Presentation Script

**Slide 1**

Welcome ladies and gentlemen. Today we are going to give a little bit of insight into our project – Rostering in a pharmacy department. We’ve given it the sub-heading you see on the screen, which offers some perspective on the details of this project.

**Slide 2**

A little bit about us. We’re both mathematical modelling students within the bachelor of science. Kipp is a bit more orientated towards the arts and education side of things and Chris more towards problems solving and computer science, but together we’ve found we’ve formed a solid partnership.

**Slide 3**

Our Client is the pharmacy department of monash medical centre. Specifically, our contact points there have been Andrew Chong, the site manager at Clayton, and Jeff Davies, the Intern Coordinator there.

**Slide 4**

So a little about how this all began. In trimester 1 of 2018, we undertook the unit Optimisation Modelling and Decision Analysis with Dr Vicky Mak. Following this, Chris began discussing the possibility of using optimisation to generate rosters at Monash. Jeff in particular was excited about this and gave Chris the details around how he had constructed the intern rosters for the previous 30 years.

Together, Kipp and Chris discussed this with Vicky (as we can see here) who gave us some tips on how to go about creating the systems of equations for this kind of optimisation model. So began our journey…

**Slide 5**

The scrap of paper that Vicky wrote was somewhat cryptic to us at the time, but throughout the course of the project they provided some foundation as we looked at different ways to approach our problem. Vicky also gave us some additional tips at critical times, although was initially reluctant to do so as she was sure we were making money and not sharing it with her. Ironically, after Vicky agreed to help us, we managed well enough on our own anyway. But nonetheless, a big thank you to Vicky for not only giving us some invaluable pointers, but for teaching us Linear Integer Programming in the first place?

**Slide 6**

So what is Linear or Integer Programming? I suppose in their most basic forms, Linear programs are just extended versions of simultaneous equations that we all learnt in high school. But this is probably an oversimplification.

We define it as problem solving using systems of linear equations that generally describe real world constraints to solve an objective. They can take on many forms. The class we have been using is within the family of scheduling problems, on the more sophisticated end of the spectrum.

We are probably all familiar with the mathematical convention using a variable x. In our case, we are using *binary* decision variables, which can take on a value of either 1 or 0; which, as in computer programming, translates to off and on.

**Slide 7**

Before we jump too deep into the parameters of our model, we’ll talk a bit about how linear programming works, and the distinction of integer programming. In the example of the screen, we see it has two dimension – two variables – x1 and x2. The equation we see here is a constraint – from the look of it, it is of the form 2x1 + x2 =6. From the way the graph is presented, this constraint is actually 2x1 + x2 <= 6. As the graph cuts off at the axes, it seems there are also constraints x1>=0 and x2>=0. Therefore, the feasible region of this problem – the area were solutions can occur – is this shaded blue zone between the axes and the line.

An objective function is another equation, but one that does not have an equality or inequality. Indeed, what this function will equate to is called the objective value. An optimisation problem will have objective to maximise or minimize this value. In these examples, we see a maximization problem.

So what about integer programming? As you might have guessed, integer programs can only have integers for their values. As we see with the blue nodes in this example, the feasible regions might be large, but the feasible integers are rather more restricted. Integer programs are particularly applicable because in many situations, we need to use whole numbers for problem solving, particularly when dealing with people. Such is our situation.

**Slide 7.5**

We were hoping to give a bit of an explanation of how integer programming works, and the technique of branch and bound that makes it possible. However, time runs away from us so suffice to say that one solves an integer program by first solving a linear program, picking a variable that has resulted in a non-integer (ie a decimal) and branching in either direction from it. So if you got a value for x1 that is 3.5, you would then attempt the program again, but with two branches – one with x1 set to 3 and the other with x1 set to 4. There are other facets and strategies to branch and bound, and if anyone is curious to hear more, please feel free to ask us after the presentation.

**Slide 7.8**

Our main decision variable is that which you see on the screen, “x-i-j-k”, which when on, means intern i is doing rotation j in week k. The parameters of these are that we have 11 interns doing 17 rotations across 54 weeks. A few of these parameters are misleading – for example, the intern year is 50 weeks long, but they have a staggered start, with 5 starting 4 weeks after the other 6 and finishing 4 weeks later. There are in fact only 14 rotations, but the extra 3 are to accommodate this staggered start and the break up their annual leave.

Our examples above show a 2 variable ILP – 2 dimensions. In fact, our program has a total of 16,265 variables. We don’t encourage anyone trying to think about 16 thousand dimension space. But I think it is enough to say that such a feasible region is altogether more constrained that one of two dimensions.

The consequence of this is to solve such a system by hand would take years for a person to do on their own. Thankfully, we live in the digital age and we have optimisation solvers available to us. For this we used IBM CPLEX, one of the best on the market.

We have a few other decision variables as well, that you can see listed on the screen. Those other than the x’s relate to our If-Then constraints that we shall discuss presently.

**Slide 8**

We’ll move on now to a bit of a look at the systems of equations that make up our Integer Program model. This will serve to give you a bit of insight into what sort of structures make up an ILP.

This system states that for each intern, weeks 51 to 54 must be occupied by j =17, which is our “blank” rotation. We can see here that a “for all” creates a new equation for each intern whilst the summation (or just sum as it is called in AMPL) picks the parameters for each of those equations to add together.

**Slide 9**

Our physical constraint is the rather logical system restricting a person from doing more than one job at a time, for any given week. In this case, we see that a new equation is created for each intern, for each week – a total of 594 equations. We can begin to see how this multitude of constraints constrains our hyperdimensional feasible region.

**Slide 10**

Following on from this, we have the rotation completion constraint, which ensures that each intern does the allotted amount of weeks in each rotation. In this case, we are just using the example of the first rotation, CPD-G or j=1.

**Slide 11**

The rotation capacity constraint is again unique for each rotation – with different rotations being able to handle differing numbers of interns at a time. In our example case of CPD-G, there can be two at a time. We can see here that between constraint systems, as we sum over different parameters of the variables, different statements occur. Indeed, one can frame almost any form of constraint as a linear equation. With some notable exceptions in the field of physics.

**Slide 12**

And so we come to the duration constraints. These are the giants of the model – the ones that suck up much of the computation time. Without these, the model will run to completion in seconds or milliseconds.

This is where the additional decision variables come into play. They are used to enable us to make conditional statements – “if, thens” as they are often known in maths and IT. In this case, the conditional statement is to create ‘blocks’ of the rotations – as we see here, we are saying that if an intern starts a rotation, then they must be continuing it for the next 7 weeks that follow.

It was here that we really began learning new dynamics about ILP. Using ternary variables like x-i-j-k was one thing, but we had only encountered these ‘Big M’ constraints as they are called in theory.

Big M constraint systems are enormous, and computational strenuous. Once we began using these, it began crashing my little computer and gave me the excuse I needed to get a brand new laptop. These systems for the duration constraints were our own innovations on Big M constraints, building on the idea of Vicky’s last year to use another variable alpha to iterate upon a given week. We probably spent as much time developing these (with a lot of trial and error) as we did any other aspect of the model.

**Slide 13**

The last innovation of significance was the pairing of the CPD-G rotations amongst 10 interns. This offers an example of what a pure Big M constraint might look like. This constraint is saying that for any given week, if there is one intern doing CPD-G, then the sum of interns for that week must be greater than 2. Given our rotation capacity constraint limit it to two anyway, this limited the system.

It is an important note that we usually use inequalities rather than equalities. This may seem trivial, but by using this convention, it allows flexibility in the order that the model treats the equations. A model that uses only equalities is much more rigid. In using the equals sign for the ‘hard’ constraints, such as we did for the leave, it meant that the model dealt with these first, and then treated with the others.

**Slide 14**

So we come at last to the model itself. We’ll just show you a quick snapshot of it, really to convey its scale. With each line of code often representing dozens or hundreds of constraints, it is an enormous model. But it serves as an example of how complex rostering can be – something that is often trivialized within workplaces.

**Slide 15**