European Consortium for Mathematics in Industry

28th ECMI Modelling Week
Final Report

19.07.2015—26.07.2015
Lisboa, Portugal

Group 9

Reduce OD matrix dimensions (Railway context)

Anastasia Pavlova
Lappeenranta University of Technology
Mario Cekic
University of Novi Sad
Nathan Eizenberg
University of Oxford
Stephanie Nargang
Technische Universität Dresden
Quanjiang Yu
Chalmers University of Technology
Roman Stasiński
University of Warsaw

Instructor: João Gouveia
CMUC, University of Coimbra
Instructor: Elsa Carvalho
SISCOG

Abstract

Line planning is a classical optimization problem in the design of a public transportation system. It involves the selection of paths in the railway network on which train lines are operated. The aim is to select a set of lines with corresponding operation frequencies, such that a given travel demand can be satisfied. This demand data is usually given for pairs of origins and destinations in a so-called origin-destination matrix (OD matrix).To make it easier to work with real and big instances, a possible approach is to reduce the original dimension of the OD matrix, removing certain stations and adapting demand accordingly. This report presents different methods capable of reducing the OD matrix dimension, given a set of constraints that must be met and an optimization criterion that minimizes the cancelled demand and the transferred demand between stations. The implementations of the various approaches are given and tested with realistic data supplied by SISCOG.

9.1 Introduction

In the modelling of transportation systems, the origin-destination (OD) matrix describes the commuter demand between every pair of destinations in a network. The commuter demand information is used to maximise efficiency of the transport network when developments are made. Within the railway context the OD matrix is particularly important when designing a high-speed train system on top of an existing network. The logistics software company SISCOG designs software systems for major railways networks including the Lisbon metro, London underground and other international networks around the world. A particular problem that SISCOG has proposed is the grouping of stations within a network to reduce commuter demand within a group whilst still partitioning the network into a smaller number of groups. Mathematically, this corresponds to reducing the OD matrix and absorbing the demand between stations into and between the new group structures. This report explores some novel algorithms that reduce any given OD matrix by grouping stations in order to minimise a cost function supplied by SISCOG. We introduce the algorithms, compare them numerically and present their solutions to the OD matrix reduction problem as tested on four railway networks.

In the first section of this report we formulate the problem and introduce the relevant mathematical background. In the next section we describe our naive approaches to the problem, starting with algorithms that make greedy choices at each iteration. We then outline some more sophisticated variations that employ approximations in order to reduce their computational cost. In Section 9.3 we describe a different approach that utilises similar solutions to the well known Facility Location problem to approximately solve our problem. In Section 9.4 we present numerical results for each of our algorithms' performance in time and optimal solution cost function value. In Section 9.5 we discuss methods of post-processing that aim to improve near optimal solutions and to determine if the solutions are local or global. In the final section we discuss possible developments for the project.

Background

For the purposes of line-planning and partitioning of a railway network, the information can be represented by an *adjacency matrix* and an *origin-demand* (OD) *matrix*. An *adjacency matrix* D, contains distance information between adjacent nodes so that

$$[D]_{i,j} = \begin{cases} e_{i,j} & \text{stations share a direct edge} \\ 0 & \text{otherwise} \end{cases}, \tag{9.1}$$

where $e_{i,j}$ is the edge weight between stations i and j. The edge weight between two stations can be considered to be the direct distance between

them. In this report we only consider simple, symmetric networks where $e_{i,j} = e_{j,i}$ and $e_{i,i} = 0$ for all stations i and j, meaning that the distance between adjacent stations is identical in either direction and train journeys that start and end at the same station. The adjacency matrix can be represented as a graph by taking the edge weights from (9.1). Figure 9.1 shows a graph that contains the information of the adjacency matrix for an example 'Toy' network with 8 stations.

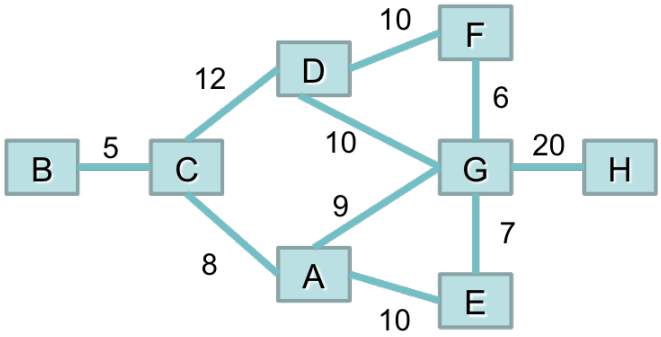


Figure 9.1: Example graph for the 'Toy' network

The OD matrix contains the demand information of the network, capturing the popularity of journeys between any two stations at peak hour. Elements of the OD matrix are defined as

$$[OD]_{i,j} = \text{peak demand } \textit{from} \text{ station } i \textit{ to} \text{ station } j \tag{9.2}$$

, for all stations i and j. In practice, OD matrices are often approximated from real measurements of customer flow and are used in most traffic and transport modelling. An example OD matrix for the 'Toy' example network is shown in Figure 9.2. By coupling the information of the network structure given by the adjacency matrix with the demand structure given by the OD matrix we can make a complete picture of customer flow on the train network.

OD matrix reduction

It is sometimes important in application to have the full OD matrix in order to understand how traffic flows between stations. On the larger scale, however, it is often more enlightening to group stations together into *zones* so that traffic flow can be understood at a courser scale. This grouping of stations, hereafter called *zoning*, is equivalent to collapsing the OD matrix where the number of rows and columns are now the number of zones, and is the subject of this report. For example, by introducing 5 zones in the 'Toy' example network, we can reduce the OD matrix to as shown in Figure

	A	B	C	D	E	F	G	H
A	0	2	13	4	5	6	17	11
B	2	0	20	7	2	3	14	19
C	13	20	0	15	16	19	25	28
D	4	7	15	0	4	8	19	12
E	5	2	16	4	0	1	12	16
F	6	3	19	8	1	0	19	15
G	17	14	25	19	12	19	0	23
H	11	19	28	12	16	15	23	0

Figure 9.2: Origin-demand (OD) matrix for the 'Toy' network. Elements $[OD]_{i,j}$ give the customer demand for the journey from station i to station j. In this case, the OD matrix is symmetrical meaning that demand is the same in either direction.

9.3. The stations B and C have been merged into a zone and the stations E, F and G belong to a zone together, otherwise all the rest are solo station zones.

	A	BC	D	EFG	H
A		15	4	28	11
BC	15	40	22	79	47
D	4	22		4	2
EFG	28	79	4	64	54
H	11	47	2	54	

Figure 9.3: Reduced OD matrix for the 'Toy' network. The columns and rows now represent different zones containing one or more stations. While still symmetrical, there are now nonzero diagonal elements. These diagonal values represent demand within a zone.

Zones are particularly important when planning a high speed train system on top of an existing network. In this case, key stations for each zone are interconnected by a high speed train line, while stations within a zone are connected by the regular train services. For such applications, the zones should be chosen to best suit customer demand on the network. The key stations that represent each zone also need to be chosen carefully. The company SISCOG provided us with a model which they had developed to formulate this zoning problem as a discrete optimisation problem.

SISCOG model

The model proposed by SISCOG aims to group stations in order to minimises the amount of travel within a zone, preferring travel between zones. In the context of high-speed trains, this approach is very reasonable: a customer wanting to travel between two small towns a and b both near larger cities A and B respectively, should prefer a journey

$$a \to A \to B \to b, \tag{9.3}$$

where the cities A and B are serviced by a high-speed train and the smaller towns a and b are only serviced by the regular train network. Minimising the traffic on regular train routes can increase network efficiency and can also mean that customers spend less time travelling. The model defines two different types of travel: *transfer* and *inner*. Transfer travel accounts for journeys between zones and inner travel for transfer within a zone. In Figure 9.4 the route in orange between stations B and C is an inner travel route because the two stations are within the same zone, the route between stations G and H, however, is a transfer route because they are not in the same zone. In the SISCOG model, these two types of travelling are considered to be a cost to the customer that we aim to minimise. We define the cost of travel between two stations as the product of distance and demand for any two stations i and j. The model also introduces the concept of a *representative station* - a station that represents an entire zone. Transfer travel can only occur between two representative stations in two different zones. Figure 9.4 shows a 5 zone solution for the 'Toy' example network where the zones are grouped by ovals superimposed on the network. Here, the representative stations A,C,D,G and H are highlighted in green. The consequence of this zone structure is that any journey between, say, stations E and A must go through the representative station, G.

In the SISCOG model the distance is taken as the weighted product of the demand and the shortest path between two stations. The cost between station i and j is therefore defined as

$$C(i,j) = \text{shortestPath}(i,j)\,[OD]_{i,j}\,, \tag{9.4}$$

for any two stations.

However the weights for the costs are different depending on the zone structure: if i and j are in the same zone then the inner cost is

$$C_{inner}(i,j) = 0.9C(i,j), \tag{9.5}$$

and if i and j are in different zones with representative stations i^* and j^* respectively, then the transfer cost is

$$C_{transfer}(i,j) = 0.9\,(C(i,i^*) + C(j^*,j)) + 0.1C(i^*,j^*), \tag{9.6}$$

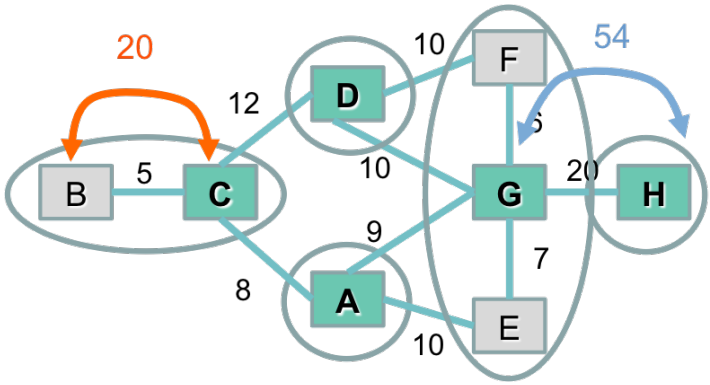


Figure 9.4: A 5 zone solution to the 'Toy' example network. The representative stations are highlighted in green. Two examples of travel cost are superimposed between stations B and C and G and H and correspond to the one way origin-demand elements of the reduced OD matrix taken from 9.3.

for all i, j and all zones. The total cost is therefore

$$C_{total} = \sum_{i,j} \sum_{\text{zones}} (C_{inner}(i,j) + C_{transfer}(i,j)) \tag{9.7}$$

so the travel within a zone is 9 times more costly than travel between zones. This reflects the importance of minimising customer travel within the same zone, preferring but still minimising travel between zones.

Goals and Challenges

The goal for this project was to develop an algorithm that could reduce any train network into a set of zones that minimises the cost function (9.7). We were asked not to use any commercial linear programming software and that the algorithm has fast execution times. In particular we were asked to produce zoning solutions and cost function values for four test networks each with n stations:

1. 'Toy' network $n = 8$
2. Beijing–Shanghai intercity network $n = 28$
3. Washington DC metro $n = 88$
4. 'Big' network $n = 884$

These networks provided an adequate framework to test algorithms however, without multiple networks with the same number of stations to test each algorithm, our analysis lacked robustness. The main challenge in this report was producing an algorithm that could produce solutions in a short time for the larger networks.

9.2 Greedy Approaches

Naive approach

Our first approach utilised simple and greedy zoning mechanisms. The code may be found in `coregreediestR.m` file in the Appendix in Section .1. We start with a situation when all stations belong to separate zones. In each iteration of the main loop we compute the increase in cost function when merging all possible pairs of zones and choose the pair with the smallest increase.

To reduce the computational cost of this algorithm we reformulated the formula for the cost of zoning. Firstly we symmetrise the OD matrix

$$\mathrm{OD} := \left(\mathrm{OD} + \mathrm{OD}^T\right) \cdot 0.5$$

and observe that the cost of zoning does not change. Then we observe that we may rewrite the formula for the cost in the following way:

$$2 \sum_{\text{zones}} \sum_{\substack{\text{members} \\ \text{of zone}}} \Bigg(0.8 \cdot \text{shortestPath}\left({}^{\text{representant}}_{\text{of zone}}, {}^{\text{member}}_{\text{of zone}}\right) \cdot {}^{\text{demand from member}}_{\text{to stations within zone}} + \\ + 0.1 \cdot \text{shortestPath}\left({}^{\text{representant}}_{\text{of zone}}, {}^{\text{member}}_{\text{of zone}}\right) \cdot {}^{\text{total demand}}_{\text{from member}} \Bigg)$$

which is equal to

$$\sum_{A \in \text{zones}} \text{partialCost}(A)$$

where $\text{partialCost}(A)$ is equal to

$$\sum_{\substack{\text{members} \\ \text{of zone } A}} \Bigg(2 \cdot 0.8 \cdot \text{shortestPath}\left({}^{\text{representant}}_{\text{of zone}}, {}^{\text{member}}_{\text{of zone}}\right) \cdot {}^{\text{demand from member}}_{\text{to stations within zone}} + \\ + 0.1 \cdot \text{shortestPath}\left({}^{\text{representant}}_{\text{of zone}}, {}^{\text{member}}_{\text{of zone}}\right) \cdot {}^{\text{total demand}}_{\text{from and to member}} \Bigg) \tag{9.8}$$

In the above, we have grouped together the total demand *to* and *from* each station in the last term (explaining the loss of a factor of 2). This way of evaluating cost has been implemented in the `Runner2.m` file and showed significant improvements to the computational time. Using this formulation of the cost function we have also written the MATLAB function `bestCentralStation.m` which, given a set of zones, finds the optimal representative station. The formula (9.8) justifies computing the partial cost of each zone separately and allows us to compute the change in cost of joining zones A and B as

$$\text{partialCost}(A \cup B) - \text{partialCost}(A) - \text{partialCost}(B).$$

Despite the improved method of computing the cost, this algorithm performed too slowly even in moderately large cases (see Section 9.4), therefore in the further work we introduced more sophisticated approaches.

Alternative greedy approaches: 'Full Greedy' and 'Partial Greedy'

The main idea behind the following two algorithms is to choose a pair of stations, merge the demand between them (change the OD matrix) and then remove one station from the data set. Of the two stations, a representative station is chosen to absorb the other and its demand, and the non-representative is then removed. We pick the pairs according to the rule that we want to minimise the following function – cost of merging station j to station i equal to

$$\begin{aligned}\text{cost}(i,j) = \text{totalDemand}(j) \cdot \text{shortestPath}(i,j) \cdot 0.1 + \\ +2 \cdot 0.8 \cdot \text{OD}(i,j) \cdot \text{shortestPath}(i,j),\end{aligned}$$

where totalDemand stands for total demand from and to a station and OD is the origin-destination matrix. The formula above is a simplified version of (9.8). What is important is that we do not chose optimal representatives at each iteration – instead do it only once at the end using the `bestCentralStation.m` function. The basic version of this algorithm is implemented in the `mergeStation.m` file and is referred to as *Full Greedy* later on.

Less expensive version (found in the `mergeStation2.m` file) where instead of checking *all* pairs of stations we chose from a smaller set of close stations. We introduce notions of distance between stations as a product of demand and shortest path, then for each station we look for the nearest one (considering this distance) and these pairs form the set that we check the costs for. This reduces the complexity of the algorithm and performs much faster. We refer to this algorithm as the *Partial Greedy* algorithm due to its use of the partial cost equation and greedy zoning choice mechanism.

Low memory greedy algorithm

Another variant approach is bases on determining representatives *before* any station merging has occurred. Given a specified reduction factor, the number of representatives can be calculated and identified *a priori*, and then the rest of the stations can be distributed thereafter.

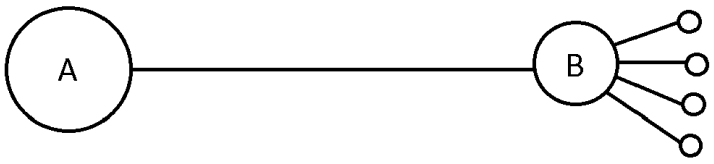
In addition, this approach accounts for stations which do not have any demand to or from them. These unpopular stations can be assigned to the nearest station immediately, thus reducing the size of the problem from the onset and saving on computation. Furthermore, this immediate pairing of unpopular stations has no influence on the cost function. For example,

if there are enough stations with zero demand to satisfy a given network reduction factor then the entire network reduction can be performed without an any analysis of stations. In this case, we are able to fulfil the requirement by merging stations with zero demand to the predetermined representatives with almost no calculation of the cost function. However, when the number of stations with zero demand do not match the reduction factor, further analysis is required.

Determining the candidates for representative stations requires estimating the total demand of every station by

$$\text{TotalDemand(i)} = \sum_j [OD]_{i,j} + [OD]_{j,i}, \tag{9.9}$$

for any station i and j. We assume that representatives should have the largest total demand with respect to distances to other stations. It can be explained as follows. Assigning the station with the considerable demand leads to great increasing the cost function since the whole demand of the station will be transferred to another station. In addition, we suggest taking into account the distances between stations. Let us introduce an example in order to clarify out suggestions. The example is illustrated in Figure 9.2. We specify one representative station, A, and attach one other station, B, to it so that TotalDemand(A) $>$ TotalDemand(B). However, all demand from other stations to station B is then absorbed into station A, which is further away, and because they are in the same zone, contribute considerable increasing of the cost function. Therefore, if the cost to merge stations is high we suggest attaching all stations to the station B, despite it not having the largest total demand.



Next, we specify the stations which should be attached to the representatives. It is worth mentioning that here we do not take into account stations with zero demand because they can already be assigned. The number of candidates is equal to $(l - m - h)$, where m is a number of representatives, l is the total number of stations and h is the number of stations with zero demand. After that, we estimate the best pairs to be candidates for merging.

We determine the cost of merging the station i to station j as following:

$$C_{i,j} = 0.1 \cdot \text{TotalDemand(i)} \cdot \text{ShortestPath}(i,j) + \\ +2 \cdot 0.8 \cdot \text{demand(i,j)} \cdot \text{dist(i,j)},$$

and the pair with the lowest cost is optimal. The advantage of this current approach is that since we already specified all representatives and all candidates for merging we only have to decide to which representatives every residual station should be attached. In other words, we go through all candidates and decide to which representative each station can be optimally paired. This approach uses less memory and results in faster performance because we do not calculate all possible merging pairs as in other algorithms. The current approach is similar to the previous greedy algorithms, however it has considerable advantages being faster and uses less memory.

9.3 Approximation Approach

In the previous chapters we have shown some ways of constructing the zones step-by-step. In this chapter, however, we are going to try to reduce the problem of zone selection to a more familiar one, p-median facility location problem (FLP).

The FLP is well-known in the field of operational research. The main aim is to optimise the positions of facilities - which, for example, may be storage depots for large supermarket chains - in order to minimise transportation costs between the depots while considering additional constraints. In particular, the p-median FLP deals with the optimal placement of p facilities in order to minimise the demand-weighted average distance between demand nodes and the nearest of the selected facilities.

The SISCOG problem relies on minimising (9.7) where the inner cost is highly penalised and the transfer cost, relatively discounted. However, in the canonical case of the p-median FLP the cost function does not make a distinction between transfer costs, giving the relation

$$C_{total} = 0.5\, C_{transfer} + 0.5\, C_{inner}, \tag{9.10}$$

so the transfer and inner sub-costs have the same weight.

Although the FLP is generally NP-hard, there are numerous heuristics which provide efficient solutions to this problem. While the cost function for the FLP is very different from the SISCOG cost function in (9.7), the hope is that a solution to (9.10) will be close to a solution to the SISCOG problem. We therefore posed the SISCOG problem as a typical FLP by choosing to assign as many stations as possible to p zones so that the demand-weighted distance (but equally weighting the distance within a zone and between zones) is minimised. We then applied a well developed p-median FLP solver,

PMCLUSTER[1] which uses a *simulated annealing* method to find optimal solutions.

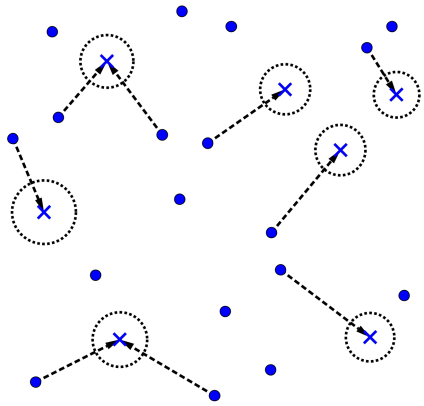


Figure 9.5: Zone representatives obtained using p-median facility location approach.

Using this tool, we obtain the zones with their representatives (figure 9.5) for some of the stations. The next step is to assign the rest of the stations to the p zones. We then utilise the simple greedy approach and assign the residual stations to their closest representative and zone in terms of demand weighted distance (cost C_{ij} in 9.4), giving the zone structure for all stations in the network (figures 9.5 and 9.6).

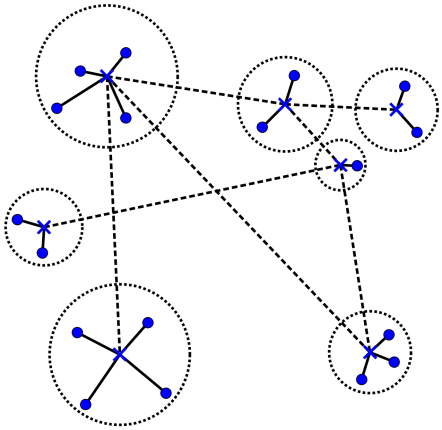


Figure 9.6: Zoning after assigning all the stations optimal representatives.

The results obtained using this approach seem to provide fairly good

[1]This tool has been developed by the Michael J. Brusco from Florida State University and represents a collection of MATLAB programs for p-median clustering.

solutions (as shown in chapter 9.4). Moreover, this approach can be used as a good starting point for further improvements, which will be discussed later.

9.4 Results

We implemented all the algorithms described above. Each algorithm was tested with four different examples from SISCOG and with different reduction percentages.
The following table in Figure 9.7 shows the results of the different algorithms. For each algorithm there are two columns, one column contains the costs of the best result and one column contains the time which was needed to calculate this result.
Unfortunately, the 'Big' example was too large to compute results using the naive greedy algorithm.
All in all, the greedy singleton merging algorithm performed the best, in matters of cost function, in very reasonable time. The full greedy algorithm also performed very well. Compared to the full greedy algorithm, the partial greedy algorithm performed faster but had slightly worse cost function values. Our first naive approach was only applicable to small examples, taking too long to compute solutions for the 'Big' example.

	% reduced	Full greedy	t	Full greedy variation	t	Partial greedy	t	Naive	t	Low Memory Greedy	t
TOY	62.5	1.7846e+03	0.001129			1.7846e+03	0.003113	1.7846e+03	0.014495	1.7846e+03	0.0048
	45	772	0.000937			772	0.003479	693.6000	0.009784	694.2000	0.0020
BEIJING	90	2.0595e+06	0.006211			2.0595e+06	0.004867	2.5819e+06	0.442622	5.9544e+06	0.0067
	70	4.2824e+05	0.004052			4.2824e+05	0.005138	3.7918e+05	0.396549	7.3804e+05	0.0020
	50	9.8181e+04	0.003291			9.8181e+04	0.006974	9.8181e+04	0.330634	1.3998e+05	0.0017
	30	3.0768e+04	0.002427			3.0768e+04	0.004410	3.0768e+04	0.192996	5.1109e+04	0.0016
WASHINGTON	90	1.1704e+05	0.058753			1.4175e+05	0.039880	1.0042e+05	27.455203	3.9042e+05	0.0065
	70	1.7292e+04	0.052929			3.8323e+04	0.032093	1.7080e+04	23.830571	3.7813e+04	0.0061
	50	6.9429e+03	0.044008			9.8164e+03	0.026415	6.7437e+03	18.262172	1.1953e+04	0.0047
	30	2.1847e+03	0.032556			2.4824e+03	0.018924	2.1786e+03	12.394535	3.5715e+03	0.0041
BIG	90	3.7623e+08	55.823449			14.864e+08	13.362606	??	??	4.6042e+08	0.1456
	70	9.1184e+06	54.771955			2.1439e+08	12.252310	??	??	1.4787e+07	0.1477
	50	0	47.303236			0	10.739585	0	??	0	0.0238
	30	0	33.711365			0	8.184871	0	??	0	0.0205

Figure 9.7: Table of computation results

9.5 Improving the Results

In this chapter we will discuss the possible additional improvements of the solutions obtained by the explained approaches. All of the approaches outlined in this report use station zoning heuristics that attempt to provide us with optimal solutions. Sometimes these solutions are not stable and a very small change in zoning structure gives a more optimal solution. In this section, we consider small perturbations in zoning and their impact on the cost function.

We considered two possible perturbations, *flipping* and *shuffling*. The first type, flipping, represents swapping two randomly chosen, non-representative, stations from separate zones, also chosen at random (Figure 9.8).

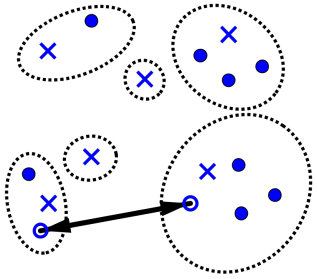


Figure 9.8: Station flipping.

The other perturbation considered: shuffling, is accomplished by choosing a non-representative station at random from a random zone and moving it to another zone, also chosen at random (Figure 9.9).

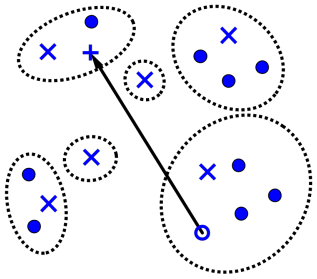


Figure 9.9: Station shuffling.

Comment on results These perturbations were implemented on previously computed solutions on all four network examples to give no significant improvement of the cost function. Given thousands of single perturbations generated randomly, the optimal cost function value did not improve significantly. Further experimentation would be valuable, however, due to time

restraints these preliminary results allowed us to conclude that the zoning solutions produced by our algorithms could not be significantly improved by either 'flipping' or 'shuffling' of stations. We suggest that this is due to a local stability of the obtained solutions.

9.6 Outlook and possible future work

Due to time constraints it was not possible to develop the algorithms or post-processing method any further, however, we end this report with some suggestions for further directions and improvements. Firstly, by constructing ensembles of random networks for any given size and testing each algorithm on all of them would give more robust statistical insight into their performance. With only a single network for each size, as included in this report, the results are susceptible to being influenced by network characteristics instead of algorithm performance. We suggest performing a similar analysis as outlined in this report on ensembles of random networks.

Secondly, after encouraging results from the simulated annealing inspired post-processing method, we suggest developing these methods to try and find global solutions to the OD matrix reduction problem. We found that initial solutions, given by each algorithm, were rarely improved dramatically by small perturbations. This implies that the solutions are local minima of the cost function. We suggest developing a method that perturbs local solutions intermittently to try and ensure that the final solution is close to the global minimum. Ideally, instead of post-processing being conducted once a solution is given by one of the primary algorithms, we could apply perturbations after every few iterations.

9.7 Conclusions

In this report we explored the origin-demand matrix reduction problem as applied to railway network planning as proposed by the logistics software company SISCOG. We presented novel algorithms that reduce any given network into groups of stations while optimising efficiency for commuters travelling between these groups. We presented results of each algorithm tested on four sets of railway networks, either real or realistically inspired. We also presented methods to improve solutions by locally perturbing sets of groups of stations in an aim to further reduce the given cost function. Finally, we describe further directions for this project that may improve our algorithms to provide near global solutions to the SISCOG problem. In conclusion, we have presented algorithms that meet the original aims of this project - taking a given network and providing a set of zones that minimises commuter disruption - giving an optimal reduced OD matrix.

.1 Appendix

Runner

```
function cost = Runner2(OD, weight, zones,
    representatives, shortestPath)
%
%This is our version of evaluator
%
        OD = (OD + OD').*0.5;
        numStations = size(OD, 1);
        totalDemand = zeros(numStations,1);
        for i = 1 : numStations
            totalDemand(i) = sum(OD(:,i)) + sum(OD(i,:))
                ;
        end
        cost = 0;
        for i = 1 : length(representatives)
              membersOfZone = find(zones(i,:)==1);
              for k = 1 : length(membersOfZone)
                  inCost = shortestPath(representatives(i
                      ),membersOfZone(k)) * sum(OD(
                      membersOfZone(k),membersOfZone));
                  outCost = shortestPath(representatives(
                      i),membersOfZone(k)) * totalDemand(
                      membersOfZone(k));
                  cost = cost + (4*weight - 2) * inCost +
                       (1 - weight) * outCost;
              end
        end
end
```

Best Central Station

```
function [representatives, cost] = bestCentralStation(
    mergedStations, totalDemand, OD, shortestPath)
%
%This function may be used to find optimal
    representative stations when we know the zoning
%
     representatives = zeros(size(mergedStations,1),1);
     %symmetrization of the OD matrix simplifies the
         algorithm
     OD = (OD + OD').*0.5;
     % loop for the number of zones, for each zone we
         pick the representative station separately
     cost = 0;
     for i = 1 : length(representatives)
        membersOfZone = find(mergedStations(i,:) == 1);
        costs = zeros(length(membersOfZone),1);
        % loop for each station in one zone
        for j = 1 : length(membersOfZone)
            % loop for the summing of the cost
            for k = 1 : length(membersOfZone)
                inCost = shortestPath(membersOfZone(j),
                    membersOfZone(k)) * sum(OD(
                    membersOfZone(k),membersOfZone));
                outCost = shortestPath(membersOfZone(j)
                    ,membersOfZone(k)) * totalDemand(
                    membersOfZone(k));
                costs(j) = costs(j) + 2 * 0.8 * inCost
                    + 0.1 * outCost;
           end
        end
        [M,I] = min(costs);
        cost = cost +M;
        representatives(i) = membersOfZone(I);
     end
     cost;
end
```

Naive Greedy

```
function [zones, representatives, cost] =
    NaiveGreediest(inputFileName, reductionPercentage)

    %loading OD, trainNetwork, shortestPath, UniqueSt
    load(strcat('input/', inputFileName));

    %just changing name
    network = trainNetwork;

    numStations = size(OD, 1);

    %symmetrization needed for easier formulation of
        the problem
    OD = (OD + OD').*0.5;

    %additional varible needed for simplicity of the
        code - total demand from and to station
    totalDemand = zeros(numStations,1);
    for i = 1 : numStations
        totalDemand(i) = sum(OD(:,i)) + sum(OD(i,:));
    end

% Initialise zone arrays, all stations belong to
    separete zones
rep = (1:numStations)';
zones = speye(numStations);

% Calculate final number of zones to be reduced
m = floor(numStations * reductionPercentage);

%itinial partial cost of zones
initialPartialCost = zeros(length(rep),1);
for i = 1:length(rep)
    membersOfZone = find(zones(i,:)==1);
    C_1 = 0;
    for k = 1 : length(membersOfZone)
        inCost = shortestPath(rep(i),membersOfZone(k))
            * sum(OD(membersOfZone(k),membersOfZone));
        outCost = shortestPath(rep(i),membersOfZone(k))
            * totalDemand(membersOfZone(k));
        C_1 = C_1 + 2 * 0.8 * inCost + 0.1 * outCost;
    end
```

```
    initialPartialCost(i) = C_1;
end

% Start loop to combine m zones
for j = 1:m
    %r = randi([1, length(rep)], 1);      % Chose
        uniform random station to start at
    C_0 = Inf;                            % Initialise
        objective


    %double loop for all possible zone combinations
    for r = 1:length(rep)
        z_temp = r+1 : length(rep);
        for i = z_temp

            % Store temporary zone arrays
            z_1 = zones;

            % Change temporary zone array to combine
                zone r with zone i
            z_1(r,:) = z_1(r,:) + z_1(i,:);
            z_1(i,:) = [];

            % Now set up loop to find optimal
                representative station
            rep_set = [find(zones(r,:)==1), find(zones
                (i,:)==1)];

            % Loop over all rep. stations
            for ii = rep_set
                rep_1 = rep; rep_1(r) = ii; rep_1(i) =
                    [];
                membersOfZone = rep_set;
                C_1 = 0;
                for k = 1 : length(membersOfZone)
                    inCost = shortestPath(ii,
                        membersOfZone(k)) * sum(OD(
                        membersOfZone(k),membersOfZone))
                        ;
                    outCost = shortestPath(ii,
                        membersOfZone(k)) * totalDemand(
                        membersOfZone(k));
```

```
                    C_1 = C_1 + 2 * 0.8 * inCost + 0.1
                        * outCost;
                end

                %newCostOfMergedZones is the partial
                    cost of new zone obtained by
                    merging zones r and i
                newCostOfMergedZones = C_1;
                C_1 = C_1 - initialPartialCost(r) -
                    initialPartialCost(i);

                % Store rep. station and test station
                    that improves the cost
                % function
                if C_1 < C_0
                    C_0 = C_1;

                    %merged zones
                    i_0 = i;
                    r_0 = r;

                    %optimal representant
                    ii_0= ii;

                    %new partial cost
                    newCostOfMergedZones_0 =
                        newCostOfMergedZones;
                end
            end
        end
        % Update the zone array
    end
    zones(r_0,:) = zones(r_0,:) + zones(i_0,:);
    zones(i_0,:) = [];

    rep(r_0) = ii_0;
    rep(i_0) = [];

    initialPartialCost(r_0) = newCostOfMergedZones_0;
    initialPartialCost(i_0) = [];
end
    representatives = rep;
    load(strcat('input/', inputFileName));
    %we run our evaluator
```

```
    cost = Runner2(OD, 0.9, zones, representatives,
        shortestPath);
end
```

Full Greedy

```
function [mergedStations, representatives, cost] =
    FullGreedy(inputFileName, reductionPercentage) %
    mergeStation

    %loading OD, trainNetwork, shortestPath, UniqueSt
    load(strcat('input/', inputFileName));

    %just changing name
    network = trainNetwork;

    %we symmetrize
    OD = (OD + OD').*0.5;

    %we need later original data
    originalOD = OD;
    originalShortestPath = shortestPath;

    numStations = size(OD, 1);

    %number of iterations
    m = floor(numStations * reductionPercentage);

    %at the beginning each station belongs to separete
        zone
    mergeStations = speye(numStations, numStations);

    totalDemand = zeros(numStations,1);
    for i = 1 : numStations
        totalDemand(i) = sum(OD(:,i)) + sum(OD(i,:));
    end
    originalTotalDemand = totalDemand;

    for i = 1 : m
        %we keep there there the approximated cost of
            merging all possible pairs of zoning,
        %note that it is not symmetric!
        cost = zeros(numStations, numStations);

        for i = 1 : numStations
            for j = 1 : numStations
                if i == j
                    cost(i,j) = inf;
```

```
                else
                    cost(i,j) = totalDemand(j) *
                       shortestPath(i,j) * 0.1 + 2 *
                       0.8 * OD(i,j) * shortestPath(i,
                       j);
                end
            end
        end

        [M1,I1] = min(cost);
        [M2,I2] = min(M1);
        j = I2; %column index for the minimal entry
        i = I1(I2); %row index for the minimal entry

        mergeStations(i,:) = mergeStations(i,:) +
           mergeStations(j,:);
        mergeStations(j,:) = [];

        shortestPath(j,:)= [];
        shortestPath(:,j) = [];

        totalDemand(i,:) = totalDemand(i,:) +
           totalDemand(j,:);
        totalDemand(j,:) = [];

        OD(i,:) = OD(i,:) + OD(j,:);
        OD(:,i) = OD(:,i) + OD(:,j);
        OD(j,:) = [];
        OD(:,j) = [];

        numStations = numStations - 1;
    end
    mergedStations = mergeStations;

    %we find optimal main stations
    [representatives, cost] = bestCentralStation(
       mergedStations, originalTotalDemand, originalOD
       , originalShortestPath);
end
```

Partial Greedy

```
function [mergedStations,representatives, cost] =
    PartialGreedy(inputFileName, reductionPercentage) %
    mergeStation2

    %loading OD, trainNetwork, shortestPath, UniqueSt
    load(strcat('input/', inputFileName));

    %just changing name
    network = trainNetwork;

    %we symmetrize
    OD = (OD + OD').*0.5;

    %we need later original data
    originalOD = OD;
    originalShortestPath = shortestPath;

    numStations = size(OD, 1);

    %number of iterations
    m = floor(numStations * reductionPercentage);

    %Initialise zone arrays
    mergeStations = speye(numStations, numStations);

    %'metric' introduced on pairs of stations to find
        the 'closest' ones
    partialCost = OD .* shortestPath;
    for i = 1:size(partialCost, 1)
        partialCost(i,i) = inf;
    end

    %total demand from and to station
    totalDemand = zeros(numStations,1);
    for i = 1 : numStations
        totalDemand(i) = sum(OD(:,i)) + sum(OD(i,:));
    end
    originalTotalDemand = totalDemand;

    %main loop
    for i = 1 : m
```

```
        %for each station we find the 'closest' one
            according to the metric above
        [C,I1] = min(partialCost);
        cost = zeros(numStations, 1);

        for i = 1 : numStations
            %cost of merging station I1(i) to station
                i
            cost(i) = 0.1 * totalDemand(I1(i)) *
                shortestPath(i,I1(i)) + 2 * 0.8 * OD(i,
                I1(i)) * shortestPath(i,I1(i));
        end
        [M,I2] = min(cost);
        i = I2; %column index for the minimal entry
        j = I1(i); %row index for the minimal entry

        %update the data
        mergeStations(i,:) = mergeStations(i,:) +
            mergeStations(j,:);
        mergeStations(j,:) = [];

        shortestPath(j,:) = [];
        shortestPath(:,j) = [];

        totalDemand(i,:) = totalDemand(i,:) +
            totalDemand(j,:);
        totalDemand(j,:) = [];

        OD(i,:) = OD(i,:) + OD(j,:);
        OD(:,i) = OD(:,i) + OD(:,j);
        OD(j,:) = [];
        OD(:,j) = [];

        partialCost(i,:) = partialCost(i,:) +
            partialCost(j,:);
        partialCost(:,i) = partialCost(:,i) +
            partialCost(:,j);
        partialCost(j,:) = [];
        partialCost(:,j) = [];
        numStations = numStations - 1;
    end
    mergedStations = mergeStations;
    %find optimal representant stations for the zoning
```

```
    [representatives, cost] = bestCentralStation(
        mergedStations, originalTotalDemand, originalOD
        , originalShortestPath);
end
```

Main

```
%clc; clear;
load('Big_input.mat');
load('Big_shortest_path');
demand = OD;
dist = shortestPath;

% demand = rand(5);
% dist = rand(5);
% n = 8;

tic;
candidates = 0; %how many candidates we want to use to
    reduce the OD matrix
P = 90;
n = floor(P*size(demand,1)/100);% number of stations
   we want to remove
[zones,representatives ] = create_zonesFINAL(demand,
   dist,n );
time = toc;
time
% zones;
%
%create zones in matrix notation
%case without zeros rows
z = sparse(length(representatives),size(demand,2));
for i = 1:length(zones)
    ind_i = find(zones{i}(end) == representatives,1,'
       first');
    ind_j = zones{i};
    z(ind_i,ind_j) = 1;
end
%sum(sum(z))
%
%sum(sum(z(:,representatives)))
%cost = Runner(demand,network,dist,0.9,z,
   representatives) % 0.9 is weightcost
%cost = Runner(demand,network,0.9,z,representatives,
   dist) % 0.9 is weightcost
dist(dist==inf)=0;
cost2=Runner2(demand, 0.9, z, representatives, dist)
```

Create Zones

```
function [zones,heads] = create_zonesFINAL(demand,dist
    ,n)

%weight includes total demand of every station
num_stations = size(demand,1);
zones = num2cell(1:num_stations); %initial zones
n_heads=num_stations-n; % number of representatives

%calculate the total demand of every city
total_demand=sum(demand,2)+sum(demand,1)';
% part for elimination of zero demand cities.
zero_demand_cities=find(total_demand==0);


if length(zero_demand_cities)>=n %it means we can
    reduce matrix by just removing zero cost cities
     heads = 1:num_stations;
     dist(zero_demand_cities,zero_demand_cities)=inf;
     for i=1:n
         zones{zero_demand_cities(i)} = [];
         [~,ind] = min(dist(zero_demand_cities(i),:));
         zones{ind} = [zero_demand_cities(i) zones{ind
             }];
     end

     %remove empty zones
     zones(cellfun(@isempty,zones)) = [];
     %remove empty representatives
     heads(zero_demand_cities(1:n)) = [];
     return;
end

%estimate weight of cities to be representative
total_weight=zeros(size(total_demand));
for i=1:length(total_demand)
     temp_dist=sort(dist(i,dist(i,:)~=inf));
     nnn=length(dist(i,dist(i,:)~=inf));
     %total_weight(i)=total_demand(i)*median(dist(i,
         dist(i,:)~=inf));
     total_weight(i)=total_demand(i) * median(temp_dist
         (1:round(nnn*2/5)));
end
```

```
[~,Ind_heads] = sort(total_weight,'descend');



%stations which can be representatives
heads=sort(Ind_heads(1:n_heads));

%stations which can be merged with representatives
tails = 1:num_stations;
tails([heads; zero_demand_cities])=[];

%attach every tail to the best representative
for i = tails
    value = total_demand(i)*dist(i,heads)*0.1 + 2*
        demand(i,heads).*dist(i,heads)*0.8; %current
        cost of adding the ith station to the jth
        station
    [~,Ind] = min(value);
    zones{i} = [];
    zones{heads(Ind)} = [i zones{heads(Ind)}];
end


% attach cities with zero demand
for i=1:length(zero_demand_cities)
    zones{zero_demand_cities(i)} = [];
    [~,ind] = min(dist(zero_demand_cities(i),heads));
    zones{heads(ind)} = [zero_demand_cities(i) zones{
        heads(ind)}];
end

%remove empty zones
zones(cellfun(@isempty,zones)) = [];

end
```