Imperial College London

MEng Individual Project

Imperial College London

Department of Computing

A Practical Analysis of Optimisation and Recovery Under Uncertainty

Author:
Suraj G

Supervisor:
Dr. Ruth Misener

Second Marker:
Dr. Aldo Faisal

June 30, 2019

Abstract

Optimisation is vital in modern society as it allows us to make mathematically justifiable decisions. In live systems, uncertainty is inevitable, and optimisation under such situations is challenging, but still of importance. Royal Mail's problem seeks to minimise the number of vans used per delivery office, and requires that mail is delivered on the same day. This must be done while considering the many number of things could go wrong. We look at both scheduling and rescheduling processes to address this problem, restricting our focus to Integer Linear Programs.

We critically analyse and extend upon the original modelling which lacked expressivity; in doing so we think about tractability issues with our proposed reformulations. These reformulations are the basis of an extensive study into scheduling and rescheduling algorithms. In this dissertation, we make three significant contributions to the study of the Variable Size Bin Packing Problem under uncertainty. First, we propose effective rescheduling methods to restore perturbed schedules under uncertainty. Next, we translate the stochastic delay-resistant method, one which intelligently anticipates uncertainty, from a different class of optimisations problems. Lastly, we propose a novel generalisation of *recoverable robustness*, a method whose optimal solution comes with a rescheduling strategy that is guaranteed to render it feasible.

Acknowledgements

Many thanks to Dr. Ruth Misener, for her invaluable advice, for putting up with me, and cultivating within me a passion for research.

To my mother, for endlessly supporting me, and to whom I undoubtedly owe everything.

To my family and friends, who have always been by my side.

To the innumerable artists whose aesthetic pursuits endlessly challenge and inspire me.

This project is run in conjunction with the Royal Mail Data Science Group.

Contents

Chapter 1

Introduction

Optimisation's need in modern society needs little justification. It allows us to make justifiably better decisions for a myriad of problems. This is invaluable in complicated, costly systems in which even minor readjustments can result in major operational alleviation. In live systems, however, operations do not always go according to original optimised plan. Upon such disturbances, or *perturbations*, we are no longer certain that our once justifiable and sound decision, still is.

1.1 The Royal Mail Problem

Prior to the day in which mail is delivered to its recipients, Royal Mail must perform certain scheduling decisions to co-ordinate its fleet of vans. These decisions must be performed with several considerations in mind, namely, the scale at which it is occurring. As of 2017, this includes deciding on a schedule for mail across 1,350 delivery offices and 48,000 vehicles [40]. The scale of such an operation tends to lead to uncertainty in the parameters of the problem, which can easily render naive scheduling methodologies infeasible. However, it is still necessary for the mail to be delivered within a day.

Royal Mail's problem aims to **minimise the number of vans used** at each delivery office, and can be thought of as having three constituent phases. The first phase is done on the eve of the delivery. Here, with all the information available, an initial optimal schedule is decided upon. At some point in the day of delivery, we observe the original schedule has experienced several perturbations. The duration of some jobs may have grown in size, and some may have shrunk; a van or two might be unavailable. The original schedule has, at this point, realised uncertainty, which is the second phase of the problem. At this point, the original schedule is likely non-optimal and infeasible. After this uncertainty realisation, we proceed to the third phase by performing rescheduling decisions such that we *recover* from the stochastic shocks of uncertainty. These rescheduling decisions need to consider the real life costs of its actions; it cannot be performed arbitrarily as the operational overhead of unconstrained rescheduling decisions can be prohibitive.

The problem of generating the original schedule is an optimisation process, and responding reactively to perturbations is a rescheduling process commonly termed *recovery*. More concretely, recovery is the process of finding a new optimal delivery schedule, given uncertainty realisations, and the original optimal schedule.

1.2 Objectives and Contributions

The primary challenge of this dissertation is associated with the field of optimisation under uncertainty. A budding field, there are several takes on modelling and hedging effectively against uncertainty. The primary objective of this dissertation is to formulate robust scheduling and rescheduling strategies for the Royal Mail problem under uncertain premises.

To do this, we must proceed in the following fashion. First, we should seek to understand and critically analyse prior work done on the Royal Mail Problem [36]. Next, we require a systematic framework for understanding and evaluating how schedules behave under uncertainty. Following that, we must better understand the nature and efficacy of rescheduling for the Royal Mail problem. Lastly, we must prescribe scheduling decisions for the Royal Mail problem with the processes and systems set up thus far. Our objectives can be succinctly summed up in the following problem statement:

To what extent can we measure the efficacy of, and then design schedules which are both robust to and recoverable from uncertainty?

In tackling this problem, Royal Mail have provided van usage data from three different delivery offices, which we will henceforth refer to as delivery office A, B, and C. This raw data in essence provides the time at which a van leaves and returns back to the delivery office. The contributions of this dissertation are as follows

- **Criticisms of Prior Work and Proposed Improvements** - A weakness with the original modelling proposed by Page [36] is that it does not model which job a van executes. We thus propose a generalisation that does include this relationship in its modelling.
- **Critical Analysis of Proposed Improvements** - We identify a critical drawback in the generalisation we propose, namely that it has constraints of a cubic order, resulting in intractability. As such, we propose an approximate and tractable reformulation of the generalisation. We also prove that the approximation is an instance of the Generalised Bin Packing Problem, commonly known as the Variable Size Bin Packing Problem (VSBPP).
- **Study of Rescheduling** - We explore the notion of constrained rescheduling methods. We confirm the inefficacy of a heuristic approach and the efficacy of modelling rescheduling strategies as an integer program for the Royal Mail problem. The latter method of utilising integer programs contributes to the study of the VSBPP under uncertainty, which is a largely understudied field.
- **Buffered Schedules** - We explore the notion of designing schedules with *buffers*, i.e. adding extra time to jobs as a safety margin. We evaluate the trade-offs of a naive approach, namely the benefits of tractability as opposed to the increased number of vans needed. We also explore a smarter class of methods that utilise a budget of buffers [28, 30], for the VSBPP. In doing so we propose a model which minimises the amount of vans (or bins for VSBPP) used.
- **Delay-Resistant Formulations** - We also contribute, to the best of our knowledge, the first stochastic delay-resistant formulation [30] as a linear program for the VSBPP, contributing to its study under uncertainty. This has only been previously been

modelled for the aperiodic timetabling problem [30], which is a somewhat simpler and very different class of models.

- **Implementation and Criticisms of Recoverable Robustness** - *Recoverable robustness* is an optimisation strategy which incorporates rescheduling decisions in its modelling. Solutions from this strategy come with a guarantee that it can be recovered, or rescheduled in a set of limited scenarios. After proposing a recoverable robust model for the Royal Mail problem, we identify a critical weakness with said approach, namely that the VBSPP is prone to mild infeasibility, which this optimisation strategy cannot handle.
- **Extending Recoverable Robustness** - We term our extension *light recoverable robustness*. It borrows from the idea of *light* robustness [18], which vies for a best-effort approach to solving optimisation problems that cannot meet all of its constraints. To the best of our knowledge, this is the first amalgamation of the aforementioned optimisation approaches. We prove that our new methodology is a true generalisation and propose an implementation of it for the Royal Mail problem. Consequentially, we have proposed a novel strategy to model VSBPP under uncertainty.

1.3 Report Layout

- Chapter 2 covers the background of related material necessary to understand the dissertation and a critical literature review.
- Chapter 3 formalises a method of data cleansing. It then discusses the original modelling and proposes new modelling approaches. It also explores the relationship of the new modelling to the Generalised Bin Packing Problem
- Chapter 4 discusses how statistical analyses was carried out. After this, it proposes a framework to systematically generate uncertainty and score schedules.
- Chapter 5 explores different recovery methods in isolation. This is done experimentally, as it studies how effectively perturbed schedules can be recovered with a multitude of methods.
- Chapter 6 explores two different methods of the design of holistic schedules. These methods are buffering and recoverable robustness.
- Chapter 7 evaluates the work that has been done thus far.
- Chapter 8 concludes the work done in this report, summarises our contributions in further depth, and makes recommendations for future work.

Chapter 2

Background

This section introduces the mathematical background for the paper. Assumed is a rudimentary mathematical acumen. Furthermore, an basic understanding of complexity theory and problems related to computational tractability is needed. The fundamental ideas of the relevant concepts are discussed here, but for a deeper reading, one should consult the referenced material.

The notation used here and in the rest of the report is collated in Table A.1. The terminology and abbreviations used throughout the paper is collated in Table B. Furthermore, any numbers presented in this dissertation will be at most 3 decimal places. Sections 2.1 to 2.3 are an introduction to the preliminary mathematical background. Sections 2.4 onwards deal with more advanced and state of the art methods which include a critical analyses of relevant literature alongside.

2.1 Linear Programming

A linear program is an optimisation problem which consists of an objective function a set and constraints, both of which must be linear maps. The objective function is what is to be optimised. A prevalent representation of this problem is as follows [14]

$$\begin{aligned} \min \quad & c_1x_1 + c_2x_2 + \ldots + c_nx_n \\ \text{s.t.} \quad & a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \leq b_1 \\ & a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \leq b_2 \\ & \ldots \\ & a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \leq b_m \\ & \forall i.x_i \geq 0 \end{aligned} \tag{2.1}$$

The above formulation in Equation (2.1) can be more compactly represented in vector notation as in (LP).

$$\begin{aligned} \min \quad & \boldsymbol{c}^T\boldsymbol{x} \\ \text{s.t.} \quad & \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \end{aligned} \tag{LP}$$

$$c, x \in \mathbb{R}^n \quad b \in \mathbb{R}^m \quad A \in \mathbb{R}^{m \times n}$$

Note that there is an equivalent representation of (LP) as a maximisation problem. An example of this is in chapter 1, section 1.1 of Dantzig and Thapa [14]. This dissertation will, however, only deal with the above representation and minimisation problems. Each x_i is termed as a *decision variable*, as we are trying to *decide* on its optimal value. The space of solutions that obey the constraints is known as a *feasible region*.

There are multiple distinct ways of solving this problem, such as interior point methods or Dantzig's Simplex algorithm. The tractability of linear programs are a complicated matter. While linear programs can be solved by algorithms in class $\mathcal{P}$, these algorithms often exhibit poor behaviour in practical examples as compared to the Simplex method, which is thus most commonly used [33]. This is despite it have a poorer worst-case complexity.

For a further reading on the Simplex and interior point algorithms, and related mathematics one should consult Dantzig and Thapa [14, 15].

2.2 Integer Linear Programming

Integer Linear Programming is akin to linear programming except for one restriction, the decision variables must be non-negative integers. As such the problem is simply Equation (LP) with an additional integrality constraint. This is herafter referred to simply as *Integer Programming*, bearing in mind that it still deals with linear formulations as non-linearity is out of the scope of this dissertation.

$$\begin{aligned} \min \quad & \boldsymbol{c}^T \boldsymbol{x} \\ \text{s.t.} \quad & \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \\ c \in \mathbb{R}^n \quad & b \in \mathbb{R}^m \quad A \in \mathbb{R}^{m \times n} \\ & x \in^{+} \mathbb{N} \end{aligned} \tag{IP}$$

This is an important class of problems and one which is needed for this dissertation. This because, if an optimal number of vans were to be decided upon, it have to be an integer.

Though integer programs appear to be of a similar difficulty to linear programs, this is not the case. It must be noted that linear programs are essentially an optimisation problem on a continuous and convex feasible region [15]. The discontinuity in integer variables results in an optimisation problem over a non-continuous feasible region. This is significantly harder to solve [24]. In fact, integer programs are in $\mathcal{NP}$ [38], significantly harder than linear programs.

We are hence no longer able to immediately exploit algorithms like simplex as extreme points are not necessarily in the feasible region. Furthermore, enumeration methods are not possible for large problem sets as the growth is, at best, exponential for binary decision variables (i.e. values of 1 or 0) [24]. Regardless of this, there exist techniques which are effectively able to leverage the power of the simplex algorithm and exploit heuristics to arrive at an optimal solution. As integer programs are an important aspect of this dissertation, some main techniques common to most solvers are described below. [12, 17, 22]

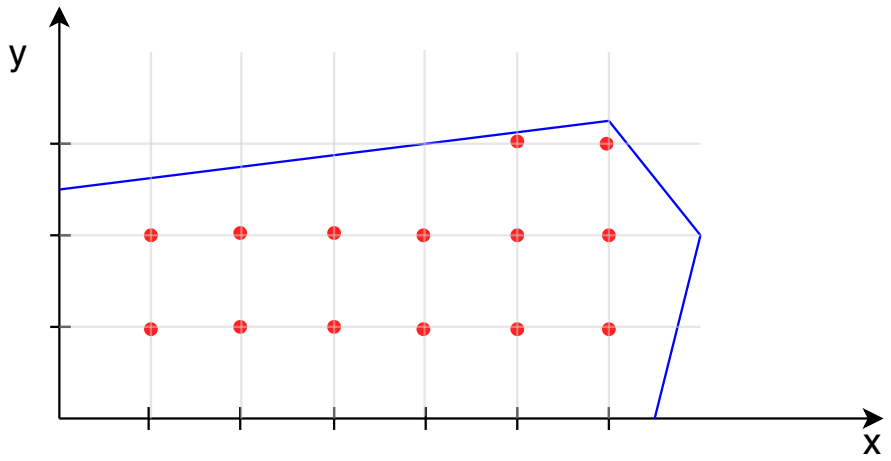


Figure 2.1: The continuous feasible region of linear versus the discrete one of integer programs. The linear program's feasible region is the area between the blue line and the x and y axes. The integer program's feasible region is however, discrete, and is represented by the red points.

2.2.1 Lagrangian Relaxations

A relaxation, as its name suggests, relaxes integer constraints in an integer program. By doing so, the simplex algorithm can then be used to find an optimal solution. This is used in conjunction with other techniques such as branch and bound to move towards an integer solution from the optimal solution of the relaxation. A simple example of a relaxation is presented in Equation (2.2) below, where a binary variable is relaxed.

$$x_i \in \{0, 1\} \quad \rightarrow \quad 0 \leq x_i \leq 1 \tag{2.2}$$

More on relaxations can be found in Geoffrion [20].

2.2.2 Branch and Bound

The enumeration process of an integer program with k decision variables, each with lower and upper bounds x_k^{LB} and x_k^{UB}, can be thought of as in the following code snippet[1].

Algorithm 1 Pseudocode for enumerating an Integer Program

```
for x_1 ∈ [x_1^LB, x_1^UB] do
    for x_2 ∈ [x_2^LB, x_2^UB] do
        ...
        for x_k ∈ [x_k^LB, x_k^UB] do
            Obtain optimal solution for fixed values x_1 to x_k
            Record optimal solution x*
```

[1]This code snippet has been taken from Imperial College London's Department of Computing Course 343 slides

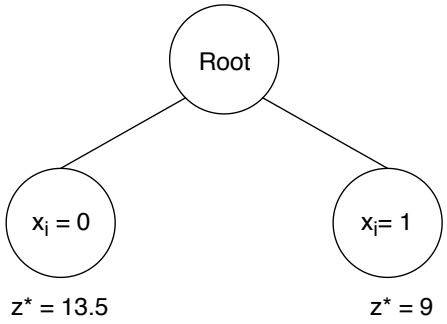


Figure 2.2: This image depicts a branching step. In this branch and bound tree, we have applied two different types of constraints. In one case we set decision variable x_i to be 0, and in the other, we set it to 1. These subproblems are represented as the leaf nodes. Upon solving the subproblem we get different optimal values for each leaf node.

However, as discussed above, although the size of this problem is finite, it can be extremely large. *Branch and bound* is a key divide and conquer technique that greatly helps speed up the process of solving integer programs by preventing the enumeration of solutions that are guaranteed to be worse than a incumbent feasible solution [16]. This process involves three steps: branching, bounding and fathoming. These three steps are described below.

To explain the process of *branching*, let us for example take a binary decision variable, $x_i \in \{0, 1\}$. Branching on this variable would create a tree as seen in Figure 2.2. In the case of decision variable with more than two values, two separate approaches are possible. Either fan out the tree for the number of distinct values in a range, or, use two (or more) ranges to control the fan out (for example, $x_j \leq 2$ and $x_j \geq 3$).

The second step is *bounding* which is to "obtain a bound on how good [the] best feasible solution can be". This is done by solving the linear program with all the constraints present at a specific node. The values we obtain from the linear program are a bound on the best possible solution at each node. Conceptually, adding any more constraints at a given node only restricts the feasible region further. Thus it is not possible to obtain a better solution further down the tree, as by doing so, the original optimal solution at the aforementioned node is no longer optimal (contradiction). To extend this idea to integer programs, we would have to apply a relaxation first, then solve the problem to obtain a bound. Figure 2.2 demonstrates this, where z^* denotes the optimal value at a node. Thus if this were a minimisation problem, we have that for the left and right branches respectively $z^* \geq 13.5$ and $z^* \geq 9$.

The third step is *fathoming*, which prevents the further enumeration of of a subproblem. There are three scenarios where nodes are fathomed. This process is done with respect to the incumbent optimal value, represented as OPT in Figure 2.3. This value is initially OPT $= \infty$. Upon encountering the first **feasible** solution (for integer programs this means respecting all integrality constraints), we set OPT to this value.

In the first scenario, when the solution at a node produces a feasible solution, we do not need to enumerate the sub problems for this node as the parent node is a bound to the problem. As such this node is fathomed.

The second scenario wherein a node needs no further enumeration is when its bound is worse than the best incumbent solution, OPT. The following explanation is depicted in Figure 2.3. If, for an integer program, currently OPT $= 10$, and solving a relaxation for a node gives us 11.3 we can can stop enumerating the subtree as it is at minimum going to result in a value of 11.3. Thus it is "fathomed" and its children not enumerated.

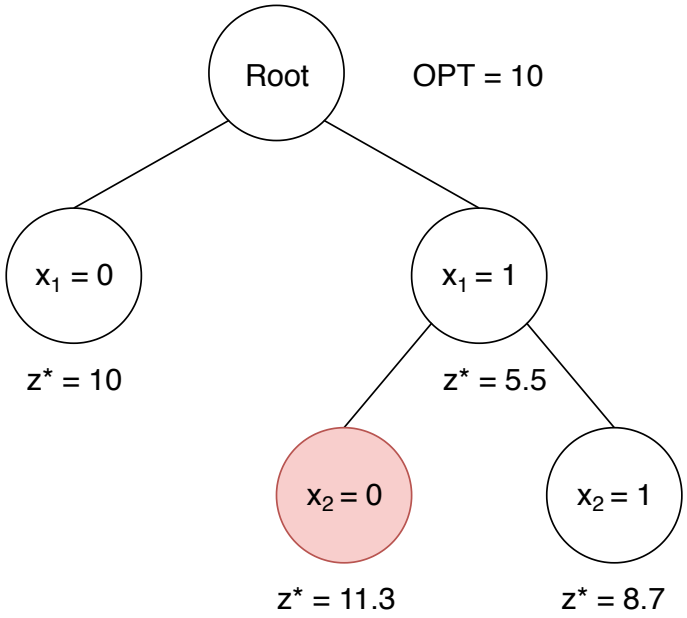


Figure 2.3: This image depicts one scenario of the fathoming step. In this minimisation problem's branch and bound tree, the current incumbent optimal solution, i.e. OPT, is 10. The red node depicts a subproblem whose best solution is 11.3. We can ignore enumerating all the subproblems of this node as its solutions can only at greater than or equal to 11.3, which is worse than the incumbent solution.

The last scenario is when a node introduces a constraint resulting in infeasiblity. Hillier and Lieberman [24] provide more detailed examples and diagrams on the branch and bound process.

2.2.3 Cutting Planes

Cutting planes, or *cuts*, are an important technique that introduces a "functional constraint that reduces the feasible region for a linear program relaxation without eliminating any feasible solutions for the IP problem" [24]. Introduced first by Gomory [21], the main steps of applying cuts are as follows:

Algorithm 2 Pseudocode for Applying Cuts

1: Perform the optimisation on a relaxed integer program. If decision variables are integer, end with optimal solution. Else go to step 2.
2: Generate a cut based on the solution and add it as a constraint. This constraint would be satisfied by integer solutions but not by non-integer optimal solutions we currently have.
3: The solution is infeasible. Go back to step 1.

There are multiple types of cuts, such as Gomory or Knapsack cuts, to name a few. Such cuts are used in conjunction with branch and bound techniques to greatly speed up the entire process of solving. Such techniques are known as *branch and cut* [24].

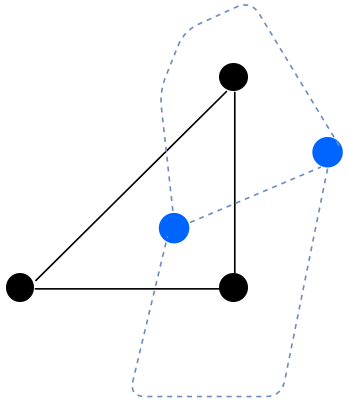


Figure 2.4: This picture depicts the notion of duality in graphs. The dual of each vertex is a face, and the dual of each face is a vertex. The blue graph is thus a dual of the black graph, and the dual of the dual of any graph is an isomorph of said graph.

2.3 Duality

In the field of mathematical programming and optimisation, a problem has an associated *dual* problem and the given problem is termed the *primal* problem. The dual is related closely to the primal in ways we shall enumerate below. Mathematical duality itself is hard to define but Figure 2.4 depicts this concept in a simple graph, where the blue and black graphs are duals of each other. The dual of the dual is the original primal problem.

For a deeper understanding of Duality, on should consult the literature, but here we summarise the idea of strong duality [14, 15, 24].

Strong duality essentially states that if the primal and dual both have a feasible solution then there exists an optimal feasible solution to both systems which are equal. However this only applies to linear programs, and the discontinuity in an integer programs feasible region breaks this guarantee. This means that, for linear programs, we are able to solve the primal problem through solving the dual problem. Furthermore, we also know that if we have objective solutions $\hat{x}$ and $\hat{y}$ from the primal and dual problems respectively , and if $\hat{\boldsymbol{y}} = \hat{\boldsymbol{x}}$, then they must be optimal solutions. This property along with the other corollaries can be exploited by solvers to obtain asymptotes and speed up computations.

Duality is also the basis upon which sensitivity analysis is conducted. The assumption made until this point is the fact that an optimiser has all the associated parameters needed in order to apply some algorithm like simplex to solve their problem. In reality however, this is often not the case, as they are more likely to be rough predictions. The classical method of approaching this is sensitivity analysis, often also referred to as post-optimality analysis [14, 44].

Sensitivity analysis deals with the following question: after solving an optimisation problem, how does changing the input parameters change our solution? It seeks to exploit knowledge from a current solution to obtain information about the new solution upon perturbation because, in large-scale operations, re-solving a problem can be costly or computationally prohibitive.

Sensitivity analysis is a potentially viable method of analysing and handling uncertainty in the data. However this is certainly not a preemptive approach. This certainly carries some flaws with it as we optimise without any assumption of uncertainty in the data. This does not give us any sort of guarantee in the face of perturbations. Detailed next are two

methods, Stochastic and Robust optimisation, which take into account the uncertainty in the model itself, and in certain ways *hedge* themselves against failures.

2.4 Stochastic Optimisation

In a nutshell, stochastic optimisation utilises random variables in its modelling to minimise both risks associated with perturbations and optimise the objective function. The motivation for this is as follows.

In a situation where there is uncertainty in the data, we would like the model to encapsulate this. Practically, it is not possible to make the best decision for all possible realisations of the uncertainty. A corresponding idea would be to optimise the expected value of our objective. Consequently, the problem is now split into two "stages". Firstly, the problem is hedged and solved against the expected value of all possible realisations of uncertainty. In the next stage, we realise our uncertainty, and we adjust for this. As such this is commonly referred to as 2-stage stochastic optimisation. A general model of this problem is presented below.

$$\begin{aligned} \min \quad & \boldsymbol{c}^T\boldsymbol{x} + E_\Delta[Q(\boldsymbol{x}, \Delta)] \\ \text{s.t.} \quad & \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \end{aligned} \tag{2.3}$$

Here, the symbol Δ represents the uncertainty associated in this problem, E_Δ is the expected value w.r.t to the uncertainty, and $\boldsymbol{c}, \boldsymbol{x}, \boldsymbol{A}$, and $\boldsymbol{b}$ represent the regular variables in an linear program problem as in (LP). In the context of stochastic optimisation, $\boldsymbol{x}$ represents the first-stage decision variables. The symbol Q represents a function which is composed of entries dependent of the realisation of Δ. It also represents the second stage, another optimisation problem which is to be solved now that we have the realisation. So the two steps can be written as

Algorithm 3 2-Stage Stochastic Optimisation

1: **procedure** OPTIMISE
2: Solve Equation (2.3)
3: Upon realisation of Δ, find $\boldsymbol{x}^*$ s.t. $Q(\boldsymbol{x}^*, \Delta)$ is minimal
4: **return** $\boldsymbol{x}$

Thorough and well explained introductory examples of modelling a stochastic problem can be seen in chapter 1 of Birge and Louveaux [7]. For a deeper reading of this topic, the readers are pointed towards the following Birge and Louveaux and Dantzig and Thapa [7, 15].

2.4.1 Sample Average Approximation and Probabilistic Constraints

Sample average approximation [27] is a method to discretise the random formulation of Equation (2.3). In practice it is very likely that the distribution of Δ, the random variable modelling uncertainty, is not known. In such situations it is not possible to compute $E_\Delta[Q(\boldsymbol{x}, \Delta)]$, the expectation of the problem with respect to the uncertainty realisations. Instead, a sampling approach can be used to approximate this value. For R samples of realisations of uncertainty Δ, we can approximate the expectation as follows.

$$E_{\Delta}[Q(\boldsymbol{x}, \Delta)] \approx \frac{1}{R} \sum_{i=1}^{R} Q(\boldsymbol{x}, \Delta) \tag{2.4}$$

Another stochastic modelling approach is probabilistic constraints, first introduced by Charnes et al. [10]. An example of this is as follows.

$$\begin{aligned} \min \quad & \boldsymbol{c}^T \boldsymbol{x} \\ \text{s.t.} \quad & prob(Q(x, \Delta) \leq 0) \leq \alpha \end{aligned} \tag{2.5}$$

While extensively studied, even linear formulations of the function Q might be "extremely difficult to solve numerically" due to the nature and complexity of multidimensional probabilistic integrals. Sample average approximation techniques can however be utilised, which provides convergence properties, and provides good candidate solutions. [32, 37]

2.5 Robust Optimisation

Robust optimisation is a relatively new and novel approach of dealing with uncertainty pre-emptively. The first formulations were proposed by Soyster [41] but only recently has it been fleshed out as a field. The term "robust" implies that it is hedged against uncertainty and solutions remain feasible under realisations of such uncertainty. This is much like stochastic optimisation, with some key differences.

Trait	Robust Optimisation	Stochastic Optimisation
Uncertainty	Modelled by discrete sets	Modelled by random variables
Worst Case	More conservative	Less conservative
Expressivity	Limited	Rich
Tractability	Easier	Harder

Table 2.1: Robust vs. Stochastic Optimisation [3, 44]

The main concepts of robust optimisation are defined and explained below. Parts of this section have been adapted from Ben-Tal et al. [3], whom the readers should consult for examples and a deeper reading. Let us take the original linear programming problem (LP), but assume that there exists some uncertainty in all the non-decision variables, i.e. $\boldsymbol{c}, \boldsymbol{A}$, and $\boldsymbol{b}$. Robust optimisation fundamentally works with an *uncertainty set*, Δ, that contains variations of the non-decision variables.

First the definition and construction of an uncertainty set is formalised. The original linear program (LP) can be compactly represented as the following.

$$\left[\begin{array}{c|c} \boldsymbol{c}^T & 0 \\ \hline \boldsymbol{A} & \boldsymbol{b} \end{array}\right] \in \mathbb{R}^{(m+1)\times(n+1)} \tag{2.6}$$

Each realisation of uncertainty, or a perturbation can be represented in a similar, compact manner as Equation (2.6) above, henceforth referred to as a perturbation matrix. These

are superscripted with ℓ. The number of perturbations is represented by R. Then, the uncertainty set can be defined as follows

$$\Delta = \left\{ \left[\begin{array}{c|c} \boldsymbol{c}^T & 0 \\ \hline \boldsymbol{A} & \boldsymbol{b} \end{array}\right] + \left[\begin{array}{c|c} \boldsymbol{c}^T & 0 \\ \hline \boldsymbol{A} & \boldsymbol{b} \end{array}\right]^{\ell} \in \mathbb{R}^{(m+1)\times(n+1)} \;\middle|\; \forall \ell \in \{1, \cdots R\} \right\} \tag{2.7}$$

Let us take a contrived example, where we are given the following data and perturbation matrices in (2.8) and (2.9) respectively. The red text is in the same position the data matrices. Matrix (2.9) then essentially says that the coefficient 2 in the first constraint is susceptible to a positive perturbation of magnitude 0.5.

$$\left[\begin{array}{ccc|c} 50 & 20 & -30 & 0 \\ \hline 5 & 2 & 4 & 1 \\ 2 & 2 & 2 & 300 \end{array}\right] \tag{2.8}$$

$$\left[\begin{array}{ccc|c} 0 & 0 & 0 & 0 \\ \hline 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right] \tag{2.9}$$

Thus, the uncertainty set has this rather loose and imprecise definition. The precision comes from the modeller, as their job is to rigorously define the R perturbation matrices and how they are constructed. For example, if it were to be modelled after statistical data, the perturbation matrices could be related to a confidence interval.

Thus, as should be evident now, the uncertainty set contains all permutations of perturbed matrix data caused by affine transformations of uncertainty on the nominal data. A variation of the original problem (LP) with the present uncertainty can be defined as in Equation (Uncertain LP).

$$\begin{aligned} &\min \boldsymbol{c}^T \boldsymbol{x} \\ &\text{s.t.} \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \\ &\quad \forall (\boldsymbol{c}, \boldsymbol{A}, \boldsymbol{b}) \in \Delta \end{aligned} \tag{Uncertain LP}$$

The final step to a robust optimisation formulation is to generate the *robust counterpart* of (Uncertain LP). A robust counter part is a model which defines the "best uncertainty-immunised solution" [3]. To do this we need to find the best objective value for the worst realisations of uncertainty. This can be thought of as a two-player game in a game-theoretic context. Thus the formulation is as follows in Equation (2.10)

$$c^* = \min_{\boldsymbol{x}} \left\{ \max_{(\boldsymbol{c},\boldsymbol{A},\boldsymbol{b})\in\Delta} \boldsymbol{c}^T \boldsymbol{x} : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \wedge \forall (\boldsymbol{c}, \boldsymbol{A}, \boldsymbol{b}) \in \Delta \right\} \tag{2.10}$$

2.5.1 Uncertainty Sets

It is vital to note that the method of constructing the uncertainty sets of the robust counterpart is non-trivial and it dictates the tractability of the problem. As is stated by Ben-Tal et al., "the robust counterpart of the [Uncertain LP] problem with uncertainty set Δ is computationally tractable whenever the convex uncertainty set Δ itself is computationally tractable" [3].

Firstly, it is safe to take Δ as both convex and closed. Given a robust feasible solution to a problem, it can be shown that its convex hull still preserves the constraints. The same

applies for the set closure. The interested reader is directed towards Berg et al.[4] and Ben-Tal et al.[3] for a deeper reading on this.

Although an uncertainty set's structure dictates a robust counterpart's tractability, it is counter-productive to define a highly simplistic model for the uncertainty sets in favor of tractability, as it forgoes expressivity. Described below are some common constructions of Δ. [9]

First, there is the rudimentary *discrete* construction where the uncertainty set is just a set of with several unrelated scenarios. A finite list of scenarios are generally mathematically unrelated and with no real structure, must be computed through brute force enumeration. This is $\mathcal{NP}$-Hard.

Next is a *polytopal* construction. Polytopes are n-dimensional generalisations of a polyhedral, i.e., an object with flat sides. As Buchheim and Kurtz state, it might be the most intuitive to specify discrete scenarios, but there is no reason not to include convex combinations of these scenarios [9]. In doing so, we are providing some structure to the unstructured discrete uncertainty sets, which is the same as a convex hull. Furthermore, convex hulls of Δ for the RC of a problem does not modify the solution [9].

This property suggests an equivalence between discrete and polytopal uncertainty sets. While technically true, it is not so in the perspective of complexity theory. A convex hull of a set of m points can have a exponential vertices in the number of facets in m.

An interval based uncertainty i.e. for some $x \in [l, u]$, is an intuitive way of modelling uncertainty, as is done so in statistical interval estimates (e.g. confidence intervals). These also possesses a polytope with an exponential number of vertices, but the complexity of such a model can be avoided. As per Equation (2.10), the worst case is picked in the uncertainty set. For constraints $\boldsymbol{Ax} \leq \boldsymbol{b}$, this means picking a bound in the interval. Thus, by picking the worst case values of each interval, it is now much easier to solve. A consequence of the simplification is that the model is now too conservative. This is further explicated in Section 2.5.2.

The final common construction of uncertainty set is an *ellipsoid*. An ellipsoid is a sphere without the need for equal radii. This generally refers to a 3-dimensional ellipsoid, but can also be generalised to any n dimensions. A higher dimensional distribution's confidence interval takes the form of ellipsoids. There are varying methods of modelling such problems, and some cases are tractable. However, in general, ellipsoidal uncertainty sets lead to intractable counterparts [9].

2.5.2 Conservatism and Relevant Work

Classical robust optimisation hedges against predefined uncertainty sets, forbidding solutions that extend beyond it. Equivalently, this means that decisions are being made before the uncertainty is realised [2]. Thus, by design, a compromise is reached in the model. Sub-optimal results are accepted with the guarantee that the solution remains feasible when the uncertainty is realised [6]. This results in conservative, less than optimal solutions. Discussed here are prevalent methods of avoiding this trap.

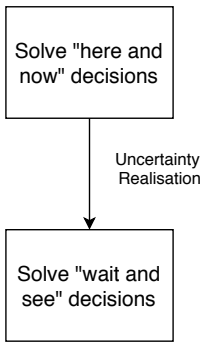


Figure 2.5: A depiction of the basic premise of 2-Stage Robust Optimisation

The first method is a reformulation of the problem into a two stages. It should be evident that most situations which require optimisation under uncertainty should not occur in a manner as described above. The problems associated variables can be separated into two distinct categories. As Ben-Tal et al. [2] have put it, there are the "here and now" variables, and the "wait and see" variables.

To this end, robust optimisation problems can be split into two stages, similar to what happens in stochastic optimisation. Introduced by Ben-Tal et al. This is termed 2-Stage robust optimisation and there are multiple formulations of this problem [2, 5, 9], and for both simplicity and clarity's sake, a diagram depicting the process in abstract is presented in Figure 2.5. This split allows for more flexibility in the second stage, as the 2-stage problem has a larger feasible set compared to the single stage problem [2]. This naturally results in less conservative solutions, except for certain special cases when the two problems are identical.

Bertsimas and Sim [6], on the other hand, propose the idea of budgeted uncertainty sets to deal with this conservatism. Here a budget constraint is used in polytopal interval uncertainty sets to limit the amount of uncertain variables that differ from their nominal value. This allows one to tune the level of conservativeness in the model. They also propose a simple metric, that we term the price function. This function is defined as follows

$$\text{price function} \quad \triangleq \quad \frac{v^*_{new}}{v^*_{best}} \tag{2.11}$$

The idea of the price function is that it is a measure of optimal vans required by a schedule with robust optimisation techniques with respect to one with none, i.e. v^*_{best}. This is a simple measure of model conservatism.

Nohadani and Sharma [35] propose an endogenous method of defining uncertainty sets, as opposed to the predetermined, exogenous method of doing so discussed thus far. Here uncertainty sets are *decision dependent*, in a one-stage robust optimisation program. The "decision" which the uncertainty set is dependent on is the "here and now" decisions as in 2-stage robust optimisation. Roughly, the formulation of the uncertain linear program with decision dependent uncertainty sets is as follows, noting the parametrisation of the

uncertainty set Δ.

$$\begin{aligned} &\min \boldsymbol{c}^T \boldsymbol{x} \\ &\text{s.t.} \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \\ &\forall (\boldsymbol{c}, \boldsymbol{A}, \boldsymbol{b}) \in \Delta(x) \end{aligned} \tag{2.12}$$

The "proactive uncertainty control" of parametrised decision dependent uncertainty helps alleviate the over conservatism of robust optimisation as in Equation (2.10).

Fischetti and Monaci [18] propose a new method of modelling robust optimisation termed *Light Robustness*. In this approach, instead of guaranteeing robustness, it models the problem such that we find solutions which are "not too far from optimality". This is done by doing two things. First, a deterioration limit for the objective is determined and the objective is moved to the constraints. The objective cannot be worst than the deterioration limit. Next, as an objective we use measures to represent how far the solution is from optimality, or feasiblity.

2.5.3 Recourse

A 2-stage optimisation problem is a "tri-level optimisation model" that presents tractability issues. It is often represented as a three player game. First the optimiser decides on the best "here and now" variables. The adversary then picks an uncertainty realisation, presumably the worst. Finally, the optimiser then picks the best "wait and see" variables. The last set of decisions is known as *recourse*. [45]

An entirely linear formulation of a 2-stage robust optimisation problem is still computationally complicated. Ben-Tal et al. [2] propose a formulation where recourse is some affine transformation of uncertainty, and thus compacting the problem down into a single-stage robust problem, which is a simpler problem. Zhao and Zeng [45] propose a method of recourse for mixed integer programs.

2.6 Recoverable Robustness

Liebchen et al. [31] propose a new set of ideas as an alternative to both classical robust and stochastic optimisation. There exist a set of situations where neither methods are appropriate. Certain classes of problems have distributions too large for stochastic programming (i.e. for tractability reasons) and cannot form too conservative solutions, which classical robust optimisation necessitates by being feasible in all possible scenarios.

The idea of recoverable robustness is roughly as follows. It seeks to obtain solutions to a problem, which in a constricted set of uncertainty realisations, can be recovered with a limited amount of effort, into feasibility. The process of recovery is defined in a looser manner compared to recourse, defined as $\mathcal{A}$, a class of admissible recovery algorithms. To formulate a recovery robust program, let us define the feasible solution space as P. For any scenario of uncertainty $s \in \Delta$, the set of feasible solutions in scenario s is P^s.

$$\begin{aligned} &\min_{(\boldsymbol{x}, \boldsymbol{A}) \in P \times \mathcal{A}} \boldsymbol{c}^T \boldsymbol{x} \\ &\text{s.t.} \quad A(x, s) \in P^s \quad \forall s \in \Delta \end{aligned} \tag{2.13}$$

To limit the capabilities of $\mathcal{A}$, there are two sensible restrictions proposed. First, and most obviously, is the computational complexity. Next is a limit on the actions of recovery. This is to say, if in a resource allocation problem, the allocation of two resources were to be swapped around to recover, this action comes with a *cost*. The total *cost of recovery* should be limited. Another consideration here is that it is in some cases imperative to stay close to the original planning first stage solution. The costs of recovery can be either fixed or part of the recovery problem itself.

Liebchen et al. [31] use these ideas to further formulate recovery algorithms in $\mathcal{A}$ as linear programs. Here the idea is to minimise both costs of recovery while aiming for feasible solutions.

2.7 Lexicographical Optimisation and Recovery

The *makespan* of a schedule is the time from the start to the end of all processes. It is a common measure in scheduling problems [8]. The makespan minimisation problem we concern ourselves with (informally, the *makespan problem*) is one as follows. Given are some number of jobs i in the set I, each with some processing duration d_i, and, a set of parallel and identical machines J. The goal is to minimise the processing duration required to complete all jobs in I. More formally, the problem is given in equations (2.14) to (2.18) below. First we detail the notation used in this section in Table 2.2

Symbol	Description
I	The index set of all jobs
J	The index set of all machines
$y_{i,j}$	A binary decision variable which is true when a job i is processed by machine j.
C_j	The completion time for some machine j.
C_{max}	The longest completion time amongst all machines in J.
d_i	This is the time it takes to process i.

Table 2.2: Table Detailing the Bin Packing Problem Notation

$$\min_{C_{max}, C_i, y_{i,j}} \quad C_{max} \tag{2.14}$$

$$C_{max} \geq C_i \qquad \forall j \in J \tag{2.15}$$

$$C_i = \sum_{j=1}^{n} y_{i,j} \cdot d_i \qquad \forall j \in J \tag{2.16}$$

$$\sum_{i=1}^{m} y_{i,j} = 1 \qquad \forall i \in I \tag{2.17}$$

$$y_{i,j} \in \{0, 1\} \qquad \forall j \in J . i \in I \tag{2.18}$$

(2.14)	The objective is to minimise the makespan C_{max}.
(2.15)	The makespan is defined as the maximum of completion times.
(2.16)	The completion time of a machine is defined as the sum of processing duration of the jobs executed on that machine.
(2.17)	A job can only be executed by one machine at most.

Lexicographical Optimisation (henceforth *LexOpt*) is a multi-objective optimisation approach. In such approaches, the goal is to minimise multiple objective functions simultaneously. In LexOpt, multiple objective functions are minimised while preserving a lexicographical ordering. Conceptually, they are ordered in a descending order of priority.

Letsios and Misener [29] have shown that the LexOpt makespan problem, coupled with the Longest Processing Time recovery method, has certain strong guarantees. Effectively, they codify this lexicographical ordering into the makespan problem described above by considering each machine's completion time as an objective function. Then the optimisation problem can be recursively thought of as follows.

The first objective function is optimised as per usual. For the second objective function, we look for an optimal solution where the first objective's value remains the same. Recursively, for the n^{th} objective function, we find an optimal solution with the fixed values defined for the first $n-1$ objective functions.

2.7.1 Computing LexOpt Schedules

From the above formulation, the reader should reasonably question how the recursive definition of lexicographical optimality is to be solved. There are three methods of *scalarisation* of this recursive definition presented in the aforementioned paper. Discussed below is the *weighting* method.

$$\text{lexmin} \quad C_1, \cdots, C_{|J|} \quad \approx \quad \min \left\{ \sum_{i=0}^{|J|} 2^{|J|-i} C_i \right\} \tag{2.19}$$

Here *lexmin* refers to the recursive definition of the optimisation problem discussed above. The scalar can also be interchangeably be thought of as a measure of lexicographical optimality, the lower the better.

2.7.2 Longest Processing Time Recovery

The Longest Processing Time (LPT) algorithm, a common approximate heuristic approach, is the recovery algorithm used for the LexOpt makespan problem above. The strong guarantees of this are discussed in the next part. Here we describe the workings of this algorithm.

To do this, we first make a distinction between two types of jobs in a schedule here. The first type of job, remains feasible even after a schedule has been perturbed by uncertainty,

Algorithm 4 LPT Recovery

1: **procedure** RECOVER
2: 1. Assign all decisions with respect to the original schedule that are feasible.
3: 2. Sort all variables that are no longer feasible in descending order with respect to processing duration
4: 3. Assign above variables to the processor with the least work assigned to it.

whereas in the second type, their original assignment breaks some constraint. The algorithm is then detailed in Algorithm 4 below.

The algorithm is clearly in $\mathcal{P}$, as opposed to the optimisation problem itself, which is an integer problem thus $\mathcal{NP}$-hard. The LPT recovery strategy maintains all decision variables that are still feasible and only assigns those that don't decisions. From a glance, this already proves beneficial, as relevant information from the original optimal schedule is being exploited to remain as close as possible to it in the new solution.

2.7.3 Strong guarantees

There are multiple strong guarantees of the LexOpt makespan problem with LPT recovery that have been proven by Letsios and Misener[29]. For example, a lexopt makespan solution that experiences one perturbation, when recovered, will have at worst two times the best optimal solution for the perturbed schedule.

There are strong guarantees for each type of perturbation defined in the above paper. This includes both contractions and expansions in processing time, and machine activations and deactivations. For a further reading on this, the reader is pointed toward the aforementioned paper.

2.8 Bin Packing Problem

The bin packing problem, as originally proposed by Johnson [26], is an ubiquitous one which aims to fit a set of weighted items into as little fixed weight bins as possible.

The formal definition of this problem is as follows. Given is an infinite amount of bins $b_1, b_2, \cdots$ of the unit size and a finite list of items with indices $1, \cdots, m$ and sizes $a_1, \cdots, a_n$, where all items are of some size $0 \leq a_i < 1$, the goal is to find a mapping of items to bins which minimises the number of bins used. A most primitive formulation of this can be mathematically represented as a linear program. First, the notation is detailed in Table 2.3, before the problem is defined.

Symbol	Description
e_j	A binary decision which is set to true when bin j has some item in it, i.e. in use.
$y_{i,j}$	A binary decision variable which is set to true when item index i is put into bin j.
d_i	This is the weight of an item i. For the traditional bin packing problem, $0 \leq d_i < 1$.

Table 2.3: Table Detailing the Bin Packing Problem Notation

$$\min \quad \sum e_j \tag{2.20}$$

$$\text{s.t.} \quad e_j \geq \sum_i d_i y_{i,j} \tag{2.21}$$

$$\sum y_{ij} = 1 \tag{2.22}$$

$$y_{i,j}, e_j \in \{0, 1\} \tag{2.23}$$

(2.20)	The objective is to minimise the number of bins used.
(2.21)	The binary variable is set to be larger than the total sum of weights in bin j. Conveniently, each item's weight is a fraction, thus with any item in a bin and the binary constraint, this value is set to 1.
(2.22)	We are only able to assign one item to one bin, i.e. we cannot break an item down into smaller sub items.
(2.23)	The decisions variables are binary.

2.8.1 Generalised Bin Packing Problem

Baldi et al. [1] generalise the above notion and other common variations of bin packing problems. In the Generalised Bin Packing Problem (*GBPP*), there exists two classes of items, one are compulsory (i.e. must be assigned a bin) and the other is not. Each item has an associated weight and profit. There also exists an other set of bins with associated weights and costs.

In such a problem, we must assign the compulsory items, and are free, but not obliged to assign non-compulsory ones. The objective here is to maximise the *net profit*, i.e. the total profit of the items less the total costs of the bins. The notation used for the GBPP is detailed in table 2.4 below.

Symbol	Description
I	The index set of all itineraries. $I = I^{\mathcal{C}} \cup I^{\mathcal{NC}}$
$I^{\mathcal{C}}$	The index set of all compulsory itineraries
$I^{\mathcal{NC}}$	The index set of all non-compulsory itineraries $I^{\mathcal{NC}} = I \setminus I^{\mathcal{C}}$
J	The index set of all bins
$\tau, \tau(.)$	Let τ be overloaded to mean both the set of bin types, and a function that takes a bin index in J and returns its bin type
L_t, U_t	For some type $t \in \tau$, defined are the lower and upper limits of the number of the corresponding bin type
U	The total number of available bins of any type such that $U \leq \sum_{t \in \tau} U_t$
d_i	The weight of item $i \in I$
π_i	The profit associated with item $i \in I$
W_t	The total weight assignable to a bin of type $t \in \tau$
Π_t	The cost associated with bin of type $t \in \tau$

Table 2.4: Table Detailing the Generalised Bin Packing Problem Notation

We then define the GBPP as follows.

$$\min \quad \sum_{j \in J} \Pi_{\tau(j)} \cdot e_j - \sum_{j \in J} \sum_{i \in I^{\mathcal{NC}}} \pi_i \cdot y_{i,j} \tag{2.24}$$

$$\text{s.t.} \quad \sum_{i \in I} d_i \cdot y_{i,j} \leq W_{\tau(j)} \qquad \forall j \in J \tag{2.25}$$

$$\sum_{j \in J} y_{i,j} = 1 \qquad \forall i \in I^{\mathcal{C}} \tag{2.26}$$

$$\sum_{j \in J} y_{i,j} \leq 1 \qquad \forall i \in I^{\mathcal{NC}} \tag{2.27}$$

$$\sum_{j \in \{j' | \tau(j') = t\}} e_j \leq U_t \qquad \forall t \in \tau \tag{2.28}$$

$$\sum_{j \in \{j' | \tau(j') = t\}} e_j \geq L_t \qquad \forall t \in \tau \tag{2.29}$$

$$\sum_{j \in J} e_j \leq U \tag{2.30}$$

$$e_j, y_{i,j} \in \{0, 1\} \qquad \forall i \in I . j \in J \tag{2.31}$$

(2.24)	Here we seek to minimise the net cost of the problem. This is defined by the total cost of the active bins less the total profit of the selected non-compulsory items.
(2.25)	We make sure that in the assignment of items to bins, a bin only has as many items as it can fit.
(2.26)	For all compulsory items, it must only be assigned to one bin.
(2.27)	For all non-compulsory items, it must either be assigned to strictly one bin, or not be assigned at all.
(2.28)	The number of bin of a certain type that are active must not be larger than the upper bound.
(2.29)	The number of bin of a certain type that are active must not be smaller than the lower bound.
(2.30)	The total number of bins used must not be more than the total number of bins available.
(2.31)	These are binary decision variables.

The above formulation generalises multiple classical packing problems. Such problems include the Variable Size Bin Packing Problem (VSBPP), the Variable Cost and Size Bin Packing Problem (VCSBPP) and the Knapsack Problem [11, 19].

2.8.2 Generalised Bin Packing Under Uncertainty

The problem of Generalised Bin Packing Under Uncertainty is not an intensively studied one. To the best of our knowledge, there is only one set of related papers that explore this idea.

Tadei and Perboli and Perboli et al. [39, 42] propose this notion in a paper, with a stochastic formulation. These works propose a deterministic approximation of stochastic formulations of the GBPP.

However, this is quite limited in scope as the authors only consider an uncertainty in the profit of items to be assigned. This notion of uncertainty studied is not truly generalised in a similar fashion that the GBPP itself is a generalisation of the bin packing problem. This is because their notion of uncertainty is tangential to fixed cost instances of the GBPP, e.g. the VSBPP. That is to say the way uncertainty is realised in the current literature does not apply to problems like the VSBPP. Thus, we also make note that the VSBPP, with scalar item weights (i.e. on dimensional), under uncertainty is not well studied in the literature.

2.9 Robust Statistical Analysis

A rather unfortunate collision of terminology, the field of robust statistics has little to do with robust optimisation. Robust statistics deals with obtaining the "true underlying distribution" [25] of "high quality data obtained with the greatest care" as it will more often than not contain "gross errors" [23], as will any real data.

Such "gross errors", or henceforth *outliers*, should ideally only create a small error with our statistical measures or estimators. However such well conditioned behavior is not always the case. The three main tenets of a robust statistical model are as follows [25].

- It should have a reasonably good (optimal or nearly optimal) efficiency at the assumed model.
- A small deviation from the model assumptions should only give rise to a proportional and small error.
- Larger deviations from the model assumptions should not result in a situation where the new model is radically different.

The mathematics involved in this field is dense and relatively unrelated to this dissertation, and as such the interested reader should consult Huber [25] for a deeper reading. However, some key aspects of this field are important and shall be discussed further.

1. The Breakdown Point refers to the maximum number of outliers that a statistic can deal with before it "gives way", or a measure of robustness of a statistic. A sample mean, for example, would be affected by a single outlier. A median (which is robust), on the other hand, is not affected by one outlier observation. It would instead require half the data to move in order for it to be affected.
2. The median absolute deviation (MAD), is a robust estimator of deviation. It measures the median of absolute deviations from the median. For some data set $X_1, \cdots X_m$as defined

$$MAD = \text{median}\left\{|X_i - \text{median}\{X_i\}|\right\}$$

3. Trimmed means are a method of describing central tendency, however, it discards the extremities of a distribution, e.g. top and bottom 5%. This proves to be a quick and easy robust statistic with certain obvious flaws, but for the purposes of this dissertation, will be sufficient.

Chapter 3

Initial Modelling

3.1 Motivation

The first step in the process of answering the problem statement at hand is its practical and theoretical contextualisation. To be able to use the data from Royal Mail, we must first define methods to clean it of errors. After doing this, we need to analyse and critique the original model as proposed by Page [36]. This step is highly necessary as it is, in essence, a gateway to the rest of the problem.

3.2 Data Format

To reiterate, Royal Mail have provided data from three different delivery offices A, B, and C. This data in essence provides several entries of the time at which a van leaves and returns back to the delivery office (DO) along with some extraneous information. Each entry shall be henceforth termed an *itinerary*.

3.3 Delivery Prerequisites and Assumptions

This and the next sections are adapted from Page [36]. Each day, there is the need for mail to be both sorted and delivered. The delivery of the mail is done within a time horizon

$$[start_{abs},\ end_{abs}] \qquad \text{(Time-Horizon)}$$

The sorting is assumed to be completed prior. Each van performs some work throughout the day delivering mail before returning to the DO. This corresponds to an itinerary.

The exact way that this is done will vary across DOs. A van might be occupied by two persons who might complete two separate itineraries. However, it is assumed that a single van is associated with one itinerary. This assumption is made so as to be able to coherently work with the data provided, which only provides van arrival and departure times. More generally, a consistency assumption is made in the approach to delivery, for example, if two (or more) workers were to share a van it would happen consistently and not sporadically. An assumption such as this, which would be easily justifiable on a large scale, would make any forthcoming conclusions resistant to varied uses of vans.

To deal with representing time in this problem, a discrete time formulation is used. This means that time is represented as *timeslots* in the set $S = \{1 \cdots T\}$. The set S is essentially (Time-Horizon) made discrete, into T equal timeslots. As the value of T tends to infinity, this models real life more precisely.

3.4 Data Cleaning

Here, methods for cleaning the data are proposed. The goal of the cleaning phase is to deal with erroneous or anomalous data. It is best to do so such that our observations are not coloured by anomalies, which could subsequently colour our conclusions. The entire process can be visualised as Figure 3.1

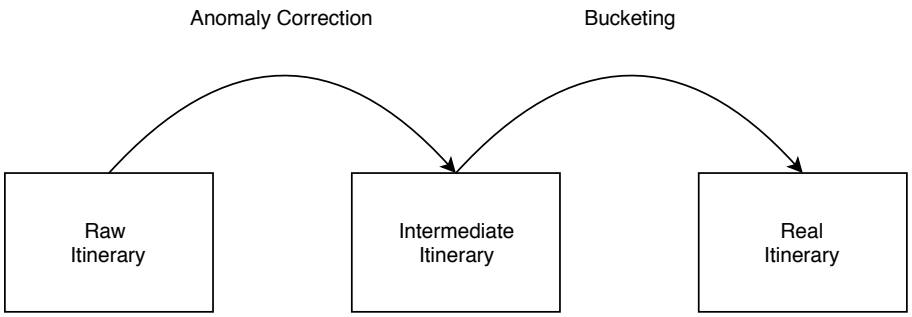


Figure 3.1: The Data Cleaning Process Visualised

Raw itineraries are entries in the data provided. *Intermediate* itineraries are the raw itineraries once they have undergone *anomaly correction* (discussed in Section 3.4.1). Finally these intermediate itineraries undergo the *bucketing* process upon which it is a *real* itinerary (discussed in Section 3.4.2). These terms are further explained in Table 3.2. Furthermore, the multiple parameters used have both their definitions and their values picked in Table 3.3.

3.4.1 Anomaly correction

First, it is necessary to classify which itineraries stand out. It is generally the case that people work from roughly 9 a.m. to 5p.m., with flexibility in starting times. Work hours usually last anywhere from 7 to 9 hours. Anomalous itineraries shall be classified into 5 categories as described in Table 3.1 below. The anomalies in this table have been characterised with respect to certain parameters.

The following steps shall be taken to clean the data so as to account for these anomalies.

1. `no_work` itineraries are disposed of. These are taken to be erroneous sensor data.
2. `late_work` itineraries are left as it is. While it is not favourable for the vans to be out this late, the arrival time does not tell us about the amount of work done. It is entirely possible that a worker had a shift which was later than usual.
3. `long_work` itineraries are to be broken up into smaller instances. The size of these *shards* are parametrised by `shard`. The itinerary is broken into multiples of this parameter and some remainder itinerary. These types of itineraries are assumed

Anomaly	Description
`no_work`	When the difference between arrival and departure time in hours is smaller than ϵ, for some small positive ϵ. This categorises vans that leave and enter the DO withing a short time period, signifying no work done or faulty sensor data.
`late_work`	When the hour of arrival is later than a user defined parameter, `late_hour`. This categorises vans that arrive later than an acceptable time.
`long_work`	When the number of hours between departure and arrival is larger than a user defined parameter, `long_delta`, but lesser than the next parameter associated with `abuse_work`. This represents vans that have been working for longer than acceptable, as specified by union hours, but not an abuse of the van.
`abuse_work`	When the number of hours between departure and arrival is larger than a user defined parameter, `abuse_delta`. This represents, for lack of a better term, the "abuse" of a van by being taken out of the DO for an inordinate amount of time. For example a van could have been taken home overnight instead of being returned to the DO in a timely fashion.
`short_work`	When the number of hours between departure and arrival is lesser than a user defined parameter, `short_delta`. This represents vans which have worked for, what might be perceived as, shorter than usual amount of time.

Table 3.1: Type of data anomalies

to be meaningful work that has been conducted over an amount of time which is unacceptable (e.g. longer than union working hours).

4. `abuse_work` itineraries are to be capped into one itinerary of a full length shift itinerary, which is parametrised by `regular_hours`. While this might might seem quite liberal, the underlying assumption is one which presumes that meaningful work has only been carried out over the regular shift, after which the van has not been promptly returned to the DO. The parameter `regular_hours` is taken to be 9 hours, which is usually the length of a working day.

5. `short_work` itineraries are hard to define exactly. Since there are overheads associated with the entire process, e.g. getting the van out and time on the road, it might be unfair to reintroduce the entire itinerary back into the process. Since it is of a short period of time, the overhead, if there was to be any, would be a large proportion of the itinerary. To deal with this we introduce the parameter `short_work_overhead` in hours. The itinerary will be reintroduced after cleaning with `short_work_overhead` less time.

Apart from all of these measures, itineraries occurring on Sunday are disposed of. Sundays are meant as days off. There are instances where a van or two is in use but these are thought of as an exception and not the rule, as per the consistency assumption. After these processing steps are applied, we have intermediate itineraries (IR) and we proceed to the bucketing phase.

3.4.2 Bucketing

The notation and terms used in for process are detailed in Table 3.2.

In this section, the IRs are to be converted into real itineraries of the deterministic model. This involves transforming times in $\mathbb{T}$ to something in the set S. This process will be

henceforth termed *bucketing*.

Symbol	Criteria
$\mathbb{T}$	Space of all times in a day, for example 4.30 p.m.
Raw Itinerary	An entry in the data from Royal Mail. This itinerary has its start and end times in $\mathbb{T}$
Intermediate Itinerary	Raw itineraries post anomaly corrections. This itinerary still has its start and end times in $\mathbb{T}$
Real Itinerary	An itinerary whose start and end times are in S^2.
$(start_r, end_r) \in \mathbb{T}^2$	"Real" start and end times. This represents an intermediate or raw itinerary's actual start and end times.

Table 3.2: Table Detailing Bucketing Terminology and notation

The actual values of $start_{abs}$ and end_{abs} do not terribly matter. Instead we define the parameter `work_hours`, which is in essence the value of $(end_{abs} - start_{abs})$. The process after this is relatively simple. Given an itinerary and its real times, its duration is calculated. This duration is in seconds, which should be converted to some timeslot in the set S, which can be conceptualised as "buckets". The specifics of this is detailed below in equations (3.1) to (3.4).

$$b = \frac{\texttt{work\_hours} \cdot 60^2}{T} \tag{3.1}$$

$$d = \text{seconds}(end_r - start_r) \tag{3.2}$$

$$x = \lceil \frac{d}{b} \rceil \tag{3.3}$$

$$x \in S \tag{3.4}$$

(3.1)	Each time slot in the set S can be conceptually thought of as a bucket. b represents the size of this bucket in seconds.
(3.2)	The variable d represents the total duration of the itinerary in seconds.
(3.3)	The variable x is the number of buckets the itinerary fills. I.e. its duration in terms of the discretised time set. We remain conservative by taking the ceiling of the value. Once again, as the value of T tends to infinity, the value of x would also tend towards a true duration.
(3.4)	This duration must exist as a value in the discretised time set.

3.5 Parameterisation

As should be evident, the process of data cleansing involves picking certain parameter values. At this stage we will pick parameter values which are the most intuitive. However, at a later point it is imperative to analyse how sensitive our conclusions are to these parameters. The parameters used up to this point are detailed in the table below.

Parameter	Value	Description
T	50	The granularity of the discretisation of the time horizon, i.e., the number of timeslots in the time horizon.
$(start_{abs}, end_{abs}) \in \mathbb{Z}^2$	(5, 15)	Absolute values for the time horizon. All deliveries must start and end between these. These are defined as hours from midnight. For example if deliveries were to happen from 10am to 5pm, the pair would be defined as (10, 17).
`work_hours`	10	The number of hours the work day is to be. This is the value of $(end_{abs} - start_{abs})$. This is the time horizon in hours.
`short_work_overhead` $\in \mathbb{Z}$	0	The amount of time in hours to be removed from an itinerary classified as a `short_work` anomaly
`epsilon` $\in \mathbb{Z}$	0.005	An upper bound for the amount of time in hours to be classified as a `no_work` anomaly
`late_hour` $\in \mathbb{Z}$	22	The lower bound for the amount of time in hours to be classified as a `late_work` anomaly
`long_delta` $\in \mathbb{Z}$	9	The lower bound for the amount of time in hours to be classified as a `long_work` anomaly
`abuse_delta` $\in \mathbb{Z}$	11	The upper bound for the amount of time in hours to be classified as a `long_work` anomaly and a lower bound for an itinerary classified as a `abuse_work` anomaly
`short_delta` $\in \mathbb{Z}$	4	The upper bound for the amount of time in hours to be classified as a `short_work` anomaly
`regular_hours` $\in \mathbb{Z}$	9	Defined in hours, this is how long an itinerary is truncated down too if it is an `abuse_work` anomaly.
`shard` $\in \mathbb{S}$	4	Defined in hours, this is the maximum size of the new itineraries constructed when an itinerary is a `long_work` anomaly.

Table 3.3: Anomaly correction parameters

3.6 Original Model

The main problem lies in minimising the number of van required per DO. However, there are various restrictions on the exact method in which itineraries can be executed. The restrictions are enumerated along with the model in this section.

The original model is entirely the work of Page [36], and shall henceforth be referred to as the *deterministic model*. The only part of their thesis accessed was the model as arranged by the dissertation supervisor. Since prior communication and collaboration has already gone into constructing this model, it seemed wasteful to start from scratch. Do note that this dissertation's approach to uncertainty analysis is unrelated to the aforementioned thesis.

The problem is to be modelled as an integer program. Presented below are two models.

The second model incorporates the optional restriction on when vans can leave a DO. Before describing these models, its notation used is explained in Table 3.4 below.

Symbol	Description
T	The granularity of the discretisation of the time horizon, i.e., the number of timeslots in the time horizon. As T tends to infinity, it models the time horizon more accurately.
S	Discretised time set where $S = \{1 \cdots T\}$.
I	The set of all itineraries.
d_i	The duration of each itinerary. Values of this correspond to how many discrete time slots of some itinerary takes up.
g	Upper bound on the number of vans that can physically leave the DO at any given time slot. This is a predefined value. For example, if it were to be desired to represent something along the lines of "only 1 van can leave the DO per minute", this could be done by $g = \lfloor(\texttt{work\_hours} * 60)/(T)\rfloor$. The parameter `work_hours` is described in Table 3.2.
$x_{i,s}$	A binary decision variable that indicates if itinerary i has begun at some time slot $s \in S$
$S_{i,t}$	Defined as the time slots in which itinerary i must have begun in order to be being executed at time slot t. Of the form $S_{i,t} = \{t - d_i + 1, t - d_i + 2, \cdots, t\}$.
F_i	The set of time slots in which itinerary i may begin in order to still be completed by the deadline (i.e. the end of the time horizon). Of the form $F_i = \{1, 2, \cdots, T - d_i + 1\}$.
E	The optional constraint - the time after which no van is allowed to leave.
S_e	The set of time slots in which the vans are not allowed to leave. Of the form $S_e = \{E + 1, \cdots, T\}$.

Table 3.4: Table Detailing the Deterministic Model Notation

3.6.1 Deterministic Model 1

$$\min \; v \tag{3.5}$$

$$\text{s.t.} \quad \sum_{i \in I} \sum_{s \in S_{i,t}} x_{i,s} \leq v \qquad \forall t \in S \tag{3.6}$$

$$\sum_{s \in F_i} x_{i,s} = 1 \qquad \forall i \in I \tag{3.7}$$

$$\sum_{i \in I} x_{i,t} \leq g \qquad \forall t \in S \tag{3.8}$$

$$x_{i,t} \in \{0, 1\} \qquad \forall i \in I, t \in S \tag{3.9}$$

$$v \in \mathbb{N} \tag{3.10}$$

The problem is summarised as follows:

(3.11)	The objective is to minimise the number of vans v.
(3.6)	At each time slot the number of vans used is not more vans than that which we own
(3.7)	Each itinerary has exactly one starting time that allows the driver to deliver the mail on time
(3.8)	No more than g vans can leave the DO in any given time slot. This is henceforth referred to as the *gate constraint*.
(3.9)	These are binary decision variables
(3.10)	These are positive integers

3.6.2 Deterministic Model 2

Is the same as the first model, but with an added constraint of the latest slot a van can depart the DO.

$$\min \quad v \tag{3.11}$$

$$\text{s.t.} \quad \sum_{i \in I} \sum_{s \in S_{i,t}} x_{i,s} \leq v \qquad \forall t \in S \tag{3.12}$$

$$\sum_{s \in F_i} x_{i,s} = 1 \qquad \forall i \in I \tag{3.13}$$

$$\sum_{i \in I} x_{i,t} \leq g \qquad \forall t \in S \tag{3.14}$$

$$\sum_{i \in I} \sum_{s \in Se} x_{i,s} = 0 \tag{3.15}$$

$$x_{i,t} \in \{0, 1\} \qquad \forall i \in I, t \in S \tag{3.16}$$

$$v \in \mathbb{N} \tag{3.17}$$

The added constraint is (3.15). No van can leave after the timeslots in S_e. In this dissertation, the first model will be used. We assume this extension was provided solely for an optimiser to potentially extend their modelling. This constraint does not model anything particularly of value for us.

3.6.3 Packing and visualisation

The model above computes a feasible solution with the decision variables $x_{i,s}$. Some effort must go into constructing a timetable with these decision variables that do not indicate which van uses which itinerary. However, due to the model, we know that the decision variables spell out a *feasible* solution. This means that if an itinerary starts at some prescribed time, it is not going to exceed the time horizon or behave in any way that causes infeasibility. The method for constructing the time table is detailed in this section.

First, we create v empty vans (i.e. the optimal value from deterministic model (3.6)) with no itineraries assigned. Next we extract all decision variables which are set to 1. This subset of decision variables tell us the time slot at which the itinerary begins. We also have prior information on the length of each itinerary. The itineraries are sorted by start times in ascending order. Then we proceed as follows in Algorithm 5.

Algorithm 5 Greedy Pack Algorithm

```
1: procedure PACK(itineraries)
2:     for itinerary i in itineraries do
3:         find van j that is both
4:         1. Not assigned time T worth of itineraries
5:         2. Has total duration of itineraries assigned lesser than i's start time.
6:         j assigned itinerary i
7:         j.duration += i.duration
       return vans
```

Though this approach might seem like an approximate greedy method, it exploits the feasibility of the solution to produce a feasible optimal schedule. By sorting prior to the assignment, we always assign jobs in ascending order. And as such there will always exist a slot for it to be assigned to. It is just a matter of finding it.

The internal representation of the schedule is simple. Each van is an array, with two elements, the first is the duration to execute all itineraries (including any breaks or gaps in between) and a list of the itineraries themselves. Each itinerary contains three parameters, the original itinerary number, the start bucket, and, the duration. With this, the schedules can also be easily visualised. An example of this is in Figure 3.3.

3.7 Remodelling

It should be evident that there are three properties that are critical in solving the scheduling problem for Royal Mail, namely the vans, the itineraries and the timeslots. To elaborate, the deterministic model solves the problem with respect to binary decision variables $x_{i,s}$, which is to do with itinerary i and timeslot s, and encodes whether the itinerary begun at that timeslot. This does not model the relationship with vans, that is, we do not know which van executes which itinerary. The packing process (Section 3.6.3) does this as a post-processing step, rather than as part of the model itself. It would be necessary to model the relationship with vans in order to design schedules better. This section introduces two models, the *unified* and the *approximate* model.

The unified model incorporates all three properties: vans, timeslots and itineraries; it however comes with one critical drawback. The approximate model is a not truly a subset of the unified model, and ignores the relationship with timeslots in S. The model is however only able to approximate the gate constraint.

Before going further, Table 3.5 describes the notation used in the models described in this section. The unified model takes from the deterministic model, whose notation can be found in Table 3.4.

Symbol	Description
J	The index set of all vans where $J = \{1 \cdots \lvert I \rvert\}$. It is overestimated to the size of all itineraries, that is one itinerary executed per van.
$y_{i,j}$	A binary decision variable that indicates if itinerary i is being executed by some van $j \in J$
$x_{i,j,s}$	A binary decision variable that indicates if itinerary i is being executed by some van $j \in J$ in timeslot $s \in S$
b_j	The maximum processing time van j has in the entire time horizon.
e_j	A binary variable that indicates whether van j is in use. It is not in use if it is not executing any itineraries. This is required as we overestimate the index set J.

Table 3.5: Table Detailing the Approximate and Unified Model Notation

3.7.1 Unified Model

The unified model *unifies* all three properties of the Royal Mail problem: vans, timeslots and itineraries. The deterministic model contains the gate constraint which is hard to model without the timeslots. Thus, we can proceed by adding to the deterministic model, the relationship between vans and itineraries. The alternative approach would be to link the vans to timeslots but this would not only be convoluted but also unintuitive. The model introduces decision variables $y_{i,j}$, which are true if van j executes itinerary i.

The unified model can be constructed with just the decision variables $x_{i,s}$ and $y_{i,j}$. However, it is not rigorous to declare this as an unified and generalised model in such a fashion. A rigorous method of doing so would involve creating binary decision variables $x_{i,j,s}$, and making it obey the deterministic model's constraints for all vans in J. Also, this binary decision variable should be true when itinerary i is executed by van j in timeslot s. However, these two approaches are essentially equivalent, as we shall see in the forthcoming formulation, and its accompanying description.

The reader should note that $\psi_{i,j,k}$, which is a binary decision variable, is introduced in the formulation below to help with the "either or" linearisation of the two relevant constraints.

$$\min \quad v \qquad (\bigcirc)$$

s.t.

$$\sum_{j \in J} x_{i,j,s} = x_{i,s} \qquad \forall i \in I . s \in S \qquad (\dagger)$$

$$\sum_{s \in S} x_{i,j,s} = y_{i,j} \qquad \forall i \in I . j \in J \qquad (\dagger)$$

$$\sum_{j \in J} e_j \leq v \qquad (3.18)$$

$$\sum_{j \in J} y_{i,j} = 1 \qquad \forall i \in I \quad (3.19)$$

$$e_j = \min(\sum_{i \in I} y_{i,j}, 1) \qquad \forall j \in J \quad (3.20)$$

$$\sum_{i \in I} \sum_{s \in S_{i,t}} x_{i,s} \leq v \qquad \forall t \in S \qquad (\bigcirc)$$

$$\sum_{i \in I} x_{i,s} \leq g \qquad \forall s \in S \qquad (\bigcirc)$$

$$\sum_{s \in F_i} x_{i,s} = 1 \qquad \forall i \in I \qquad (\bigcirc)$$

$$\sum_{s \in S} (x_{i,s} \cdot s) + d_i - \sum_{s \in S} (x_{k,s} \cdot s)$$

$$\leq M \cdot (\psi_{i,j,k} + (1 - y_{i,j}) + (1 - y_{k,j})) \qquad \forall j \in J . i, k \in I . k \neq i \qquad (\square)$$

$$\sum_{s \in S} (x_{k,s} \cdot s) + d_k - \sum_{s \in S} (x_{i,s} \cdot s)$$

$$\leq M \cdot ((1 - \psi_{i,j,k}) + (1 - y_{i,j}) + (1 - y_{k,j})) \qquad \forall j \in J . i, k \in I . k \neq i \qquad (\square)$$

$$x_{i,s},\ y_{i,j},\ e_j,\ \psi_{i,j,k} \in \{0, 1\} \qquad \forall i \in I . j \in J . s \in S$$

$$v \in \mathbb{N}$$

(3.18)	The number of vans used must not be more than we own.
(3.19)	Each itinerary is executed by only one van.
(3.20)	A van is in use if it has at least one itinerary executing on it.
○	These constraints come from the Deterministic Model
†	These constraints are the link between a $x_{i,j,s}$ and $x_{i,s}$ and $y_{i,j}$.
□	These two constraints express the following logical statement: *for some job i executing on van j, if there exists a job k also executing in the same van, its durations must not overlap.* These constraints essentially co-ordinate between the two different paradigms of decision variables: $x_{i,s}$ and $y_{i,j}$. To express this, firstly, the statements here need to be expressed as an either or problem. Either the entirety of itinerary i, from start to end, happens before k or vice versa. This notion of "either or" is expressed with $\psi_{i,j,k}$. Secondly, we must only check these constraints are valid when both itineraries i and k are being executed by the same van j. The logical and of decision variables $y_{i,j}$ and $y_{k,j}$ encode whether this is true or not. One could easily multiply the three aforementioned decision variables, $y_{i,j}, y_{k,j}$ and $\psi_{i,j,k}$ to the left-hand side of the constraint, which would be an intuitive representation of this constraint. However, this would result in a *non-linear* constraint. To avoid this, the big-M method is used to linearise this, where M is some large value that can. When either one of these three decision variables are not true, the less-than constraint is trivially true if the value of M picked is sufficiently large. Since the left hand side of these expressions measure the difference between start and end times, it cannot be larger than the value of T in a feasible schedule. Thus, we can pick set $M = T$. [13]

The unified model, is in fact a generalisation of the deterministic one. This can be proved quite simply, and should be evident solely by the construction of the unified model. More rigorously, an optimal solution of the unified model satisfies all the constraints of the deterministic model, by construction, *but not vice versa*. We thus have a trivial proof that the unified model is a generalisation of the deterministic model. If this is the case, we can also declare that there exists a set of constraints, which when added to the unified model, derives the deterministic one.

The model is, however, significantly more complex than the deterministic model. We have increased the number of decision variables by a considerably large amount and the (□) constraints are quantified across an almost prohibitively very large space, i.e. $|I| \times |I| \times |J|$. Since J is overestimated to the size of all itineraries, this is in fact quantified over a cubic space $|I|^3$. Thus, in Section 3.7.2 a compromise is reached with the approximate model, which solely models the relationship between vans and itineraries.

3.7.2 Approximate Model

This modelling approach comes with a pre-processing step as is described in Algorithm 6 below. After the pre-processing, the model in Equations (3.21) to (3.27) is used to obtain a solution.

Algorithm 6 Approximate Model Preprocessing Algorithm

procedure PREPROCESSING

Create vector $\boldsymbol{b} \in \mathbb{N}^{|J|}$

With predefined $g \in \mathbb{N}$, as the gate constraint value

for $i \in \{1, \cdots |J|\}$ **do**

$b_i \leftarrow T - \lceil i \div g \rceil$

return $\boldsymbol{b}$

$$\min \ v \tag{3.21}$$

$$\text{s.t.} \quad \sum_{j \in J} e_j = v \tag{3.22}$$

$$e_j = \min(\sum_{i \in I} y_{i,j}, 1) \qquad \forall j \in J \tag{3.23}$$

$$\sum_{j \in J} y_{i,j} = 1 \qquad \forall i \in I \tag{3.24}$$

$$\sum_{i \in I} (y_{i,j} \cdot d_i) \leq b_j \qquad \forall j \in J \tag{3.25}$$

$$e_j, y_{i,j} \in \{0, 1\} \qquad \forall i \in I . j \in J \tag{3.26}$$

$$b_j, v \in \mathbb{N} \tag{3.27}$$

(3.21)	The objective is to minimise the number of vans v.
(3.22)	The number of vans used is equal to the total number of active vans.
(3.23)	A van j is only active if it has some itineraries executing on it. If it has none, the first argument to the min function evaluates to 0. If not, this evaluates to a positive number.
(3.24)	An itinerary can only be executed by one van at most.
(3.25)	The amount of time a van can execute itineraries is bounded by the pre-processed vector $\boldsymbol{b}$
(3.26)	These are binary decision variables
(3.27)	These are positive integers

The pre-processing algorithm roughly approximates the gate constraint as reiterated below.

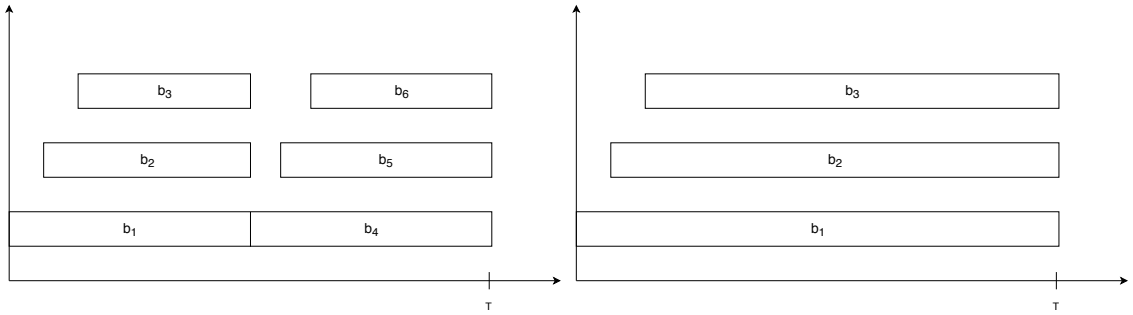
$$\sum_{i \in I} x_{i,t} \leq g \qquad \forall t \in S$$

It constructs *bins* of sizes which approximately obeys the gate constraint. This is also visualised in Figure 3.2 below. The preprocessing step removes the starting bit of each

bin. By doing this, we are packing the itineraries in truncated bins, forcing it to obey the approximated gate constraint. The reason why the algorithm only approximates the constraint, is that it only guarantees that the gate constraint is to be true at the start of the day. The real gate constraint could however result in a situation as visualised in Figure 3.2b, which the approximate model will not respect.

This will impact the process of recovery. As will be further explicated in Section 4, uncertainty is to be realised at one arbitrary point in time. However, if we were to use the approximate model, we would lose this flexibility, and only deal with situations in which the uncertainty is realised on the day of, and, before delivery. This approximation also means that when a van executes multiple itineraries, it does not re-enter the DO after finishing one itinerary to execute the next. The real gate constraint, however, states that at any time period, there must not be more than g itineraries being started. This is a compromise that needs to be reached in order to model the problem.

Another drawback of this approximation is that it can no longer be derived from the unified model; this is unlike the deterministic model, which can be derived from the unified model. This is further elaborated in Section 3.7.3 below.



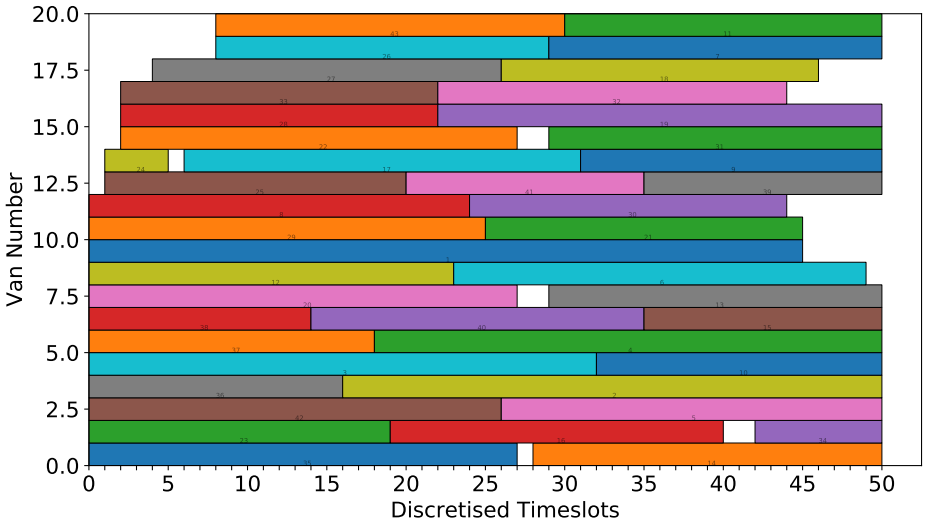
(a) The gate constraint **(b)** The approximate gate constraint

Figure 3.2: Visualising the gate constraint and the approximation of it. Depicted above is a specific scenario where the approximate constraint cannot model the requirements of the original constraint.

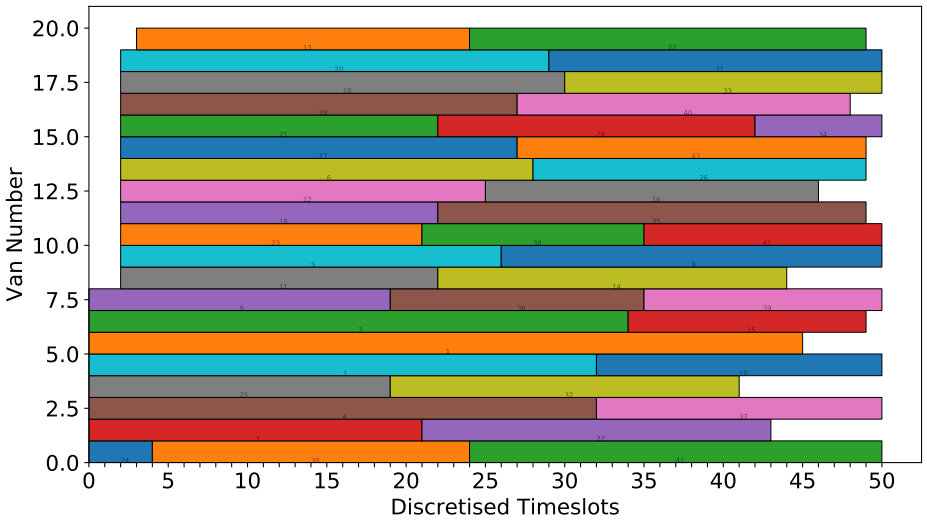
For all practical purposes, the approximate model and the deterministic model perform almost identically in terms of optimal values. There are only a handful of instances where the optimal value is only slightly different. The approximate model always performs worse but by only at most 3 vans in two instances.

If it were the case that we had many instances where the scenario depicted in Figure 3.2a were to be true, this would mean that the optimal value of the deterministic model would be higher. This is because there is a lesser amount of viable space for an itinerary to assigned to, in Figure 3.2b compared to Figure 3.2a.

This suggests that the approximation and assumption made is a reasonable one and not many itineraries are executed at the same time beyond the start of the day. A solution from both the models from the same day in delivery office A is depicted in Figure 3.3 below.



(a) Deterministic Model



(b) Approximate Model

Figure 3.3: Depicted above are schedules from the Deterministic and Approximate Model. These can be thought of as timetables, or Gantt Charts, where each bar represents some itinerary.

The x-axis represents the discretised timeslots S. In this figure, $T = 50$, i.e. there are 50 timeslots in the time horizon of the day. The y-axis represents each van, i.e, each row is some van's schedule.

3.7.3 Relationship to the Generalised Bin Packing Problem

We observe that the approximate model is in fact an instance of the Generalised Bin Packing Problem (GBPP) as previously described in Section 2.8.1. To show this relationship, we recall the GBPP notation in Table 2.4.

First we define bins types. We simply define it as $\tau = \{1, 2, \cdots\}$. With this we then have to define the following terms as such:

$$L_t = 0$$
$$U_t = g$$
$$W_t = T - \lceil t \div g \rceil$$

The above does not modify or quantify any constraints, as they are proposed as constants. We require some specific quantifications, as follows. First, we require that $I = I^{\mathcal{C}}$, or equivalently, $I^{\mathcal{NC}} = \emptyset$. By doing so we can effectively ignore the value of π_i. Lastly, we also require that

$$\forall t, t' \in \tau \quad \Pi_t = \Pi'_t$$

This is to say that the cost of a bin is irrelevant, and the fundamental goal is to reduce the number of bins used. This could also simply be expressed as

$$\forall t \in \tau \quad \Pi_t = 1$$

Collectively, and as the literature does, we term this problem as multiple bin problem with $\Pi_t = 1, I^{\mathcal{NC}} = \emptyset$. This is in effect, a Variable Size Bin Packing Problem (VSBPP) [19].

The unified model is, however, not an instance of the GBPP. This is due to the nature of the gate constraint, which is at ends with the bin packing problem itself. We can assert this quite easily as Figure 3.2 is itself a depiction of a counter-example.

3.8 Computationally Deciding on Schedules

After the data cleaning, we can now effectively decide on schedules. The data is prepared such that it is coherent with our modelling, and is largely free of anomalies. Several commercial and free software solutions exist that are able to decide on (integer) linear programs. For this, we opt to use *Gurobi*, a "state-of-the-art solver for mathematical programming" [22]. The decision was made because of its highly expressive and intuitive interface, as well as its competitively fast processing times.[34].

3.9 Summary

Thus, this chapter has contextualised the problem in both a practical and mathematical manner. First it proposes a method of cleaning the data so that it may be used in practical analyses. This chapter then details the original model (i.e. the deterministic model) as proposed by Page [36], noting its limitations, and seeking to rectify them as such.

This chapter's main contributions include an extension by generalisation of the original model, to overcome its restrictive modelling, and a critical evaluation of the generalised formulation. It justifies the generalisation's problems with tractability. To tackle this, this chapter's contributions also includes an approximate alternative reformulation. Furthermore, it shows that this approximate reformulation is a specific instance of the GBPP, known commonly as the Variable Size Bin Packing Problem (VSBPP). Thus, it also contextualises any following efforts utilising the approximate reformulation, which can now be classified as contributions to the study of the VSBPP.

Chapter 4

Statistical and Uncertainty Analysis

4.1 Motivation

At this point, we have prepared our data and consolidated our modelling approaches. Before beginning to approach performing optimisation under uncertainty, we must first characterise the uncertainty we experience with statistical analyses of the prior data. Upon doing so, we also require a framework from which schedules can be analysed under uncertainty. The comparison of schedules is also not a trivial one so we must devise a simple and configurable method of scoring a schedule to enable such comparisons.

4.2 Statistical Analysis

This section justifies how statistical information is extracted from the data. This statistical information can be used to augment processes (as is done in later sections) to make them more applicable. It should however be noted that this section is dedicated to how the statistical information is obtained. In a practical scenario, the aforementioned processes can be augmented with whatever statistical data one may deem fit for the scenario.

Since the DO data can be erratic, robust statistic procedures shall be employed as described previously in Section 2.9. We henceforth use the symbols μ_r and σ_r with the subscript r to represent robust statistical measures. We refer to traditional non-robust statistical measures without the subscript.

4.2.1 Desired Statistics

The primary statistic that is of interest to us is the trend of van usage. This can be done at varying granularities. Coarsely the mean and variance of van usage per DO can be found. However, this could also be done at a finer level, that is per day of the week.

From this we could proceed to make reasonable estimates on the magnitudes of perturbations that can be experienced. That is to say, when things go wrong, we can reasonably estimate the degree to which things go wrong.

Lastly confidence intervals on the amount of vans used each day can be modelled so as to augment the construction of polytopal uncertainty sets as previously described in Section 2.5.1.

Before diving into the statistical analysis the reader should note two points. Firstly, as mentioned in Section 3, itineraries that occur on Sunday are disposed of. There are instances where a van or two is in use but these are thought of as an exception and not the rule. Furthermore, days where no vans are used are also discarded from analysis, as the DOs are assumed to be shut (e.g. bank holidays, strikes, et cetera). Second, the amount of data per DO varies. The sample size of the data is as follows:

DOs	Sample Size
A	112
B	78
C	111
Total	301

Table 4.1: Sample Sizes of Royal Mail Data

4.2.2 Frequency Analysis

Regardless of the granularity of the statistical data (per day of week or per entire data set), one thing is evident. If the raw data were to be analysed, the trends observed are very minute. Here raw data which is `no_work` is obviously disposed of. Other cleaning methods as described in Section 3.4 are not applied as some data cleaning processes add new itineraries.

Trends in the data are much more pronounced when they have been put through a solver with the deterministic model as described in Section 3.6.1. This is shown for both levels of granularity in the tables following. The change in van utilisation tables details the mean and deviation of the following set:

$$\left\{ v_d \mid v_d = \frac{v_i - v_{i-1}}{v_{i-1}} \times 100 \right\}$$

where v_i represents the number of vans required on a given day i. Table 4.2 and 4.3 show the van usage and change in van utilisation with raw RMDSG data before a solver. Table 4.4 and 4.5 show the van usage and change in van utilisation with after a solver. As a reminder, numbers are presented with up to three decimal places.

DOs	Median	μ_r	σ_r
A	56.0	55.5	2.965
B	95.0	95.042	1.483
C	111.0	112.842	8.896

Table 4.2: Van Usage from RMDSG data

DOs	Median	μ_r	σ_r
A	38.0	35.971	2.965
B	62.0	59.792	4.448
C	66.0	64.762	7.413

Table 4.3: Van Usage after solver

DOs	μ_r	σ_r
A	4.079	3.097
B	1.543	1.544
C	3.468	2.784

Table 4.4: Utilisation change from RMDSG data

DOs	μ_r	σ_r
A	21.779	14.279
B	16.324	12.724
C	15.179	14.385

Table 4.5: Utilisation change after solver

Looking at the mean van usage per day of the week also highlights the discrepancies between the raw data and the solver data. This can be seen in Figures 4.1 and 4.2 below.

The raw itinerary data exhibits a very minute trend as opposed to the solver data. If the one can agree to the following two premises, then it should easily follow that the data after solver more accurately represents trends, and should be the basis of our statistical modelling.

The first premise is that van usage *does* trend across the days of the week. In fact from Figure 4.2, that is the data after the solver, one can observe a more aggressive trending, one which is not observable in the raw RMDSG data in Figure 4.1. The next premise, is that the van usage currently occurring at the DO's are not optimal.

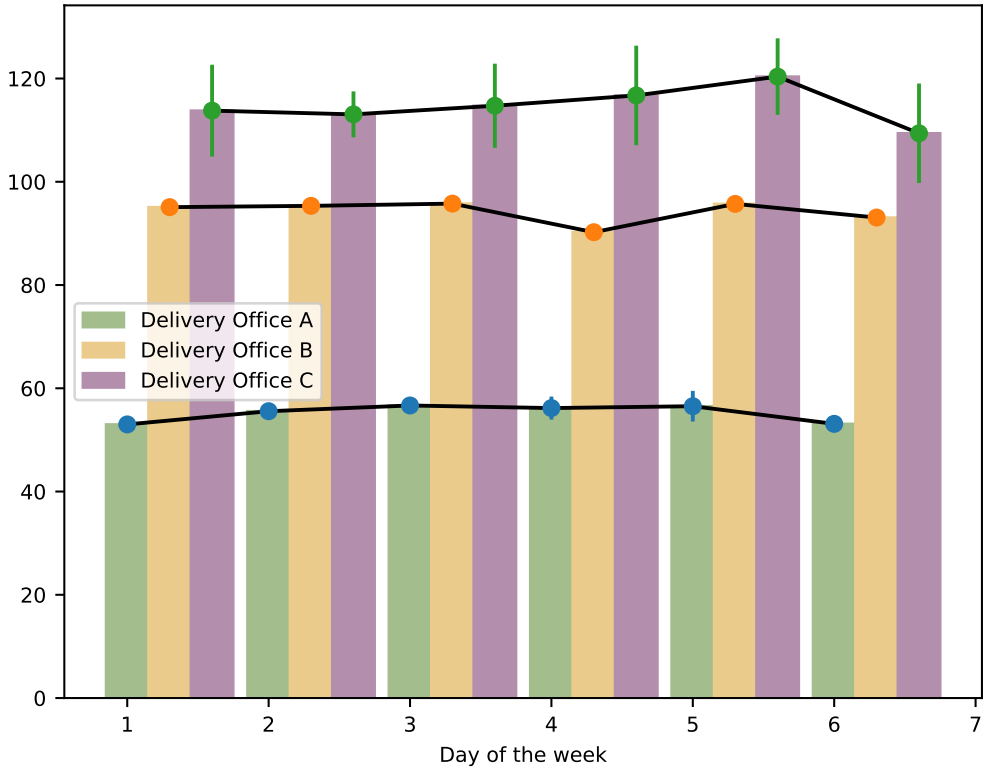
This statistical analysis was in part inspired by the San Francisco's police department's approach to optimising the number of deployed police officers [43], where demand across the days of the week was measured by their "CAD System", which is in turn used as a basis for further optimisation decisions.

With this established, *robust* confidence intervals for days of the week can be computed. As an example, the 95% confidence intervals on the number of vans needed per day of the week is presented in Table 4.6 below.

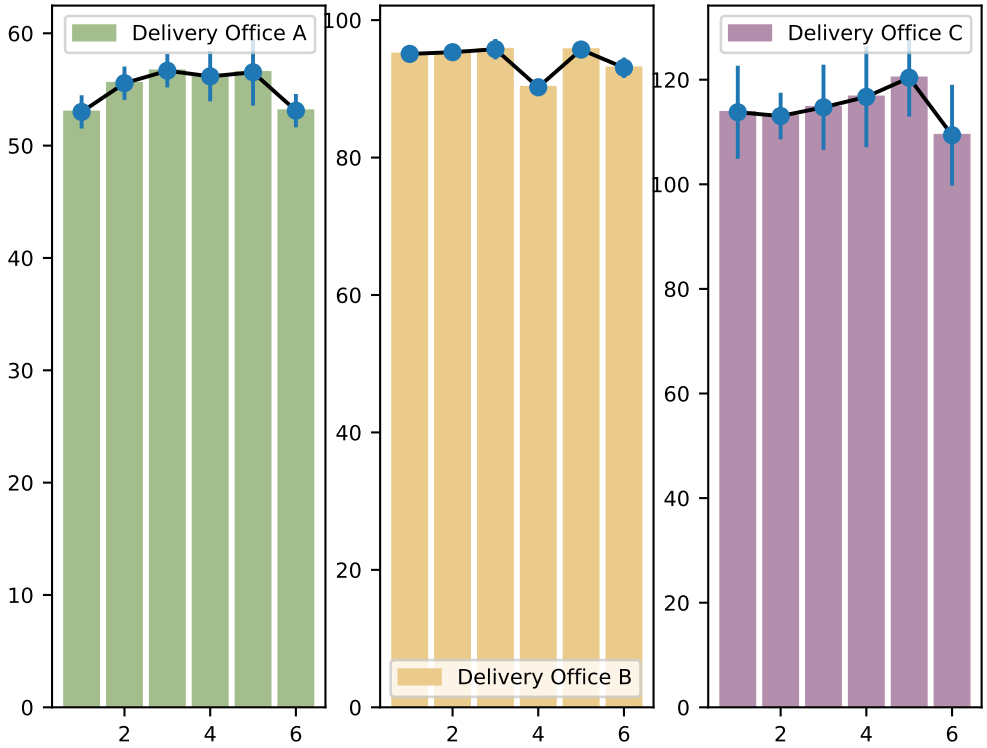
DO	Mon	Tue	Wed
A	(26.777, 29.684, 32.591)	(26.197, 30.556, 34.915)	(27.919, 32.278, 36.637)
B	(50.689, 56.5, 62.311)	(49.965, 51.417, 52.869)	(55.744, 64.462, 73.18)
C	(56.967, 62.778, 68.589)	(55.363, 59.722, 64.081)	(65.085, 69.444, 73.803)

DO	Thu	Fri	Sat
A	(29.482, 32.389, 35.296)	(29.83, 32.737, 35.644)	(21.204, 24.111, 27.018)
B	(51.958, 57.769, 63.58)	(55.07, 59.429, 63.788)	(42.879, 45.786, 48.693)
C	(64.578, 70.389, 76.2)	(67.198, 70.105, 73.012)	(46.53, 50.889, 55.248)

Table 4.6: 95% Confidence intervals on van usage across DOs per day of the week. Each triple here represents a robust mean sandwiched between the lower and upper bounds of the confidence interval.

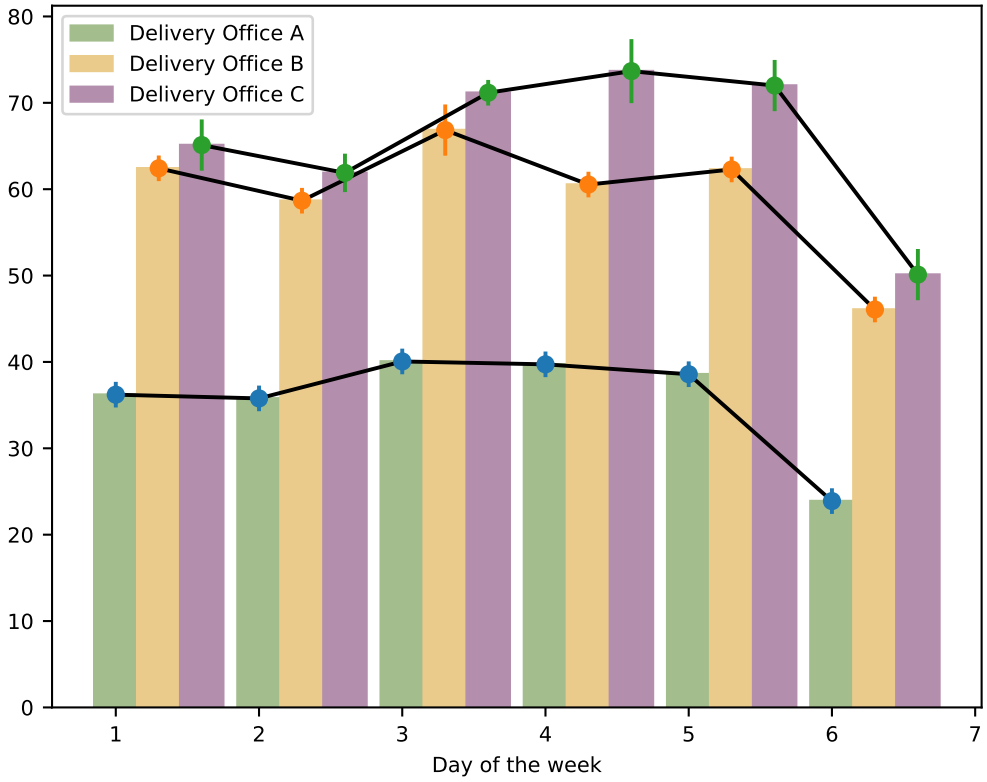


(a) Superimposed for all DOs

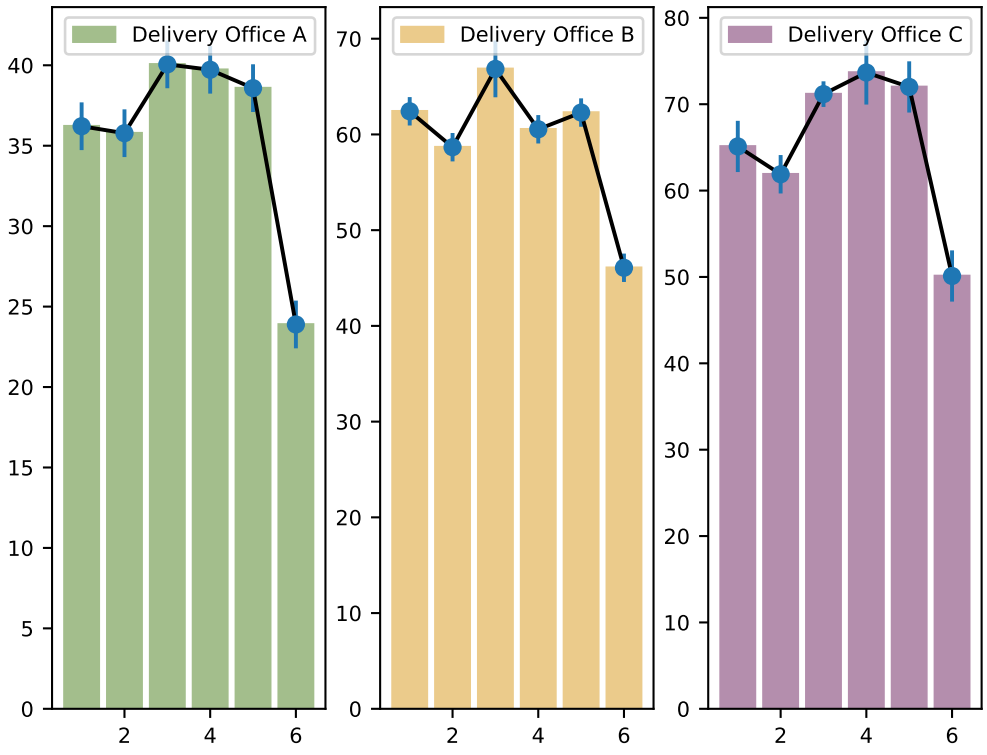


(b) All DOs separately

Figure 4.1: The graphs depict the robust mean van usage across different days of the week from Royal Mail data. Here we observe a very light trend of van usage across the different days of the week.



(a) Superimposed for all DOs



(b) All DOs separately

Figure 4.2: The graphs depict the robust mean van usage across different days of the week after the Royal Mail data has been optimised with the deterministic model. Here we observe a much stronger trend of van usage across the different days of the week.

4.3 The 3-step Process

The deterministic model, as detailed in Chapter 3, principally works to minimise the number of vans used per each DO. However, the problem statement is also concerned with notions of uncertainty and recoverability. Classically this problem is modelled as a two stage problem [15, 29, 31]. This dissertation will use the exact same idea broken into three distinct steps instead, as depicted in Figure 4.3 below.

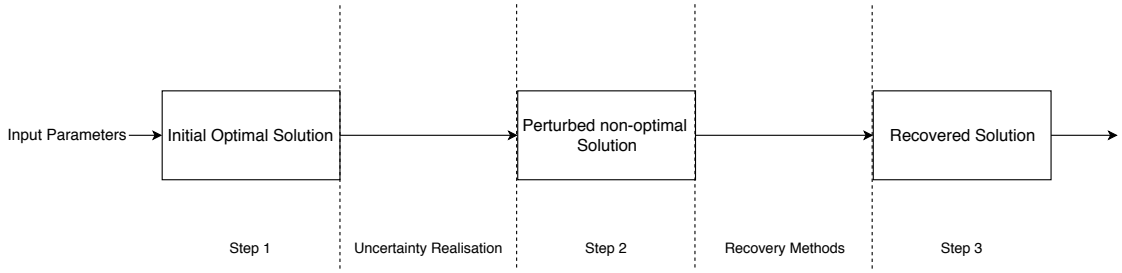


Figure 4.3: The 3-step Process is depicted as above. Problems of optimisation under uncertainty can be essentially visualised in this manner. It is canonically termed a 2-stage problem, as there are two stages of actions - uncertainty realisation and recovery methods.

This is effectively how the problem can be viewed from Royal Mail's perspective. On the night before, they have sufficient information (i.e. input parameters) on the workload the next day, upon which they can construct an optimal schedule (Step 1). On the day of delivery, several types of uncertainty are realised throughout different points in the schedule. To limit the scope of this dissertation, it is assumed that uncertainty is realised at one arbitrary point in time, upon which the initial optimal solution is most probably no longer feasible, let alone optimal (Step 2). After this, Royal Mail will have to apply some recovery or rescheduling strategy as recourse. After doing this, they would ideally now possess some solution which is good, but not necessarily perfect.

The different steps in this process will try its best to obey the *timetabling condition* as described in Liebchen et al. [31]. This condition re-contextualised would in essence state that no van should execute an itinerary earlier than scheduled.

As different strategies are explored, this 3-step process is what will be used as a framework to understand and compare how schedules behave throughout the entire life cycle of a delivery day at a DO.

4.4 Perturbation Generation

After the optimal schedule has been designed, there are several scenarios wherein "things fall apart" (between steps 1 and 2 of the 3-step process). A van, or an itinerary, may experience many different types of perturbations. More abstractly, perturbations can be classified into 6 types as described in Table 4.7 below.

Since the main goal is to reduce the number of vans used per DO, type 4 perturbations can be ignored. Furthermore, perturbations of types 5 and 6 can also be ignored. As we rely on historical data, this sort of perturbation makes little sense to us. Situations of this sort are unlikely to happen in practice, given that mail is sorted methodically prior to delivery. It can also be assumed that it is unlikely for mail to be lost, and given delivery constraints

Type	Description	Example
1	An itinerary expands in time	Delay in start time
2	An itinerary contracts in time	Lesser traffic on that particular day
3	A van goes down	Van breaks down
4	A new van is available	New vans are bought for the DO
5	A new itinerary is required to be schedule	More mail is discovered
6	An itinerary is no longer needed	Mail is accidentally misplaced and lost

Table 4.7: Detailed above are the different types of perturbations that can possibly be experienced in the operations of Royal Mail.

Royal Mail are operating under, it is also unlikely for mail to not need delivering. These are in line with the consistency assumptions made in Section 3.

Perturbations are introduced systematically as described in Algorithms 7 to 9 below. Additionally, it will stick to the timetabling condition as mentioned above. A counter argument to this condition would be that if this were to be taken into account, type 2 perturbations could effectively be ignored. Furthermore, in Royal Mail's case, vans mostly do not depend on the schedules of other vans, unlike the problem presented in the aforementioned paper. However, there are reasonable justifications for such an condition. First, in practical scenarios, nothing goes according to plan, and more often than not, things take longer than they should. Next, an optimiser generally should not concern themselves about uncertainty realisations where more free time is present. Lastly, it is better to stick to schedule start times for the person utilising the schedule. Moving these start times around too much can be considered troublesome.

Algorithm 7 Probability Roll Algorithm

```
1: procedure ROLL(probability)
2:     r ← random(0, 1)
3:     return (r ≤ probability)
```

The function `random(0, 1)` is a pseudo-random number generator sampling from a uniform distribution between 0 and 1. The algorithm is used to supplement the following algorithm. The parameters involved in this step as discussed in Table 4.8 below.

Algorithm 8 Perturbation Generation Algorithm

```
1: procedure GENPERTURBATION(vans, dayOfWeek)
2:     for each van do
3:         if !roll(perturbation_chance) then
4:             continue
5:         if roll(type3_chance) then
6:             Increase backup vans required
7:         else
8:             Pick random itinerary i
9:             s ← shift(dayOfWeek, i.duration)
10:            if roll(type1_chance) then
11:                i.duration ← i.duration + s
12:                Move itineraries after i
13:            else
14:                i.duration ← i.duration − s
```

Parameter Name	Description
perturbation_chance	The probability that a van experiences a perturbation
type3_chance	The probability that a van experiences a type 3 perturbation
type1_chance	The probability that a van experiences a type 1 perturbation given that it does not experience a type 3 perturbation i.e. (1 - type3_chance). Consequentially, the chance that a type 2 perturbation occurs is (1 - type1_chance)

Table 4.8: Parameters involved in Perturbation Generation

In Line 8 of Algorithm 8, it states that an itinerary is picked at random. This is done with a uniform distribution pseudo-random number generator as in Algorithm 7. A van will execute one or more itineraries, and the itinerary which is affected is picked at random. When a type 1 perturbation occurs, the itineraries after the itinerary affected will have to be moved. When a type 2 perturbation occurs, the itineraries occurring after are left as is.

It is rather complicated to pick values for the above parameters which are statistically informed by prior data. The data currently presented lacks any information about how often a van breaks down or how punctual vans are. As such in this dissertation, certain parameters are picked and worked with consistently. These values are described in the Table 4.9 on the right.

Parameter Name	Values
perturbation_chance	0.5
type1_chance	0.7
type2_chance	0.3
type3_chance	0.1

Table 4.9: Perturbation parameter values

However, the size of type 1 and 2 perturbations can be reasonably estimated from prior data. As shown in Section 4.2, it is entirely possible to obtain statistics on optimal van usage per day of the week. This can then be used to inform the shift function as described in Algorithm 9 below, which is tied in with type 1 and 2 perturbations.

Algorithm 9 Perturbation Shift Algorithm

1: **procedure** SHIFT(dayOfWeek, duration)
2: μ_r, $\sigma_r \leftarrow$ DailyStats[DO][dayOfWeek]
3: $X \sim \mathcal{N}(0, \sigma_r^2)$
4: shift $\leftarrow$ abs(sample from X)
5: shift-percentage $\leftarrow \frac{\text{shift}}{\mu_r}$
6: shift-timeslots $\leftarrow \lceil$shift-percentage $\cdot$ duration$\rceil$
7: **return** shift-timeslots

In Algorithm 9, a few things take place. Firstly, a set of daily stats is precomputed as is done in Section 4.2, which includes the robust mean and deviation. Using the deviation, we sample from normal distribution and get the absolute value of the sample. In doing so, we have constructed a half-normal distribution centered around a mean of 0, depicted by Figure 4.4, as opposed to a normal distribution. A half-normal distribution is used as negative values are not desired. Furthermore, it is in our interest to preserve the deviation of the distribution, although the mean is no longer the same. A lognormal distribution could have also been used but much more effort would be required to obtain similar parameters. Next, we use this sampled value to calculate the percentage deviation of the sample from the mean. In doing so, perturbation sizes are being sampled as a percentage of mean. Then this percentage is multiplied by the affected itinerary's duration (in timeslots S) to give the number of timeslots increase or decrease a perturbation causes.

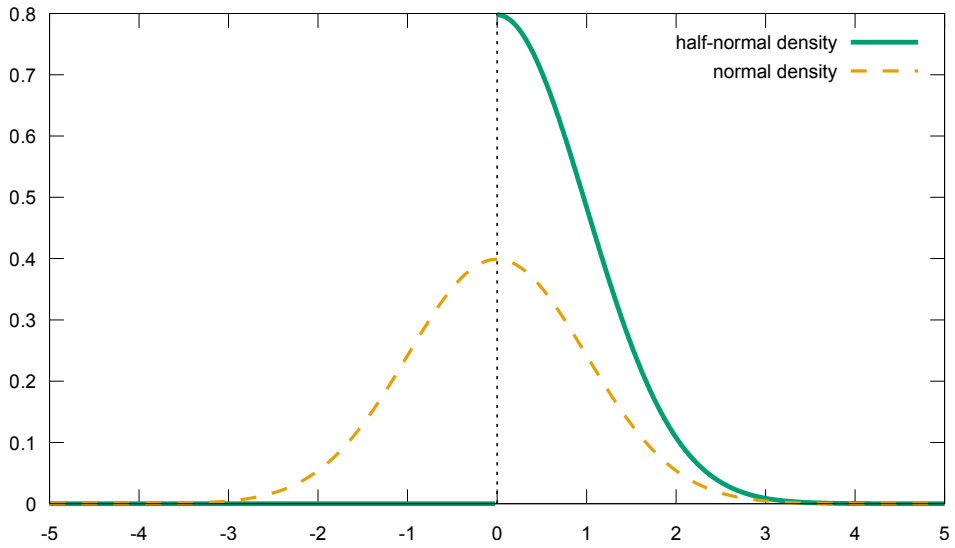


Figure 4.4: Half-normal vs. Normal distribution. The half-normal distribution allows us to sample non-negative values from a distribution with the same behaviour as a normal distribution. Note we preserve the "shape" of the normal distribution with a half-normal distribution.. [1]

Whilst performing the perturbation generation algorithm, certain things kept track of. The number of type 3 perturbations is recorded, and, on every occurrence of a type 1 perturbation, the number of itineraries after the target itinerary affected is kept count of. This only counts the number of itineraries that have to be shifted because of a type 1 perturbation. If two itineraries had some slack between it, and the type 1 perturbation on the first itinerary does not affect the second itinerary, it is not counted. This will tie in with the next section.

4.5 Scoring Schedules

To enable the comparing and contrasting of scheduling strategies, this dissertation proposes a method to score schedules, termed *disturbance*. As it's name suggests, it is a measure of the amount of disturbances experienced in a schedule. To understand the reasoning behind the scoring method, enumerated below are what makes a schedule unfavourable (i.e. introduces disturbances). We take note that the following reasoning can also more generally be applied to any open shop scheduling problem [8], given that type 4 perturbations are ignored.

1. A van is executing a set of itineraries throughout the course of a day. It would be unfavourable if one itinerary were to expand in time, causing itineraries after it to be delayed.

2. If a van experiences a perturbation that causes it to work longer than desired, i.e., working over time horizon, this would be unfavourable. This is given that there are union hours and going over the time horizon would be violating this [2]. This is henceforth referred to as *spillage*.

[1] Figure courtesy of Wikimedia Commons. Obtained here and accessed on 16th April 2019
[2] More information on this can be found here: https://www.gov.uk/part-time-worker-rights

3. Requiring backup vans is unfavourable. Type 3 perturbations, i.e. when vans break down is an example of this. This effectively increases the total number of vans required on that day. Furthermore, the number of additional vans required also relies on how much time spills over when vans work past the end of the time horizon (Time-Horizon).

Using the intuition from the points above, the corresponding disturbance parameters are defined in Table 4.10 below.

Disturbance Parameter Name	Description
`num_affected`	The number of itineraries that have had its start time shifted due to a preceding itinerary experiencing a type 1 perturbation.
`num_spill`	Number of vans whose total time of execution is spilling over the Time-Horizon. This is the number of vans that have gone over the value of end_{abs}
`spill_time`	The total amount of time slots spilled over the value of end_{abs}. This is some value in set S (described in Table 3.4)
`num_vans_req`	Number of new vans required. This is a measure of the number of type 3 perturbations experienced.
`total_vans_req`	Total Number of new vans required from the number of type 3 perturbations and `spill_time`

Table 4.10: Parameters involved in measuring disturbance

Then, the score is calculated as described in Equation (DIST) below.

$$\begin{aligned} \texttt{total\_vans\_req} &= \texttt{num\_vans\_req} + \frac{\texttt{spill\_time}}{T} \\ \texttt{disturbance} &\triangleq \frac{(a \cdot \texttt{num\_affected}) + (b \cdot \texttt{num\_spill}) + (c \cdot \texttt{total\_vans\_req})}{\text{Optimal Number of vans}} \\ &\text{s.t.} \quad a < b < c \\ &\qquad a + b + c = 100 \end{aligned} \tag{DIST}$$

The first line in (DIST) calculates the *total number of new vans required* given both total `spill_time` and type 3 perturbations. We divide the total `spill_time` by the size of the time horizon T, as we roughly assume that this is the total amount of work done by a van.

The second line in (DIST) scales parameters `num_affected`, `num_spill` and `total_vans_req` by *penalties* a, b and c respectively. The first of these parameters has its units in the number of itineraries and the latter two have units in number of vans. To perform a normalisation, the expression is divided by the original optimal number of vans. This implies that the disturbance score is largely independent of the number of vans used.

Lastly, the values of a, b and c or ordered with a being the smallest and c the largest. The decision to do this is to scale the disturbance parameters differently, and penalise the more pertinent parameters. For example, it is more problematic to have an extra van (`total_vans_req`) than to have one itinerary shift (`num_affected`). As should be evident

at this point, the lower the disturbance, the better. For the time being the penalties are picked as follow

$$a = 10 \qquad b = 40 \qquad c = 50$$

The sum of the penalties is kept to 100 for consistency's sake.

In the 3-step process, a schedule's disturbance would start of with a perfect score for the intial optimal solution (step 1), which is a score of 0. It is the perturbed and recovered solutions (steps 2 and 3) where the amount of disturbance in a schedule increases. One limitation of the disturbance scoring method is that is had no regard for the optimal number of vans. For example, a schedule could have a relatively lower disturbance score but could use more vans. As such we will also need to keep the optimal number of vans in mind, by using other measures such as the price function (recall Section 2.5.2). The value of the parameter `num_affected`, which is set in for the perturbed solution, remains the same for the recovered solution. The other parameters will however vary.

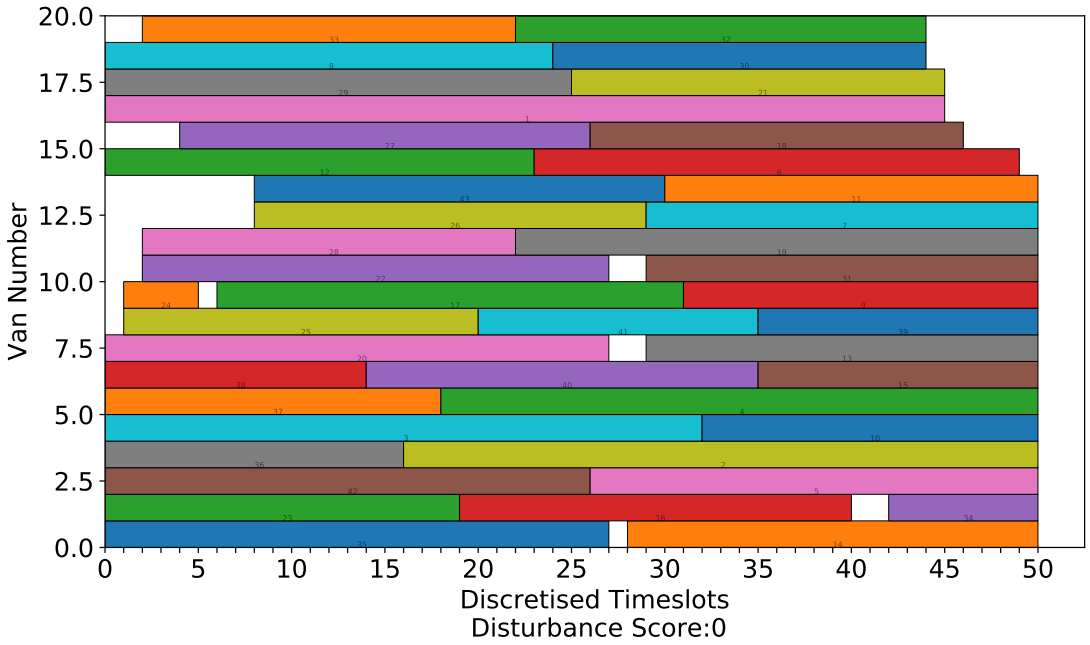
As an example of the disturbance score in action, two schedules are depicted with their respective scores annotated in Figures 4.5a and 4.5b. Each bar represents an original itinerary (i.e. a raw itinerary). Both are schedules generated from the deterministic model and from the same day in the same DO. Figure 4.5a is a schedule without any perturbations, and an optimal one. As such this has a "perfect score" of 0. The schedule in Figure 4.5b is affected by the perturbation generation algorithm.

As should be observable, the first schedules are spilling over, there are more gaps and it is not as good of a schedule. This is clearly reflected in the disturbance score which shoots up to ≈ 13.

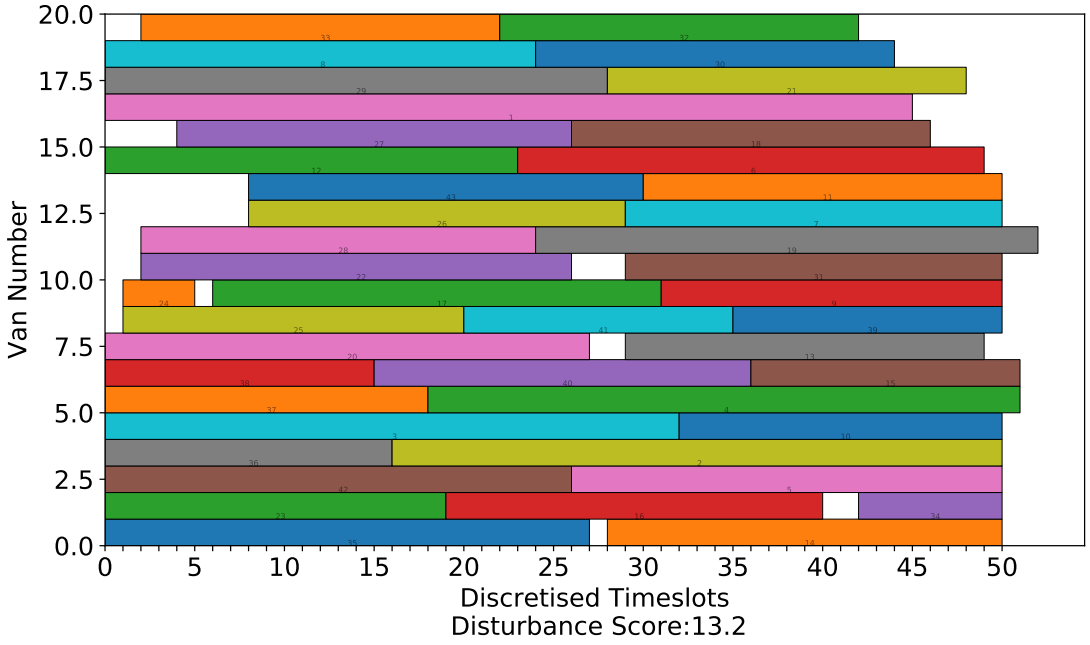
4.6 Summary

This chapter proposes a method of scoring and comparing schedules termed *disturbance*. This method should extensible to a variety of multiprocessor scheduling problems under uncertainty (e.g. bin packing, open shop problems), and is independent of the number of processors, thus allowing a simplistic and tunable method of comparing schedules that have experienced uncertainty.

Before detailing the scoring method, this section's contributions also formalise exactly how we can study schedules under uncertainty. The three prior contributions are as follows. First we define how statistical information can be extracted from the raw data. After this, we describe the *3-step process*, a framework for the study of a schedule's behaviour under uncertainty. Lastly, we formalise methods of simulating uncertainty that are informed by a priori statistical information.



(a) A schedule with no disturbance



(b) A schedule with disturbance

Figure 4.5: Depicted are two schedules for the same day, both of which have been decided with the deterministic model. However, unlike the top one, the schedule on the bottom has experienced perturbations. Some of the bottom schedule's itineraries have realised this uncertainty. We note that the picture on the bottom is much more disturbed, and thus has a much higher disturbance score.

The x-axis here represents the discretised timeslots S. In this figure, $T = 50$, i.e. there are 50 timeslots in the time horizon of the day. The y-axis represents each van, i.e, each row is some van's schedule.

Chapter 5

Recovery Methods

5.1 Motivation

If we were to design schedules from start to end, we would need to take into account all of the steps of the 3-step process as was defined in the previous chapter. The process of recovery, at this point and in the context of our problem, needs to be understood and grappled with. This is juxtaposed with the process of initial scheduling, which can rely on a large toolbox of classical modelling methods.

It is imperative to explore different methods of recovery, and understand each method's trade-offs. Before this process, we must first figure out sensible limits on recovery; it is pointless for the problem of scheduling and rescheduling to be solved during the recovery phase itself. In this endeavour, we should analyse and compare recovery methods experimentally, with the groundwork formalised in the previous chapter, namely the 3-step process, perturbation generation algorithm and disturbance scoring (Sections 4.3, 4.4 and 4.5 respectively).

The first step in exploring recovery methods begins with a suspicion that the deterministic model (Section 3.6.1) has some equivalency or similarity to the makespan problem. Here we recall Section 2.7, where we detail how an heuristic approximate recovery method (LPT) provides strong guarantees (recall Section 2.7.3) to the makespan problem. As such, we first analyse and determine experimentally whether this is in fact the case for the Royal Mail problem. The second step here is analysing the notion of modelling recovery methods as linear programs. As linear programs are exact methods (as opposed to heuristic approaches like LPT), they give an optimiser much better control over the behaviour of recovery.

5.2 Recovery constraints

As per Section 2.6, recovery methods have two constraints. First, the computational complexity of recovery should be limited, [29, 31], else the entire problem could just be resolved for the recovered solution in step 3 of the 3-step process. This would not only be computationally prohibitive, but also limiting to this dissertations extensibility. This is because, if future work were to consider modelling paradigms such as multi-stage optimisation, and uncertainty was to be realised in more than one point of time, repeated applications of the recovery algorithm would become very costly. This can be attributed to the fact that integer programming problems are $\mathcal{NP}$-hard.

Next, the *costs* (henceforth and interchangeable with *cost functions*) of recovery should also be limited [31]. This is to say that the actions of recovery should each have an associated cost incurred, and the sum total of the costs incurred should be limited. A simple example of such a cost in the context our problem is as follows. If in the process of recovery, we decide to swap the assignments of two itineraries, this action should have an associated cost incurred. We observe that the timetabling condition (recall Section 4.3) is related to the costs of recovery. This is because minimising the costs of recovery forces us to stay close to the original planning solution, which is correlated with the condition.

5.3 Longest Processing Time

As far as for heuristic and approximate approaches to recovery, the options available are limited. One of these options is the Longest Processing Time (LPT) method, though it is a compromise between the two constraints for recovery (Section 5.2). While it performs well with respect to computational complexity, it does not provide any guarantees for costs of recovery. We will have to relax this constraint and note that this is due to the nature of LPT (Section 2.7.2), which reassigns jobs that results in infeasibility after realising uncertainty. This means that the only decisions that are preserved are those that are still feasible after uncertainty realisation, i.e. in the perturbed solution (step 2). Furthermore, the timetabling condition is broken for jobs that result in infeasibility.

However, it is still of interest to validate if LPT is a viable method of recovery, and if it is applicable to and effective in the case of the Royal Mail problem. As a reminder, LPT provides strong guarantees for the makespan problem, and as such we would like to analyse and experimentally validate if the deterministic model is able to benefit from these strong guarantees.

Due to the limited amount of data that is present, and to strengthen our conclusions, Gurobi's solution pool feature shall be utilised [22]. This feature is able to obtain multiple optimal feasible solutions to the same problem. This parameter is tuned such that it is possible to obtain up to 5 different solutions at best.

5.3.1 Initial Attempt

Here we detail some insight that was obtained through an initial failed attempt at conducting this experiment. Regardless of this, the failings elucidated how the LPT method could be applied in a much simpler way.

We initially attempted to reformulate the deterministic model as a LexOpt one. This meant that we attempted to redefined the problem as of makespan problem with lexicographical ordering constraints. This failed for two reasons.

Firstly, it was not possible to incorporate the gate constraint into the makespan model (recall Equation (3.8)). The makespan model itself is more akin to the GBPP's formulation than it is to the deterministic model's. This is because it is ignorant of the nature of timeslots. We could have instead used the unified model to convert this into a makespan problem, but this formulation would be extremely complicated to solve, and this amount of effort was not warranted to test the efficacy of a heuristic approach.

Next, in such a formulation, the number of objective functions is not prescribed. This could be solved by accepting another complication, that is, over-estimating the number of objective functions needed, and keeping track of which objective function, i.e. a van, is in

use. This again relies on properties that are only found in the approximate and unified model.

With the last two points in mind, even we relax the gate constraint, and overestimate the objective functions a little more intelligently, the resulting problem is still computationally hard. The weighting method presented a number of computational issues as the scalarised objective function could have very large numbers, e.g. $\geq 2^{100}$, which caused both computational slow down and numerical issues. Even when a specific day of work was truncated to just 30 itineraries, a solver ended up taking roughly 20 minutes to decide on an optimal schedule. At this point, this approach was abandoned.

From this failure, a simpler method became much clearer. We could obey the gate constraints and simply sort a schedule from the deterministic model before applying LPT. Surely, this would not be makespan optimal but it could potentially still prove beneficial with some experimental analysis. Upon this thought, we had an intuition that the deterministic model with the gate constraint could be makespan optimal.

5.3.2 The Gate Constraint

The gate constraint is of interest to us because there was a suspicion that the deterministic model without the gate constraint would be equivalent or very similar to the makespan problem. The deterministic model without the gate constraints is henceforth referred to as the *nogate* model. If this is in fact the case, it could be used as leverage to determine properties about LPT applied to the deterministic model. To test this, we could simply obtain all the makespan values of optimal solutions to the nogate model and a makespan model.

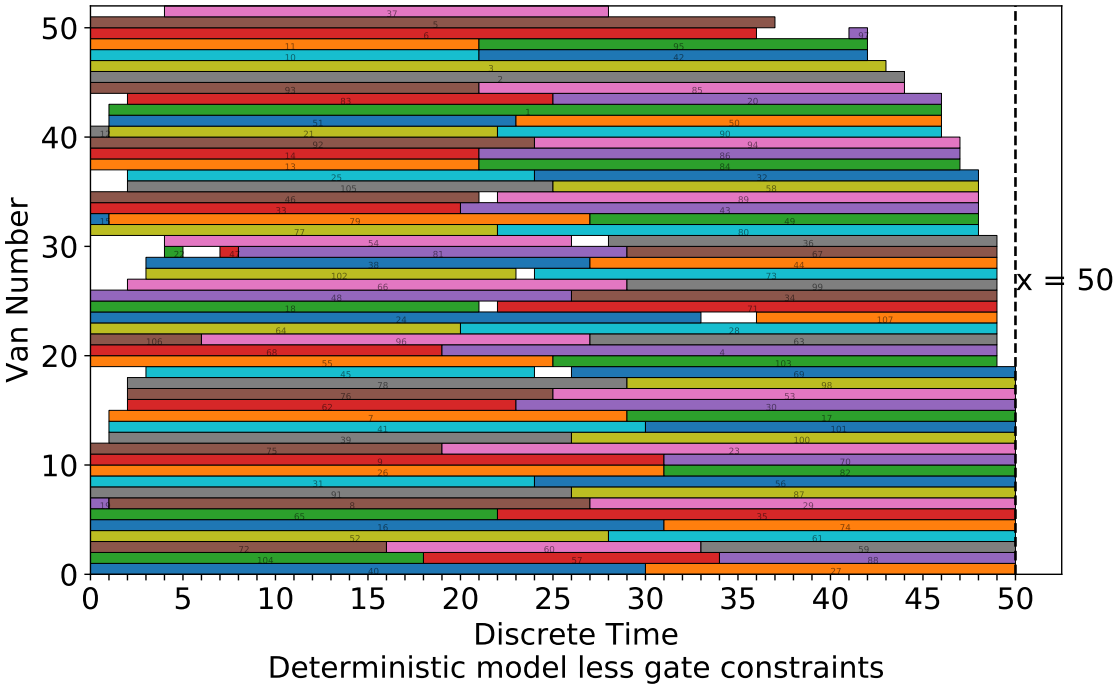
However, if we are to make comparisons with the makespan model, we are required to specify the number of vans available. This is because the makespan problem does not prescribe the optimal number of vans available.

As such, two makespan models are used, termed *makespan-1* and *makespan-2*. Their formulations are identical (Equations (2.14) to (2.18)), but the number of vans available for the models differ. For makespan-1, the number of vans available is set to the optimal value of a schedule from the deterministic model. For makespan-2, it is set to an optimal value from a schedule from the nogate model.

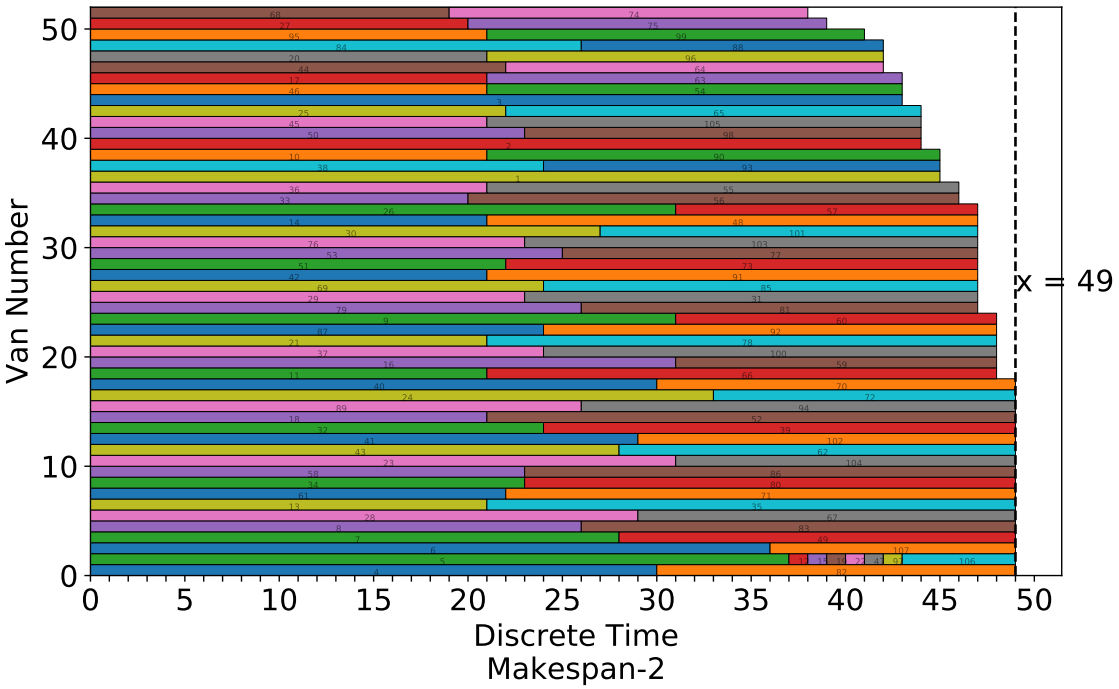
The reasoning for having two formulations is as follows. The gate constraint results in the "drifting" of schedules to the right as can be seen in Figure 4.5a. Without the gate constraint, solutions should be more tightly packed, and as it lacks this drifting, and it could fit into a smaller number of vans. Thus having a gate constrain could more vans available, which could mean that the optimal makespan value can be smaller.

This justification was quickly proved with some basic tests. In the data set, there exists some instances where makespan-1 is smaller than makespan-2 (in terms of the makespan value), but the reverse is never the case.

From the data analysis, it can be concluded that the nogate model is not makespan optimal as there exist multiple instances where the makespan-2 problem produces a smaller optimal makespan value than the nogate model. As an example two schedules produced from the same day in delivery office B are plotted in Figure 5.1 below. If it is the case that the nogate model is not makespan optimal, surely the deterministic model is also not makespan optimal. This is because the latter is a constrained version of the former.



(a) Optimal nogate schedule with makespan of 50



(b) Optimal makespan-2 schedule with makespan of 49

Figure 5.1: Above we present a scenario where the deterministic model *without* the gate constraint decides on an optimal schedule that is *not* makespan optimal. Here the dotted lines refer to the makespan value, i.e. the longest time taken by any one van.

The x-axis represents the discretised timeslots S. In this figure, $T = 50$, i.e. there are 50 timeslots in the time horizon of the day. The y-axis represents each van, i.e, each row is some van's schedule.

5.3.3 Disturbance Analysis

We computationally evaluate the effectiveness of LPT recovery applied to the deterministic model, and to do so, the following steps were taken. First the Royal Mail data was solved in four separate ways, using the makespan models (makespan-1 and makespan-2), the deterministic model, and the nogate model. The schedules were then run through the perturbation generation algorithm (Algorithms 7 to 9) and recovered with LPT, while measuring its disturbance score at all steps. The scores are aggregated as this process is rerun $1,000$ times for each day.

The scores were aggregated to get the (non-robust) mean of the score. The results from this process are detailed in Tables 5.2 and 5.3 below. The measures are aliased to α, β, and γ for simplicity's sake. The measures are explained in Table 5.1 below.

Measure	Description
α	This is the absolute decrease in the disturbance score achieved by recovery methods on the perturbed solution (from step 2 to 3)
β	This is the disturbance score of the perturbed solutions (step 2)
γ	This is the disturbance score of the recovered solutions (step 3)

Table 5.1: Disturbance Analysis Measures

As a reminder, the absolute decrease in the disturbance, α, is largely independent of the number of vans used (recall 4.5). This makes it a highly suitable measure to base our forthcoming analyses on. Here we look at the non-robust means and deviations of the above statistical measures, as we do not seek to discredit outliers.

		Model			
		Deterministic	nogate	Makespan-1	Makespan-2
α	μ	0.662	0.423	0.072	0.07
	σ	0.564	0.462	0.172	0.172
β	μ	10.108	8.788	6.805	6.834
	σ	2.334	1.918	2.524	2.583
γ	μ	9.446	8.365	6.734	6.764
	σ	2.39	1.927	2.484	2.544

Table 5.2: Disturbance score comparisons for the above four models across the entire data set

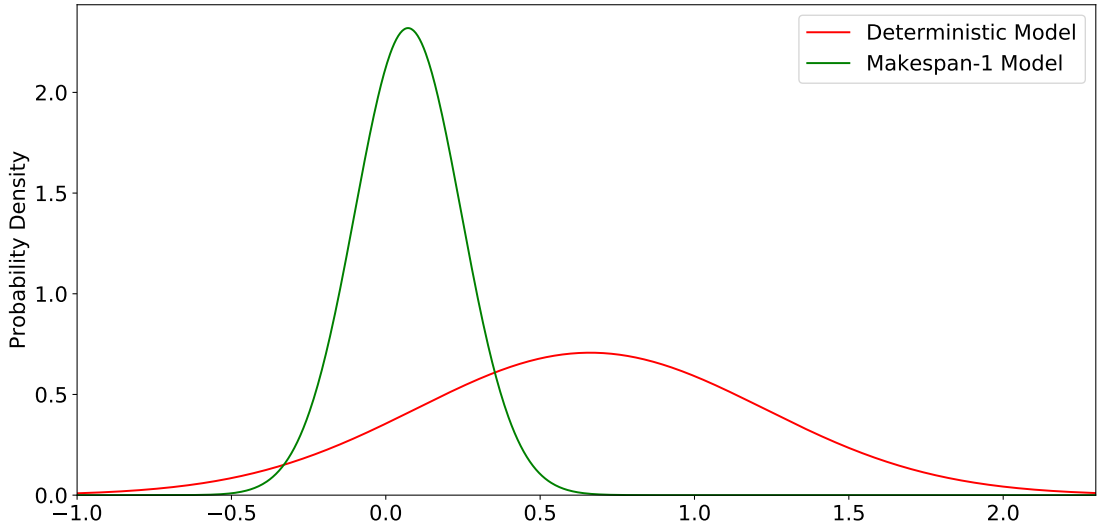
We also note the sample sizes for each of the four models, given the use of the solution pool feature:

Model	Sample size (n)
Deterministic	1502
nogate	1494
makespan-1	1174
makespan-2	1169

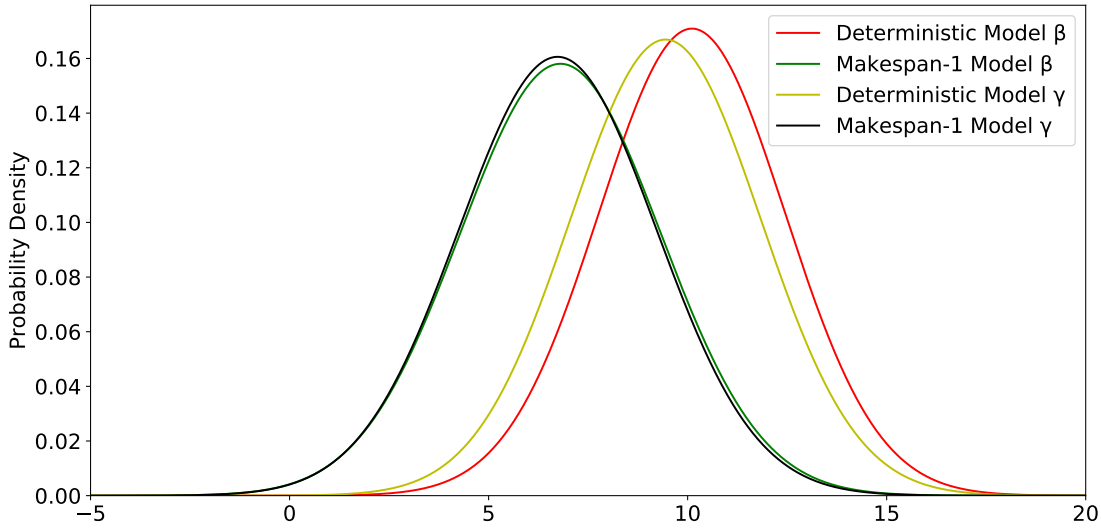
Table 5.3: Sample sizes obtained for each of the four modelling methods with Gurobi's solution pool feature

First, note that both makespan models barely differ with respect to all measures, suggesting that they are largely equivalent. As such, we henceforth opt to only make comparisons to makespan-1, ignoring makespan-2 as we assume to same comparisons can be made to the latter.

From the data above, the assumption is made that the measures are normally distributed. This is not unreasonable given the size of the sample of data. We then plot the distributions of the statistical measures α, β and γ. We will at a later point, we evaluate the validity of assuming such parameters are normally distributed.



(a) Depicted are the distributions of the α values for the deterministic and makespan models. Here we observe the strong guarantees of the makespan problem as observed by its tighter distribution.



(b) Depicted are the distributions of the β and γ values for the deterministic and makespan models. Here we observe the optimal substructure of the makespan model providing a better initial solution, which suggests the alpha value cannot be solely used to understand the efficacy of Lexicographical Optimisation and LPT.

Looking at Figure 5.2a, the distribution of the deterministic model's α measure is not particularly convincing. Even though it has a higher mean α, it's deviation is large and non-negligible.

As LPT provides strong guarantees for the makespan problem, it is logical that it possesses a tighter distribution as depicted above. This is observably different for the deterministic model. Furthermore, the distributions have a non negligible amount of overlap.

We also observe further the distributions of β and γ in Figure 5.2b. The makespan model produces much lower scores. This can be attributed to the optimal substructure imposed by lexicographical optimisation, which reduces `spill_time` [29]. This is a drawback of the deterministic model with the LPT algorithm as it is ultimately unable to overcome degeneracies without the imposition of an optimal substructure of the initial optimal solution.

With both the high γ value and poor α distribution, we conclude that we *cannot* confidently guarantee the efficacy of LPT applied to the deterministic model. At a later point we will evaluate how this conclusion is affected by the choices of our parameters thus far.

5.3.4 LexOpt Proximity and Recoverability

With the knowledge we have obtained from the previous part, a separate question that arose was the following: *does proximity to lexicographical optimality imply recoverability?*. Perhaps when a solution of any which model maintained some sort of proximity to lexicographical optimality, it was provided the benefits of LPT recovery. In this statement, proximity refers to how far a schedule deviates from lexicographical optimality for the perturbed and recovered solutions from the initial optimal solution.

To gauge proximity, a way to measure lexicographical optimality was needed. To this end, the weighting method (recall Section 2.7.1) was used as an ad hoc scoring method. Before computing weighting value at each step in the 3-step process, the vans had to be sorted based on completion times.

We hypothesise that this is possibly true. A correlation between the weighting value for the perturbed solution and a corresponding reduction in the disturbance score (α) was computed to test our hypothesis. The correlation values, $\rho = 0.24$, suggests a weak correlation. It can thus be said with confidence that proximity to lexicographical optimality does not imply recoverability with respect to the disturbance scoring method.

5.4 Integer Programming Recovery

Here we explore the effectiveness of modelling recovery as a integer program (henceforth referred to as *integer recovery programs*). Liebchen et al. [31] propose the idea of using linear programs, but given the nature of our problem, we will have to use integer programs instead. Here we opt to use the approximate model as a baseline, as the most important relationship for integer recovery programs is that between itineraries and vans, i.e., $y_{i,j}$.

We say that our findings in this section will roughly apply to the unified or deterministic models as well. We make this claim as there is a reasonable expectation that the approximate and deterministic model possess similar disturbance score behaviours after realising uncertainty. This is indeed the case as can be seen in Table 5.4, which measures β (previously defined in Table 5.1) for both the deterministic and approximate models. For the approximate model, our measures are for a smaller sample size of $n = 301$.

Disturbance Parameter Name	Cost Function Application
`num_affected`	The number of vans that would be affected is entirely dependent on the uncertainty realised. It would not be possible to encode this into the cost function without some information about stochastic disturbances. Furthermore, it is not part of the model itself, so additional extraneous modelling will have to be done.
`num_spill`	As above.
`num_vans_req`	This is part of the perturbation generation method. If the recovery program could allocate new vans, this can be encoded as a cost function.
`total_vans_req`	This is an expression of `num_spill` and `num_vans_req`, as such the logic for the related parameters applies.
`spill_time`	This can be easily modelled with realisation of uncertainty.

Table 5.5: Linking disturbance parameters with cost functions

		Model	
		Deterministic	Approximate
β	μ	10.108	11.117
	σ	2.334	2.587

Table 5.4: Disturbance score comparison for deterministic and approximate model

Thus, we believe that our recovery methods, if applied to either the unified or deterministic model in the same precise fashion, should effectively produce results as if it were applied to the approximate model.

By using the approximate model, we will in essence define recovery methods for the Variable Size Bin Packing Problem (recall Section 3.7.3).

There are a few considerations to be had with this method. The goal is to perform an approximate recovery. It is of no use for us if the entire problem to was resolved as previously mentioned. Out of the two restrictions on recovery (as discussed in Section 5.2), the restriction on computational complexity will have to be relaxed for this method. Computational complexity in this method can only be managed with the size and complexity of the models used. However, this method will give us a much firmer grasp on the costs of recovery.

To model these integer recovery programs, we shall tie it back to the disturbance score (Section 4.5), as it already encodes what is unfavourable in a schedule (Table 4.10). The goal here is to link the costs of recovery to the ideas underpinning the disturbance score, as we do in Table 5.5.

Thus the cost functions, which are to be minimised, can be modelled after disturbance score parameters `spill_time` and `num_vans_req`. An implicit cost function of the recovery process is that we require the recovered solution to remain close to the initial solution [31]. If the recovery were to be given full reign, it could completely deviate from the initial optimal solution, but it is in our interest to remain close to it.

We propose that recovery must ultimately be with respect to `spill_time`, as it dictates feasibility, and as such we propose two implementable approaches and one theoretical

approach. The first of the three approaches is not able to create any new vans. The second approach can create vans as an approach to recovery, albeit with a cost. The last approach is one which can create new vans and "guarantees" feasibility. As shown in the following sections, the first approach produces the smallest model. The first two approaches are thus termed the *simple* and *complex* recovery programs and the last is the *feasibility* recovery program.

Furthermore, since these approaches utilise the approximate model, we can only deal with situations in which the uncertainty is realised on the day of, before delivery. That is to say, itineraries occurring after a certain time, for this recovery method using the approximate model, cannot be fixed. However, if the unified model were to be used instead, we would also have a relationship with timeslots $s \in S$, with which we can increase the flexibility of the recovery programs.

The integer recovery programs' notation is described in Table 5.6, after which all three approaches shall be detailed. After this, all the three methods of recovery will be analysed and compared.

Symbol	Description
J^*	This refers to the vans used from the solution to the approximate model. As such $J^* \subseteq J$
$y_{i,j}$	In this context, this symbol is a constant and it refers to a *fixed solution* from the approximate model. These are **not** decision variables in the integer recovery program.
$z_{i,j}$	These are the recovery decision variables. They have the exact meaning as the decision variables $y_{i,j}$. It refers to whether itinerary i is being executed by van j.
λ_i	These are the cost functions indexed by i
$\boldsymbol{p}$	This is a vector of a realisation of perturbations. p_i refers to the change in processing time that itinerary i experiences.

Table 5.6: Table Detailing the Linear Recovery Program Notation

5.4.1 Simple Recovery Program

$$\min \quad \nu\lambda_1 + \kappa\lambda_2 \tag{5.1}$$

$$\text{s.t.} \quad \lambda_1 \geq \sum_{j \in J^*} \sum_{i \in I} (z_{i,j} - y_{i,j})^2 \tag{5.2}$$

$$\lambda_2 \geq \sum_{j \in J^*} \max\left(\sum_{i \in I} (z_{i,j} \cdot (d_i + \max(p_i, 0))) - b_j, 0 \right) \tag{5.3}$$

$$\sum_{j \in J^*} z_{i,j} = 1 \qquad \forall i \in I \tag{5.4}$$

$$z_{i,j} \in \{0, 1\} \qquad \forall i \in I, j \in J \tag{5.5}$$

(5.1)	The objective is to minimise the cost functions.
(5.2)	The first cost function is the implicit one, where it is imperative to stay close to the initial solution. The inner term in the sum evaluates to 1 when the solution changes, and 0 otherwise.
(5.3)	The second cost function is one that measures total spill time. $\sum_{i \in I} (z_{i,j} \cdot (d_i + \max(p_i, 0)))$ evaluates to the amount of time spent by a van. It is then subtracted with the value b_j as calculated by Algorithm 6. The right hand side then evaluates to the mathematical equivalent of the `spill_time`. Here, to stick to the timetabling condition, we only consider positive perturbations to the processing time. The inner max term can easily be linearised.
(5.4)	An itinerary can only be executed by one van at most.

As should be observable, the entire process does not allow for new vans to be constructed. Though the cost coefficients can be tweaked, but not to ensure feasibility in all cases. This strategy reads along the lines of "given the vans we have how can we avoid overworking them without swapping too many itineraries around?".

Note that the constraint in (5.2) can easily be linearised. In the scenario where $y_{i,j}$ is 0, just utilise the value of $z_{i,j}$. Otherwise the inner term is equivalent to $y_{i,j} - z_{i,j}$. Furthermore, the term $max(p_i, 0)$ can also be linearised as a preprocessing step before the using the model. This is because the processing time of itinerary i is information obtained prior, i.e. a constant.

5.4.2 Complex Recovery Program

$$\min \quad \nu\lambda_1 + \kappa\lambda_2 + \rho\lambda_3 \tag{5.6}$$

$$\text{s.t.} \quad \lambda_1 \geq \sum_{j \in J^*} \sum_{i \in I} (z_{i,j} - y_{i,j})^2 \tag{5.7}$$

$$\lambda_2 \geq \sum_{j \in J} \max \left(\sum_{i \in I} (z_{i,j} \cdot (d_i + \max(p_i, 0)) - b_j, 0 \right) \tag{5.8}$$

$$\lambda_3 \geq \sum_{j \in J \setminus J^*} e_j \tag{5.9}$$

$$e_j = \min(\sum_{i \in I} z_{i,j}, 1) \qquad \forall j \in J \setminus J^* \tag{5.10}$$

$$\sum_{j \in J} z_{i,j} = 1 \qquad \forall i \in I \tag{5.11}$$

$$e_j, z_{i,j} \in \{0, 1\} \qquad \forall i \in I, j \in J^* \tag{5.12}$$

(5.6)	The objective is to minimise the cost functions.
(5.7)	The is the cost function same as (5.2). Note that it might seem that this should be done over the entire space J instead of J^*. However, if a new van were to be used the new cost function λ_3 would already account for this.
(5.8)	This is the same as (5.3), however it is performed over the entire space of J. This means we measure `spill_time` across both preexisting and new vans.
(5.9)	λ_3 is the number of new vans used.
(5.10)	This is the same formulation as the Approximate Model in (3.23). However it is quantified over the restricted space of vans not previously used to measure the number of new vans used.
(5.11)	This is the same formulation as the Approximate Model in (3.24). This should be quantified over the entire space of vans J.

This model on the other hand, can assign new vans. The model is larger but could provide stronger guarantees like feasibility if the cost coefficients are picked correctly. A recovery program that specifically encodes feasibility while also minimising associated costs is detailed below.

5.4.3 Feasibility Recovery Program

$$\min \quad \nu\lambda_1 + \rho\lambda_3 \tag{5.13}$$

$$\text{s.t.} \quad \sum_{i\in I} (z_{i,j} \cdot (d_i + \max(p_i, 0)) - b_j \leq 0 \qquad \forall j \in J \tag{5.14}$$

$$\lambda_1 \geq \sum_{j\in J^*} \sum_{i\in I} (z_{i,j} - y_{i,j})^2 \tag{5.15}$$

$$\lambda_3 \geq \sum_{j\in J\setminus J^*} e_j \tag{5.16}$$

$$e_j = \min(\sum_{i\in I} z_{i,j}, 1) \qquad \forall j \in J \setminus J^* \tag{5.17}$$

$$\sum_{j\in J} z_{i,j} = 1 \qquad \forall i \in I \tag{5.18}$$

$$e_j, z_{i,j} \in \{0, 1\} \qquad \forall i \in I, j \in J^* \tag{5.19}$$

(5.13)	The objective is to minimise the cost functions, excluding spill times λ_2
(5.14)	Similar to the complex and simple recovery programs, however, feasibility is a requirement, i.e. no `spill_time`. Thus a simpler formulation for the constraint is possible with no need for the binary min function.
(5.15)-(5.18)	Identical to the complex recovery program

Although this model is made to guarantee feasibility, there will still on occasion be scenarios where feasibility is not possible. This is because the bin sizes, b_j, get progressively smaller. This means, there could be long itineraries which makes it such that it is impossible to avoid `spill_time`. As such, we only concern ourselves with the simple and complex recovery program.

5.4.4 Relationship to the Generalised Bin Packing Problem

Recall Section 3.7.3, in which it was shown that the approximate model is in fact an instance of the Generalised Bin Packing Problem (GBPP), termed the Variable Size Bin Packing Problem (VSBPP).

A common issue of the GBPP, and instances of it, is cases of infeasibility. In specific scenarios, it may occur that an item that is to be assigned, i.e. a compulsory item, cannot be assigned while retaining a feasible solution. This is because the "available bins are not sufficient to load all compulsory items" [1]. We observe this phenomena, which essentially dictates our separate formulations of the complex and feasibility recovery programs. This phenomena is what makes the feasibility recovery program impractical to utilise.

To tackle this, in the formulation of the recovery methods for Royal Mail, we penalise `spill_time`, i.e. what makes a schedule infeasible. As such we are applying a cost to this, and this mirrors exactly the way in which it is handled in the literature. Baldi et al. [1] similarly propose using a special bin for this, with a sufficiently large volume and cost larger than the total cost of all regular bins. The GBPP formulation then tries to minimise use of this special bin.

5.4.5 Analysis

Here we analyse the performance of integer recovery programs proposed above. There are several items of comparison here. The disturbance score and the price function (not to be confused with the cost functions) are mandatory measures. Henceforth, when using the price function, we set v^*_{best} as the optimal value from the deterministic model.

Furthermore, since the computational complexity is no longer limited like the LPT recovery method, it would be imperative to measure the execution times. The cost functions are also prefixed with *cost coefficients*, κ, ν and ρ. The values of these coefficients are reasoned about below. Lastly, the optimal values of the cost functions after recovery will need to be compared.

The process of comparison will be similar to that carried out in 5.3.3, with the 3-step process. Schedules will be run through the perturbation generation algorithm (Algorithms

7 to 9) and then recovered several times. Since the running times would be larger, the disturbance scores that are measured of the schedules are averaged over 10 runs of perturbations instead of 1,000, which is done in Section 5.3.3. Furthermore, when using the complex and feasibility recovery programs, the values of λ_3 will have to be reintroduced back into the disturbance score via `num_vans_req` (recall Section 4.5).

The permutations of cost function coefficients are reasoned about as follows. For the simple recovery program, If we fix $\nu = 1$, the objective function then reads "minimise the costs but for every unit of `spill_time` it is worth taking up to κ swaps to prevent it". Without fixing ν, it would then read "minimise the costs but for every unit of `spill_time` it is worth taking up to $\frac{\kappa}{\nu}$ swaps to prevent it". It can also be read as "minimise the costs but for every unit of `swap` it is worth taking up to $\frac{\nu}{\kappa}$ units of `spill_time` to prevent it".

Next, the complex recovery program's objective is a little more convoluted to reason about. First note that the objective function is effectively equivalent to

$$\lambda_1 + \frac{\kappa}{\nu}\lambda_2 + \frac{\rho}{\nu}\lambda_3$$

So if one new van were to be allocated (i.e. λ_3 increases by 1), the total cost would increase by $\frac{\rho}{\nu}$. Thus for each new van potentially being used, if one were able to find x swaps that reduces spill time by y such that $x - \frac{\kappa}{\nu}y < -\frac{\rho}{\nu}$, then the new van is not needed. The left hand side of the above resulting inequality can be recursively reasoned with again.

The different combinations of the coefficients used as part of analysis are detailed in Table 5.7. It will then be shown that if the cost coefficients are carefully thought out, it can prove to be beneficial.

Coefficient Index	Type	ν	κ	ρ	Reasoning
0	Simple	1	1	$\varnothing$	Minimise all costs equally
1	Simple	1	2	$\varnothing$	It is worth performing two swaps if it reduces `spill_time`. In the Simple recovery model, with the limited set of machines J^*, swaps already happen in pairs.
2	Complex	1	1	1	Minimise all costs equally
3	Complex	1	1	0	Use as many vans needed to minimise `spill_time` while not performing too many swaps.
4	Complex	1	2	10	Only use a new van if there is no amount of x swaps that results in a reduction y units of `spill_time` such that $x - 2y < -10$
5	Complex	1	15	10	With a strong penalty on spills, this will seek to obtain a feasible solution. Only allow for unit `spill_time` if there is no amount of swaps and new vans that prevents the spill such that $x + 10z < 20$

Table 5.7: Coefficient Combinations for Analysis

The processes are run on an internal Imperial College London CSG Cloudstack Machine. This is an Ubuntu machine which is equipped with a 2-core Intel(R) Xeon(R) CPU model E5-4650 v2 running at 2.40GHz. It also comes with roughly 2030 megabytes of memory. Furthermore, running times are capped at two minutes, and the best incumbent solution is used.

		Coefficient Index					
		0	1	2	3	4	5
α	μ	0.054	0.892	0.457	-0.271	1.32	5.456
	σ	0.0	0.925	0.457	0.452	1.613	3.015
β	μ	9.755	9.83	9.679	9.943	9.812	9.844
	σ	2.638	2.304	2.526	2.687	2.534	2.33
γ	μ	9.677	8.85	9.199	10.185	8.349	4.307
	σ	2.661	2.588	2.327	2.715	3.375	1.75

Table 5.8: Disturbance score comparisons for the above four models across all cost coefficient combinations

		Coefficient Index					
		0	1	2	3	4	5
λ_1	μ	0.229	11.434	5.473	9.124	11.885	20.428
	σ	0.453	6.993	3.001	4.626	6.997	10.119
λ_2 (step 2)	μ	11.77	20.437	20.662	20.313	20.229	20.281
	σ	6.838	10.884	11.106	10.972	11.127	11.131
λ_2 (step 3)	μ	11.431	10.514	4.479	0.625	4.757	0.097
	σ	6.708	8.608	3.363	1.438	3.018	0.271
λ_2 (% decrease)	μ	0.033	0.507	0.765	0.973	0.726	0.992
	σ	0.07	0.215	0.16	0.07	0.197	0.059
λ_3	μ	0.0	0.0	3.034	7.723	0.642	0.973
	σ	0.0	0.0	1.646	3.942	0.555	0.661
Price function	μ	1.0	1.0	1.061	1.152	1.013	1.02
	σ	0.0	0.0	0.022	0.04	0.01	0.012

Table 5.9: Cost and price function comparisons for all the models across all cost coefficient combinations

		Coefficient Index					
		0	1	2	3	4	5
Runtime (in seconds)	μ_r	0.149	8.724	3.483	0.229	39.747	40.901
	σ_r	0.136	7.288	1.986	0.223	41.722	43.027

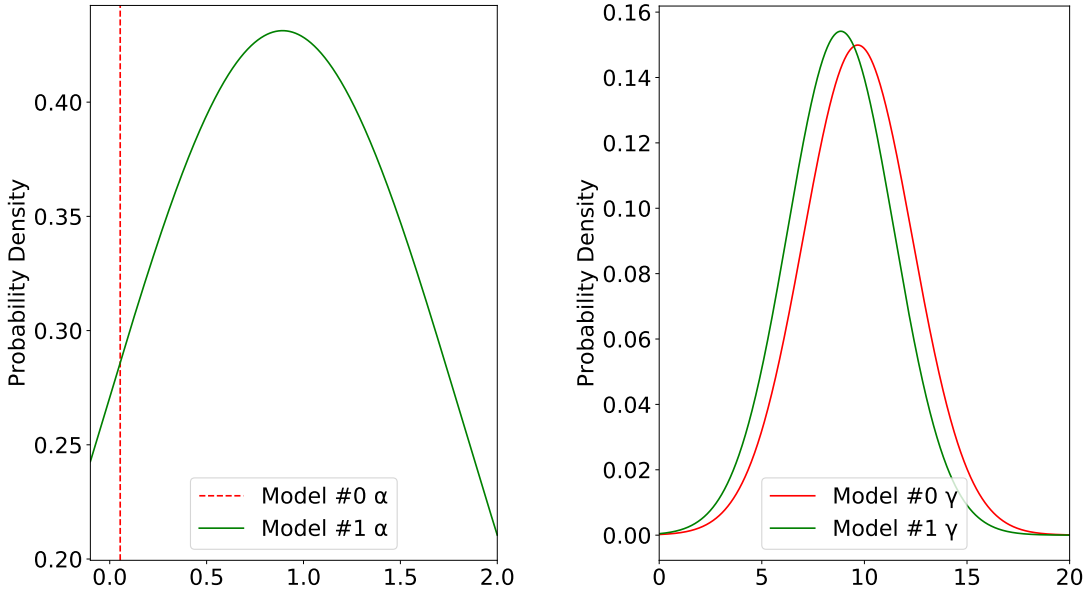
Table 5.10: Runtime comparisons for all the models across all cost coefficient combinations

The first table, Table 5.8, presents the same set of measures as Table 5.2, i.e., an analysis with respect to the disturbance score.

Table 5.9 presents an analysis of the cost function values. For λ_2, which is a measure of `spill_time`, the value of `spill_time` is measured for the perturbed and recovered solutions (step 2 and 3 respectively), along with its decrease as a percentage. The table also highlights λ_3, which is a measure of new vans used. This is also coupled with a price function (recall Section 2.5.2). Obviously the first two models are simple recovery programs and as such they assign no new vans. Lastly, Table 5.10 presents the running time of all

models. Here we opt to use robust statistical measures (i.e. μ_r and σ_r) given the nature of running times.

The values presented are convincing of the fact that recovery works across the board for integer recovery programs, as we shall show below. Once again, we assume that the data in the tables above is normally distributed. For this analysis, given the more complicated and computationally costlier nature of the analysis, we shall opt to *not* use Gurobi's solution pool feature, and thus, sample sizes are as previously described in Table 4.1.



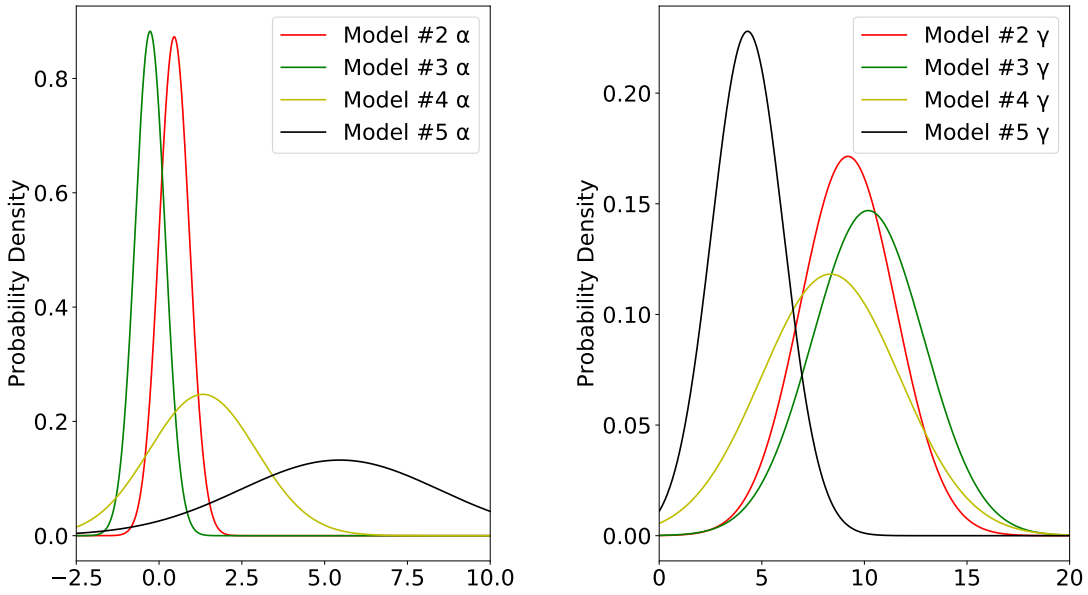
(a) Comparison of the Distribution of α

(b) Comparison of the Distribution of γ

Figure 5.2: A Comparison of the Simple Recovery Programs. Both distributions suggest that model #1 outperforms #0. In figure (a), model #0 has a negligible amount of variance in its α

Below we refer to some coefficient change index simply as "#1", for example. The best performers for the simple recovery programs are rather clear, as can be seen in Figure 5.2. Model #1, which is reasoned about more meaningfully clearly outperforms model #0. It is the better of the two simple models. It performs better both with respect to α and γ. This is to say that it reduces disturbance better overall during recovery, and produces a better recovered solution. One thing to note is the spread of α of #1 and the lack of spread of #0. This indicates that with some small probability the reduction of disturbance performs worse than model #0, but for the most part #1 performs better.

Since model # 0 has equal cost coefficients, it does not accurately penalise `spill_time`. This can clearly be observed by the % decrease in λ_2. It is the case that the latter model takes a larger amount of time on average. It still runs reasonably quick as compared to the extreme cases of the complex model.



(a) Comparison of the Distribution of α

(b) Comparison of the Distribution of γ

Figure 5.3: A Comparison of the Complex Recovery Programs. We observe a wide range of behaviours, but they are predictable in nature, as the deviation of each models α is not erratic. We see that the most thoughtful selection of cost coefficient results in the best performing models.

The complex recovery programs exhibit different, yet intended behaviours. We look at the above four models.

First it is certain, models #2 and #3, whose cost coefficients are not reasoned about well, performs the least favourably. They do not have a competitive α distribution, and regardless of their large decrease of λ_2, it uses a non-negligible amount of vans, resulting and has a high γ. This is to say that because of the poorly reasoned cost coefficients, it not effectively able to reduce the disturbance score. Not surprisingly, model #3 on average *increases* the disturbance score. This is because the amount of new vans it can assign is not penalised. This can be seen by the large values of λ_3 and the price function.

Models #4 and #5 are by far the most interesting. In essence, these models give us a convincing display of recovery properties and have the most well reasoned and intentioned cost coefficients. Model #4 places a reasonable penalty on vans, and we only use a van in a limited subset of scenarios. Model #5 on the other hand places a strict penalty on `spill_time`, trying to coerce feasibility while still having a reasonable restriction on vans. Both models have much more reasonable price functions as compared to both other complex models, with model # 5's obviously being slightly higher. Both models are performant and exhibit predictable behaviour which imply that they can be used in practice with expected behaviour.

Thus, model #4 is one which can be reliably used to recover with as few vans as possible while reducing spill time. Likewise, model #5 can be used to coerce solutions into feasibility with as few vans as possible. Ultimately, we can assume that one would prefer to use model #5 as it checks all the necessary boxes.

One caveat of the last two complex models is the running times. The high mean running times are not encouraging, and the high deviation is even less so. This implies that the

programs are hitting the running time cap of 2 minutes. Still, the best incumbent solution is performant, implying potential symmetry issues in the recovery programs.

Ultimately, from this we conclude that recovery integer programs are in fact an effective method of reducing disturbance and recovering to a good solution after realising uncertainty. This is given that the costs of recovery are well defined and reasoned about.

5.5 Summary

Thus, this chapter contributes an analysis of the applicability and efficacy of two different approaches to recovery methods in the process of optimisation under uncertainty. Namely, the heuristic longest processing time (LPT) recovery method, and integer programs as recovery.

To do so, we first define the constraints on recovery methods, taken from Liebchen et al. [31]. We then show experimentally that the LPT heuristic method of recovery does not conclusively work for the deterministic model as it does for a lexicographically optimal makespan problem [29]. Lastly, we explore different representations of recovery as integer programs. In doing this, we propose two practical representations of recovery programs and one theoretical. We then reason about and verify the positive performance guarantees, while exploring the extent of these guarantees.

Furthermore, and to the best of our knowledge, the VSBPP under uncertainty is not studied extensively. Although its generalisation (the GBPP) has been studied under uncertainty, it is done so in a limited manner, as previously discussed in Section 2.8.2 [39, 42]. Thus, to the best of our knowledge, in the second half of this chapter, we propose new and effective methods of constrained recovery for the VSBPP under uncertainty.

In Appendix C, we attach multiple visualisations of some of the different methods of recovery applied to scheduling problems.

Chapter 6

Schedule Design

6.1 Motivation

We now have a firm understanding of the recovery process. More specifically, we now know that integer recovery programs are a largely viable approach to performing recovery in the case of Royal Mail's problem.

With this knowledge, it is in our interests to explore and prescribe holistic scheduling strategies under uncertainty. We would like to incorporate our findings on recovery, and use the framework we have laid for comparing schedules to do this. However, we can also qualitatively asses the models that we design. Once again, it is imperative that we look at multiple approaches to designing robust schedules, some theoretical, some practical.

6.2 Design Criteria

The deterministic and approximate model both exhibit a high disturbance scores for the perturbed solution (that is, in step 2 of the 3-step process). We have shown the efficacy of using integer programs as a method of recovery (between steps 2 and 3). In this section we look to analyse how we can design schedules so that the disturbance score does not spike radically for the perturbed solution. Once again, we will have to do this with respect to the price function (recall Section 2.5.2), as it is imperative that we try to minimise the number of vans. Thus our main point of focus should be both β, the mean disturbance score for the perturbed solution (recall Table 5.1), and the price function (recall Section 2.5.2).

6.3 Buffering

First we explore the viability of generating schedules with buffers, or, "running time supplements" [28, 30].

Here we will explore two types of buffering methods. First is static buffering, which is a method of pre-computing buffer values of each itinerary prior to the optimisation process. This would mean inserting buf_i amount of space after itinerary i, to absorb "shocks" from stochastic disturbances. The next method is budgeted buffering. In this method, buffers are allocated from a fixed budget of total buffering time B. The notation used for buffering is described in Table 6.1.

Symbol	Description
B	Budget of total buffering time.
N	Budget discretisation size. A budget of total buffering time is can be represented as set of buffers whose cardinality is N.
B_n	Some discrete buffer as part of the budget such that $\sum_{n=1}^{N} B_n = B$
$q_{i,n}$	A binary decision variable which is true when itinerary $i \in I$ uses some buffer B_n after it.
R	The number of uncertainty realisations
$\mathbf{\Delta} \in \mathbb{R}^{\lvert I \rvert \times R}$	Matrix of uncertainty realisation. Each row represents uncertainty realisations for some itinerary i. Each column represents a set of realisations.. Each column represents a set of realisations.
buf_i	Buffer appended *after* itinerary $i \in I$
$shock_z$	The size of a perturbation experienced in z% of the cases.

Table 6.1: Table Detailing the Buffering Terminology

6.3.1 Buffering and Robust Optimisation

Buffering can be seen as a method of robust optimisation for our scheduling problem. If buffers are informed by prior data, for example a confidence interval, it would hedge against what are in essence uncertainty sets. However, as with robust optimisation, buffering could also bring along with it same set of problems of robust optimisation like over conservative solutions (Section 2.5.2) [6]. As we explore different anticipative buffering strategies, it is imperative we keep the price function in mind.

6.3.2 Static Buffering

Static buffering is an obvious and largely naive method of buffering. Still it is in our interests to see how well this would work. A highly unproductive method of doing this would be to insert static buffers, e.g. $buf_i = 1$. Regardless, doing this has noticeable and expected benefits. It dampens the "shocks" from stochastic disturbances introduced by the perturbation generation algorithm. When the value of T is 50, and `work_hours` $= 10$, a single timeslot of buffer correlates to 20 minutes.

However, as previously mentioned this is an unproductive and unfounded method. Furthermore, the amount of buffering time decided upon cannot be done arbitrarily in this manner as the size of T can be modified by the optimiser.

A more grounded method would be to model buffering periods by statistical data. To do this, the robust statistic for each day of the week per DO is pre-computed. Then with the robust mean and standard deviation, a z% (robust) confidence interval can be obtained. With the mean μ_r and upper bound of the confidence interval u, we can say that we are z% confident that on average the perturbation shock is

$$\text{shock}_{z\%} = \frac{u - \mu_r}{u}$$

If we assume that on average the total time executed by a van is T, the value above multiplied with the T would give us approximately the total perturbation that would be experienced by a van for z% of the time.

$$buf_i = \lceil \text{shock}_{z\%} \cdot T \rceil$$

This could be the constant buffer value for every itinerary. There are some obvious issues with this, namely its conservatism. Another approach is to assign different buffer values per itinerary, proportional to its duration. As such for an itinerary i, its buffer would be

$$buf_i = \lceil \text{shock}_{z\%} \cdot d_i \rceil$$

This is more in line with the perturbation generation algorithm (Algorithms 7 to 9).

To implement buffers, it is sufficient to use the deterministic model and make an extension to the definition of set $S_{i,t}$. As previously described in Table 3.4, it is the time slots in which itinerary i must have begun in order to be being executed at time slot t. If an itinerary i has a buffering time buf_i after it, the set can be instead defined as

$$S_{i,t,buf_i} = \{t - d_i + 1, t - d_i + 2, \cdots, t + buf_i\}$$

By doing so, a buffer is being appended to each itinerary. It does not matter if for any t that $t + buf_i > T$, since all constraints are only quantified and restricted to T.

The results from the three buffering methods are given below. The relevant values from the deterministic model are provided as a frame of reference. We use 95% confidence intervals in for $\text{shock}_{z\%}$.

#	buf_i	β		Price Function	
		μ	σ	μ	σ
0	0	10.108	2.334	1	0
1	1	11.198	2.101	1.022	0.017
2	$\lceil \text{shock}_{95\%} \cdot T \rceil$	12.604	1.779	1.11	0.069
3	$\lceil \text{shock}_{95\%} \cdot d_i \rceil$	9.546	2.388	1.05164	0.037

Table 6.2: Comparison of buffering methods

The results here are generally as one would expect. Static buffering can be used to improve a schedules β value, and if applied with a little care, one can avoid the over conservatism of robust optimisation and an inflation in the price function. However, this method is ultimately still naive and lacks intentions. The optimiser here is also presented with much lesser control over a schedule, as we shall show with budgeted buffering.

6.3.3 Budgeted Buffering

In the context of railway timetabling, Kroon et al. along with Liebchen and Stiller have proven that a proportional allocation of buffers is in fact suboptimal [28, 30]. Though this is "common in practice", the former paper argues and proves that it is better to allocate buffers earlier rather than later. This is because a perturbation to an earlier itinerary affects

a larger number of itineraries after it (`num_affected`). It also argues that a timetable can be made more robust by "reallocating" buffers.

These findings are certainly applicable to the context of Royal Mail's problem and the techniques employed to reach these conclusions in the former paper, is of a great deal of interest to our problem. From this paper's methodology we propose further methods of buffering schedules, and what we shall term *Budgeted Buffering*, where buffers are allocated from a budget B of total buffering time. Furthermore, the problem of allocating buffers itself is now part of the model. Once again we will utilise the approximate model as the deterministic model does not have the flexibility to represent buffers.

First, we note that the size of B can be quite easily modelled by a priori statistical data. For example we can reasonably say for 95% of time, the total buffering needed is

$$B = \left(\sum_{i \in I} d_i \right) \cdot \text{shock}_{95\%}$$

Next, it must be noted that, a truly flexible allocation of buffers leads to complicated non-linear models for the Generalised Bin Packing Problem, which the approximate model is. By "truly flexible", we mean that both the buffer size and the choice of where to allocate a buffer are part of the optimisation process. This is possibly linearisable, but a less flexible formulation already presents itself as tricky to linearise, as we show below. In this less flexible approach, we discretise the budget B into smaller buffers B_n such that

$$\sum_{n=1}^{N} B_n = B$$

where N is the size of the discretisation. This discretisation should be done such that for all buffer $B_n \neq 0$. After doing this we can then define buf_i as follows.

$$buf_i = \sum_{n=1}^{N} q_{i,n} B_n$$

Where $q_{i,n}$ is a binary decision variable which is true when itinerary $i \in I$ uses some buffer B_n after it. However, we still require some sophistication to be able to model this. As has always been a constraint, i.e. Equation (3.25), we make sure the allocation of itineraries in some van j does not spill over the size of a bin b_j. Here we must ensure that the allocated itinerary *and* buffer does not spill in the case of the initial optimal solution (step 1). This is expressed as the following

$$\begin{aligned} \sum_{i \in I} (y_{i,j} \cdot (d_i + \sum_{n=1}^{N} q_{i,n} \cdot B_n)) \leq b_j \qquad & \forall j \in J \\ = \sum_{i \in I} (y_{i,j} \cdot d_i) + \sum_{i \in I} (y_{i,j} \cdot (\sum_{n=1}^{N} q_{i,n} \cdot B_n)) \leq b_j \qquad & \forall j \in J \\ = \sum_{i \in I} (y_{i,j} \cdot d_i) + \sum_{i \in I} \sum_{n=1}^{N} q_{i,n} \cdot y_{i,j} \cdot B_n \leq b_j \qquad & \forall j \in J \end{aligned} \tag{6.1}$$

Note the product of two binary variables $q_{i,n} \cdot y_{i,j}$. This is non-linear in its current representation. Fortunately, the product of two binary variables is in essence a logical AND

operator, which is linearisable. Thus we express $\sum_{i \in I} \sum_{n=1}^{N} q_{i,n} \cdot y_{i,j} \cdot B_n \leq b_j$ as follows

$$\begin{aligned} \sum_{i \in I} \sum_{n=1}^{N} \omega_{i,n,j} \cdot B_n = \sum_{i \in I} \sum_{n=1}^{N} q_{i,n} \cdot y_{i,j} \cdot B_n \leq b_j \qquad & \forall j \in J \\ \text{s.t.} \quad \omega_{i,n,j} \leq q_{i,n} & \\ \omega_{i,n,j} \leq y_{i,j} & \\ \omega_{i,n,j} \geq q_{i,n} + y_{i,j} - 1 & \\ 0 \leq \omega_{i,n,j} \leq 1 & \end{aligned} \tag{6.2}$$

Here, we use ω to represent a linearisation of the product of the two binary decision variables. It is in itself a binary decision variable, however we do not need to impose a binary constraint as the values it can take are only 1 and 0, thus we apply a relaxation to its bounds.

With all this in mind, the first method of budgeted buffering can seek to minimise the price function, given the inherent conservatism of confidence intervals.

Formulation 1

$$\min \ v \qquad (\square)$$

$$\text{s.t.} \quad \sum_{j \in J} e_j = v \qquad (\square)$$

$$e_j = \min(\sum_{i \in I} y_{i,j}, 1) \qquad \forall j \in J \qquad (\square)$$

$$\sum_{j \in J} y_{i,j} = 1 \qquad \forall i \in I \qquad (\square)$$

$$\sum_{i \in I} y_{i,j} \cdot d_i + \sum_{i \in I} \sum_{n=1}^{N} \omega_{i,n,j} \cdot B_n \leq b_j \qquad \forall j \in J \qquad (6.3)$$

$$\sum_{i \in I} q_{i,n} = 1 \qquad \forall n \in \{1, \cdots, N\} \qquad (6.4)$$

$$\omega_{i,n,j} \leq q_{i,n} \qquad \forall n \in \{1, \cdots, N\}.i \in I.j \in J \qquad (\triangle)$$

$$\omega_{i,n,j} \leq y_{i,j} \qquad \forall n \in \{1, \cdots, N\}.i \in I.j \in J \qquad (\triangle)$$

$$\omega_{i,n,j} \geq q_{i,n} + y_{i,j} - 1 \qquad \forall n \in \{1, \cdots, N\}.i \in I.j \in J \qquad (\triangle)$$

$$0 \leq \omega_{i,n,j} \leq 1 \qquad \forall n \in \{1, \cdots, N\}.i \in I.j \in J \qquad (\triangle)$$

$$q_{i,n}, e_j, y_{i,j} \in \{0, 1\} \qquad \forall i \in I, j \in J \qquad (\square)$$

$$buf_i, b_j, v \in \mathbb{N} \qquad (\square)$$

$\square$	These are the same as the approximate model.
$\triangle$	These are the constraints expressing the linearisation of the product of binary decision variables $q_{i,n}$ and $y_{i,j}$.
(6.3)	Similar to the approximate model, where the duration of an itinerary also includes the buffer time afterwards. The original optimal solution must be feasible.
(6.4)	Each buffer must only used by one itinerary

This method, however, has no regard, for the schedule after the uncertainty realisation (step 2). It's primary goal is to allocate a budget of buffers while minimising the price function / number of vans used. This means that we are not realising the benefits of the buffers allocated.

Thus, this methodology can however be applied with a little more intent and intelligence. It is in our interests to minimise the price, however, we must also anticipatively minimise `spill_time`. To do this, we propose a stochastic delay-resistant scheduling methodology. The goal here is to find an allocation of buffers that minimises both the number of vans used (price function) and minimises the expected spill time across a matrix of uncertainty realisations $\mathbf{\Delta}$.

Formulation 2

$$\min \quad v + E_{\Delta}[\texttt{spill\_time}] \tag{6.5}$$

$$\text{s.t.} \quad \sum_{j \in J} e_j = v \tag{$\square$}$$

$$e_j = \min(\sum_{i \in I} y_{i,j}, 1) \qquad \forall j \in J \tag{$\square$}$$

$$\sum_{j \in J} y_{i,j} = 1 \qquad \forall i \in I \tag{$\square$}$$

$$buf_i = \sum_{n=1}^{N} q_{i,n} \cdot B_n \qquad \forall i \in I \tag{6.6}$$

$$\sum_{i \in I} (y_{i,j} \cdot (d_i + buf_i)) \leq b_j \qquad \forall j \in J \tag{6.7}$$

$$E_{\Delta}[\texttt{spill\_time}] = \frac{1}{R} \sum_{j \in J} \sum_{r=1}^{R} \text{spill}_{r,j} \tag{6.8}$$

$$\text{spill}_{r,j} \geq \left(\sum_{i \in I} y_{i,j} \cdot (d_i + \Delta_{i,r} - buf_i) \right) - b_j \qquad \forall r \in \{1, \cdots, R\}. j \in J \tag{6.9}$$

$$\text{spill}_{r,j} \geq 0 \tag{6.10}$$

$$q_{i,n}, e_j, y_{i,j} \in \{0,1\} \quad spill_{r,j}, buf_i, b_j, v \in \mathbb{N} \qquad \forall i \in I, j \in J \tag{6.11}$$

$$\boldsymbol{\Delta} \in \mathbb{R}^{|I| \times R} \tag{6.12}$$

$\square$	These are the same as the approximate model.
(6.5)	Here we minimise both the number of vans and the average amount of `spill_time` experienced experienced across all realisations of uncertainty.
(6.6)	This is the amount of buffering time appended after itinerary i
(6.7)	Same as (6.3). The nominal allocation of buffers and itineraries decided upon must be a feasible solution.
(6.8)	The expectation of `spill_time` is done with a sample average approximation method. Recalling Section 2.4.1, we average the spill time across R realisations of uncertainty, for all $\|J\|$ vans.
(6.9), (6.10)	The total amount of `spill_time` experienced by van j in realisation r is all its itinerary's durations and how much an itinerary's buffer absorbs the stochastic disturbance. The amount of spill time must be strictly positive. There are two cases to consider here. If the buffering time is larger than the disturbance, $\text{spill}_{r,j}$ is set to 0 by construction. Else, if the buffer time is smaller, we add on to the duration of an itinerary the perturbation length less the buffering time and the spill time is then measures as per usual.
(6.12)	We reappropriate the uncertainty set as an uncertainty matrix here. In essence its function is identical, but the matrix representation lends a hand to convenient syntax.

In this formulation, $\text{spill}_{r,j}$ refers to the `spill_time` experienced by van j in realisation r. We note that the variable buf_i is used purely for readability's sake. Each instance of the product of $y_{i,j}$ and buf_i must be linearised in the same way as in Equation (6.2). We rewrite linearised formulation below

$$
\begin{aligned}
\min \quad & v + E_{\Delta}[\texttt{spill\_time}] \\
\text{s.t.} \quad & \sum_{j \in J} e_j = v \\
& e_j = \min(\sum_{i \in I} y_{i,j}, 1) && \forall j \in J \\
& \sum_{j \in J} y_{i,j} = 1 && \forall i \in I \\
& \sum_{i \in I} y_{i,j} \cdot d_i + \sum_{i \in I} \sum_{n=1}^{N} \omega_{i,n,j} \cdot B_n \leq b_j && \forall j \in J \\
& E_{\Delta}[\texttt{spill\_time}] = \frac{1}{R} \sum_{j \in J} \sum_{r=1}^{R} \text{spill}_{r,j} \\
& \text{spill}_{r,j} \geq \sum_{i \in I} y_{i,j} \cdot (d_i + \Delta_{i,r}) + \sum_{i \in I} \sum_{n=1}^{N} \omega_{i,n,j} \cdot B_n - b_j && \forall r \in \{1, \cdots, R\}.j \in J \\
& \text{spill}_{r,j} \geq 0 \\
& \omega_{i,n,j} \leq q_{i,n} && \forall n \in \{1, \cdots, N\}.i \in I.j \in J \\
& \omega_{i,n,j} \leq y_{i,j} && \forall n \in \{1, \cdots, N\}.i \in I.j \in J \\
& \omega_{i,n,j} \geq q_{i,n} + y_{i,j} - 1 && \forall n \in \{1, \cdots, N\}.i \in I.j \in J \\
& 0 \leq \omega_{i,n,j} \leq 1 && \forall n \in \{1, \cdots, N\}.i \in I.j \in J \\
& q_{i,n}, e_j, y_{i,j} \in \{0, 1\} \quad spill_{r,j}, b_j, v \in \mathbb{N} && \forall i \in I, j \in J \\
& \boldsymbol{\Delta} \in \mathbb{R}^{|I| \times R}
\end{aligned}
$$

The model presented here requires a population of uncertainty matrix $\boldsymbol{\Delta}$. This can be done by sampling uncertainty exactly as done in the perturbation generation algorithms (recall Section 4.4). This is a stochastic optimisation process as previously described in Section 2.4. As we increase the number of realisation R, we are able to more accurately model the expectation of `spill_time`. A solution from this model stochastically hedges against realisations of uncertainty, minimising *both* the disturbance score of a perturbed solution (i.e., β) and the price function.

However, as with the unified model, we are faced with tractability limitations of such an expressive model. Stochastic optimisation is already notoriously difficult, and the size of this model only worsens this.

6.4 Recoverable Robustness

Recall section 2.6, where we roughly introduce the concept of recoverable robustness. Here we flesh out the definition a little more, after which we provide a recoverable robust formulation of the Royal Mail problem. The notation used in this section is presented in Table 6.3

Symbol	Description
P	Set of feasible solutions
$\mathcal{A}$	Class of admissible recovery algorithms
$\hat{\boldsymbol{A}}$	A recovery matrix. This is the matrix of coefficients of constraints imposed during recovery.
$(.)^s$	A realisation of uncertainty for some variable for some scenario $s \in \Delta$
$\boldsymbol{A}^0, \boldsymbol{b}^0$	Initial original coefficients of a linear program.
$\boldsymbol{y}$	Vector of recovery decision variables.
$\boldsymbol{Y}$	matrix of recovery decision variables for all scenarios of uncertainty. This implies that the matrix has R columns for R realisations of uncertainty.
$\boldsymbol{\lambda}$	This represents a vector of cost functions.
D	This scalar represents a limit on the total incurred costs.

Table 6.3: Table Detailing Notation for Recoverable Robustness

The previous general formulation, Equation (2.13), obfuscates the necessary cost functions. We rewrite the problem for (integer) linear programs (LP) as follows and explain the formulation afterwards

$$\min_{\boldsymbol{x}} \boldsymbol{c}^T \boldsymbol{x} \tag{6.13}$$

$$\text{s.t.} \quad \boldsymbol{A}^0 \boldsymbol{x} \leq \boldsymbol{b}^0 \tag{6.14}$$

$$\forall (\boldsymbol{A}, \boldsymbol{b}) \in \Delta \exists \boldsymbol{y} \quad \text{s.t.} \tag{6.15}$$

$$\boldsymbol{A}\boldsymbol{x} + \hat{\boldsymbol{A}}\boldsymbol{y} \leq \boldsymbol{b} \tag{6.16}$$

$$\boldsymbol{\lambda}^T \boldsymbol{y} \leq D \tag{6.17}$$

(6.13)	Minimise the objective.
(6.14)	The solution must be feasible with respect to the original constraints
(6.15)	Δ once again represents realisations of uncertainty. Uncertainty is realised in coefficients $\boldsymbol{A}$ and $\boldsymbol{b}$ [1]. For each realisation of uncertainty, they must exist a vector of recovery decisions y, which fulfills the following properties.
(6.16)	For each realisation of uncertainty, the constraints must still hold true. The coefficients have changed, but it *must* still be feasible.
(6.17)	For each realisation of uncertainty, the recovery decisions $\boldsymbol{y}$ must not incur costs that total a value larger than the cost limit. That is to say, the costs of recovery must be lesser than the upper bound D.

This expression can be simplified with an equivalent *scenario expansion*, which is as fol-

[1] Note that we can ignore uncertainty realisations in the objective without any loss of generalisation. [3]

lows.

$$\begin{aligned} & \min_{\boldsymbol{x},\boldsymbol{Y}} \boldsymbol{c}^T\boldsymbol{x} \\ \text{s.t.} \quad & \boldsymbol{A}^0\boldsymbol{x} \leq \boldsymbol{b}^0 \\ & \boldsymbol{A}^s\boldsymbol{x} + \hat{\boldsymbol{A}}\boldsymbol{y}^s \leq \boldsymbol{b}^s \quad \forall s \in \Delta \\ & \boldsymbol{\lambda}\boldsymbol{y}^s \leq D \quad \forall s \in \Delta \end{aligned} \tag{RecRobust}$$

A scenario expansion essentially is a tractable representation as opposed to the previous definition. We get rid of the existential qualifiers, and we quantify the problem across all scenarios in Δ instead.

We can further improve on this by also minimising on the value of D, instead of a fixed value. We note that scenario sets can be large, providing computational difficulty. With this, and with the linear recovery programs we have previously defined, we shall show that it is in fact very simple to extend upon it. For an example, we describe the recoverable robust model for cost function λ_1 (Section 5.4.1) below.

$$\min \quad v + \lambda_1 \tag{6.18}$$

$$\text{s.t.} \quad \sum_{j \in J} e_j = v \tag{□}$$

$$e_j = \min(\sum_{i \in I} y_{i,j}, 1) \qquad \forall j \in J \tag{□}$$

$$\sum_{j \in J} y_{i,j} = 1 \qquad \forall i \in I \tag{□}$$

$$\sum_{i \in I} (y_{i,j} \cdot d_i) \leq b_j \qquad \forall j \in J \tag{□}$$

$$\lambda_1 \geq \sum_{j \in J^*} \sum_{i \in I} (z^s_{i,j} - y_{i,j})^2 \qquad \forall s \in \Delta \tag{6.19}$$

$$\sum_{i \in I} (z^s_{i,j} \cdot (d_i + p^s_i))) \leq b_j \qquad \forall j \in J.s \in \Delta \tag{6.20}$$

$$\sum_{j \in J^*} z^s_{i,j} = 1 \qquad \forall i \in I.s \in \Delta \tag{6.21}$$

$$z^s_{i,j}, e_j, y_{i,j} \in \{0,1\} \qquad \forall i \in I.j \in J.s \in S \tag{6.22}$$

$$b_j, v \in \mathbb{N} \tag{6.23}$$

(□)	Same as the approximate model.
(6.18)	Minimise the objective *and* cost function λ_1
(6.19)	λ_1 is the cost function measuring how far we are from the original solution.
(6.20)	Solutions must remain feasible in all scenarios. In all scenarios, the total duration of itineraries for some van j cannot exceed the value of b_j, which is the maximum running time.
(6.21)	The solution must remain feasible in all scenarios. In all scenarios, an itinerary can only be assigned to one van.

At this point, it might seem that the model has sufficient expressivity and hedges against uncertainty convincingly, and that might be very well the case. However, we note one critical drawback in the notion of a recoverable robust program. If one were to model the cost function λ_2, it would be largely counter intuitive. A recoverable robust formulation does not allow for any `spill_time`, but, what if it were unavoidable? Recall the feasibility recovery program and its assosciated issues with not guranteeing feasibility (Section 5.4.3). Thus we propose the idea of lightly recoverable robust programs in the following section.

6.5 Light Recoverable Robustness

Here we propose, to the best of our knowledge, the first generalisation of recoverable robust programs. We term this class of problems as *lightly recoverable robust programs* (LRRPs). Here we incorporate ideas from the concept of light robustness as proposed by Fischetti and Monaci [18] (recall Section 2.5.2). We shall also show that the notion of recoverable robust programs are a subset of our generalised method. The notation used in this section is described in Table 6.4.

Symbol	Description
ξ_i	Some slack variable used for row i of an optimisation problem
$\boldsymbol{\xi}$	A vector of slack variables. This must be of the same dimensionality of $\boldsymbol{b}$ as in (LP).
Pr	We use this to represent a price function value. It is essentially a ratio of a solution to some optimal solution. We use this as an upper-bound for the deterioration of our objective.

Table 6.4: Table Detailing the Anticipative Scheduling Terminology

6.5.1 Motivation

The critical drawback in recoverable robust models such as the one proposed in Section 6.4 is that it insists on feasibility. While this might seem like a reasonable thing to do, note that the idea of recoverable robustness and its surrounding literature was proposed with railway scheduling in mind [18, 28, 30, 31].

Railway scheduling is an *aperiodic timetabling problem* [30]. Here, the timetables are not bound by a time horizon in the same way as our problem is. For railway scheduling and aperiodic timetabling, the problem is generally modelled as a directed graph. Each node in the graph is some event, e.g. arrival or departure. The goal is to minimise passenger travel times. Feasibility is attained so long as the durations between two related events, i.e. an edge in the directed graph, is of some time long enough for the activities that occur on the graph.

In fact, there might be scenarios where the constraints need to be a little flexible. For our example, if one van were to spill over the time horizon by one timeslot, it is not necessarily the end of the world. However, this scenario realisation, for some $s \in \Delta$, would mean that our recoverable robust program will fail.

To abstract this notion further, if we were dealing with an optimisation problem where some scenarios would result in infeasibility that cannot be recovered from, it is possible that we might still want to recover *as best as we can*.

6.5.2 Formulation

First we formulate a linear LRRP, after which it should be evident how it extends to both integer programs and non-linear programs. We first define the notion of *slack*. Note that our constraints in a linear optimisation problem are of the form

$$\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$$

and each row of this expression can be represented as

$$\sum_j a_{i,j} x_j \leq b_i \qquad \forall i$$

When an uncertainty is to break this constraint, we can can instead use a slack ξ_i on the right hand side to preserve the constraint. Slack values must be strictly non-negative. With the slacks, we will seek to minimise the amount of slack used. This effectively means that we are trying to stay as close to an *optimal feasible* solution as possible. Revisiting the case of one van spilling over the time horizon by one slot; we now can use at most one timeslot of slack to get a good solution that is "not too far" from optimality. [18, 31].

The objective function also has to modified. Certainly, such near-optimal solutions result in non-optimal objective functions. Thus as Fischetti and Monaci propose, the objective should instead be bounded as a constraint. Furthermore, it should be bounded by an optimal nominal value. In an ideal scenario, this would be an optimal solution from a recoverable robust program. We can use the notion of price functions defined previously in this dissertation to control the level of deterioration from the nominal optimal solution. A linear LRRP is then defined as follows.

$$\min_{\boldsymbol{x},\boldsymbol{Y},\boldsymbol{\xi}} \quad \sum \xi_i \tag{6.24}$$

$$\text{s.t.} \quad \boldsymbol{A}^0\boldsymbol{x} \leq \boldsymbol{b}^0 \tag{6.25}$$

$$\boldsymbol{c}^T\boldsymbol{x} \leq Pr \cdot (\boldsymbol{c}^T\boldsymbol{x}^*) \tag{6.26}$$

$$\boldsymbol{A}^s\boldsymbol{x} + \hat{\boldsymbol{A}}\boldsymbol{y}^s \leq \boldsymbol{b}^s + \boldsymbol{\xi} \qquad \forall s \in \Delta \tag{6.27}$$

$$\boldsymbol{\lambda}\boldsymbol{y}^s \leq D \qquad \forall s \in \Delta \tag{6.28}$$

$$\boldsymbol{\xi} \geq 0 \tag{6.29}$$

$$\tag{6.30}$$

(6.24)	Minimise with respect to $\boldsymbol{x}$, all $\boldsymbol{y}^s$ for each scenario, and $\boldsymbol{\xi}$ the total slack used
(6.25)	The solution must be feasible with respect to the original coefficients.
(6.26)	The objective value $\boldsymbol{c}^T\boldsymbol{x}$ of our first stage decisions $\boldsymbol{x}$ must not deteriorate beyond the price function multiplied with a nominal optimal solution.
(6.27)	For all scenarios, there is some second-stage recovery vector $\boldsymbol{y}^s$ that can restore feasibility with some slack $\boldsymbol{\xi}$. Therefore, $\boldsymbol{\xi}$ is an upper bound of slack values across all scenarios $s \in \Delta$
(6.28)	As always, our actions of recovery must be limited. This is the same as in (RecRobust).
(6.29)	Slack must be strictly non-negative as previously mentioned.

As should be observable, this method can be quite easily generalised to both integer and non-linear optimisation problems, in a similar way that (RecRobust) is [31]. What is fundamentally needed in an LRRP formulation is a representation of slackness to restore feasibility, and a nominal optimal value combined with a target price function value.

6.5.3 Proof of Generalisation

We assert that recoverable robustness is a subset of LRRP, with respect to the space of feasible solutions. By adding constraints to an LRRP problem, we obtain an *equivalent* recoverable robust formulation which demands feasibility. We once again prove this is the case for linear formulations of LRRPs and recoverable robustness, and observe that the case for integer or non-linear programs is rather trivial. The proof of generalisation is as follows. We do this with a proof by contradiction.

1. Let us now take a linear LRRP formulation as in Equations (6.24) to (6.28).
2. We add constraints to the original problem that force slack variables $\forall i.\xi_i = 0$
3. We also set price function $Pr = 1$.
4. We also take $\hat{\boldsymbol{x}}^*$ as an optimal solution to (RecRobust).
5. We assume that this constrained LRRP is not equivalent to (RecRobust). This is to say we assume that $\exists x \; . \; \boldsymbol{c}^T\boldsymbol{x} \neq \boldsymbol{c}^T\hat{\boldsymbol{x}}^*$ [Assumption]
6. The LRRP program now decides on some $\boldsymbol{x}$ and recovery decisions $\boldsymbol{y}^s$ for all scenarios in Δ which are feasible, without any slack. Furthermore, the solution is such that

$$\boldsymbol{c}^T\boldsymbol{x} \leq Pr \cdot (\boldsymbol{c}^T\hat{\boldsymbol{x}}^*)$$

   .
7. If it is the case the LRRP is not equivalent as per our assumption, then the optimal solution from (RecRobust) need not be an lower bound to an LRRP's solution. Thus there might exist some solution $\boldsymbol{x}$ to the linear LRRP problem such that

$$\boldsymbol{c}^T\boldsymbol{x} \neq \boldsymbol{c}^T\hat{\boldsymbol{x}}^* \wedge \boldsymbol{c}^T\boldsymbol{x} \leq \boldsymbol{c}^T\hat{\boldsymbol{x}}^*$$

8. However, this implies that the optimal solution $\hat{\boldsymbol{x}}^*$ is in fact non-optimal. This is because the solution we obtain, $\boldsymbol{x}$, is feasible in (RecRobust) which is better than $\hat{\boldsymbol{x}}^*$. Constraints (6.25) and (6.28) hold trivially as they are identical to the recoverable robust constraints. Constraint (6.27) holds for the recoverable robust model as the slack is now all zero valued. Constraint (6.26) holds as the above point implies that $\boldsymbol{c}^T\boldsymbol{x} < \boldsymbol{c}^T\boldsymbol{x}^*$
9. Thus we have reached a contradiction. We assert $\hat{\boldsymbol{x}}^*$ is optimal for (RecRobust), but through our assumption show that it is not. Thus our assumption must be invalid.
10. Thus the constrained LRRP is equivalent to (RecRobust) if the latter model has some optimal solution $\hat{\boldsymbol{x}}^*$.

If a constrained formulation of a linear LRRP is equivalent to the recoverable robust problem, we can thus conclude that the linear LRRP is a generalisation with respect to its feasible region. This exact same proof can be applied to a non-linear optimisation problem, with which we can conclude that LRRP's are a generalisation of recoverable robust programs.

6.5.4 Formulation of an LRRP for the Royal Mail problem

Presented below is a formulation of an LRRP for the Royal Mail problem specifically. With the notion of LRRPs, we can now model the notion of cost function λ_2. In fact, this cost function is in itself a notion of slack. As always, we would also like to remain close to the planning solution, we have to concern ourselves with minimising λ_1 as well.

$$\min \quad \lambda_2 \tag{6.31}$$

$$\text{s.t.} \quad v \leq Pr \cdot v^* \tag{6.32}$$

$$\sum_{j \in J} e_j = v \tag{□}$$

$$e_j = \min(\sum_{i \in I} y_{i,j}, 1) \qquad \forall j \in J \tag{□}$$

$$\sum_{j \in J} y_{i,j} = 1 \qquad \forall i \in I \tag{□}$$

$$\sum_{i \in I} (y_{i,j} \cdot d_i) \leq b_j \qquad \forall j \in J \tag{□}$$

$$\sum_{i \in I} (z_{i,j} \cdot (d_i + p_i^s))) \leq b_j + \xi_j \qquad \forall j \in J \tag{6.33}$$

$$\lambda_1 \geq \sum_{j \in J^*} \sum_{i \in I} (z_{i,j}^s - y_{i,j})^2 \qquad \forall s \in \Delta \tag{6.34}$$

$$\lambda_1 \leq D \qquad \forall s \in \Delta \tag{6.35}$$

$$\lambda_2 \geq \sum_{j \in J} \xi_j \tag{6.36}$$

$$\sum_{j \in J^*} z_{i,j}^s = 1 \qquad \forall i \in I . s \in \Delta \tag{6.37}$$

$$z_{i,j}^s, e_j, y_{i,j} \in \{0, 1\} \qquad \forall i \in I . j \in J . s \in S \tag{6.38}$$

$$b_j, v \in \mathbb{N} \tag{6.39}$$

(□)	Same as the approximate model.
(6.31)	Minimise the amount of `spill_time`, or total slack.
(6.32)	Limit the amount of deterioration from some optimal nominal solution
(6.33)	For each scenario realisation, the total amount of time must not exceed bin b_j with some slack ξ_j for bin j
(6.34)	Cost function λ_1. Previously defined in Section 5.4.1
(6.35)	Total cost must be limited
(6.36)	Total `spill_time` is measured by the total amount of slack.
(6.37)	Each recovery decision must have a feasible and meaningful assignment of itineraries to vans.

It should be evident now that an LLRP gives us more flexibility in the defining recoverable robust programs. This is further evidenced by the fact that we now have control over the

conservatism of robustness. We could very well pick an optimal solution from the deterministic model as v^* and then set the price function we are comfortable with. Obviously we need to pick the price with care else we might require infeasible solutions. Similarly with D, we require some careful reasoning to pick it properly. This is essentially a limitation of our current definition of LLRP, as we would like to also simultaneously minimise both the price and other costs not related to slack.

6.6 Summary

We first formalises criteria for designing schedules. After doing so, we explore two different approaches to designing schedules.

The first approach involves dealing directly with robustness by utilising buffers. We first show the efficacy of a naive static buffering, where each itinerary is appended with some buffer. We then propose a more intelligent method of buffering, i.e. budgeted buffering, where buffers are part of the optimisation problem itself. We propose two models for this. The first one minimises the number of vans and the second model on the other hand is a stochastic delay-resistant model, which also minimises the expected amount of `spill_time`. In this process we explore linearising these complicated models as best as we can, and we propose linear formulation for both models.

Through this first approach of budgeted buffering, we contribute, to the best of our knowledge, the first *linear* stochastic delay-resistant formulation for the VSBPP with uncertain item weights. This has only been previously been modelled for the aperiodic timetabling problem [30].

The second approach involves exploiting the idea of recoverable robustness as proposed by Liebchen et al.[31]. We then propose a formulation of the recoverable robust model for the Royal Mail problem and identify a fundamental weakness with this methodology. Finally, we propose a generalised method of recoverable robustness. We prove that this is a generalisation of regular recoverable robustness and also provide a formulation for the Royal Mail problem.

Through this second approach of recoverable robustness, we contribute, to the best of our knowledge, a new generalisation of recoverable robustness we term *light recoverable robustness*. We show that both recoverable and light recoverable robustness are capable of modelling VSBPP problems under uncertainty, furthering the contributions of this understudied field.

Chapter 7

Evaluation

7.1 Disturbance Scoring Method

The disturbance scoring method (DIST), introduced in Section 4.5, is parametrised with coefficients penalties, a, b and c. These are picked with values which are heuristically deemed appropriate. We then use the final value as a tool for enabling the comparison of schedules, i.e. as a score, with a lower score meaning a better schedule. The score is a tool to study schedules after they have experienced uncertainty or have recovered from it.

7.1.1 Objective and Methodology

Thus far, we have utilised this score and hand picked penalties to justify certain conclusions. Here, we must evaluate the choice of these penalties and how it affects the aforementioned conclusions. As a reminder, the formula (DIST) is reiterated below.

$$\begin{aligned} \texttt{total\_vans\_req} &= \texttt{num\_vans\_req} + \frac{\texttt{spill\_time}}{T} \\ \texttt{disturbance} &\triangleq \frac{(a \cdot \texttt{num\_affected}) + (b \cdot \texttt{num\_spill}) + (c \cdot \texttt{total\_vans\_req})}{\text{Optimal Number of vans}} \\ &\text{s.t.} \quad a < b < c \quad \wedge \quad a + b + c = 100 \end{aligned} \tag{DIST}$$

The goal here is to evaluate how sensitive our conclusions are to the values of these penalties. We shall do this by analysing the individual parameters that make up the score: `num_affected`, `num_spill` and `total_vans_req`.

7.1.2 Longest Processing Time Recovery

The first conclusion to evaluate was that the LPT recovery method did not work for the deterministic model because of the significant variance in it's α value (recall Table 5.1).

The results obtained in Table 5.2 are further broken down into its individual parameters. The same table with these parameters instead is presented in Table 7.2. The symbol α here is the same as in Table 5.1. So for example, the α of `num_spill` is the is the mean decrease in `num_spill` from recovery being applied to a perturbed solution. As a

reminder, `num_affected` does not change during recovery, so we omit it from the following table (recall Section 4.5).

			Model		
	Parameter	Statistic	Deterministic	Makespan-1	Makespan-2
α	`num_spill`	μ	0.016	0.002	0.005
		σ	0.013	0.002	0.005
	`total_vans_req`	μ	0.0	0.0	0.0
		σ	0.001	0.002	0.002

Table 7.1: Disturbance score comparisons for all the models across all DOs

Here we observe results that confirm our conclusion about the LPT heuristic are independent of our choice of penalties. Once again, we assume each penalty is normally distributed and their distributions are plotted in Figure 7.1.

Firstly, we observe that `total_vans_req` is distributed similarly across all models as is expected, as this parameter is largely dependent on type 3 perturbations, which is uniformly generated (Section 4.4). Note that its mean is a small value and not actually zero valued, and this is observed due to us rounding measures to three decimal places. Its relatively smaller dependence on `spill_time` causes a very slight change in the deterministic models distribution.

Next, we note that `num_spill` is purely a reflection of of how a schedule behaves under uncertainty (recall the perturbation generation methodology detailed in Section 4.4). The distributions of `num_spill` parallels Figure 5.2a, the distribution on which we based our original conclusions. Based on `num_spill` alone we observe the large deviation of the deterministic model's α as compared to the makespan models.

Thus we still *cannot* guarantee the efficacy of LPT applied to the deterministic model with confidence, *unless* the penalties are picked such that `num_spill` is effectively ignored.

7.1.3 Static Buffering

The analysis of static buffering was done on the value of β, which is the disturbance score of a perturbed schedule. Once again, we need to evaluate how the choice of our penalties affect the analysis that was originally conducted. The table below depicts all the parameters this time round, including `num_affected`.

			Buffer Amount		
	Parameter	Statistic	1	$\lceil \text{shock}_{95\%} \cdot T \rceil$	$\lceil \text{shock}_{95\%} \cdot d_i \rceil$
β	`num_affected`	μ	0.028	0	0.047
		σ	0.015	0	0.016
	`num_spill`	μ	0.203	0.243	0.159
		σ	0.049	0.043	0.055
	`total_vans_req`	μ	0.056	0.057	0.054
		σ	0.002	0.002	0.002

Table 7.2: β comparisons for all the models across all DOs

From the underlying parameter values, we conclude that if the penalties are such that `num_affected` was penalised significantly more than `num_spill`, only then would our findings would be altered. This is to say if an optimiser greatly values itineraries not being disturbed after realising uncertainty, our understanding of static buffering changes slightly.

For example, let us pick the following penalties,

$$a = 70 \qquad b = 10 \qquad c = 20$$

We then observe that the $buf_i = \lceil \text{shock}_{95\%} \cdot T \rceil$ performs the best in terms of β. Thus our understanding of static buffering is a slightly sensitive to the penalties.

7.2 Parameter Evaluation

The process of data cleansing fundamentally involves picking a handful of parameters. These choices are reasonably grounded, however, it is still in need of a rigorous evaluation. The relevant parameters and its associated values picked have been previously described in Table 3.3.

7.2.1 Objective and Methodology

Ultimately, we would like to gauge how sensitive the Royal Mail problem is to the associated data cleansing parameters.

To evaluate this we opt to observe how our conclusion on the efficacy of the LPT recovery method is affected when parameters are tweaked. We then generalise our findings. In essence we would like to observe if some change in a certain parameter drastically changes our original findings. We do this as per the method previously employed in determining the efficacy of LPT. We shall look at the distribution of disturbance score measures α, β and γ (recall Table 5.1).

Before doing so we must deal with the fact that while certain parameters might introduce sensitivity, certain other parameters likely do not. We must understand the nature of the raw data provided to us to make such assessments. To this end, the distribution of the length of itineraries is plotted in Figure 7.2. To understand and justify which certain parameters would introduce sensitivity, we shall look to this distribution, to augment our arguments. This is done because the parameters we analyse are largely to do with an itinerary's duration.

7.2.2 Observations

This distribution plotted in Figure 7.2 is one across all DOs. For the rest of this section, let H denote the random variable representing the number of hours worked. Another point of note is that the graph is limited to 17 hours on the x-axis as a negligible number of itineraries occur for longer.

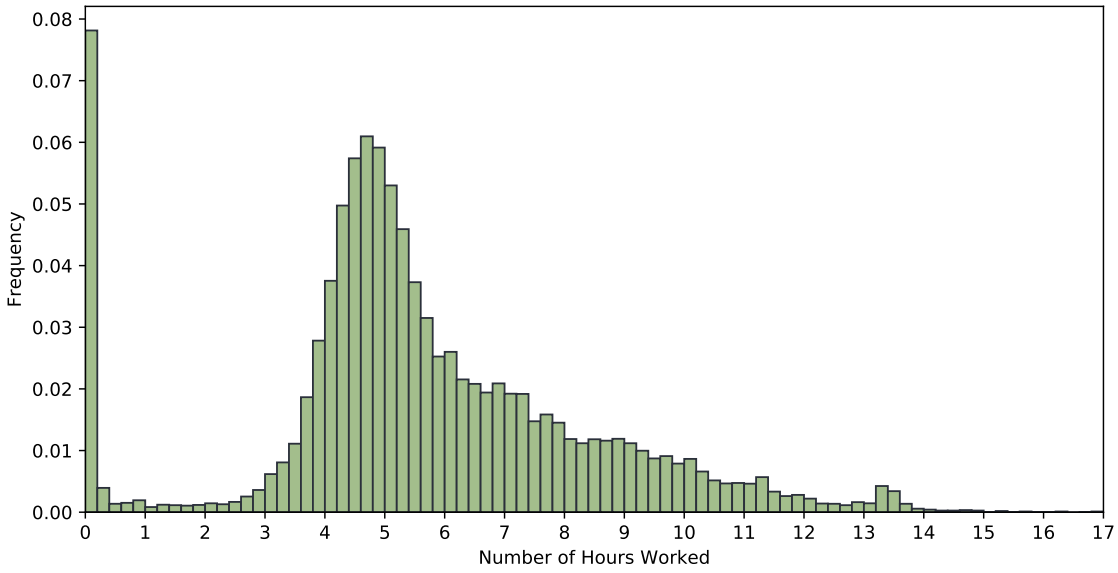


Figure 7.2: The Distribution of itinerary durations. We observe a non-negligible amount of anomalous entries with close to 0 hours worked.

First, we justify why certain parameters do not affect our conclusion. Most obviously, the value of ϵ does not matter if picked with intent. Note that from the data $p(H = 0) \approx 0.078$ and $p(H < 1 \mid H \neq 0) \approx 0.01$. Thus if ϵ is sufficiently small, it does not affect our conclusions. Also, `late_hour` anomalies are not processed so they do not affect our conclusions.

Apart from these, we cannot easily assert that the rest of the parameters do not introduce sensitivity into the Royal Mail problem.

The values of `short_delta`, `long_delta` and `abuse_delta` are set with a certain intent. The parameters are also ordered, where `short_delta` $<$ `long_delta` $<$ `abuse_delta`. Without the ordering the parameters lack semantics. The probabilities of incurring the associated anomalies as it stands are as follows.

$$
\begin{aligned}
p(\texttt{short\_work}) &= p(H \leq \texttt{short\_delta}) = 0.212 \\
p(\texttt{short\_work} \mid H \neq 0) &= p(H \leq \texttt{short\_delta} \mid H \neq 0) = 0.134 \\
p(\texttt{long\_work}) &= p(\texttt{long\_delta} \leq H \leq \texttt{abuse\_delta}) = 0.071 \\
p(\texttt{abuse\_work}) &= p(H > \texttt{abuse\_delta}) = 0.036
\end{aligned}
$$

The mode of our data lies at $p(H = 0)$. However, these are clearly anomalies. Ignoring this, the next mode of our data lies around the 4.5 hour mark, and a large number of itineraries are of this length. If the data suggests anything, it is that the `short_delta` parameter might have been picked at a larger than true value.

As a reminder, `short_work` itineraries get some amount of time removed from them as part of the cleaning process. This is parametrised by `short_work_overhead`.

Since the value of `short_work_overhead` is initially picked as 0, this means that changing `short_delta` does not affect our conclusions. If `short_work_overhead` were to instead set to 1, we need to evaluate how this introduces sensitivity. This is considered an upper limit

to `short_work_overhead`, as we assume that in the worst case not more than on hour of overhead is experienced.

Next it is also necessary to evaluate how a reduction in the value of `short_delta` to 3 with the overhead parameter also affects the conclusions, given that it has been originally picked with a value that is probably larger than its true value. We can say with confidence that `short_delta` ≤ 2 will have little to no impact on our results given that $P(H \leq 2 \mid H \neq 0) = 0.016$ and a reduction in an hour of work would impact the data very minutely. The parameter also has its upper limits at 4 hour, as any value beyond that makes little semantic sense within the context of the distribution of itinerary times.

Work that is classified as `long_work` lies between parameters `long_delta` to `abuse_delta`. The lower of end of the probability (i.e. `long_delta` = 9 hours) introduces volatility when it comes to the probability p(`long_work`). This should be expected given the nature of the distribution in Figure 7.2. On the other hand, the upper bound, does not make a considerable difference. Furthermore, this is coupled with the fact that itineraries classified as `long_work` need to be broken in to smaller instances parametrised by `shard`. This parameter was chosen arbitrarily. We should thus analyse the sensitivity of decreasing `long_delta`. With this we also vary the value of `shard`.

Similar to this is `abuse_work`, and each itinerary classified as this anomaly is truncated down to the length set by parameter `regular_hours`. The values of `regular_hours` and `long_delta` are semantically correlated, thus possessing similar values. These values were initially picked rather liberally. In reality, the (robust) mean of work times excluding $H = 0$ is approximately 5.8 hours. Furthermore, in reality, this value is most likely much smaller. Thus we need to evaluate a reduction in `regular_hours`, to observe if the break in correlation between the two aforementioned parameters introduces sensitivity.

The parameter `work_hours` describes the duration of the time horizon. Note that the values of $start_{abs}$ and end_{abs} themselves do not matter. This ultimately should not affect our solutions. Furthermore, its value has to be congruent with certain other parameters, for instance, something which is `long_work` obviously cannot be longer than `work_hours`. To test our hypothesis we also evaluate a case where this value is decreased.

For the parameter T, we will roughly obtain the same problem problem with different values of it that make sense. Obviously, prohibitively small values will affect our conclusions. We shall also evaluate what a significant reduction of T results in, and determine its approximate lower bound. It should be obvious that an increase in its value will not affect our conclusions.

All the different parameter combinations are detailed in Table 7.3. The changes in parameters are done with respect to the current values picked, as previously done so in Table 3.3. The values in the former table are thus represented as "+1" to represent an increase by one from the original parameter value. Furthermore, each change has an associated index. Here we are limited in terms of the practicality of cross testing every parameter change with another. Thus we only evaluate each change with respect to the original parameters.

Parameter	Current Value	Changes Index								
		1	2	3	4	5	6	7	8	9
`work_hours`	10	0	0	0	0	0	-2	0	0	0
`short_work_overhead`	0	+1	+1	0	0	0	0	0	0	0
`long_delta`	9	0	0	-1	-2	0	-3	0	0	0
`abuse_delta`	11	0	0	0	0	0	0	0	0	0
`short_delta`	4	0	-1	0	0	0	-1	0	0	0
`regular_hours`	9	0	0	0	0	-2	-3	0	0	0
`shard`	4	0	0	+2	+1	0	0	0	0	0
T	50	0	0	0	0	0	0	-20	-30	-40

Table 7.3: Table of parameter changes with respect to the current set of parameter values

First, let us consider the distribution of α values of both the makespan and deterministic models with the above parameters tweaked, as seen in Figure 7.3. Each bar represents a mean of the α value for the deterministic and makespan models for a specific set of changes (as per the "Changes Index" in the table above). We also plot the unchanged values (as per "Current Values") for comparisons sake. The error bars represent one standard deviation.

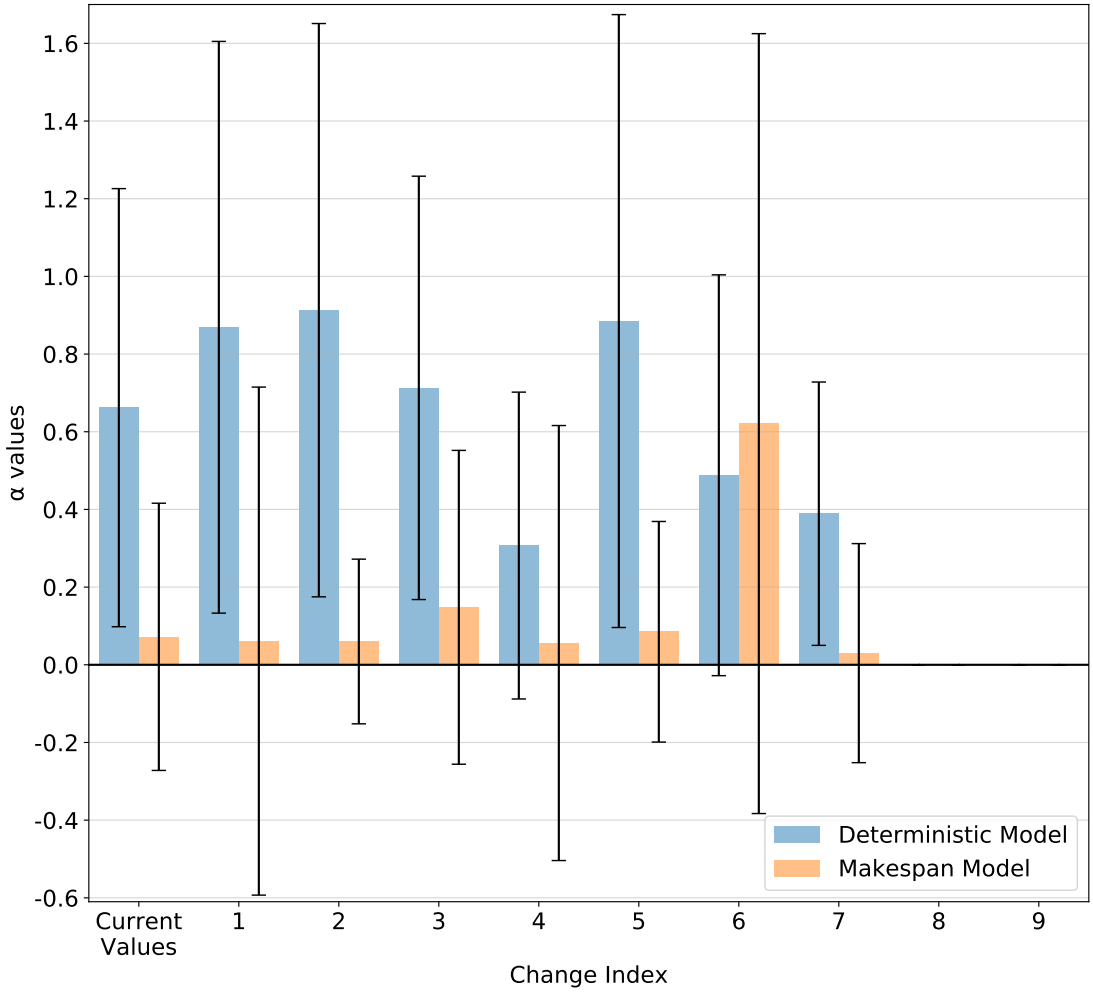


Figure 7.3: Depicted is the α values of all the parameter change indices. Each bar here represents the mean value of α, and each error bar represents one standard deviation.

In the figure above, we look for behaviour that is "comparable" to the results from the current values. This is to mean that the measures of mean and deviaiton are proportional and similar, and could be easily used to derive the same set of conclusions about LPT's inefficacy.

First, we observe change indices 7, 8, and 9. These are modifications to the value of the discretisation size T. We observe a largely comparable behaviour between index 7 and the unchanged observation. Indices 8 and 9 however observe an complete reduction in α for both the makespan and deterministic model. The fact there is no reduction in the disturbance score at all suggests that the granularity of discretisation is too coarse, i.e. T is set too low, to observe any recovery benefits. Thus we conclude that our observations are largely insensitive to the value of T until it is roughly around 20 or less, after which it is highly sensitive.

Next we look at change indices 1 and 2. These are to do with `short_work` anomalies. Here we notice some sensitivity, specifically in change index 1. There is a spike in the makespan model's deviation which is not proportional to our current values' measures. This is however not the case in index 2. This implies there exists some sensitivity in our current definition of `short_delta`, which is understandable given the high probability value of `short_work` anomalies. From this we gather that if we are to pick a `short_delta` of 4, any `short_work_overhead` introduces some sensitivity, however if `short_delta` were lesser, this would not be the case. All this being said, although the changes in parameters slightly warp our observations, they are largely similar, and can still be used to deduce the same conclusions.

Indices 3 and 4 are to do with `long_work` anomalies. They appear to show little to no sensitivity. Index 3's makespan gets a mild boost in terms of mean α but also in its deviation. Still neither indices have noticeably disproportional performances with respect to the current values. This shows that the lower bound of `long_delta` does not introduce much sensitivity, and `shard` introduces minor sensitivity, to observed values.

Index 5 is to do with reducing the amount of hours `abuse_work` itineraries are truncated down to. In this we observe little to no sensitivity and distributions are proportional.

Index 6 explores how a reduction in multiple related parameters affect sensitivity. However we leave out a reduction in `abuse_delta`, to observe any possible sensitivity. We observe index 6 is extremely sensitive. Here we observe a role reversal where the makespan model has a much higher mean. Further tests were carried out with a proportional decrease in `abuse_delta` as well.

When this is also decreased, our observations are slightly more normal. Still we observe that the makespan model performs better than the current set of values, and disproportionately so. We cannot conclude much about the reduction of multiple parameters at once due to the complexity of testing this. All we can confidently say is that the decrease in a large number of parameters warps our observations. They most likely introduce some volatility into our conclusions, but from our limited set of observations, it seems as though the LPT recovery method is still ineffective for the deterministic model.

7.3 Perturbation Generation

Recall the perturbation generation algorithm in Section 4.4. We have previously systemised the process of generating uncertainty, and noted that in this process, certain parameters are involved.

7.3.1 Objective and Methodology

The perturbation generation method has some parameters which are not informed by apriori statistical information. Here, we need to evaluate how fluctuations in these parameters introduce sensitivity into the Royal Mail problem once again. We shall do this in a similar manner as the parameter evaluation, by looking at our conclusions on the efficacy of the LPT recovery method.

The parameters of concern are: `perturbation_chance`, `type3_chance` and `type1_chance`. As a reminder, they are defined in Table 4.8, and the values picked are in Table 4.9.

7.3.2 Observations

We hypothesise that the choice of parameters largely do not matter as long as they do not dramatically minimise the amount of disturbance a schedule experiences. When this is the case, the effects of recovery are obviously much less noticeable. This because the perturbed schedule is no longer as perturbed, i.e. there is less to recover from.

In this vein, we can guarantee that fluctuations in the value of `type3_chance` do not matter since they does not directly affect schedules. This is unless of course, the disturbance score penalties significantly penalise the value of `total_vans_req`. However, doing this only means that the effects of recovery are numerically less visible. This would mean that we would still observe the same conclusions.

If either `type1_chance` or `perturbation_chance` were to be minimised, the amount of perturbations a schedule experiences reduces. We seek a lower bound to both parameters, noting that the reduction in `perturbation_chance` effectively reduces `type1_chance`. Thus we seek to evaluate the lower bound of `perturbation_chance`, where sensitivity is introduced into conclusions. Plotted below in Figure 7.4 is the distribution of α for both the deterministic and makespan model as the value of `perturbation_chance` is decreased.

Notice that for a value of `perturbation_chance` of 0.2, we still observe similar behaviour. Once it is of the value 0.1 and below, we notice a warping in our observations of α. In fact, we might conclude that the LPT is effective for the deterministic model upon first glance. However, looking further at the β and γ values, as is seen in Figure 7.5, we see a different picture. We notice that as `perturbation_chance` tends to 0, the disturbance score of the perturbed and recovered schedules also tend to 0, expectedly so. However, the makespan model retains its superiority in terms of disturbance score of its perturbed and recovered schedules. At the point where `perturbation_chance` is 0.05, the change of disturbance score is so minimal that the makespan model's α appears to be worse of. Thus we say that this is `perturbation_chance`'s lower bound, as observations around this area are hard to reason about, due to the sporadic nature of perturbations. The effective value for `type1_chance` at the lower bound is ≈ 0.03, when with the original values of parameters it was ≈ 0.31.

We can say with confidence that the amount of expansion in an itinerary's time (recall the *shift* function in Algorithm 9) largely does not matter. If the size of expansion tends towards 0, we are left with the same situation as above, where there is lesser to recover from. We seek a lower bound to the value of deviation of the distribution we sample from in the shift function. The exact same process is repeated, and we find that our α observations start to warp when the deviation of the shift function is ≈ 0.5.

When an itinerary's expansion in size is arbitrarily large, the only thing that changes is the amount of `spill_time`. The algorithm still performs in a similar manner, and would be scored proportionally.

Lastly, it would also be in our interests to evaluate how an increase in `perturbation_chance` and `type1_chance` affects recovery. We hypothesise that this does not in fact affect our findings. We test this with an extreme case, where every van experiences a perturbation and it is always an expansion in time (type 1 perturbation). In Figure 7.6, we observe values of α, β and γ that confirm our hypothesis. LPT applied to the makespan problem still largely outperforms being applied to the deterministic model for all the same reasons.

7.3.3 Alternative Method

There are perturbations in which bins become unavailable or vans breakdown (type 3 perturbations). These perturbations are effectively ignored in the process of recovery. We count the number of these type 3 perturbations, and add it to the disturbance score. While this is a viable strategy, there exists an alternative method of handling this.

When a van experiences a type 3 perturbation, its itineraries could be made free and reassigned randomly. The disturbance score could avoid counting these perturbations with `num_van_req`, and solely `spill_time` could be measures. This is a more intuitive method of dealing with type 3 perturbations, and feels more realistic. Furthermore, the score can truly encode the effects of type 3 perturbations.

The drawbacks of this method are as follows. Firstly, in this spirit, we are no longer able to penalise the number of new vans used separately. This approach takes the position that when a bin or a van becomes unavailable, the cost associated with it can be completely optimised away, if it is possible. However, this is not the case in live systems. If a van were to break down in the middle of its operations, the associated costs of getting the remaining mail delivered is not something that can be completely minimised by recovery methods. Thus, there are trade-offs to both approaches of handling such perturbations.

7.4 Integer Programming Recovery

Here we evaluate the performance of linear recovery programs proposed in Section 5.4. Our baseline of comparison is the performance of LPT recovery with the makespan implementation of the problem (as done in Section 5.3). LPT gives us a "ground truth" frame of reference when it comes to recovery with its strong guarantees.

Recalling Section 5.4, the integer recovery programs we defined came with cost coefficients ν, κ and ρ. We reason about the cost coefficients and compare different combinations and values of these. We showed that the most meaningful coefficients result in the most significant recovery properties in a relative to the other integer recovery programs.

However, not all method exceedingly outperform LPT. In fact, it is only model #5 that outperforms LPT applied to the makespan problem, based on the measures we used to originally assess LPT's efficacy. Model #5 outperforms in terms of both α and γ. This is to say that it is able to recover much better on average and its recovered solution is also much better on average, respectively. We plot the distributions of α of all 6 integer recovery programs and the LPT recovery methods applied to the makespan problem in Figure 7.7.

However, it is harder to solely judge the usefulness integer recovery program approaches purely on the measures of α, β and γ. These recovery methods provide much more control to us over the process of recovery than say LPT. Through these more sophisticated methods, we are able to control levels of infeasibility (through λ_2) and even assign new vans to recover (for complex recovery programs).

Thus, although we are only able to outperform LPT significantly in one specific instance, integer recovery programs provide a variety of other benefits that still aid us greatly. This being said, the performance of some other integer recovery programs are still highly competitive (#1,#2,#4) and viable methods.

7.5 Schedule Design

The dissertation proposes multiple methods of scheduling. Here we try to qualitatively evaluate the benefits of all such approaches with respect to the original deterministic model as proposed by Page [36].

First we take a look at the unified and approximate models. The unified model is clearly a superset of the deterministic model. An optimal solution for the unified model is an optimal feasible solution for the deterministic model. The unified model however is a much larger model, and though it is linear, the current formulation is hard to compute. The model is however much more general and also models the relationship with vans.

The approximate model, however, is not a subset of the unified model. The approximation with the bins results in the lack of equivalency. However, the approximate model is much more tractable. An obvious drawback is that it truly does not model the gate constraint. The approximate model preserves the relationship between vans and itineraries, which is a much powerful relationship. With this, the approximate model incurs a cost of being harder to solve compared relative to the deterministic model.

The next models to consider are the two formulations of budgeted buffering models. We note that both formulations, as opposed to the deterministic model, hedge for uncertainty. The first formulation does so in a naive way as compared to the second one. The second formulation is ultimately the most expressive, and encodes truly what is required of the Royal Mail problem for perturbed solutions (i.e. minimising β). We have also managed to linearise both methods with some care. Both formulations are still hard to solve in practice due to their size.

The last modelling to consider are the two methods of recoverable and light recoverable robustness. Unlike the buffering methods, these methods also model the process of recovery, thus, we are able to express much more than aforementioned buffering methods. With these models we are able to encode the need for a perturbed solution that is hedged against uncertainty, *and* the method recovery for the perturbed schedule. We define exactly the recovery method to use in the model itself.

With the new light recoverable robustness method, are able to forgo the constraints of feasibility imposed by the recoverable robustness model. This means that we can encode a "best effort" method of recovery. Furthermore, we are able to *control* the conservatism of the robust solution. These recoverable robust models are once again hard to solve, but are still linear models. They effectively encodes truly what is required of the Royal Mail problem for perturbed *and* recovered solutions (i.e. minimising both β and γ).

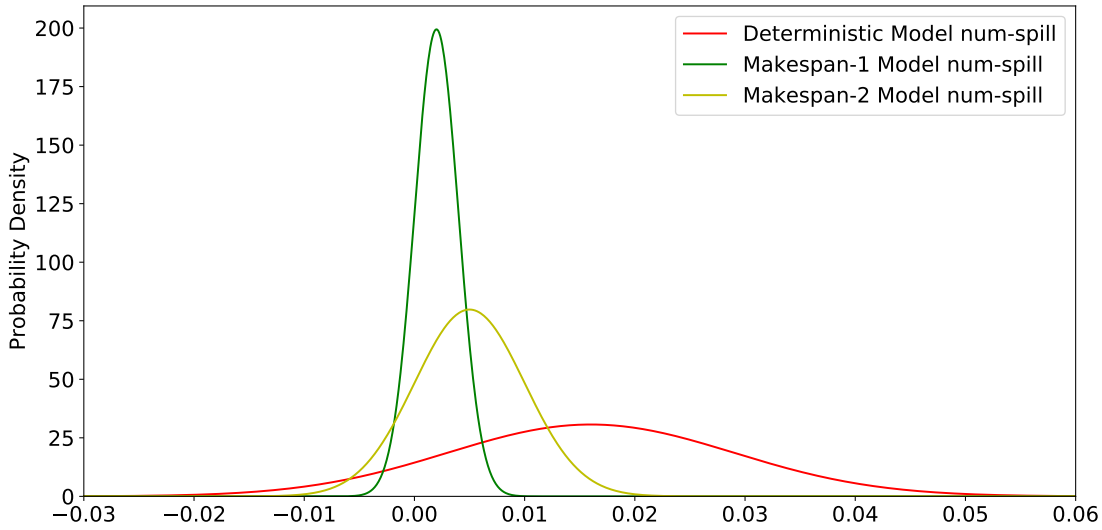
7.6 Criticisms Of Analysis and Evaluation Techniques

Here we discuss some potential criticisms of the analyses or evaluation methodologies in this dissertation.

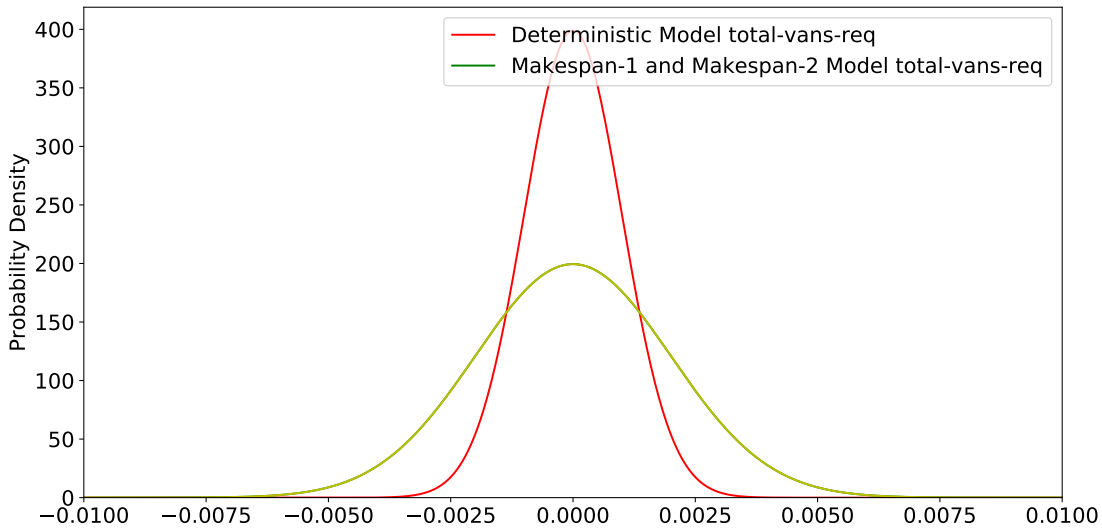
Firstly, the process of parameter evaluation has some clear limitations. Since we are not able to truly test out every combination of tunings of parameters, we measure them one at a time to observe for sensitivity. This is largely due to the limited duration of the dissertation and a cross checking of parameters would take prohibitively long.

We consistently look at the parameters α, β and γ as a basis of comparison (recall Table 5.1). There are certainly more sophisticated methods, but for the extent of this dissertation, we believed that this would suffice.

Throughout the evaluation and parts of the report, we assume certain random parameters are normally distributed, for example, the value of α. This reasonable assumption allows us to distill the samples we obtain of statistical measures such as α in order to simplify the process of making our conclusions. However, the true nature of the samples might not be exactly normally distributed. We hypothesise that the amount of negative measures, e.g. a negative α, is much lesser than a normal distribution would suggest, as it does in Figure 7.4 or Figure 5.2a. This hypothesis of ours is found to be largely true with some basic visualisations of the true sample distribution. One such example of this is plotted in Figure 7.8. Here we see both the distribution's mean in blue and the true sample distribution in green.

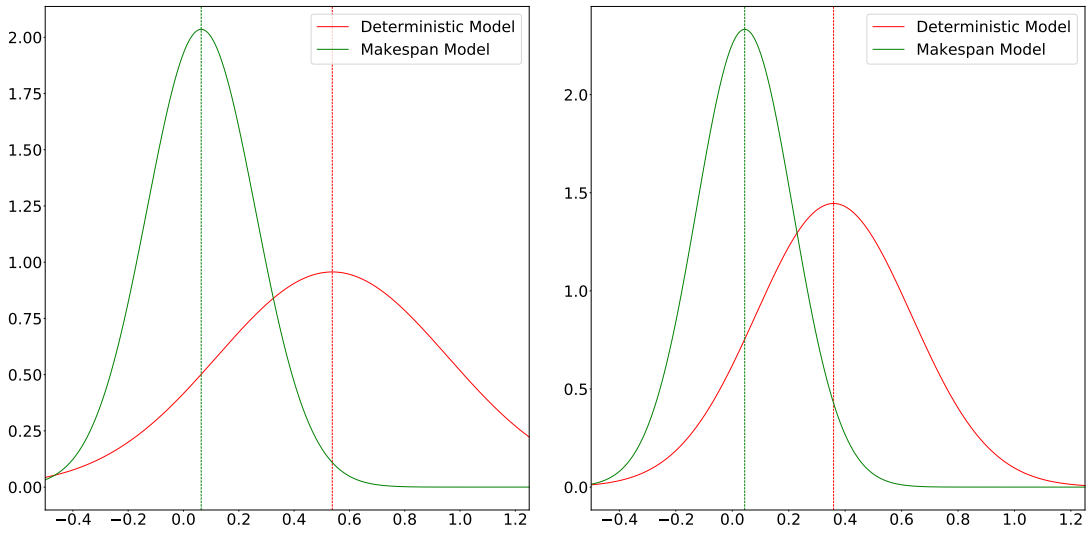


(a) The distribution of disturbance score parameter `num_spill`



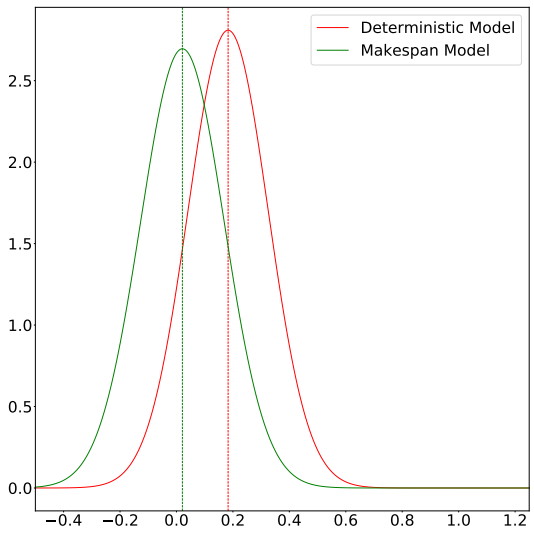
(b) The distribution of disturbance score parameter `total_vans_req`

Figure 7.1: Plotted above are the distributions of the average disturbance score decrease in parameters `num_spill` and `total_vans_req` after recovery.

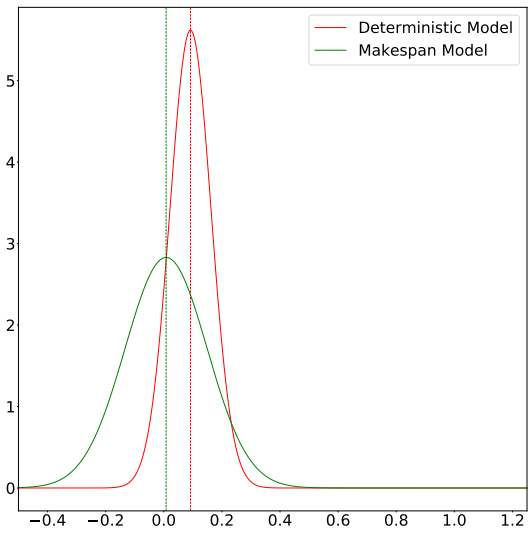


(a) `perturbation_chance` $= 0.3$

(b) `perturbation_chance` $= 0.2$

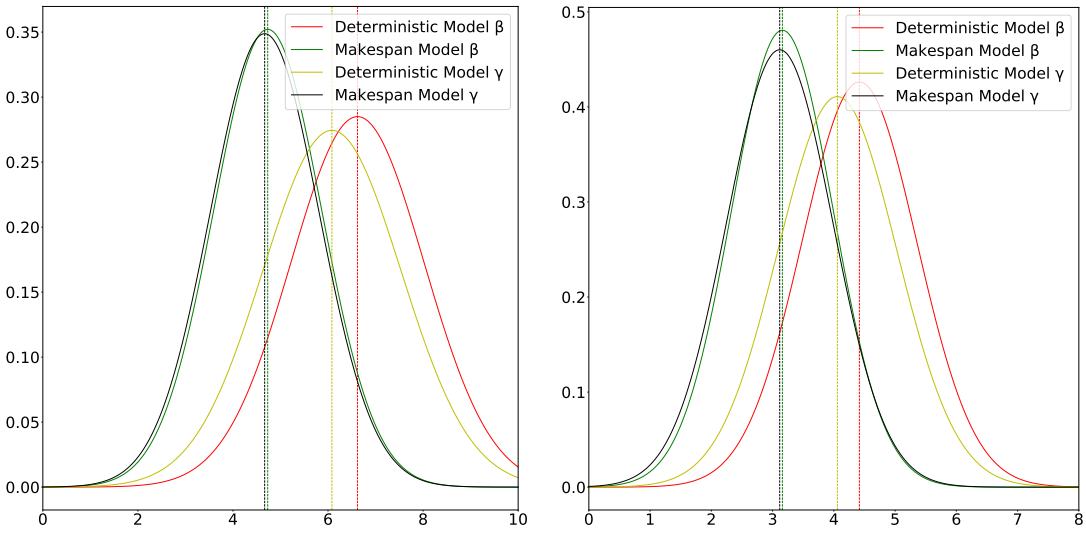


(c) `perturbation_chance` $= 0.1$

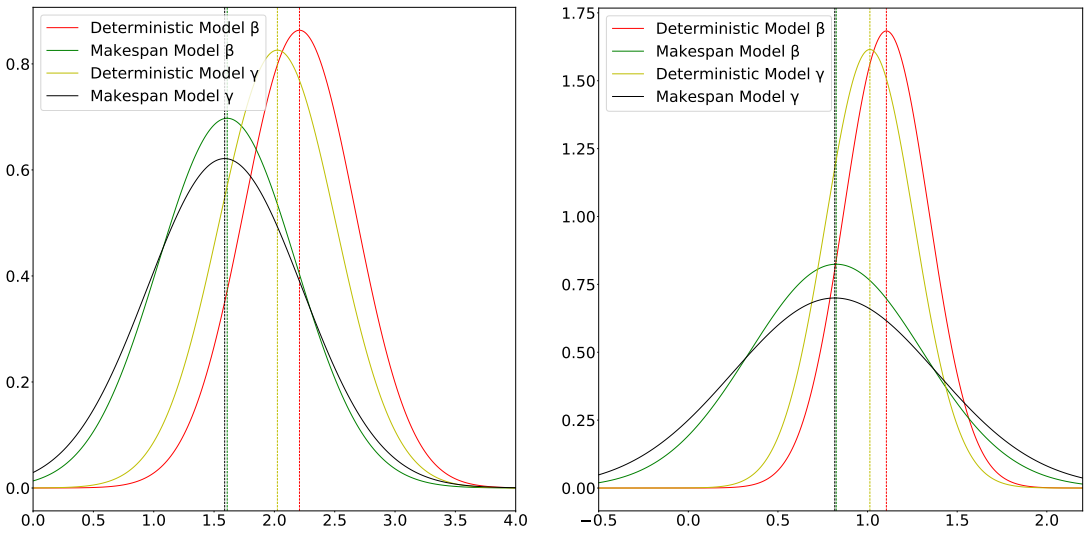


(d) `perturbation_chance` $= 0.05$

Figure 7.4: The distribution of α for decreasing values of`perturbation_chance`. Here we see that decreasing the value of `perturbation_chance` does not drastically affect our observations until it is of value 0.1

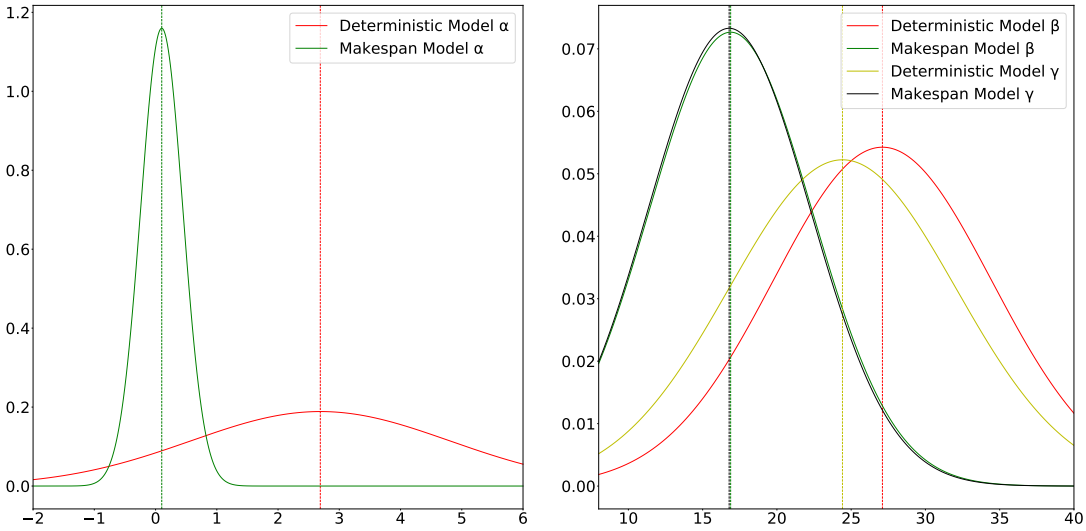


(a) `perturbation_chance` $= 0.3$ **(b)** `perturbation_chance` $= 0.2$



(c) `perturbation_chance` $= 0.1$ **(d)** `perturbation_chance` $= 0.05$

Figure 7.5: The distribution of β and γ for decreasing values of `perturbation_chance`. Here we also see that decreasing the value of `perturbation_chance` does not affect our observations until it is of value 0.1



(a) α Value of the Deterministic and Makespan Models

(b) β and γ Values of the Deterministic and Makespan Models

Figure 7.6: Depicted above is the behaviour of LPT recovery for both the deterministic and makespan models when disruptive type 1 perturbations are experienced 100% of the time. This means that every van experiences one perturbation, which is a worst case scenario. We notice behaviour in these distributions that are consistent with past observations.

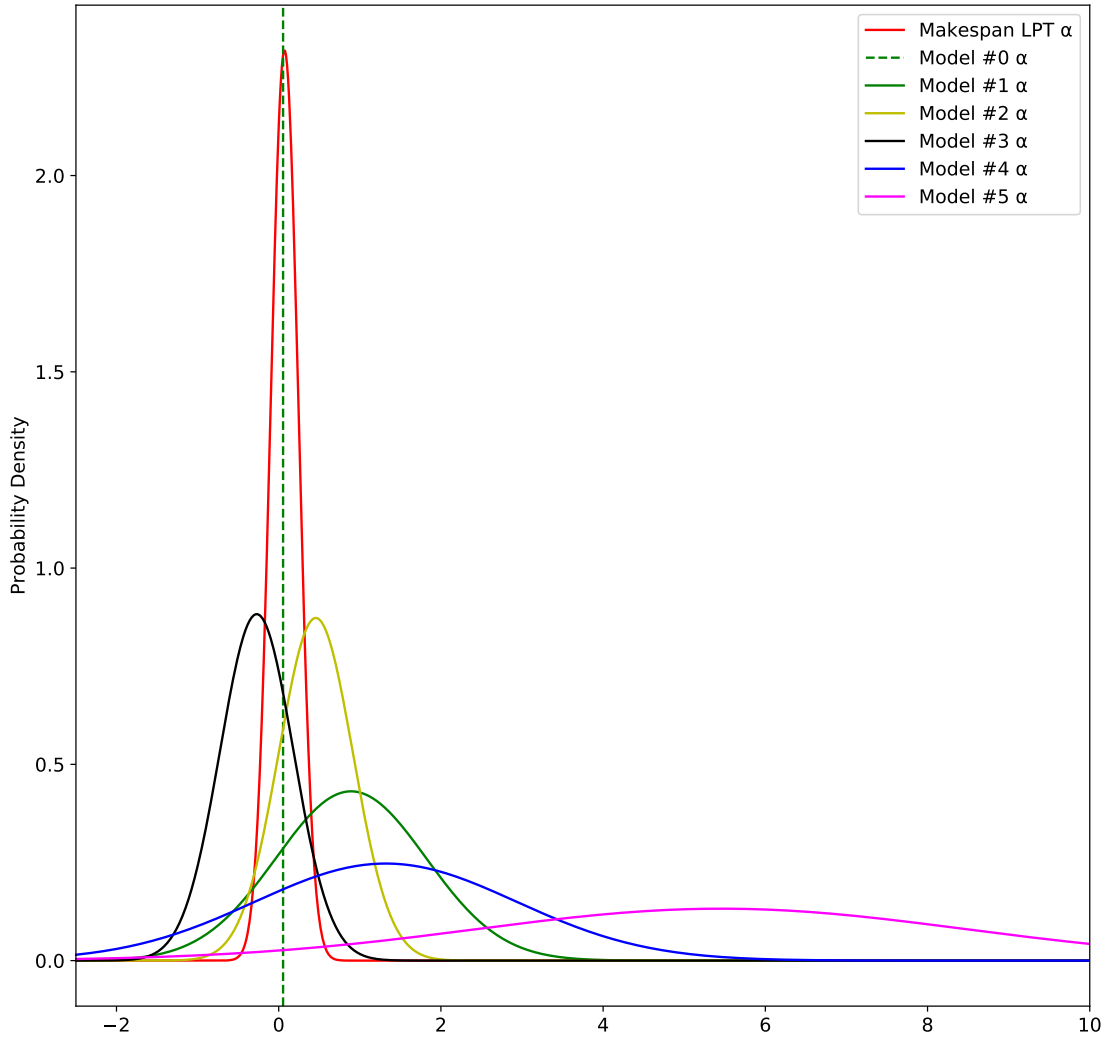
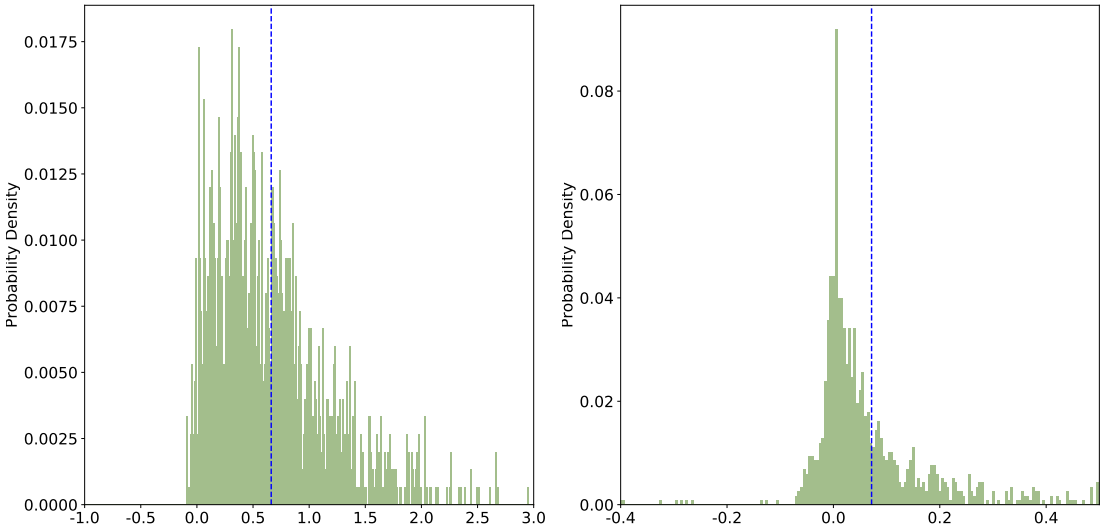


Figure 7.7: α values for all integer recovery program's as well as LPT applied to the makespan problem



(a) Deterministic Model **(b)** Makespan Model

Figure 7.8: Depicted above is the true distribution of α values for the Makespan and Deterministic models. Here we observe that although a normal distribution is justifiable, it possibly includes too many negative values that do not truly exist in the sample data. Here the dotted blue line represents the mean value.

Chapter 8

Conclusions and Future Work

With the Royal Mail problem as a framework, we have quite successfully conducted a practical and theoretical exploration of the field of optimisation under uncertainty, and significantly expanded upon the original modelling. Stemming from a critical analysis of the existing modelling, we prescribe both rescheduling strategies and holistic modelling methodologies (stochastic delay-resistance, recoverable robust and LRRP). These practical prescriptions have ultimately come from the theoretical studies and contributions we made to the field of uncertainty under optimisation.

8.1 Contributions

The four primary contributions of this dissertation lies in the expanding of the original modelling of the Royal Mail problem and contributions made to studying the Variable Size Bin Packing Problem (VSBPP) under uncertainty, a field that lacks extensive study.

First we identify the limitations in the original modelling of the Royal Mail problem, namely its lack of expressivity; it lacks information regarding the relationship between vans and itineraries. We generalise this modelling by including the relationship with the vans, but critically observe that this sprawling generalisation contains a large number of constraints, thus being computationally challenging. We also then provide an alternate approximate reformulation, which is more tractable than the generalisation. This reformulation includes the relationship with vans in its modelling, but disposes of the relationship with time, thus resulting in said approximation. We however show that the approximation is a largely justifiable one. The disposal of the relationship with times is a limitation we have to incur, as it allows us much more breadth and depth in the subsequent research.

After identifying a relationship between the approximate reformulation and the VSBPP, we confirm that the Longest Processing Time rescheduling method *does not* provide the set of strong guarantees it provides to the makespan minimisation problem, through experimental results. Makespan minimisations are a separate class of scheduling problems but exhibit some similarities, which led to a suspicion of an equivalency; we have shown this to be untrue in the context of our problem. Through these same experimental results, we show the efficacy of modelling the problem of rescheduling as an integer program itself. We further demonstrate the control and flexibility that the latter approach lends to the optimiser. Through the latter approach, we have contributed viable recovery (or rescheduling) methods for the VSBPP problem under uncertainty.

We also propose, to the best of our knowledge, the first stochastic delay-resistant formulation for the VSBPP. The process of linearising this problem is a tricky one given the bin packing paradigm, and its tendency to lend itself to non-linearities when extended further. This methodology has only previously been applied to a somewhat simpler and very different class of problems known as the aperiodic timetabling problem [30].

Lastly, we propose, to the best of our knowledge, a new generalised method of the Recoverable Robustness that we term *Light Recoverable Robustness*. The Generalised Bin Packing Problem (GBPP), which is a superset of the VSBPP, is prone to mild infeasibility which could break some constraints in certain scenarios when it realises uncertainty. Bin Packing Problems in all their forms are ubiquitous. Any live system that uses both a bin packing approach to optimise their operations and Recoverable Robustness to hedge against uncertainty will still need to to reschedule as best as it can, when some constraints are broken. Systems should not come to a grinding halt, however the regular Recoverable Robustness's approach, will ultimately fail in such scenarios. *Light Recoverable Robustness* takes from the ideas of *light robustness* and vies for a best-effort approach in scenarios where all the constraints cannot be met. It effectively minimises on the number of constraints broken while aiming for a good solution. Through this, we effectively propose a novel and more flexible method for VSBPP optimisation problems under uncertainty.

8.2 Future Work

We separate the future work into two categories: practical and theoretical. In the practical, we talk about the extensions that can be made directly to the Royal Mail problem. As for the theoretical, we identify the next necessary steps to further the study of the field of optimisation under uncertainty.

8.2.1 Perturbation Generation and Disturbance Scoring

As mentioned in Section 7.3.3, a conscious choice was made in the process of generating perturbations for a schedule. In the aforementioned section, we presented two methods of handling perturbations where a van is no longer available. These methods, though simple, are sufficient for the nature of our experimentation. However, the two methods can be combined to model more accurately the Royal Mail problem. To do so the following needs to be done.

The disturbance score needs be modified to measure for two different types of `spill_time`: one from the expansion of itinerary times and one from instances where a van is no longer available. During the process of uncertainty realisation, the jobs from a van that has gone down needs to be reassigned temporarily, before recovery can take place.

The process of this reassignment has to be done so thoughtfully as well. Perhaps the Longest Processing Time heuristic might work in this scenario. The process of recovery should also have new cost functions defined for the reassignment of jobs from a van breakdown. The nature of the real life costs associated with such a situation also need to be better understood. It could potentially be argued that the work that needs reassignment should all be redirected to one van, rather than be spread out. This is at ends with our current ideas of recovery which aims to minimise `spill_time`.

8.2.2 Tractability Problems and Further Experimental Evaluation

The unified model, as previously mentioned, has a large number of constraints which makes it less tractable than, say the deterministic or approximate model. It is thus, much less appealing to build holistic scheduling methods on top of it (e.g. Recoverable Robustness formulations). One point of future work can be a study on how to speed the computations up. The problem suffers from some issues with symmetry, and as such the work to aid tractability should be directed towards effective lower/upper bound generation and introducing symmetry breaking cutting planes.

An example of this symmetry which is prevalent is as follows. Given one bins and two items to pack, if we have a feasible solution, we can reverse the order of the two packed items and obtain the same optimal feasible solution. This ordering matters greatly to our problem due to the ordered nature of the timeslots.

Furthermore, Liebchen and Stiller [30] have proposed methods to solve recoverable robustness program efficiently when the nature of the uncertainty realisations are restricted. Another avenue for research is in exploring whether this could be successfully used to boost the computation speeds of recoverable robust programs.

Due to time constraints, the models in the very last chapter could not successfully evaluated with the framework we set up (perturbation generation, disturbance scoring, etc.). The models presented are linear in their formulation, but are much harder to solver relative to the original modelling work or even the approximate model, due to the size of some of the models (number of constraints and decision variables).

It would be of use to computationally and experimentally verify that the more elaborate formulations of the Royal Mail Problem that hedge against uncertainty do exactly as they claim. To do this, it would be necessary to have methods of speeding up computation as mentioned previously above.

8.2.3 Multi-Stage Optimisation

We have limited the scope of our dissertation thus far to "2-stage" problems. This is to mean we only concern ourselves with one realisation of uncertainty in the entire process of optimisation under uncertainty. This already provides sufficient breadth in terms of research, but in real live systems it is more than likely that uncertainty is to be realised more than once. The field of multistage optimisation is not quite as simple as repeated applications of recovery. Thus, this could be a possible avenue for future research, but its feasibility as a research topic must also be evaluated. The difficult nature of this field could result in modelling approaches being too difficult for practical purposes.

8.2.4 Light Recoverable Robustness

The limitation with our current formulation of Light Recoverable Robust Programs (LRRP) is that there exist two constants that should ideally be part of the optimisation problem itself. Recalling Section 6.5, the models have to have the level of conservatism (in constant Pr) and the total allowed costs (in constant D) predefined. As an optimisation strategy this lacks finesse as we would ideally also like to minimise these things. Furthermore, if an optimiser were to set these constants to be too restrictive, we might not be able to find a solution. As a concrete example, say that an optimiser naively sets $Pr = 1$. It is likely

that model will not be feasible. This goes against the spirit of LRRP, which seeks to relax notions of feasibility as minutely as possible.

However, the price function and limit of total costs cannot be added ipso facto to the objective function as it effectively alter the meaning of it. The approach that has to be taken is one that involves multi-objective (Pareto) optimisation. We need to simultaneously minimise the slack, price function and total costs incurred. This way, we further improve the flexibility of LRRPs. To do this scalarisation techniques such as those proposed in LexOpt need to be explored.

Bibliography

[1] Mauro Maria Baldi, Teodor Gabriel Crainic, Guido Perboli, and Roberto Tadei. The generalized bin packing problem. *Transportation Research Part E: Logistics and Transportation Review*, 48(6):1205–1220, 2012.

[2] Aharon Ben-Tal, Alexander Goryashko, Elana Guslitzer, and Arkadi Nemirovski. Adjustable robust solutions of uncertain linear programs. *Mathematical Programming*, 99(2):351–376, 2004.

[3] Aharon Ben-Tal, Laurent El Ghaoui, and Arkadi Nemirovski. *Robust optimization. Princeton series in applied mathematics.* Princeton University Press Princeton, 2009.

[4] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational geometry: algorithms and applications.* Springer-Verlag [U+2121]OS, 2008.

[5] Dimitris Bertsimas and Constantine Caramanis. Finite adaptability in multistage linear optimization. *IEEE Transactions on Automatic Control*, 55(12):2751–2766, 2010.

[6] Dimitris Bertsimas and Melvyn Sim. The price of robustness. *Operations research*, 52 (1):35–53, 2004.

[7] John R Birge and Francois Louveaux. *Introduction to stochastic programming.* Springer Science & Business Media, 2011.

[8] Peter Brucker. *Scheduling algorithms.* Springer, Berlin ; New York, 5th ed edition, 2007. ISBN 978-3-540-69515-8.

[9] Christoph Buchheim and Jannis Kurtz. Robust combinatorial optimization under convex and discrete cost uncertainty. *EURO Journal on Computational Optimization*, 6(3):211–238, 2018.

[10] Abraham Charnes, William W Cooper, and Gifford H Symonds. Cost horizons and certainty equivalents: an approach to stochastic programming of heating oil. *Management science*, 4(3):235–263, 1958.

[11] Isabel Correia, Luís Gouveia, and Francisco Saldanha-da Gama. Solving the variable size bin packing problem with discretized formulations. *Computers & Operations Research*, 35(6):2103–2113, 2008.

[12] IBM ILOG CPLEX. V12. 1: User's Manual for CPLEX. *International Business Machines Corporation*, 46(53):157, 2009.

[13] George B Dantzig. On the significance of solving linear programming problems with some integer variables. *Econometrica, Journal of the Econometric Society*, pages 30–44, 1960.

[14] George B Dantzig and Mukund N Thapa. *Linear programming 1: introduction.* Springer Science & Business Media, 2006.

[15] George B Dantzig and Mukund N Thapa. *Linear programming 2: theory and extensions*. Springer Science & Business Media, 2006.

[16] AG Doig, AH Land, and AG Doig. An automatic method for solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[17] FICO. *Xpress-optimizer–reference manual, release 20.00*. Fair Isaac Corporation Warwickshire, UK, 2009.

[18] Matteo Fischetti and Michele Monaci. Light robustness. In *Robust and online large-scale optimization*, pages 61–84. Springer, 2009.

[19] Donald K. Friesen and Michael A. Langston. Variable sized bin packing. *SIAM journal on computing*, 15(1):222–230, 1986.

[20] Arthur M Geoffrion. Lagrangean relaxation for integer programming. In *Approaches to integer programming*, pages 82–114. Springer, 1974.

[21] Ralph E Gomory. An algorithm for integer solutions to linear programs. Princeton IBM Mathematics Research Project. *Techn. Report*, (1), 1958.

[22] LLC Gurobi Optimization. *Gurobi Optimizer Reference Manual*. 2018. URL http://www.gurobi.com.

[23] Frank R Hampel. Robust statistics: A brief introduction and overview. In *Research report/Seminar für Statistik, Eidgenössische Technische Hochschule (ETH)*, volume 94. Seminar für Statistik, Eidgenössische Technische Hochschule, 2001.

[24] Frederick S Hillier and Gerald J Lieberman. *Introduction to operations research*. Number BOOK. Holden-Day San Francisco, 1967.

[25] Peter J Huber. Robust statistics. In *International Encyclopedia of Statistical Science*, pages 1248–1251. Springer, 2011.

[26] David S Johnson. *Near-optimal bin packing algorithms*. PhD Thesis, Massachusetts Institute of Technology, 1973.

[27] Anton J Kleywegt, Alexander Shapiro, and Tito Homem-de Mello. The sample average approximation method for stochastic discrete optimization. *SIAM Journal on Optimization*, 12(2):479–502, 2002.

[28] Leo G. Kroon, Rommert Dekker, and Michiel J. C. M. Vromans. Cyclic Railway Timetabling: A Stochastic Optimization Approach. In Frank Geraets, Leo Kroon, Anita Schoebel, Dorothea Wagner, and Christos D. Zaroliagis, editors, *Algorithmic Methods for Railway Optimization*, volume 4359, pages 41–66. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. ISBN 978-3-540-74245-6. doi: 10.1007/978-3-540-74247-0_2. URL http://link.springer.com/10.1007/978-3-540-74247-0_2.

[29] Dimitrios Letsios and Ruth Misener. Exact Lexicographic Scheduling and Approximate Rescheduling. *arXiv:1805.03437 [cs, math]*, May 2018. URL http://arxiv.org/abs/1805.03437. arXiv: 1805.03437.

[30] Christian Liebchen and Sebastian Stiller. Delay resistant timetabling. *Public transport*, 1(1):55–72, 2009.

[31] Christian Liebchen, Marco Lübbecke, Rolf Möhring, and Sebastian Stiller. The concept of recoverable robustness, linear programming recovery, and railway applications. In *Robust and online large-scale optimization*, pages 1–27. Springer, 2009.

[32] James Luedtke and Shabbir Ahmed. A sample approximation approach for optimization with probabilistic constraints. *SIAM Journal on Optimization*, 19(2):674–699, 2008.

[33] Nimrod Megiddo and others. *On the complexity of linear programming*. IBM Thomas J. Watson Research Division, 1986.

[34] Bernhard Meindl and Matthias Templ. Analysis of commercial and free and open source solvers for linear optimization problems. *Eurostat and Statistics Netherlands within the project ESSnet on common tools and harmonised methodology for SDC in the ESS*, 20, 2012.

[35] Omid Nohadani and Kartikey Sharma. Optimization under decision-dependent uncertainty. *SIAM Journal on Optimization*, 28(2):1773–1795, 2018.

[36] Natasha Page. *Vehicle Optimisation in Royal Mail Delivery Offices*. PhD thesis, Imperial College London, 2018.

[37] Bernardo K Pagnoncelli, Shabbir Ahmed, and Alexander Shapiro. Sample average approximation method for chance constrained programming: theory and applications. *Journal of optimization theory and applications*, 142(2):399–416, 2009.

[38] Christos H Papadimitriou. On the complexity of integer programming. *Journal of the ACM (JACM)*, 28(4):765–768, 1981.

[39] Guido Perboli, Roberto Tadei, and Mauro M Baldi. The stochastic generalized bin packing problem. *Discrete Applied Mathematics*, 160(7-8):1291–1297, 2012.

[40] Royal Mail plc. Royal Mail plc – Annual Report and Financial Statements 2017-18. Technical report.

[41] Allen L Soyster. Convex programming with set-inclusive constraints and applications to inexact linear programming. *Operations research*, 21(5):1154–1157, 1973.

[42] Roberto Tadei and Guido Perboli. The Generalized Bin Packing Problem under uncertainty. In *International Conference on Applied and Computational Mathematics*, pages 163–168, 2011.

[43] Philip E Taylor and Stephen J Huxley. A break from tradition for the San Francisco police: Patrol officer scheduling using an optimization-based decision support system. *Interfaces*, 19(1):4–24, 1989.

[44] Hong-Xia Yu and Li Jin. AN BRIEF INTRODUCTION TO ROBUST OPTIMIZATION APPROACH.

[45] Long Zhao and Bo Zeng. An exact algorithm for two-stage robust optimization with mixed integer recourse problems. 2012.

Appendix A

Notation

Symbol	Definition
$\mathbb{T}$	Space of time in seconds from the start of the day
μ_r, σ_r	Robust mean and deviation
$\boldsymbol{A}$	Uppercase bold letters are matrices
$\boldsymbol{a}$	Lowercase bold letters are column vectors
$\boldsymbol{(.)}^T$	Transpose of either a matrix or vectors
$(.)^*$	An optimal solution of a some variable
$(.)^s$	A realisation of uncertainty for some variable for some scenario $s \in \Delta$
$\boldsymbol{c}, \boldsymbol{x}, \boldsymbol{A}, \boldsymbol{b}$	Vector and matrix terms in a linear program of the form $\min \boldsymbol{c}^T \boldsymbol{x}$ s.t. $\boldsymbol{Ax} \leq \boldsymbol{b}$.
$\boldsymbol{A}^0, \boldsymbol{b}^0$	Initial original coefficients of a linear program.
$\hat{\boldsymbol{A}}$	A recovery matrix. This is the matrix of coefficients of constraints imposed during recovery.
$\boldsymbol{\xi}$	A vector of slack variables. This must be of the same dimensionality of $\boldsymbol{b}$. ξ_i is some slack variable used for row i of an optimisation problem
$\boldsymbol{y}$	Vector of recovery decision variables.
$\boldsymbol{Y}$	matrix of recovery decision variables for all scenarios of uncertainty. This implies that the matrix has R columns for R realisations of uncertainty.
α	This is the absolute decrease in the disturbance score achieved by recovery methods on the perturbed solution
β	This is the disturbance score of the perturbed solutions
γ	This is the disturbance score of the recovered solutions
T	The value of discretisation. As T tends to infinity, it models the time horizon more accurately.
S	Discretised time set where $S = \{1 \cdots T\}$.
I	The set of all itineraries. This also doubles as index the set of all items in the bin packing problem.
$I^{\mathcal{C}}$	The index set of all compulsory itineraries
$I^{\mathcal{NC}}$	The index set of all non-compulsory itineraries $I^{\mathcal{NC}} = I \setminus I^{\mathcal{C}}$

J	The index set of all vans where $J = \{1 \cdots \lvert I \rvert\}$. It is overestimated to the size of all itineraries, that is one itinerary executed per van. This also doubles as the index set of all bins in the bin packing problem.
J^*	This refers to the vans used from an optimal feasible solution to the approximate model. As such $J^* \subseteq J$
d_i	The duration of each itinerary. Value of this corresponds to how many discrete time slots of S the itinerary takes up. Equivalantly, this is the weight of an item in the bin packing problem.
$\boldsymbol{p}$	This is a vector of a realisation of perturbations. p_i refers to the change processing time that itinerary i experiences.
g	Upper bound on the number of vans that can physically leave the DO at any given time slot. This is a predefined value. For example, if it were to be desired to represent something along the lines of "only 1 van can leave the DO per minute", this could be done by $g = \lfloor(\texttt{work\_hours} * 60)/(T)\rfloor$. The parameter `work_hours` is described in Table 3.2.
$S_{i,t}$	Defined as the time slots in which itinerary i must have begun in order to be being executed at time slot t. Of the form $S_{i,t} = \{t - d_i + 1, t - d_i + 2, \cdots, t\}$.
F_i	The set of time slots in which itinerary i may begin in order to still be completed by the deadline (i.e. the end of the time horizon). Of the form $F_i = \{1, 2, \cdots, T - d_i + 1\}$.
E	The optional constraint - the time after which no van is allowed to leave.
S_e	The set of time slots in which the vans are not allowed to leave. Of the form $S_e = \{E + 1, \cdots, T\}$.
$x_{i,s}$	A binary decision variable that indicates if itinerary i has begun at some time slot $s \in S$
$y_{i,j}$	A binary decision variable that indicates if itinerary i is being executed by some van $j \in J$
$z_{i,j}$	These are the recovery decision variables. They have the exact meaning as the decision variables $y_{i,j}$. It refers to whether itinerary i is being executed by van j.
$x_{i,j,s}$	A binary decision variable that indicates if itinerary i is being executed by some van $j \in J$ in timeslot $s \in S$
$q_{i,n}$	A binary decision variable which is true when itinerary $i \in I$ uses some buffer B_n after it.
b_j	The maximum processing time van j has in the entire time horizon.
e_j	A binary variable that indicates whether van j is in use. It is not in use if it is not executing any itineraries. This is required as we overestimate the index set J. Similarly, for bin packing, it is true when bin j is in use.

Δ	A representation for uncertainty or an uncertainty set in optimisation problems
$\mathbf{\Delta} \in \mathbb{R}^{\lvert I \rvert \times R}$	Matrix of uncertainty realisation. Each row represents uncertainty realisations for some itinerary i. Each column represents a set of realisations.. Each column represents a set of realisations.
R	The number of uncertainty realisations
$\boldsymbol{\lambda}$	This represents a vector of cost functions. λ_i represents are cost functions indexed by i
λ_1	Cost function measuring number of changes in decision variables from original solution.
λ_2	Cost function measuring amount of `spill_time`.
λ_3	Cost function measuring new vans used.
D	This scalar represents a limit on the total incurred costs.
B	Budget of total buffering time.
B_n	Some discrete buffer as part of the budget such that $\sum_{n=1}^{N} B_n = B$
N	Budget discretisation size. A budget of total buffering time is can be represented as set of buffers whose cardinality is N.
buf_i	Buffer appended *after* itinerary $i \in I$
$shock_z$	The size of a perturbation experienced in z% of the cases.
P	Set of feasible solutions
$\mathcal{A}$	Class of admissible recovery algorithms
Pr	We use this to represent a price function value. It is essentially a ratio of a solution to some optimal solution. We use this as an upper-bound for the deterioration of an objective in light recoverable robust programs.
$\tau, \tau\,(.)$	Let τ be overloaded to mean both the set of bin types, and a function that takes a bin index in J and returns its bin type
L_t, U_t	For some type $t \in \tau$, defined are the lower and upper limits of the number of the corresponding bin type
U	The total number of available bins of any type such that $U \leq \sum_{t \in \tau} U_t$
π_i	The profit associated with item $i \in I$ in the bin packing problem.
W_t	The total weight assignable to a bin of type $t \in \tau$
Π_t	The cost associated with bin of type $t \in \tau$

Table A.1: Table of notation used

Appendix B

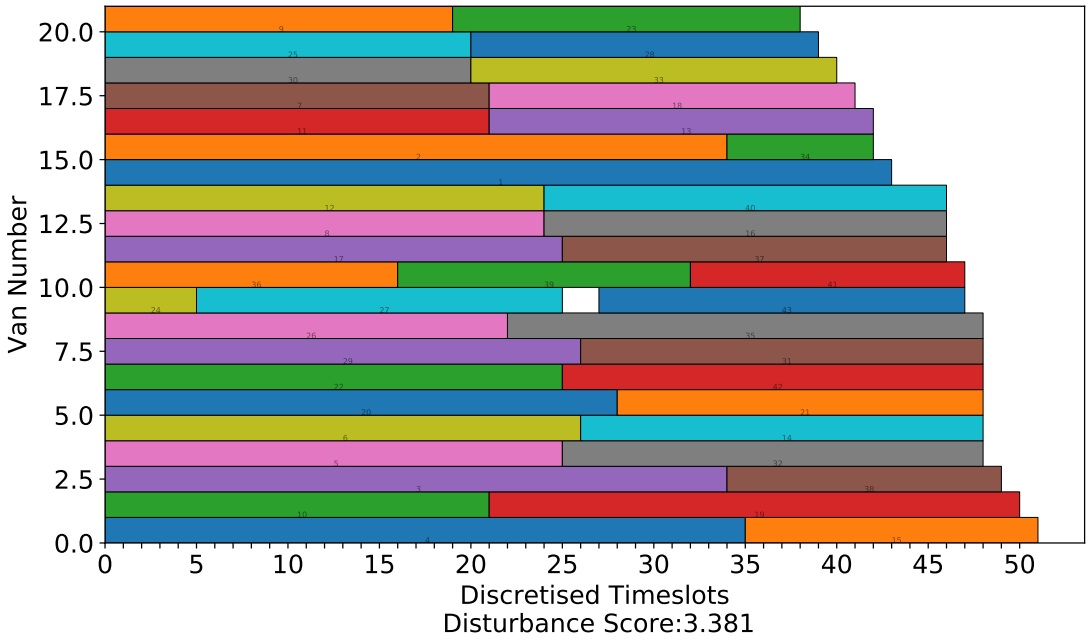
Terminology

Term	**Definition**
DO	Delivery Office
Itinerary	An entry in the data provided by Royal Mail
Deterministic Model	The original model as per Page [36], reiterated in Section 3.6.
LPT	Longest processing time
LRRP	Light Recoverable Robust Programs
GBPP	Generalised Bin Packing Problem
VSBPP	Variable Size Bin Packing Problem

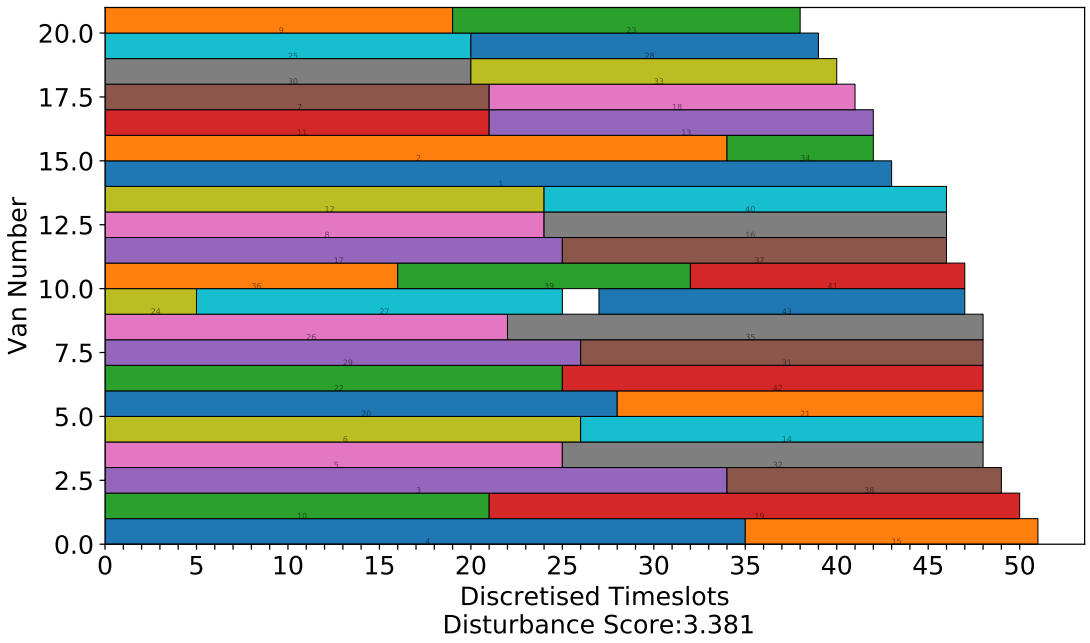
Appendix C

Recovery Visualisations

Presented below are some visualisations of the different recovery methods in action. For each one of these figures, the x-axis represents the discretised timeslots $S = \{1 \cdots 50\}$ of the working day. The y-axis represents each van, i.e, each row is some van's schedule.

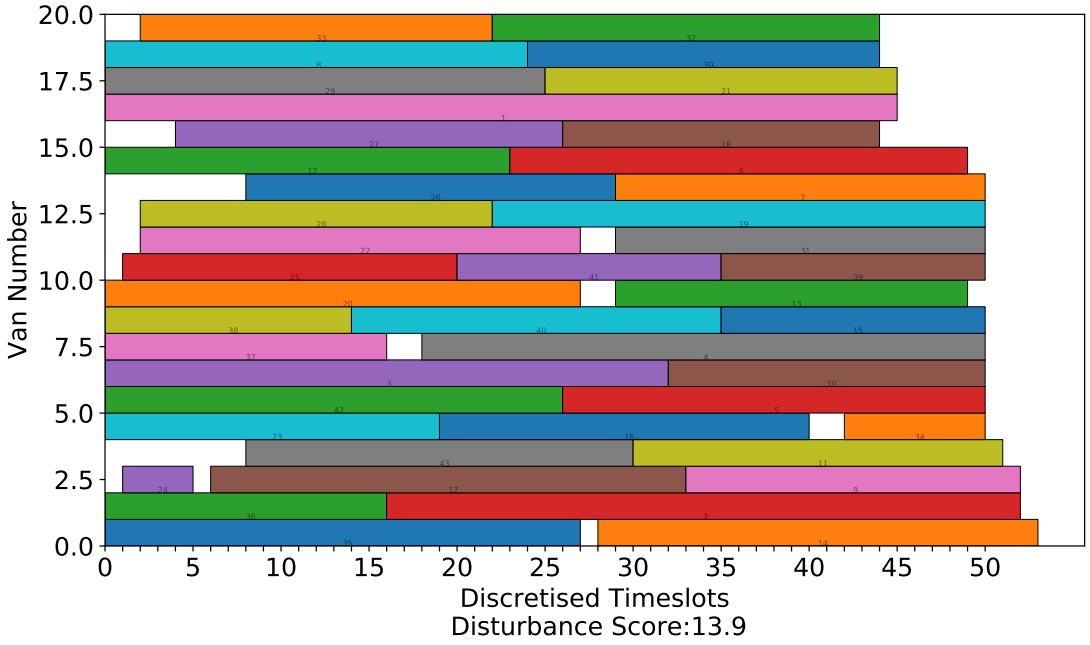


(a) Perturbed Schedule

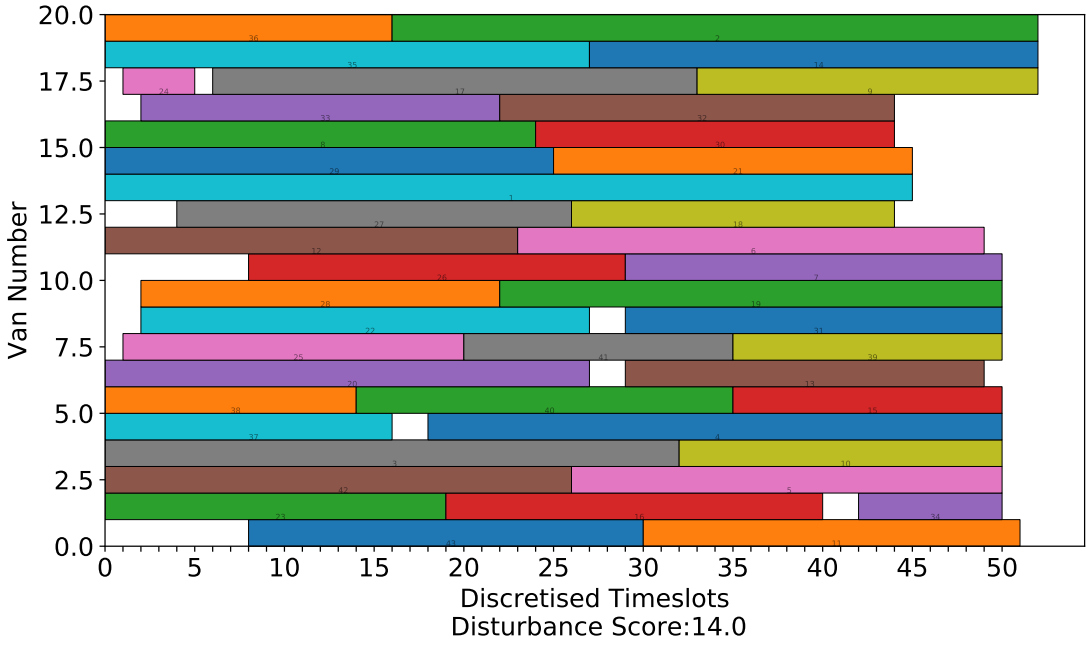


(b) Recovered Schedule

Figure C.1: A visualisation of the properties of the longest processing time recovery applied to an optimal makespan problem. Here we note the optimal substructure imposed my makespan optimality provides a much lower initial score than other methods.

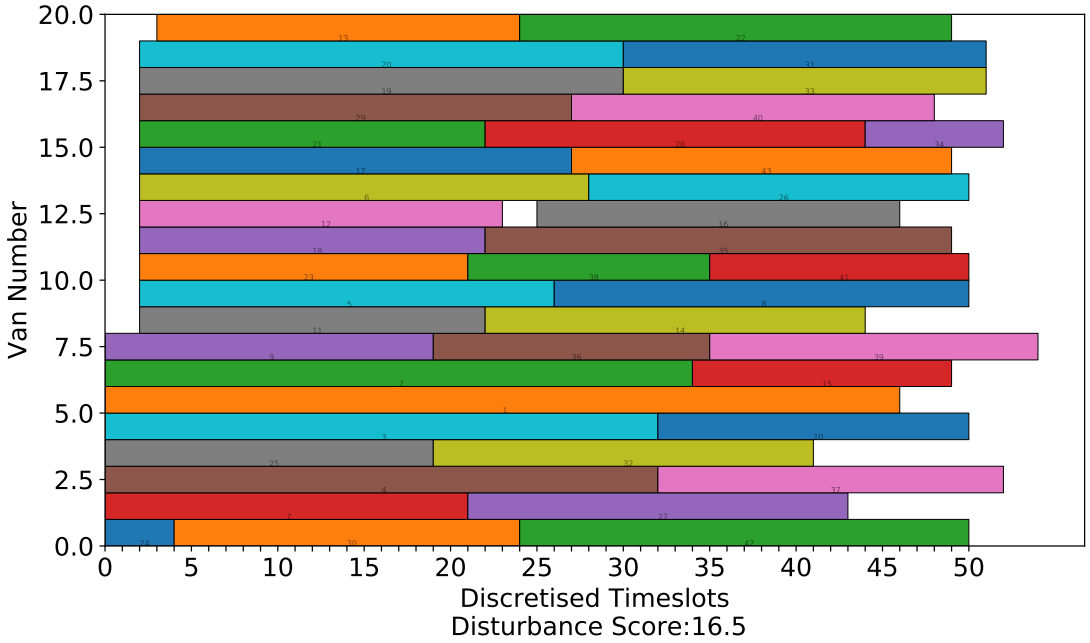


(a) Perturbed Schedule

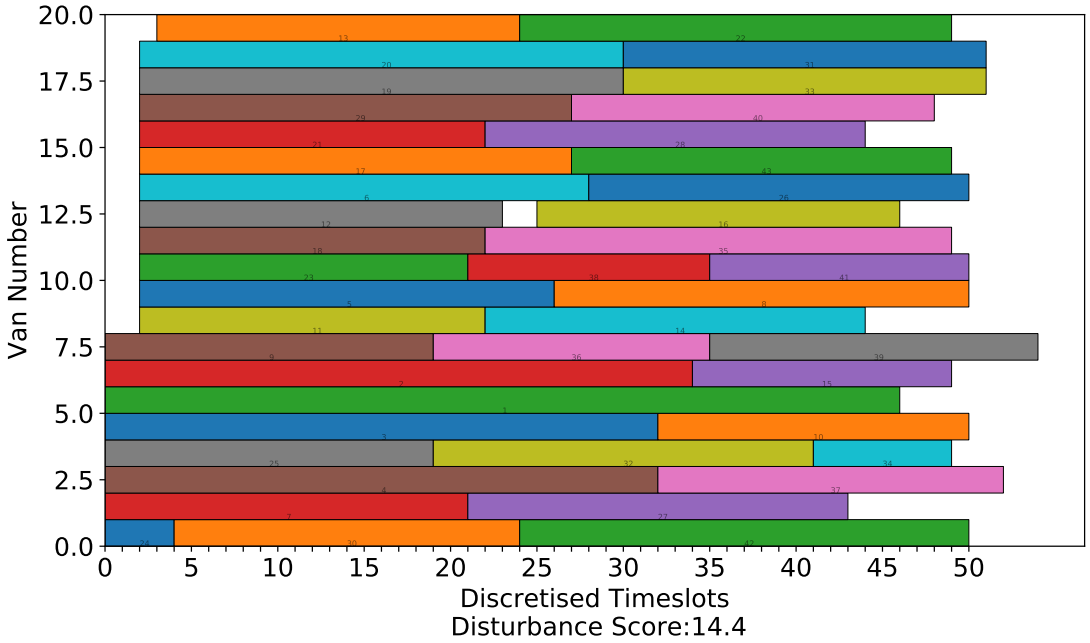


(b) Recovered Schedule

Figure C.2: A visualisation of the longest processing time recovery applied to the deterministic model. The lack of optimal substructure in the initial solution results in a poorly recovered schedule.

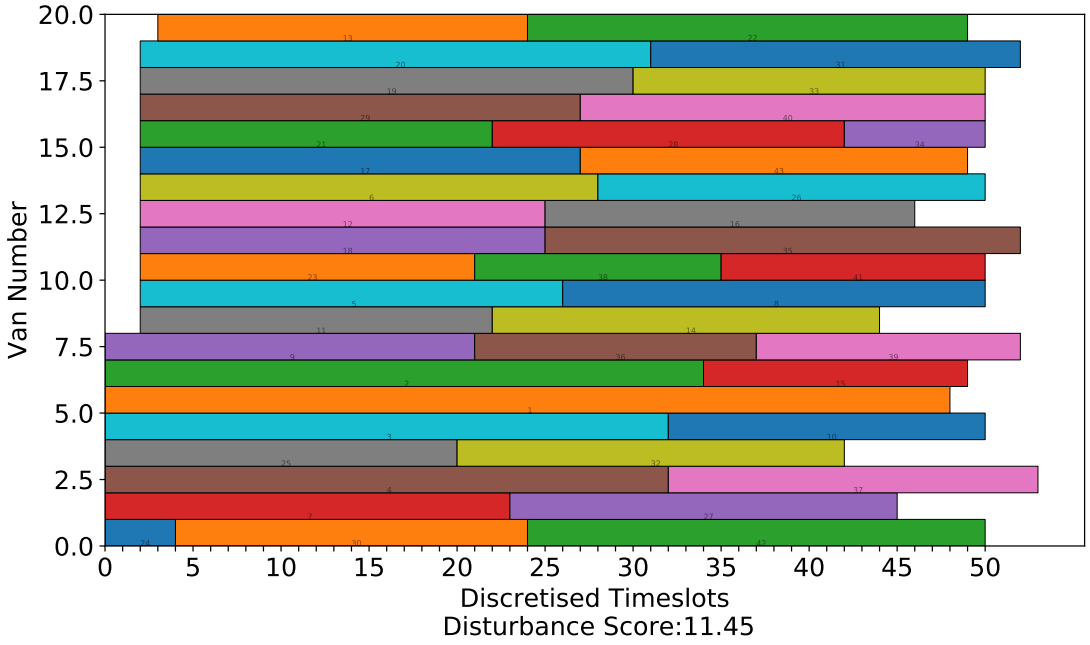


(a) Perturbed Schedule

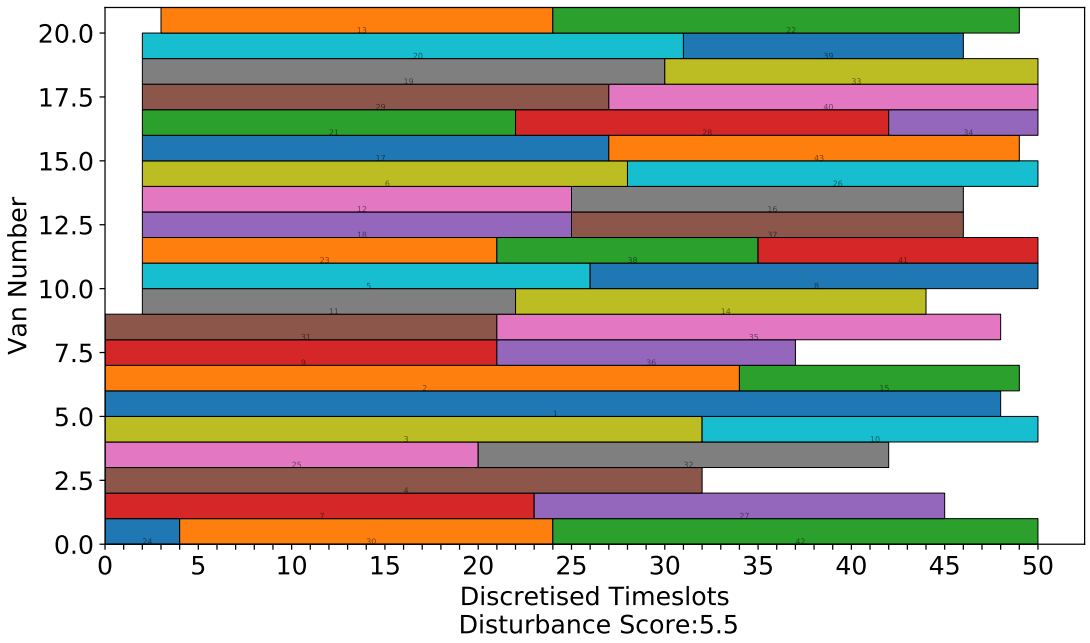


(b) Recovered Schedule

Figure C.3: A simple integer recovery program that is not able to assign new vans. In this integer recovery program we essentially allow two itineraries to be swapped only if it reduces spillage over the time horizon $T = 50$.



(a) Perturbed Schedule



(b) Recovered Schedule

Figure C.4: A complex integer recovery program that is able to assign new vans. However, it is able to find a combination of swaps that reduces costs more than the creation of a new van. Notice the large drop in the disturbance score.