# MINET A FAST NETWORK LP SOLVER

*I. Maros*

June 1987
WP-87-50

# MINET A FAST NETWORK LP SOLVER

*I. Maros*

June 1987
WP-87-50

## FOREWORD

In comparison with already existing software for the solution of network type linear programming problems, MINET gives a possibility of very flexible pricing that can be further fitted to the special structure of the network and using a suitable interface, it can reflect the need for changing the network structure.

Alexander B. Kurzhanski
Chairman
System and Decision Sciences Program

## CONTENTS

# MINET A FAST NETWORK LP SOLVER

*I. Maros*

Computer and Automation Institute of
the Hungarian Academy of Sciences, Budapest

## 1. INTRODUCTION

Within the frame of IIASA project "Modeling of Interconnected Power Systems" it was necessary to create an advanced implementation of a network linear programming (LP) algorithm. The program is supposed to work in a hierarchical system of programs under conditions which are different from the routine practice of network LP applications. The main purpose of this new development is to have an efficient network LP solver which can easily be included in other systems and which is under full control of the designers and implementers of the power system model. As a by-product, the system can be made available for stand-alone usage. The program is based on the unpublished implementation PNS (Pure Network System) of Maros (Eindhoven University of Technology, Eindhoven, The Netherlands, 1983). For further reference the new implementation will be called MINET.

The special structure of the network LP problems has challenged many researchers to exploit this feature in favor of efficiency of the solution process and capacity of programs. In this context efficiency is measured in solution time while capacity is understood as the maximum size of problems that can be solved by a given program.

The literature is rich in the theoretical foundation of the specialized network simplex method and some important algorithmic details are also published. Less attention was paid to certain implementational details which, however, also considerably influence the efficiency of a program.

The purpose of this paper is (a) to give account of the theoretical background, implementational tools, capabilities, and special features of MINET, (b) to serve as a user's guide. It also reports some limited computational experiences.

## 2. PROBLEM STATEMENT

The minimal cost network flow (MCNF) problem or capacitated transshipment problem can be stated as follows:

$$\text{minimize } z = \sum_{(i,j) \in E} c(i,j)^* x(i,j) \tag{1}$$

$$\text{subject to } \sum_{i \in I(k)} x(i,k) - \sum_{j \in O(k)} x(k,j) = b(k) \quad \text{for } k \in M \tag{2}$$

$$0 \le x(i,j) \le u(i,j) \quad \text{for } (i,j) \in E \ , \tag{3}$$

where

$$I(k) = \{i \in M : (i,k) \in E\} \ ,$$

$$O(k) = \{j \in M : (k,j) \in E\} \ ,$$

$E$ is a set of arcs $(i, j)$ of the network $G(M, E)$,

$M$ is the set of nodes.

The cardinality of $M$ is denoted by $m$, and that of $E$ is denoted by $n$. The constant $b(k)$ represents the requirement at node $k$. A node $k \in M$, for which

$b(k) < 0$ is called a supply node,

$b(k) > 0$ is called a demand node,

$b(k) = 0$ is called a pure transshipment node.

It is assumed that

$$V = \sum_{k \in M} b(k) = 0 \ . \tag{4}$$

If $V < 0$ then there is no feasible solution. If $V > 0$ then the problem can be converted to the prescribed form by adding a dummy destination with a requirement of $-V$ and slack arcs from each source to the dummy destination. The slack arcs are given unit costs of zero and infinite arc capacities.

It should be noted that (3) is not a restriction of generality because individual lower bounds of the variables different from zero can always be moved to zero by a simple transformation.

Associated with each node $k \in M$ is a dual variable $\pi(k)$ called its node potential or simplex multiplier. An arc $(i, j)$ is directed from node $i$ to node $j$. An arc $(i, j)$ is said to be out-directed from node $i$ and in-directed in node $j$. In this sense $I(k)$ is the set of tail nodes of arcs that are in-directed to node $k$, and $O(k)$ is the set of head nodes of arcs that are out-directed from node $k$.

The flow, cost, and upper bound of arc $(i, j)$ are represented, respectively by $x(i, j)$, $c(i, j)$, and $u(i, j)$. The objective is to determine a set of arc flows which satisfies the node requirements and capacity restrictions at a minimum total cost.

REMARK In the above description we followed the terminology of [3].

Let us denote the matrix of constraints on the left hand side of (2) by $A$. It is easy to see that each column of $A$ is associated with an arc $(i, j)$ and contains one coefficient with value of -1 and one with +1 corresponding to the starting node and the ending node, respectively. In addition to this usual case we allow in MINET such arcs which correspond to loops (self-loops). In this case the "from" and the "to" nodes coincide. The matrix column of such an arc contains one -1 coefficient.

Using a more concise notation we can write problem (1)-(3) in the following form (with evident interpretation of the correspondence):

$$\text{minimize } z = c'x \tag{5}$$

$$\text{subject to } Ax = b \tag{6}$$

$$0 \leq x \leq u \ , \tag{7}$$

where $A$ is an m by $n$ matrix, $c$, $x$, and $u$ are $n$-vectors, $b$ is an $m$-vector and $'$ (prime) denotes the transpose.

The dual of this problem is the following:

$$\text{maximize } Z = b'w - u'v \tag{8}$$

$$\text{subject to } A'w - v \leq c \tag{9}$$

$$w \quad \text{unrestricted} \ , \tag{10}$$

$$v \geq 0 \ ,$$

where $w$ is an $m$-vector and $v$ is an $n$-vector of dual variables.

It is a rather characteristic feature of the network LP problems that usually there are much more arcs than nodes, i.e. the number of variables is much larger than the number of node constraints, $m \ll n$.

## 3. SOLUTION OF NETWORK LP PROBLEMS

For solving problem formulated in (5)–(7) primal (e.g.[3]), dual (e.g.[9]), and primal-dual (e.g.[2]) algorithms have been proposed. As a result of recent developments the primal type algorithms show clear superiority over the others. The dramatic improvement of the performance of network LP solvers is largely due to the successful application of new achievements in computer implementation technology for optimization algorithms.

Following [7] we can say that computer implementation technology seeks to discover efficient procedures for carrying out subalgorithms of a general method on a computer by investigating (a) what kind of information to generate and maintain for executing operations most effectively, (b) which data structures are best to record, access, and update this information, and (c) what methods are most suitable for processing these data to make the the desired information available when it is needed. Such knowledge can be the result of long experience in the theory and practice of optimization and the proper combination of the best elements of mathematics and computer science.

In the case of the solution of network LP problems the underlying algorithm is the revised primal simplex method where the very special structure of the problem is highly exploited. This is done in two aspects:

- mathematical considerations,
- implementational considerations.

A bounded variable simplex basis for a network flow problem corresponds to a spanning tree with $m - 1$ arcs. Knowing this tree and the nonbasic variables at bound the actual basic solution can easily be calculated. This practically corresponds to the case when the basis of a general LP problem is triangular (or combinatorially triangularizable). Thus – in contrast with the standard revised simplex method – we do not need the basis inverse (or equivalents of it) in any form. This is one of the main points in the specialization. Any information required for the steps of the simplex method can directly be provided by this triangular basis or rather by proper operations on the basis tree. The required arithmetic operations are addition, subtraction, and multiplication. It means that if our starting values are integers then we can simply use integer arithmetic all the time. This (a) considerably improves the efficiency of the algorithm, and (b) completely excludes computational errors which is a frequent problem with general purpose linear programming algorithms.

## 4. IMPLEMENTATIONAL CONSIDERATIONS

The reports on efficient network simplex algorithms usually describe the theoretical achievements but do not indicate explicitly how the operations are to be organized to minimize the computational and updating effort per iteration. The only exception in this respect seems to be [1] which presents a number of algorithms in detail but unfortunately they are not free of errors and therefore cannot directly be implemented. A very recent paper [12] (which was not available when either PNS in 1982–83, or MINET in 1985 were designed) gives an elaborate algorithm but due to the shortage of time it could not be checked. In any case the algorithm shows a careful design and is surely a valuable contribution to the network LP literature.

It should be noted here that the minimization of the computational effort per iteration usually does not result in the minimization of the total computational effort to solve a problem. This latter is heavily influenced by the pricing strategy applied while this is a rather problem dependent factor. To achieve good overall performance with a network LP program properly refined pricing strategy must be implemented.

The basic idea in the efficient specialization of the simplex method is the special representation, use, and updating of the basis. By this we can achieve the efficient performance of the basic operations of the revised simplex method (BTRAN, PRICING, and FTRAN). The developed techniques use the rooted tree representation of the basis. MINET adds one node to M and this node is considered the root of the basis tree. The root node is regarded as being on the top in the rooted tree with all other nodes hanging below it. If nodes $i$ and $j$ denote endpoints of an arc in the rooted tree such that node $i$ is closer to the root, then $i$ is called the predecessor of node $j$ and node $j$ is called the immediate successor of node $i$.

To facilitate the description of the list functions used in the network simplex algorithms we will use some notations:

| | | |
|---|---|---|
| $T$ | = | the basis tree. |
| $T(x)$ | = | subtree of $T$ headed by node $x$ (i.e. the subtree that includes $x$ and all its successors in the predecessor ordering). |

For handling the basis the following list functions have been defined:

| | | |
|---|---|---|
| $p(x)$ | = | the predecessor of node $x$. If $x$ is the root node then $p(x) = 0$. |
| $s(x)$ | = | "thread successor" of $x$. |

Function $s(x)$ may be thought as a thread which passes through each node exactly once in a top to bottom, left to right order, starting from the root node. The thread-successor of the last node is the root node. If we denote the root node by 1, then the set $\{s(1), s2(1), \ldots, sn(1) = 1\}$ is exactly the set of nodes of the rooted tree, where $s2(1) = s(s(1))$, $s3(1) = s(s2(1))$, etc.

| | | |
|---|---|---|
| $r(x)$ | = | reverse thread of $x$. |
| $t(x)$ | = | number of nodes in $T(x)$. |
| $f(x)$ | = | last node in $T(x)$ in thread ordering. |
| $h(x)$ | = | distance of node $x$ from the root node (i.e. the number of arcs to be passed from $x$ to the root in the predecessor order). $h(1)$ is defined to be 0. |
| $g(x)$ | = | preorder distance of node $x$. This denotes the sequential position of node $x$ in the thread ordering. By this definition $g(1)$=1. |
| $id(x)$ | = | directed arc identifier. Suppose arc $E(ib(x))$ connects nodes $x$ and $p(x)$ [$ib(i)$ denotes the column number of the $i$-th basic arc in the matrix]. $id(x)$ is defined as follows: |

$$id(x) = ib(x) \text{ if } E(ib(x)) = (p(x), x) \quad [p(x) \rightarrow x],$$

$$id(x) = -ib(x) \text{ if } E(ib(x)) = (x, p(x)) \quad [x \rightarrow p(x)].$$

For representing $T$ the minimum requirement is to keep the predecessor, thread, and directed arc identifier functions. Note that each of the above defined functions requires an $m + 1$ length integer array. In general it is true that if we use more functions and at the same time occupy more memory space for the arrays the algorithm will be faster (even if logically more complicated). This leads us to the well known space-time conflict. (It also should be noted that not all the above functions can be used simultaneously because some of them are replacements for the others.) Setting up the design criteria we can decide on how far to go in using more functions and complicated algorithm to gain speed. This decision influences the maximum problem size as well.

Glover, Klingman and Stutz in [11] use the predecessor, thread, reverse thread, and directed arc identifier functions.

Srinivasan and Thompson in [14] agumented the data structure with the distance function which resulted in some simplifications and speed improvement of the algorithm at the expense of updating the distance function in each iteration.

Ali et al. reported in [1] that Glover and Klingman had replaced the distance function by the number of nodes in subtree function. The resulting procedure retained the benefits of the previous algorithm while giving a cheap updating of the newly used function.

Bradley, Brown, and Graves in [4] replaced the distance or the number of nodes in subtree functions by the preorder distance function. In this way tracing the cycle (to find the leaving arc of the basis tree) is simpler but the updating is more difficult.

Ali et al. reported in [1] the following observation. If a basic arc is removed from $T$, then $T$ is partitioned into two trees $T1$ and $T2$, where the root is contained in $T1$. The dual variables of either $T1$ or $T2$ must be updated. If we augment the data structure with the number of nodes in subtree function then the smaller subtree for updating can easily be selected. We transform $T1$ and $T2$ into independent trees, update the dual variables in the tree with fewer nodes, then reconnect $T1$ and $T2$ and complete the updating.

## 5. IMPLEMENTATIONAL TOOLS OF MINET

As noted earlier the available literature does not provide a full description of an implementation and many of the published sub-algorithms contain errors. When developing the predecessor of MINET (PNS, 1983) we simply wanted to make an efficiently operating correct network implementation for problem (1)–(3). The testing of PNS has given promising results and therefore it is considered a good starting basis for developing MINET.

Algorithmically MINET is a specialized primal revised simplex method for solving MCNF problems. For the representation of the rooted basis tree $T$ it uses the following functions (just as does it PNS):

- directed arc identifier,
- predecessor,
- thread,
- reverse thread,
- number of nodes in subtree,
- last node in subtree.

Matrix $A$ of (6) is represented in MINET in the following way: The indices of the nonzero entries are stored in a column-wise order in the linear list $ind(\cdot)$. To identify the beginning of columns in this list an array of column pointers $cp(\cdot)$ is used. In each column the index of the 'from' node is supposed to be listed first and it is followed by the index of the 'to' node if it exists. The upper bounds and the cost coefficients of the arcs are stored in arrays $u(\cdot)$ and $cst(\cdot)$, respectively. The status of the arc variables can be found in array $mark(\cdot)$. Basically this can assume three different values corresponding to basic, nonbasic at lower bound, and nonbasic at upper bound situations.

The flexibility of this scheme is evident and will be recalled later.

As a starting basis MINET creates an all-slack basis, where all the nodes are connected to the artificial root [Master Root]. The capacity of these arcs is unbounded. A directed arc identifier is negative if the corresponding node requirement is negative (supply node), else it is positive. It is easy to see that for this trivial basis the corresponding administration is the following:

| $i$ | $p(i)$ | $s(i)$ | $t(i)$ | $f(i)$ | $id(i)$ |
|---|---|---|---|---|---|
| 1 | $m+1$ | 2 | 1 | 1 | $(n+1)$ |
| 2 | $m+1$ | 3 | 1 | 2 | $(n+2)$ |
| 3 | $m+1$ | 4 | 1 | 3 | $(n+3)$ |
| ... | ... | ... | ... | ... | ... |
| $m$ | $m+1$ | $m+1$ | 1 | $m$ | $(n+m)$ |
| $m+1$ | 0 | 1 | $m+1$ | $m$ | – |

The index of the Master Root is $m + 1$.

The starting basic solution is usually infeasible. In phase-1 the program attempts to reduce the flows on all artificial arcs to zero. For this purpose a special algorithm is used taking the advantage of ideas described in [13]. In the present interpretation it means the setting up of the vector of simplex multipliers stored in pi(.) with the following values:

$$\pi(i) = 0, \quad \text{if } b(i) = 0$$

$$\pi(i) = 1, \quad \text{if } b(i) > 0$$

$$\pi(i) = -1, \quad \text{if } b(i) < 0 \ .$$

Note that (4) must hold for the problem, otherwise feasible solution cannot be found.

An iteration starts with the pricing operation. This serves to compute the $d(j)$ reduced costs for the nonbasic variables and to select one candidate for entering the basis. If such a variable cannot be found then either (a) an optimal solution is reached if we are in phase-2, or (b) phase-1 is terminated in which case the situation is still to be evaluated: if all the artificial flows are driven to zero then phase-2 is initiated, otherwise the problem has no feasible solution.

Variables corresponding to artificial arcs are never priced because they are type-0 variables (see [13]).

The pricing operation of a column consists of the following computation if $i$ points to the first nonzero entry of column $j$:

$$d(j) = -\pi(ind(i)) + \pi(ind(i+1)) - cst(j), \quad \text{if the arc is ordinary} \ , \tag{11}$$

$$d(j) = -\pi(ind(i)) + \pi(m+1) - cst(j), \quad \text{if the arc is a self-loop} \ . \tag{12}$$

In phase-1 the $cst(\cdot)$ part is not present. If nonbasic variable $j$ is at upper bound then we change the sign of $d(j)$ for evaluation.

It is easy to see from (11) and (12) that pricing a column is a 'cheap' operation in the sense of the number of memory accesses and arithmetic operations.

If there exists at least one nonbasic column with $d(j) > 0$, then the optimality condition is not satisfied and variable $j$ is introduced to the basis.

The simplicity of the pricing operation can be misleading. Since in practical cases the number of arcs ($n$) is much larger than the number of nodes ($m$), there are many nonbasic variables that must be scanned in the course of iterations before optimality can be declared. Many of the pricing strategies work in such a way that a great number (or all) of the nonbasic variables are priced in one iteration (c.f. steepest ascent). This can result in a tremendous amount of computations to find an entering arc. This is true even for the simplest selection rule 'first positive'. There are statistics showing that 50–90 percent of the computational effort to solve a network LP problem is spent on pricing. With special problems these figures can even be worse. Though these facts have been recognized by some researchers, a general purpose 'optimal' pricing strategy has not been found yet. Observations show that the efficiency of the pricing strategy is generally problem dependent (at least with the known pricing strategies). Since we have no information on the structure and 'behavior' of the problems to be solved in project "Modeling of Interconnected Power Systems" we implemented some known pricing strategies and designed a new one which has some free parameters. These parameters enable the user of the system to tune the program to a given class of problems to solve the members of the class in an efficient way.

The pricing strategy to be used can be defined in the starting phase of MINET. When the standard pricing [ST] is selected we have the chance to further specify the way it should work. In this case column selection strategy is controlled by the user defined variable '$np$' as follows:

| | |
|---|---|
| $np = 0$ | first positive $d(j)$ in phase-1, largest $d(j)$ in phase-2, |
| $np = 1$ | largest $d(j)$ in both phases, |
| $np = 2$ | first positive $d(j)$ in both phases, |
| $np = 3$ | first positive $d(j)$ in phase-1, but scanning starts at the last column and goes backwards, largest $d(j)$ in phase-2. |

In MINET standard pricing is carried out by SUBROUTINE PRICE 1.

When the new pricing strategy (called sectional pricing and referred as SC) is selected then some further parameters have to be provided by the user. Before describing the meaning of the parameters we briefly outline the basic idea of SC.

In a general pricing step SC scans only a subset of the nonbasic variables. For this purpose matrix $A$ of (6) is partitioned into adjacent sections of the same size (only the last section can somewhat be smaller). The size is defined as percent NPERC of $n$ (the number of arcs) which, at the same time, defines the number of sections (NSC). There is a parameter KVEC that gives the number of improving vectors (the ones with positive $d(j)$) to be found in one section if there are any.

Pricing of a sections stops when this number of vectors have been found and the column where it happened is recorded. Next scanning will start at this point. There is an other parameter KSEC that controls the number of sections to be scanned in one pricing operation. In MINET KSEC is temporarily set to the number of sections (NSC), that is all sections are scanned with the above technique. The final decision on the entering column is based on the magnitude of the corresponding $d(j)$ and the largest one is selected from among the scanned candidates. Special care must be taken when the prescribed number of improving vectors cannot be found (typically in the neighborhood of an optimal solution), especially to declare optimality a 'full pricing' is necessary. The logical structure of SC (which can easily be reconstructed from the source code of SUBROUTINE SPRICE) automatically covers all these cases.

It is easy to see that SC is a rather flexible pricing scheme (which still can be extended to multiple pricing) and many of the known pricing strategies are contained in it as special cases that can be reproduced by special definition of parameters NPERC, KVEC, and KSEC. The power of SC has already been proved in a general LP implementation (MAPS) of the author and in the limited number of experiences that we had with MINET (where the improvement was dramatic).

Using the above definitions now we can easily describe the parameters to be provided for MINET when SC is selected. These parameters are simply NPERC and KVEC. (As mentioned earlier KSEC is not a user accessible parameter yet.)

After pricing (which resulted in finding a candidate arc variable) the next question is the determination of the outgoing variable. This is done in the pivot operation. To carry out this step in general LP we need the updated form of the incoming column. This can be obtained by the expensive FTRAN operation. In network LP FTRAN can completely be omitted and the pivot step (ratio test) can be carried out by tracing the loop (created by the incoming arc) on the basic spanning tree.

In MINET ratio test is carried out by SUBROUTINE RTEST1.

The subsequent updating operation requires the modification of the administration of the basic spanning tree, the simplex multipliers, and the basic solution as well. This is a rather complicated procedure and is partly based on some ideas of [1] and [3].

The updating steps of MINET can be found in the lengthy SUBROUTINE UD1.

## 6. USE OF MINET

The present version of MINET can be used in a stand-alone mode on the VAX computer of IIASA. The program is written in FORTRAN77 language of the VAX. The source code of the program is portable to other FORTRAN77 systems with two possible exceptions:

(a) The VAX FORTRAN uses the Dollar sign ($) in a FORMAT statement where the standard FORTRAN77 uses backslash (\).

(b) The INCLUDE statement of VAX FORTRAN is non-standard.

The subroutines of MINET are commented so that the reader can have a first idea of the function and operation of them. The source code of MINET can be found in files 'minet1.f' containing the program, and 'minetcom.for' containing the common variables. The latter file also provides information on the meaning of variables and the way how the maximum size of the solvable problems can be changed in the declarations.

MINET is a fully in-core program, i.e. after loading and reading in the problem it does not use the background store (disk). The compact data storage scheme makes it possible that very large problems can be solved by this in-core version.

The problem to be solved must be provided in a text file with the following structure. (The structure is very simple and can easily be changed if necessary.)

| | |
|---|---|
| 1 | rec. M,N, format: 2I5 (i.e. two fields, 5 characters wide each, right justified) |
| 2 | rec. 'from' node, 'to' node, upper-bound, cost coefficient, format: 2I5,2I10 |
| | • |
| | • |
| $N+2$ | rec. right-hand-side elements, five in one record, format: 5I10. |
| | • |
| | • |

To invoke MINET we simply type the name of the file where the executable program resides (presently it is 'minet1.out'). The start procedure of MINET consists of a dialogue part. The output of MINET appears in capital letters if a user response is required, otherwise in small-case letters.

For program maintenance purposes a single variable is used to control the extent of intermediate output. First this trace variable is questioned. The recommended value for normal use is 0 (zero). After this the name of the input file, the pricing strategy, and the parameters of the selected pricing strategy must be given. At trace level 0 only two lines of intermediate results appear. One after completion of phase-1, and one after an optimal solution has been found (if such exists). In both cases the number of iterations and the value of the true objective function are displayed.

A detailed tabular output of the solution is not automatically produced. The user is asked if he/she wants to have it. In the case of yes, the name of the output file must be given.

The structure of the tabular output is traditional. It contains information on the column and row variables as well.

The shadow prices of the row variables are actually the values of the dual variables $w(i)$, while the shadow prices of the column variable are the values of dual variables $v(j)$, or the negative of them if a variable is at bound.

## 7. CONCLUDING REMARKS

After checking the correct operation of MINET we generated some larger problems to test efficiency. For this purpose we used the trailer subproblems of R.R. Love, jr. 'Traffic Scheduling via Benders Decomposition', Mathematical Programming Study 15. The largest of these problems contained 124 nodes and 3894 arcs. This specific type of problems was very sensitive to the pricing strategy applied and interestingly enough produced the best run statistics with NPERC=100 and KVEC=1 which corresponds to a special partial pricing strategy. The solution time on the VAX 11/780 computer of IIASA was less than 4 seconds (residence time) under low workload conditions of the machine.

It should be noted that the above procedure does not mean a heavy testing of MINET, and this job must be done in the next phase of the project "Modeling of Interconnected Power Systems". This will also be a good occasion to tune the algorithm to the specific needs of the modeled power networks. One of these requirements is already known and this is the ability of modifying the structure of the network. Clearly, this can be done in a rather easy way. If an arc is to be deleted then the marker *mark*$(\cdot)$ of it can simply be set to 0. If an arc is to be added then it can be put after the last arc using the data scheme described above. Adding a node requires a little bit more work with the present assumption that the Master Root (MR) is indexed $m + 1$. Now the new node(s) can simply be added to the existing ones and the new value for m must be established which also defines the new index of MR. Now the new arc(s) connecting the new node(s) can be added in the above described way. This administration could be made easier if we used 0 for indexing MR. While on the VAX FORTRAN it is possible, the FORTRAN77 system which we used for program development it was impossible. This point is also considered as a possible further improvement on 'user friendliness' and flexibility of the system.

## REFERENCES

[1] Ali, A.I., R.V.Helgason, J.L.Kennington and H.S.Lall: "Primal Simplex Network Codes: State-of-the-Art Implementation Technology", Networks 8(1978), pp. 315–339.

[2] Barr, R., F.Glover and D.Klingman: "An improved version of the Out-of-Kilter Method and a comparative study of computer codes", Mathematical Programming, vol.7. No. 1.(1974), pp. 60-87.

[3] Barr, R., F.Glover and D.Klingman: "Enhancements of Spanning Tree Labeling Procedures for Network Optimization", Infor 17(1979) pp. 16–34.

[4] Bradley, G.H., G.G.Brown and G.W.Graves: "Design and Implementation of Large Scale Primal Transshipment Algorithms", Management Science, 24, 1, 1977, pp. 1–34.

[5] Glover, F., D.Karney and D.Klingman: "Implementation and Computational Study on Start Procedures and Basis Change Criteria for a Primal Network Code", Networks 4(3) (1974) 191–212.

[6] Glover, F., D.Karney, D.Klingman and A.Napier: "A Computational Study on Start Procedures, Basis Change Criteria, and Solution Algorithms for Transportation Problems", Management Science 20(1974), pp. 793–813.

[7] Glover, F. and D.Klingman: "Recent Developments in Computer Implementation Technology for Network Flow Algorithms", International Workshop on Advances in Linear Optimization Algorithms and Software, Pisa, Italy, July 28–31, 1980.

[8] Glover, F. and D.Klingman: "The Simplex SON Algorithm for LP/Embedded Network Problems", Mathematical Programming Study 15(1981) pp. 148–176.

[9] Glover, F., D.Karney and D.Klingman: "Implementation and Computational Comparison of Primal, Dual and Primal-Dual Computer Codes for Minimum Cost Network Flow Problems", Networks, 4, 3(1974), pp. 191–212.

[10] Glover, F., D.Karney and D.Klingman: "Double-Pricing, Dual and Feasible Start Algorithms for the Capacitated Transportation (Distribution) Problem", ccs Research Report 105, Center for Cybernetic Studies, University of Texas, Austin, Texas (undated).

[11] Glover, F., D.Klingman and J.Stutz: "Augmented Threaded Index Method for Network Optimization", INFOR, 12, 3, 1974, pp. 293–298.

[12] Grigoriadis, M.D.: "An Efficient Implementation of the Network Simplex Method", Mathematical Programming Study 26(1986), pp. 83–111.

[13] Maros, I.: "A General Phase-I Method in Linear Programming", European Journal of Operational Research, 23(1986), pp.64–77.

[14] Srinivasan, V. and G.L.Thompson: "Accelerated Algorithms for Labeling and Relabeling of Trees with Applications to Distribution Problems", Journal of the Association of Computing Machinery, 19, 4, 1972, pp. 712–726.