Math6002 - Deterministic OR Methods
Coursework Report
Electoral Shenanigans

Emma Tarmey, 2940 4045

December 2021

Contents

Chapter 1

Problem Formulation

1.1 Introduction

In this report, we seek to determine the best possible division of 14 shires on a given map into 5 constituencies, each composed of one or more shires. We do so towards the goal of maximising the number of such constituencies in which "Joris Bohnson" wins a majority of the total votes inside this constituency and thus is able to send the maximum possible of representatives to Parliament.

Within each shire, the total number of voters and the total number of votes for Joris are already set. As such, we need only draw the boundaries upon the map in order to secure representatives. We do so subject to constraints regarding what makes a "valid" constituency, and the desire to maintain appearances by not allowing shire 10 to become a singleton (one shire) constituency.

1.2 Formulation of Model

1.2.1 Determining all Valid Constituencies

To begin, we must determine the set of all valid constituencies. A constituency is considered valid if it satisfies the following:

- The constituency is composed of 1 or more shires
- All shires in the constituency are adjacent
- The total number of voters in the constituency (T) satisfies $T \geq 30,000$
- The total number of voters in the constituency (T) satisfies $T \leq 100,000$
- Shire 10 may not become a single constituency.

Now, we may determine the set of valid constituencies by means of looking at our map of shires as a network graph. We may then represent each shire as a node, whose weight is given as the number of voters in the shire and each adjacency in the map is represented as an arc within the graph. This produces the following network graph:

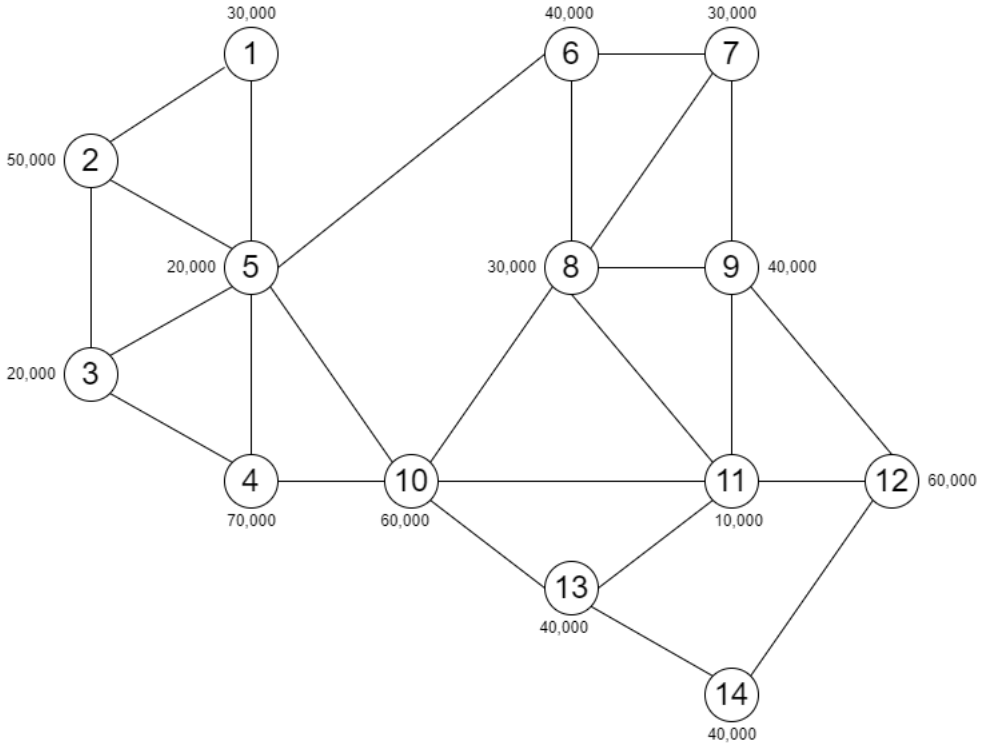

Figure 1.1: Network Representation of the 14 Shires

From this above network graph, the problem becomes forming constituencies connected nodes in the graph such that the validity conditions are always obeyed. We did so using Python code (see 3.1.1), producing a set of 49 valid constituencies (see 3.1.2). Notably, shire 10 is not allowed to become a singleton shire, in order to keep appearances up. As a result, this constituency is not valid.

In summary, from our investigation, we have found that there exist:

- 3 Valid Constituencies containing precisely 1 shire.
- 24 Valid Constituencies containing precisely 2 shires.
- 21 Valid Constituencies containing precisely 3 shires.
- 1 Valid Constituency containing precisely 4 shires.

1.2.2 Building our Allocation Matrix M

We seek to divide the 14 shires into precisely 5 constituencies. This requires us to compose any 5 of the above valid constituencies to partition the entire area. This in turn, requires us to determine which 5 of the above 49 valid constituencies should appear in our solution.

Towards this goal, we begin building a matrix M with 1 row for every shire and 1 column for every valid constituency. Our matrix M represents whether a shire in row i is present in the constituency in column j, summarising the allocations of shires across all possible constituency choices. The elements of M are hence given by the following:

$$M_{i,j} = \begin{cases} 1 & \text{if shire } i \text{ is in constituency } j \\ 0 & \text{otherwise} \end{cases}$$

We may build M in 4 parts:

- $M_1 = (14 \times 3)$ matrix, representing the section of M pertaining to all 3 valid constituencies containing precisely 1 shire
- $M_2 = (14 \times 24)$ matrix, representing the section of M pertaining to all 24 valid constituencies containing precisely 2 shires
- $M_3 = (14 \times 21)$ matrix, representing the section of M pertaining to all 21 valid constituencies containing precisely 3 shires
- $M_4 = (14 \times 1)$ matrix, representing the section of M pertaining to the 1 valid constituency containing precisely 4 shires

For readability, we have used white-space to separate the matrix entries by groups of 5 entries. This gives the following:

$$M_1 = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \qquad M_4 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$M_2 = \begin{pmatrix}
1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&1&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&1&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&1&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&0&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&1&0&0
\end{pmatrix}$$

$$M_3 = \begin{pmatrix}
1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&1&0&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&1&0&1&1&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&1&0&1&1&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&1&1&1&1&0&1&1&1&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&1&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&1&0&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1
\end{pmatrix}$$

Hence, we may compose the above matrices horizontally to form our matrix M:

$$M = \begin{pmatrix} M_1 & M_2 & M_3 & M_4 \end{pmatrix}$$

$$M \in B^{14\times 49}$$

Where $B = \{0, 1\}$ denotes the set of binary numbers.

Interpreting the above, we may take column 1 of M_2 as an example. This column represents the 2-shire constituency containing shires 1 and 2 only. As a result, this column contains a 1 in the horizontal positions 1 and 2, with 0's everywhere else. The remaining columns in all of M follow the same rule.

1.2.3 Building our Election Result Vector y

To maximise the number of representatives that Joris will be able to send to Parliament, we must maximise the number of constituencies in which Joris wins the election. Towards the goal of doing so, we must determine whether or not Joris will win the election in each of the 49 valid constituencies. We did so using Python code (see 3.2.1), producing the following vector:

$$y_1 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \qquad y_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} \qquad y_3 = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ \\ 1 \end{pmatrix} \qquad y_4 = \begin{pmatrix} 0 \end{pmatrix}$$

Hence, we may compose the above vectors vertically to form our vector y representing whether Joris will win the election in each of the 49 constituencies:

$$y = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix}$$

$$y \in B^{49}$$

Where $B = \{0, 1\}$ denotes the set of binary numbers. As previously, the subscript in each of the sub-vectors y_i corresponds to the number of shires i in each on the constituencies to which an entry (1 or 0) in the vector refers.

In summary, we have found that:

$$\text{Number of valid constituencies} = 49$$

$$\text{Number of valid constituencies where Joris wins} = 24$$

1.2.4 Building our Linear Programming Model

Using the above matrix M, we may build a linear program in the following way:

$$\begin{aligned} \text{maximise} \quad & Z = \textstyle\sum_{i=1}^{N}(x_i \times y_i) \\ \text{subject to} \quad & \textstyle\sum_{i=1}^{N}(x_i) = C \\ & Mx = b \\ & x \in B^{49} \end{aligned}$$

where we have:

Z = Objective function, number of representatives to sent to Parliament
x = Binary decision vector, decides which of the constituencies to include in the solution

M = Data matrix concerning allocation of shires within constituencies
y = Data vector concerning election results in every constituency
b = Vector of all ones

N = Constant given by total number of valid constituencies
C = Constant given by total number of constituencies allowed in a partition of the map

In our case in particular, we have from the data:

$M \in B^{14\times 49}$
$x \in B^{49}$
$y \in B^{49}$

$b = \mathbf{1}^{14}$
$N = 49$
$C = 5$

Where $B = \{0, 1\}$ denotes the set of binary numbers and $\mathbf{1}^{14}$ denotes a vector of length 14 where every entry is equal to 1. Notably, since our decision variable x is a binary vector, our model is hence a Mixed Integer Linear Programming model.

1.3 Explanation of Model

The above Linear Programming model has been designed to maximise the number of representatives that Joris is able to send to Parliament, given that an election win in 1 constituency will result in 1 representative being sent. Towards this goal, we have our objective function:

$$Z = \sum_{i=1}^{N} (x_i \times y_i)$$

This objective function multiplies element-by-element the vector x (representing which constituencies are chosen) and the vector y (representing which constituencies Joris wins the election in) and then sums across the resultant values. This objective will then tell us the number of representatives that Joris is able to send to Parliament given the choice of partition of the map given by x. Our objective function has a lower limit of 0 and an upper limit of C (in this particular case, $C = 5$).

In our problem, the partition of the map must have precisely $C = 5$ constituencies. To represent this, we have the constraint:

$$\sum_{i=1}^{N} (x_i) = C$$

This has the result that, because x is a binary decision vector, x may only have a value $x_i = 1$ in exactly 5 positions, and must then have $x_i = 0$ in the remaining 44 positions.

A valid partition of the area requires that the 5 selected constituencies must all together contain each of the 14 shires precisely once, meaning that no 2 constituencies in our solution may overlap and every shire must be contained in the partition. To represent this, we have the constraint:

$$Mx = b$$

Multiplying our (14×49) allocation matrix M by our (49×1) decision vector x gives a vector of length 14. This (14×1) binary vector Mx represents whether or not the each of the 14 shires are contained in the current solution x.

By asserting that $Mx = b$, where b is a vector of ones of length 14, we ensure that each shire is represented precisely 1 time. If a shire i was present more than once in the solution x, it would have value $(Mx)_i > 1$, and the constraint would call the solution infeasible. Moreover, if a shire i was missing from the solution x, it would have value $(Mx)_i = 0$ and also be deemed infeasible.

Chapter 2

Solution and Results

2.1 Solution Method Description

We have formed an optimal solution to our linear program by means of using XPress IVE software to formulate the above linear program (see 3.3.1), and solving this problem formulation using the XPress IVE solver (see 3.3.2).

However, the solver was initially unable to find any valid solution for the problem as specified. Upon further investigation, it became apparent that the number of constituencies, C, was causing this. A brief sanity-check yielded that:

$$\text{Total Population across the 14 Shires} = 540,000$$

$$\text{Maximum Possible Population across 5 Valid Constituencies} = 500,000$$

As a result, it is impossible to partition the entire map into 5 constituencies such that each of the constituencies has at most 100,000 voters in the electorate. Consequently, we proceed to allowing for 6 possible constituencies. Setting $C = 6$ in our model then allowed a valid solution to be generated.

In our particular XPress model, the XPress Optimizer is specified as our solver with the command "uses "mmxprs"" in our code (see 3.3.1). This solver is capable of solving Mixed Integer Programs such as ours using a "branch and bound framework" as per the FICO Xpress Optimizer documentation (see 3.4).

2.2 Presentation of Results

We have found the following optimal solution to our above linear program, which would advise Joris to create the following 6 constituencies on the map:

- Constituency 8: Shires {4, 3}
- Constituency 21: Shires {12, 9}
- Constituency 22: Shires {11, 10}
- Constituency 27: Shires {13, 14}
- Constituency 28: Shires {1, 5, 2}
- Constituency 38: Shires {7, 6, 8}

This choice of partition would then result in 5 total representatives being sent to Parliament as Joris would win the election in 5 of the 6 constituencies as per our election-results vector y. The election results hence become:

```
Constituency  8 :  {'4', '3'}
Votes for Joris =  56200
Total Voters =  90000
Joris wins

Constituency  21 :  {'12', '9'}
Votes for Joris =  53000
Total Voters =  100000
Joris wins

Constituency  22 :  {'11', '10'}
Votes for Joris =  36500
Total Voters =  70000
Joris wins

Constituency  27 :  {'13', '14'}
Votes for Joris =  44000
Total Voters =  80000
Joris wins

Constituency  28 :  {'1', '5', '2'}
Votes for Joris =  50500
Total Voters =  100000
Joris wins
```

```
Constituency  38 :  {'7', '6', '8'}
Votes for Joris =  31000
Total Voters =  100000
Joris loses
```

The solution above fulfills all of the criteria outlines in the brief, as all constituencies drawn are valid, every single shire is represented precisely once, shire 10 is not a singleton constituency and Joris wins as many of the 6 constituencies as possible.

We may visualise this partition as follows:

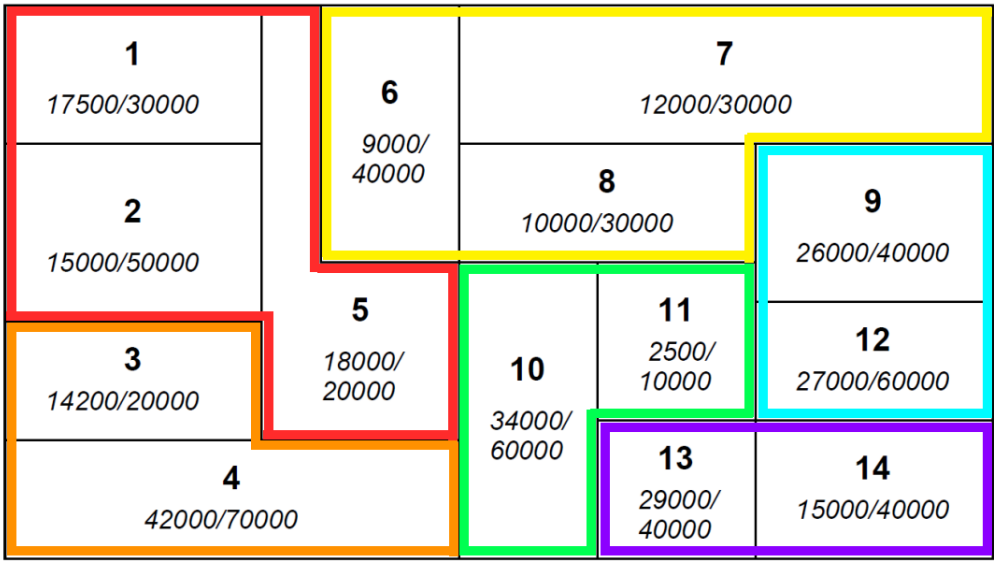

Figure 2.1: Map of the Optimal Six Constituencies

2.3 Summary of Recommended Policies

As a result, we may recommend that Joris partitions the 14 shires into the following 6 constituencies:

- Constituency 8: Shires {4, 3}
- Constituency 21: Shires {12, 9}
- Constituency 22: Shires {11, 10}
- Constituency 27: Shires {13, 14}
- Constituency 28: Shires {1, 5, 2}
- Constituency 38: Shires {7, 6, 8}

In conclusion, Joris would win the election in 5 of the 6 shires despite only winning the local elections in 7 of the 14 shires and only winning 24 of the 49 possible constituencies we may draw on the map. As a result, Joris would not only maintain appearances by not allowing shire 10 to become a single constituency, but he would also consolidate his power by means of sending 5 representatives to Parliament, only allowing 1 to slip away to another political party.

2.4 Suggestions for Further Investigation

A possible avenue to investigate further would be allowing for 7 or more constituencies to be drawn upon the map. Given the minimum requirement of 30,000 voters per constituency, it may even be worth we may be able to construct a partition of the map with as many as 12 constituencies, with shire 10 being paired up with precisely 1 other shire and all others staying as single-shire constituencies.

A brief survey of this question conducted by varying the value of C (the number of constituencies allowed within our partition) within our XPress model produces the following results:

- $C = 7$ results in Joris optimally winning 5 of the 7 constituencies
- $C = 8$ results in Joris optimally winning 5 of the 8 constituencies
- $C = 9$ does not allow for any solutions given the existing map and set of all valid constituencies

As a result, in this particular case, $C = 6$ is an optimal value for the number of constituencies. Moreover, values of C greater than 8 do not allow for solutions to be generated. This is likely because many shires would not be valid as single-shire constituencies, thus severely limiting our options . Though, with a different set of shires and voting results, the optimal value of C may change.

Moreover, even if changing C may provide more representatives in Parliament for Joris, it may also provide more representatives for non-Joris political parties. As a result, it would be worth investigating different optimal values for an objective function that minimises constituency wins for parties other than Joris. An objective function capable of this would be:

$$\text{minimise} \quad Z = \sum_{i=1}^{N} (x_i \times y'_i)$$

Where y' is a vector representing whether Joris loses the election in a given constituency.

Chapter 3

Appendices

3.1 Appendix A - Determining all Valid Constituencies

3.1.1 Python Code

```
# Math6002: Deterministic OR Methods
# Coursework 1
# Name: Emma Tarmey
# ID: 2940 4045


class graph:

    def __init__(self, graph_data = None):
        if (graph_data is None):
            graph_data = {}
        self.graph_data = graph_data

    def arcs(self):
        return self.find_all_arcs()

    # Find the distinct list of edges
    def find_all_arcs(self):
        arcs = []

        for arc in self.graph_data:
            for next_arc in self.graph_data[arc]:
                if {next_arc, arc} not in arcs:
                    arcs.append({arc, next_arc})
```

```
        return (arcs)


def find_ones(shires, data):
    ones    = []
    current = 0

    for shire in shires:
        current = data[shire]
        if ((current >= 50000) and (current <= 100000)):

            # shire 10 may not become a 1-shire constituency
            if (shire == "10"):
                pass
            else:
                ones.append(shire)
    return (ones)


def find_twos(pairs, data):
    twos    = []
    current = 0
    shire_1 = ""
    shire_2 = ""

    for pair in pairs:
        shire_1 = list(pair)[0]
        shire_2 = list(pair)[1]
        current = data[shire_1] + data[shire_2]

        if ((current >= 30000) and (current <= 100000)):
            # remove duplicates
            if (pair in twos):
                pass
            else:
                twos.append(pair)

    return (twos)


def find_all_threes(pairs, data):
    all_threes = []
    current    = {}

    for pair_1 in pairs:
        for pair_2 in pairs:
```

```
            current = pair_1.union(pair_2)

            # only accept a new constituency if we have 2 arcs between 3 shires
            # pairs of the same are rejected as union will have 2 elements
            # non-adjacent pairs also rejected as union will have 4 elements
            if (len(current) == 3):
                all_threes.append(current)

    return (all_threes)


def find_threes(pairs, data):
    all_threes = find_all_threes(pairs, data)

    threes  = []
    current = 0
    shire_1 = ""
    shire_2 = ""
    shire_3 = ""

    for three in all_threes:
        shire_1 = list(three)[0]
        shire_2 = list(three)[1]
        shire_3 = list(three)[2]
        current = data[shire_1] + data[shire_2] + data[shire_3]

        if ((current >= 30000) and (current <= 100000)):

            # remove duplicates
            if (three in threes):
                pass
            else:
                threes.append(three)

    return (threes)


def find_all_fours(pairs, data):
    all_fours = []
    current   = {}
    triples   = find_all_threes(pairs, data)

    for pair in pairs:
        for triple in triples:
            current = pair.union(triple)
```

```
            # only accept a new constituency if we have 3 arcs between 4 shires
            # pairs of the same are rejected as union will have 3 elements
            # non-adjacent pairs also rejected as union will have 5 elements
            if (len(current) == 4):
                all_fours.append(current)

    return (all_fours)


def find_fours(pairs, data):
    all_fours = find_all_fours(pairs, data)

    fours   = []
    current = 0
    shire_1 = ""
    shire_2 = ""
    shire_3 = ""
    shire_4 = ""

    for four in all_fours:
        shire_1 = list(four)[0]
        shire_2 = list(four)[1]
        shire_3 = list(four)[2]
        shire_4 = list(four)[3]
        current = data[shire_1] + data[shire_2] + data[shire_3] + data[shire_4]

        if ((current >= 30000) and (current <= 100000)):

            # remove duplicates
            if (four in fours):
                pass
            else:
                fours.append(four)

    return (fours)

def find_fives(pairs, data):
    all_fives = []
    current   = {}
    quads     = find_all_fours(pairs, data)

    for pair in pairs:
        for quad in quads:
            current = pair.union(quad)

            # only accept a new constituency if we have 4 arcs between 5 shires
```

```
            # pairs of the same are rejected as union will have 4 elements
            # non-adjacent pairs also rejected as union will have 6 elements
            if (len(current) == 5):
                all_fives.append(current)

    fives   = []
    current = 0
    shire_1 = ""
    shire_2 = ""
    shire_3 = ""
    shire_4 = ""
    shire_5 = ""

    for five in all_fives:
        shire_1 = list(five)[0]
        shire_2 = list(five)[1]
        shire_3 = list(five)[2]
        shire_4 = list(five)[3]
        shire_5 = list(five)[4]
        current = data[shire_1] + data[shire_2] + data[shire_3] + \
        data[shire_4] + data[shire_5]

        if ((current >= 30000) and (current <= 100000)):

            # remove duplicates
            if (five in fives):
                pass
            else:
                fives.append(five)

    return (fives)


# Create the dictionary with graph elements
graph_data = {
   "1"  : ["2","5"],
   "2"  : ["1", "3", "5"],
   "3"  : ["2", "4", "5"],
   "4"  : ["3", "5", "10"],
   "5"  : ["1", "2", "3", "4", "6", "10"],
   "6"  : ["5", "7", "8"],
   "7"  : ["6", "8", "9"],
   "8"  : ["6", "7", "9", "10", "11"],
   "9"  : ["7", "8", "11", "12"],
   "10" : ["4", "5", "8", "11", "13"],
   "11" : ["8", "9", "10", "12", "13"],
```

```
    "12" : ["9", "11", "14"],
    "13" : ["10", "11", "14"],
    "14" : ["12", "13"]
}

electorate_data = {
    "1"  : 30000,
    "2"  : 50000,
    "3"  : 20000,
    "4"  : 70000,
    "5"  : 20000,
    "6"  : 40000,
    "7"  : 30000,
    "8"  : 30000,
    "9"  : 40000,
    "10" : 60000,
    "11" : 10000,
    "12" : 60000,
    "13" : 40000,
    "14" : 40000
}

votes_data = {
    "1"  : 17500,
    "2"  : 15000,
    "3"  : 14200,
    "4"  : 42000,
    "5"  : 18000,
    "6"  : 9000,
    "7"  : 12000,
    "8"  : 10000,
    "9"  : 26000,
    "10" : 34000,
    "11" : 2500,
    "12" : 27000,
    "13" : 29000,
    "14" : 15000
}

g = graph(graph_data)

shires = graph_data.keys()
arcs  = g.find_all_arcs()

print("Number of shires = ", len(shires))
for shire in shires:
```

```
    print(shire)
print("\n")

print("Number of possible adjacency connections = ", len(arcs))
for arc in arcs:
    print(sorted( [int(i) for i in arc] ))
print("\n")

# Find every valid 1-shire constituency
ones = find_ones(shires, electorate_data)
print("Number of valid 1-shire constituencies = ", len(ones))
for one in ones:
    print(one)
print("\n")

# Find every valid 2-shire constituency
twos = find_twos(arcs, electorate_data)
print("Number of valid 2-shire constituencies = ", len(twos))
for two in twos:
    print(sorted( [int(i) for i in two] ))
print("\n")

# Find every valid 3-shire constituency
threes = find_threes(arcs, electorate_data)
print("Number of valid 3-shire constituencies = ", len(threes))
for three in threes:
    print(sorted( [int(i) for i in three] ))
print("\n")

# Find every valid 4-shire constituency
fours  = find_fours(arcs, electorate_data)
print("Number of valid 4-shire constituencies = ", len(fours))
for four in fours:
    print(sorted( [int(i) for i in four] ))
print("\n")

# Find every valid 5-shire constituency
fives  = find_fives(arcs, electorate_data)
print("Number of valid 5-shire constituencies = ", len(fives))
for five in fives:
    print(sorted( [int(i) for i in five] ))
print("\n")


valid_constituencies = ones + twos + threes + fours + fives
print("Total number of valid constituencies = ", len(valid_constituencies))
```

```
for i in range(0, len(valid_constituencies)):
    con = valid_constituencies[i]

    if (i < 3):
        print("Constituency ", i+1, ": \t", [int(con)])
    else:
        print("Constituency ", i+1, ": \t", sorted( [int(i) for i in con] ))
print("\n")
```

3.1.2 Command Line Output

```
Number of shires =  14
1
2
3
4
5
6
7
8
9
10
11
12
13
14


Number of possible adjacency connections =  25
[1, 2]
[1, 5]
[2, 3]
[2, 5]
[3, 4]
[3, 5]
[4, 5]
[4, 10]
[5, 6]
[5, 10]
[6, 7]
[6, 8]
[7, 8]
[7, 9]
[8, 9]
[8, 10]
[8, 11]
[9, 11]
[9, 12]
[10, 11]
[10, 13]
[11, 12]
[11, 13]
[12, 14]
[13, 14]
```

```
Number of valid 1-shire constituencies =  3
2
4
12


Number of valid 2-shire constituencies =  24
[1, 2]
[1, 5]
[2, 3]
[2, 5]
[3, 4]
[3, 5]
[4, 5]
[5, 6]
[5, 10]
[6, 7]
[6, 8]
[7, 8]
[7, 9]
[8, 9]
[8, 10]
[8, 11]
[9, 11]
[9, 12]
[10, 11]
[10, 13]
[11, 12]
[11, 13]
[12, 14]
[13, 14]


Number of valid 3-shire constituencies =  21
[1, 2, 5]
[1, 2, 3]
[1, 3, 5]
[1, 5, 6]
[2, 3, 5]
[3, 5, 6]
[3, 5, 10]
[5, 6, 7]
[5, 6, 8]
[5, 10, 11]
[6, 7, 8]
```

```
[6, 8, 11]
[7, 8, 9]
[7, 8, 11]
[7, 9, 11]
[8, 9, 11]
[8, 10, 11]
[8, 11, 12]
[8, 11, 13]
[9, 11, 13]
[11, 13, 14]


Number of valid 4-shire constituencies =  1
[5, 6, 8, 11]


Number of valid 5-shire constituencies =  0


Total number of valid constituencies =  49
Constituency  1 :          [2]
Constituency  2 :          [4]
Constituency  3 :          [12]
Constituency  4 :          [1, 2]
Constituency  5 :          [1, 5]
Constituency  6 :          [2, 3]
Constituency  7 :          [2, 5]
Constituency  8 :          [3, 4]
Constituency  9 :          [3, 5]
Constituency  10 :          [4, 5]
Constituency  11 :          [5, 6]
Constituency  12 :          [5, 10]
Constituency  13 :          [6, 7]
Constituency  14 :          [6, 8]
Constituency  15 :          [7, 8]
Constituency  16 :          [7, 9]
Constituency  17 :          [8, 9]
Constituency  18 :          [8, 10]
Constituency  19 :          [8, 11]
Constituency  20 :          [9, 11]
Constituency  21 :          [9, 12]
Constituency  22 :          [10, 11]
Constituency  23 :          [10, 13]
Constituency  24 :          [11, 12]
Constituency  25 :          [11, 13]
Constituency  26 :          [12, 14]
```

```
Constituency  27 :          [13, 14]
Constituency  28 :          [1, 2, 5]
Constituency  29 :          [1, 2, 3]
Constituency  30 :          [1, 3, 5]
Constituency  31 :          [1, 5, 6]
Constituency  32 :          [2, 3, 5]
Constituency  33 :          [3, 5, 6]
Constituency  34 :          [3, 5, 10]
Constituency  35 :          [5, 6, 7]
Constituency  36 :          [5, 6, 8]
Constituency  37 :          [5, 10, 11]
Constituency  38 :          [6, 7, 8]
Constituency  39 :          [6, 8, 11]
Constituency  40 :          [7, 8, 9]
Constituency  41 :          [7, 8, 11]
Constituency  42 :          [7, 9, 11]
Constituency  43 :          [8, 9, 11]
Constituency  44 :          [8, 10, 11]
Constituency  45 :          [8, 11, 12]
Constituency  46 :          [8, 11, 13]
Constituency  47 :          [9, 11, 13]
Constituency  48 :          [11, 13, 14]
Constituency  49 :          [5, 6, 8, 11]
```

3.2 Appendix B - Determining Election Results by Constituency

3.2.1 Python Code

```
# Math6002: Deterministic OR Methods
# Coursework 1
# Name: Emma Tarmey
# ID: 2940 4045

valid_constituencies = [{'2'}, {'4'}, {'12'}, {'2', '1'}, {'1', '5'},
                        {'2', '3'}, {'2', '5'}, {'4', '3'}, {'3', '5'},
                        {'4', '5'}, {'6', '5'}, {'10', '5'}, {'6', '7'},
                        {'6', '8'}, {'8', '7'}, {'7', '9'}, {'8', '9'},
                        {'8', '10'}, {'8', '11'}, {'9', '11'}, {'12', '9'},
                        {'10', '11'}, {'10', '13'}, {'12', '11'},
                        {'13', '11'}, {'14', '12'}, {'14', '13'},
                        {'2', '1', '5'}, {'2', '1', '3'}, {'1', '3', '5'},
                        {'6', '1', '5'}, {'2', '3', '5'}, {'6', '3', '5'},
                        {'10', '3', '5'}, {'6', '7', '5'}, {'6', '5', '8'},
                        {'10', '5', '11'}, {'6', '7', '8'}, {'6', '11', '8'},
                        {'8', '7', '9'}, {'8', '7', '11'}, {'7', '9', '11'},
                        {'8', '9', '11'}, {'8', '10', '11'}, {'8', '12', '11'},
                        {'8', '13', '11'}, {'13', '9', '11'},
                        {'14', '13', '11'}, {'6', '5', '11', '8'}]

electorate_data = {
   "1"  : 30000,
   "2"  : 50000,
   "3"  : 20000,
   "4"  : 70000,
   "5"  : 20000,
   "6"  : 40000,
   "7"  : 30000,
   "8"  : 30000,
   "9"  : 40000,
   "10" : 60000,
   "11" : 10000,
   "12" : 60000,
   "13" : 40000,
   "14" : 40000
}

votes_data = {
   "1"  : 17500,
```

```
    "2"  : 15000,
    "3"  : 14200,
    "4"  : 42000,
    "5"  : 18000,
    "6"  : 9000,
    "7"  : 12000,
    "8"  : 10000,
    "9"  : 26000,
    "10" : 34000,
    "11" : 2500,
    "12" : 27000,
    "13" : 29000,
    "14" : 15000
}


def determine_wins(constituencies, votes, totals):
    results = []
    total_votes  = 0
    total_voters = 0

    for i in range(0, len(constituencies)):
        con = constituencies[i]
        total_votes  = 0
        total_voters = 0

        print("Constituency ", i+1, ": ", con)

        for shire in con:
            total_votes  += votes_data[shire]
            total_voters += electorate_data[shire]

        print("Votes for Joris = ", total_votes)
        print("Total Voters = ", total_voters)

        if ((total_votes / total_voters) >= 0.5):
            results.append(1)
            print("Joris wins")
        else:
            results.append(0)
            print("Joris loses")

        print("")

    return results
```

```
results = determine_wins(valid_constituencies, votes_data, electorate_data)

print("Number of constituencies = ", len(valid_constituencies))
print("Number of constituencies where Joris wins = ", len([result for result \
    in results if (result == 1)]), "\n")

for i in range(0, len(results)):
    print("Constituency ", i+1, ": ", results[i])

    if ((i+1) % 5 == 0):
        print()
```

3.2.2 Command Line Output

```
Constituency  1 :  {'2'}
Votes for Joris =  15000
Total Voters =  50000
Joris loses

Constituency  2 :  {'4'}
Votes for Joris =  42000
Total Voters =  70000
Joris wins

Constituency  3 :  {'12'}
Votes for Joris =  27000
Total Voters =  60000
Joris loses

Constituency  4 :  {'1', '2'}
Votes for Joris =  32500
Total Voters =  80000
Joris loses

Constituency  5 :  {'1', '5'}
Votes for Joris =  35500
Total Voters =  50000
Joris wins

Constituency  6 :  {'2', '3'}
Votes for Joris =  29200
Total Voters =  70000
Joris loses

Constituency  7 :  {'5', '2'}
Votes for Joris =  33000
Total Voters =  70000
Joris loses

Constituency  8 :  {'4', '3'}
Votes for Joris =  56200
Total Voters =  90000
Joris wins

Constituency  9 :  {'5', '3'}
Votes for Joris =  32200
Total Voters =  40000
Joris wins
```

```
Constituency  10 :  {'5', '4'}
Votes for Joris =  60000
Total Voters =  90000
Joris wins

Constituency  11 :  {'5', '6'}
Votes for Joris =  27000
Total Voters =  60000
Joris loses

Constituency  12 :  {'5', '10'}
Votes for Joris =  52000
Total Voters =  80000
Joris wins

Constituency  13 :  {'7', '6'}
Votes for Joris =  21000
Total Voters =  70000
Joris loses

Constituency  14 :  {'6', '8'}
Votes for Joris =  19000
Total Voters =  70000
Joris loses

Constituency  15 :  {'7', '8'}
Votes for Joris =  22000
Total Voters =  60000
Joris loses

Constituency  16 :  {'7', '9'}
Votes for Joris =  38000
Total Voters =  70000
Joris wins

Constituency  17 :  {'9', '8'}
Votes for Joris =  36000
Total Voters =  70000
Joris wins

Constituency  18 :  {'10', '8'}
Votes for Joris =  44000
Total Voters =  90000
Joris loses
```

```
Constituency  19 :  {'11', '8'}
Votes for Joris =  12500
Total Voters =  40000
Joris loses

Constituency  20 :  {'11', '9'}
Votes for Joris =  28500
Total Voters =  50000
Joris wins

Constituency  21 :  {'12', '9'}
Votes for Joris =  53000
Total Voters =  100000
Joris wins

Constituency  22 :  {'11', '10'}
Votes for Joris =  36500
Total Voters =  70000
Joris wins

Constituency  23 :  {'10', '13'}
Votes for Joris =  63000
Total Voters =  100000
Joris wins

Constituency  24 :  {'11', '12'}
Votes for Joris =  29500
Total Voters =  70000
Joris loses

Constituency  25 :  {'11', '13'}
Votes for Joris =  31500
Total Voters =  50000
Joris wins

Constituency  26 :  {'12', '14'}
Votes for Joris =  42000
Total Voters =  100000
Joris loses

Constituency  27 :  {'13', '14'}
Votes for Joris =  44000
Total Voters =  80000
Joris wins

Constituency  28 :  {'1', '5', '2'}
```

```
Votes for Joris =  50500
Total Voters =  100000
Joris wins

Constituency  29 :  {'1', '2', '3'}
Votes for Joris =  46700
Total Voters =  100000
Joris loses

Constituency  30 :  {'1', '5', '3'}
Votes for Joris =  49700
Total Voters =  70000
Joris wins

Constituency  31 :  {'1', '5', '6'}
Votes for Joris =  44500
Total Voters =  90000
Joris loses

Constituency  32 :  {'5', '2', '3'}
Votes for Joris =  47200
Total Voters =  90000
Joris wins

Constituency  33 :  {'5', '6', '3'}
Votes for Joris =  41200
Total Voters =  80000
Joris wins

Constituency  34 :  {'5', '10', '3'}
Votes for Joris =  66200
Total Voters =  100000
Joris wins

Constituency  35 :  {'7', '5', '6'}
Votes for Joris =  39000
Total Voters =  90000
Joris loses

Constituency  36 :  {'5', '6', '8'}
Votes for Joris =  37000
Total Voters =  90000
Joris loses

Constituency  37 :  {'11', '5', '10'}
Votes for Joris =  54500
```

```
Total Voters =  90000
Joris wins

Constituency  38 :  {'7', '6', '8'}
Votes for Joris =  31000
Total Voters =  100000
Joris loses

Constituency  39 :  {'11', '6', '8'}
Votes for Joris =  21500
Total Voters =  80000
Joris loses

Constituency  40 :  {'7', '9', '8'}
Votes for Joris =  48000
Total Voters =  100000
Joris loses

Constituency  41 :  {'11', '7', '8'}
Votes for Joris =  24500
Total Voters =  70000
Joris loses

Constituency  42 :  {'11', '7', '9'}
Votes for Joris =  40500
Total Voters =  80000
Joris wins

Constituency  43 :  {'11', '9', '8'}
Votes for Joris =  38500
Total Voters =  80000
Joris loses

Constituency  44 :  {'11', '10', '8'}
Votes for Joris =  46500
Total Voters =  100000
Joris loses

Constituency  45 :  {'11', '12', '8'}
Votes for Joris =  39500
Total Voters =  100000
Joris loses

Constituency  46 :  {'11', '8', '13'}
Votes for Joris =  41500
Total Voters =  80000
```

```
Joris wins

Constituency  47 :  {'11', '9', '13'}
Votes for Joris =  57500
Total Voters =  90000
Joris wins

Constituency  48 :  {'11', '13', '14'}
Votes for Joris =  46500
Total Voters =  90000
Joris wins

Constituency  49 :  {'11', '5', '6', '8'}
Votes for Joris =  39500
Total Voters =  100000
Joris loses

Number of constituencies =  49
Number of constituencies where Joris wins =  24

Constituency  1 :  0
Constituency  2 :  1
Constituency  3 :  0
Constituency  4 :  0
Constituency  5 :  1

Constituency  6 :  0
Constituency  7 :  0
Constituency  8 :  1
Constituency  9 :  1
Constituency  10 :  1

Constituency  11 :  0
Constituency  12 :  1
Constituency  13 :  0
Constituency  14 :  0
Constituency  15 :  0

Constituency  16 :  1
Constituency  17 :  1
Constituency  18 :  0
Constituency  19 :  0
Constituency  20 :  1

Constituency  21 :  1
Constituency  22 :  1
```

```
Constituency  23 :  1
Constituency  24 :  0
Constituency  25 :  1

Constituency  26 :  0
Constituency  27 :  1
Constituency  28 :  1
Constituency  29 :  0
Constituency  30 :  1

Constituency  31 :  0
Constituency  32 :  1
Constituency  33 :  1
Constituency  34 :  1
Constituency  35 :  0

Constituency  36 :  0
Constituency  37 :  1
Constituency  38 :  0
Constituency  39 :  0
Constituency  40 :  0

Constituency  41 :  0
Constituency  42 :  1
Constituency  43 :  0
Constituency  44 :  0
Constituency  45 :  0

Constituency  46 :  1
Constituency  47 :  1
Constituency  48 :  1
Constituency  49 :  0
```

3.3 Appendix C - Linear Programming Model

3.3.1 XPress-IVE Code

```
(!
        Math6002: Deterministic OR Methods
        Coursework XPress IVE Model
        Name: Emma Tarmey
        ID: 2940 4045
!)

model LPModel
        uses "mmxprs";

        declarations
                ! constants
                C = 6
                constituencies = 1..49
                shires         = 1..14

                ! data
                b: array(shires) of real
                y: array(constituencies) of real
                M: array(shires, constituencies) of real

                ! variables
                x: array(constituencies) of mpvar ! decision variable
        end-declarations

        ! vector of size 14
        b :: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                  1, 1, 1, 1]

        ! vector of size 49
        y :: [0, 1, 0,

              0, 1, 0, 0, 1,
              1, 1, 0, 1, 0,
              0, 0, 1, 1, 0,
              0, 1, 1, 1, 1,
              0, 1, 0, 1,

              1, 0, 1, 0, 1,
              1, 1, 0, 0, 1,
              0, 0, 0, 0, 1,
              0, 0, 0, 1, 1,
```

```
      1,

      0]

! M is matrix of size (14 x 49)
! linebreaks inserted for formatting when included in report
M :: [0, 0, 0,   1, 1, 0, 0, 0,   0, 0, 0, 0, 0,   0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0,   1, 1, 1, 1, 0,   0, 0, 0, 0, 0,
      0, 0, 0, 0, 0,   0, 0, 0, 0, 0,   0,   0,

          1, 0, 0,   1, 0, 1, 1, 0,   0, 0, 0, 0, 0,   0, 0, 0, 0, 0,
          0, 0, 0, 0, 0,   0, 0, 0, 0,   1, 1, 0, 0, 1,
          0, 0, 0, 0, 0,   0, 0, 0, 0, 0,   0, 0, 0, 0, 0,   0,   0,

          0, 0, 0,   0, 0, 1, 0, 1,   1, 0, 0, 0, 0,   0, 0, 0, 0, 0,
          0, 0, 0, 0, 0,   0, 0, 0, 0,   0, 1, 1, 0, 1,
          1, 1, 0, 0, 0,   0, 0, 0, 0, 0,   0, 0, 0, 0, 0,   0,   0,

          0, 1, 0,   0, 0, 0, 0, 1,   0, 1, 0, 0, 0,   0, 0, 0, 0, 0,
          0, 0, 0, 0, 0,   0, 0, 0, 0,   0, 0, 0, 0, 0,
          0, 0, 0, 0, 0,   0, 0, 0, 0, 0,   0, 0, 0, 0, 0,   0,   0,

          0, 0, 0,   0, 1, 0, 1, 0,   1, 1, 1, 1, 0,   0, 0, 0, 0, 0,
          0, 0, 0, 0, 0,   0, 0, 0, 0,   1, 0, 1, 1, 1,
          1, 1, 1, 1, 1,   0, 0, 0, 0, 0,   0, 0, 0, 0, 0,   0,   1,


          0, 0, 0,   0, 0, 0, 0, 0,   0, 0, 1, 0, 1,   1, 0, 0, 0, 0,
          0, 0, 0, 0, 0,   0, 0, 0, 0,   0, 0, 0, 1, 0,
          1, 0, 1, 1, 0,   1, 1, 0, 0, 0,   0, 0, 0, 0, 0,   0,   1,

          0, 0, 0,   0, 0, 0, 0, 0,   0, 0, 0, 0, 1,   0, 1, 1, 0, 0,
          0, 0, 0, 0, 0,   0, 0, 0, 0,   0, 0, 0, 0, 0,
          0, 0, 1, 0, 0,   1, 0, 1, 1, 1,   0, 0, 0, 0, 0,   0,   0,

          0, 0, 0,   0, 0, 0, 0, 0,   0, 0, 0, 0, 0,   1, 1, 0, 1, 1,
          1, 0, 0, 0, 0,   0, 0, 0, 0,   0, 0, 0, 0, 0,
          0, 0, 0, 1, 0,   1, 1, 1, 1, 0,   1, 1, 1, 1, 0,   0,   1,

          0, 0, 0,   0, 0, 0, 0, 0,   0, 0, 0, 0, 0,   0, 0, 1, 1, 0,
          0, 1, 1, 0, 0,   0, 0, 0, 0,   0, 0, 0, 0, 0,
          0, 0, 0, 0, 0,   0, 0, 1, 0, 1,   1, 0, 0, 0, 1,   0,   0,

          0, 0, 0,   0, 0, 0, 0, 0,   0, 0, 0, 1, 0,   0, 0, 0, 0, 1,
          0, 0, 0, 1, 1,   0, 0, 0, 0,   0, 0, 0, 0, 0,
          0, 1, 0, 0, 1,   0, 0, 0, 0, 0,   0, 1, 0, 0, 0,   0,   0,
```

```
          0, 0, 0,   0, 0, 0, 0, 0,   0, 0, 0, 0, 0,   0, 0, 0, 0, 0,
          1, 1, 0, 1, 0,   1, 1, 0, 0,   0, 0, 0, 0, 0,
          0, 0, 0, 0, 1,   0, 1, 0, 1, 1,   1, 1, 1, 1, 1,   1,   1,

          0, 0, 1,   0, 0, 0, 0, 0,   0, 0, 0, 0, 0,   0, 0, 0, 0, 0,
          0, 0, 1, 0, 0,   1, 0, 1, 0,   0, 0, 0, 0, 0,
          0, 0, 0, 0, 0,   0, 0, 0, 0, 0,   0, 0, 1, 0, 0,   0,   0,

          0, 0, 0,   0, 0, 0, 0, 0,   0, 0, 0, 0, 0,   0, 0, 0, 0, 0,
          0, 0, 0, 0, 1,   0, 1, 0, 1,   0, 0, 0, 0, 0,
          0, 0, 0, 0, 0,   0, 0, 0, 0, 0,   0, 0, 0, 1, 1,   1,   0,

          0, 0, 0,   0, 0, 0, 0, 0,   0, 0, 0, 0, 0,   0, 0, 0, 0, 0,
          0, 0, 0, 0, 0,   0, 0, 1, 1,   0, 0, 0, 0, 0,
          0, 0, 0, 0, 0,   0, 0, 0, 0, 0,   0, 0, 0, 0, 0,   1,   0]

! As a sanity check, we may check that all of the columns in M
! columns should sum to 1, 2, 3 or 4 as appropriate
writeln("\nSanity check of columns sums in M:")
forall (j in constituencies) do
        if (j = 4 or j = 28 or j = 49)
        then writeln("\n")
        end-if

        writeln("Con j = ", j, ", sum (of shires i) of M(i, j) = ",
                 sum (i in shires) (M(i, j)))
end-do

! objective function
total_representatives := sum(i in constituencies) ( x(i) * y(i) )

! constraints

! decision variable x is binary vector
forall (i in constituencies) x(i) is_binary

forall (i in shires) do
        ! check each shire appears exactly once
        allocation(i) := sum (j in constituencies) (M(i, j) * x(j)) = 1
end-do

! map partition must contain C constituencies
sum (i in constituencies) x(i) = C
```

```
        ! run optimisation
        maximise(total_representatives)

        ! output results
        writeln("\nBegin running model\n")
        writeln("Optimal number of representatives = ", getobjval, "\n")
        writeln("Solution of Chosen Constituencies: ")

        forall (i in constituencies) do
                write  ("\tcon = ", i)
                write  (", x = ", (getsol(x(i))))
                write  ("\n")
        end-do

        writeln("\nEnd running model")
end-model
```

3.3.2 Command Line Output

```
Thu Dec 09 2021 19:29:47 GMT+0000 (Greenwich Mean Time)
FICO Xpress Mosel 64-bit v5.8.0, FICO Xpress v8.13.0
(c) Copyright Fair Isaac Corporation 2001-2021. All rights reserved
Compiling lp_model.mos to out\lp_model.bim with -g
Running model

Sanity check of columns sums in M:
Con j = 1, sum (of shires i) of M(i, j) = 1
Con j = 2, sum (of shires i) of M(i, j) = 1
Con j = 3, sum (of shires i) of M(i, j) = 1


Con j = 4, sum (of shires i) of M(i, j) = 2
Con j = 5, sum (of shires i) of M(i, j) = 2
Con j = 6, sum (of shires i) of M(i, j) = 2
Con j = 7, sum (of shires i) of M(i, j) = 2
Con j = 8, sum (of shires i) of M(i, j) = 2
Con j = 9, sum (of shires i) of M(i, j) = 2
Con j = 10, sum (of shires i) of M(i, j) = 2
Con j = 11, sum (of shires i) of M(i, j) = 2
Con j = 12, sum (of shires i) of M(i, j) = 2
Con j = 13, sum (of shires i) of M(i, j) = 2
Con j = 14, sum (of shires i) of M(i, j) = 2
Con j = 15, sum (of shires i) of M(i, j) = 2
Con j = 16, sum (of shires i) of M(i, j) = 2
Con j = 17, sum (of shires i) of M(i, j) = 2
Con j = 18, sum (of shires i) of M(i, j) = 2
Con j = 19, sum (of shires i) of M(i, j) = 2
Con j = 20, sum (of shires i) of M(i, j) = 2
Con j = 21, sum (of shires i) of M(i, j) = 2
Con j = 22, sum (of shires i) of M(i, j) = 2
Con j = 23, sum (of shires i) of M(i, j) = 2
Con j = 24, sum (of shires i) of M(i, j) = 2
Con j = 25, sum (of shires i) of M(i, j) = 2
Con j = 26, sum (of shires i) of M(i, j) = 2
Con j = 27, sum (of shires i) of M(i, j) = 2


Con j = 28, sum (of shires i) of M(i, j) = 3
Con j = 29, sum (of shires i) of M(i, j) = 3
Con j = 30, sum (of shires i) of M(i, j) = 3
Con j = 31, sum (of shires i) of M(i, j) = 3
Con j = 32, sum (of shires i) of M(i, j) = 3
Con j = 33, sum (of shires i) of M(i, j) = 3
```

```
Con j = 34, sum (of shires i) of M(i, j) = 3
Con j = 35, sum (of shires i) of M(i, j) = 3
Con j = 36, sum (of shires i) of M(i, j) = 3
Con j = 37, sum (of shires i) of M(i, j) = 3
Con j = 38, sum (of shires i) of M(i, j) = 3
Con j = 39, sum (of shires i) of M(i, j) = 3
Con j = 40, sum (of shires i) of M(i, j) = 3
Con j = 41, sum (of shires i) of M(i, j) = 3
Con j = 42, sum (of shires i) of M(i, j) = 3
Con j = 43, sum (of shires i) of M(i, j) = 3
Con j = 44, sum (of shires i) of M(i, j) = 3
Con j = 45, sum (of shires i) of M(i, j) = 3
Con j = 46, sum (of shires i) of M(i, j) = 3
Con j = 47, sum (of shires i) of M(i, j) = 3
Con j = 48, sum (of shires i) of M(i, j) = 3


Con j = 49, sum (of shires i) of M(i, j) = 4

Begin running model

Optimal number of representatives = 5

Solution of Chosen Constituencies:
        con = 1, x = 0
        con = 2, x = 0
        con = 3, x = 0
        con = 4, x = 0
        con = 5, x = 0
        con = 6, x = 0
        con = 7, x = 0
        con = 8, x = 1
        con = 9, x = 0
        con = 10, x = 0
        con = 11, x = 0
        con = 12, x = 0
        con = 13, x = 0
        con = 14, x = 0
        con = 15, x = 0
        con = 16, x = 0
        con = 17, x = 0
        con = 18, x = 0
        con = 19, x = 0
        con = 20, x = 0
        con = 21, x = 1
        con = 22, x = 1
```

```
        con = 23, x = 0
        con = 24, x = 0
        con = 25, x = 0
        con = 26, x = 0
        con = 27, x = 1
        con = 28, x = 1
        con = 29, x = 0
        con = 30, x = 0
        con = 31, x = 0
        con = 32, x = 0
        con = 33, x = 0
        con = 34, x = 0
        con = 35, x = 0
        con = 36, x = 0
        con = 37, x = 0
        con = 38, x = 1
        con = 39, x = 0
        con = 40, x = 0
        con = 41, x = 0
        con = 42, x = 0
        con = 43, x = 0
        con = 44, x = 0
        con = 45, x = 0
        con = 46, x = 0
        con = 47, x = 0
        con = 48, x = 0
        con = 49, x = 0

End running model

Process exited with code: 0
```

3.4 Appendix D - Acknowledgements

- The Python graph tutorial available at `https://www.tutorialspoint.com/python_data_structure/python_graphs.htm` was consulted towards implementing graphs to answer the first question.
- The XPress IVE coded examples available at `https://examples.xpress.fico.com/example.pl` were consulted towards learning the syntax of the Mosel language.
- The FICO XPress Optimization documentation available at `https://www.fico.com/fico-xpress-optimization/docs/latest/solver/optimizer/HTML/chapter1_sec_section1001.html` was consulted to investigate how the XPress Optimizer works as a solver for models.