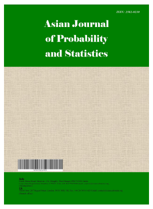

Asian Journal of Probability and Statistics

7(3): 28-57, 2020; Article no.AJPAS.55184

ISSN: 2582-0230

The Maximum Flow and Minimum Cost–Maximum Flow Problems: Computing and Applications

W. H. Moolman[1*]

[1]*Department of Mathematical Sciences and Computing, Walter Sisulu University, Mthatha, South Africa.*

Author's contribution

The sole author designed, analysed, interpreted and prepared the manuscript.

Article Information

DOI: 10.9734/AJPAS/2020/v7i330185
Editor(s):
(1) Dr. S. M. Aqil Burney, University of Karachi, Pakistan.
Reviewers:
(1) Tajini Reda, National School of Mineral Industry, Morocco.
(2) Muhammad Kamal Amjad, National University of Sciences and Technology, Pakistan.
(3) Nalini V. Vaidya, G. H. Raisoni College of Engineering, India.
Complete Peer review History: http://www.sdiarticle4.com/review-history/55184

Received: 08 January 2020
Accepted: 12 March 2020
Published: 20 June 2020

Review Article

Abstract

The maximum flow and minimum cost-maximum flow problems are both concerned with determining flows through a network between a source and a destination. Both these problems can be formulated as linear programming problems. When given information about a network (network flow diagram, capacities, costs), computing enables one to arrive at a solution to the problem. Once the solution becomes available, it has to be applied to a real world problem. The use of the following computer software in solving these problems will be discussed: R (several packages and functions), specially written Pascal programs and Excel SOLVER. The minimum cost-maximum flow solutions to the following problems will also be discussed: maximum flow, minimum cost-maximum flow, transportation problem, assignment problem, shortest path problem, caterer problem.

Keywords: Maximum flow; minimum cost-maximum flow; nodes; arcs; source; sink; capacities; costs; objective function; flow conservation and capacity constraints; algorithm; optimal solution; network optimization; transportation problem; assignment problem; shortest path problem; caterer problem.

1 Introduction

Maximum flow applies to any problem where the objective is to move as many as possible goods/objects/people between two locations via intermediate locations (optimal solution).

**Corresponding author: E-mail: moolman.henri@gmail.com;*

This will include problems such as maximizing oil/water flow via various pipelines, traffic flow via various routes, cell phone data traffic via network towers and scheduling of flights of airlines between cities via intermediate cities.

Examples

1 Minimizing traffic congestion see e.g. [1,2]. They solved the traffic congestion problem i.e. maximum flow of goods in a dynamic network with the help of a Lingo Model.
2 Schwarz [3] considered partially completed baseball league games and showed that the problem of eliminating teams (that cannot win a league) from a list of potential league winners can be reduced to a maximum network flow problem.

Minimum cost-maximum flow applies to flow problems where both capacities and costs are involved. An example of a situation where this will be applicable can be found in [4]. In the problem it is desired to maximize the number of cars that pass through a network of roads (defined by nodes and arcs) between two points (start and finish nodes). Each road (arc) in the network has a maximum capacity and a time taken to traverse it. The problem is to find the maximum number of cars that can travel between the two points at a minimum total time (optimal solution). In this example, time can be seen as the cost. It is obvious that a minimum cost-maximum flow problem where all the costs associated with the arcs are equal is the same as a maximum flow problem.

Besides finding a solution to a pure minimum cost-maximum flow problem (as described in the above example), the solution methods can also be used to solve special cases of the problem. Computer solutions of the following such cases will also be discussed.

1 The **transportation problem** (moving supplies at minimum costs from a certain number of sources to various destinations). The original transportation model of Dantzig [5] involved two sources of tin cans (located at Seattle and San Diego) and three destinations (located at New York, Chicago and Topeka). For given supplies, demands and transportation costs per case of cans an optimal solution that minimizes the total cost was calculated.
2 The **assignment problem** (e.g. assigning people to jobs / matching males and females for compatibility). This problem is a special case of the transportation model. A common algorithm of solving this problem is the Hungarian method developed by Kuhn [6].

Examples

An application of the assignment method in agriculture using R is given by [7].

Hultberg and Cardoso [8] formulated a model of the task of assigning classes to teachers such that the average number of subjects assigned to each teacher is minimized. This problem turns out to be a special case of the fixed charge transportation problem.

Chen et al [9] considered an application of bipartite matching (optimal matching of people and jobs) of assigning staff to tasks.

3 The **shortest path problem** (finding the shortest distance between two points via intermediate points). Demaine and Goldwasser [10] show how the Bellman-Ford implementation of the shortest path method can be applied to use discrepancies in currency exchange rates (arbitrage) to make money by converting currencies.
4 The **caterer problem** (cleaning napkins at minimum cost to satisfy demands at various future times). Szwarc and Posner [11] developed a one-pass transportation solution that solves the caterer problem with a fixed number of new napkins.

Ahuja et al. [12] devoted a chapter in a book to the topic of network optimization. According to them, solutions to the maximum flow and minimum cost-maximum flow problems are part of a wider class of problems called network optimization. The chapter describes 42 applications of network optimization drawn from different fields. Each of the problems that are considered can be presented as a network graph. Solution approaches are discussed.

Due to the vast amount of mathematical and computing details that are available on the various methods mentioned above, the following approach will be followed in the sections that follow.

1 The mathematical details will be kept to a minimum.
2 The important algorithms used to solve the above mentioned problems will be reviewed. Interested readers can follow up on the references given.
3 Manual solution of the maximum flow problem using the labelling algorithm.
4 Manual solution of the minimum cost-maximum flow problem using the Busacker-Gowen minimum-cost flow algorithm.
5 Computing the solutions of the maximum flow and minimum cost-maximum flow problems using (i) R packages (ii) Pascal programs and (iii) SOLVER in an excel spreadsheet.
6 The minimum cost - maximum flow solutions to the assignment and transportation problems by (i) formulating them as linear programming problems and using R and (ii) using a Pascal program to find the solution.
7 The minimum cost – maximum flow solution to the shortest route problem by (i) formulating it as a linear programming problem and using R to find the solution and (ii) solving it using SOLVER in an excel spreadsheet.
8 The minimum cost – maximum flow solution to the caterer problem by (i) formulating this problem as a transportation problem and solving it with the use of R and (ii) using SOLVER in an excel spreadsheet.

2 Problem Formulation and Algorithms

2.1 Problem statement

Let $G=(N,A)$ be a directed network defined by a set $N=\{1,2,\cdots,n\}$ of nodes and a set $A=\{(i,j),i\neq j:i,j\in N\}$ of arcs that connect certain pairs of nodes. Denote the numbers of nodes and arcs by n and m respectively. Each arc (i,j) has a capacity of u_{ij}, a lower bound of ℓ_{ij} and a flow of x_{ij}. The node $i=1$ will be called the source (origin of the flow) and the node $i=n$ the sink (destination of the flow). The **maximum flow** is defined as the maximum amount that can be sent from the source to the sink via the network and is formulated as

Maximize $\sum_{\substack{j\in A\\ j\neq 1}} x_{1j}$ (Total flow out of the source) or

$\sum_{\substack{j\in A\\ j\neq 1}} x_{jn}$ (Total flow into the sink) ,

subject to

$\sum_{\{j:(i,j)\in A\}} x_{ij} = \sum_{\{j::(i,j)\in A} x_{ji}$ (Flow conservation constraint)

$\ell_{ij} \leq x_{ij} \leq u_{ij}$. (Capacity constraint)

Denote the cost associated with a flow of 1 unit along the arc (i, j) by c_{ij} and the maximum flow amount determined by solving the above mentioned problem by M .

The **minimum cost-maximum flow** problem can be formulated as

Minimize $\sum\limits_{\substack{j \in A \\ j \neq 1}} c_{ij} x_{1j}$ (Total flow cost out of the source) or

$\sum\limits_{\substack{j \in A \\ j \neq 1}} c_{jn} x_{jn}$ (Total flow cost into the sink),

subject to

$$\sum_{\{j:(i,j)\in A\}} x_{ij} = \sum_{\{j::(i,j)\in A} x_{ji} \ .$$

$$\ell_{ij} \le x_{ij} \le u_{ij}$$

$\sum\limits_{i,j \in A} x_{ij} = M$. (Maximum flow constraint)

2.2 Algorithms for solving the maximum flow problem

2.2.1 Maximum flow

The maximum flow problem was first formulated by [13] as a simplified model of Soviet railway traffic flow. Ford and Fulkerson [14] created the first known algorithm to solve this problem. Over the years various improved solutions to this algorithm were proposed e.g. Dinitz [15] – Blocking Flow algorithm, Edmonds and Karp [16] – The shortest Augmenting Path algorithm, Goldberg and Tarjan [17] – Push-Relabel algorithm and Goldberg and Rao [18] – Binary Blocking Flow algorithm, KRT (King, Rao, Tarjan) algorithm [19] , the Orlin and KRT algorithm [20].

Some of the algorithms that were proposed only apply to specialized graphs e.g. Sherman [21] – undirected graphs and Malhotra, Pramodh-Kumar and Maheshwari [22] – acyclic networks. Ahmed et al. [23] proposed an algorithm of the maximum flow problem that requires less iterations and augmentation than the Ford-Fulkerson algorithm. Orlin [24] gives an $O(mn)$, $m = O(n^{16/15})$ time algorithm which is currently the fastest strongly polynomial time algorithm (e.g. of $O(n^2)$) for the maximum flow problem.

The following Table 1, obtained from Wikipedia [25], shows the complexities of the various algorithms.

2.2.2 Minimum cost-maximum flow

Many algorithms for solving the minimum cost-maximum flow problem were proposed over the past 6 decades. These include those by Fulkerson [26] – Out-of-Kilter algorithm, Busaker & Gowan [27] – Cheapest Path Augmentation, Klein [28] – Cycle cancelling, Engquist [29] – Successive shortest path, Carpaneto & Toth [30] – Primal-Dual, Goldberg & Tarjan [17] – Push/Relabel and [31,32,33] – Network Simplex. A summary of the algorithms and their complexities is shown in the Table 3.

Table 1. Complexities of some maximum flow algorithms

Algorithm	Complexity
Ford-Fulkerson	$O(m\,f)$, where f is the maximum amount of flow from source to sink.
Edmonds-Karp	$O(nm^2)$
Dinic blocking flow	$O(n^2m)$
Malhotra, Pramodh-Kumar and Maheshwari (MPM)	$O(n^3)$
Dinic's	$O(nm\log n)$
General Push-Relabel	$O(n^2m)$
King, Rao, Tarjan (KRT)	$O(mn\log_a n)$, where $a=\frac{m}{n\log n}$.
Binary Blocking	$O(m\times\min(n^{2/3},m^{1/2})\times\log\frac{n^2}{m}\times\log U)$, where U is the maximum capacity of the network.
Orlin + KRT	$O(nm)$

Table 2. Limitations of some maximum flow algorithms

Algorithm	Limitation
Ford-Fulkerson	Only guaranteed to terminate if all weights are rational.
MPM	Only works on acyclic networks

Table 3. Complexities of some minimum cost-maximum flow algorithms

Algorithm	Complexity
Out-of-Kilter	$O(m\,U)$
Cheapest Path Augmentation	$O(n^3v)$, where v is the number of augmentations required.
Cycle cancelling	$O(n^2mUW)$, where W is the maximal cost of an arc in the graph.
Successive shortest path	$O(nm)\times T(nm)$, where $T(nm)$ is the time required to find the shortest path in a graph with n nodes and m arcs.
Primal-Dual	$O(n\,U)$
Network Simplex (Orlin)	$O(n^2m\log(\frac{n}{C}))$, where C is the maximum of cost values.
Network Simplex (Tarjan)	$O(nm\times\log n\times\log(\frac{n}{C}))$

A polynomial time algorithm is an algorithm whose execution time is polynomial on the size of the input, or can be bounded by such a polynomial e.g. $O(n^2)$ is polynomial time.

Orlin [34] proposed a faster polynomial time minimum cost flow algorithm. He also presented a summary table of polynomial and strongly polynomial algorithms showing authors, dates and running times. Parpalea [35] discussed some residual networks and successive short paths algorithms that can be used to solve the minimum cost flow problem. As applications, Dijkstra's shortest path algorithm and the minimum cost flow application are discussed. Sokkalingam, Ahuja, and Orlin [36] discussed polynomial time cycle-cancelling algorithms for minimum-cost flows.

Goldberg and Tarjan [37] survey basic techniques behind efficient maximum flow algorithms, starting with the history and basic ideas behind the fundamental maximum flow algorithms and then explores the algorithms in more detail. The study is restricted to basic maximum flow algorithms and does not include special cases and generalizations.

3 Solution of the Maximum Flow Problem

3.1 Example

The application of the maximum flow and minimum cost-maximum flow methods will be demonstrated by calculating flows for a network which has 6 nodes and 10 arcs as shown in the Table below. The computer solution methods described can easily be adapted to solve these problems for any numbers of nodes and arcs.

Table 4. Arcs, upper capacities and costs of network

arc	(1,2)	(1,3)	(2,4)	(2,5)	(3,2)	(3,4)	(3,5)	(4,6)	(5,4)	(5,6)
u_{ij}	7	10	7	6	2	3	4	7	1	10
c_{ij}	4	0	2	4	1	11	1	2	2	5

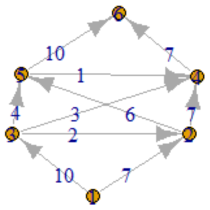

Fig. 1. Graph of arcs and upper capacities in the network with 6 nodes

3.2 Finding the maximum flow by using the labelling algorithm

According to the Ford-Fulkerson algorithm each node is labelled (*pre, flow*), where *pre* is the node preceding the node and *flow* the amount that can be sent from the source to that particular node. This labelling of nodes is continued until the sink can be labelled and a path from source to sink is established. Using this labelling approach the following paths from source (node 1) to sink (node 6) can be found.

Table 5. Paths between source and sink using the labelling approach

Path	Flow
1-2-4-6	7
1-3-2-5-6	2
1-3-5-6	4
Total flow	13

The amount of flow along a given path from source to sink is the flow of the arc (on the path) with the minimum flow (flow of bottleneck arc). For each of the above flows from source to sink the upper capacities of the arcs on the path are adjusted by subtracting the amount of flow from their existing upper capacities. After updating the upper capacities (residual capacities) the capacities on the flow chart change to that are shown below.

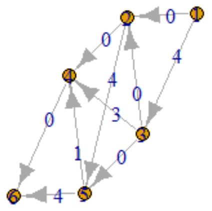

Fig. 2. Graph of arcs and adjusted capacities

The only path starting at the source with a positive flow in this network is 1-3-4 with a flow of 3. However, the sink node cannot be labelled along this path, since the arc (4,6) has an adjusted capacity of 0. This problem can be overcome by sending 3 units back to node 2 along the reverse path 6-4-2. This allows the following flows.

Table 6. Adjusted flows obtained after introducing a reverse path

Path	Flow
6-4-2	-3
1-3-4-6	3
2-5-6	3

After introducing these flows, the total flow increases to 13+3+3-3=16. After adjusting the upper capacities by subtracting the additional flows it is found that the only positive flow starting at the source is at arc (1,3) with an adjusted capacity of 1. Since all the arcs starting from node 3 have adjusted capacities of 0, no further increase in flow is possible and the optimal flow is reached. The solution to the maximum flow is therefore

$$x_{12} = 7, x_{13} = 9, x_{24} = 4, x_{25} = 5, x_{32} = 2, x_{34} = 3, x_{35} = 4, x_{46} = 7, x_{56} = 9 .$$

3.3 Using the lpSolve library in R

The maximum flow problem can be formulated and solved as a linear programming problem. The code below shows the solution using the lpSolve library in R.

```
library(lpSolve)
# variables x12 x13 x24 x25 x32 x34 x35 x46 x54 x56
# max flow calculation
# Objective function
f.obj=c(0,0,0,0,0,0,0,1,0,1)
# Flow conservation constraints (first 4)
# Capacity constraints (next 10)
f.con = matrix(c(1,0,-1,-1,1,0,0,0,0,0,
         0,1,0,0,-1,-1,-1,0,0,0,
         0,0,1,0,0,1,0,-1,1,0,
         0,0,0,1,0,0,1,0,-1,-1,
         1,0,0,0,0,0,0,0,0,0,
         0,1,0,0,0,0,0,0,0,0,
         0,0,1,0,0,0,0,0,0,0,
         0,0,0,1,0,0,0,0,0,0,
         0,0,0,0,1,0,0,0,0,0,
         0,0,0,0,0,1,0,0,0,0,
         0,0,0,0,0,0,1,0,0,0,
         0,0,0,0,0,0,0,1,0,0,
         0,0,0,0,0,0,0,0,1,0,
         0,0,0,0,0,0,0,0,0,1),nrow=14,byrow=TRUE)
# Signs for constraints
f.dir = c(rep("=",4),rep("<=",10))
# Right hand sides of constraints
f.rhs = c(rep(0,4),7,10,7,6,2,3,4,7,1,10)
# Execute Linear Programming function
lp ("max", f.obj, f.con, f.dir, f.rhs)
Success: the objective function is 16
```

```
# Write the solution
lp ("max", f.obj, f.con, f.dir, f.rhs)$solution
[1] 7 9 4 5 2 3 4 7 0 9
```

This solution is the same as that found in 3.2 above.

3.4 Using the igraph library in R

The igraph library in R has a built-in max_flow function that can find the solution to the maximum flow problem.

```
library(igraph)
# Enter matrix with columns "node from", "node to", "capacity"
E <- rbind(c(1,2,7), c(1,3,10), c(2,4,7), c(2,5,6), c(3,2,2), c(3,4,3), c
(3,5,4), c(4,6,7), c(5,4,1), c(5,6,10))
colnames(E) <- c("from", "to", "capacity")
# Define graph
g1 <- graph_from_data_frame(as.data.frame(E))
# Find max flow value and flows
max_flow(g1, source=V(g1)["1"], target=V(g1)["6"])
$value
[1] 16

$flow
[1] 7  9  3  6  2  3  4  6  0 10
```

The igraph solution shown below is slightly different to that obtained by using lpSolve, but it has the same objective function value. This shows that the solution to a maximum flow problem is not necessarily unique.

$$x_{12} = 7, x_{13} = 9, x_{24} = 3, x_{25} = 6, x_{32} = 2, x_{34} = 3, x_{35} = 4, x_{46} = 6, x_{56} = 10.$$

3.5 Using the Pascal procedure MAXFLOW

The Pascal procedure MAXFLOW, that was written by Syslo, Deo and Kowalik [38], can be used to calculate the maximum flow and the flow along the various arcs. The algorithm used is due to Dinitz [15] with modifications by Malhotra, Pramodh-Kumar and Maheshwari [22] and is therefore called the DMKM algorithm.

The DMKM algorithm is based on constructing a series of layered networks i.e. partitioning the network into layers. In a layered network the first layer is the source (label =1), the second layer all immediate successors (nodes) of the source (labels=2), the third layer all immediate successors of the second layer (labels=3) etc. The final layer is the sink. All nodes that are not labelled and arcs incident on them are deleted. For each layered network an attempt is made to find a saturating flow (a flow path between source and sink where for at least one arc on the path the flow = upper capacity). After sending an amount of flow between source and sink, the residual capacities are adjusted i.e. upper capacity of arc $(i, j) = u_{ij} - x_{ij}$ and upper capacity of arc $(j,i) = x_{ij}$. After deleting all saturated arcs and using the updated residual capacities, a new layered network is constructed and the procedure of finding a saturating flow through the network repeated. This algorithm of combining a layered network with a saturating flow continues until a saturating flow cannot be found. Once this happens the maximum flow has been found. The maximum flow is the sum of all the saturating flows found (cumulative flow).

The MAXFLOW procedure can be found in [38] p.289-295. The Pascal programming code needed for the data input and writing the results are shown in A1 in the appendix. The input data and output results are shown below.

Input and output

The (i, j)th entry in the data input matrix is the capacity of the arc (i, j). Nodes which are not connected are allocated a capacity of 0. The input matrix for the network data in Table 4.

0	7	10	0	0	0
0	0	0	7	6	0
0	2	0	3	4	0
0	0	0	0	0	7
0	0	0	1	0	10
0	0	0	0	0	0

The output is shown below.

0	7	9	0	0	0
0	0	0	3	6	0
0	2	0	3	4	0
0	0	0	0	0	6
0	0	0	0	0	10
0	0	0	0	0	0

The maximum flow for arc (i, j) is the (i, j)th entry in the above table. The solution is

$$x_{12} = 7, x_{13} = 9, x_{24} = 3, x_{25} = 6, x_{32} = 2, x_{34} = 3, x_{35} = 4, x_{46} = 6, x_{56} = 10,$$

which is the same as the solution obtained from igraph.

3.6 Using SOLVER in an excel spreadsheet

Below is the excel spreadsheet showing the SOLVER solution (see column D) to the maximum flow for the data in Table 1.

Table 7. SOLVER solution to maximum flow for the data in Table 1

	B	C	D	E	F	G	H
	from	**to**	**flow**	**capacity**	**nodes**	**net flow**	**flow cons.**
3	1	2	7	7	1	16	
4	1	3	9	10	2	0	0
5	2	4	4	7	3	0	0
6	2	5	5	6	4	0	0
7	3	2	2	2	5	0	0
8	3	4	3	3	6	0	0
9	3	5	4	4			
10	4	6	7	7			
11	5	4	0	1			
12	5	6	9	10			
13							
14			max flow	16			

Explanation of obtaining Table 3 entries

Columns B and C show the nodes that define the different arcs.

Column D shows the flow along the various arcs. Initially all the flows are 0. The final flows are calculated by the solver program.

Column E shows the upper capacities of the different arcs.

Column F is a list of the nodes.

Column G defines the net flows for each of the nodes. These are obtained from the column D flows as follows.

Net flow – Node 1(G3): =D3+D4
Node 2(G4): =D3+D7-D5-D6
Node 3(G5): =D4-D7-D8-D9
Node 4(G6): =D5+D8+D11-D10
Node 5(G7): =D6+D9-D11-D12
Node 6(G8): =D10+D12-D3-D4

Column H shows the flow conservation constraints (all 0's in this case).

The maximum flow is calculated in cell G3 and indicated in cell E14 (=G3).

When using SOLVER the objective cell is G3, the objective Max, the changing variable cells D3-D12, the constraints D3-D12<= E3-E12 and G4-G8 = H4-H8 and the Solving Method the Simplex LP. The "Make Unconstrained Variables Non-Negative" box is ticked.

4 Solution of the Minimum Cost– Maximum Flow Problem

4.1 Using the lpSolve library in R

The lpSolve solution to the minimum cost – maximum flow problem is very similar to that for the maximum flow problem.

The following changes are needed.

1 The objective function coefficients are the costs (see f.obj values).
2 An additional constraint total flow = maximum flow is added (5th row in f.con matrix).
3 The objective function needs to be minimized.

```
library(lpSolve)
# variables x12 x13 x24 x25 x32 x34 x35 x46 x54 x56
# min cost calculation
f.obj=c(4,0,2,4,1,11,1,2,2,5)
f.con = matrix(c(1,0,-1,-1,1,0,0,0,0,0,
          0,1,0,0,-1,-1,-1,0,0,0,
          0,0,1,0,0,1,0,-1,1,0,
          0,0,0,1,0,0,1,0,-1,-1,
          0,0,0,0,0,0,0,1,0,1,
          1,0,0,0,0,0,0,0,0,0,
          0,1,0,0,0,0,0,0,0,0,
            0,0,1,0,0,0,0,0,0,0,
            0,0,0,1,0,0,0,0,0,0,
            0,0,0,0,1,0,0,0,0,0,
            0,0,0,0,0,1,0,0,0,0,
            0,0,0,0,0,0,1,0,0,0,
            0,0,0,0,0,0,0,1,0,0,
            0,0,0,0,0,0,0,0,1,0,
            0,0,0,0,0,0,0,0,0,1),nrow=15,byrow=TRUE)
 f.dir = c(rep("=",5),rep("<=",10))
 f.rhs = c(rep(0,4),16,7,10,7,6,2,3,4,7,1,10)
 lp ("min", f.obj, f.con, f.dir, f.rhs)
Success: the objective function is 154
 lp ("min", f.obj, f.con, f.dir, f.rhs)$solution
[1] 7 9 4 5 2 3 4 7 0 9
```

The solution here is the same as that for the maximum flow problem. This will not be the case in general.

4.2 Using the Busacker-Gowen minimum-cost flow algorithm

The following steps are followed in the application of this algorithm.

1 Start with a zero flow in the network *i.e.* $x_{ij} = 0$ for all arcs (i, j) that form part of the network.

Calculate residual capacities for the arcs.

1.1 Arc (i, j) has residual capacity $r_{ij} = u_{ij} - x_{ij}$ and cost c_{ij}.

1.2 Arc (j, i) has residual capacity $r_{ji} = x_{ij}$ and cost $c_{ji} = -c_{ij}$.

2 Using the cost of each arc as length, find the shortest path in the network from the source to the sink node.

3 Determine the amount of flow that can be sent along the path identified in step 2. This amount of flow will be the minimum of the capacities of all the arcs that are on this path.

4 Update the capacities and costs of the residual network and return to step 1. The algorithm stops when a path from source to sink with positive flows cannot be found.

Table 8. Application of the Busacker-Gowen min-cost flow algorithm

Shortest path	x_{ij}	comment
1-3-2-4-6	min(10,2,7,7)=2	
1-3-5-4-6	min(8,4,1,5)=1	
1-3-5-6	min(7,3,10)=3	
1-2-4-6	min(7,5,4)=4	
1-2-4-5-6	min(3,1,1,7)=1	Flow of arc (5,4) changes from 1 to 0.
1-2-5-6	min(2,6,6)=2	
1-3-4-2-5-6	min(4,3,7,4,4)=3	Flow of arc (2,4) changes from 7 to 4.

From the above table the following solution can be written down.

$$x_{12} = 7, x_{13} = 9, x_{24} = 4, x_{25} = 5, x_{32} = 2, x_{34} = 3, x_{35} = 4, x_{46} = 7, x_{56} = 9.$$

4.3 Using the Pascal procedure BUSACKER

This procedure is the computer implementation of the Busacker-Gowen min-cost flow algorithm. This procedure makes use of the PDM procedure which is the Pape, d'Esopo implementation of the Moore-Bellman shortest path algorithm (Moore [39] and Bellman [40]). This algorithm was modified by d'Esopo and refined by Pape [41].

The Pascal procedure BUSACKER can be found in [38] p.310-312. The Pascal programming code needed for the data input and writing the results are shown in A2 in the appendix. The input data and output results are shown below.

Input and output

This procedure needs as input both the capacity and cost matrices. In the matrices below the (i, j)th entry refers to the capacity/unit cost of arc (i, j). The unit costs of the arcs which are not connected is set equal to a large number e.g. 9999 which is much larger than any of the unit costs associated with the arcs which are connected.

Capacity matrix:

0	7	10	0	0	0
0	0	0	7	6	0
0	2	0	3	4	0
0	0	0	0	0	7
0	0	0	1	0	10
0	0	0	0	0	0

Cost matrix:

9999	4	0	9999	9999	9999
9999	9999	9999	2	4	9999
9999	1	9999	11	1	9999
9999	9999	9999	9999	9999	2
9999	9999	9999	2	9999	5
9999	9999	9999	9999	9999	9999

The output is shown below.

0	7	9	0	0	0
0	0	0	4	5	0
0	2	0	3	4	0
0	0	0	0	0	7
0	0	0	0	0	9
0	0	0	0	0	0

The minimum cost/maximum flow for arc (i, j) is the (i, j)th entry in the above table. The solution is

$$x_{12} = 7, x_{13} = 9, x_{24} = 4, x_{25} = 5, x_{32} = 2, x_{34} = 3, x_{35} = 4, x_{46} = 7, x_{56} = 9.$$

4.4 Using SOLVER in an excel spreadsheet

Below is the excel spreadsheet showing the SOLVER solution (see column E) to the minimum cost – maximum flow for the data in Table 4. See [42].

Table 9. SOLVER solution to minimum cost – maximum flow for the data in Table 1

	B	C	D	E	F	G	H	I
	from	**to**	**flow**	**capacity**	**nodes**	**Net flow**	**Flow cons.**	**cost**
3	1	2	7	7	1	16	16	4
4	1	3	9	10	2	0	0	0
5	2	4	4	7	3	0	0	2
6	2	5	5	6	4	0	0	4
7	3	2	2	2	5	0	0	1
8	3	4	3	3	6	0	0	11
9	3	5	4	4				1
10	4	6	7	7				2
11	5	4	0	1				2
12	5	6	9	10				5
13								
14			min cost	154				

The above spreadsheet is a modified version of the one used for maximum flow. The following modifications are made.

1 Cell H3 has 16 (maximum flow) instead of a blank.
2 Column I with costs per unit is added.
3 The value of the minimum cost cell E14 is calculated as =SUMPRODUCT(D3:D12,I3:I12).
4 When using SOLVER the objective cell is G3 and the objective Min.

5 The Minimum Cost– Maximum Flow Solution to the Transportation and Assignment Problems

5.1 Transportation problem using the lp.transport function in lpSolve in R

In the transportation problem shown below there are 4 supply sources and 6 demand destinations. The (i,j)th entry in the cost matrix below is the cost of transporting one unit from source $i = 1,2,3,4$ to destination $j = 1,2,\cdots,6$.

$$\text{Cost matrix } = \begin{pmatrix} 30 & 11 & 5 & 35 & 8 & 29 \\ 2 & 5 & 2 & 5 & 1 & 9 \\ 35 & 20 & 6 & 40 & 8 & 33 \\ 19 & 2 & 4 & 30 & 10 & 25 \end{pmatrix}, \text{ supply vector} = \begin{pmatrix} 30 \\ 10 \\ 45 \\ 30 \end{pmatrix}, \text{ demand vector } = \begin{pmatrix} 25 \\ 20 \\ 6 \\ 7 \\ 22 \\ 35 \end{pmatrix}.$$

In the R code shown below the transportation problem is formulated as a linear programming problem.

```
library(lpSolve)
 costs = matrix(c(30,11,5,35,8,29,
               2,5,2,5,1,9,
               35,20,6,40,8,33,
               19,2,4,30,10,25),nrow=4, byrow=TRUE)
 # Supply constraints
 row.signs = rep ("<", 4)
 row.rhs = c(30,10,45,30)
 # Demand constraints
 col.signs = rep (">", 6)
 col.rhs <- c(25,20,6,7,22,35)
 lp.transport (costs, "min", row.signs, row.rhs, col.signs, col.rhs)
 Success: the objective function is 1902
 lp.transport (costs, "min", row.signs, row.rhs, col.signs, col.rhs)$solut
 ion
      [,1] [,2]  [,3]  [,4] [,5]  [,6]
[1,]    0   12    0    0    0   18
[2,]    3    0    0    7    0    0
[3,]    0    0    6    0   22   17
[4,]   22    8    0    0    0    0
```

In the above solution matrix the entry in row i, column j is the optimum amount to be transported from supply source i to demand destination j. The solution is

$$x_{12} = 12, x_{16} = 18, x_{21} = 3, x_{24} = 7, x_{33} = 6, x_{35} = 22, x_{36} = 17, x_{41} = 22, x_{42} = 8.$$

5.2 Transportation problem using the Pascal procedure TRANSPORT

The transportation problem can be formulated as

Minimize $\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij}$ subject to

$$\sum_{j=1}^{n} x_{ij} \le a_i, i = 1,2,\cdots,m \quad \text{(supply constraints)}$$

$$\sum_{i=1}^{m} x_{ij} \ge b_j, j = 1,2,\cdots,n \quad \text{(demand constraints)}$$

$$x_{ij} \ge 0, i = 1,2,\cdots,m; j = 1,2,\cdots,n.$$

In the above, c_{ij} is the cost of transporting 1 unit from supply point i to demand point j, x_{ij} is the amount sent from supply point i to demand point j, a_i the maximum amount available at supply point i and b_j the minimum amount demanded at demand point j.

The network representation of this problem is done in the following way. Arcs connect each of the m supply nodes to each of the n demand nodes (mn arcs). Each of these arcs have an upper capacity of ∞. The source node is connected to each of the supply nodes (each of these arcs have an upper capacity of the supply available) and the sink node to each of the demand nodes (each of these arcs have a lower capacity of the minimum amount demanded).

By using the approach, explained by [38], the above mentioned problem can be solved by solving the following maximum flow problem.

Maximize $\sum_{i=1}^{m}\sum_{j=1}^{n} x_{ij}$ subject to

$$\sum_{j=1}^{n} x_{ij} \le a_i, i = 1,2,\cdots,m \quad \text{(supply constraints)}$$

$$\sum_{i=1}^{m} x_{ij} \ge b_j, j = 1,2,\cdots,n \quad \text{(demand constraints)}$$

$$x_{ij} = 0 \text{ for } -u_i + v_j < c_{ij}$$

$$\ge 0 \text{ otherwise.}$$

In the above u_i and v_j are the variables in the dual of the transportation problem which is

maximize $-\sum_{i=1}^{m} u_i a_i + \sum_{j=1}^{n} v_j b_j$ subject to

$$-u_i + v_j \le c_{ij},\ u_i, v_j \ge 0.$$

In the maximum flow solution of the transportation problem the arcs for which $-u_i + v_j < c_{ij}$ are removed from the network. According to the complementary slackness theorem $x_{ij} = 0$ for these arcs and therefore they can be removed from the network.

The TRANSPORT procedure can be found in [38] p.68-71. The Pascal programming code needed for the data input and writing the results are shown in A3 in the appendix. The output results are shown below.

Input and output

The inputs to the pascal procedure are the cost matrix, supply and demand vectors as shown at the beginning of the previous section.

The optimal solution output is

0	12	0	0	0	18
3	0	0	7	0	0
0	0	6	0	22	17
22	8	0	0	0	0

In the above solution matrix the entry in row i, column j is the optimum amount to be transported from supply point i to demand point j. The solution is

$x_{12} = 12, x_{16} = 18, x_{21} = 3, x_{24} = 7, x_{33} = 6, x_{35} = 22, x_{36} = 17, x_{41} = 22, x_{42} = 8$, which is the same as that found by using lpSolve in R.

5.3 Assignment problem solution using the Pascal procedure TRANSPORT

In the assignment problem, each of n persons are to be assigned to one of n jobs (one person per job) where c_{ij} is the cost associated with assigning person i to job j. The minimum cost assignment problem can be formulated as

Minimize $\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$ subject to

$$\sum_{j=1}^{n} x_{ij} = 1, i = 1,2,\cdots,n \quad \text{(one person per job)}$$

$$\sum_{i=1}^{n} x_{ij} = 1, j = 1,2,\cdots,n \quad \text{(one job per person)}$$

$$x_{ij} = 0 \text{ or } 1, i = 1,2,\cdots,n; j = 1,2,\cdots,n .$$

The assignment problem can be solved by using the TRANSPORT procedure with input the cost matrix and vectors of n 1's for both the supply and demand.

Example 1 – Minimization assignment.

Assignment of 4 persons to 4 jobs.

$$\text{Cost matrix} = \begin{pmatrix} 4 & 5 & 2 & 5 \\ 3 & 1 & 1 & 4 \\ 12 & 3 & 6 & 3 \\ 12 & 6 & 5 & 9 \end{pmatrix}.$$

The solution is given below.

1	0	0	0
0	1	0	0
0	0	0	1
0	0	1	0

According to this solution person 1 is assigned to job 1, person 2 to job 2, person 3 to job 4 and person 4 to job 3.

Example 2 – Maximization assignment.

Four workers ($(w_1 \quad w_2 \quad w_3 \quad w_4)$ are to be assigned to 4 machines $(m_1 \quad m_2 \quad m_3 \quad m_4)$ such that the productivity is maximized. The productivity matrix is

$$\begin{array}{ccccc} & m_1 & m_2 & m_3 & m_4 \\ w_1 & 9 & 8 & 10 & 14 \\ w_2 & 12 & 11 & 13 & 13 \\ w_3 & 7 & 5 & 22 & 18 \\ w_4 & 8 & 10 & 14 & 9 \end{array}.$$

Since the formulation is a minimization problem, the entries in the productivity matrix will be negative (maximizing is the same as minimizing the negative productivity). With this productivity matrix and supply and demand vectors of 1's the solution is

0	0	0	1
1	0	0	0
0	0	1	0
0	1	0	0

According to this solution worker 1 is assigned to machine 4, worker 2 to machine 1, worker 3 to machine 3 and worker 4 to machine 2.

5.4 Assignment problem solution using the lp.assign function in lpSolve in R

Example 1 – Minimization problem

```
library(lpSolve)
assign.costs = matrix (c(4,5,2,5,3,1,1,4,12,3,6,3,12,6,5,9), 4, 4)
lp.assign (assign.costs)
Success: the objective function is 13
lp.assign (assign.costs)$solution
    [,1] [,2] [,3] [,4]
[1,]   1    0    0    0
[2,]   0    1    0    0
[3,]   0    0    0    1
[4,]   0    0    1    0
```

Example 2 – Maximization problem

```
library(lpSolve)
assign.prod = matrix (c(-9,-12,-7,-8,-8,-11,-5,-10,-10,-13,-22,-14,-14,-1
3,-18,-9), 4, 4)
lp.assign (assign.prod)
Success: the objective function is -58
lp.assign (assign.prod)$solution

     [,1] [,2] [,3] [,4]
[1,]   0    0    0    1
[2,]   1    0    0    0
[3,]   0    0    1    0
[4,]   0    1    0    0
```

The assignments are the same as those obtained by using the Pascal procedure TRANSPORT.

6 The Minimum Cost– Maximum Flow Solution to the Shortest Route Problem

6.1 Shortest route problem

The solution approaches will be described by solving the following problem.

A cargo is to be transported from Los Angeles to St. Louis. As can be seen from the diagram below, there are various possible routes between these two cities. The travelling times between cities (in hours) are indicated on the diagram. The shortest route in terms of travelling time between these two cities is to be determined.

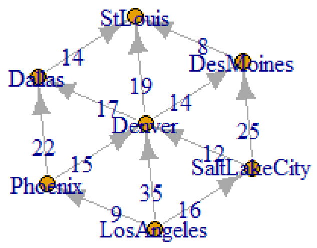

Fig. 3. Travelling times between cities situated on routes from Los Angeles to St. Louis

In order to write a linear programming formulation of this shortest route problem the cities on the routes will be labelled as follows.

Los Angeles-1, Salt Lake City-2, Phoenix-3, Denver-4, Des Moines-5, Dallas-6, St.Louis-7.

Let $x_{ij} = 1$ or 0 depending on whether or not the route between cities i and j is included in the route between Los Angeles to St. Louis. This problem can be formulated as the following linear programming problem.

Minimize $z = 16x_{12} + 9x_{13} + 35x_{14} + 12x_{24} + 25x_{25} + 15x_{34} + 22x_{36} + 14x_{45} + 17x_{46} + 19x_{47} + 8x_{57} + 14x_{67}$
Subject to

$x_{12} + x_{13} + x_{14} = 1$, $x_{12} - x_{24} - x_{25} = 0$, $x_{13} - x_{34} - x_{36} = 0$, $x_{14} + x_{24} + x_{34} - x_{45} - x_{46} - x_{47} = 0$, $x_{25} + x_{45} - x_{57} = 0$, $x_{36} + x_{46} - x_{67} = 0$, $x_{47} + x_{57} + x_{67} = 1$, $x_{ij} = 1$ or 0.

6.2 Shortest route solution using linear programming formulation in lpSolve

```
library(lpSolve)
# Cities Los Angeles-1,Salt Lake City-2,Phoenix-3,Denver-4,Des Moines-5,
# Dallas-6,St.Louis-7
# variables x12 x13 x14 x24 x25 x34 x36 x45 x46 x47 x57 x67
 f.obj = c(16,9,35,12,25,15,22,14,17,19,8,14)
 f.con = matrix(c(1,1,1,0,0,0,0,0,0,0,0,0,
                1,0,0,-1,-1,0,0,0,0,0,0,0,
                0,1,0,0,0,-1,-1,0,0,0,0,0,
                0,0,1,1,0,1,0,-1,-1,-1,0,0,
                0,0,0,0,1,0,0,1,0,0,-1,0,
                0,0,0,0,0,0,1,0,1,0,0,-1,
                0,0,0,0,0,0,0,0,0,1,1,1),nrow=7,byrow=TRUE)
 f.dir = c(rep("=",7))
 f.rhs = c(1,rep(0,5),1)
 lp ("min", f.obj, f.con, f.dir, f.rhs)
Success: the objective function is 43
 lp ("min", f.obj, f.con, f.dir, f.rhs)$solution
 [1] 0 1 0 0 0 1 0 0 0 1 0 0
```

The solution is $x_{13} = 1, x_{34} = 1, x_{47} = 1$ with all other variables 0. This means that the shortest route is Los Angeles-Phoenix-Denver-St. Louis with a total time of 9 + 15 + 19 = 43 hours.

6.3 Shortest route solution using SOLVER in excel

Below is the excel spreadsheet showing the SOLVER solution (see column D) to the shortest route problem for the distances shown in Fig. 3.

Table 10. SOLVER solution to the shortest route solution for the network in Fig. 3

	B	**C**	**D**	**E**	**F**	**G**	**H**
2	**from**	**to**	**flow**	**Cities**	**Net flow**	**Flow cons.**	**Time**
3	1	2	0	1	1	1	16
4	1	3	1	2	0	0	9
5	1	4	0	3	0	0	35
6	2	4	0	4	0	0	12
7	2	5	0	5	0	0	25
8	3	4	1	6	0	0	15
9	3	6	0	7	1	1	22
10	4	5	0				14
11	4	6	0				17
12	4	7	1				19
13	5	7	0				8
14	6	7	0				14
15							
16			min time	43			

Cities: Los Angeles-1, Salt Lake City-2, Phoenix-3, Denver-4, Des Moines-5, Dallas-6, St.Louis-7.

Explanation of obtaining Table 7 entries.

Columns B and C show the labels of the cities "from" and "to".

Column D shows the values of the decision variables (0's or 1's). Initially all the values are 0. The final values shown above are calculated by the solver program.

Column E shows the labels of the 7 cities.

Column F defines the net flows for each of the cities. These are obtained from the column D flows as follows.

Net flow – City 1(F3): =D3+D4+D5
City 2(F4): =D3-D6-D7
City 3(F5): =D4-D8-D9
City 4(F6): =D5+D6+D8-D10-D11-D12
City 5(F7): =D7+D10-D13
City 6(F8): =D9+D11-D14
City 7(F9): =D12+D13+D14

Column G shows the flow conservation constraints (1's for cities 1 and 7, 0's for cities 2 to 6).

Column H shows the travelling times between the cities.

The minimum time is calculated in cell E16 by using the formula SUMPRODUCT(D3:D14,H4:H14).

When using SOLVER the objective cell is E16, the objective Min, the changing variable cells D3-D14, the constraints F3-F9 = G3-G9 and the Solving Method is Simplex LP. The "Make Unconstrained Variables Non-Negative" box is ticked.

The solution $x_{13} = 1, x_{34} = 1, x_{47} = 1$ with all other variables 0 is the same as that obtained from using lpSolve.

7 The Minimum Cost–Maximum Flow Solution to the Caterer Problem

7.1 Caterer problem description

The solution approaches will be described by solving the following problem.

A caterer has to provide food for meals in the next 8 days. The number of cloth napkins that are to be used in the meals for these 8 days are 90, 140, 84, 95, 130, 170, 155 and 200 respectively. New napkins cost 5. Dirty napkins can be washed at a laundry and used on subsequent days. The following types of laundry are available: A fast laundry charges 1.20 per napkin and will deliver for use on the second day, while a slow laundry charges 0.80 per napkin and can deliver for use on the third day. The problem is to find the least cost “purchase and use” plan for the caterer.

7.2 Transportation problem solution of the caterer problem

The caterer problem can be formulated as a transportation problem. The table below shows a summary of the supply, demand and costs in such a formulation.

Table 11. Transportation formulation of caterer problem

										Supply	**Source**
Costs	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**0**	**1064**	**1**
	M	M	1.2	0.8	0.8	0.8	0.8	0.8	0	90	2
	M	M	M	1.2	0.8	0.8	0.8	0.8	0	140	3
	M	M	M	M	1.2	0.8	0.8	0.8	0	84	4
	M	M	M	M	M	1.2	0.8	0.8	0	95	5
	M	M	M	M	M	M	1.2	0.8	0	130	6
	M	M	M	M	M	M	M	1.2	0	170	7
Demand	90	140	84	95	130	170	155	200	709		
Day	1	2	3	4	5	6	7	8	Surplus		

Explanation of table entries:

1 Big M entries. M is taken as a cost per napkin which is much larger than the known costs (in this case 5, 1.2 and 0.8). These cost entries appear in cells where transport from source to demand cannot be done.

2 Supply sources. Source 1 refers to the option of supplying new napkins every day i.e. the total supply of new napkins is 90+140+84+95+130+170+155+200=1064. Sources 2 to 7 refer to using washed napkins on days 3 to 8. For these sources the costs are entered by taking into account the days on which these washed napkins would become available.

3 Surplus. This is the difference between the total supply and total demand i.e. 90+140+84+95+130+170=709. A dummy demand of 709 with 0 costs is created to balance the transportation supplies and demands.

```
library(lpSolve)
costs = matrix(c(5,5,5,5,5,5,5,5,0,
            100,100,1.2,0.8,0.8,0.8,0.8,0.8,0,
            100,100,100,1.2,0.8,0.8,0.8,0.8,0,
            100,100,100,100,1.2,0.8,0.8,0.8,0,
            100,100,100,100,100,1.2,0.8,0.8,0,
            100,100,100,100,100,100,1.2,0.8,0,
            100,100,100,100,100,100,100,1.2,0),nrow=7, byrow=TRUE)
row.signs = rep ("<=", 7)
row.rhs = c(1064,90,140,84,95,130,170)
col.signs = rep (">=", 9)
col.rhs <- c(90,140,84,95,130,170,155,200,709)
lp.transport (costs, "min", row.signs, row.rhs, col.signs, col.rhs)
Success: the objective function is 2466.2
lp.transport (costs, "min", row.signs, row.rhs, col.signs, col.rhs)$solution

     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
[1,]   90  140   84   41    0    0    0    0  709
[2,]    0    0    0   54   36    0    0    0    0
[3,]    0    0    0    0   94   46    0    0    0
[4,]    0    0    0    0    0   84    0    0    0
[5,]    0    0    0    0    0   40   55    0    0
[6,]    0    0    0    0    0    0  100   30    0
[7,]    0    0    0    0    0    0    0  170    0
```

From the above output the following solution can be written down.

Table 12. Solution of caterer problem

Supply/day	1	2	3	4	5	6	7	8	Total	Cost per napkin
new	90	140	84	41					355	5
fast						40	100	170	310	1.2
slow				54	130	130	55	30	399	0.8

Total cost = $355 \times 5 + 310 \times 1.2 + 399 \times 0.8 = 2466.2$.

7.3 Solution of the caterer problem using solver in excel

The caterer problem can be formulated as a linear programming problem and solved using SOLVER in excel. The formulation is given below.

Let x_{ij} be the quantity transported from supply i to demand j.

Minimize

$$z = 5(x_{11} + x_{12} + x_{13} + x_{14} + x_{15} + x_{16} + x_{17} + x_{18}) + 1.2(x_{23} + x_{34} + x_{45} + x_{56} + x_{67} + x_{78})$$
$$+ 0.8(x_{24} + x_{25} + x_{26} + x_{27} + x_{28} + x_{35} + x_{36} + x_{37} + x_{38} + x_{46} + x_{47} + x_{48} + x_{57} + x_{58} + x_{68})$$

Subject to

$$x_{11} + x_{12} + x_{13} + x_{14} + x_{15} + x_{16} + x_{17} + x_{18} \le 1064$$
$$x_{23} + x_{24} + x_{25} + x_{26} + x_{27} + x_{28} \le 90$$
$$x_{34} + x_{35} + x_{36} + x_{37} + x_{38} \le 140$$
$$x_{45} + x_{46} + x_{47} + x_{48} \le 84$$
$$x_{56} + x_{57} + x_{58} \le 95$$

$$x_{67} + x_{68} \leq 130$$
$$x_{78} \leq 170$$
$$x_{11} \geq 90$$
$$x_{12} \geq 140$$
$$x_{13} + x_{23} \geq 84$$
$$x_{14} + x_{24} + x_{34} \geq 95$$
$$x_{15} + x_{25} + x_{35} + x_{45} \geq 130$$
$$x_{16} + x_{26} + x_{36} + x_{46} + x_{56} \geq 170$$
$$x_{17} + x_{27} + x_{37} + x_{47} + x_{57} + x_{67} \geq 155$$
$$x_{18} + x_{28} + x_{38} + x_{48} + x_{58} + x_{68} + x_{78} \geq 200$$
$$x_{ij} \geq 0$$

Table 13. SOLVER solution to the caterer problem

	B	**C**	**D**	**E**	**F**	**G**
	supply	**demand**	**quantity**	**cost**	**net flow**	**flow cons**
2	1	1	90	5	355	1064
3	1	2	140	5	90	90
4	1	3	84	5	140	140
5	1	4	5	5	84	84
6	1	5	0	5	95	95
7	1	6	36	5	130	130
8	1	7	0	5	170	170
9	1	8	0	5	90	90
10	2	3	0	1.2	140	140
11	3	4	0	1.2	84	84
12	4	5	0	1.2	95	95
13	5	6	40	1.2	130	130
14	6	7	100	1.2	170	170
15	7	8	170	1.2	155	155
16	2	4	90	0.8	200	200
17	2	5	0	0.8		
18	2	6	0	0.8		
19	2	7	0	0.8		
20	2	8	0	0.8		
21	3	5	130	0.8		
22	3	6	10	0.8		
23	3	7	0	0.8		
24	3	8	0	0.8		
25	4	6	84	0.8		
26	4	7	0	0.8		
27	4	8	0	0.8		
28	5	7	55	0.8		
29	5	8	0	0.8		
30	6	8	30	0.8		
31						
32			cost	2466.2		

Explanation of obtaining Table 8 column entries.

1 Supply and demand. These show the supply source and demand day numbers as they appear in Table 8.
2 Quantity. These are the decision variables i.e. amounts of napkins to be bought/cleaned. Initially these values are all 0.
3 Cost. These show the costs per napkin as they appear in the objective function.
4 Net flow. These are the left hand sides of the constraints in the linear programming formulation. These are obtained from the column D quantities as follows.

F2: = SUM(D2:D9)
F3: = D10+SUM(D16:D20)
F4: = D11+SUM(D21:D24)
F5: = D12+SUM(D25:D27)
F6: = D13+SUM(D28:D29)
F7: = D14+D30
F8: = D15
F9: = D2
F10: = D3
F11: = D4+D10
F12: = D5+D16+D11
F13: = D6+D17+D21+D12
F14: = D7+D18+D22+D25+D13
F15: = D8+D19+D23+D26+D28+D14
F16: = D9+D20+D24+D27+D29+D30+D15

5 Flow conservations. These are the right hand sides of the constraints in the linear programming formulation.

When using SOLVER the objective cell is E32, the objective Min, the changing variable cells D2-D30 and the constraints F2-F8 <= G2-G8 and F9-F16 <= G9-G16. The Solving Method is Simplex LP and the "Make Unconstrained Variables Non-Negative" box is ticked.

When written in matrix form, the SOLVER solution shown in the above table is as shown below.

```
     [,1]  [,2] [,3] [,4]  [,5] [,6]  [,7]  [,8]  [,9]
[1,]   90  140   84    5     0   36     0     0  709
[2,]    0    0    0   90     0    0     0     0    0
[3,]    0    0    0    0   130   10     0     0    0
[4,]    0    0    0    0     0   84     0     0    0
[5,]    0    0    0    0     0   40    55     0    0
[6,]    0    0    0    0     0    0   100    30    0
[7,]    0    0    0    0     0    0     0   170    0
```

From the above output the following solution can be written down.

Table 14. Costs for SOLVER solution

Supply/day	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**Total**	**Cost per napkin**
new	90	140	84	5		36			355	5
fast						40	100	170	310	1.2
slow				90	130	94	55	30	399	0.8

Total cost = $355 \times 5 + 310 \times 1.2 + 399 \times 0.8 = 2466.2$.

The SOLVER solution is slightly different to the transportation solution obtained from lpSolve. However the costs for these two solutions are the same. This shows that the optimal solution for the caterer problem is not necessarily unique.

8 Conclusion

The maximum flow and minimum cost - maximum flow problems were defined and formulated as linear programming problems. Each of the problems considered was motivated by examples in the form of short explanations of published applications. A review of the algorithms used to solve the above mentioned problems was conducted. This included presentations (in the form of tables) of the complexity of the various algorithms.

The major part of the article was devoted to the practical implementation of the above mentioned methods. This involved explanations (using data obtained from given problems) of each of the following.

1 A manual solution of the maximum flow problem using the labelling algorithm.
2 A manual solution of the minimum cost-maximum flow problem using the Busacker-Gowen minimum-cost flow algorithm.
3 Solutions of the maximum flow and minimum cost-maximum flow problems using (i) R packages (ii) Pascal programs and (iii) SOLVER in an excel spreadsheet.
4 The minimum cost-maximum flow solutions to the assignment and transportation problems by (i) formulating them as linear programming problems and using R and (ii) using a Pascal program to find the solution.
5 The minimum cost – maximum flow solution to the shortest route problem by (i) formulating it as a linear programming problem and using R to find the solution and (ii) solving it using SOLVER in an excel spreadsheet.
6 The minimum cost – maximum flow solution to the caterer problem by (i) formulating this problem as a transportation problem and solving it with the use of R and (ii) using SOLVER in an excel spreadsheet.

Acknowledgements

The author expresses a word of thanks to anonymous reviewers for helpful suggestions that improved the quality of the article.

Competing Interests

Author has declared that no competing interests exist.

References

[1] Abdullah N, Hua TK. The application of the shortest path and maximum flow with bottleneck in traffic flow of Kota Kinabalu. Journal of Computer Science & Computational Mathematics. 2017;7(2):37-43.

[2] Kaanodiya KK, Rizwanullah M. Minimize traffic congestion: An application of maximum flow in dynamic networks. Journal of Applied Mathematics, Statistics and Informatics (JAMSI). 2012;8(1): 63-74.

[3] Schwartz BL. Possible winners in partially completed tournaments. SIAM Review. 1966;8(3):302-308.
DOI: 10.1137/1008062

[4] Winston WL. Operations research applications and algorithms. 4th Ed. Belmont CA, USA, Brooks/ColeThomson Learning; 2004.

[5] Dantzig GB. Linear programming and extensions. Princeton University Press, Princeton, NJ; 1963.

[6] Kuhn HW. The Hungarian method for the assignment problem. Naval Res. Logist. Quart. 1955; 2(1-2):83-97.
Available:CiteSeerX10.1.1.228.3906
DOI: 10.1002/nav.3800020109

[7] Lone MA, Mir SA, Wani MS. An application of assignment problem in agriculture using R. Journal of Scientific Research & Reports. 2017;13(2):1-5.

[8] Hultberg TH, Cardoso DM. The teacher assignment problem: A special case of the fixed charge transportation problem. European Journal of Operation Research. 1997;101:463-473.

[9] Feiyang Chen, Nan Chen, Hanyang Mao, Hanlin Hu. An application of bipartite matching in assignment problem. Chuangxinban Journal of Computing. 2018;1-2.
Available:https://arxiv.org/pdf/1902.00256.pdf

[10] Demaine E, Goldwasser S. Introduction to algorithms. Massachusetts Institute of Technology Hand Out 25; 2004.

[11] Szwarc W, Posner ME. The caterer problem. Operations Research. 1985;33(6):1215-1224.

[12] Ahuja RK, Magnanti TL, Orlin JB, Reddy MR. Applications of network optimization. Chapter in Handbooks in OR & MS, Elsevier. 1995;7.

[13] Harris TE, Ross FS. Fundamentals of a method for evaluating rail net capacities. Research Memorandum, Rand Corporation; 1955.

[14] Ford LR, Fulkerson DR. Maximal flow through a network. Canadian Journal of Mathematics. 1956;8:399-404.

[15] Dinitz Y. Algorithm for solution of a problem of maximum flow in a network with power estimation. Doklady Akademii Nauk SSSR. 1970;11:1277-1280.

[16] Edmonds J, Karp JM. Theoretical improvements in algorithmic efficiency for network flow problems. Journal of the ACM. 1972;19(2):248-264.

[17] Goldberg AV, Tarjan RE. A new approach to the maximum flow problem. Journal of the ACM. 1988;35(4):921-940.

[18] Goldberg AV, Rao S. Beyond the flow decomposition barrier. Journal of the ACM. 1998;45(5):783-797.

[19] King V, Rao S, Tarjan R. A faster deterministic maximal flow algorithm. Journal of Algorithms. 1994;17:447-474.

[20] Orlin JB. Max flows in O(nm) time, or better. STOC '13 Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing. 2013;765–774.
Available:CiteSeerX10.1.1.259.5759
DOI: 10.1145/2488608.2488705
ISBN: 9781450320290

[21] Sherman J. Nearly maximum flows in nearly linear time. Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science. 2013;263-269.

[22] Malhotra VM, Pramodh-Kumar SN, Maheshwari SN. An algorithm for finding maximum flows in networks. Information Processing Letters. 1978;7(6):277-278.

[23] Ahmed Faruque, Al-Amin Khan Md, Rahman Khan Aminur, Ahmed Syed Sabbir, Sharif Uddin Md. An efficient algorithm for finding maximum flow in a network-flow. Journal of Physical Sciences. 2014;19:41-50.

[24] Orlin JB. Max flows in $O(mn)$ time, or better; 2018.
Available:https://dspace.mit.edu/openaccess-disseminate/1721.1/8820

[25] Wikipedia maximum flow problem.
Available:https://en.wikipedia.org/wiki/Maximum_flow_problem

[26] Fulkerson DR. An out-of-kilter method for minimal cost flow problems. Journal of the Society for Industrial and Applied Mathematics. 1961;9(1):18-27.

[27] Busacker RG, Gowen PG. A procedure for determining a family of minimum cost network flow patterns. Operations Research Office Technical Report 15, John Hopkins University, Baltimore; 1961.

[28] Klein M. A primal method for minimum cost flows with applications to the assignment and transportation problems. Management Sci. 1967;14(3):205-220.

[29] Engquist M. A successive shortest path algorithm for the assignment problem. Research Report, Center for Cybernetic Studies (CCS) 375, University of Texas, Austin; 1980.

[30] Carpaneto G, Toth P. Primal-dual algorithms for the assignment problem. DAM. 1987;18:137-153.

[31] Orlin JB. A polynomial time primal network simplex algorithm for minimum cost flows. Mathematical Programming. 1997;78(2):109–129.

[32] Eroglu E. An application of the network simplex method for minimum cost flow problems. Balkan Journal of Mathematics. 2013;1:117-130.

[33] Tarjan RE. Dynamic trees as search trees via Euler tours, applied to the network simplex algorithm. Mathematical Programming. 1997;78:169-77.

[34] Orlin JB. A faster strongly polynomial minimum cost flow algorithm. Operations Research. 1993;41(2):338-350.

[35] Parpalea Mircea. Interactive tool for the successive shortest paths algorithm in solving the minimum cost flow problem. Bulletin of the Transilvania University of Brasov, Series III: Mathematics, Informatics, Physics. 2009;2(51):255-262.

[36] Sokkalingam PT, Ahuja RK, Orlin JB. New polynomial-time cycle-cancelling algorithms for minimum cost flows. Networks. 2000;36:53–63.

[37] Goldberg AV, Tarjan LE. Efficient maximum flow algorithms. Communications of the ACM. 2014;57:82-89.

[38] Syslo MM, Deo N, Kowalik JS. Discrete optimization algorithms with Pascal programs. Englewood Cliffs, New Jersey: Prentice-Hall; 1983.

[39] Moore EF. The shortest path through a maze. Proc. Int. Symp. On the Theory of Switching, Part II; 1957.

[40] Bellmann RE. On a routing problem. Quart. Appl. Math. 1958;16:87-90.

[41] Pape U. Algorithm 562: Shortest path lengths. ACM Trans. Math. Software. 1980;5:450-455.

[42] Panyotova G, Slavona SL. Modelling network flow by excel solver. Trakia Journal of Sciences. 2009;8(3):12-15.

Appendix

A1 Input and output code for MAXFLOW program

Programming: Pascal code

```
Program maxflow(input,output);
CONST     N = 11;
          MAX = 60;
TYPE ARRNN = ARRAY[1..N,1...N] OF INTEGER;
          ARRN = ARRAY[1..N] OF INTEGER;
     ARRMAX = ARRAY[1..MAX] OF INTEGER;
VAR S1,T1,I,J  :  INTEGER;
CAPA1,FLOW1 :  ARRNN;
```

The main body of the program succeeding the procedure is given below. This part is need for data entry and writing the results.

```
BEGIN (* MAIN PROGRAM*)
    S1 := 1;
    T1 := N;
    WRITELN('ENTER CAPACITIES');
     FOR I := 1 TO N DO
     BEGIN
        FOR J := 1 TO N DO
        BEGIN
           WRITE('CAPACITY ', ' I=',I:2, ' J=', J:2, '  ' );
           READLN(CAPA1[I,J])
        END;
     END;
     MAXFLOW(N,S1,T1,MAX,CAPA1,FLOW1);
     WRITELN('THE FLOW IS');
     FOR I := 1 TO N DO
     BEGIN
         FOR J := 1 TO N DO
         BEGIN
            WRITE('   ',FLOW1[I,J])
         END;
         WRITELN
     END;
END.
```

A2 Input and output code for BUSACKER program

Programming: Pascal code

```
Program minimum_cost(input,output);
CONST      N = 5;
TYPE ARRNN = ARRAY[1..N,1...N] OF INTEGER;
          ARRN = ARRAY[1..N] OF INTEGER;
VAR S1,T1,INF1,TARDETFLOW1,TOTALCOST1,I,J  :  INTEGER;
                                 COST1,CAPA1,FLOW1 :  ARRNN;
```

The main body of the program succeeding the procedure is given below. This part is need for data entry and writing the results.

```
BEGIN  (*MAIN PROGRAM*)
   S1 := 1;
   T1 := N;
   INF1 := 9999;
   TARGETFLOW1 := 31;
   WRITELN(‘ENTER CAPACITIES AND COSTS’);
    FOR I := 1 TO N DO
    BEGIN
         FOR J := 1 TO N DO
         BEGIN
             WRITE(‘CAPACITY ’, ‘ I=’,I:2, ‘ J=’, J:2, ‘  ’ );
             READLN(CAPA1[I,J])
         END;
   END;
   FOR I := 1 TO N DO
   BEGIN
        FOR J := 1 TO N DO
       BEGIN
           WRITE(‘COST ’, ‘ I=’,I:2, ‘ J=’, J:2, ‘  ’ );
           READLN(COST1[I,J])
       END;
   END;
  BUSACKER(N,S1,T1,INF1,TARGETFLOW1,COST1,CAPA1,FLOW1,TOTALCOST1);
   WRITELN(‘THE MIN COST FLOW IS’);
    FOR I := 1 TO N DO
    BEGIN
        FOR J := 1 TO N DO
        BEGIN
            WRITE(‘   ’, FLOW1[I,J])
        END;
        WRITELN
    END;
   WRITELN(‘THE TOTAL COST IS ’, TOTALCOST1);
END.
```

A3 Input and output code for TRANSPORT program

Programming: Pascal code

```
program transport(input,output);
CONST M=4;
          N=4;
TYPE ARRMN = ARRAY[1..M,1...N] OF INTEGER;
         ARRM = ARRAY[1..M] OF INTEGER;
        ARRN = ARRAY[1..N] OF INTEGER;
VAR       INF1 : LONGINT;
        I,J,KO1 : INTEGER;
              A1 : ARRM;
              B1 : ARRN;
         C1,X1 : ARRMN;
```

The main body of the program succeeding the procedure is given below. This part is needed for data entry and writing the results.

```
BEGIN  (* MAIN PROGRAM *)
   INF1 := 1000;
  WRITELN('COSTS');
   FOR I := 1 TO M DO
   BEGIN
       FOR J := 1 TO N DO
       BEGIN
          WRITE('COST ', ' I=',I:2, ' J=', J:2, '  ' );
          READLN(C1[I,J])
       END;
   END;
      FOR I := 1 TO M DO
      BEGIN
         WRITE('SUPPLY ', ' I=',I:2, '  ' );
         READLN(A1[I])
     END;
     FOR J := 1 TO N DO
      BEGIN
         WRITE('DEMAND ', ' J=',J:2, '  ' );
         READLN(B1[J])
     END;
     TRANSPORT(M,N,INF1,A1,B1,C1,X1,KO1);
     WRITELN('OPTIMAL SOLUTION');
     FOR I := 1 TO M DO
     BEGIN
         FOR J := 1 TO N DO
         BEGIN
            WRITE('    ', X1[I,J]:6 )
         END;
        WRITELN
    END;
    WRITELN('THE OPTIMAL COST IS ',KO1);
END.
```