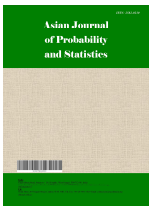

**Asian Journal of Probability and Statistics**

**7(3): 76-97, 2020; Article no.AJPAS.55363**
***ISSN: 2582-0230***

# The Out-of-Kilter Algorithm and Its Applications to Network Flow Problems

**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/v7i330187
Editor(s):
(1) Dr. Monika Hadas-Dyduch, University of Economics in Katowice, Poland.
(2) Dr. Manuel Alberto M. Ferreira, School of Technology and Architecture (ISTA), Portugal.
(3) Dr. S. M. Aqil Burney, University of Karachi, Pakistan.
Reviewers:
(1) Sabo John, Adamawa State University, Nigeria.
(2) Pasupuleti Venkata Siva Kumar, Vallurupalli Nageswara Rao Vignana Jyothi Institute of Engineering & Technology, India.
(3) Mubariz Nuriyev, Center for Scientific Research and Statistical Innovations, Russia.
Complete Peer review History: http://www.sdiarticle4.com/review-history/55363

***Review Article***

***Received: 20 January 2020***
***Accepted: 27 March 2020***
***Published: 20 June 2020***

## Abstract

The out-of-kilter algorithm, which was published by D.R. Fulkerson [1], is an algorithm that computes the solution to the minimum-cost flow problem in a flow network. To begin, the algorithm starts with an initial flow along the arcs and a number for each of the nodes in the network. By making use of Complementary Slackness Optimality Conditions (CSOC) [2], the algorithm searches for out-of-kilter arcs (those that do not satisfy CSOC conditions). If none are found the algorithm is complete. For arcs that do not satisfy the CSOC theorem, the flow needs to be increased or decreased to bring them into kilter. The algorithm will look for a path that either increases or decreases the flow according to the need. This is done until all arcs are in-kilter, at which point the algorithm is complete. If no paths are found to improve the system then there is no feasible flow. The Out-of-Kilter algorithm is applied to find the optimal solution to any problem that involves network flows. This includes problems such as transportation, assignment and shortest path problems. Computer solutions using a Pascal program and Matlab are demonstrated.

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

*Keywords:* *Network flow diagram; nodes; arcs; source; sink; capacity; residual capacity; node potentials; reduced cost; augmenting path; feasible solution; optimal solution; complementary slackness optimality condition; Kilter diagram; Out-of-Kilter algorithm; transportation problem; assignment problem; shortest path problem.*

# 1 Introduction

The Out-of-Kilter algorithm can be applied to solve the maximum flow and minimum cost – maximum flow problems as well as problems that can be formulated as such problems e.g. the transportation problem. The problem to be solved is displayed in the form of a network flow diagram (explained in the next section). The formulations of the maximum flow and minimum cost – maximum flow problems are shown.

A key result that is used when applying the Out-of-Kilter theorem is the Complementary Slackness Optimality Conditions (CSOC) theorem [2]. This theorem specifies the conditions for the optimality of the Out-of-Kilter theorem. The theorem as well as its use together with the Kilter diagram, that accompanies it, are discussed. The details of the steps to be followed when applying the Out-of-Kilter theorem are shown and some technical issues (return flow arc and setting of arc capacities) are discussed. An example is presented where the CSOC theorem is used to verify the optimality of a particular solution. The verification steps are shown.

Computer programs that are available in the literature are discussed. The following computer applications of the theorem using Pascal and Matlab programs are shown.

1. Transportation problem.
2. Assignment problem.
3. Shortest Path problem.

# 2 Network Terminology

A **network flow diagram** is a diagram consisting of points called **nodes** (vertices), where some of the nodes are joined by straight lines called **arcs** (edges).

The start node in the diagram is called the **source** and the end node the **sink**.

An **arc** $(i, j)$ is formed by connecting two nodes labelled $i$ and $j$ using a straight line.

A **forward** arc is an arc that contains a flow from source to sink. A **backward** (reverse) arc is an arc that contains a flow from the sink to source.

The **flow, lower and upper capacities** of the arc $(i, j)$ are denoted by $x_{ij}$, $\ell_{ij}$ and $u_{ij}$ respectively.

For arc, $(i, j)$ the **residual capacity** is $r_{ij} = u_{ij} - x_{ij}$ and for $(j, i)$ the arc it is $r_{ji} = x_{ij}$.

The **cost** for the arc $(i, j)$ is $c_{ij}$ and for $(j, i)$ the arc it is $-c_{ij}$.

The dual variables $\pi_i$ of the minimum cost flow problem associated with nodes $i = 1,2,\cdots,n$ are called the **node potentials** (prices).

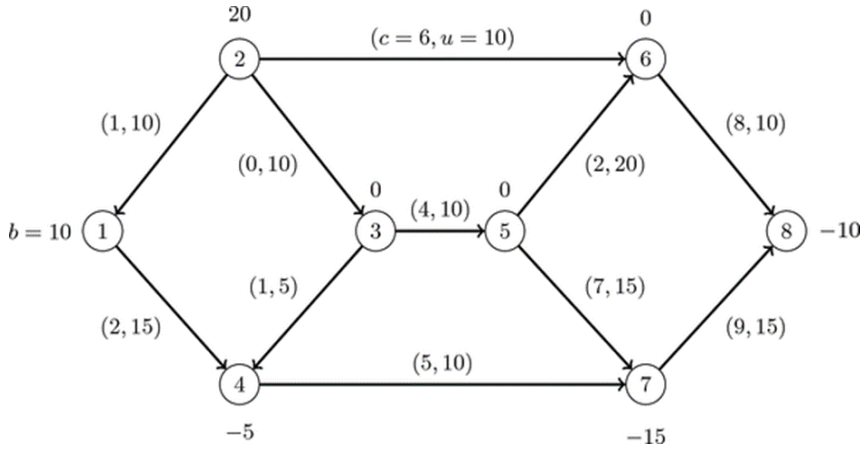


**Fig. 1. Network with costs, upper capacities and node potentials**

The node potentials are the numbers written next to each of the nodes. The amount $c_{ij}^{\pi} = c_{ij} + \pi_i - \pi_j$ is defined as the **reduced cost** of arc $(i, j)$. This can be interpreted as the change in the cost of arc $(i, j)$ as result of the effects of the node prices $\pi_i$ and $\pi_j$. An arc $(i, j)$ with $c_{ij}^{\pi} < 0$ can be thought of as productive and one with $c_{ij}^{\pi} > 0$ as unproductive. When $c_{ij}^{\pi} = 0$ it is neither productive nor unproductive.

An **augmenting path** (chain) is a simple path (a path that does not contain cycles) from the source to the sink of a graph using only arcs (edges) with positive capacities.

When seeking a solution to a network flow problem, the nodes between the source and sink nodes are given labels. **Labelling** a node is an entry of the form $(s, a)$, where $s$ denotes the sign (+ or –) and $a$ the amount of flow out of the node. A + sign means that the flow out of the node is increased and a – sign means it is decreased.

# 3 Problem Statement and Motivation

Consider a network with $m$ nodes (vertices) and $n$ arcs (edges).

**The maximum flow problem:**

If the objective is to determine the maximum flow that can be sent from node 1 (source) to node $m$ (sink) the problem can be formulated as

Maximize $\sum_{j=2}^{m} x_{1j}$ (maximize the total flow out of the source) or

$\sum_{j=1}^{m-1} x_{jm}$ (maximize the total flow into the sink)

subject to $\sum_{\substack{j=1 \\ j\neq i}}^{m} x_{ij} - \sum_{\substack{k=1 \\ k\neq i}}^{m} x_{ki} = 0$, $i = 1,2,\cdots,m$ (flow conservation constraints)

$$\ell_{ij} \le x_{ij} \le u_{ij}, \quad i, j = 1,2,\cdots,m, \text{ (capacity constraints).}$$

Often $\ell_{ij}$ is taken as 0 for all $(i, j)$.

**The minimum cost-maximum flow problem:**

If the objective is to find the minimum cost of the maximum flow $M$ from node 1 to node $m$ the problem can be formulated as

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

subject to $\sum_{\substack{j=1 \\ j\neq i}}^{m} x_{ij} - \sum_{\substack{k=1 \\ k\neq i}}^{m} x_{ki} = 0$, $i = 1,2,\cdots,m$ (flow conservation constraints)

$$\ell_{ij} \le x_{ij} \le u_{ij}, \quad i, j = 1,2,\cdots,m, \text{ (capacity constraints)}$$

$$\sum_{i=2}^{m} x_{1i} = M \text{ or } \sum_{i=1}^{m-1} x_{im} = M, \quad \text{(maximum flow constraint).}$$

In the above-mentioned formulations, the sums and inequalities are taken over existing arcs in the network.

A **feasible solution** to a network problem is a solution that satisfies all the constraints imposed.

An **optimal solution** to a network problem is a solution that satisfies all the constraints imposed and maximizes/minimizes the objective function.

Many algorithms for solving this problem were proposed over the past 6 decades. These include those by Fulkerson [1] – out-of-kilter algorithm, Busacker & Gowan [3] – Cheapest Path Augmentation, Klein [4] – Cycle cancelling, Engquist [5] – Successive shortest path, Carpaneto & Toth [6] – Primal-Dual, Goldberg & Tarjan [7] – Push/Relabel and Orlin [8] – Network Simplex.

For the following reasons the out-of-kilter algorithm for solving this problem will be discussed.

1. The algorithm applies an iterative solution strategy that satisfies the flow conservation constraints but might violate capacity constraints (feasibility) and optimality. The purpose of the algorithm is to decrease non-feasibility and move towards optimality. This is different from the approaches followed in other algorithms that are used to solve this problem.

2. The algorithm can be used to solve a wider range of network problems e.g. transportation, assignment, shortest path, caterer problems.

# 4 The Complementary Slackness Optimality Condition (CSOC) Theorem and Kilter Diagram

**Complementary Slackness Optimality Conditions (CSOC) theorem [9]:**

Consider a network with a set of arcs $A = \{(i, j) \in m \times m\}$, where $i$ is the "from" node and $j$ the "to" node. A feasible solution $x^*$ is an optimal solution of the minimum cost flow problem if and only if some set of node potentials $\pi = (\pi_1, \pi_2, \cdots, \pi_m)$ satisfy the following reduced cost optimality conditions for every arc $(i, j) \in A$.

I If $c_{ij}^{\pi} > 0$, then $x_{ij}^* = \ell_{ij}$.

II If $\ell_{ij} < x_{ij}^* < u_{ij}$, then $c_{ij}^{\pi} = 0$. If $\ell_{ij} \le x_{ij}^* \le u_{ij}$, then $c_{ij}^{\pi} = 0$ can also be used.

III If $c_{ij}^{\pi} < 0$, then $x_{ij}^* = u_{ij}$.

The lower bounds $\ell_{ij}, i, j \in A$ can be taken as 0.

Proof: See appendix.

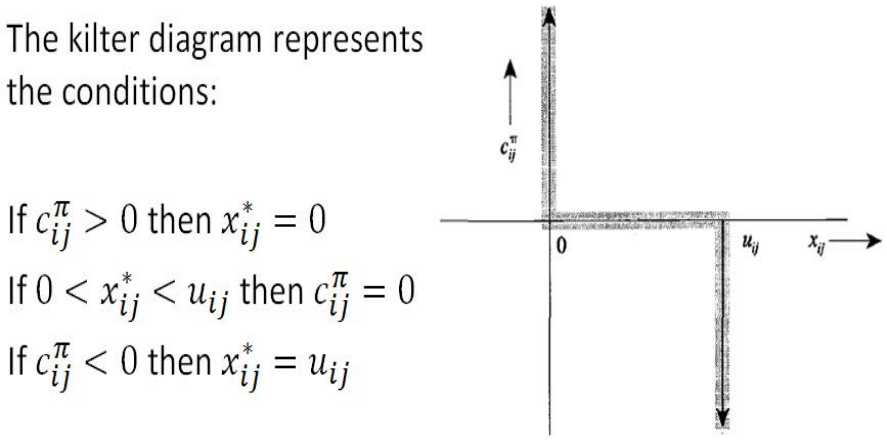


**Fig. 2. The Kilter diagram**

**Kilter diagram explanation:**

A particular solution $x = x^*$ to the minimum cost flow problem will generate flows $x_{ij}^*, (i, j) \in A$ along the arcs and node potential values $\pi_i, i = 1,2, \cdots, m$. For each arc $(i, j)$, $c_{ij}^{\pi} = c_{ij} + \pi_i - \pi_j$ can be calculated and the pair of values $(x_{ij}^*, c_{ij}^{\pi})$ plotted on the kilter diagram. If the plotted point for a particular arc is on the shaded line in the kilter diagram, the arc is labelled as "in kilter", if not it is labelled "out-of-kilter". The purpose of the out-of-kilter algorithm is to bring all out-of-kilter arcs in kilter, while not changing the kilter

status of the existing in-kilter arcs. An out-of-kilter arc can be brought in-kilter by (i) changing the flow values $x_{ij}, (i,j) \in A$ (horizontal axis value) or (ii) changing the reduced cost values $c_{ij}^{\pi}, (i,j) \in A$ by altering the node potential values $\pi_i, i = 1,2,\cdots,m$. Once all the arcs are in-kilter, the solution found is optimal.

# 5 The Out-of-Kilter Algorithm Steps [10,11]

**Initial values:**

Find an initial flow satisfying the flow conservation constraints and a set of node potentials $\pi_i, i = 1,2,\cdots,m$. All flows and potentials equal to 0 can be used for this purpose. Let $c_{ij}^{\pi} = c_{ij} + \pi_i - \pi_j$.

If $c_{ij}^{\pi} > 0$, $x_{\min}(i,j) = \min(x_{ij}, \ell_{ij}), x_{\max}(i,j) = \max(x_{ij}, \ell_{ij})$.

If $c_{ij}^{\pi} = 0$, $x_{\min}(i,j) = \min(x_{ij}, \ell_{ij}), x_{\max}(i,j) = \max(x_{ij}, u_{ij})$.

If $c_{ij}^{\pi} < 0$, $x_{\min}(i,j) = \min(x_{ij}, u_{ij}), x_{\max}(i,j) = \max(x_{ij}, u_{ij})$.

Arcs that are "in-kilter" should satisfy

I If $c_{ij}^{\pi} > 0$, then $x_{ij} = \ell_{ij}$.

II If $\ell_{ij} < x_{ij} < u_{ij}$, then $c_{ij}^{\pi} = 0$. If $\ell_{ij} \le x_{ij} \le u_{ij}$, then $c_{ij}^{\pi} = 0$ is also valid.

III If $c_{ij}^{\pi} < 0$, then $x_{ij} = u_{ij}$.

Any arcs that do not satisfy these conditions are labelled "out-of-kilter". While any arcs are out of kilter and the procedure described below is successful, repeat it for all out-of-kilter arcs.

**Attempt to update flows:**

Select an out-of-kilter arc ( $p,q$ ).

If $x_{pq} > x_{\min}(p,q)$, set $s = p, t = q, v = x_{pq} - x_{\min}(p,q)$.

If $x_{pq} < x_{\max}(p,q)$, set $s = q, t = p, v = x_{\max}(p,q) - x_{pq}$.

Attempt to find a flow-augmenting chain from $s$ to $t$ carry up to *v* additional units of flow without using the arc ( $p,q$ ) and without exceeding $x_{\max}(i,j)$ in forwarding arcs or falling below $x_{\min}(i,j)$ in backward

arcs. This can be achieved by starting the labelling algorithm with node $s$ labels $(-, v)$ and adhering to the values of $x_{\max}$ and $x_{\min}$ when adjusting the flows.

If **successful**, increase flow in the chain and increase/decrease flow in $(p, q)$ by $b_t$ the flow possible in the chain. If **not successful** (failing to label all the nodes on the path from $s$ to $t$), attempt to update node potentials as follows.

**Attempt to update node potentials:**

Let $L$ be the set of all arcs $(i, j)$ labelled at one end and not at the other such that $\ell_{ij} \le x_{ij} \le u_{ij}$.

For those arcs in $L$ labelled at $i$: if $c_{ij}^{\pi} > 0$, set $\delta_{ij} = c_{ij}^{\pi}$, otherwise set $\delta_{ij} = \infty$.

For those arcs in $L$ labelled at $j$: if $c_{ij}^{\pi} < 0$, set $\delta_{ij} = -c_{ij}^{\pi}$, otherwise set $\delta_{ij} = \infty$.

Set $\delta = \min\{\delta_{ij} : (i, j) \in L\}$.

If $\delta$ =0, then stop – no feasible flow.

Otherwise set $\pi_k = \pi_k + \delta$ for all unlabelled nodes and update $c_{ij}^{\pi}$ for all arcs labelled at one end and not the other. When the algorithm terminates, either all the arcs are in kilter and the current flows are optimal or no feasible solution exists.

# 6 The Out-of-Kilter Algorithm Application [12,13]

**Maximum flow – Ford-Fulkerson algorithm steps**

0 Initialize flow at $x = 0$
1 While an augmentation path $p$ exists
2 Do augment flow = $x$ = minimum residual capacity of arcs on path $p$.
3 Adjust residual capacities of arcs on the path $p$ and return to step 2.

The algorithm finishes when no more augmentation paths can be found.

The augmentation path can involve using forward arcs (increasing flow from source to sink) or backward arcs (increasing flow in the reverse direction from the sink to source).

**Out-of-kilter implementation – return flow arc:**

1. A return flow arc from the sink to the source (node $m$ to node1) is added to ensure circulation of the flow.

2. In the implementation of the maximum flow problem, the costs associated with all the arcs except the return flow arc are 0.

3. For the return flow arc, the cost is $-\varepsilon$ ($\varepsilon > 0$) for the maximum flow problem and 0 for the minimum cost-maximum flow problem.

4. When implementing the maximum flow problem the return flow arc has a lower capacity of 0 and an upper capacity of at least min( $s$,$t$ ), where $s$ = a total of upper capacities of all arcs connected to the source and $t$ = total of upper capacities of all arcs connected to the sink.

5. When implementing the minimum cost-maximum flow problem both the lower and upper capacities are $M$ = maximum flow.

# 7 Example of Maximum Flow and Minimum Cost-Maximum Flow – Manual Implementation

## 7.1 Problem statement

Cars enter a road network at node 1 and travel to node 6. The capacities and times to traverse for each of the arcs in the network are shown in the diagram below. The following is needed:

1. The maximum number of cars that can travel between nodes 1 and 6.
2. The maximum number of cars that can travel between nodes 1 and 6 in minimum time.

In the diagram that follows the first number in the rectangular boxes is the capacity (number of vehicles) of the route and the second number the time to traverse it (in minutes).

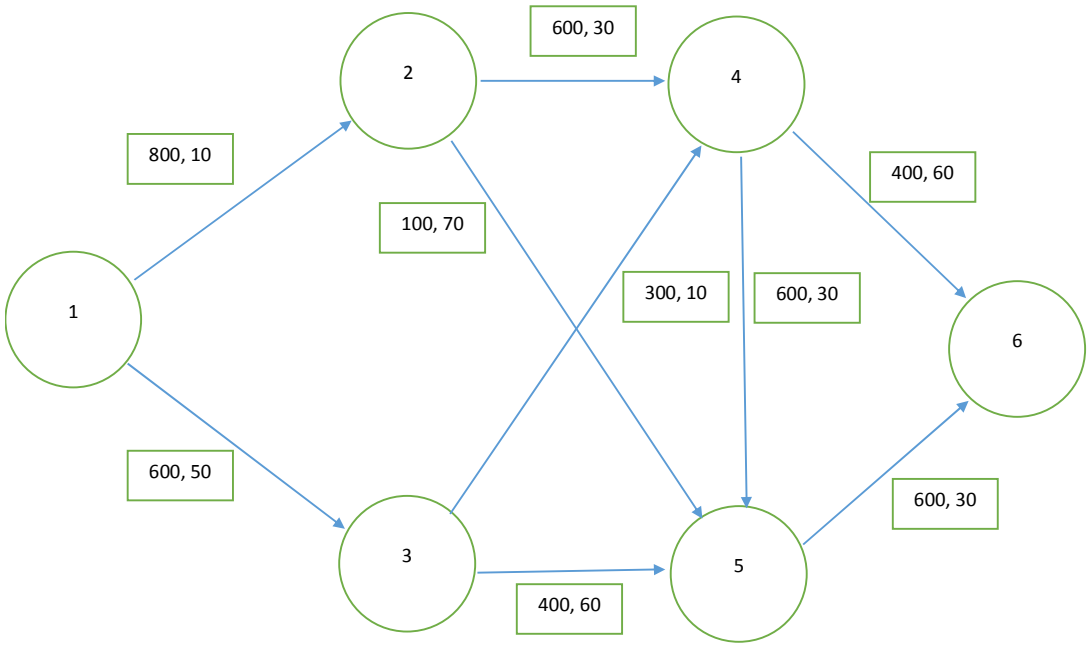


**Fig. 3. Capacities and times to traverse for roads in a network**

## 7.2 Writing down solutions by inspection of the network diagram

The return arc (6,1) with an upper capacity of min(1400, 1000) = 1000 is added.

The following feasible solutions can be written down by increasing flow along the following paths between nodes 1 and 6.

Solution 1 – paths and flows

| **Path** | **Flow** |
|---|---|
| 1-2-4-6 | min(800, 600, 400) = 400 |
| 1-3-5-6 | min(600, 400, 400) = 400 |
| 1-2-5-6 | min(400, 100, 200) = 100 |
| 1-2-4-5-6 | min(300, 200, 600, 100) = 100 |

Solution 2 – Flows and residuals

| arc | 1,2 | 1,3 | 2,4 | 2,5 | 3,4 | 3,5 | 4,5 | 4,6 | 5,6 | 6,1 |
|---|---|---|---|---|---|---|---|---|---|---|
| flow | 600 | 400 | 500 | 100 | 0 | 400 | 100 | 400 | 600 | 1000 |
| residual | 200 | 200 | 100 | 0 | 300 | 0 | 500 | 0 | 0 | 0 |

Total time = 117 000

Solution 3 – modified solution 1

Add path 1-2-4-5-6 with forward flow 100 and path 6-5-3-1 with backward flow 100.

| arc | 1,2 | 1,3 | 2,4 | 2,5 | 3,4 | 3,5 | 4,5 | 4,6 | 5,6 | 6,1 |
|---|---|---|---|---|---|---|---|---|---|---|
| flow | 700 | 300 | 600 | 100 | 0 | 300 | 200 | 400 | 600 | 1000 |
| residual | 100 | 300 | 0 | 0 | 300 | 100 | 400 | 0 | 0 | 0 |

Total time = 113 000

Solution 4 – modified solution 1

Add path 1-3-4-5-6 with forward flow 200 and path 6-5-3-1 with backward flow 200.

| arc | 1,2 | 1,3 | 2,4 | 2,5 | 3,4 | 3,5 | 4,5 | 4,6 | 5,6 | 6,1 |
|---|---|---|---|---|---|---|---|---|---|---|
| flow | 600 | 400 | 500 | 100 | 200 | 200 | 300 | 400 | 600 | 1000 |
| residual | 200 | 200 | 100 | 0 | 100 | 200 | 300 | 0 | 0 | 0 |

Total time = 113 000

Solution 5 – modified solution 2

Increase (1,2) by 100, decrease (1,3) by 100.
Increase (2,4) and (3,4) by 400 (combined) and decrease (3,5) by 400.
Increase (4,5) by 400.

| arc | 1,2 | 1,3 | 2,4 | 2,5 | 3,4 | 3,5 | 4,5 | 4,6 | 5,6 | 6,1 |
|---|---|---|---|---|---|---|---|---|---|---|
| flow | 700 | 300 | 600 | 100 | 300 | 0 | 500 | 400 | 600 | 1000 |
| residual | 100 | 300 | 0 | 0 | 0 | 400 | 100 | 0 | 0 | 0 |

Total time = 107 000

## 7.3 Verification of optimality of solutions

**Maximum flow problem – solution 1**

arcs where $x_{ij} = \ell_{ij}$ : (3,4)

arcs where $\ell_{ij} < x_{ij} < u_{ij}$: (1,2), (1,3), (2,4), (4,5)

arcs where $x_{ij} = u_{ij}$: (2,5), (3,5), (4,6), (5,6), (6,1)

Using $c_{ij}^{\pi} = c_{ij} + \pi_i - \pi_j$ and $c_{ij} = 0$ for all arcs $(i, j)$, $\pi_1 = 0$ and the CSOC theorem the following should hold for an optimal solution.

$\pi_1 - \pi_2 = 0$, $\pi_1 - \pi_3 = 0$, $\pi_2 - \pi_4 = 0$, $\pi_4 - \pi_5 = 0 \Rightarrow \pi_i = 0$ for $i = 1,2,\cdots,5$.
$\pi_4 - \pi_6 \le 0$, $\pi_5 - \pi_6 \le 0$, $\pi_6 - \pi_1 \le 0$
$\pi_6 \le 0$ and $\pi_6 \ge 0 \Rightarrow \pi_6 = 0$.

There are no inconsistencies in the values of the $\pi$'s (agree with optimality conditions). Since the other 3 solutions are all feasible and have the same maximum flow as solution 1, they are also optimal solutions.

**Minimum cost-maximum flow problem – solution 1**

The total flow has to be specified. Since this is treated as a maximum flow minimum-cost problem, the flow as found in the solution to the maximum flow problem will be used. As an initial solution to the problem, the solution 1 to the maximum flow problem will be used. Since this a feasible solution, the optimality of the solution needs to be checked. For this solution to be optimal

$c_{ij} + \pi_i - \pi_j = 0$ for arcs (1,2), (1,3), (2,4), (4,5)
$< 0$ for arcs (2,5), (3,5), (4,6), (5,6),
$> 0$ for arcs (3,4), (6,1)

From the above $10 + \pi_1 - \pi_2 = 0$, $50 + \pi_1 - \pi_3 = 0$, $30 + \pi_2 - \pi_4 = 0$, $30 + \pi_4 - \pi_5 = 0$,
$70 + \pi_2 - \pi_5 < 0$, $60 + \pi_3 - \pi_5 < 0$, $60 + \pi_4 - \pi_6 < 0$, $30 + \pi_5 - \pi_6 < 0$,
$10 + \pi_3 - \pi_4 > 0$, $\pi_6 - \pi_1 > 0$.

From the first 4 equations assuming $\pi_1 = 0$: $\pi_2 = 10$, $\pi_3 = 50$, $\pi_4 = 40$, $\pi_5 = 70$.

From the next 4 inequalities: $\pi_5 > 110$, $\pi_6 > 100$.

Since $\pi_5 = 70$ is inconsistent with $\pi_5 > 110$, this solution is not optimal.

**Minimum cost-maximum flow problem – solution 4**

$c_{ij} + \pi_i - \pi_j = 0$ for arcs (1,2), (1,3), (4,5)
$< 0$ for arcs (2,4), (2,5), (3,4), (4,6), (5,6)
$> 0$ for arcs (3,5), (6,1)

From the above $10 + \pi_1 - \pi_2 = 0$, $50 + \pi_1 - \pi_3 = 0$, $30 + \pi_4 - \pi_5 = 0$, $30 + \pi_2 - \pi_4 < 0$,
$70 + \pi_2 - \pi_5 < 0$, $10 + \pi_3 - \pi_4 \le 0$, $60 + \pi_4 - \pi_6 < 0$, $30 + \pi_5 - \pi_6 \le 0$, $\pi_6 - \pi_1 > 0$

$60+\pi_3-\pi_5>0$ (upper bound solutions can also be equalities).

From the first 3 equations assuming $\pi_1=0$: $\pi_2=10$, $\pi_3=50$, $\pi_5-\pi_4=30\Rightarrow\pi_5=\pi_4+30$.

From the next 5 inequalities: $\pi_4-\pi_2>30\Rightarrow\pi_4>40$, $\pi_5-\pi_2>70\Rightarrow\pi_5>80$,
$\pi_4-\pi_3\geq 10\Rightarrow\pi_4\geq 60$, $\pi_6-\pi_4>60$, $\pi_6-\pi_5\geq 30$.

For the last 2 inequalities: $\pi_5-\pi_3<60\Rightarrow\pi_5<110$, $\pi_6-\pi_1>0$.

From $\pi_4>40$ and $\pi_4\geq 60$ it follows that $\pi_4=60$ and $\pi_5=\pi_4+30=60+30=90$,
$\pi_6=\pi_5+30=90+30=120$.

There are no inconsistencies in the above solutions. Therefore the solution is optimal.

The computer solution to the above-mentioned problem is shown in the appendix.

# 8 Computer Programs to Implement the Out-of-Kilter Algorithm [14]

Bray and Witzgall [15] suggested a procedure called NETFLOW [16] to determine the minimum cost flow in a network using the Out-of-Kilter algorithm. Later in 1970 [17] they also published a correction to the original algorithm.

Clasen [18] published an ALGOL procedure that executes the Out-of-Kilter algorithm. The Fortran program written by the Share Distribution Agency [19] is designed to save space by arranging arcs so that the source nodes are in order.

Kindler [20] published a Fortran program called OKAY that uses the Out-of-Kilter algorithm to allocate flows in a network to minimize its total cost of flow. The program uses a subroutine called PACKUP that constructs a packed list of arcs entering each node and a subroutine called KILT to implement the algorithm.

Woolsey and Swanson [21] wrote an out-of-kilter algorithm in Fortran and illustrated its use with an assignment problem.

Smith [22] wrote programs that execute the Out-of-Kilter algorithm in Pascal and Basic. Shen [23] pointed out that these programs contain some errors and modified them to correct these errors.

Du Preez and Van Der Merwe [24] described the implementation of a user-friendly trans-shipment program (written in Pascal) based on the Out-of-Kilter algorithm. The validation of the program, time tests and applications were also discussed.

Nowicki [25] wrote a program in Matlab to implement the Out-of-Kilter algorithm. The data input to the program is using a file with the 3 matrices (cost, upper bound, lower bound). These $m\times m$ matrices denoted by $c,u$ $\ell$ and respectively contain the cost, upper bound and lower bound associated with each of the arcs $A=\{(i,j)\in m\times m\}$. The functions OOK (with parameters $c,u$ and $\ell$ that calculates the optimal flow in the various arcs) and koszt (calculates the total cost of the optimal solution) need to be called to get a solution. If there is only one optimal solution the program implements the algorithm correctly, but for more than one optimal solution for an assignment problem, it gets into an infinite loop. The tie in optimal solutions can be removed by adding small positive numbers (all different) to each of the values in the cost matrix.

Examples of the implementation of the maximum flow and minimum cost-maximum flow problems are shown in the appendix. The implementations of the corrected Pascal program of Smith [22] and a Matlab translation of the COBOL program by Bray and Witzgall [15] are illustrated.

The Out-of-Kilter algorithm can be used to determine the optimal flow of any problem that can be formulated as a network flow problem. This includes the transportation problem, the assignment problem and the shortest route problem.

Computer solutions to maximum flow and minimum cost – maximum flow problems and these problems are shown in the appendix.

# 9 The Out-of-Kilter Algorithm Applied to the Transportation Problem

At the $i$ th supply point, an amount at most $a_i, i = 1,2,\cdots,I$ is available. At the $j$ th demand point, an amount at least $b_j, j = 1,2,\cdots,J$ is required. In the diagram, each supply point node is connected to each demand point node by an arc ( $IJ$ arcs). Each of these arcs has a lower bound of 0, an upper bound of $\infty$ and a cost per unit as given in the problem. All the supply points' nodes (plants 1, 2 and 3) are connected to the source node. Each of these arcs has a lower bound of 0, an upper bound of the amount available and a cost per unit of 0. Similarly, all the demand points' nodes (regions 1, 2, 3 and 4) are connected to the sink node. Each of these arcs has a lower bound of the amount demanded, an upper bound of $\infty$ and a cost per unit of 0. If the total supply is greater than the total demand, a dummy destination **(dummy column)** with demand equal to the supply surplus is added. If the total demand is greater than the total supply, a dummy source (dummy row) with supply equal to the demand surplus is added. Because no shipment is made in case of a dummy source and dummy destination, the cost per unit for the dummy column and dummy row are assigned zero values. The minimum cost solution is sought. An additional arc is created where the demand node is connected to the source node with lower and upper bound total supply = total demand and cost per unit of 0.

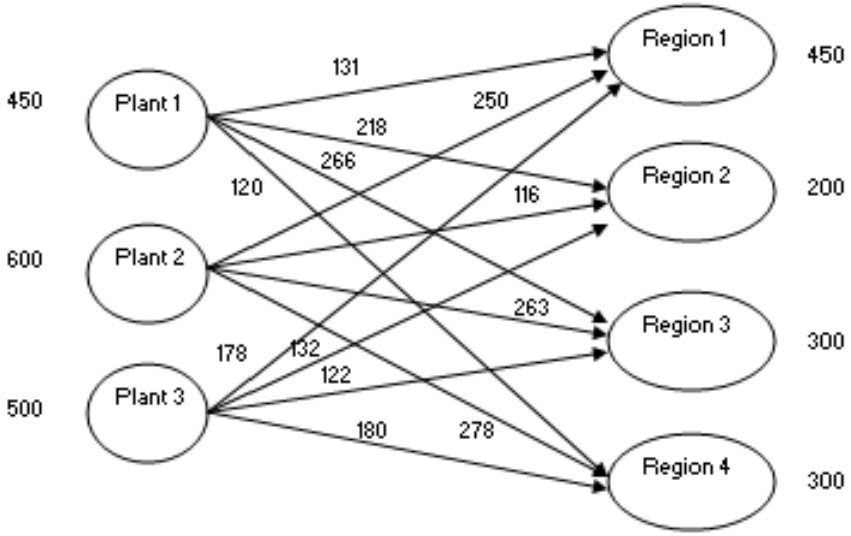


**Fig. 4. Network representation of a transportation problem**

The computer solution to the above-mentioned problem, using the Matlab implementation of the Netflow Out-of-Kilter algorithm, is shown in the appendix.

# 10 The Out-of-Kilter Algorithm Applied to the ASSIGNMENT Problem

Assignment models are used to assign members of one set to members of another set in the least cost or maximum profit manner e.g. $n$ men are to be assigned to $n$ jobs (one man per job) such that the total time taken to perform the jobs is a minimum. This can be seen as a transportation problem with $n$ supply points (men) and $n$ demand points (jobs). In the diagram, each supply point node is connected to each demand point node by an arc ( $n^2$ arcs). Each of these arcs has a lower bound of 0, an upper bound of 1 and a time as given in the problem. All the supply points' nodes are connected to the source node. Each of these arcs has a lower bound and upper bound of 1 and a time of 0. Similarly, all the demand points' nodes are connected to the sink node. Each of these arcs has a lower bound and upper bound of 1 and a time of 0. The arc connecting the sink node to the source node has lower and upper bound of $n$ and a cost of 0.

When the number of men (jobs) exceeds the number of jobs (men), dummy jobs (men) are created with zero costs and the solution proceeds as in the balanced case (men = jobs). In the case of a maximization assignment problem, the signs of all the profits are changed and the problem still treated as a minimization assignment problem.

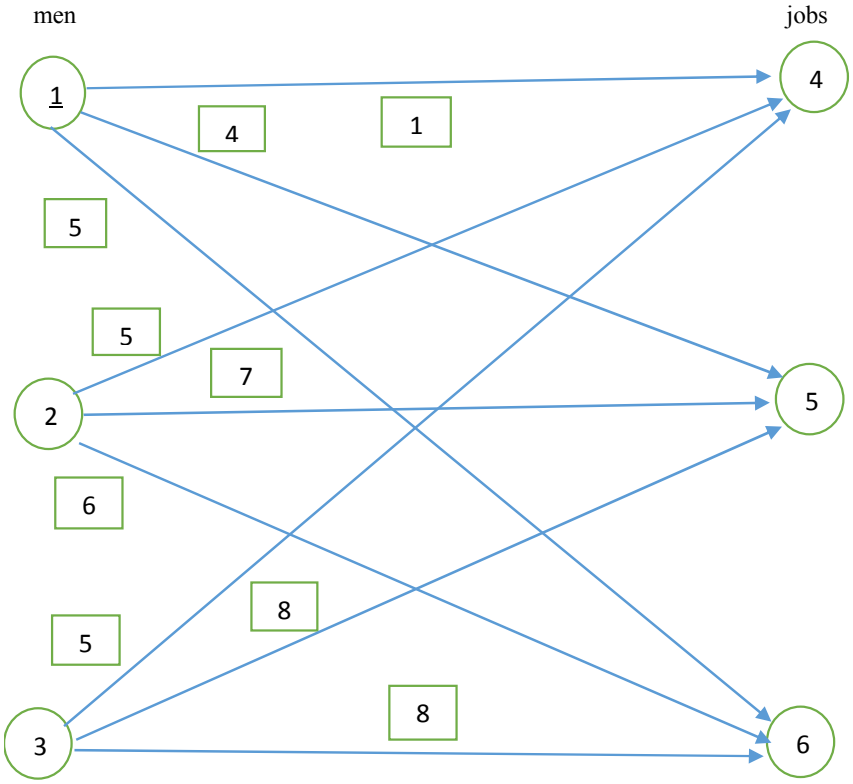


**Fig. 5. Network representation of an assignment problem**

In the above diagram, there are 3 men (1, 2 and 3) that are to be assigned to 3 jobs (4, 5 and 6). The time taken for each man-job combination is printed in the rectangle above the arrow that connects the man and job nodes. The computer solution to the above-mentioned problem, using the Matlab implementation of the Netflow Out-of-Kilter algorithm, is shown in the appendix.

## 11 The Out-of-Kilter Algorithm Applied to the SHORTEST PATH Problem

The shortest distance between a starting point (in this diagram Los Angeles) and endpoint (in this diagram St. Louis) is sought. This can be treated as a minimum cost problem. The cost associated with an arc that connects nodes shown on the diagram is the distance between the nodes that are connected by the arc. The lower and upper bounds of all arcs are 0 (route not selected) and 1 (the route is selected) respectively. An arc that connects the endpoint (in this case St. Louis) with the start point (in this case Los Angeles) with cost 0 and lower and upper bounds of 1 is added to the network.

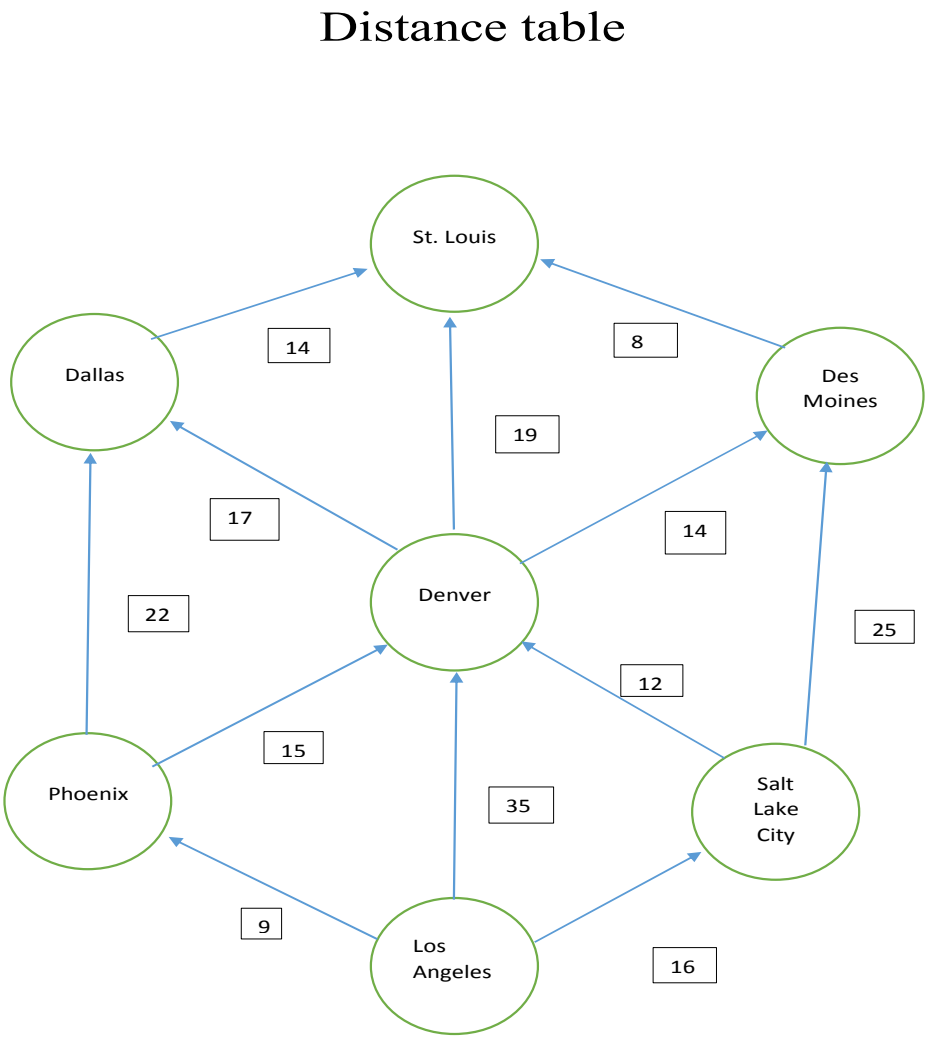


**Fig. 6. Network representation of the shortest path problem**

The computer solution to the above-mentioned problem, using the Matlab implementation of the Netflow Out-of-Kilter algorithm, is shown in the appendix.

## 12 Conclusion

The Out-of-Kilter algorithm is a procedure that can solve the maximum flow problem, the minimum cost - maximum flow problem and any problems involving network flow that can be formulated as these problems. When applying the algorithm, the Complementary Slackness Optimality Conditions (CSOC) theorem is used to classify arcs in the network as either "Out-of-Kilter" or "In-Kilter". "Out-of-Kilter" arcs are adjusted. An optimal solution has been found when all the arcs are "In-Kilter". Both the theory and computing implementation of the algorithm and its applications are demonstrated.

## Competing Interests

Author has declared that no competing interests exist.

## References

[1] 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.

[2] Ahuja RK, Magnanti TL, Orlin JB. Network flows theory, algorithms and applications. Upper Saddle River, New Jersey, Prentice-Hall; 1993.

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

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

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

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

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

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

[9] Vaidyanathan B, Ahuja RK, Magnanti TL, Orlin JB. Minimum cost flow problem. Chapter in Handbook of Graph Theory, Combinatorial Optimization and Algorithms, Thulasiraman K, Arumugam S, Brändstadt A & Nishizeki T (Editors), Boca Raton, FL. CRC Press; 2016.

[10] Dowsland K. Classical Techniques. 19-68, In Search Methodologies Introductory Tutorials in Optimization and Decision Support Techniques, Burke EK & Kendall G (Editors). New York, Springer Science + Business Media; 2005.

[11] Durbin EP, Kroenke DM. The out-of-kilter algorithm: A primer. United States Air Force Project RAND, Memorandum RM-5472-PR; 1967.

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

[13] Cormen TH, Leiserson CE, Rivest RL, Stein C. Introduction to algorithms, 2nd Ed. Cambridge, Massachusetts, MIT Press; 2001.

[14] Barr R, Glover F, Klingman G. An improved version of the out-of-kilter method and a comparative study of computer codes. Research Report C.S. 102, University of Texas, Austin, Texas; 1972.

[15] Bray TA, Witzgall C. Algorithm 336: Netflow. Comm. ACM. 1968;11:631-632.

[16] Briggs WA. Algorithm 248: Netflow. Comm. ACM. 1965;8(2):103-104.

[17] Bray TA, Witzgall C. Remark on Algorithm 336: Netflow. Comm. ACM. 1970;13:192.

[18] Clasen RJ. The numerical solution of network problems using the out-of-kilter algorithm. The United States Air Force Project RAND, Memorandum RM-5456-PR; 1968.

[19] Share Distribution Agency. Out-of-Kilter network routine, Share Distribution 3536, Share Distribution Agency, Hawthorne, New York; 1967.

[20] Kindler J. The out-of-kilter algorithm and some of its applications in water resources. International Institute for Applied Systems Analysis (IIASA) Working Paper WP-75-019; 1975.

[21] Woolsey RED, Swanson HS. Operations research for immediate application. A Quick and Dirty Manual, Harper & Row, New York; 1975.

[22] Smith DK. Network optimization practice: A computational guide. Chichester, Ellis Horwood; 1982.

[23] Shen Z. Log truck scheduling by network programming. Thesis Submitted to the Department of Forest Engineering for Degree of Master of Forestry, Univ. of Washington, Seattle, WA; 1989.

[24] Du Preez ND, Van Der Merwe. Die ontwikkeling van 'n gebruikersvriendelike transskeppingpakket vir IBM-aanpasbare persoonlike rekenaars. SA Journal of Industrial Engineering. 1988;2(2):25-39.

[25] Nowicki M. Adding out-of-kilter algorithm; 2014.
Available:https://github.com/MichalNowicki/NumericalAlgebraCodes/blob/master/OOKmatlab/OOK.m

# Appendix

### A1 Proof of the Complementary Slackness Optimality Conditions (CSOC) theorem

The following result by Vaidyanathan, Ahuja, Magnanti & Orlin [9] taken from the textbook on Graph Theory can be found by making use of the fact that the objective functions of the primal and dual minimum cost flow problems are equal. From this result it follows that

$\max(0,-c_{ij}^{\pi})u_{ij} = -c_{ij}^{\pi}x_{ij}^{*}$ for every arc $(i,j) \in A$.

I If $c_{ij}^{\pi} > 0,$ the left hand side of the above is 0. Therefore the right hand side can only be 0 if $x_{ij}^{*} = 0$.

II If $0 < x_{ij}^{*} < u_{ij}$, it must be that $c_{ij}^{\pi} = 0$. Otherwise the right hand side of will be negative.

III If $c_{ij}^{\pi} < 0,$ the left hand side of (A) is $-c_{ij}^{\pi} u_{ij}$ . To have equality in the above, $x_{ij}^{*} = u_{ij}$ must hold.

**A2 Examples of implementation of maximum flow and minimum cost-maximum flow problems**

The implementations of the corrected Pascal program of Smith [22] and a Matlab translation of the COBOL program, when solving the problem in 7.1, will be illustrated.

**Maximum flow solution using Pascal.**

**Input**

Data are input into the program in an interactive manner as shown below.

Input number of nodes and number of arcs 6 10
Input details of the arcs in the following order:
Start node Finish node Lower bound Upper bound Cost

| | S | F | L | U | C |
|---|---|---|---|---|---|
| Arc number 1 | 1 | 2 | 0 | 800 | 0 |
| . . . . | | | | | |
| Arc number 9 | 5 | 6 | 0 | 600 | 0 |
| Arc number 10 | 6 | 1 | 0 | 1000 | –1 |

**Output**

| Arc number | Start node | Finish node | Lower bound | Upper bound | Cost | Optimal flow |
|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 0 | 800 | 0 | 600 |
| 2 | 1 | 3 | 0 | 600 | 0 | 400 |
| 3 | 2 | 4 | 0 | 600 | 0 | 500 |
| 4 | 2 | 5 | 0 | 100 | 0 | 100 |
| 5 | 3 | 4 | 0 | 300 | 0 | 0 |
| 6 | 3 | 5 | 0 | 400 | 0 | 400 |
| 7 | 4 | 5 | 0 | 600 | 0 | 100 |
| 8 | 4 | 6 | 0 | 400 | 0 | 400 |
| 9 | 5 | 6 | 0 | 600 | 0 | 600 |
| 10 | 6 | 1 | 0 | 1000 | -1 | 1000 |

| Node number | Pi(n) |
|---|---|
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |

The upper bound for arc (6,1) is taken as min (800+600, 400+600) = 1000, but can be taken as any integer $\geq$ 1000.

**Minimum cost-maximum flow solution using Pascal.**

**Input**

Input number of nodes and number of arcs 6 10
Input details of the arcs in the following order:
Start node Finish node Lower bound Upper bound Cost

| | S | F | L | U | C |
|---|---|---|---|---|---|
| Arc number 1 | 1 | 2 | 0 | 800 | 10 |
| . . . . | | | | | |
| Arc number 9 | 5 | 6 | 0 | 600 | 30 |
| Arc number 10 | 6 | 1 | 1000 | 1000 | 0 |

**Output**

| Arc number | Start node | Finish node | Lower bound | Upper bound | Cost | Optimal flow |
|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 0 | 800 | 10 | 700 |
| 2 | 1 | 3 | 0 | 600 | 50 | 300 |
| 3 | 2 | 4 | 0 | 600 | 30 | 600 |
| 4 | 2 | 5 | 0 | 100 | 70 | 100 |
| 5 | 3 | 4 | 0 | 300 | 10 | 300 |
| 6 | 3 | 5 | 0 | 400 | 60 | 0 |
| 7 | 4 | 5 | 0 | 600 | 30 | 500 |
| 8 | 4 | 6 | 0 | 400 | 60 | 400 |
| 9 | 5 | 6 | 0 | 600 | 30 | 600 |
| 10 | 6 | 1 | 1000 | 1000 | 0 | 1000 |

| Node number | Pi(n) |
|---|---|
| 1 | 0 |
| 2 | 10 |
| 3 | 50 |
| 4 | 60 |
| 5 | 90 |
| 6 | 120 |

**Maximum flow solution Matlab.**

**Input**

The function used is a Matlab translation of function called Netflow (using the algorithm described by Bray and Witzgall [15] ) with arguments iNode (vector of start nodes), eNode (vector of end nodes), cost (vector of costs per unit) , lo (vector of lower bounds), hi (vector of upper bounds) and nNodes (number of nodes).

```
iNode = [1 1 2 2 3 3 4 4 5 6];
eNode = [2 3 4 5 4 5 5 6 6 1];
cost  = [0 0 0 0 0 0 0 0 0 -1];
lo    = [0 0 0 0 0 0 0 0 0 0];
hi    = [800 600 600 100 300 400 600 400 600 9999];
nNodes = 6;
```

1 Only the starting and end nodes of the arcs in the network are specified.
2 The cost for arc (6,1) is -1, while all other costs are 0.
3 The lower bounds for all arcs are 0.
4 The upper bound for arc (6,1) is a large number 9999, but can be any integer $\geq$ min (800+600, 400+600) = 1000.

**Output**

```
netProblems(1)
out = iNode eNode Lower Upper Cost Flow
        1     2     0     800    0   600
        1     3     0     600    0   400
        2     4     0     600    0   500
        2     5     0     100    0   100
        3     4     0     300    0     0
        3     5     0     400    0   400
        4     5     0     600    0   100
        4     6     0     400    0   400
        5     6     0     600    0   600
        6     1     0    9999   -1  1000
```

The starting and end nodes are shown in the first 2 columns and the optimal flows in the last column.

**Minimum cost-maximum flow solution Matlab.**

The same function as for maximum flow is called, but the input is changed.

**Input**

```
iNode = [1 1 2 2 3 3 4 4 5 6];
eNode = [2 3 4 5 4 5 5 6 6 1];
cost  = [10 50 30 70 10 60 30 60 30 0];
lo    = [0 0 0 0 0 0 0 0 0 1000];
hi    = [800 600 600 100 300 400 600 400 600 1000];
nNodes = 6;
```

1 Only the starting and end nodes of the arcs in the network are specified.
2 The cost for arc (6,1) is 0, while all other costs are as given in the problem.
3 The lower bounds for all arcs except arc (6,1) are 0.
4 The upper bound for all arcs except arc (6,1) are as given.
5 For arc (6,1) the lower and upper bounds are the maximum flow = 1000.

**Output**

```
      netProblems(2)
out = iNode eNode Lower Upper Cost Flow
        1     2     0     800    10   700
        1     3     0     600    50   300
        2     4     0     600    30   600
        2     5     0     100    70   100
        3     4     0     300    10   300
        3     5     0     400    60     0
        4     5     0     600    30   500
        4     6     0     400    60   400
        5     6     0     600    30   600
        6     1  1000    1000     0  1000
```

The starting and end nodes shown in the first 2 columns and the optimal flows in the last column.

**Transportation problem (section 9) solution using Matlab.**

**Input**

```
iNode = [1 1 1    2 2 2 2   3 3 3 3   4 4 4 4   5 6 7 8   9];
eNode = [2 3 4 5 6 7 8 5 6 7 8 5 6 7 8 9 9 9 9 1];
cost  = [0 0 0 131 218 266 120 250 116 263 278 178 132 122 189 0 0 0 0 0];
 lo   = [450 300 500 0 0 0 0 0 0 0 0 0 0 0 0 0 450 200 300 300 1250];
 hi   = [450 300 500 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 450 200 300 300
1250];
nNodes = 9;
```

**Output**

```
netProblems(3)
out = iNode eNode Lower Upper Cost  Flow
        1     2     450   450    0   450
        1     3     300   300    0   300
        1     4     500   500    0   500
        2     5       0  9999  131   150
        2     6       0  9999  218     0
        2     7       0  9999  266     0
        2     8       0  9999  120   300
        3     5       0  9999  250   100
        3     6       0  9999  116   200
        3     7       0  9999  263     0
        3     8       0  9999  278     0
        4     5       0  9999  178   200
        4     6       0  9999  132     0
        4     7       0  9999  122   300
        4     8       0  9999  189     0
        5     9     450   450    0   450
        6     9     200   200    0   200
        7     9     300   300    0   300
        8     9     300   300    0   300
```

**Solution:**

| supply | demand | amount | cost per unit | total cost |
|---|---|---|---|---|
| 2 | 5 | 150 | 131 | 19650 |
| 2 | 8 | 300 | 120 | 36000 |
| 3 | 5 | 100 | 250 | 25000 |
| 3 | 6 | 200 | 116 | 23200 |
| 4 | 5 | 200 | 178 | 35600 |
| 4 | 7 | 300 | 122 | 36600 |

Total cost = 176 050.

**Assignment problem (section 10) solution using Matlab.**

**Input**

```
iNode = [1 1 1 2 2 2 3 3 3 4 4 4 5 6 7 8];
eNode = [2 3 4 5 6 7 5 6 7 5 6 7 8 8 8 1];
cost  = [0 0 0 1 4 5 5 7 6 5 8 8 0 0 0 0];
lo    = [1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 3];
```

```
hi    = [1 1 1 9999 9999 9999 9999 9999 9999 9999 9999 9999 1 1 1 3];
nNodes = 8;
```

**Output**

```
netProblems(4)
out = iNode eNode Lower Upper Cost Flow
          1     2     1     1     0     1
          1     3     1     1     0     1
          1     4     1     1     0     1
          2     5     0  9999     1     0
          2     6     0  9999     4     1
          2     7     0  9999     5     0
          3     5     0  9999     5     0
          3     6     0  9999     7     0
          3     7     0  9999     6     1
          4     5     0  9999     5     1
          4     6     0  9999     8     0
          4     7     0  9999     8     0
          5     8     1     1     0     1
          6     8     1     1     0     1
          7     8     1     1     0     1
          8     1     3     3     0     3
```

The optimum allocation shown is man 2 to job 6, man 3 to job 7 and man 4 to job 5 with a total time of 4+6+5=15. An allocation of man 2 to job 5, man 3 to job 7 and man 4 to job 6 with a total time of 1+6+8=15 is also optimum.

**Shortest route problem (section 11) solution using Matlab.**

The labels attached to the different cities are

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

**Input**

```
iNode = [1 1 1     2 2 3 3    4 4 4 5    6 7];
eNode = [2 3 4 4 5 4 6 5 6 7 7 7 1];
cost  = [16 9 35 12 25        15 22 14 17 19 8 14 0];
lo    = [0 0 0 0      0 0 0 0    0 0 0 0    1];
hi    = [1 1 1 1      1 1 1 1    1 1 1 1    1];
nNodes = 7;
```

**Output**

```
netProblems(5)
out = iNode eNode Lower Upper Cost Flow
          1     2     0     1    16     0
          1     3     0     1     9     1
```

| | | | | | |
|---|---|---|---|---|---|
| 1 | 4 | 0 | 1 | 35 | 0 |
| 2 | 4 | 0 | 1 | 12 | 0 |
| 2 | 5 | 0 | 1 | 25 | 0 |
| 3 | 4 | 0 | 1 | 15 | 1 |
| 3 | 6 | 0 | 1 | 22 | 0 |
| 4 | 5 | 0 | 1 | 14 | 0 |
| 4 | 6 | 0 | 1 | 17 | 0 |
| 4 | 7 | 0 | 1 | 19 | 1 |
| 5 | 7 | 0 | 1 | 8 | 0 |
| 6 | 7 | 0 | 1 | 14 | 0 |
| 7 | 1 | 1 | 1 | 0 | 1 |

From the above it follows that the shortest route is 1-3-4-7 i.e. Los Angeles-Phoenix-Denver-St. Louis with a distance of 9+15+19 = 43.



***Peer-review history:***
*The peer review history for this paper can be accessed here (Please copy paste the total link in your browser address bar)*
*http://www.sdiarticle4.com/review-history/55363*