KY3: Shop Allocation

Kieran Molloy

January 2020

Abstract

Investigate a Shop Allocation MIQP, modelling weighted flows to minimise flow-distance between popular shops. Modelling the MIQP, then solving using MATLAB and custom functions.

1 Introduction

There are four shops and each shop is to be placed at a empty letting point in a new shopping centre. The distance between the available letting locations is shown in table 1. The store manager wishes to assign each shop to a letting position such that the customer inconvenience is minimised with respect to how far they have to walk to get between the stores they visit, the number of shoppers travelling from shop to shop is in table 2. function.*KY3 Specification* 2019

2 Analysis

2.1 Details of the Problem

	1	2	3	4
1	*	80	150	170
2	80	*	130	100
3	150	130	*	120
4	150	100	120	*

Table 1: Distance Between Locations

	1	2	3	4
1	*	5	2	7
2	5	*	3	8
3	2	3	*	3
4	7	8	3	*

Table 2: Customer Flow Between Shops

2.2 Formulation

This shop allocation problem is MIQP (Mixed Integer Quadratic Problem), and as such standard integer solving methods do not apply here.

Variable	Description
L	Set of Locations
S	Set of Shops
F_{s_1,s_2}	Customer Flow between shop s_1 and s_2
D_{l_1,l_2}	Distance between location l_1 and l_2
$x_{l,s}$	1, if shop s is assigned to location l. 0, if not
f_{s_1,s_2}	Flow-Distance value for s_1 and s_2

Table 3: Shop Allocation Notation

$$\begin{aligned}
\underset{f}{\text{minimise}} \quad & f_{s_1,s_2} + f_{s_1,s_3} + f_{s_1,s_4} + f_{s_2,s_3} + f_{s_2,s_4} + f_{s_3,s_4} \\
\text{subject to} \quad & f_{s_i,s_j} = F_{s_i,s_j} * D_{l_1,l_2}x_{1,1}x_{2,2} + D_{l_1,l_3}x_{1,1}x_{2,3} + D_{l_1,l_4}x_{1,1}x_{2,4} + \\
& D_{l_2,l_1}x_{1,2}x_{2,1} + D_{l_2,l_3}x_{1,2}x_{2,3} + D_{l_2,l_4}x_{1,2}x_{2,4} + D_{l_3,l_1}x_{1,3}x_{2,1} + \\
& D_{l_3,l_2}x_{1,3}x_{2,2} + D_{l_3,l_4}x_{1,3}x_{2,4} + D_{l_4,l_1}x_{1,4}x_{2,1} + D_{l_4,l_2}x_{1,4}x_{2,2} + D_{l_4,l_3}x_{1,4}x_{2,3} \\
& x_{s,l} = 0,1, \; s \in S, l \in L.
\end{aligned} \tag{1}$$

Ensure Shop Singularity

$$\begin{aligned}
x_{1,1} + x_{1,2} + x_{1,3} + x_{1,4} &= 1, \\
x_{2,1} + x_{2,2} + x_{2,3} + x_{2,4} &= 1, \\
x_{3,1} + x_{3,2} + x_{3,3} + x_{3,4} &= 1, \\
x_{4,1} + x_{4,2} + x_{4,3} + x_{4,4} &= 1, \\
x_{s,l} &= 0,1, \; s \in S, l \in L.
\end{aligned} \tag{2}$$

Ensure Location Singularity

$$\begin{aligned}
x_{1,1} + x_{2,1} + x_{3,1} + x_{4,1} &= 1, \\
x_{1,2} + x_{2,2} + x_{3,2} + x_{4,2} &= 1, \\
x_{1,3} + x_{2,3} + x_{3,3} + x_{4,3} &= 1, \\
x_{1,4} + x_{2,4} + x_{3,4} + x_{4,4} &= 1, \\
x_{s,l} &= 0,1, \; s \in S, l \in L.
\end{aligned} \tag{3}$$

This model was then created in MATLAB r2018a *Matlab* 2019

3 Results

The code to make this section run is in Appendix A.

Running the model gives:

Total flow-distance: 2000

Allocation plan:

	1	2	3	4
1	*	1	*	*
2	1	*	*	*
3	*	*	*	1
4	*	*	1	*

Table 4: Optimal Allocation

There are a total of 24 feasible solutions, as is the number of permutations of $\{0,1\}^{16}$. 4 of these hold the value of 2000, below are the other 3 possible optimal solutions.

	1	2	3	4
1	*	1	*	*
2	1	*	*	*
3	*	*	1	*
4	*	*	*	1
1	1	*	*	*
2	*	1	*	*
3	*	*	*	1
4	*	*	1	*
1	1	*	*	*
2	*	1	*	*
3	*	*	1	*
4	*	*	*	1

Table 5: Other Optimal Allocations

The 4 possible solutions to provide a flow-distance cost of 2000:

Shop 1 → Location 2	Shop 2 → Location 1	Shop 3 → Location 4	Shop 4 → Location 3
Shop 1 → Location 2	Shop 2 → Location 1	Shop 3 → Location 3	Shop 4 → Location 4
Shop 1 → Location 1	Shop 2 → Location 2	Shop 3 → Location 4	Shop 4 → Location 3
Shop 1 → Location 1	Shop 2 → Location 2	Shop 3 → Location 3	Shop 4 → Location 4

It can be seen that the optimal solutions are shop combinations with 2 locations $(1 \in \{1,2\}, 2 \in \{1,2\}, 3 \in \{3,4\}, 4 \in \{3,4\})$

4 Conclusion

The shopping centre owners should use one of the above allocations in their centre. They should assess the details not modelled here, such as size of location or needs or shop, and best apply those details to use one of the solutions provided.

References

KY3 Specification (2019). Keith Yates. URL: `https://moodle.mmu.ac.uk/mod/resource/view.php?id=2296191` (visited on 12/03/2019).

Matlab (2019). Mathworks. URL: `https://www.mathworks.com/products/matlab.html` (visited on 12/03/2019).

A Matlab Code

```matlab
% KY3 Shop Allocation
% Kieran Molloy

clc
clear

% Number of Shops to be modelled
numberOfShops = 4;

% Number of Locations to be modelled
numberOfLocations = 4;

% Array of Common Customers
% Customers of 1&4 = 7000
C =[0 5 2 7;
    5 0 3 8;
    7 8 3 0];

% Get all combinations
N=numberOfShops*numberOfLocations;
% Iterate all logical values
[c{1:N}] = ndgrid(logical([0 1]));
c = cat(N+1,c{N:-1:1});
c = reshape(c,[],N);

% Retrieve length of matrix
[cM,cN] = size(c);

% Tracking Optimal Values
solutionValues = [];
solutionMatrix = [];

for i=1:cM

    % Convert 65536x16 to individual 4x4 problems
    selection = c(i,:);
    selection = reshape(selection,[4,4]);

    % Check for feasibility & add to solutions
    if checkBasicFeasibility(selection) == true
        solutionValues=[solutionValues ; evaluateLocations(C,selection)];
        solutionMatrix=[solutionMatrix ; selection];
    end
```

```
end

% Nicely Format a Print
prettyPrint('selection',solutionValues,solutionMatrix)


function sol=checkBasicFeasibility(x)
    % x = Decision Variable (nxn Array)

    % Constraints to ensure singularity
    locationSingularity = false;
    shopSingularity = false;

    % Location Singularity
    if x(1,1) + x(1,2) + x(1,3) + x(1,4) == 1
        if x(2,1) + x(2,2) + x(2,3) + x(2,4) == 1
            if x(3,1) + x(3,2) + x(3,3) + x(3,4) == 1
                if x(4,1) + x(4,2) + x(4,3) + x(4,4) == 1
                    locationSingularity=true;
                end
            end
        end
    end

    % Shop Singularity
    if x(1,1) + x(2,1) + x(3,1) + x(4,1) == 1
        if x(1,2) + x(2,2) + x(3,2) + x(4,2) == 1
            if x(1,3) + x(2,3) + x(3,3) + x(4,3) == 1
                if x(1,4) + x(2,4) + x(3,4) + x(4,4) == 1
                    shopSingularity=true;
                end
            end
        end
    end
    % Check for both | Return True or False
    if locationSingularity == true && shopSingularity == true
        sol = true;
    else
        sol = false;
    end

end

function totalFlow=evaluateLocations(C,x)
```

```
    % C = Common Customers Matrix (nxn Array)
    % x = Decision Variable (nxn Array)

    % Calculate Shop x -> y  Flow
    flow12=evaluateFlow([1,2],C,x);
    flow13=evaluateFlow([1,3],C,x);
    flow14=evaluateFlow([1,4],C,x);
    flow23=evaluateFlow([2,3],C,x);
    flow24=evaluateFlow([2,4],C,x);
    flow34=evaluateFlow([3,4],C,x);

    totalFlow = flow12 + flow13 + flow14 + flow23 + flow24 + flow34;

end

function flow=evaluateFlow(T,C,x)
    % T = Flow Target [1,2]
    % C = Common Customers Matrix (nxn Array)
    % x = Decision Variable (nxn Array)


    % Array of Distances between Locations
    % Distance between 1 & 4 = 170
    D=[0   80  150  170;
       80   0  130  100;
       150  130  0  120;
       170  100  120  0];

    % Size of Customer Flow Between Location T(1) and T(2)
    flowSize = C(T(1),T(2));

    % Calculate Total Flow of Shops
    flow = flowSize*(D(1,2)*x(1,1)*x(2,2) + D(1,3)*x(1,1)*x(2,3) + D(1,4)*x(1,1)*x(2,4)
    ...
                  +D(2,1)*x(1,2)*x(2,1) + D(2,3)*x(1,2)*x(2,3) + D(2,4)*x(1,2)*x(2,4)
    ...
                  +D(3,1)*x(1,3)*x(2,1) + D(3,2)*x(1,3)*x(2,2) + D(3,4)*x(1,3)*x(2,4)
    ...
                  +D(4,1)*x(1,4)*x(2,1) + D(4,2)*x(1,4)*x(2,2) + D(4,3)*x(1,4)*x(2,3))
    ;

end

function prettyPrint(type,sV,sM)
```

```
    % type - max,min,pathmin,pathmax
    % sV - Solution Values (1xn Array)
    % sM - Solution Matrices (nxnn Array)
    fprintf('\nSolutions!\n')
    fprintf('================\n')
    fprintf('| ID Value |\n')
    switch type
        case 'list'
            %Typesetting
            for i=1:size(sV,1)
                fprintf('| %2i %4i |\n',i,sV(i))
            end
        case 'order-list'
            [sorted,indexes] = sort(sV(:,1));
            % Typesetting
            for i=1:size(sorted,1)
                fprintf('| %%2i  %4i |\n',indexes(i),sorted(i))
            end
        case 'min'
            % Get minimum value & index
            [val,idx]=min(sV);
            % Calculate location in matrix
            mat = sM(4*idx+1:(4*idx)+4,:);
            % Typesetting
            clc
            printSingle(val,mat)
        case 'selection'
            %Typesetting
            for i=1:size(sV,1)
                fprintf('| %2i %4i |\n',i,sV(i))
            end

            % Selection Logic
            menuSel = input('Please select a solution Index: ');
            while menuSel > 0

                % Calculate location in matrix
                mat = sM(4*menuSel+1:(4*menuSel)+4,:);

                printSingle(sV(menuSel),mat)

                menuSel = input('Input 0 for Stop | Input another solution index:  ');
            end

```

```matlab
        case 'pathmin'
            % To be implemented later
        case 'max'
            % To be implemented later
        case 'pathmax'
            % To be implemented later
    end
end

function printSingle(v,M)
    % v - Objective Value
    % M - Decision Variables
    fprintf('\nSolutions!\n')
    fprintf('Value = %5i\n',v)
    fprintf('====================\n')
    fprintf('| ID Value |\n')
    fprintf('====================\n')
    for i=1:size(M,2)
        fprintf('| %2i %2i %2i %2i |\n',M(i,1),M(i,2),M(i,3),M(i,4))
    end
    fprintf('====================\n')
end
```