Frequency Assignment Problems

Pri Balachandran

1109960

Third Year

School of Mathematics

Cardiff University

20 Credit Project submitted in partial fulfilment of

the requirements of the degree scheme:

BSc. Mathematics, Operational Research & Statistics

Supervised by Dr. J. M. Thompson

8th May 2014

Abstract

In this project, we aim to construct a program capable of optimising frequency assignment problems. We seek to not only produce a program that can find near-optimal solutions, but to use techniques which may not have been applied to the problem much before. Initial solution formulation methods are constructed in order to provide basic solutions promptly. A number of different optimisation techniques are then appraised, each of which seek to improve upon the initial solutions. This results in the selection of a hyper-heuristic method for implementation in the project.

It is later found that one of the initial solution methods produces excellent results, which rival the Hyper-Heuristic Method as a whole. Specifically, we find that the Spaced Algorithm produces excellent results for all given datasets, in a fraction of the time that the Hyper-Heuristic a whole. However, we conclude that the Hyper-Heuristic algorithm is more likely to perform for a variety of different datasets, given its flexible nature. Furthermore, we identify areas in which this solution can be developed further in future work.

Acknowledgements

I would like to express my gratitude to Dr. Jonathan Thompson for giving me the opportunity to work under his supervision and for guiding me throughout the academic year. I would also like to thank Dr. Vince Knight for introducing me to LaTeXtypesetting.

BLANK PAGE

Contents

BLANK PAGE

Chapter 1

Introduction

In Economics, scarcity is defined as the imbalance of having almost unlimited human wants in a world of limited resources [1]. Oil, land and water are widely considered to be the scarcest resources due to their visibly limited supply, however there is a set of resources that is arguably scarcer still. This set of resources is the radio frequency spectrum. In parts of the world, the demand for radio frequencies has been at saturation point for decades [2], with practically no spare frequencies to allocate. This has pushed academics and policy makers to find new optimisation techniques to further increase utilisation of the existing radio frequency spectrum.

1.1 Context

There is a limited spectrum from which frequencies can be assigned to all communications devices. Thanks to television broadcasts, FM/AM radio, satellite-navigation systems and even the humble mobile phone, people are always keeping connected to something, somewhere. There is a seemingly endless list of wireless communication systems, not least Wi-Fi, GPS and 3G, so it would be logical to think that mankind has acquired all the wireless witchcraft and wizardry necessary to keep this gadgetry happily ticking away. However, the average person now demands ever faster speeds and better quality from their service providers. At present, we are seeing a national roll-out of high-speed 4G internet and an increasing number of HD television broadcasts, all of which need frequencies from the already crowded radio spectrum. Multi-national companies battle on our behalf to procure as much of the frequency spectrum as possible. With British mobile network providers like EE spending hundreds of millions of pounds for frequency bands in OFCOM auctions [3], it is clear that it makes financial sense to invest greatly in this area of Mathematics.

In an ideal world, there would be an infinite range of frequencies. This would have allowed the national roll-out of 4G to have been completed months earlier and there could be analogue and digital television in parallel operation. Perhaps the British public would also have had the luxury of having even more unnecessary television channels such as Channel5HD+1. Unfortunately the reality is that this is nothing more than a broadcaster's fantasy. The radio spectrum from which we can allocate frequencies is highly limited. In the UK, OFCOM limit frequencies from 8.3kHz to 275GHz[4]. Even then, the most extreme ends of the frequency spectrum are only used in fields such as meteorology and astronomy, so the spectrum from which everyday devices are assigned frequencies is even narrower.

Naturally, it is in the interest of the regulator and of the wireless network companies to get as much use out of every frequency as possible whilst making sure constraints are satisfied so that there are no interference problems. The general problem can now be noted as an optimisation problem in which we wish to minimise the number of frequencies we assign to all the different users of the system.

1.2 A Brief History

The earliest evidence of a national-scale attempt to allocate radio frequencies according to some order is in the U.S.A. in 1927. The Federal Radio Commission was created in order to regulate use of the frequency spectrum. The Radio Act of 1927 is one of the earliest examples of a national effort to prevent interference between radio frequencies. It states that "every station shall be required to designate a certain definite wavelength" [5](frequency is inversely proportional to wavelength).

Frequencies were being allocated whenever they were required by a station whilst avoiding selecting frequencies that have already being assigned to a station. There were far more frequencies than there were stations which required them, hence this simple method of finding solutions worked well. This method appears to have sufficed for a few decades until soon the frequency spectrum began to reach saturation point, as there were too many services trying to use the same number of frequencies. Many spectrum managers simply assumed that they had reached the maximum capacity and that there was little more they could do.

Reports were starting to be published in the 1960s, that modelled these problems mathematically [6]. For the first time, the problem had shifted from solely being in the fields of Physics and Engineering, and entered the field of Mathematics. However despite all the work being done by Mathematicians, a large proportion of policy makers, spectrum managers and frequency assigners remained unconvinced about the application of these models to the real world. Even into the 1970s, the vast majority of publications on Spectrum Management would completely ignore the existence of formal mathematical models [7].

In the late 1970s, Zoellner and Beall published a paper that quantified the efficiency increase that could be attained by implementing mathematical models to real-world spectrum management. They suggested that it was possible to apply newly understood graph theoretical techniques to the frequency assignment problems and showed that it was possible to increase usage of the frequency spectrum by 35% or more, through the use of these techniques [8, p. 319].

In William Hale's paper, 'Frequency Assignment: Theory and Applications', he wrote:

> ***"There exists no unifying theory which demonstrates that formal models are a viable approach to the wide range of problems which arise in the real world."*** Hale, 1980.

He said that this was one reason that policy makers were so sceptical about the application of mathematical optimisation techniques to frequency assignment problems. They had no reason to believe that any mathematical model would work in the real world. He stated that his paper was written primarily for policy makers and spectrum managers, in the hope of demonstrating the real-world benefits of modelling these problems [2].

In the decades that followed, a number of different techniques were outlined and applied to frequency assignment problems, including Tabu-Search and ANTS algorithms. Allen, Smith & Hurley published a paper in 1999, which explored techniques to find lower bounds to frequency assignment problems. They stated that the Hamiltonian path (Travelling Salesman) has proved successful for many cases, but not all [9].

Even in the last decade, numerous publications have been written that explore a wide variety of techniques which may further improve the optimisation of frequency assignment problems. An interesting example of some newer work in the field is an optimisation algorithm based on the foraging behaviour of honey bees, which was formally conceived by Karaboga in 2005 [10]. Named the Artificial Bee Colony (ABC) algorithm, its application to the frequency assignment problem was only described last year, in a paper looking at the applications of the algorithm to GSM networks [11].

There are many everyday examples of where technological advancements have depended on frequency allocation. A recent example is the addition of 4G technology to mobile phones. Many manufacturers have had the 4G technology ready and available to them, but had to wait a number of years for sections of the frequency spectrum to be reallocated to network providers. Today, with many technological advancements depending on procurement of wireless frequencies, the importance of optimisation techniques must not be underestimated. This a very exciting area of Mathematics, where fascinating developments are being made on a regular basis. These developments are crucial if mankind is to continue on its current trajectory of becoming more reliant on wireless communication technology.

1.3 Description of the Problem

A set of requests are presented. Each request can be thought of as a person or wireless device, hoping to be assigned a wireless frequency. There will be a list of constraints, specifying the minimum difference that a pair of requests must have between their frequencies. These constraints must be satisfied to ensure that there will be no interference. Finally, there will be a list of frequencies which we can assign to these requests. The crux of the problem is to minimise the number of frequencies used, since this would free up those frequencies for use in other systems.

1.4 Objectives

In this project, the aim is not just to find solutions to a given frequency assignment problem, but to also use a method which may not have been applied to frequency assignment problems much before. This aim will be taken as being of two key parts: the first concerning the project as a whole, the second concerning just the Excel/VBA program.

Firstly, the project itself must consist of in-depth analyses on a variety of solution methods, selecting one which is of interest and has fewer publications specifically outlining its application to frequency assignment problems. Secondly, the program should be able to produce a variety of solutions with different runtimes. A basic solution should be quick to calculate and near-optimal solutions may take a longer period of time.

In order to satisfy these two objectives, we break them down further into more specific aims.

- The project must detail what types of frequency assignment problems exist, if the datasets presented are of any particular type.

- The project must detail possible solution methods to the problem, showing algorithms that have already been applied to the problem and others that have not.
- The project should summarise the capabilities and limitations of the program produced
- The program must produce at least a basic solution for a dataset of any size, up to the limit of the Excel sheets and VBA variable sizes.
- The program should be able to produce a basic solution in a short time, regardless of how many frequencies are used.
- The program should be able to produce a solution that is near-optimal
- The program's VBA code should be well-written, so that alterations can be made to the code by other people.
- The program should be easy to use, so that those who are not as skilled in Microsoft Excel can operate the program on some level
- The project must output useful data for analysis.

1.5 Brief Project Overview

Before continuing with the project, it is important to decide on what should be studied and discussed.

- Study of related mathematics (eg. Graph Theory)
- Possible Solution Methods
- Possible Appraisal Methods/Cost Functions
- Conclusion

1.6 Graph Theory

1.6.1 The Graph Colouring Problem

The Graph Colouring Problem, found within Graph Theory, is a problem in which each of the vertices in the graph must be coloured such that no adjacent vertices share a colour. In other words, any two vertices found to share an edge, must be of different colours. The aim is to minimise the number of colours used. Considering frequency assignment problems, it can be seen that any two requests found to share a constraint, must take different frequencies. This problem can be compared to graph colouring problems, since the requests can be represented as vertices and the constraints as an edge [2, pp.1509-1512].

1.6.2 Cliques

If a larger situation is taken, in which there are a number of different vertices all connected in different ways, a study of cliques may become important. A clique is a group of vertices which are all directly connected to each other. In other words, any pair of vertices in a clique are connected by an edge. For the frequency assignment problem, this means that a clique of requests is a set of requests in which any pair of those requests have a constraint between them. Where the clique is the largest in a dataset, it is referred to as the **maximal clique**.

In Figure 2.1 there are 3 separate cliques. There are two small cliques, one of which is formed of r_3 and r_5, and the other formed of r_{18} and r_4. The clique formed by the vertices r_1, r_2, r_6 and r_8, is a maximal clique of size 4. It is a clique since any pair of those four vertices have an edge between them. It is a maximal clique because it is the largest clique in the dataset.

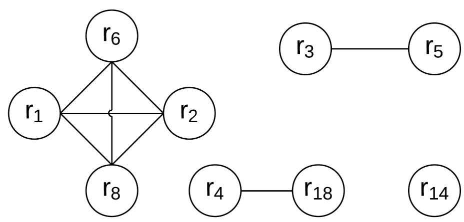

Figure 1.1: Simple Clique Diagram with Maximal Clique of 4

In Figure 1.2, the maximal clique is still 4. If this diagram was approached as through it was a graph colouring problem, it could be seen that only 4 colours would be required. These colours would be used for vertices r_1, r_2, r_6 and r_8. Vertex r_3 could use the same colour as vertex r_1, vertex r_4 could use the same colour as r_6, and vertices r_5 and r_{18} could be the same colour as r_8.

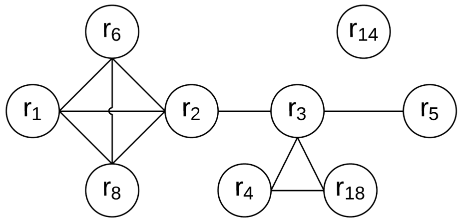

Figure 1.2: Another Clique Diagram with Maximal Clique of 4

Since this project is an NP-hard problem, it will prove very difficult to find a perfect solution, so we shall simply look for the best possible solution. Using the graph colouring problem and a study of maximal cliques, it may be possible to find the largest clique within a given dataset, to find the lower bound for the number of frequencies used.

Chapter 2

Candidate Solution Methods

In this chapter, a number of different solution methods are appraised, with the intention of selecting one to implement in this project. Prior to that, the types of frequency assignment problems must be studied, in order to decide which best reflects the aim of this project. This decision will ensure that any literature is relevant to what this project is seeking to do.

2.1 Categorising the Problems

Frequency assignment problems of different types must be solved in different ways. For example, some problems might not produce feasible solutions, whereas others may require a reduction in the span of the frequencies used [12]. The datasets provided must all be categorised as a specific type or types of frequency assignment problem.

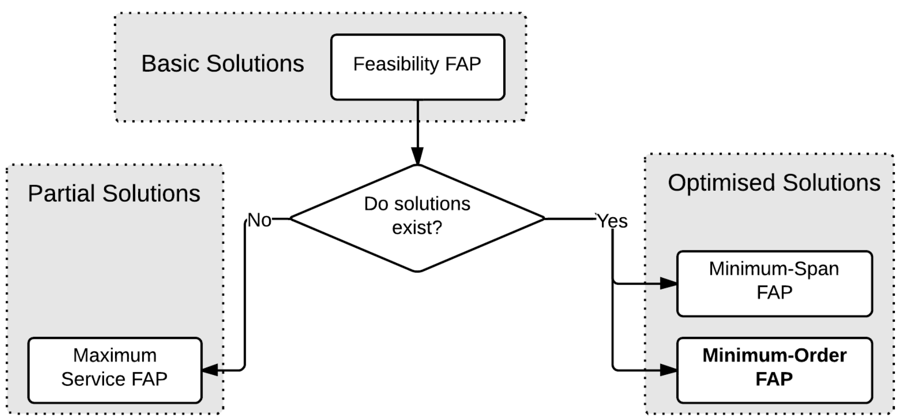

Figure 2.1: Diagram Showing the types of Frequency Assignment Problems

Feasibility FAP (F-FAP)

The basic problem in which we simply seek **any** feasible solution, is called the feasibility frequency assignment problem (F-FAP) [12, p. 90]. When setting out the objectives in Section 1.4, one of them was to able to produce a solution. Hence,

Maximum Service FAP (Max-FAP)

If solutions to the F-FAP **cannot** be found then a partial solution to the problem must be found, in which as many requests are assigned with frequencies as possible. This reduced problem has been called the Maximum Service FAP (Max-FAP). It describes a frequency assignment problem in which the algorithm seeks to maximise the number of requests that can be assigned frequencies in a feasible solution [12, pp.90-91].

If solutions to the F-FAP **can** be found, then some sort of optimisation may be possible. The FAP can then be redefined as one of two types of optimisation problems:

Minimum-Span FAP (MS-FAP)

The Minimum-Span FAP is where the program attempts to minimise the span of the frequencies used. The span is defined as the difference between the highest and lowest frequencies that have been used. The MS-FAP seeks to minimise this, which may be interpreted as a reduction in the bandwidth of the solution [12, pp.92-93]. One possible solution method for MS-FAP would be to use an approximate non-deterministic tree search (ANTS) algorithm, which is derived from the way in which ant colonies behave [13]. However, it must be noted that the MS-FAP is not likely to produce a solution that uses the minimum number of frequencies. Interestingly, for many common examples of frequency assignment problems, it is almost impossible to find an MS-FAP which uses the minimum number of frequencies required [2, p. 1498].

Minimum-Order FAP (Min-Order FAP)

The Minimum-Order Frequency Assignment Problem is another category seeking FAP optimisation [12, p. 92]. This type of optimisation attempts to not reduce the span, but the actual number of frequencies used. It can be seen that this best reflects what this project is aiming for, hence any future reading should be based around MO-FAP literature rather than other FAP types. If it is later found that a feasible solution to the F-FAP cannot be found, then perhaps Max-FAP would have to be used, in order to find the best possible solution.

2.2 Existing Solutions to the Min-Order FAP

2.2.1 Tabu-Search Algorithm

The tabu-search algorithm was first described by Fred Glover in the late 80s as a "meta-heuristic superimposed on another heuristic". It is a relatively new technique, which Glover himself says originated from his work in heuristics for integer programming [14]. The fundamental principle of the tabu-search algorithm is that all paths already traversed are stored in a tabu-list, which is also called the tabu search memory [15]. In 1997, Castelino, Hurley and Stephens published a paper in which they discuss how the tabu-search algorithm can be applied to minimum-order frequency assignment problems [16].

2.2.2 Simulated Annealing Algorithm

Kirkpatrick, Vecchi and Gelatt were the first to notice the potential in solving optimisation problems by quite literally simulating the process of annealing used in metallurgy [17]. Around a decade later, Duque-Antón, Kunz and Ruber published a paper in which they applied this algorithm to the channel assignment problem, which is closely related to the frequency assignment problem. They concluded that the algorithm was successful, despite having issues concerning run-time efficiency and solution quality [18].

2.3 Other Possible Solutions to the Min-Order FAP

2.3.1 Genetic Algorithm

The genetic algorithm emulates the process of natural selection. It uses a combination of techniques that are seen in nature, for example inheritance and mutation. The phrase 'survival of the fittest' was first coined by Herbert Spencer [19]. This is a simple way of describing the manner in which fitter animals are more likely to reproduce. In constructing this method, a cost function would be required, that describes how good a result is. Good solutions can be thought of as fitter than poor solutions. The algorithm can be broken down into three key parts:

Selection

In the wild, the fitter two animals are, the more likely they are to reproduce. Similarly, the program should select two parent solutions based on the cost function. Better solutions have a lower cost. Therefore, the lower the cost of a pair of parent solutions, the more likely they are to be selected to produce offspring. More specifically, selection methods like the roulette wheel selection or tournament selection methods can be used [20].

Crossover

Crossover is the genetic process in which a selection of each parent's chromosomes (genetic information) are taken and combined to form the offspring's chromosomes. There are a number of ways in which this can occur. A simple crossover method would be to pick a random crossover point, and the offspring solution will take the information from one side of the point from one parent solution, and the information from the other side of the point from the other parent solution. Similarly, any n-point crossover could be used, with n separate points from which the genetic information is split and selected. There are many more methods, including three parent crossover and uniform crossover, that can be used here. This variety of solutions is vital since it ensures that the offspring always have different genetic information [20]. In other words, we are ensuring that we continue to have a variety in the population, with some animals fitter than others, allowing the process of selection to continue. The program would perform crossover by selecting different parts of each parent solution, with a given probability.

Mutation

This is the natural process in which changes occur to the genetic information of the offspring. Mutation produces results that may not have been possible using only the genetic information of the parents. This can produce better long-term results, since it allows for the introduction of new bodily functions or techniques that may make the offspring solutions better than other solutions. It can be noted that mutation prevents the genetic algorithm from getting trapped in local minima. Mutation allows a greater variety of solutions to be created, resulting in a greater chance of the algorithm finding the global minimum [20].

2.3.2 Hyper-Heuristic Algorithm

A hyper-heuristic method is one which automates the selection and implementation of a variety of heuristics. Using a variety of different heuristics and an intelligent heuristic selection process, it is possible to have a program that functions in different ways depending on the nature of the given dataset [21][22][23]. The program should be able to find suitable solutions for a wide variety of datasets. For example, the program may be presented with a dataset which consists of a large number of constraints, or perhaps consists of requests and constraints with a very large maximal clique size. It is likely that different datasets will require different approaches to find solutions. Hyper-heuristic methods can be broken down into two parts:

Formulation

An initial solution must be formulated, upon which improvement heuristics can be performed. There are a number of methods by which the initial solution could be constructed. Firstly, a random frequency method could be created, in which requests are randomly assigned frequencies which do not break the constraints. Alternatively, an ascending frequency method could be used, where requests are assigned frequencies in ascending order, using the first frequency found to not break any constraints. Similarly, a descending formulation could be used. This would be the same as the ascending formulation but in reverse order. Importantly, more heuristic methods like these can be added at a later stage, giving the algorithm greater flexibility.

Improvement Heuristics

Improvement heuristics are methods which make small improvements to the solution, each heuristic conducting very specific changes. Possible heuristics include picking a random request and changing it to a frequency giving a lower cost. Alternatively, same cost heuristics could be used, to produce another set of results without any noticeable change in the solution.

2.4 Chosen Algorithm

After appraising the candidate algorithms the Hyper-Heuristic algorithm was selected because of its flexibility and applicability to the problem at hand. The advantages and pitfalls of each algorithm have been outlined below, along with an explanation of why the Hyper-Heuristic Method was selected.

Genetic Algorithm

The genetic algorithm is a very strong solution method, however the implementation of this algorithm to minimum-order frequency assignment problems may be difficult for one key reason. In its most basic form, the algorithm works by taking two halves of feasible solutions and combining them to form a solution which is unlikely to be feasible. This is because the frequency assignment problem is highly constrained problem, so it is likely that many of the solutions produced at each run of this algorithm will be infeasible. The algorithm would have to re-run to produce another offspring solution, which would mean that the algorithm as a whole would be inefficient due to the wasted runtime. In summary, it can be seen that the algorithm can be inefficient if the offspring solution are infeasible and it may be difficult to analyse the performance of the algorithm if alterations are made in order to correct infeasible solutions.

Hyper-Heuristic Method

The key advantage of this solution is that it gives the flexibility of using a variety of different formulation methods and heuristic methods. The program is not limited to one specific method. This will allow for a greater amount of variety in the solutions. There also exists the possibility of incorporating some random changes in order to move the solution closer to the global minimum.

Chapter 3

The General Frequency Assignment Problem

In this chapter, the problem will first be looked at in a more general sense. All the variables and mathematical functions will be defined and explained, so that a formal mathematical description of the problem can then be made. This will then be followed by an appraisal of different cost functions.

3.1 Defining the Problem

The crux of the problem is to minimise the cost function, which may simply be the number of frequencies used. We refer to the number of frequencies used as the order. In the next chapter, a small study of cost functions will be conducted. However, prior to any in-depth mathematical work, the definitions must be looked at.

3.1.1 Variable Definitions

Requests

There are k requests, numbered $1, 2, 3, ..., k$. Each must be assigned a frequency.

Frequencies

There are n frequencies $x_1, x_2, \ldots, x_n$ which are always in ascending order:

$$x_1 < x_2 < \ldots < x_n \tag{3.0}$$

For many of the cost functions, these frequencies will be stored in ascending order of popularity (which is defined later in this section). These rearranged frequencies are $x'_1, x'_2, \ldots, x'_n$.

Distance

Many of the pairs of requests will have a constraint between them, which will state the minimum difference required between each of their assigned frequencies. For example, a constraint may state that requests 16 and 47 must have a difference of at least 100 between their frequencies. The variable $d_{i,j}$ shall hold the required distance between requests i and j. So here $d_{16,47} = 100$. Note that since the minimum distance between i and j is equivalent to the minimum distance between j and i, the symmetry property holds, which means that $d_{i,j} = d_{j,i}$. Also note that the difference between any requests frequency and itself is 0, which implies that $d_{i,i} = 0$.

Popularity

Popularity refers to the number of requests that a frequency has been assigned to. It can be noted that it is more attractive to have a few very popular frequencies than a large number of moderately popular frequencies, since this would result in fewer frequencies being used. As a result, it may be necessary to compare the popularity of each frequency. The function g will be used to denote the popularity. For example $g(x_5) = 12$ implies that frequency x_5 has a popularity of 12, which means it has been assigned to 12 requests.

Popularity-Ordered Frequencies

We define another set of frequencies, which hold the same values as the frequencies x_i, but instead of than having them in ascending order of frequency (as shown in Inequality 3.0), they are now in ascending order of popularity.

So, we have n the popularity-ordered frequencies $x'_1, x'_2, \ldots, x'_n$ which have popularities of $g(x'_1), g(x'_2), \leq, g(x'_n)$. Since they are in ascending order of popularity, each frequency will have a smaller popularity than the next frequency. Therefore, the following inequality holds for any one solution.

$$g(x'_1) \leq g(x'_2) \leq ... \leq g(x'_n) \tag{3.1}$$

The reordering of frequencies in ascending order of popularity, allows for easier calculation of cost, without compromising the mathematics whatsoever. The frequencies

x_i will be referred to throughout the project, whereas the popularity-ordered frequencies x'_i will be used only when looking at the cost function. In the VBA code itself, the popularity-ordered frequencies can only be seen within the cost function.(The relationship between x_i and x'_i are shown in the 'frequency-specific cost' section, on page 17). This of course means that either

Sum of Popularities

Also, since popularity of a frequency is defined as the number of requests it has been assigned to, and every request is assigned with only one frequency, we find that the sum of all popularities is equal to the total number of requests, k:

$$k = \sum_{i=1}^{n} g(x'_i) \tag{3.2}$$

and similarly

$$k = \sum_{i=1}^{n} g(x_i) \tag{3.2a}$$

3.1.2 Function and Cost Definitions

Prior to appraising candidate cost functions, some function definitions are required. As stated when defining popularity-ordered frequencies in Section 3.1.1, the popularity-ordered frequencies x'_i are used whenever working with a cost function, so will be seen throughout this section.

Indicator Function

A binary function is required, to show whether or not a frequency is being used. In other words, a function is required that is equal to one where the popularity is greater than one, and zero otherwise. Naturally, the indicator function is used. We require it to give the value of one for all frequencies x'_i which have been assigned to at least one request, i.e. $g(x'_i) \geq 1$, we define it on the set $\mathbb{Z}^+$, which is the set of positive integers. It is defined as follows

$$I_{\mathbb{Z}^+} : x' \to \{0, 1\}$$

Defined as

$$I_{\mathbb{Z}^+}(x'_i) = \begin{cases} 1 & \text{if } g(x'_i) \in \mathbb{Z}^+ \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

where $\mathbb{Z}^+$ is the set of positive integers.

Frequency Cost

The function $f(x'_i)$ denotes the cost function of each popularity-ordered frequency x'_i. Note that since all of the popularity-ordered frequencies $x'_1, \ldots, x'_n$ are the same as the original frequencies $x_1, \ldots, x_n$ but with the frequencies in different positions, it can be seen that the costs of the popularity-ordered frequencies $f(x'_i)$ have the same values as the costs of the original frequencies $f(x_i)$ but in different positions. For example, for $i = 1, 2, 3, 4$, the following could be true: $f(x'_1) = f(x_3), f(x'_2) = f(x_1), f(x'_3) = f(x_2), f(x'_4) = f(x_4)$.

Total Cost

The total cost is simply the sum of these frequency-specific costs over all frequencies, which will be denoted by $F(X')$ where $X' = (x'_1, x'_2, ..., x'_n)$. Naturally,

$$F(X') = \sum_{i=1}^{n} f(x'_i)$$

This is the total cost of the popularity-ordered frequencies $x'_1...x'_n$. However since the popularity-ordered frequencies only exist for the purposes of calculating the cost function, the total cost of the popularity-ordered frequencies $F(X')$ must be shown to be equivalent to the total cost of the frequencies $F(x)$.

Proof:

Since the $x'_1...x'_n$ are simply a popularity-ordered version of the $x_1...x_n$, their costs $f(x'_i)$ and $f(x_i)$ are also just a re-ordered version of each other. As a result,

$$\sum_{i=1}^{n} f(x_i) = \sum_{i=1}^{n} f(x'_i)$$

Hence, the total cost for the frequencies (without popularity-ordering), F(X) is:

$$F(X) = F(X')$$

Therefore

$$F(X) = \sum_{i=1}^{n} f(x'_i) \tag{3.4}$$

as required. We have shown that the sum of the costs of each of the popularity-ordered frequencies is equal to the total cost of the frequencies (regardless of ordering). This means that we can continue to use the popularity-ordered frequencies to find total cost of the frequencies.

3.1.3 Iteration

An iteration is defined as being a single run of a repeated cycle of code. For example, one iteration of the hyper-heuristic algorithm, would involve selecting and running one improvement heuristic, regardless of the outcome. Iterations can be successful, implying that the solution produced in that iteration did not break constraints, or unsuccessful which implies that the solution produced broke at least one constraint.

3.1.4 Improvements

To help differentiate between successful and unsuccessful iterations, the word 'improvement' is used. An improvement is defined as being a single successful iteration. For example, after 1000 iterations of the algorithm, perhaps only a few hundred improvements were made. 'Improvement' will be a particularly useful term as the hyper-heuristic algorithm approaches the global minimum. This is because as the solution approaches the global minimum, it is likely that a greater proportion of iterations will be unsuccessful. In these unsuccessful iterations, there is no change to the solution, so if we wish to simply discuss the successful iterations, we use the word 'improvement' instead.

3.2 Cost Function Selection

The problem has been described mathematically, and the total cost has been defined as $F(X)$. This cost must now be defined as a function more formally. It will be defined in a number of different ways, each of which will be appraised. The cost function must be able to differentiate between separate solutions that are of the same order. Suppose there exists a situation in which solution A requires fewer alterations than solution B to reduce the order by one. Obviously, solution A is a better solution since it is closer to using one fewer frequency. The cost function must reflect every minute change in the frequency allocations. It must ensure that assigning requests with more popular frequencies gives a lower cost.

Requirements

The cost function must be quick to calculate, since it will be used many times. However, it must also be able to differentiate between two solutions that are of the same order, but where one solution is better than the other. It is likely that selecting the cost function will involve a trade-off of efficiency and effectiveness.

3.2.1 Number Of Frequencies Used (Order)

Description

The simplest cost function to implement in the project would be to use the number of frequencies. Since the primary aim of the program is to minimise the number of frequencies used, it seems logical to start by assessing the feasibility of equating the cost to the number of frequencies used.

Formulation

Any frequency x'_i that has a popularity of at least one, has been assigned to at least one request, and is therefore regarded as 'being used'. This cost function finds the total number of frequencies that have been used, which is equal to the order.
Using the definition of the indicator function in Equation 3.3, the cost for each frequency $x'_1, \ldots, x'_n$ is:

$$f(x'_i) = I_A(x'_i)$$

Giving the sum of all frequency-specific costs:

$$\sum_{i=1}^{n} f(x'_i) = \sum_{i=1}^{n} I_A(x'_i)$$

Using the definition of total cost, from Equation 3.4:

$$F(x) = \sum_{i=1}^{n} I_A(x'_i) \tag{3.5}$$

Analysis

Table 3.1 shows three solutions to a problem with 5 frequencies and 50 requests. The table shows the popularity of each frequency x_i and the total cost $F(x)$ for each solution. Looking at the values in the table, it can be seen that the function works well between solutions A and B . The program takes a request that has used the least popular frequency x'_1, and assigns frequency x'_5 to it. Solution B is better than A and the cost function is lower, as required. When comparing solutions B and C, it can be noted that in Solution B, the lowest non-zero popularity is $g(x'_2) = 2$, which means two changes are needed to make x'_2 redundant and reduce the order. Solution C however, has its lowest non-zero popularity $g(x'_2) = 1$, which is less than that of solution B. This means that C requires only one change to reduce the order, whereas B requires two changes. It is clear that C is a better solution than B, however the cost function remains the same. In other words, the cost function would be unable to show that C is a better solution than B.

Solution	$g(x'_1)$	$g(x'_2)$	$g(x'_3)$	$g(x'_4)$	$g(x'_5)$	Total Cost $F(x)$
A	1	2	5	17	25	5
B	0	2	5	17	26	4
C	0	1	5	18	26	4

Table 3.1: Three different solutions with popularities and total cost, where cost function is 'Order', $k = 50$ and $n = 5$

3.2.2 Popularity Over i

Description

To improve on the previous method, the cost function must now take the popularities, g,of the frequencies into account.

Formulation

Using the popularity-ordered frequencies $x'_1, x'_2, x'_3, \ldots, x'_n$, their frequency-specific cost functions are produced:

$$\begin{aligned}
f(x'_1) &= g(x'_1) \\
f(x'_2) &= \frac{g(x'_2)}{2} \\
f(x'_3) &= \frac{g(x'_3)}{3} \\
&\vdots \\
f(x'_n) &= \frac{g(x'_n)}{n}
\end{aligned}$$

Summing over all frequencies

$$\sum_{i=1}^{n} f(x'_i) = \sum_{i=1}^{n} \frac{g(x'_i)}{i}$$

Using the definition of total cost, from Equation 3.4

$$F(x) = \sum_{i=1}^{n} \frac{g(x'_i)}{i} \tag{3.6}$$

Solution	$g(x'_1)$	$g(x'_2)$	$g(x'_3)$	$g(x'_4)$	$g(x'_5)$	Total Cost, $F(x)$
A	1	2	5	19	23	$13.01\dot{6}$
B	0	2	5	19	24	$12.21\dot{6}$
D	0	0	12	14	24	12.3

Table 3.2: Three different solutions with popularities and total cost, where cost function is 'Popularity Over i', $k = 50$ and $n = 5$.

Analysis

Table 3.2 shows three different solutions to a problem. It shows the popularity $g(x_i)$ of the 5 frequencies x_i as well as the total cost. Comparing solutions A and B in Table 3.2 it can be found that the number of frequencies used has decreased. It is clear that B is a better solution than A, and the cost function is lower as required. Solution B is now compared to solution D. Many requests have changed frequencies between these two solutions. In solution B, x_2 has the lowest non-zero popularity of $g(x'_2) = 2$. In solution D it can be seen that $g(x'_2) = 0$ and so the order of the solution has decreased. However, due to the large number of requests that have been assigned to x'_3, the cost function gives a larger value for D than B. Effectively, the cost function does not recognise that solution D is better than solution B.

In mathematical terms the total cost function $F(x)$ fails if the frequency-specific cost of any individual frequency is greater than the frequency-specific cost of a less popular frequency of a separate solution. In the example above, $f(x'_3)$ in solution D is greater than $f(x'_2)$ in solution B.

Crucially,this method will perform well if only step-wise comparisons are to be made, i.e. if the cost function will only be required to compare solutions in which only one request has changed frequency, then this cost function will perform perfectly. However, it is likely that we will want to compare the costs of many different solutions for a dataset, so an alternative cost function will be required.

3.2.3 Popularity Divided By k^i

Description

To improve on the previous method, the cost of a single frequency must be less than or equal to less popular ones. Again, we use the popularity-ordered frequencies $x'_1, x'_2, ..., x'_n$, where from Inequality 3.1, we know $g(x'_1) \leq g(x'_2) \leq ... \leq g(x'_n)$. We construct an inequality so that frequency-specific cost of the popular frequencies is always less than the frequency-specific cost of less popular frequencies. For **any** solutions to a given dataset, the following equality must hold.

$$f(x'_1) \geq f(x'_2) \geq \ldots \geq f(x'_n) \tag{3.7}$$

Inequality 3.7 holds true for any solutions to a dataset. In other words the different values of x_i can come from different solutions. Since the inequality will always hold, we can say that to reduce the total cost $F(x) = \sum_{i=1}^{n} f(x'_i)$ it is more attractive to assign more popular frequencies such as x'_n (which will have a lower cost).

Formulation

The only way that Inequality 3.7 will hold is if each cost $f(x_i)$ is equal to the popularity $g(x'_i)$, divided by the largest possible value that $g(x'i)$ could take, which is the sum of all popularities. From Inequality 3.2, we know that the the sum of all popularities is equal to the number of requests, k. So, we create the frequencies' cost functions as popularities over powers of k.

For the frequencies $x'_1...x'_n$ with popularities $g(x'_1)...g(x'_n)$, the frequency-specific cost functions are produced as follows:

$$\begin{aligned}
f(x'_1) &= \frac{g(x'_1)}{k} \\
f(x'_2) &= \frac{g(x'_2)}{k^2} \\
f(x'_3) &= \frac{g(x'_3)}{k^3} \\
&\vdots \\
f(x'_n) &= \frac{g(x'_n)}{k^n} \\
\sum_{i=1}^{n} f(x'_i) &= \sum_{i=1}^{n} \frac{g(x'_i)}{k^i}
\end{aligned}$$

Using the definition of total cost, from Equation 3.4:

$$F(x) = \sum_{i=1}^{n} f(x'_i)$$

So the total cost function can be found:

$$F(x) = \sum_{i=1}^{n} \frac{g(x'_i)}{k^i} \tag{3.8}$$

which observes (3.7), or in other words:

$$\frac{g(x'_1)}{k^1} \geq \frac{g(x'_2)}{k^2} \geq \ldots \frac{g(x'_n)}{k^n} \forall x'_i, n, k$$

Solution	$g(x'_1)$	$g(x'_2)$	$g(x'_3)$	$g(x'_4)$	$g(x'_5)$	Total Cost, $F(x)$
A	1	2	5	19	23	0.35776
B	0	2	5	19	24	0.15808
D	0	0	12	14	24	0.12608

Table 3.3: Three different solutions with popularities and total cost, where cost function is 'Popularity Divided By k^i', $k = 50$ and $n = 5$.

Analysis

Table 3.3 shows three solutions with their popularities $g(x'_i)$ and total cost. Comparing solutions A and B in Table 3.3, it can be seen that that the number of frequencies used has decreased. It is clear that B is a better solution than A and the cost function is lower, as required.

Solution B is now compared to Solution D. A number of requests have changed frequencies between these solutions. Again, the number of frequencies has decreased and so has the cost function. This cost function is performing exactly as required.

The only problem that can be found with this cost function is that it limits the size of the dataset that can be used. This cost function works by giving the most popular frequencies a lower cost. It known that the smallest positive value that can be stored by a double data type variable in VBA is $4.94065645841246544 \times 10^{-324}$. Therefore for the total cost function $F(x)$ to work as required, each of its constituent frequency-specific cost functions $f(x'_i)$ must be greater than or equal to that value. From Inequality (3.7), it is clear that the lowest of these frequency specific cost functions is $f(x'_n)$. Hence the limits of the program are defined as follows

$$f(x'_n) = \frac{g(x'_n)}{k^n} \geq 4.94\ldots \times 10^{-324} \tag{*}$$

The sum of all popularities, given in Equation 3.2:

$$k = \sum_{i=1}^{n} g(x'_i)$$

$$k \geq g(x'_n)$$

Dividing through by k^n

$$\frac{1}{k^{n-1}} \geq \frac{g(x'_n)}{k^n}$$

Substituting (*)

$$\frac{1}{k^{n-1}} \geq 4.94... \times 10^{-324}$$

$$k^{n-1} \leq 2.02... \times 10^{323} \tag{3.9}$$

This is the bound for the number of requests k, and number of frequencies n, for which this cost function is known to work. Examples of the bound of this cost function, would be to have $n = 100, k = 1840$ or $n = 50, k = 3.96 \times 10^6$. All datasets provided are well within the bounds of this cost function.

Possible Improvements

The values produced by this cost function can be very very small. Some can be as small as 1×10^{-150} or smaller. It is possible to increase these values to make them more readable for non-mathematicians. Perhaps by putting the frequency-specific cost function as a power of the exponential function. However, to keep the cost function as pure and true as possible, this was not done. Interpreting the cost function and understanding exactly how it works is far easier without adding additional complexity to it. Furthermore, and additional calculations will of course increase the runtime.

3.3 Chosen Cost Function

The Popularity Divided By k^i cost function will be used for this project, despite the merits of the other cost functions. The advantages and pitfalls of each of the cost functions have been outlined below, followed by justification of the final choice of cost function.

Number of Frequencies Used

This is by far the most efficient cost function, since it requires the least calculation to be done. It is also the easiest to interpret since it it always relates directly to the problem, telling the user exactly how many frequencies are used at any one point in time.

However, it is unable to differentiate between two solutions using the same number of frequencies.

Popularity Over i

This cost function requires slightly more computation when compared to the 'Number of Frequencies Used' cost function, and is therefore slightly less efficient. However it is slightly more effective since it is able to differentiate between solutions using the same number of frequencies with only one request change between them, though it must be noted that it is not guaranteed to be able to differentiate between any two random solutions which use the same number of frequencies.

Popularity Divided By k^i

This is by far the least efficient cost function because of the amount of calculations that need to be performed, however it is able to differentiate between any two solutions which makes it the most effective cost function. However, this effectiveness has come at a price, in that this cost function has a much lower limit on the size of dataset which can be used. Fortunately, it would be quite simple to replace the cost function within the VBA code if needed.

3.4 Mathematical Description of the Problem

The problem can now be defined, using the 'Popularity Divided By k^i' cost function that has been selected. Using the total cost that was defined in Equation 3.4 and the total cost equation, Equation 3.8, we form the problem:

Assign each of the k requests, $r_1, r_2, \ldots, r_k$ with one of the n frequencies $x_1, x_2, \ldots, x_n$, subject to the constraints $d_{i,j}$ between all pairs of requests i and j.

$$
\begin{aligned}
\text{minimise } F(X) &= \sum_{i=1}^{n} \frac{g(x_i')}{k^i} \\
\text{s.t. } |r_i - r_j| &\geq d_{i,j} \qquad\qquad \forall i, j \in \{1...k\} \\
\text{where } d_{i,j} &= d_{j,i} \\
d_{i,i} &= 0
\end{aligned}
$$

Chapter 4

Hyper-Heuristic Methods

In this chapter, each of the key aspects of the Hyper-Heuristic at outlined. Their application to the program has been explain and shown in Pseudocode. This has been done for each of the formulation algorithms, each of the heuristic algorithms, as well as the main algorithms which allow the whole hyper-heuristic to work.

4.1 Background

Since the Hyper-Heuristic algorithm has been selected, its application to this project must now be described. The algorithm can be broken down into two key parts, Formulation Methods and Improvement Heuristic Methods. The Formulation Methods will all be algorithms which produce basic, initial solutions. The Improvement Heuristic Methods will all be separate heuristics which will each improve or alter the initial solution in different ways. Alongside this, Main Methods which change the probabilities and call the heuristics, must also be used. Figure 4.1 is a vastly simplified flowchart showing the Hyper-Heuristic method being broken down into three key parts. There are three further sections in this chapter, the first of which will explain the way in which each of the Formulation Methods work. The second outlines how the Improvement Heuristics each work. The final section will explain the Main Methods from which everything is called.

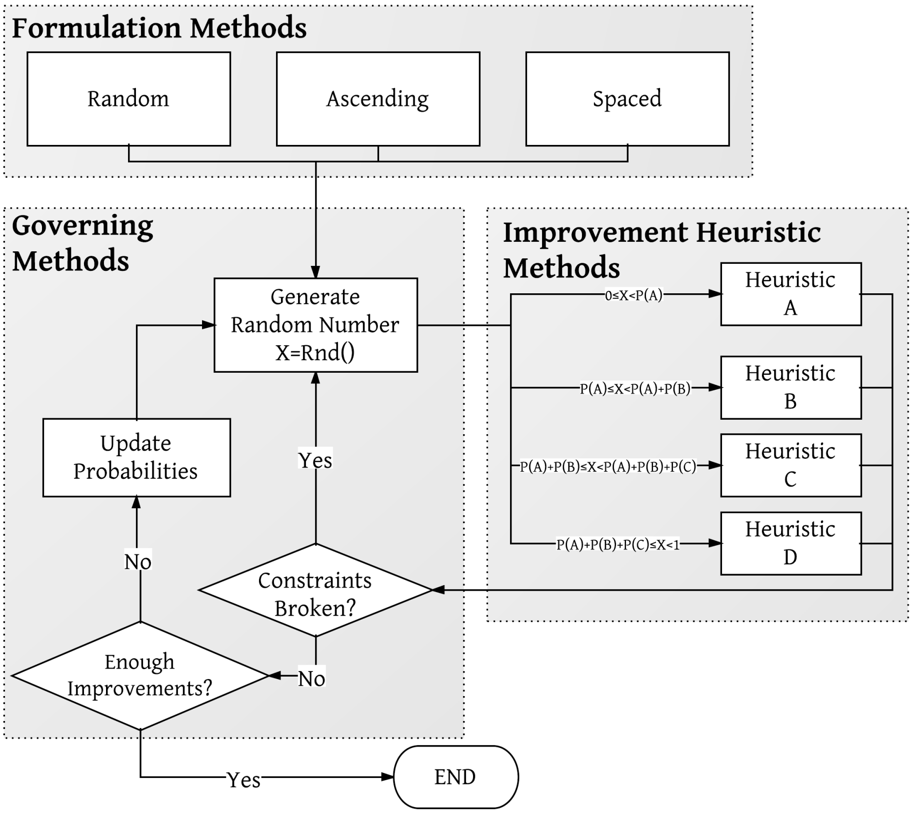

Figure 4.1: Vastly simplified structural overview of the whole Hyper-Heuristic algorithm

4.2 Formulation Methods

Firstly, the program will have a few different methods which can be used to formulate solutions. These are the Random Algorithm, the Ascending Frequency Algorithm and the Spaced Frequency Algorithm. Each of these algorithms are seen to work in slightly different ways and each of them have their own advantages and problems.

4.2.1 Random Algorithm

Overview

In this method, it is hoped that a solution will be formed in which little or no effort is made to minimise the order of the solution. Requests are simply assigned frequencies without breaking any of the constraints. No further parameters are required, simply the constraints and frequencies.

Method

Within the code, it can be seen that the algorithm works by assigning a random frequency to the first request in the domain, then assigning another random frequency to the second request in the domain. This is then checked with the previous request against the constraints. This continues until the last request has been assigned a frequency. Where a frequency assignment has been rejected because it breaks the constraint, it will then be reassigned with another completely random frequency.

It is possible that after many repetitions, no suitable frequency will be found for a particular request. To prevent an infinite loop, an escape value is set. This escape value is held in *FreqAssignIterationsEscape*. The default value equal to the number of requests, meaning that in dataset 1, after 916 successive unsuccessful attempts to find a random frequency for a request, the algorithm is said to have failed. A more efficient method would be to have remove frequencies from a list if they have already been tried, however since this algorithm has been able to produce results within a matter of seconds, there was very little need to such an improvement. Furthermore, it was felt that this would add unnecessary complexity to the code.

Pseudocode

$Request \leftarrow 1$
$Domain(Request) \leftarrow$ RandomFrequency x_i
for $Request \leftarrow 2$ to k **do**
    repeat
        $Domain(Request) \leftarrow$ RandomFrequency x_i
        Check Constraints
        if Any constraint has been broken **then**
            $Failures \leftarrow Failures + 1$
        end if
        if $Failures \geq FreqAssignIterationsEscape$ **then**
            Restart Algorithm with $Domain(1) \leftarrow RandomFrequency$
        end if
    until No Constraints Broken
end for

Notes

Since the algorithm assigns frequencies randomly, it is likely that the solution it gives will use a large amount of the frequencies. For example, if the algorithm was presented with a dataset in which there were only a few constraints, the algorithm would still assign the requests with a random variety of frequencies. As an initial solution, this algorithm is not very effective, however it may prove to be useful when making comparisons and judgments of the improvement heuristic methods.

It is found that this algorithm It is more efficient to have an array with frequencies, then remove the ones that have already been tried (so max of 48 iterations) Comment on this, saying that after trying with all datasets, this algorithm worked after just a few iterations, so there is no need for the extra complexity. However this is a possible improvement.

4.2.2 Ascending Algorithm

Overview

Looking at the random algortihm, it was clear that selecting frequencies at random simply forced more frequencies into the solution, where they may not have been necessary. Here, a more linear approach is adopted, by selecting the frequencies in order rather than at random.

Method

In this method, the requests are worked through in order much like the Random Algorithm. However, the frequencies are also assigned in order. The frequencies are already held in ascending order in the array *FrequencyList()*, however the user may decide to use a different frequency as the first in the list. Should the user decide to change the start frequency, the sub-procedure *ReOrderFrequencyList()* will reorganise the list, so that the selected start frequency is at position one in the array. The array then continues in ascending order as usual, and ends with the frequencies left over.

For example, take the frequencies $(x_1, x_2, ..., x_5, x_6, x_7, ..., x_n)$. The starting frequency here is x_1. If the user was to select x_6 as the starting frequency, then *ReOrderFrequencyList()* would leave the frequencies in this order $(x_6, x_7, ..., x_n, x_1, x_2, ..., x_5)$. The algorithm will take the first request and assign it the first frequency. It will then take the second request and attempt to assign it to the first frequency. If any constraints are broken, it will attempt to assign it to the second frequency, then the third frequency and so on. This process is repeated with every single request.

Pseudocode

The following pseudocode is a basic overview of the way in which this algorithm works. It assumes that the frequencies x_i have already been reordered so that the user-selected start frequency is in x_1 and the other frequencies follow in ascending order.

$Request \leftarrow 1$
$Domain(Request) \leftarrow x_1$
for $Request \leftarrow 2$ to k **do**
    $i \leftarrow 1$
    repeat
        $Domain(Request) \leftarrow$ FrequencyList(i) x_i
        Check Constraints
        if any constraints are broken **then**
            $i \leftarrow i + 1$
        end if
    until No Constraints Broken or $i = NumberOfFrequencies$
    if $i = NumberOfFrequencies$ **then**
        Algorithm failed. Re-order FrequencyList()...
        $Temp \leftarrow FrequencyList(1)$
        for $i \leftarrow 2toNumberOfFrequencies$ **do**
            $FrequencyList(i-1) \leftarrow FrequencyList(i)$
        end for
        $FrequencyList(NumberOfFrequencies) \leftarrow Temp$
        Restart algorithm with $Domain(Request) \leftarrow x_1$, which was previously x_2
    end if
end for

Notes

However, one of the problems with this method, is that it produces the same result for each start frequency, every time. It may be better to have a variety of initial solutions upon which improvement methods can be constructed. To perform a large amount of improvement methods for analysis, the random method may be better.

4.2.3 Spaced Algorithm

Overview

Since the constraints for this problem will all be minimum distance (greater than) constraints, it may prove to be beneficial to have an algorithm in which the frequencies that are most likely to be selected, are all spaced-out. For example, if the largest constraint has a value of 58, if the frequencies used have a difference of at least 58, it may be possible to reduce the number of frequencies that are used overall.

Method

The way in which requests are assigned with frequencies happens in exactly the same way as the Ascending Algorithm. The only difference is that rather than using *FrequencyList()* to assign from, *SpacedFrequencyList()* is used. *SpacedFrequencyList()* is an array in which the frequencies have been arranged such that the first few frequencies satisfy the *SpacingDistance*. For example, if *SpacingDistance*=42, then another frequency x_i is required, such that $x_i - x_1 \geq 42$. As such, it can be seen that the new array *SpacedFrequencyList()* will now contain the frequencies in a different order. For example $(x_1, x_4, x_5, x_8, x_9, x_{11}, x_2, x_3, x_6, x_{10})$, where $x_1, x_4, x_5, x_8, x_9, x_1 1$ are the spaced frequencies, which are followed by the unspaced frequencies.

Pseudocode

Again, the following pseudocode is a very basic overview of the way in which this algorithm works. As usual, the frequencies x_i are in ascending order, i.e. $x_1 \leq x_2 \leq \ldots \leq x_n$.

i) Spaced Frequencies: Firstly, a set of frequencies are found such that each frequency is at least the *SpacingDistance* apart from the previous frequency. These frequencies are all placed in ascending order into the array, *SpacedFrequencies()*.

$i \leftarrow 1$

$CurrentFrequency \leftarrow Frequencies(i)$

repeat

  $SpacedFrequencies(Position) \leftarrow CurrentFrequency$

  $MinimumNextFrequency = CurrentFrequency + SpacingDistance$

  Find the first $FrequencyList(i)$ that is $\geq MinimumNextFrequency$

  $CurrentFrequency = FrequencyList(i)$

until End of Frequencies()

ii) Other Frequencies: Secondly, we find all the frequencies which have not already been included in*SpacedFrequencies()* and add them to it.

for $i \leftarrow 1$ to $NumberOfFrequencies$ **do**
  if $FrequencyList(i)$ does NOT exist in the $SpacedFrequencies()$ **then**
    Add $FrequencyList(i)$ to the end of $SpacedFrequencies()$
  end if
end for

iii) Assigning to Requests: Finally, we assign requests with frequencies from *SpacedFrequencies()* with preference given to the frequencies occurring earlier in the array.

$Request \leftarrow 1$
$Domain(Request) \leftarrow x_1$
for $Request \leftarrow 2$ to k **do**
  $i \leftarrow 1$
  repeat
    $Domain(Request) \leftarrow$ SpacedFrequencies(i) x_i
    Check Constraints
    if any constraints are broken **then**
      $i \leftarrow i + 1$
    end if
  until No Constraints Broken
end for

4.3 Improvement Heuristic Methods

In this section, each of the four Improvement Heuristic Methods are outlined individually. For each method an overview is given, explaining what the method aims to do. This is followed by an in-depth explanation of how it will work in the context of the Hyper-Heuristic algorithm as a whole.

4.3.1 A - Remove Least Popular Frequency

Overview

The program revolves around the popularity of each frequency. From Equation 3.3, it can be seen that the greatest reduction in cost can be brought about by taking a request that has been assigned the least popular frequency and assigning it with the most popular frequency. A function will be required to find the most and least popular frequencies.

Method

This heuristic is called from the main *HyperHeuristic* sub-procedure. The heuristic uses the function *ChooseLeastPopularFrequency* to find the frequency with the fewest (but non-zero) requests assigned to it. The heuristic then assigns a different frequency to it. This heuristic is designed to loop until a lower cost solution is found. Once a lower cost solution is produced, the program returns to the main *HyperHeuristic* sub-procedure. If the solution breaks any constraints, then the variable *HeuristicOkay* returns *FALSE* to the main *HyperHeuristic* sub-procedure, otherwise *HeuristicOkay* returns *TRUE*.

Pseudocode

$LeastPopFreq \leftarrow ChooseLeastPopularFrequency$
$Request \leftarrow FindRequest(LeastPopFreq)$
$OldFreq \leftarrow Domain(Request)$
$OldCost \leftarrow CalculateCost(Domain())$
repeat
  $Domain(Request) \leftarrow ChooseRandomFrequency$
  $NewCost \leftarrow CalculateCost(Domain())$
until $NewCost \leq OldCost$
Check All Constraints
if any constraints are broken **then**
  $HeuristicOkay \leftarrow FALSE$
else
  $HeuristicOkay \leftarrow TRUE$
end if

4.3.2 B - Lower Cost Move

Overview

This heuristic will pick any *RandomRequest* to assign a new frequency to, provided the solution is an improvement and therefore lower cost.

Method

Again, this heuristic is called from the main *HyperHeuristic* sub-procedure. The heuristic will pick a request by using *ChooseRandomRequest* then it will replace its current frequency with a new one found by *ChooseRandomFrequency*. The heuristic is designed to loop until a lower cost solution is found. If the solution breaks any constraints, then the variable *HeuristicOkay* returns *FALSE* to the main *HyperHeuristic* sub-procedure, otherwise *HeuristicOkay* returns *TRUE*.

Pseudocode

repeat
    $NewDomain() \leftarrow Domain()$
    $Request \leftarrow ChooseRandomRequest$
    $NewDomain(Request) \leftarrow ChooseRandomFrequency$
    $OldCost \leftarrow CalculateCost(Domain())$
    $NewCost \leftarrow CalculateCost(NewDomain())$
    if $NewCost \leq OldCost$ **then**
        $Domain(Request)\ getsNewDomain(Request)$
        $FreqOkay \leftarrow TRUE$
    else
        $FreqOkay \leftarrow FALSE$
    end if
until $FreqOkay$ returns $TRUE$
Check All Constraints
if any constraints are broken **then**
    $HeuristicOkay \leftarrow FALSE$
else
    $HeuristicOkay \leftarrow TRUE$
end if

4.3.3 C - Same Cost Swap

Overview

The aim of this heuristic is to move to other parts of the set of possible solutions, wherever improvements using heuristics A and B are starting to fail a large proportion of the time.

Method

This heuristic will pick two random requests by using the *ChooseRandomRequest* function then swap their frequencies. No looping is performed so this is a very quick sub-procedure. Again, this heuristic is called from the main *HyperHeuristic* sub-procedure. If the solution breaks any constraints, then the variable *HeuristicOkay* returns *FALSE* to the main *HyperHeuristic* sub-procedure, otherwise *HeuristicOkay* returns *TRUE*.

Pseudocode

repeat
  $RequestA \leftarrow ChooseRandomRequest$
  $RequestB \leftarrow ChooseRandomRequest$
until $RequestA \neq RequestB$
$Temp \leftarrow Domain(RequestA)$
$Domain(RequestA) \leftarrow Domain(RequestB)$
$Domain(RequestB) \leftarrow Temp$
Check All Constraints
if any constraints are broken **then**
  $Temp \leftarrow Domain(RequestA)$
  $Domain(RequestA) \leftarrow Domain(RequestB)$
  $Domain(RequestB) \leftarrow Temp$
  $HeuristicOkay \leftarrow FALSE$
else
  $HeuristicOkay \leftarrow TRUE$
end if

4.3.4 D - Worse Move

Overview

This can be regarded as the opposite to Heuristic B. Rather than moving a request to a more popular frequency, this heuristic will assign it to a less popular frequency. As a result, this heuristic will increase the cost of the solution, so will only be desirable when we are seeking a different set of solutions.

Method

This heuristic is called from the main *HyperHeuristic* sub-procedure. The heuristic will pick a request by using the function *ChooseRandomRequest*. It will then replace its current frequency with a new one found by *ChooseRandomFrequency*. The heuristic is designed to loop until a higher cost solution is found. If the solution breaks any constraints, then the variable *HeuristicOkay* returns *FALSE* to the main *HyperHeuristic* sub-procedure, otherwise *HeuristicOkay* returns *TRUE*.

Pseudocode

repeat
    $NewDomain() \leftarrow Domain()$
    $Request \leftarrow ChooseRandomRequest$
    $NewDomain(Request) \leftarrow ChooseRandomFrequency$
    $OldCost \leftarrow CalculateCost(Domain())$
    $NewCost \leftarrow CalculateCost(NewDomain())$
    if $NewCost > OldCost$ **then**
        $Domain(Request)\ getsNewDomain(Request)$
        $FreqOkay \leftarrow TRUE$
    else
        $FreqOkay \leftarrow FALSE$
    end if
until $FreqOkay$ returns $TRUE$
Check All Constraints
if any constraints are broken **then**
    $HeuristicOkay \leftarrow FALSE$
else
    $HeuristicOkay \leftarrow TRUE$
end if

4.4 Governing Methods

Now that the Formulation Methods and Improvement Heuristic Methods have been defined, the methods which link and manage all of these methods must be defined. As shown in Figure 4.1, (on page 28), there are a variety of tasks that must be performed by these methods, including taking the initial solution, and passing it to different Improvement Heuristic Methods until sufficient Improvements have been performed. The Heuristic Selection Process is defined in Section 4.4.1, which describes the main code from which all Hyper-Heuristic-related procedures are called. The adjustment of probabilities is defined in Section 4.4.2, which describes the Adjust Probabilities sub-procedure.

4.4.1 Main

Outline

As shown in Figure 4.1 the heuristics are each assigned probabilities. By using a random number generator, the program will then select heuristic to run. The pseudocode below is a slightly simplified version of the code, showing only the core statements. It is important to note that as defined in Section 3.1.4, each iteration of the below algorithm does not necessarily correspond to an improvement, since an iteration is a loop of the algorithm, whereas an improvement is a loop that successfully improves the solution.

Pseudocode

Each of the Heuristics have probabilities assigned to them P(A), P(B), P(C), P(D). The initial values of these probabilities is inputted by the user on the RunProgram sheet of the Excel program. The function $RND()$ is used within VBA to pick a random number to select a heuristic. (*Randomize* is also used, to give $RND()$ a new seed value).

repeat
  $X \leftarrow RND()$
  P(A) ← Input Start Probability for Heuristic A
  P(B) ← Input Start Probability for Heuristic B
  P(C) ← Input Start Probability for Heuristic C
  P(D) ← Input Start Probability for Heuristic D
  if $X <$ P(A) **then**
    Call $HeuristicA$

else if $X < \mathrm{P}(A) + \mathrm{P}(B)$ **then**
  Call $HeuristicB$
else if $X < \mathrm{P}(A) + \mathrm{P}(B) + \mathrm{P}(C)$ **then**
  Call $HeuristicC$
else
  Call $HeuristicD$
end if
if $HeuristicOkay = TRUE$ **then**
  $Improvement \leftarrow Improvement + 1$
end if
until $Improvement = MaxNumberOfImprovements$

4.4.2 Adjust Probabilities

Overview

Since the program must be able to work for a variety of different datasets, it is important that the heuristics are called in different ways. This sub-procedure aims to adjust the probabilities based on the performance of the heuristics so far.

Method

There are four separate Improvement Heuristics, each of which are likely to have different success rates. The probability of each Heuristic also has a percentage change, Δ_A, Δ_B, Δ_C, Δ_D. These dictate how much the probabilities should be altered by. The values of these differ slightly depending on which Heuristic has just been run. The values which have been set for these Δ, have been tested and work for all datasets used in this project. For that reason, these values are not inputted by the user but are instead set within the code. The values of these were found by making an initial estimate, then making minor adjustments until the values were found that functioned the best. Also note, that a temporary change made to a probability, say $\mathrm{P}(A)$, is denoted by $\mathrm{P}'(A)$. These temporary probabilities are required so that invalid changes can be undone (for example a probability of less than 0.

Pseudocode

Cool Heuristic D:

$\Delta_D = 1/(MaxNumImprovements * 5)$
$\mathrm{P}(D) \leftarrow \mathrm{P}(D)(1 - \Delta_D)$

$\mathrm{P}(C) \leftarrow 1 - \mathrm{P}(A) - \mathrm{P}(B) - \mathrm{P}(D)$

if $\mathrm{P}(C) < 0 then$ **then**

  $\mathrm{P}(C) \leftarrow 0$

  $\mathrm{P}(B) \leftarrow 1 - \mathrm{P}(A) - \mathrm{P}(D)$

end if

If Heuristic A was run:

$\Delta_A \leftarrow 0.005$

if $HeuristicOkay = TRUE$ **then**

  $\mathrm{P}'(A) \leftarrow \mathrm{P}(A)(1 + \Delta_A)$

else

  $\mathrm{P}'(A) \leftarrow \mathrm{P}(A)(1 - \Delta_A)$

end if

$\mathrm{P}'(B) \leftarrow 1 - \mathrm{P}(A) - \mathrm{P}(C) - \mathrm{P}(D)$

if $0 < \mathrm{P}'(A) < 1$ and $0 < \mathrm{P}'(B) < 1$ **then**

  $\mathrm{P}(A) \leftarrow \mathrm{P}'(A)$

  $\mathrm{P}(B) \leftarrow \mathrm{P}'(B)$

end if

If Heuristic B was run:

$\Delta_A \leftarrow 0.005$

$\Delta_B \leftarrow 0.005$

if $HeuristicOkay = TRUE$ **then**

  $\mathrm{P}'(A) \leftarrow \mathrm{P}(A)(1 + \Delta_A)$

  $\mathrm{P}'(B) \leftarrow \mathrm{P}(B)(1 + \Delta_B)$

else

  $\mathrm{P}'(A) \leftarrow \mathrm{P}(A)(1 - \Delta_A)$

  $\mathrm{P}'(B) \leftarrow \mathrm{P}(B)(1 - \Delta_B)$

end if

$\mathrm{P}'(C) \leftarrow 1 - \mathrm{P}(A) - \mathrm{P}(B) - \mathrm{P}(D)$

if $0 < \mathrm{P}'(A) + \mathrm{P}'(B) < 1$ and $0 < \mathrm{P}'(C) < 1$ **then**

  $\mathrm{P}(A) \leftarrow \mathrm{P}(A)$

  $\mathrm{P}(B) \leftarrow \mathrm{P}(B)$

  $\mathrm{P}(C) \leftarrow \mathrm{P}(C)$

end if

If Heuristic C was run:

$\Delta_A \leftarrow 0.3$

$\Delta_B \leftarrow 0.1$

if HeuristicC success **then**

  $\mathrm{P}(A) \leftarrow \mathrm{P}(A)(1+\Delta_A)$

  $\mathrm{P}(B) \leftarrow \mathrm{P}(B)(1+\Delta_B)$

end if

$\mathrm{P}'(C) \leftarrow 1-\mathrm{P}(A)-\mathrm{P}(B)-\mathrm{P}(D)$

if $\mathrm{P}'(A)+\mathrm{P}'(B)<1$ and $0<\mathrm{P}'(C)$ **then**

  $\mathrm{P}(A) \leftarrow \mathrm{P}(A)$

  $\mathrm{P}(B) \leftarrow \mathrm{P}(B)$

  $\mathrm{P}(C) \leftarrow \mathrm{P}(C)$

end if

Chapter 5

Hyper-Heuristic Runtime Appraisal

When looking at various candidate solutions in Chapter 2, it was found that other mathematicians encountered problems due to exponential runtimes [18, p. 20]. Therefore, it is important that the runtimes of the Hyper-Heuristic Algorithm as a whole are appraised. Since Dataset 1 is the largest dataset, it is probable that for any given run size, the runtime will be longer for this dataset than for Datasets 2 or 3.

5.1 Observed Runtime

The Random Algorithm was selected to form initial solutions to Dataset 1, since it produces a large variety of different solutions.

Number of	Runtime, β (seconds)					
Improvements, α	Run 1	Run 2	Run 3	Run 4	Run 5	Mean
0	4.9	0.7	2.7	1.2	0.9	2.0
200	3.0	3.2	2.8	6.6	10.1	5.1
400	7.0	5.5	5.8	6.2	9.8	6.9
600	18.5	15.0	12.7	12.3	11.1	13.9
800	23.6	22.0	20.0	21.2	20.1	21.4
1000	25.1	28.7	36.7	21.2	24.6	27.2
1200	34.3	33.4	39.7	57.9	53.9	43.8

Table 5.1: Runtimes for the Hyper-Heuristic with the Random Formulation Algorithm

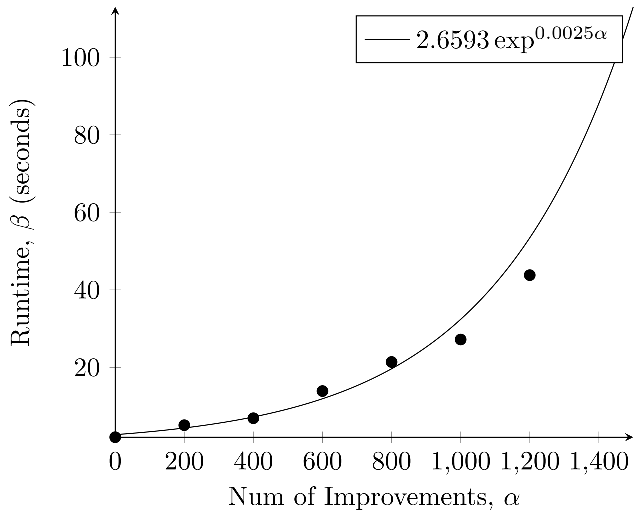

Figure 5.1: A Graph Showing of the Hyper-Heuristic Runtime Plots with an Exponential Line of Best Fit

Table 5.1 shows the runtimes for each of the five runs of the Hyper-Heuristic Algorithm and their means. The algorithm was run with the following probabilities of heuristics: $\mathrm{P}(HeuristicA) = 0.3, \mathrm{P}(HeuristicB) = 0.25, \mathrm{P}(HeuristicC) = 0.3, \mathrm{P}(HeuristicD) = 0.15$. The mean runtimes from Table 5.1 have been plotted in Figure 5.1.

From the graph in Figure 5.1 it can be seen that the line of best fit is exponential and is described by Equation 5.1 (with α denoting the number of improvements, i.e. the number of successful iterations, defined in Section 3.1.4 and β denoting the runtime).

$$\beta = 2.6593 \exp^{0.0025\alpha} \tag{5.1}$$

5.2 Expected Runtime

Using Equation 5.1 for the line of best fit, the expected runtimes have been calculated for a range of different numbers of improvements, which is shown in Table 5.2. The table shows that as the number of improvements α increases, the runtime increases exponentially. This exponential behaviour is largely because as the program progresses, each iteration is less likely to produce a successful result. In other words, it takes an increasing amount of time for an improvement to occur. This knowledge of the runtimes of the Hyper-Heuristic will affect the way in which the program's analysis is conducted, since it would be impractical to perform a large number of time-consuming runs.

Num of Improvements α	Expected Run-time β (seconds)
1000	32
2000	395
3000	4808
4000	58575
5000	713589

Table 5.2: Expected Runtimes for the Hyper-Heuristic Algorithm with a Random starting solution

Chapter 6

Formulation Method Appraisal

The Formulation Methods each produce initial solutions which then feed into the Improvement Heuristic Methods. Any evaluation of the Formulation Methods must be carried out prior to evaluation of the Improvement Heuristic Methods or the Hyper-Heuristic Algorithm as a whole. The fundamental behaviour of the Formulation Methods is assessed by looking at the frequency popularities, to see how many frequencies have been assigned to requests. This is followed by an appraisal of the order and cost of the solutions. Each of the Formulation Methods have been run against all datasets, but detailed analysis has been focused on the largest dataset, Dataset 1, with full solutions given in the appendix. Comments have also been made to confirm whether or not the observed behaviour is also true for the remaining datasets.

6.1 Random Algorithm

6.1.1 Frequency Popularity

The Random Algorithm was run with Dataset 1 and the solution it produced has been shown in full in Appendix B.1. The graph in Figure 6.1 shows the popularity of each of the frequencies in that solution. In other words, the graph shows how many requests used each frequency. The popularities are all within the range 11-30, so it is appears as though the assignments are randomly spread, as expected. The algorithm performs similarly with Datasets 2 and 3. In Dataset 2 the popularities are well-spread with values from 0 to 11, and in Dataset 3 the spread is far greater, with popularities from 3 to 28.

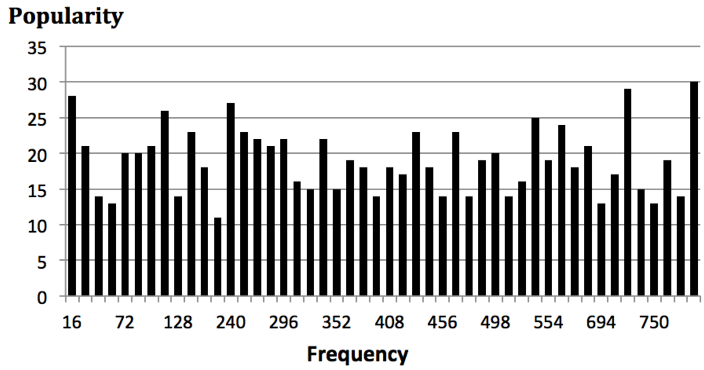

Figure 6.1: The Frequency Popularities of a Random Algorithm Solution to Dataset 1

6.1.2 Cost & Order

The Random Algorithm was now run five times in Dataset 1, each time producing a different solution. Table 6.1 shows the number of frequencies used in each of the solutions. Every single one of the 48 available frequencies were assigned to at least one request, in each of the five solutions produced. However since the frequency assignments are all done randomly, the solutions are likely to all be different. This variety of solutions may be useful when wanting to compare different heuristics later on.

Run	Order	Cost
1	48	1.31E-02
2	48	1.31E-02
3	48	8.74E-03
4	48	1.09E-02
5	48	1.09E-02
μ	48	1.13E-02

Table 6.1: Random Algorithm solutions to Dataset 1

Despite the fact that all the solutions shown in Table 6.1 have used the same number of frequencies, the cost function will show small differences in the solution. (This cost function has been discussed in greater depth in Section 3.2.3). The table shows that the third run has a slightly lower cost, which implies that compared to the other solutions, this would require fewer changes in order to reduce the order of the solution. Overall, because of the large number of requests compared to the number of frequencies (916 and 48 respectively), the Random Algorithm used all available frequencies, hence the order is 48. The algorithm produced a solution with an order of 47 for Dataset 2, which is within reason since Dataset 2 has only 197 requests to be assigned any of the 48 frequencies. Dataset 3 on the other hand, has 680 requests, which explains why the Random Algorithm produced a solution with an order of 48.

6.2 Ascending Algorithm

6.2.1 Frequency Popularity

With the start frequency set to the default (i.e. lowest frequency) of 16, the distribution of frequencies is much more skewed. Figure 6.2 shows that the program has assigned the frequencies at the lower end of the list much more than it has the higher frequencies. As Appendix B.2 shows, the frequencies are selected in order, starting from 16, hence why the lower frequencies are assigned more. For comparison Figure 6.3 shows the same algorithm but with a higher start frequency of 100. This is means that the algorithm assigns frequencies in the order of 100, 114, 128, ... 778, 792, 16, 30 etc. In other words, it assigns the requests with frequencies that have been selected in ascending order, starting from 100. As expected, the popularities follow a similar shape, to Figure 6.2, except shifted so that it starts at the frequency of 100.

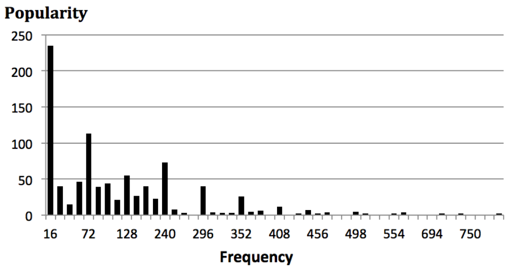

Figure 6.2: Bar Graph showing the Frequency Popularities of an Ascending Algorithm solution to Dataset 1. Start Freq=16

Both Figures 6.2 and 6.3 show certain frequencies with substantially taller bars than their neighbours. This spread of frequencies with larger popularities is caused by constraints with large distance values. In other words, if two requests share a constraint which says that their frequencies must have a difference of at least 40, and one of the frequencies has already been assigned to a popular frequency like 16, then the second request will have to take the first frequency that is higher than 56. This process is repeated, so every request with a similarly large distance constraint will also be assigned to that same frequency. As the algorithm progresses, these frequencies increase in popularity, hence they form substantially larger bars in the graph than their neighbours. The way in which the constraints force the solution to use more spaced values, is exploited in the Spaced Algorithm, which is appraised in Section 6.3.

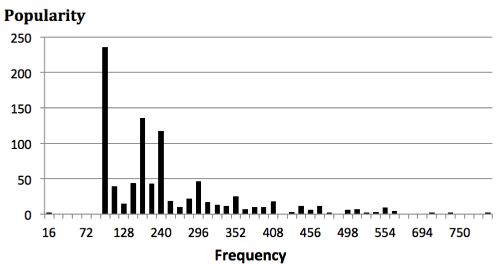

Figure 6.3: Bar Graph showing the Frequency Popularities of an Ascending Algorithm solution to Dataset 1. Start Freq=100

The Ascending Algorithm produced a similar distribution of frequencies in the other datasets. When run with Dataset 2, the algorithm produced a solution with popularities ranging from 0 to 99 with a start frequency of 16. Purely coincidentally, the same range is produced when a start frequency of 100 is used. Dataset 3 had a solution with popularities ranging from 0 to 160. When the start frequency was increased to 100, the popularities ranged from 0 to 140. Both sets of solutions followed the same general distribution as Dataset 1, shown in Figures 6.2 & 6.3, with popularities decreasing as the frequency increases.

6.2.2 Cost & Order

The Ascending Algorithm was run a number of times, each run using a different start frequency. Since there are no random elements in the algorithm, there is no need to do multiple runs with the same start frequency parameter, since they will all produce the same solution.

As shown when the algorithm was first explained in Section 4.2.2, wherever the algorithm fails for a given start frequency, the algorithm will automatically attempt using the next frequency to start with. This is shown in Table 6.2, where runs 2-6 have different chosen start frequencies but since they all failed, the algorithm produced identical solutions to run 7, with the an actual start frequency of 100.

When run with Dataset 2, the Ascending Algorithm, with a start frequency of 16, produced a solution with an orders of 21. The solution to Dataset 3 was of order 27. Table 6.2 shows that the mean order of all possible Ascending Algorithm solutions to Dataset 1 is just over 34. In summary, it is clear that for all three datasets, the Ascending Algorithm produces solutions of a significantly lower order than the Random Algorithm.

Run	Chosen Start Freq	Actual Start Freq	Order	Cost
1	16	16	35	3.42E-42
2	30	100	36	3.13E-39
3	44	100	36	3.13E-39
⋮	⋮	⋮	⋮	⋮
6	86	100	36	3.13E-39
7	100	100	36	3.13E-39
8	114	114	34	7.47E-45
9	128	128	36	3.13E-39
10	142	142	35	3.42E-42
11	156	156	31	9.71E-54
12	170	792	34	3.73E-45
13	240	792	34	3.73E-45
⋮	⋮	⋮	⋮	⋮
48	792	792	34	3.73E-45
μ			34.27	4.57E-40

Table 6.2: Ascending Algorithm solutions to Dataset 1

6.3 Spaced Algorithm

6.3.1 Assignment

The Spaced Algorithm first attempts to assign requests with pre-selected frequencies, which have been selected because they are separated by the chosen spacing distance. These pre-selected frequencies go at the front of the ordered frequencies, and the remaining frequencies follow. Appendix B.3 shows the ordered list of frequencies and the solution given when the Spaced Algorithm was run with a spacing distance of 55. The solution has been graphed in Figure 6.4, which clearly illustrates the reason that the Spaced Algorithm is expected to be successful. This pattern is also seen when the Spaced Algorithm is run with datasets 2 and 3. Their solutions only use a small number of frequencies, which are all spread out over a similar range.

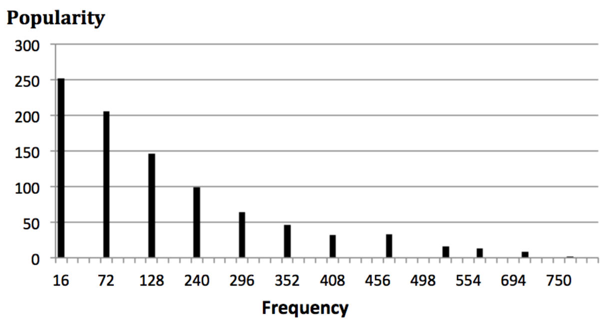

Figure 6.4: Bar Graph showing the Frequency Popularities of a Spaced Algorithm Solution to Dataset 1, with Spacing Dist=55.

6.3.2 Cost & Order

Table 6.3 shows the order (number of frequencies used) and the cost of different runs of the Spaced Algorithm on Dataset 1. The spacing distance defines the minimum distance between pairs of frequencies in the list which we want to assign from.

It can be seen that the algorithm produces solutions with the minimal order where the spacing distance is around 50-55. The reason for this can be understood by looking at the spaced frequencies that were discussed earlier, in 4.2.3. In Dataset 1, with the spacing distance set to 50, the spaced frequencies are $x_1 = 16$, $x_5 = 72$, $x_9 = 128$, $x_{13} = 240$, $x_{17} = 296$, $x_{21} = 352$, $x_{25} = 408$, $x_{30} = 470$, $x_{35} = 526$, $x_{38} = 652$, $x_{42} = 708$, $x_{46} = 764$ which are all followed by the remaining frequencies, which are labelled 'un-spaced'.

In Section 1.6, it was seen that a study of maximal cliques can find the global minimum for the order. Applying that to this dataset, it can be found that the global minimum is 12, which is also the number of spaced frequencies that are used with a spacing distance of 50-55.

Datasets 2 & 3 have a different number of requests and constraints to Dataset 1, however they all have exactly the same frequencies. As a result, a spacing distance of 55 (for example) would give the same spaced frequencies for all these datasets. The solutions produced by the Spaced Algorithm on datasets 2 and 3 are of a far lower cost and order than any of the other algorithms managed. The solutions for both datasets is of the order 10. It can be concluded, that the Spaced Algorithm has been the most successful algorithm in producing low-order initial solutions to the Minimum-Order Frequency Assignment Problem.

Run	Spacing Dist	Order	Cost
1	20	21	2.34E-83
2	25	22	2.14E-80
3	30	17	3.32E-95
4	35	26	7.54E-69
5	40	26	7.54E-69
6	45	16	1.81E-98
7	50	12	5.16E-110
8	55	12	5.16E-110
9	60	13	2.36E-107
10	65	13	2.36E-107
11	70	13	2.36E-107
12	75	31	4.86E-54
13	80	29	5.79E-60
14	85	27	6.91E-66
15	90	27	6.91E-66
16	95	29	5.79E-60
17	100	31	4.86E-54
18	105	22	1.07E-80
19	110	22	1.07E-80
20	115	31	4.86E-54
21	120	31	4.86E-54
μ		22.43	9.25E-55

Table 6.3: 21 runs of Spaced Algorithm on Dataset1

Chapter 7

Hyper-Heuristic Appraisal

To assess the performance of the Hyper-Heuristic Method, a variety of different initial solutions must be used, each of which must not have been pre-optimised in any way. The Random Algorithm was selected, since it produces random starting solutions every time, each of which is likely to use many of the available frequencies.

7.1 Assignment

The solution produced by the Random Algorithm was run through the Hyper-Heuristic with 10,000 inputted as the number of improvements to attempt. The full solution is shown in Appendix B.4. Figure 7.1 shows the frequency popularities of the initial solution which was produced by the Random Algorithm. The distribution of frequencies in the Random Algorithm has been discussed in Section 6.1.1, and here the distribution appears to follow that same pattern.

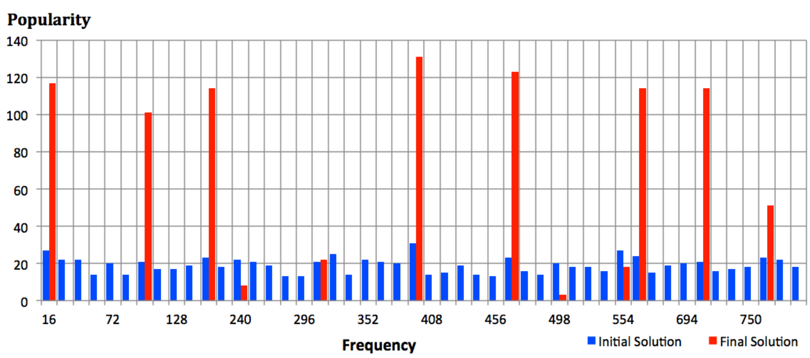

Figure 7.1: Bar Graph showing the Frequency Popularities of the Solutions before and after the Hyper-Heuristic

Also shown in Figure 7.1, is the solution produced after the initial solution was run through the Hyper-Heuristic Improvement Algorithm. The bars appear well-spaced, much like the Spaced Algorithm in Figure 6.4. However, it can be seen that the difference between frequencies assigned in the Hyper-Heuristic is not as equal here as it was in the Spaced Algorithm. In other words, the Spaced Algorithm had much more even spacing between bars in the graph, compared to the final solution shown in Figure 7.1.

7.2 Cost & Order

All improvements are conducted by comparing the cost of the solutions before and after an iteration has taken place. Figure 7.2 shows how the cost has changed as the algorithm progressed. It is clear to see the fluctuations in cost, where Heuristic D has forced a worse cost solution, to prevent the algorithm from settling on local minima. Figure 7.3 shows how the order of the solution changed as the algorithm progressed, and it is clear that the cost function is working as required, since the order is decreasing and eventually settles on the lower bound of 12.

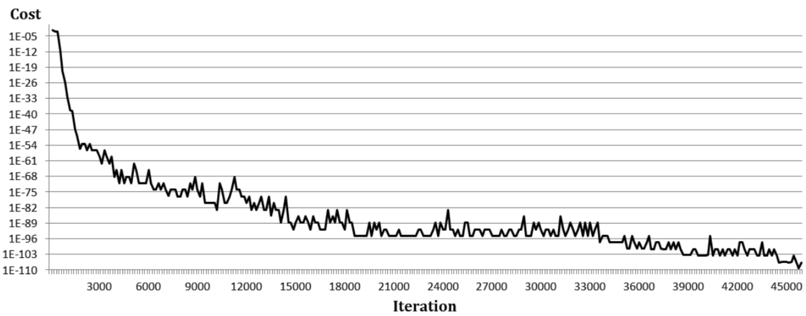

Figure 7.2: Graph Showing the Change in the Cost of the Solutions

This lower bound of 12 was given in the clique information that was provided with Dataset 1. The Hyper-Heuristic Method is also able to reach this value, even when starting with a random solution. Figure 7.3 shows how the order of the solution decreases as more iterations of the hyper-heuristic are performed. Note how the order decreases rapidly earlier in the run, but the algorithm struggles later on, as the solution approaches the lower bound.

Importantly, the Hyper-Heuristic Method has also been able to reach the lower bound by starting with an Ascending Algorithm solution as well as the Spaced Algorithm, with different parameters for each.

When run with datasets 2 and 3, the Hyper-Heuristic Method produced solutions of order 9 and 10 respectively. After multiple runs it looked clear that those were the

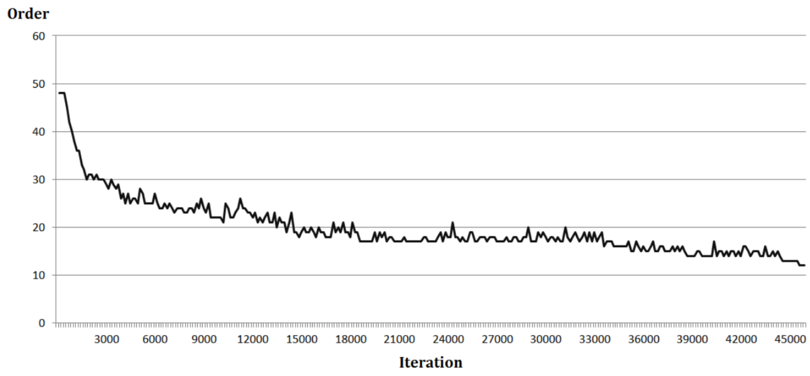

Figure 7.3: Graph Showing the Change in the Order of the Solutions

lower bound for their datasets.

7.3 Adjusting the Heuristic Method Probability

The Improvement Heuristic Methods are each run with certain probabilities, as shown previously in Figure 4.1 on page 28. Initial probabilities are inputted to the system, but these probabilities changes as the Hyper-Heuristic algorithm progresses. The adjustment of the heuristic probabilities was defined in detail in Section 4.4.2.

The hyper-heuristic was run with the following probabilities: $\text{P}(HeuristicA) = 0.5$, $\text{P}(HeuristicB) = 0.25$, $\text{P}(HeuristicC) = 0.1$, $\text{P}(HeuristicD) = 0.15$. Figure 7.4 shows each of the heuristics changing in probability. In particular, note how the probabilities change greatly between the initial solution at $iteration = 0$ and after 3000 iterations. Also notice how the probability of $HeuristicD$ decreases throughout, as it is expected to do. (Heuristic D is the worse cost solution).

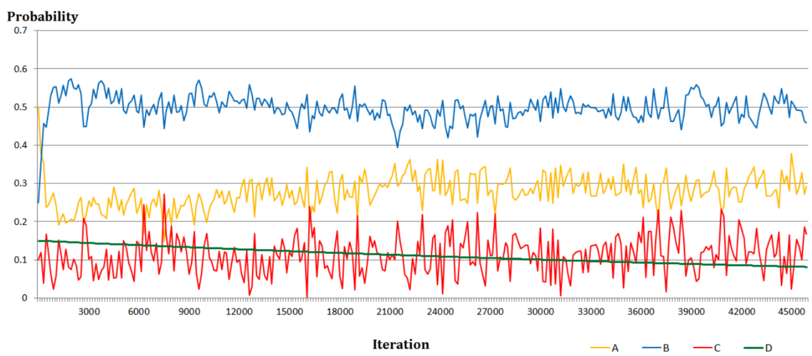

Figure 7.4: Graph Showing the Probability of each Heuristic being Selected

After running the Hyper-Heuristic Method with the two remaining datasets, similarities could be seen in the way the heuristics settle within a range of probabilities.

The *Adjust Probabilities* sub-procedure led to heuristic B often settling on a probability of around 0.5, often with a brief period of almost 0.6 during the first few thousand iterations. Heuristic A settles at around 0.3 and heuristic C at around 0.15. This shows that the code defined in Section 4.4.2 is working successfully, since it is changing the probabilities of selecting heuristics based on their performance, whilst trying to keep the probability of the strongest heuristic (Heuristic A) high wherever possible.

7.4 Heuristic Method Success

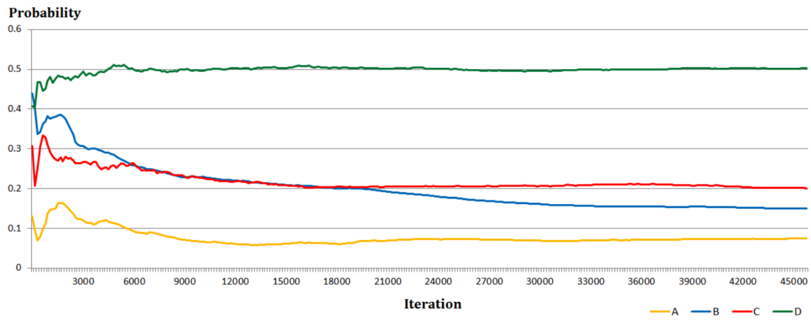

Figure 7.5: Graph Showing the Observed Success Probability of each Heuristic

Figure 7.5 shows the probability of each heuristic being successful (i.e. producing a solution which does not break any constraints). The highest performing heuristic is the 'worse cost' heuristic, Heuristic D, which plateaus at a probability of 0.5, meaning that half of the runs are successful and don't break any constraints. All datasets produce similar graphs, with the success of heuristics A and B dropping as soon as the program settles on the lower bound.

Chapter 8

Conclusion and Further Work

8.1 Limitations

The datasets given were all minimum-order frequency assignment problems, in that a solution existed, and an optimised solution needed to be found which minimised the number of solutions used. This project does not set out to minimise the span (i.e. difference between lowest and highest frequencies used), so cannot find solutions to the minimum-span frequency assignment problems. Fortunately, all the datasets were fairly easy to work with, in that solutions existed to all of them. If we were presented with a dataset for which a full solution does not exist, we would require the best possible solution, i.e. one which assigns frequencies to as many requests as possible. This called the Maximum-Service FAP, which this project does not work with.

8.2 Conclusion

The Spaced Algorithm was able to reach an initial solution of order 12 for Dataset 1, which is the lower bound. It achieved this in a far quicker time than the hyper-heuristic was able to achieve a final solution of order 12. At first glance, it is tempting to say that the Spaced Algorithm performs better than the hyper-heuristic. However, knowing the way in which both algorithms work, it can be noted that in general, the hyper-heuristic is far more likely to be produce lower cost solutions that the Spaced Algorithm. The Spaced Algorithm has performed well with the given datasets, however it is very difficult that it will do the same for any dataset. On the other hand, the hyper-heuristic is designed in such a way that it always makes intelligent decisions to adjust the route it takes, to reach a global minimum. Furthermore, in more complicated datasets it is likely that the Hyper-Heuristic algorithm will produce excellent results much faster by using the Spaced Algorithm to form the initial solution.

8.3 Further Work

Given a longer period of time, the project could have been greatly improved. One key alteration would have been to have a preliminary algorithm which assessed the dataset for the number of constraints, number of frequencies and number of requests, and use this information to conduct an intelligent selection of parameters which at present are either user-inputted or preset within the code. This will allow the algorithm to reach the global minimum much more quickly, for a wider variety of solutions. Another improvement would be to have a more in-depth study of the lower bound to each problem. If successful, it could allow the program to run to stop automatically, if the lower bound is reached.

BLANK

Appendix A

Datasets

A.1 Dataset 1

Number of constraints: 5090 (Constraints not listed to save space)

Number of requests: 916

Number of available frequencies: 48

Lower bound for number of frequencies used: 12

List of Available Frequencies: 16, 30, 44, 58, 72, 86, 100, 114, 128, 142, 156, 170, 240, 254, 268, 282, 296, 310, 324, 338, 352, 366, 380, 394, 408, 414, 428, 442, 456, 470, 478, 484, 498, 512, 526, 540, 554, 652, 666, 680, 694, 708, 722, 736, 750, 764, 778, 792

A.2 Dataset 2

Number of constraints: 1033 (Constraints not listed to save space)

Number of requests: 198

Number of available frequencies: 48

Lower bound for number of frequencies used: Not known

List of Available Frequencies: 16, 30, 44, 58, 72, 86, 100, 114, 128, 142, 156, 170, 240, 254, 268, 282, 296, 310, 324, 338, 352, 366, 380, 394, 408, 414, 428, 442, 456, 470, 478, 484, 498, 512, 526, 540, 554, 652, 666, 680, 694, 708, 722, 736, 750, 764, 778, 792.

A.3 Dataset 3

Number of constraints: 6417 (Constraints not listed to save space)

Number of requests: 680

Number of available frequencies: 48

Lower bound for number of frequencies used: Not known

List of Available Frequencies: 16, 30, 44, 58, 72, 86, 100, 114, 128, 142, 156, 170, 240, 254, 268, 282, 296, 310, 324, 338, 352, 366, 380, 394, 408, 414, 428, 442, 456, 470, 478, 484, 498, 512, 526, 540, 554, 652, 666, 680, 694, 708, 722, 736, 750, 764, 778, 792.

Appendix B

Solutions to Dataset 1

Each solution set occupies a large amount of printed space. For this reason, only one solution has been shown for each of the formulation methods alongside a solution from the Hyper-Heuristic method.

All the following solutions are also available on an Excel spreadsheet on the accompanying CD. They can all be found on the file marked 'AppendixB-SolutionsToDataset1'. All results have been verified using the 'ConstraintChecker' excel file, also found on the accompanying CD.

B.1 Random Algorithm

Frequency List: The frequencies were selected randomly from the following values: 16, 30, 44, 58, 72, 86, 100, 114, 128, 142, 156, 170, 240, 254, 268, 282, 296, 310, 324, 338, 352, 366, 380, 394, 408, 414, 428, 442, 456, 470, 478, 484, 498, 512, 526, 540, 554, 652, 666, 680, 694, 708, 722, 736, 750, 764, 778, 792.

Cost: 1.20242E-02. **Order (Number of Frequencies Used):** 48

Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i
1	380	41	254	81	100	121	722	161	114	201	254	241	652	281	338
2	442	42	484	82	554	122	778	162	114	202	16	242	442	282	540
3	170	43	652	83	324	123	428	163	114	203	652	243	100	283	16
4	666	44	58	84	484	124	470	164	778	204	680	244	414	284	736
5	58	45	428	85	708	125	114	165	30	205	240	245	512	285	366
6	240	46	750	86	44	126	540	166	366	206	394	246	310	286	240
7	736	47	764	87	792	127	792	167	254	207	484	247	414	287	764
8	554	48	540	88	470	128	338	168	484	208	142	248	694	288	100
9	428	49	456	89	478	129	100	169	680	209	666	249	428	289	694
10	484	50	268	90	268	130	540	170	722	210	44	250	722	290	100
11	268	51	540	91	652	131	428	171	498	211	156	251	30	291	708
12	652	52	652	92	694	132	352	172	72	212	792	252	792	292	408
13	240	53	352	93	296	133	254	173	44	213	240	253	442	293	16
14	324	54	792	94	114	134	142	174	366	214	680	254	30	294	254
15	72	55	764	95	414	135	394	175	512	215	16	255	722	295	268
16	498	56	44	96	484	136	30	176	100	216	666	256	16	296	114
17	708	57	722	97	338	137	338	177	722	217	428	257	456	297	442
18	764	58	778	98	366	138	540	178	268	218	296	258	352	298	282
19	394	59	86	99	366	139	142	179	282	219	380	259	470	299	282
20	324	60	30	100	722	140	652	180	750	220	408	260	470	300	694
21	282	61	352	101	414	141	240	181	526	221	296	261	442	301	414
22	30	62	456	102	666	142	394	182	428	222	296	262	512	302	128
23	86	63	58	103	428	143	268	183	86	223	478	263	408	303	498
24	254	64	310	104	16	144	778	184	58	224	708	264	764	304	666
25	456	65	268	105	114	145	254	185	722	225	16	265	652	305	338
26	470	66	240	106	114	146	72	186	428	226	414	266	540	306	16
27	366	67	428	107	778	147	156	187	498	227	114	267	736	307	484
28	170	68	694	108	526	148	498	188	498	228	338	268	58	308	680
29	86	69	324	109	498	149	44	189	428	229	128	269	86	309	428
30	114	70	338	110	338	150	736	190	414	230	352	270	100	310	254
31	554	71	680	111	470	151	338	191	86	231	114	271	498	311	142
32	540	72	540	112	170	152	484	192	428	232	792	272	296	312	792
33	380	73	114	113	366	153	540	193	282	233	142	273	666	313	394
34	540	74	268	114	156	154	722	194	764	234	296	274	652	314	554
35	708	75	694	115	456	155	540	195	478	235	498	275	512	315	792
36	764	76	540	116	380	156	296	196	554	236	652	276	156	316	722
37	324	77	240	117	310	157	708	197	44	237	722	277	428	317	394
38	296	78	792	118	736	158	394	198	282	238	282	278	142	318	778
39	128	79	114	119	380	159	366	199	16	239	30	279	540	319	268
40	680	80	30	120	380	160	652	200	470	240	114	280	338	320	142

Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i
321	100	361	540	401	128	441	498	481	778	521	526	561	16	601	380
322	778	362	428	402	764	442	72	482	694	522	16	562	352	602	352
323	170	363	100	403	366	443	680	483	680	523	240	563	498	603	254
324	764	364	296	404	408	444	100	484	114	524	736	564	526	604	352
325	478	365	722	405	16	445	310	485	268	525	526	565	526	605	408
326	100	366	778	406	324	446	128	486	778	526	100	566	652	606	366
327	414	367	554	407	86	447	554	487	680	527	680	567	128	607	408
328	268	368	240	408	58	448	470	488	470	528	456	568	114	608	240
329	428	369	268	409	540	449	680	489	694	529	512	569	750	609	72
330	240	370	498	410	310	450	792	490	366	530	254	570	736	610	240
331	282	371	296	411	764	451	442	491	100	531	142	571	310	611	296
332	100	372	324	412	414	452	408	492	16	532	456	572	680	612	296
333	484	373	470	413	666	453	282	493	540	533	156	573	540	613	170
334	652	374	282	414	16	454	30	494	498	534	156	574	414	614	254
335	114	375	680	415	554	455	442	495	156	535	86	575	58	615	380
336	324	376	380	416	72	456	282	496	240	536	240	576	456	616	470
337	254	377	240	417	282	457	498	497	554	537	100	577	708	617	394
338	498	378	666	418	16	458	526	498	478	538	86	578	86	618	778
339	128	379	554	419	142	459	324	499	254	539	722	579	44	619	792
340	268	380	498	420	380	460	380	500	408	540	324	580	16	620	16
341	666	381	338	421	394	461	792	501	408	541	484	581	114	621	792
342	484	382	666	422	86	462	666	502	72	542	352	582	310	622	526
343	722	383	408	423	170	463	128	503	470	543	296	583	366	623	680
344	478	384	484	424	254	464	722	504	44	544	324	584	470	624	736
345	442	385	394	425	478	465	86	505	310	545	792	585	58	625	170
346	764	386	100	426	30	466	324	506	792	546	324	586	296	626	394
347	352	387	310	427	666	467	428	507	722	547	240	587	254	627	72
348	72	388	44	428	338	468	366	508	16	548	526	588	100	628	442
349	652	389	72	429	708	469	282	509	338	549	414	589	554	629	30
350	394	390	16	430	456	470	296	510	456	550	792	590	498	630	694
351	708	391	282	431	708	471	680	511	694	551	142	591	778	631	792
352	128	392	86	432	526	472	296	512	310	552	240	592	366	632	736
353	240	393	478	433	722	473	666	513	268	553	680	593	156	633	142
354	428	394	30	434	156	474	540	514	512	554	380	594	240	634	512
355	456	395	750	435	338	475	512	515	142	555	310	595	442	635	456
356	540	396	128	436	338	476	442	516	72	556	792	596	792	636	72
357	764	397	296	437	100	477	114	517	554	557	16	597	44	637	722
358	142	398	540	438	408	478	736	518	128	558	694	598	652	638	254
359	792	399	428	439	470	479	652	519	394	559	512	599	380	639	414
360	694	400	282	440	16	480	792	520	428	560	470	600	352	640	366

Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i
641	310	681	722	721	282	761	792	801	470	841	142	881	478
642	680	682	352	722	86	762	652	802	30	842	652	882	484
643	240	683	282	723	408	763	30	803	708	843	296	883	442
644	736	684	16	724	708	764	470	804	652	844	540	884	554
645	764	685	30	725	268	765	792	805	442	845	352	885	16
646	680	686	142	726	652	766	268	806	72	846	44	886	408
647	156	687	666	727	268	767	268	807	554	847	526	887	736
648	366	688	526	728	16	768	540	808	296	848	100	888	282
649	30	689	736	729	72	769	708	809	750	849	456	889	156
650	478	690	750	730	478	770	254	810	680	850	282	890	792
651	652	691	666	731	170	771	750	811	170	851	456	891	652
652	470	692	156	732	338	772	414	812	540	852	428	892	498
653	484	693	30	733	750	773	408	813	338	853	268	893	44
654	142	694	694	734	100	774	722	814	484	854	736	894	512
655	16	695	310	735	338	775	666	815	114	855	722	895	86
656	352	696	680	736	324	776	792	816	72	856	666	896	526
657	86	697	478	737	526	777	254	817	254	857	470	897	114
658	408	698	86	738	722	778	750	818	394	858	310	898	540
659	58	699	428	739	282	779	792	819	114	859	58	899	240
660	442	700	470	740	380	780	526	820	414	860	142	900	16
661	526	701	366	741	114	781	240	821	338	861	512	901	44
662	470	702	310	742	72	782	666	822	484	862	764	902	142
663	722	703	764	743	380	783	72	823	792	863	254	903	380
664	310	704	142	744	156	784	156	824	414	864	170	904	128
665	470	705	554	745	764	785	58	825	72	865	240	905	408
666	86	706	442	746	708	786	380	826	652	866	708	906	254
667	30	707	268	747	652	787	680	827	142	867	338	907	268
668	254	708	408	748	408	788	442	828	366	868	338	908	86
669	282	709	792	749	58	789	156	829	44	869	764	909	478
670	114	710	484	750	128	790	708	830	442	870	722	910	296
671	554	711	380	751	114	791	324	831	72	871	722	911	554
672	722	712	142	752	764	792	16	832	170	872	156	912	498
673	100	713	442	753	338	793	484	833	478	873	296	913	58
674	268	714	142	754	512	794	778	834	86	874	72	914	366
675	722	715	414	755	414	795	352	835	30	875	114	915	554
676	778	716	750	756	30	796	428	836	750	876	750	916	750
677	30	717	554	757	722	797	792	837	296	877	128		
678	142	718	156	758	498	798	722	838	16	878	764		
679	240	719	736	759	792	799	512	839	470	879	484		
680	254	720	240	760	708	800	156	840	512	880	240		

B.2 Ascending Algorithm

Frequency List: The frequencies were selected in ascending order with a start frequency of 16. Frequencies at the start of the list being much more likely to be selected than those towards the end. These are the frequencies: 16, 30, 44, 58, 72, 86, 100, 114, 128, 142, 156, 170, 240, 254, 268, 282, 296, 310, 324, 338, 352, 366, 380, 394, 408, 414, 428, 442, 456, 470, 478, 484, 498, 512, 526, 540, 554, 652, 666, 680, 694, 708, 722, 736, 750, 764, 778, 792.

Cost: 3.42304E-42. **Order (Number of Frequencies Used):** 35

Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i
1	16	41	128	81	72	121	114	161	100	201	296	241	156	281	156
2	16	42	142	82	72	122	156	162	100	202	296	242	156	282	156
3	16	43	240	83	16	123	240	163	16	203	296	243	240	283	240
4	16	44	240	84	16	124	30	164	16	204	296	244	240	284	240
5	16	45	366	85	100	125	240	165	100	205	30	245	296	285	72
6	16	46	366	86	100	126	240	166	100	206	30	246	296	286	72
7	16	47	128	87	16	127	16	167	240	207	114	247	352	287	16
8	16	48	128	88	16	128	16	168	240	208	268	248	352	288	16
9	100	49	16	89	72	129	100	169	296	209	142	249	442	289	16
10	100	50	16	90	58	130	100	170	296	210	142	250	442	290	16
11	16	51	58	91	16	131	30	171	156	211	240	251	310	291	156
12	16	52	72	92	16	132	16	172	156	212	240	252	310	292	156
13	16	53	114	93	58	133	114	173	16	213	296	253	380	293	16
14	16	54	100	94	72	134	100	174	16	214	296	254	380	294	16
15	100	55	114	95	114	135	16	175	16	215	16	255	16	295	86
16	100	56	128	96	100	136	16	176	16	216	16	256	16	296	86
17	16	57	58	97	142	137	72	177	58	217	58	257	442	297	30
18	44	58	58	98	156	138	72	178	58	218	58	258	442	298	16
19	72	59	100	99	170	139	156	179	44	219	16	259	16	299	86
20	72	60	100	100	156	140	156	180	44	220	16	260	16	300	86
21	16	61	142	101	58	141	254	181	44	221	16	261	16	301	310
22	16	62	142	102	86	142	254	182	30	222	16	262	16	302	352
23	16	63	30	103	16	143	16	183	114	223	58	263	30	303	142
24	16	64	16	104	16	144	16	184	114	224	58	264	30	304	142
25	16	65	100	105	58	145	100	185	16	225	16	265	16	305	240
26	16	66	86	106	58	146	100	186	16	226	16	266	16	306	240
27	16	67	16	107	240	147	16	187	86	227	100	267	100	307	16
28	16	68	16	108	240	148	16	188	86	228	100	268	100	308	16
29	16	69	100	109	58	149	100	189	72	229	30	269	16	309	296
30	16	70	100	110	72	150	100	190	44	230	16	270	16	310	296
31	72	71	16	111	16	151	30	191	16	231	30	271	100	311	142
32	72	72	16	112	16	152	30	192	16	232	44	272	100	312	142
33	72	73	16	113	16	153	86	193	100	233	16	273	156	313	240
34	72	74	16	114	30	154	86	194	100	234	16	274	156	314	240
35	240	75	72	115	16	155	16	195	16	235	72	275	240	315	16
36	240	76	58	116	16	156	16	196	16	236	72	276	240	316	30
37	142	77	114	117	16	157	100	197	72	237	16	277	296	317	72
38	128	78	16	118	16	158	100	198	72	238	16	278	296	318	72
39	16	79	16	119	72	159	16	199	240	239	428	279	16	319	156
40	16	80	16	120	72	160	16	200	30	240	428	280	16	320	156

Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i
321	156	361	72	401	128	441	30	481	240	521	170	561	128	601	240
322	156	362	72	402	128	442	16	482	240	522	128	562	128	602	240
323	240	363	156	403	16	443	58	483	16	523	16	563	240	603	240
324	240	364	142	404	16	444	58	484	30	524	16	564	240	604	240
325	86	365	240	405	296	445	16	485	58	525	72	565	16	605	240
326	86	366	240	406	296	446	30	486	58	526	72	566	30	606	72
327	170	367	296	407	16	447	58	487	58	527	128	567	16	607	16
328	170	368	296	408	16	448	72	488	58	528	128	568	16	608	30
329	16	369	170	409	16	449	170	489	72	529	16	569	72	609	16
330	16	370	170	410	16	450	240	490	72	530	16	570	72	610	16
331	156	371	16	411	72	451	86	491	72	531	58	571	16	611	72
332	156	372	16	412	72	452	58	492	72	532	72	572	16	612	58
333	240	373	16	413	72	453	16	493	128	533	44	573	128	613	156
334	240	374	16	414	72	454	16	494	128	534	44	574	128	614	156
335	72	375	240	415	72	455	240	495	58	535	86	575	16	615	58
336	72	376	240	416	72	456	240	496	58	536	86	576	16	616	58
337	72	377	240	417	16	457	16	497	338	537	114	577	170	617	128
338	72	378	240	418	16	458	16	498	352	538	128	578	240	618	114
339	156	379	30	419	128	459	72	499	16	539	156	579	16	619	296
340	156	380	16	420	128	460	72	500	16	540	170	580	16	620	296
341	352	381	72	421	16	461	128	501	408	541	58	581	16	621	310
342	352	382	72	422	16	462	128	502	380	542	72	582	16	622	352
343	442	383	30	423	58	463	16	503	296	543	170	583	408	623	30
344	442	384	30	424	58	464	16	504	296	544	142	584	408	624	58
345	240	385	86	425	86	465	72	505	470	545	16	585	58	625	114
346	240	386	72	426	72	466	72	506	442	546	16	586	86	626	86
347	498	387	86	427	100	467	128	507	86	547	128	587	100	627	114
348	498	388	86	428	100	468	128	508	72	548	142	588	128	628	128
349	72	389	86	429	86	469	58	509	142	549	240	589	16	629	30
350	72	390	72	430	86	470	72	510	128	550	240	590	16	630	30
351	128	391	128	431	128	471	30	511	352	551	170	591	296	631	86
352	128	392	128	432	114	472	30	512	352	552	296	592	254	632	86
353	296	393	72	433	142	473	240	513	240	553	352	593	16	633	72
354	296	394	72	434	142	474	240	514	240	554	338	594	16	634	72
355	30	395	128	435	30	475	142	515	128	555	352	595	58	635	296
356	44	396	142	436	30	476	156	516	100	556	352	596	72	636	296
357	16	397	16	437	296	477	352	517	352	557	296	597	72	637	16
358	16	398	16	438	296	478	352	518	352	558	296	598	72	638	16
359	86	399	58	439	86	479	352	519	16	559	156	599	352	639	128
360	100	400	86	440	142	480	352	520	16	560	156	600	352	640	128

Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i
641	296	681	72	721	708	761	16	801	380	841	72	881	240
642	296	682	72	722	708	762	16	802	408	842	72	882	16
643	352	683	16	723	792	763	352	803	170	843	16	883	16
644	352	684	16	724	792	764	352	804	72	844	128	884	16
645	156	685	72	725	296	765	58	805	16	845	72	885	72
646	30	686	72	726	296	766	240	806	16	846	72	886	72
647	72	687	16	727	380	767	470	807	72	847	142	887	128
648	72	688	16	728	366	768	470	808	72	848	240	888	128
649	16	689	72	729	408	769	16	809	16	849	156	889	72
650	16	690	72	730	408	770	16	810	16	850	254	890	72
651	156	691	16	731	456	771	142	811	128	851	170	891	128
652	142	692	16	732	456	772	128	812	128	852	44	892	128
653	142	693	72	733	170	773	16	813	58	853	16	893	16
654	156	694	72	734	240	774	16	814	72	854	100	894	16
655	324	695	44	735	44	775	72	815	114	855	86	895	114
656	282	696	44	736	30	776	72	816	16	856	156	896	16
657	338	697	86	737	86	777	240	817	296	857	72	897	240
658	170	698	86	738	100	778	366	818	72	858	30	898	240
659	498	699	16	739	128	779	652	819	254	859	254	899	142
660	128	700	16	740	128	780	652	820	170	860	16	900	296
661	16	701	16	741	366	781	736	821	16	861	128	901	72
662	16	702	16	742	72	782	736	822	16	862	16	902	72
663	16	703	16	743	240	783	324	823	128	863	128	903	58
664	16	704	240	744	240	784	324	824	114	864	114	904	72
665	408	705	30	745	170	785	240	825	352	865	170	905	58
666	408	706	58	746	156	786	114	826	352	866	240	906	72
667	408	707	86	747	296	787	72	827	254	867	114	907	380
668	408	708	86	748	296	788	86	828	128	868	100	908	254
669	16	709	512	749	100	789	16	829	470	869	268	909	408
670	30	710	512	750	142	790	30	830	170	870	170	910	408
671	16	711	16	751	128	791	72	831	240	871	72	911	156
672	16	712	16	752	128	792	86	832	240	872	72	912	268
673	44	713	498	753	16	793	16	833	58	873	16	913	16
674	44	714	498	754	30	794	16	834	58	874	16	914	16
675	142	715	170	755	240	795	72	835	240	875	16	915	86
676	128	716	170	756	240	796	72	836	240	876	16	916	30
677	100	717	554	757	16	797	16	837	72	877	170		
678	114	718	554	758	16	798	16	838	58	878	156		
679	16	719	652	759	72	799	72	839	16	879	156		
680	16	720	652	760	72	800	72	840	16	880	156		

B.3 Spaced Algorithm

Frequency List:
The frequencies were selected in order, from the following list of frequencies, with a spacing distance of 55. Frequencies at the start of the list are much more likely to be selected than those towards the end. The Spaced Frequencies have been shown in bold. These are the frequencies: **16, 72, 128, 240, 296, 352, 408, 470, 526, 652, 708, 764,** 30, 44, 58, 86, 100, 114, 142, 156, 170, 254, 268, 282, 310, 324, 338, 366, 380, 394, 414, 428, 442, 456, 478, 484, 498, 512, 540, 554, 666, 680, 694, 722, 736, 750, 778, 792.
Cost: 5.16170E-110. **Order (Number of Frequencies Used):** 12

Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i
1	16	41	128	81	72	121	296	161	128	201	408	241	72	281	72
2	16	42	296	82	72	122	296	162	128	202	408	242	72	282	72
3	16	43	240	83	16	123	128	163	16	203	352	243	240	283	240
4	16	44	240	84	16	124	128	164	16	204	352	244	240	284	240
5	16	45	408	85	128	125	296	165	128	205	72	245	352	285	72
6	16	46	408	86	128	126	296	166	128	206	352	246	352	286	72
7	16	47	128	87	16	127	16	167	128	207	470	247	408	287	16
8	16	48	128	88	16	128	16	168	128	208	470	248	408	288	16
9	128	49	16	89	72	129	128	169	352	209	470	249	526	289	16
10	128	50	16	90	72	130	128	170	296	210	470	250	526	290	16
11	16	51	72	91	16	131	72	171	72	211	652	251	526	291	72
12	16	52	72	92	16	132	16	172	72	212	652	252	470	292	72
13	16	53	128	93	72	133	296	173	16	213	72	253	470	293	16
14	16	54	128	94	72	134	296	174	16	214	240	254	470	294	16
15	128	55	128	95	128	135	16	175	16	215	16	255	16	295	16
16	128	56	128	96	128	136	16	176	16	216	16	256	16	296	16
17	16	57	72	97	240	137	72	177	72	217	526	257	408	297	72
18	72	58	72	98	240	138	72	178	72	218	526	258	408	298	16
19	72	59	128	99	240	139	240	179	296	219	16	259	16	299	408
20	72	60	128	100	240	140	240	180	296	220	16	260	16	300	16
21	16	61	240	101	72	141	352	181	72	221	16	261	16	301	240
22	16	62	240	102	16	142	352	182	72	222	16	262	16	302	526
23	16	63	72	103	16	143	16	183	408	223	72	263	72	303	296
24	16	64	16	104	16	144	16	184	352	224	72	264	72	304	128
25	16	65	240	105	72	145	128	185	16	225	16	265	16	305	470
26	16	66	128	106	72	146	128	186	16	226	16	266	16	306	470
27	16	67	16	107	296	147	16	187	128	227	128	267	128	307	16
28	16	68	16	108	296	148	16	188	128	228	128	268	128	308	16
29	16	69	128	109	72	149	128	189	240	229	72	269	16	309	240
30	16	70	128	110	72	150	296	190	352	230	16	270	16	310	240
31	72	71	16	111	16	151	128	191	16	231	72	271	128	311	128
32	72	72	16	112	16	152	128	192	16	232	72	272	128	312	128
33	72	73	16	113	16	153	240	193	128	233	16	273	72	313	296
34	72	74	16	114	72	154	16	194	128	234	16	274	72	314	296
35	240	75	72	115	16	155	16	195	16	235	72	275	240	315	16
36	240	76	72	116	16	156	16	196	16	236	72	276	240	316	72
37	296	77	16	117	16	157	128	197	72	237	16	277	296	317	72
38	128	78	16	118	16	158	128	198	72	238	16	278	296	318	72
39	16	79	16	119	72	159	16	199	128	239	470	279	16	319	240
40	16	80	16	120	72	160	16	200	128	240	352	280	16	320	240

Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i		
321	72	361	72	401	470	441	72	481	240	521	128	561	128	601	240
322	72	362	72	402	240	442	16	482	240	522	128	562	128	602	240
323	240	363	352	403	16	443	16	483	16	523	16	563	652	603	240
324	240	364	352	404	16	444	16	484	72	524	16	564	128	604	240
325	128	365	16	405	470	445	16	485	72	525	72	565	16	605	240
326	128	366	408	406	470	446	72	486	72	526	72	566	128	606	240
327	240	367	470	407	16	447	240	487	72	527	128	567	16	607	16
328	240	368	240	408	16	448	240	488	72	528	128	568	16	608	296
329	16	369	240	409	16	449	526	489	72	529	16	569	72	609	16
330	16	370	16	410	16	450	240	490	72	530	16	570	72	610	16
331	72	371	72	411	72	451	128	491	72	531	128	571	16	611	72
332	72	372	72	412	72	452	128	492	72	532	128	572	16	612	72
333	240	373	16	413	72	453	16	493	128	533	72	573	128	613	240
334	240	374	16	414	72	454	16	494	128	534	128	574	128	614	240
335	72	375	296	415	72	455	352	495	72	535	240	575	72	615	72
336	72	376	296	416	72	456	352	496	72	536	16	576	72	616	470
337	72	377	296	417	16	457	16	497	352	537	128	577	240	617	128
338	72	378	128	418	16	458	16	498	652	538	128	578	240	618	240
339	72	379	72	419	128	459	72	499	16	539	240	579	16	619	352
340	72	380	16	420	128	460	72	500	16	540	240	580	16	620	296
341	352	381	72	421	16	461	128	501	526	541	72	581	16	621	352
342	352	382	72	422	16	462	128	502	470	542	72	582	16	622	352
343	470	383	72	423	72	463	16	503	352	543	296	583	408	623	72
344	470	384	72	424	72	464	16	504	352	544	296	584	408	624	72
345	240	385	128	425	16	465	72	505	708	545	16	585	72	625	128
346	240	386	128	426	16	466	72	506	708	546	16	586	72	626	128
347	408	387	16	427	128	467	128	507	128	547	240	587	240	627	128
348	408	388	128	428	128	468	128	508	128	548	240	588	240	628	128
349	72	389	128	429	128	469	72	509	240	549	352	589	16	629	240
350	72	390	16	430	128	470	72	510	240	550	352	590	16	630	240
351	128	391	128	431	240	471	72	511	408	551	296	591	296	631	128
352	128	392	128	432	240	472	72	512	470	552	296	592	296	632	72
353	296	393	72	433	240	473	240	513	352	553	352	593	16	633	72
354	296	394	72	434	240	474	240	514	296	554	352	594	16	634	72
355	72	395	240	435	72	475	16	515	240	555	352	595	72	635	296
356	72	396	240	436	72	476	16	516	240	556	352	596	72	636	296
357	16	397	72	437	296	477	526	517	352	557	296	597	72	637	16
358	16	398	72	438	296	478	526	518	352	558	296	598	72	638	16
359	128	399	128	439	72	479	408	519	16	559	72	599	352	639	128
360	296	400	128	440	470	480	408	520	16	560	72	600	352	640	128

Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i	Req.	x_i
641	352	681	72	721	408	761	16	801	408	841	72	881	16
642	240	682	72	722	408	762	16	802	408	842	72	882	16
643	408	683	16	723	708	763	352	803	72	843	16	883	16
644	470	684	16	724	708	764	352	804	72	844	128	884	16
645	72	685	72	725	296	765	296	805	16	845	72	885	72
646	72	686	72	726	296	766	296	806	16	846	240	886	72
647	72	687	16	727	470	767	470	807	72	847	296	887	128
648	72	688	16	728	470	768	470	808	72	848	296	888	128
649	16	689	72	729	470	769	240	809	16	849	352	889	72
650	16	690	72	730	470	770	16	810	16	850	352	890	72
651	240	691	16	731	526	771	296	811	128	851	352	891	128
652	240	692	16	732	16	772	296	812	128	852	408	892	128
653	352	693	72	733	296	773	16	813	72	853	16	893	16
654	240	694	240	734	296	774	16	814	72	854	16	894	16
655	352	695	128	735	72	775	72	815	16	855	72	895	16
656	408	696	128	736	72	776	72	816	16	856	72	896	16
657	470	697	16	737	128	777	240	817	128	857	128	897	296
658	652	698	16	738	128	778	708	818	128	858	128	898	296
659	764	699	16	739	128	779	296	819	240	859	16	899	72
660	652	700	16	740	128	780	652	820	240	860	16	900	72
661	16	701	16	741	652	781	296	821	16	861	240	901	128
662	16	702	16	742	352	782	652	822	16	862	16	902	128
663	16	703	16	743	296	783	352	823	128	863	128	903	72
664	16	704	16	744	296	784	352	824	128	864	128	904	72
665	408	705	72	745	296	785	352	825	526	865	240	905	72
666	408	706	72	746	296	786	128	826	526	866	240	906	72
667	708	707	128	747	72	787	408	827	408	867	128	907	296
668	708	708	128	748	72	788	72	828	708	868	128	908	526
669	16	709	470	749	240	789	16	829	296	869	408	909	408
670	72	710	652	750	240	790	128	830	240	870	408	910	470
671	72	711	16	751	128	791	72	831	240	871	240	911	652
672	72	712	16	752	240	792	72	832	240	872	240	912	764
673	128	713	526	753	16	793	16	833	72	873	16	913	16
674	128	714	526	754	72	794	16	834	72	874	16	914	16
675	470	715	296	755	296	795	72	835	72	875	16	915	128
676	240	716	296	756	296	796	72	836	72	876	16	916	128
677	128	717	470	757	16	797	16	837	128	877	72		
678	128	718	470	758	16	798	16	838	128	878	72		
679	16	719	652	759	128	799	72	839	16	879	240		
680	16	720	652	760	128	800	72	840	16	880	240		

B.4 Hyper-Heuristic

This is the solution produced by the hyper-heuristic algorithm, using the Random Algorithm to produce the initial solution. Note that Req. refers to the request, x_i is the initial frequency assigned to that request (as formulated by the Random Algorithm) and x_j is the final frequency (created by the Hyper-Heuristic Algorithm).

Req.	x_i	x_j	Req.	x_i	x_j	Req.	x_i	x_j	Req.	x_i	x_j	Req.	x_i	x_j
1	338	156	41	554	16	81	366	100	121	86	394	161	254	470
2	114	156	42	554	394	82	456	708	122	394	652	162	338	708
3	484	554	43	366	470	83	394	470	123	778	156	163	142	470
4	554	554	44	408	470	84	352	470	124	428	156	164	736	470
5	708	394	45	114	100	85	498	100	125	380	652	165	366	100
6	694	394	46	44	100	86	30	652	126	156	156	166	16	100
7	540	470	47	30	652	87	58	394	127	240	764	167	338	470
8	324	652	48	170	652	88	254	394	128	380	156	168	142	16
9	470	100	49	324	764	89	478	16	129	498	394	169	352	156
10	666	156	50	708	652	90	652	394	130	722	470	170	498	394
11	282	652	51	268	100	91	44	652	131	512	394	171	268	16
12	394	470	52	86	16	92	694	652	132	324	16	172	254	16
13	16	708	53	310	708	93	470	764	133	16	708	173	366	708
14	764	100	54	792	470	94	128	156	134	736	708	174	694	652
15	170	16	55	58	764	95	240	100	135	170	156	175	680	652
16	750	100	56	72	310	96	254	100	136	666	470	176	408	16
17	310	16	57	324	16	97	296	470	137	324	394	177	86	100
18	16	16	58	240	16	98	352	470	138	30	394	178	16	394
19	694	708	59	456	156	99	442	764	139	394	100	179	100	100
20	750	708	60	58	156	100	240	310	140	526	652	180	394	394
21	142	100	61	694	394	101	142	394	141	44	708	181	498	764
22	282	100	62	652	100	102	352	394	142	408	240	182	792	764
23	666	394	63	338	100	103	512	708	143	428	764	183	352	310
24	680	652	64	694	708	104	128	708	144	30	16	184	142	310
25	394	394	65	792	16	105	16	394	145	764	764	185	44	764
26	526	394	66	470	310	106	170	394	146	72	16	186	456	156
27	778	394	67	736	100	107	652	16	147	254	652	187	750	652
28	268	16	68	470	470	108	380	16	148	694	652	188	352	470
29	240	156	69	296	708	109	380	470	149	128	394	189	86	554
30	16	16	70	114	16	110	484	16	150	128	394	190	428	652
31	498	100	71	652	394	111	694	708	151	470	470	191	100	394
32	652	100	72	142	100	112	58	708	152	408	470	192	324	394
33	324	310	73	512	394	113	142	470	153	114	240	193	512	652
34	268	310	74	750	156	114	526	156	154	268	240	194	680	310
35	72	764	75	268	652	115	44	100	155	156	708	195	478	394
36	778	764	76	324	100	116	310	100	156	708	764	196	240	394
37	526	652	77	156	470	117	554	554	157	540	156	197	100	708
38	680	156	78	100	764	118	778	240	158	554	156	198	352	100
39	694	16	79	394	156	119	652	652	159	478	394	199	694	394
40	352	708	80	240	470	120	16	16	160	540	100	200	394	394

Req.	x_i	x_j	Req.	x_i	x_j	Req.	x_i	x_j	Req.	x_i	x_j	Req.	x_i	x_j
201	156	156	241	680	16	281	338	470	321	44	16	361	86	156
202	394	394	242	470	708	282	498	470	322	254	470	362	156	16
203	792	470	243	310	100	283	282	554	323	414	16	363	324	708
204	408	100	244	498	100	284	44	554	324	554	470	364	456	708
205	100	394	245	554	470	285	366	652	325	100	100	365	16	16
206	100	100	246	58	470	286	240	156	326	736	100	366	268	764
207	722	470	247	380	764	287	30	394	327	338	470	367	750	156
208	778	708	248	338	16	288	394	100	328	324	394	368	708	652
209	764	554	249	442	156	289	394	652	329	16	156	369	750	652
210	324	554	250	764	310	290	310	394	330	652	100	370	414	652
211	778	708	251	296	554	291	240	470	331	708	394	371	736	16
212	526	652	252	484	764	292	526	156	332	394	394	372	114	100
213	470	394	253	722	708	293	170	470	333	778	156	373	414	156
214	240	100	254	170	708	294	128	652	334	114	652	374	394	156
215	484	652	255	554	100	295	240	652	335	44	394	375	778	470
216	254	16	256	380	156	296	72	652	336	414	156	376	380	470
217	58	156	257	750	394	297	282	394	337	792	394	377	254	394
218	792	156	258	394	394	298	478	708	338	708	16	378	282	394
219	30	470	259	470	100	299	296	156	339	484	16	379	414	470
220	408	764	260	170	100	300	268	156	340	338	100	380	694	310
221	366	708	261	750	16	301	142	16	341	128	310	381	128	100
222	792	652	262	680	470	302	16	16	342	296	394	382	652	652
223	44	394	263	58	310	303	778	708	343	170	310	383	750	100
224	526	16	264	540	764	304	456	708	344	408	394	384	478	156
225	380	470	265	30	16	305	708	470	345	478	470	385	156	708
226	296	652	266	764	16	306	86	470	346	86	470	386	554	708
227	792	764	267	498	470	307	72	394	347	680	652	387	30	708
228	156	100	268	268	470	308	380	652	348	352	652	388	366	708
229	128	708	269	380	394	309	296	764	349	282	470	389	254	394
230	666	156	270	394	764	310	254	764	350	778	470	390	282	652
231	72	100	271	666	394	311	708	708	351	764	394	391	170	16
232	498	394	272	680	764	312	86	470	352	254	708	392	442	16
233	498	554	273	310	708	313	736	708	353	526	554	393	680	554
234	722	100	274	156	156	314	554	100	354	86	310	394	778	470
235	352	16	275	324	708	315	310	470	355	86	470	395	680	16
236	156	498	276	512	156	316	498	708	356	764	470	396	30	16
237	540	652	277	254	708	317	366	652	357	470	470	397	478	156
238	44	16	278	114	156	318	394	652	358	58	652	398	778	470
239	526	16	279	736	156	319	16	470	359	366	394	399	282	652
240	254	156	280	72	394	320	142	764	360	394	394	400	100	100

Req.	x_i	x_j	Req.	x_i	x_j	Req.	x_i	x_j	Req.	x_i	x_j	Req.	x_i	x_j
401	156	764	441	722	100	481	58	16	521	694	708	561	72	652
402	156	156	442	722	100	482	324	16	522	30	16	562	380	156
403	240	764	443	428	708	483	30	708	523	428	394	563	428	708
404	414	764	444	156	394	484	366	652	524	16	394	564	694	652
405	44	100	445	142	156	485	414	470	525	156	708	565	764	394
406	470	156	446	324	652	486	764	470	526	498	764	566	128	764
407	16	156	447	764	708	487	394	470	527	296	310	567	58	394
408	156	394	448	484	470	488	114	470	528	512	156	568	470	394
409	72	156	449	170	708	489	652	708	529	128	470	569	478	708
410	652	394	450	380	708	490	764	16	530	72	470	570	44	16
411	778	16	451	282	156	491	750	156	531	380	16	571	380	156
412	708	708	452	680	156	492	428	708	532	156	470	572	268	100
413	156	708	453	128	708	493	30	16	533	16	100	573	128	470
414	170	708	454	484	708	494	324	554	534	680	394	574	666	394
415	394	652	455	128	100	495	428	470	535	114	652	575	764	652
416	666	708	456	764	100	496	442	470	536	456	708	576	442	652
417	736	652	457	170	394	497	324	764	537	16	394	577	428	708
418	722	16	458	778	394	498	324	310	538	680	394	578	156	708
419	498	156	459	478	470	499	694	652	539	478	156	579	722	708
420	100	156	460	44	652	500	324	708	540	44	470	580	456	156
421	428	470	461	408	708	501	498	470	541	58	100	581	142	470
422	310	652	462	414	16	502	792	652	542	352	100	582	240	470
423	666	100	463	666	100	503	428	156	543	512	708	583	722	652
424	554	394	464	142	100	504	736	470	544	366	708	584	16	16
425	736	16	465	708	100	505	268	16	545	310	394	585	156	652
426	268	156	466	478	708	506	170	16	546	694	394	586	352	652
427	268	470	467	540	156	507	16	100	547	58	16	587	512	16
428	128	470	468	100	652	508	240	470	548	310	16	588	16	156
429	86	394	469	484	708	509	470	156	549	254	764	589	512	652
430	268	16	470	310	708	510	736	310	550	708	764	590	764	100
431	778	652	471	408	156	511	394	100	551	652	156	591	72	310
432	366	708	472	394	16	512	142	652	552	764	156	592	540	764
433	338	100	473	100	708	513	498	764	553	352	16	593	554	470
434	554	708	474	324	652	514	554	394	554	268	764	594	680	394
435	736	156	475	170	470	515	470	394	555	100	100	595	736	156
436	394	156	476	554	652	516	484	16	556	282	100	596	694	708
437	470	708	477	456	708	517	142	100	557	352	310	597	268	708
438	792	708	478	16	16	518	722	100	558	498	310	598	142	708
439	254	764	479	254	764	519	478	156	559	414	100	599	72	394
440	666	156	480	324	156	520	736	394	560	540	652	600	512	394

Req.	x_i	x_j	Req.	x_i	x_j	Req.	x_i	x_j	Req.	x_i	x_j	Req.	x_i	x_j
601	470	16	641	156	156	681	708	394	721	736	764	761	310	470
602	366	16	642	30	156	682	442	708	722	72	100	762	44	394
603	554	394	643	540	16	683	540	16	723	240	652	763	778	16
604	652	708	644	680	394	684	296	16	724	652	652	764	240	708
605	16	394	645	100	652	685	408	708	725	310	100	765	156	554
606	764	470	646	254	240	686	352	470	726	414	16	766	442	708
607	736	394	647	792	156	687	86	100	727	72	394	767	142	156
608	442	708	648	764	652	688	652	156	728	792	394	768	296	156
609	394	100	649	408	16	689	512	652	729	456	652	769	652	470
610	708	708	650	170	708	690	456	652	730	652	652	770	30	394
611	352	764	651	394	652	691	44	16	731	380	470	771	694	100
612	666	764	652	72	394	692	750	708	732	708	394	772	750	652
613	750	16	653	470	470	693	540	652	733	470	394	773	394	100
614	414	16	654	442	16	694	478	394	734	554	394	774	526	652
615	512	470	655	324	100	695	114	764	735	44	470	775	30	708
616	408	554	656	100	100	696	352	16	736	310	708	776	16	470
617	554	16	657	708	764	697	750	394	737	554	156	777	394	156
618	114	100	658	764	708	698	478	156	738	554	100	778	694	310
619	324	156	659	366	240	699	366	652	739	484	708	779	442	16
620	100	394	660	100	100	700	722	652	740	750	764	780	156	156
621	296	394	661	778	100	701	680	708	741	310	240	781	778	16
622	666	394	662	44	470	702	414	470	742	114	470	782	100	764
623	652	394	663	498	470	703	142	708	743	484	156	783	540	394
624	652	156	664	100	470	704	394	652	744	394	156	784	666	394
625	240	100	665	652	652	705	470	100	745	722	708	785	778	554
626	554	652	666	30	652	706	338	16	746	100	470	786	512	100
627	512	16	667	254	764	707	792	156	747	554	394	787	352	652
628	30	16	668	498	498	708	72	156	748	764	394	788	526	470
629	470	16	669	428	156	709	16	16	749	442	470	789	722	16
630	142	708	670	380	470	710	338	156	750	554	394	790	114	16
631	456	652	671	380	764	711	526	470	751	708	708	791	240	708
632	380	100	672	366	394	712	526	470	752	680	708	792	58	394
633	366	652	673	680	652	713	268	708	753	470	394	793	750	16
634	428	16	674	268	156	714	540	156	754	16	16	794	282	16
635	240	470	675	764	764	715	498	100	755	778	16	795	170	394
636	694	708	676	470	764	716	44	100	756	324	16	796	764	470
637	380	394	677	722	156	717	310	470	757	526	708	797	114	652
638	680	100	678	708	156	718	470	470	758	394	652	798	30	652
639	44	16	679	428	16	719	44	554	759	268	394	799	708	708
640	456	652	680	128	470	720	778	310	760	156	470	800	366	394

Req.	x_i	x_j	Req.	x_i	x_j	Req.	x_i	x_j
801	554	16	841	114	652	881	526	394
802	240	16	842	750	708	882	414	394
803	58	708	843	708	394	883	708	156
804	652	470	844	86	394	884	352	156
805	512	652	845	296	652	885	792	16
806	478	652	846	142	470	886	708	764
807	394	16	847	16	156	887	72	470
808	722	708	848	778	708	888	470	470
809	282	156	849	428	16	889	30	16
810	30	470	850	666	16	890	72	470
811	554	652	851	428	470	891	512	156
812	540	394	852	750	156	892	170	156
813	338	16	853	114	100	893	338	708
814	478	16	854	72	16	894	428	394
815	44	652	855	792	156	895	526	16
816	394	394	856	170	394	896	554	470
817	310	708	857	310	470	897	652	652
818	456	156	858	442	652	898	408	394
819	324	394	859	428	394	899	16	470
820	240	240	860	764	100	900	414	470
821	30	652	861	156	652	901	554	394
822	310	16	862	484	652	902	764	16
823	652	470	863	442	652	903	352	394
824	100	470	864	254	470	904	128	156
825	380	100	865	366	652	905	792	708
826	736	100	866	16	652	906	338	708
827	484	498	867	352	470	907	694	156
828	114	16	868	498	156	908	512	652
829	254	394	869	352	100	909	414	100
830	296	156	870	526	100	910	652	554
831	722	156	871	366	652	911	72	652
832	540	156	872	128	100	912	526	652
833	100	156	873	484	470	913	324	100
834	442	470	874	366	470	914	408	394
835	428	156	875	652	708	915	666	16
836	254	156	876	792	764	916	86	708
837	310	100	877	540	100			
838	310	708	878	30	16			
839	792	16	879	282	652			
840	240	156	880	512	652			

Bibliography

[1] Farlex Financial Dictionary, "Scarcity," 2009. http://financial-dictionary.thefreedictionary.com/scarcity.

[2] W. K. Hale, "Frequency Assignment: Theory and Applications," *Proceedings of the IEEE*, vol. 68, no. 12, pp. 1497–1514, 1980.

[3] OFCOM, "Award of the 800MHz and 2.6GHz Spectrum Bands – Publication of Final Results of Auction under Regulation 111 of the Wireless Telegraphy (Licence Award) Regulations," 2012.

[4] OFCOM, "United Kingdom Frequency Allocation Table. Issue no.17," 2013.

[5] Federal Radio Commission, "Radio Act of 1927," 1927.

[6] J. Technical Advisory Committee of the IEEE and EIA, "Spectrum Engineering - The Key to Progress," 1968. Library of Congress Cat No 68-8567.

[7] D. M. Jansky, *Spectrum Management Techniques*. Don White Consultants, 1977.

[8] J. Zoeliner and C. L. Beall, "A Breakthrough in Spectrum Conserving Frequency Assignment Technology," *Electromagnetic Compatibility, IEEE Transactions on*, no. 3, pp. 313–319, 1977.

[9] S. M. Allen, D. Smith, and S. Hurley, "Lower Bounding Techniques for Frequency Assignment," *Discrete Mathematics*, vol. 197, pp. 41–52, 1999.

[10] D. Karaboga and B. Basturk, "Artificial Bee Colony (ABC) Optimization Algorithm for solving Constrained Optimization Problems," in *Foundations of Fuzzy Logic and Soft Computing*, pp. 789–798, Springer, 2007.

[11] M. da Silva Maximiano, M. A. Vega-Rodríguez, J. A. Gómez-Pulido, and J. M. Sánchez-Pérez, "A New Multiobjective Artificial Bee Colony Algorithm to solve a Real-World Frequency Assignment Problem," *Neural Computing and Applications*, vol. 22, no. 7-8, pp. 1447–1459, 2013.

[12] K. Aardal, S. P. Van Hoesel, A. M. Koster, C. Mannino, and A. Sassano, "Models and Solution Techniques for Frequency Assignment Problems," *Annals of Operations Research*, vol. 153, pp. 79–129, 2007.

[13] R. Montemanni, D. H. Smith, and S. M. Allen, "An ants Algorithm for the Minimum-Span Frequency-Assignment Problem with Multiple Interference," *Vehicular Technology, IEEE Transactions on*, vol. 51, no. 5, pp. 949–953, 2002.

[14] F. Glover, "Heuristics for Integer Programming using Surrogate Constraints," *Decision Sciences*, vol. 8, no. 1, pp. 156–166, 1977.

[15] F. Glover, "Tabu Search – Part I.," *ORSA Journal on Computing*, vol. 1, no. 3, p. 190, 1989.

[16] D. Castelino, S. Hurley, and N. Stephens, "A Tabu Search Algorithm for Frequency Assignment," *Annals of Operations Research*, vol. 63, no. 2, pp. 301–319, 1996.

[17] S. Kirkpatrick, M. Vecchi, and C. Gelatt, "Optimization by Simulated Annealing," vol. 220, no. 4598, pp. 671–680, 1983.

[18] M. Duque-Antón, D. Kunz, and B. Ruber, "Channel Assignment for Cellular Radio using Simulated Annealing," *Vehicular Technology, IEEE Transactions on*, vol. 42, no. 1, pp. 14–21, 1993.

[19] S. Herbert, "The Principles of Biology, vol 1," pp. 48, 53, 54, and others, 1864.

[20] M. Mitchell, *An Introduction to Genetic Algorithms.* MIT Press, 1996.

[21] E. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg, "Hyper-heuristics: An Emerging Direction in Modern Search Technology," *International Series in Operations Research and Management Science*, pp. 457–474, 2003.

[22] P. Ross, "Hyper-Heuristics, Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques," pp. 529–556, 2005.

[23] E. E. Korkmaz, E. Ozcan, and B. Bilgin, "A Comprehensive Analysis of Hyper-Heuristics, Intelligent Data Analysis," *Intelligent Data Analysis*, pp. 3–23, 2008.