Question 1

a)
```
Sub main()
  Dim col As Integer, num As Integer, cell As Integer, adj(0 To 3)
     ↪ As Integer
  Application.Calculation = xlManual
  For cell = 0 To 27 'Iterate across blocks
    num = 0
    If Int(cell / 7) <> 0 Then adj(num) = cell - 7: num = num + 1 '
       ↪ If not on top edge, add above cell to adjacents
    If Int(cell / 7) <> 3 Then adj(num) = cell + 7: num = num + 1 '
       ↪ If not on bottom edge, add below cell to adjacents
    If cell Mod 7 <> 0 Then adj(num) = cell - 1: num = num + 1 'If
       ↪ not on left edge, add left cell to adjacents
    If cell Mod 7 <> 6 Then adj(num) = cell + 1: num = num + 1 'If
       ↪ not on right edge, add right cell to adjacents
    generateColumns cell, col, adj, num
  Next cell
  Application.Calculation = xlAutomatic
End Sub

Sub generateColumns(cell As Integer, col As Integer, adj() As
   ↪ Integer, num As Integer)
  Dim m As Integer
  For m = 0 To 2 ^ num - 1 'For each centre permutation
    Sheet1.Cells(cell + 1, col + 1) = 1 'Deliver to self
    For n = 0 To num - 1
      If (m And (2 ^ n)) <> 0 Then Sheet1.Cells(adj(n) + 1, col + 1)
         ↪  = 1 'Deliver to adjacents
    Next n
    col = col + 1
  Next m
End Sub
```

b) The A matrix consists of 28 rows, one for each block, and 288 columns. Each column corresponds to a binary variable that is equal to one if the set described by the column is chosen and zero otherwise.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
1	1	0.645	0.387	0	0	0	0	0.613	0.742	0	0	0	0	0	0.355	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0.645	1	0.71	0.323	0	0	0	0.29	0.419	0.355	0	0	0	0	0	0.032	0	0	0	0	0	0	0	0	0	0	0	0
3	0.387	0.71	1	0.419	0.097	0	0	0	0.581	0.516	0.29	0	0	0	0	0	0.194	0	0	0	0	0	0	0	0	0	0	0
4	0	0.323	0.419	1	0.419	0.323	0	0	0	0.581	0.355	0.581	0	0	0	0	0	0.258	0	0	0	0	0	0	0	0	0	0
5	0	0	0.097	0.419	1	0.71	0.387	0	0	0	0.29	0.516	0.581	0	0	0	0	0	0.194	0	0	0	0	0	0	0	0	0
6	0	0	0	0.323	0.71	1	0.645	0	0	0	0	0.355	0.419	0.29	0	0	0	0	0	0.032	0	0	0	0	0	0	0	0
7	0	0	0	0	0.387	0.645	1	0	0	0	0	0	0.742	0.613	0	0	0	0	0	0	0.355	0	0	0	0	0	0	0
8	0.613	0.29	0	0	0	0	0	1	0.387	0.032	0	0	0	0	0.71	0.387	0	0	0	0	0	0.355	0	0	0	0	0	0
9	0.742	0.419	0.581	0	0	0	0	0.387	1	0.226	0.194	0	0	0	0.387	0.065	0.226	0	0	0	0	0	0.032	0	0	0	0	0
10	0	0.355	0.516	0.581	0	0	0	0.032	0.226	1	0.452	0.258	0	0	0	0.226	0.387	0.452	0	0	0	0	0	0.194	0	0	0	0
11	0	0	0.29	0.355	0.29	0	0	0	0.194	0.452	1	0.452	0.194	0	0	0	0.452	0.516	0.452	0	0	0	0	0	0.258	0	0	0
12	0	0	0	0.581	0.516	0.355	0	0	0	0.258	0.452	1	0.226	0.032	0	0	0	0.452	0.387	0.226	0	0	0	0	0	0.194	0	0
13	0	0	0	0	0.581	0.419	0.742	0	0	0	0.194	0.226	1	0.387	0	0	0	0	0.226	0.065	0.387	0	0	0	0	0	0.032	0
14	0	0	0	0	0	0.29	0.613	0	0	0	0	0.032	0.387	1	0	0	0	0	0	0.387	0.71	0	0	0	0	0	0	0.355
15	0.355	0	0	0	0	0	0	0.71	0.387	0	0	0	0	0	1	0.387	0.032	0	0	0	0	0.613	0.29	0	0	0	0	0
16	0	0.032	0	0	0	0	0	0.387	0.065	0.226	0	0	0	0	0.387	1	0.226	0.194	0	0	0	0.742	0.419	0.581	0	0	0	0
17	0	0	0.194	0	0	0	0	0	0.226	0.387	0.452	0	0	0	0.032	0.226	1	0.452	0.258	0	0	0	0.355	0.516	0.581	0	0	0
18	0	0	0	0.258	0	0	0	0	0	0.452	0.516	0.452	0	0	0	0.194	0.452	1	0.452	0.194	0	0	0	0.29	0.355	0.29	0	0
19	0	0	0	0	0.194	0	0	0	0	0	0.452	0.387	0.226	0	0	0	0.258	0.452	1	0.226	0.032	0	0	0	0.581	0.516	0.355	0
20	0	0	0	0	0	0.032	0	0	0	0	0	0.226	0.065	0.387	0	0	0	0.194	0.226	1	0.387	0	0	0	0	0.581	0.419	0.742
21	0	0	0	0	0	0	0.355	0	0	0	0	0	0.387	0.71	0	0	0	0	0.032	0.387	1	0	0	0	0	0	0.29	0.613
22	0	0	0	0	0	0	0	0.355	0	0	0	0	0	0	0.613	0.742	0	0	0	0	0	1	0.645	0.387	0	0	0	0
23	0	0	0	0	0	0	0	0	0.032	0	0	0	0	0	0.29	0.419	0.355	0	0	0	0	0.645	1	0.71	0.323	0	0	0
24	0	0	0	0	0	0	0	0	0	0.194	0	0	0	0	0	0.581	0.516	0.29	0	0	0	0.387	0.71	1	0.419	0.097	0	0
25	0	0	0	0	0	0	0	0	0	0	0.258	0	0	0	0	0	0.581	0.355	0.581	0	0	0	0.323	0.419	1	0.419	0.323	0
26	0	0	0	0	0	0	0	0	0	0	0	0.194	0	0	0	0	0	0.29	0.516	0.581	0	0	0	0.097	0.419	1	0.71	0.387
27	0	0	0	0	0	0	0	0	0	0	0	0	0.032	0	0	0	0	0	0.355	0.419	0.29	0	0	0	0.323	0.71	1	0.645
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0.355	0	0	0	0	0	0.742	0.613	0	0	0	0	0.387	0.645	1

Figure 1: Sum of fractions table - highlighted yellow are numbers close to 1

c) (i) 1-branch:

$$x_i \leq 0 \quad \forall i \in \mathrm{I}^*(s,t) \text{ where } \mathrm{I}^*(s,t) = \{i | (a_{si} = 1, a_{ti} = 0) \text{ or } (a_{si} = 0, a_{ti} = 1)\}$$

(ii) 0-branch:

$$x_i \leq 0 \quad \forall i \in \mathrm{I}(s,t) \text{ where } \mathrm{I}(s,t) = \{i | (a_{si} = 1, a_{ti} = 1)\}$$

d) Branching on the pair (0, 8):
0-branch: z = 44107
1-branch: z = 43340 (integer)

e) Yes, the optimal solution is the solution from the 1-branch, as it is integer. The solution is as follows, with a total cost of $43340:

Cabinet	Location	Battery size (mAh)	Delivers to
1	1	5000	1, 2, 3, 9
2	4	3000	4, 5, 11
3	7	5000	0, 7, 8, 14
4	13	5000	6, 12, 13, 20
5	17	5000	10, 16, 17, 18, 24
6	22	5000	15, 21, 22, 23
7	26	5000	19, 25, 26, 27

Question 2

a) Order the deliveries by shortest distance first, then deliver in that order.

b) The GSPP is formulated as such:

$$\begin{aligned} \min \quad & z = c^T x \\ \text{st} \quad & Ax = 1 \\ & e^T x \leq 14 \\ & x \text{ binary} \end{aligned}$$

where A and x are defined as per the standard SPP, and e^T is the vector of ones.
The extra constraint line restricts the total number of cabinets installed to be less than 14, and the RHS is adjusted to produce a range of non-dominated solutions.

The code used to generate the A matrix:

```
# Benjamin Yi
# 925302651
# byi649

import numpy as np
from itertools import chain, combinations
import heapq

distances = np.loadtxt('distances.txt')
workable = []

# Choose only 8 closest neighbours
for i, col in enumerate(distances):
  workable.append(col <= heapq.nsmallest(8, col)[-1])

# Turn booleans into list of index
index = [[i for i, x in enumerate(sub) if x] for sub in workable]
```

```
# Create all permutations between lengths 3, 5
matrixindex = [list(chain.from_iterable(combinations(s, r) for r in
    ↪ range(3, 6))) for s in index]
# Remove duplicates
matrixindex = list(dict.fromkeys(list(filter(None, [item for sublist
    ↪  in matrixindex for item in sublist]))))

matrix = [np.zeros(48) for x in range(len(matrixindex))]

# Convert back to A matrix
for i, column in enumerate(matrix):
    for j in matrixindex[i]:
        column[j] = 1

matrix = np.transpose(matrix)

np.savetxt("matrix.txt", matrix, fmt="%d")
```

This generates all potential combinations of neighbours around a cabinet that are formed from the 8 nearest blocks, between lengths 3 and 5 inclusive.

This will give an A matrix of column length 5004. By choosing neighbours only from the 8 closest blocks and restricting the length of the set to be between 3 and 5, this is much lower than enumerating all possible combinations - which would be 2^{48} in total.

While technically choosing only the 8 closest neighbours and only choosing sets of length 3 or greater potentially cuts off the optimal solution, the remaining solutions will tend to have much better objective functions than the rest of the sets. The chance of cutting off the optimal solution is low, and the computational costs recuperated are high (Excel cannot support 2^{48} columns in a worksheet), so it is worth doing.

c) As this is a bi-objective optimisation problem, it has been solved for all feasible monetary costs, optimising average delivery time each time.

Number of cabinets: 14, cost = \$84000, average time taken: 256 seconds

Cabinet location	Delivers to
551500	551100, 551200, 551500, 551600
551401	551300, 551401, 551402, 551700
552600	552500, 552600, 552700, 553600
552900	552400, 552800, 552900, 553200
553100	553000, 553100, 553400
555200	553300, 555100, 555200
559100	556100, 559100, 559200
556400	556200, 556300, 556400, 556800
556900	556500, 556900, 557800
557000	556600, 557000, 557200
557300	557100, 557300, 557401, 557402
557600	557500, 557600, 558600
557900	557700, 557900, 558100
558200	558000, 558200, 558500

Visualisation:

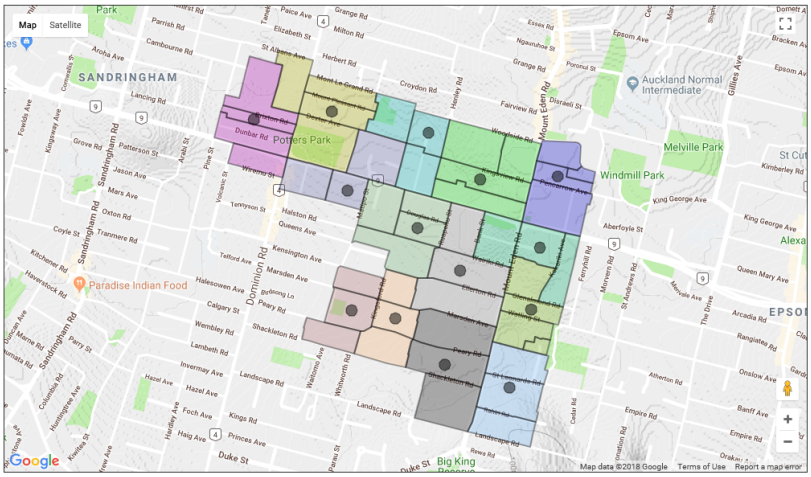

Figure 2: Cabinet distribution for 14 cabinets

Number of cabinets: 13, cost = $78000, average time taken: 279 seconds

Cabinet location	Delivers to
551500	551100, 551200, 551500, 551600
551401	551300, 551401, 551402, 551700
552600	552500, 552600, 552700, 553600
552900	552400, 552800, 552900, 553200
553100	553000, 553100, 553400
555200	553300, 555100, 555200
559100	556100, 559100, 559200
556400	556200, 556300, 556400, 556800
556900	556500, 556600, 556900, 557800
557100	557000, 557100, 557200, 557300
557500	557401, 557402, 557500, 557600
557900	557700, 557900, 558000, 558100
558500	558200, 558500, 558600

Number of cabinets: 12, cost = $72000, average time taken: 305 seconds

Cabinet location	Delivers to
551500	551100, 551200, 551500, 551600
551401	551300, 551401, 551402, 551700
552600	552500, 552600, 552700, 553600
552900	552400, 552800, 552900, 553200
553400	553000, 553100, 553400, 556300
555200	553300, 555100, 555200, 556200
559100	556100, 558100, 559100, 559200
556900	556500, 556600, 556900, 557800
557100	557000, 557100, 557200, 557300
557500	557401, 557402, 557500, 557600
558200	558000, 558200, 558500, 558600
557700	556400, 558600, 557700, 557900

Number of cabinets: 11, cost = $66000, average time taken: 340 seconds

Cabinet location	Delivers to
551500	551100, 551500, 551600, 553100
551401	551200, 551300, 551401, 551402, 551700
552600	552400, 552500, 552600, 552700, 553600
552900	552800, 552900, 553000, 553200
555200	553300, 555100, 555200, 556200
559100	556100, 557900, 558100, 559100, 559200
556400	553400, 556300, 556400, 556500, 556800
556900	556600, 556900, 557700, 557800
557100	557000, 557100, 557200, 557300
557500	557401, 557402, 557500, 557600
558200	558000, 558200, 558500, 558600

Number of cabinets: 10, cost = \$60000, average time taken: 378 seconds

Cabinet location	Delivers to
551401	551200, 551300, 551401, 551402, 551700
551600	551100, 551500, 551600, 556500, 556600
552900	552400, 552800, 552900, 553200
552700	552500, 552600, 552700, 553600, 555100
553300	553000, 553100, 553300, 553400, 555200
559100	556100, 557900, 558100, 559100, 559200
556400	556200, 556300, 556400, 556800, 557700
557100	556900, 557000, 557100, 557200, 557300
557500	557401, 557402, 557500, 557600, 558600
558000	557800, 558000, 558200, 558500

Each solution trades off monetary cost and average time for delivery, therefore none of these solutions are strictly superior to any other. The choice will depend on whatever objective the decision maker deems more important.