: MoSCoW Prioritisation" https://www.agilebusiness.org/page/ProjectFramework_10_MoSCoWPrioritisation Accessed: 2020-05-11.

[8] "Gurobi." https://www.gurobi.com/products/gurobi-optimizer/. Accessed: 2020-05-12.

[9] D. RADIGAN, "What is kanban?." https://www.atlassian.com/agile/kanban. Accessed: 2020-05-11.