Volume 40(2), pp. 135–146
`http://orion.journals.ac.za`

ORiON
ISSN 0259–191X (print)
ISSN 2224–0004 (online)
©2024

An R program to implement the Out-of-Kilter algorithm

W H Moolman[*]

Received: 12 Nov 2024; Accepted: 3 December 2024

Abstract

Several computer programs that can implement the Out-of-Kilter algorithm have been written. To date programs have been written in Fortran, Algol, Pascal, Basic, C++ and Matlab. According to the knowledge of the author, a program to do this in R has not yet been written. The R program written in this article can solve both the maximum flow and minimum cost-maximum flow problems. Since the Out-of-Kilter algorithm has a network flow as input, it can also be used to solve the flow in any problem whose information can be presented in the form of a network flow. This includes among others the transportation problem, the assignment problem, the shortest route problem and the caterer problem.

Key words: Network flows, Maximum flow, Minimum cost flow, Complementary slackness optimality conditions, In-kilter, Out-of-kilter.

1 Introduction

The Out-of-Kilter algorithm is an algorithm that computes the solution to the minimum cost flow problem in a flow network. It was published by Fulkerson (1961). When determining the minimum cost flow between two points, all out-of-kilter arcs (those not satisfying certain optimality conditions) are identified and modified. It can be shown that once all the arcs are in kilter, a minimum cost flow has been reached.

2 A short computing history of the Out-of-Kilter algorithm

A procedure to execute the Out-of-Kilter algorithm was written by Briggs (1965). A procedure called NETFLOW was suggested by Bray and Witzgall (1968) to determine

[*]Department of Mathematical Sciences, Akademia, South Africa
email: `henrim@akademia.ac.za`
http://dx.doi.org/10.5784/40-2-792

the minimum cost flow in a network using the Out-of-Kilter algorithm. In 1968 they published a correction to the original algorithm. An ALGOL procedure that executes the Out-of-Kilter algorithm was published by Clasen (1968). The Share Distribution Agency (1967) wrote a Fortran program that is designed to save space by arranging arcs so that the source nodes are arranged in order.

A Fortran program called OKAY was published by Kindler (1975). The program uses the Out-of-Kilter algorithm to allocate flows in a network to minimize its total cost of flow. A subroutine called PACKUP constructs a packed list of arcs entering each node and a subroutine called KILT implements the algorithm. Barr et al. (1972) did a comparative study of computer codes for the Out-of-Kilter algorithm.

An Out-of-kilter algorithm was written in Fortran by Woolsey and Swanson (1975) and its use was illustrated with an assignment problem. Smith (1982) wrote Pascal and Basic programs that execute the Out-of-Kilter algorithm. Errors in these programs were pointed out by Shen (1989). Modifications to the program were made to correct these errors. The implementation of a user-friendly transshipment program (written in Pascal) based on the Out-of-Kilter algorithm were described by du Preez and van der Merwe (1988). Validation of the program, time tests and applications were also discussed.

The Out-of-Kilter algorithm was implemented by Nowicki (2014) who wrote a program in Matlab. The data input to the program is by means of a file with the 3 matrices (cost, upper bound, lower bound). These $m \times m$ matrices, denoted by c, u and l 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 l which calculates the optimal flow in the various arcs) and koszt (which scalculates 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.

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

3 The maximum flow and the minimum cost-maximum flow problems

3.1 The maximum flow problem

Consider a network consisting of m nodes (vertices) and n arcs (edges) connecting some of the nodes. A flow network is denoted by $D = (V, E)$, where V is the collection of nodes and E the collection of arcs connecting some of the nodes. An arc$(i,j) \in E$ has a capacity $u_{ij} \geq 0$ (which is the number of units it can carry). Let $x_{i,j}$ denote the flow. The objective is to determine the maximum flow (total number of units) that can be sent from node s

(source) to node t (sink) i.e.,

$$\begin{aligned}
&\text{Maximise } M = \sum_{(i,j)\in E} x_{i,j},\\
&\text{subject to } \sum_{a\in V} x_{i,a} - \sum_{b\in V} x_{b,i} = 0 \text{ (flow conservation constraints)},\\
&l_{i,j} \le x_{i,j} \le u_{i,j},\ \text{arc}(i,j) \in E \text{ (capacity constraints)}.
\end{aligned}$$

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

3.2 The minimum cost-maximum flow problem

Let $c_{i,j}$ be a cost. If the objective is to find the minimum cost C of the maximum flow from node s to node t the problem can be formulated as

$$\begin{aligned}
&\text{Minimise } C = \sum_{(i,j)\in E} c_{i,j} x_{i,j},\\
&\text{subject to } \sum_{a\in V} x_{i,a} - \sum_{b\in V} x_{b,i} = 0 \text{ (flow conservation constraints)},\\
&l_{i,j} \le x_{i,j} \le u_{i,j},\ \text{arc}(i,j) \in E \text{ (capacity constraints)},\\
&\sum_{i\in V} x_{s,i} = \sum_{i\in V} x_{i,t} = M \text{ (maximum flow constraint)}.
\end{aligned}$$

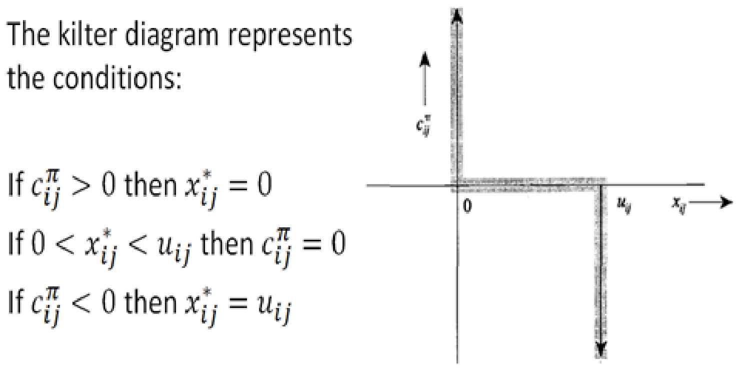
4 Complementary Slackness Optimality Conditions (CSOC) theorem

Consider a network with a set of arcs $A = \text{arc}(i,j) \in E$, where i is the "from" node and j the "to" node. A feasible solution $x^*_{i,j}$ is an optimal solution of the minimum cost flow problem if, and only if, some set of node potentials $\pi = (\pi_1, \pi_2, \ldots, \pi_m)$ with $c^{\pi}_{i,j} = c_{i,j} + \pi_i - \pi_j$ satisfy the following reduced cost optimality conditions for every $\text{arc}(i,j) \in A$:

1. If $c^{\pi}_{i,j} > 0$, then $x^*_{i,j} = l_{i,j}$.
2. If $l_{i,j} \le x^*_{i,j} \le u_{i,j}$, then $c^{\pi}_{i,j} = 0$.
3. If $c^{\pi}_{i,j} < 0$, then $x^*_{i,j} < u_{i,j}$.

5 The Kilter diagram

Below we use x^* as a shorthand for $x^*_{i,j}$. A particular solution $x = x^*$ to the minimum cost flow problem will generate flows $x^*_{i,j}, (i,j) \in A$ along the arcs and node potential values $\pi_i, i = 1, 2, \ldots, m$. For each $arc(i,j)$, $c^{\pi}_{i,j} = c_{i,j} + \pi_i - \pi_j$ can be calculated and the pair


The kilter diagram represents the conditions:

If $c_{ij}^{\pi} > 0$ then $x_{ij}^{*} = 0$

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

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

Figure 1: The Kilter diagram.

of values $(x_{i,j}^{*}, c_{i,j}^{\pi})$ 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 $x_{i,j}, (i,j) \in A$ (horizontal axis value) or (ii) changing the reduced cost values $c_{i,j}^{\pi}, (i,j) \in A$ by altering the node potential values $\pi_i, i = 1, 2, \ldots, m$. Once all the arcs are in-kilter, the solution found is optimal.

6 Example

6.1 Problem description as a flow chart

The flow chart below shows the different routes along which luxury goods can be transported from the place of manufacture (1) to the point of sale (7). The cost per unit (measured in thousands) and capacity (measured in thousands) are shown for each of the arcs in the chart. The problem of interest id to find

1. The maximum number of goods that can be moved from the place of manufacture to the point of sale.

2. The minimum cost needed to move the maximum number of goods from the place of manufacture to the point of sale.

In the flow chart the first number in brackets refers to the cost and the second one to the capacity (quantity of goods that can be carried).

6.2 R program to determine the maximum flow

Before calculating the minimum cost, the maximum flow needs to be known. The starting solution is $x_{i,j} = 0,(i,j) \in A$, $\pi_i = 0, i = 1, 2, \ldots, m$. The R program for calculating the maximum flow for a given flow chart is shown in Appendix A. Below is an explanation of how the program works.

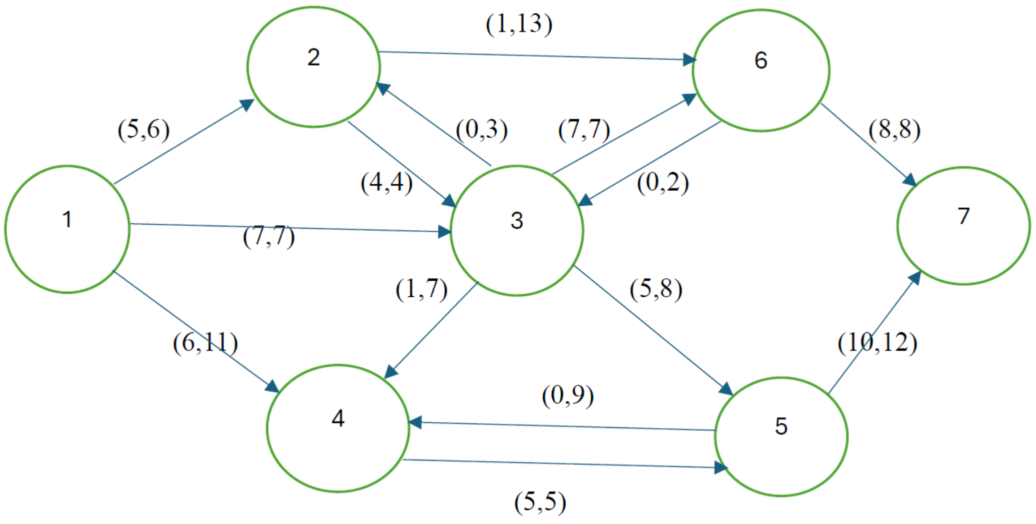


Figure 2: Flow chart of routes from the place of manufacture to the point of sale.

The *inkilter* function takes the flow $(x_{i,j})$, lower capacity $(l_{i,j})$, upper capacity $(u_{i,j})$ and cost $(c_{i,j})$ of each arc as input and, if necessary, adjusts the flow to bring the arc "inkilter". The flow is adjusted according to the "inkilter" specification of the kilter diagram.

The first part of the main program is the definition of the network. This consists of

- The arcs numbered from 1 to 15.
- The starting and end nodes for each of the arcs.
- The costs, upper and lower capacities of each of the arcs.

The remainder of the program is about adjusting the flow $(x_{i,j})$ and $c^{\pi}_{i,j}$ for each arc (i, j) until each arc has an $x_{i,j}$ and $c^{\pi}_{i,j}$ values that satisfy the CSOC. Once this done the $x_{i,j}, (i, j) \in A$ and $\pi_i, i = 1, 2, \ldots, m$ values are printed.

Output

```
arc=  1   from=  1 to=  2  upper bound=  6   cost  0   flow=  6
arc=  2   from=  1 to=  3  upper bound=  7   cost  0   flow=  7
arc=  3   from=  1 to=  4  upper bound=  11  cost  0   flow=  5
arc=  4   from=  2 to=  3  upper bound=  4   cost  0   flow=  1
arc=  5   from=  3 to=  2  upper bound=  3   cost  0   flow=  0
arc=  6   from=  2 to=  6  upper bound=  13  cost  0   flow=  5
arc=  7   from=  3 to=  4  upper bound=  7   cost  0   flow=  0
arc=  8   from=  3 to=  5  upper bound=  8   cost  0   flow=  7
arc=  9   from=  3 to=  6  upper bound=  7   cost  0   flow=  1
arc=  10  from=  4 to=  5  upper bound=  5   cost  0   flow=  5
arc=  11  from=  5 to=  4  upper bound=  9   cost  0   flow=  0
```

```
arc=  12  from=  6 to=  3  upper bound=  2   cost  0   flow=  0
arc=  13  from=  5 to=  7  upper bound=  12  cost  0   flow=  12
arc=  14  from=  6 to=  7  upper bound=  8   cost  0   flow=  6
arc=  15  from=  7 to=  1  upper bound=  24  cost  -1  flow=  18
 node  1  p=  0
 node  2  p=  1
 node  3  p=  1
 node  4  p=  0
 node  5  p=  1
 node  6  p=  1
 node  7  p=  1
```

6.3 Differences between the R program to determine maximum flow and that to determine minimum cost-maximum flow

The R program for solving the maximum flow problem differs from that of the minimum cost - maximum flow problem in the way the network is defined. These differences are discussed in Table 1.

Similarities	**Differences**
(1) The arcs are numbered from 1 to n.	(1) For the maximum flow problem all the costs associated with the arcs are the same. For the minimum cost – maximum flow problem the costs are different.
(2) Except for the arc that connects the sink to the source, all arcs have a lower bound of 0.	(2) For the maximum flow program the lower and upper bounds of the arc that connect the sink to the source are taken as maximum (outflow from source, inflow into the sink). For the minimum cost – maximum flow problem this is taken as the maximum flow obtained from the maximum flow program.
(3) For both programs the lower and upper bounds are equal for the arc that connects the sink to the source.	(3) For the maximum flow problem the unit cost for the sink to the source arc is -1. For the minimum cost - maximum flow problem it is taken as 0.

Table 1: Similarities and differences between the network definitions of the maximum flow and minimum cost - maximum flow problems.

6.4 R program to determine the minimum cost - maximum flow: Output

```
arc=  1   from=  1 to=  2  upper bound=  6   cost  5   flow=  6
arc=  2   from=  1 to=  3  upper bound=  7   cost  7   flow=  7
arc=  3   from=  1 to=  4  upper bound=  11  cost  6   flow=  5
arc=  4   from=  2 to=  3  upper bound=  4   cost  4   flow=  0
```

```
arc=  5   from=  3 to=  2  upper bound=  3   cost  0   flow=  2
arc=  6   from=  2 to=  6  upper bound=  13  cost  1   flow=  8
arc=  7   from=  3 to=  4  upper bound=  7   cost  1   flow=  0
arc=  8   from=  3 to=  5  upper bound=  8   cost  5   flow=  5
arc=  9   from=  3 to=  6  upper bound=  7   cost  7   flow=  0
arc=  10  from=  4 to=  5  upper bound=  5   cost  5   flow=  5
arc=  11  from=  5 to=  4  upper bound=  9   cost  0   flow=  0
arc=  12  from=  6 to=  3  upper bound=  2   cost  0   flow=  0
arc=  13  from=  5 to=  7  upper bound=  12  cost  10  flow=  10
arc=  14  from=  6 to=  7  upper bound=  8   cost  8   flow=  8
arc=  15  from=  7 to=  1  upper bound=  18  cost  0   flow=  18
 node  1  p=  0
 node  2  p=  7
 node  3  p=  7
 node  4  p=  6
 node  5  p=  12
 node  6  p=  8
 node  7  p=  22
```

Appendix A: R code to calculate the maximum flow in a network flow

```
inkilter<-function(flow,low,high,cst){
     inkilter<-FALSE
     changeup<-0
     changedown<-0
     if (cst>0) {
         if (flow<low) changeup<-low-flow
         if (flow==low) inkilter<-TRUE
         if (flow>low) changedown<-flow-low
     }
     if (cst==0)  {
         if (flow<low) changeup<-high-flow
         if ((flow>=low)&(flow<=high)) {
             inkilter<-TRUE
             changeup<-high-flow
             changedown<-flow-low
         }
     }
     if (flow>high) changedown<-flow-low
     if (cst<0) {
         if (flow<high)  changeup<-high-flow
         if (flow==high) inkilter<-TRUE
         if (flow>high)  changedown<- flow-high
     }
     return(list(inkilter,changeup,changedown))
 }

# Define network
arclist <- data.frame(
      arc = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15),
      i= c(1,1,1,2,3,2,3,3,3,4,5,6,5,6,7),
      j = c(2,3,4,3,2,6,4,5,6,5,4,3,7,7,1),
      l = c(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
      h= c(6,7,11,4,3,13,7,8,7,5,9,2,12,8,24),
      c = c(0,0,0,0,0,0,0,0,0,0,0,0,0,0,-1)
  )
  attach(arclist)
  arcs<-15
  nodes<-7
  inf<-9999
  # Prepare a, b and x arrays
  a<-rep(0,arcs)
  b<-rep(0,arcs)
  x<-rep(0,arcs)
  p<-rep(0,nodes)
# Initialize flow and pi values
  infeasible<-FALSE
# Check for Out of Kilter arcs
  for (arc in 1:arcs) while(!inkilter(x[arc],l[arc],h[arc],c[arc]
  +p[i[arc]]-p[j[arc]])[[1]]&(!infeasible)){
      plus<-inkilter(x[arc],l[arc],h[arc],c[arc]+p[i[arc]]-p[j[arc]])[[2]]
      minus<-inkilter(x[arc],l[arc],h[arc],c[arc]+p[i[arc]]-p[j[arc]])[[3]]
      repeat {

          if (plus>0) {
              s<-j[arc]
              t<-i[arc]
              a[s]<-t
              b[s]<-plus} else {
                  t<-j[arc]
                  s<-i[arc]
                  a[s]<- -t
                  b[s]<- minus
              }
          for (node in 1:nodes) if (node!=s) a[node]<-0
          repeat {
              newlabels<-0
              for (arc1 in 1:arcs) if (((a[i[arc1]]==0)&(a[j[arc1]]!=0))|
              ((a[i[arc1]]!=0)&(a[j[arc1]]==0))){
                  ok<-inkilter(x[arc1],l[arc1],h[arc1],
          c[arc1]+p[i[arc1]]-p[j[arc1]])[[1]]
         plus<-inkilter(x[arc1],l[arc1],h[arc1],
         c[arc1]+p[i[arc1]]-p[j[arc1]])[[2]]
         minus<-inkilter(x[arc1],l[arc1],h[arc1],
         c[arc1]+p[i[arc1]]-p[j[arc1]])[[3]]
          if ((a[i[arc1]]!=0)&(plus>0)){
              newlabels<-newlabels+1
              a[j[arc1]]<-arc1
              b[j[arc1]]<-min(b[i[arc1]],plus)
          } else { if ((a[j[arc1]]!=0)&(minus>0)){
              newlabels<-newlabels+1
              a[i[arc1]]<- -arc1
              b[i[arc1]]<- min(b[j[arc1]],minus)
          }
          }
      }
      if((newlabels==0)|(a[t]!=0)) break
  }
  if (a[t]!=0){
      node<-t
      change<-b[t]
      if (a[s]>0){
          x[arc]<-x[arc]+change
      } else {
          x[arc]<-x[arc]-change
      }
      repeat{
          arc1<-abs(a[node])
          if (a[node]>0){
              node<- i[arc1]
              pm <-1
          } else {
              node<-j[arc1]
              pm<- -1
          }
          x[arc1]<- x[arc1]+pm*change
          if(node==s) break
      }
  }  else {
      del<-inf
      infeasible<-TRUE
      for (arc1 in 1:arcs)
          if (((a[i[arc1]]==0)&(a[j[arc1]]!=0))|
          ((a[i[arc1]]!=0)&(a[j[arc1]]==0))){
              accept<-((x[arc1]<h[arc1])&(x[arc1]>l[arc1]))
              if ((c[arc1]+p[i[arc1]]-p[j[arc1]]>0)&((accept)|
                    (x[arc1]==l[arc1]))){
                        del<-min(del,c[arc1]+p[i[arc1]]-p[j[arc1]])
                        infeasible<-FALSE
                    }
                    if ((c[arc1]+p[i[arc1]]-p[j[arc1]]<0)&
                    ((accept)|(x[arc1]==h[arc1]))){
                        del<-min(del,-(c[arc1]+p[i[arc1]]-p[j[arc1]]))
                        infeasible<-FALSE
                    }
                }
            if ((c[arc1]+p[i[arc1]]-p[j[arc1]]<0)&
            ((accept)|(x[arc1]==h[arc1]))){
                del<-min(del,-(c[arc1]+p[i[arc1]]-p[j[arc1]]))
                infeasible<-FALSE
            }
            if ((c[arc1]+p[i[arc1]]-p[j[arc1]]<0)&
            ((accept)|(x[arc1]==h[arc1]))){
                del<-min(del,-(c[arc1]+p[i[arc1]]-p[j[arc1]]))
                infeasible<-FALSE
            }
            if (del==inf){
                del<-abs(c[arc]+p[i[arc]]-p[i[arc]])
                if ((x[arc]>=l[arc])&(x[arc]<=h[arc]))infeasible<-FALSE
            }
            if (infeasible){
                print("There is no feasible flow")
            }  else {
                for (node in 1:nodes) if (a[node]==0) p[node]<-p[node]+del
            }
        }
        if(infeasible | inkilter(x[arc],l[arc],h[arc],
        c[arc]+p[i[arc]]-p[j[arc]])[[1]]) break

    }
}
if(!infeasible){
    for (arc in 1:arcs) cat("arc= ",arc," from= ",i[arc],"to= ",j[arc],
    " upper bound= ",h[arc]," cost ",c[arc]," flow= ",x[arc],"\n")
    for (node in 1:nodes) cat(" node ",node," p= ",p[node],"\n")
}
```

Appendix B: Changes to the R code for maximum flow to calculate the minimum cost-maximum flow

The network definition section for the minimum cost-maximum flow program is changed to the following. All the other command remain the same.

```
  arclist <- data.frame(
    arc = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15),
    i= c(1,1,1,2,3,2,3,3,3,4,5,6,5,6,7),
    j = c(2,3,4,3,2,6,4,5,6,5,4,3,7,7,1),
    l = c(0,0,0,0,0,0,0,0,0,0,0,0,0,0,18),
    h= c(6,7,11,4,3,13,7,8,7,5,9,2,12,8,18),
    c = c(5,7,6,4,0,1,1,5,7,5,0,0,10,8,0)
)
attach(arclist)
arcs<-15
nodes<-7
```

References

Barr, R., Glover, F., and Klingman, G. (1972). An improved version of the out-of-kilter method and a comparative study of computer codes. *Research Report C.S., University of Austin, Texas.*

Bray, T. A. and Witzgall, C. (1968). Algorithm 336. *Netflow, Comm. ACM.*, (11):631–632.

Briggs, W. A. (1965). Algorithm 248. *Comm. ACM.*, (8):103–104.

Clasen, R. J. (1968). The numerical solution of network problems using the out-of-Kilter algorithm. *United States Air Force Project RAND, Memorandum RM-5456-PR.*

du Preez, N. D. and van der Merwe, J. C. (1988). Die ontwikkeling van 'n gebruikersvriendelike transskeppingpakket vir ibm-aanpasbare persoonlike rekenaars. *SA Journal of Industrial Engineering*, (2):25–39.

Fulkerson, D. R. (1961). An out-of-kilter method for minimal cost flow. *Journal of the Society for Industrial and Applied Mathematics*, (9):18–27.

Kindler, J. (1975). 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.*

Nowicki, M. (2014). Adding out-of-kilter algorithm. available: https://github.com/michalnowicki/numericalalgebracodes/blob/master/ookmatlab/ook.m.

Share Distribution Agency (1967). Out-of-kilter network routine. *Share Distribution 3536, Share Distribution Agency, Hawthorne, New York.*

Shen, Z. (1989). *Log truck scheduling by network programming.* PhD thesis, Department of Forest Engineering for the Degree of Master of Forestry, Univ. of Washington, Seattle, WA.

Smith, D. K. (1982). *Network Optimization Practice: A Computational Guide.* Chichester, Ellis Horwood.

Woolsey, R. E. D. and Swanson, H. S. (1975). *Operations Research for Immediate Application: A Quick and Dirty Manual.* Harper & Row, New York.