CODEN: LUTFD2/(TFRT-5412)/1–143/(1989)

Service Restoration In Electric Power Distribution Networks

Magnus Bergstrand

Department of Automatic Control
Lund Institute of Technology
October 1989

SERVICE RESTORATION IN ELECTRIC DISTRIBUTION NETWORKS

by
Magnus Bergstrand
Lund Institute of Technology
Lund Sweden
1989-10-01

ABSTRACT

Automatic service restoration in distributional networks has been studied. After a thorough examination of the literature in this field, one algoritm was selected and implemented as a Fortran program. This implementation was then proved to be effective. Further improvements to the algoritm was introduced and in three cases these proved to be succesful. For example: An operational aid indicating roughly the order of the switching operations was added, the number of switching operations decreased in some cases and the the priority of the zones was made more important.

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Document name
Master Thesis

Date of issue
October 1989

Document Number
CODEN: LUTFD2/(TFRT-5412)/1-143/(1989)

Author(s)
Magnus Bergstrand

Supervisor
Rolf Johansson, LTH, Börje Nilsson, ABB

Sponsoring organisation

Title and subtitle
Service Restoration in Electric Power Distribution Networks.

Abstract

Automatic service restoration in distributional networks has been studied. After a thorough examination of the literature in this field, one algorithm was selected and implemented as a Fortran program. This implementation was then proved to be effective. Further improvements to the algorithm was introduced and in three cases these proved to be succesful. For example: An operational aid indicating roughly the order of the switching operations was added, the number of switching operations decreased in some cases and the priority of the zones was made more important.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
English

Number of pages
143

Recipient's notes

Security classification

ACKNOWLEDGEMENTS

This report was written as an examination paper at the Lund Institute Of Technology. The work was performed at ABB Network Control . The author wishes to thank Dr Rolf Johansson Lund Institute of Technology department of Automatic Control, Borje Nilsson ABB Network control and other personnel at ABB Network Control for their support and good advise.

TABLE OF CONTENTS

1. BACKGROUND

1.1 TASK

The task has been to study the existing methods of automatic service restoration in distribution networks. Furthermore one algorithm should be selected, and implemented in Fortran.

1.2 HISTORY

Automatic service restoration is a new area in electric power distribution. Nothing was published in this area until 1980. In 1980 Ross (1) defined the service restoration problem as the minimization of the weighted sum of unserviced customer hours and unserviced energy subject to line capacity constraints, transformer constraints and voltage drop constraints. Since then different methods have been developed. Today two different approaches stand out, one mathematical and one based on expert system technique.

1.3 DISTRIBUTION NETWORKS

The distribution network has a radial structure, looking similar to that of a tree. In the feeding substation, there is a transformer. The transformer is connected to a feeder via a bar and a feeder breaker. Branchpoints give the treelooking structure . Several switches (cut switches or tie switches) are included, creating zones (or nodes or sections) in the network. These sectionalizing switches makes it possible to isolate a fault so that a repair can be done. Because of the radiality of the network most of the faults not only causes the faulted zone to be deenergized but also all the nonfaulted zones further down viewed from the transformer. These zones can often be supplied from other transformers. Therefore service restoration is needed.

Distribution networks are often large. The normal size is about 100-1000 feeders, each one consisting of 10-500 nodes (varying much between different power companies). The voltage in these networks is in the range from 4-43.5 kV.

1.4 RESTORATION

When a fault has occured a protective relay trips the circuit breaker. Next the fault has to be identified. One way to do this is by first opening all the sectionalizing switches and then closing the circuit breaker. The switches are then reclosed one by one in until the circuit breaker is tripped again. The faulted zone is identified and has to be isolated. At this point the operator must decide a sequence of switching operations that have to be performed in order to restore the nonfaulted deenergized zones, without causing any violation. This is done by hand calculation based on a loads forecast or by the operators' experience. Often is the sequence thus produced not optimal. A better solution would certainly be reached if a computer would produce it.

1.5 OPTIMIZATION ASPECTS

Studies of the service restoration problem show that it is a large scale combinatorial optimization problem. The only known method for solving combinatorial optimization problems is to test all possible solutions. Because of the size of the problem this solution often cannot be reached in the lifetime of a computer. In a network containing 1000 switches, there are 10^{300} possible solutions.

The most common method for solving large scale combinatorial optimization problems is to use a heuristic search method. This means that knowledge about the problem is used to guide the search. In this case a possible rule might be that all the circuit breakers are left closed unless the fault has occured in a transformer or a bus. By applying rules like this the number of possible solutions reduce and the execution time becomes reasonable. By rejecting possible solutions that seem to lead to a non-optimal solution one may loose the true optimal solution. Instead an approximate optimal solution is reached. This solution is often so good that it is unnecessary to find the true optimal solution. Such is the case in service restoration. Heuristic search methods are suitable both for ordinary programming languages such as fortran and symbol manipulating languages such as lisp or prolog.

2. INFORMATION RETRIEVAL

During my information retrieval I found two main approaches, which will be dealt with separately.

2.1 EXPERT SYSTEM APPROACH

The most common approach to service restoration is an expert system although the other methods are faster. This is due to the 'higher' programming level in lisp or prolog. It is easier to make changes in your algorithm using these languages.

Morelato (2) used the expert system approach providing a framework for a whole family of algorithms concerning distribution network automation. He proposed a true expert system search approach, using a decision tree to solve different problems such as service restoration ,the minimization of losses and costs,load balancing or combinations of the above. In fact, service restoration is a special case of load balancing. If the deenergized zones are connected to a support feeder creating a violation and then applies the load balancing algorithm , a service restoration algorithm would have been used. Any violations that may exist have to be cut off in the end. In an example he demonstrated how his service restoration algorithm works. This approach didn't seem right for fortran programming and besides there were some other weaknesses. For one thing voltage drop limits were discarded.

Lee and others(3) propose an algorithm that seems to be one of the best. They have certainly the highest level of ambition. Aspects like the service crews availability, traffic situations and such things are mentioned but not included in the algorithm. The algorithm is an extension of an earlier work (4) by the same authours. In their latest paper the authors criticize the mathematical approach for not considering such important factors as priority of zones (A hospital ought to have higher priority than a sports arena) and availability of crews. The algorithm is not a search algorithm. Instead it is something of trial and error. In short, the algorithm is as follows:

* Put all the deenergized non faulted zones in a list
* Determine priority of zones
* Try to restore the zone with highest priority first

This is done until no more zones can be restored. The restoration is done in three steps.

* First a whole group of feeders is restored.If the restoration causes a violation, the restoration is abandoned and the second step is tried.
* If the first step fails, only single zones are restored . If the restoration causes a violation, the restoration is abandoned and a third step is tried.
* The third step first tries to transfer load from the feeder that will take up the restored load so that when doing so, it will not be violated. Then the restored load will be transfered.

The last two steps seem to be quite simple but they are built by 250 rules. The proposed system may be used in a partly automated network (Some switches can be remote-controlled). This causes some switches to be prefered when it comes to switching operations and therefore the remotely controlled switches get a higher priority. The system not only produces a sequence of switching operations but also indicates the order of them. The execution time seems to be satisfactory (less than 5 seconds).

2.2 MATHEMATICAL APPROACH

This approach is the one used in the few existing physical test systems.

In (5) Castro and others show the results from a simulation of a proposed system. The simulation is a part of a project discarding economics to create a more effective distributional network. Communication links are thought to be installed in order to guide switches by remote control. He gives only hints of what the system contains and hardly anything of how it works. A major part of the paper describes the advantage of using a switchtable (an interface between pure algorithms and the real world). The system is thought to have two functions for network control.

First a load balancing algorithm which iterates a solution. Only one switch operation is suggested at a time until the network is balanced. If that is the case, the risk of overload in the network is reduced.

The second function is service restoration in case of emergency. The basic difference between these two algorithms according to Castro is that the service restoration produces the whole sequence of switching operations at one time and does not wait for the first operation to be completed.

This paper is perhaps more interesting for those who deal with the problem of installing a commercial system.

In (6) Kato and others also handles the case of a real system. Kato works for Tokyo Electric Power Co. and in the paper he discribes their installed test system in Ginza.(It was installed in 1981) Much is said about communication etc. but a paragraph about service restoration is found.

Also here only hints are given about the algorithm. It works in two steps.

First all the loads that can be transfered directly without causing any violations are transfered. Then loads in those feeders who will be violated by further restoration is transferred to other feeders before they are able to take up new restored load.

Test runs (The system was put into trial application in 1984) show that non-faulted zones could be restored in five minutes, within about a tenth of the time for manual restoration.

Aoki and others (7),(8) have produced several papers in the field. Obviously he has studied Lee (3) because (7) is an improved algorithm compared with (8). He has introduced priority for the zones thereby showing that it is as easy to use priority in the mathematical approach as in the expert system approach. There are more improvements. Voltage drop constraints are included as well as the powerful effective gradient method (9),(10). The algorithm works in four steps.

1. After fault isolation the deenergized zones will be restored to the feeder with the largest violation margin , discarding any violations.

2. If violations occured these are tried to be removed by transferring loads to adjacent feeders until no violations exist or no more loadtransfers can be made without creating new violations.
3. If violations still exist loads will be disconnected until no violations exist.
4. In the fourth step the previous deenergized loads (in step 3) will be reenergized one by one and step 2 will be tried once again among former violation feeders. If it fails, the load will be disconnected again.

The effective gradient method is used to select in which order the loads are tried to be transfered or cut off. This is the only algorithm which has been tested on a large network (1188 nodes) and it produced a solution in less than a second.

2.3 SELECTION

There were two main candidates for implementation, Aoki (7) and Lee(3). Lee seemed to be the best in some respects and Aoki in others.The greatest advantage with Lee's algorithm is that the priority of zones will be determined by the operator during execution and interpretations of the priority-measurements are postponed until this moment. (No file containing predecided priority exists). On the other hand seemed this advantage possible but not suitable for implementation in Fortran.

Aoki's algorithm on the other hand was obviously possible to implement in fortran and since it had no real disadvantages in any respect, this algorithm was selected.

3. IMPLEMENTATION

3.1 ALGORITHM

As it was previously stated the algorithm solves the problem in four steps. Below follows a flowchart of the algorithm

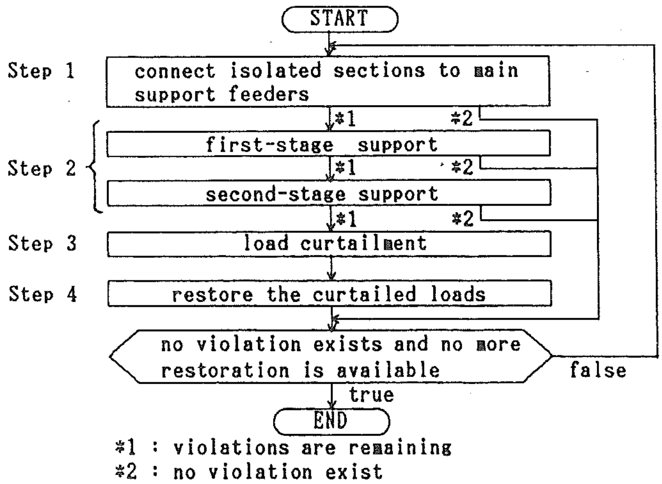


figure 1 (by Aoki). Flowchart of algorithm.

Step 1

First, the fault is isolated. This creates groups of deenergized sections. If a fault occurs in a branch- point there will be more than one group. For each of these groups all possible connections are examined. The best connection, that is the one to a feeder with the biggest margin of getting violated, is chosen and the group is reenergized.

If the connection causes a violation, the magnitude of these violations is stored in a violation vector. For each feeder both voltage drop violations and line capacity violations are stored.The violation vector is later used in order to decide between which feeder pair a load transfer should be performed.

If no violations occurred the violation vector equals zero and a feasible solution is found. The algorithm stops.

Step 2

Step 2 is only used when at least one feeder is violated. It has two principal parts, first and second stage support. Both these parts try to eliminate violations by transfering load from a violated feeder to a nonviolated.

In the first stage support, load is tried to be transfered from a violated feeder to a support feeder if it is possible. One cut switch has to be chosen for load transfering and the algorithm first choses the one that will reduce the violation the most. This is done by solving the following optimization problem.

$$Max \sum_{j \in I_b} \frac{h_j}{\alpha_j(H_j + \beta)} y_j$$

$$Sub.to \sum_j A_{ij} y_j \leq b_i (i = 1, 2, ...)$$

The variable y_j is a 0-1 variabel. It indicates the switch status. I is the set of possible cut switches that may be used for load transfer. H_j stands for remaining violation while h_j stands for the effective withdrawal of the violation. A is the capacity of the transformer, α_j the node priority and β is only included for avoiding zero divide. This problem is solved by using the dual effective gradient method (9),(10) which states that the order in which the load transfers are to be tried, is in descending order of the measure

$$\frac{h_j}{\alpha_j(H_j + \beta)}$$

This procedure is repeated until no more violations exists or until no more transfers can be performed without causing any violations in the support feeder. A flowchart of the first stage support is given in figure 2.

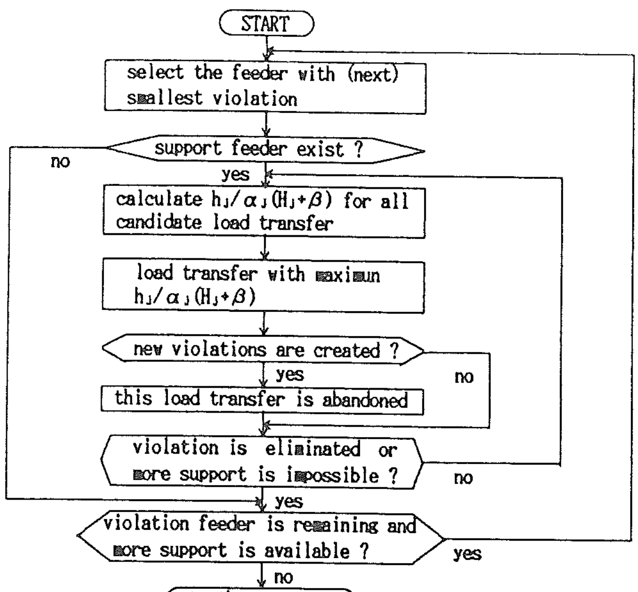


figure 2 (by Aoki). Flowchart of first stage support

The second stage support is only tried when the first stage support fails to transfer load. It works similarly to the first stage support. The second stage support begins with a modified first stage support allowing violations in the support feeder. Then this violation is eliminated by performing a first stage support on the violated support feeder. Aoki suggested a slightly different measure for the first stage support of the violated feeder (Not the violated support feeder). He suggests

$$\frac{h_j}{\alpha_j(a_j+\beta)}$$

Here a_j is the load magnitude of the transfered node. The second stage support performed until violations in the feeder is eliminated or until no secondary support feeders is able to take up any more load. In figure 3 a flowchart of the second stage support is found.

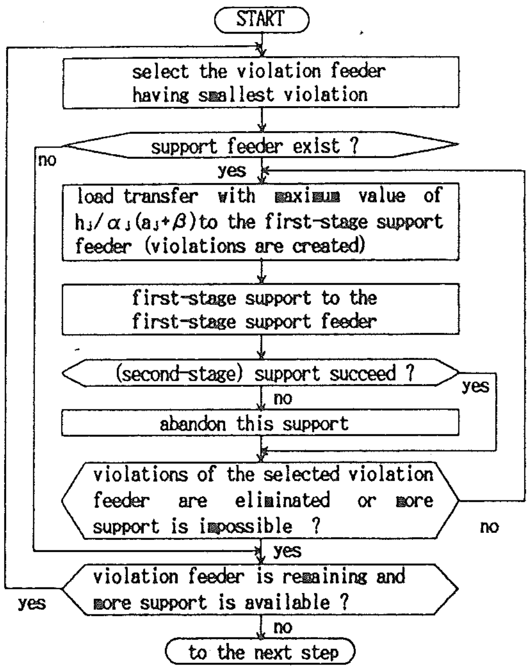


figure 3 (by Aoki) flowchart of second stage support

These two procedures are closely associated with each other and because of the similarity, a way of implementing them that could be hard to understand was selected. Some comments are needed.

First the candidate for a load transfer is to be selected. A file containing the pair of switches that is involved in the load transfer, the transfered load and an estimate of the reducement of the violations is stored in a file by a procedure called stage. This file is read by the procedure 'stage1' which calculates the different measures and selects a pair of switches able to perform a load transfer.

The two feeders involved in the load transfer is identified and all the different currents, node voltages and other data that could be changed due to a load exchange is first stored in a file called history. Then the actual load exchange is performed and new physical data is computed. If no new violations occur, all the 'backup'-files are deleted.

If new violations is created the former 'state' is read back in the net and a new trial is made until success or no more possible exchanges may be performed. Then 'stage1' terminates with ok=false, keeping all the 'backup'-files.

'Stage2' then calls 'stage1' with ok=false . This causes 'stage1' to perform a load exchange discarding violations and as an outparameter you get 'outfee' which is the violated feeder. Then an ordinary stage1 is performed for this feeder.

The 'stack of backupfiles' thus created is difficult to handle and if someone ever needs to make any changes in these procedures they are advised to think at least twice, before making any changes in order to fully understand them. If any bad instruction occurs in these procedures, the algorithm will produce undesired solutions, and the fault might be hard to detect.

Step 3

If violations still remain after step 2 then step 3 is executed. In step 3 a load curtailment is performed. The dual effective gradient method is also this time used to produce a measurement helpful in selecting the end section that will be disconnected. The measurement will be

$$a_j(H_j + \beta)$$

For every turn of step 3 the section with the smallest value of the above will be selected. A small load causing a small remaining violation is chosen. When no more violations remain, step 4 is executed

step 4

In step 4 the previously curtailed loads are tried to be restored. This can work if more than one feeder were violated. Then there is a possibility of restoring loads between two former violation feeders. The dual effective gradient method produces this time the following measure

$$\frac{a_j}{(|R_j| + \beta)}$$

R_j is the violation vector.When the cutswitch j is closed, one feeder will be violated.

The curtailed sections will one by one be reenergized in decreasing order of the measurement above and a first stage support will be performed. If the first stage support fails the load will once again be disconnected. In figure 4, a flowchart of step 3 and 4 is found

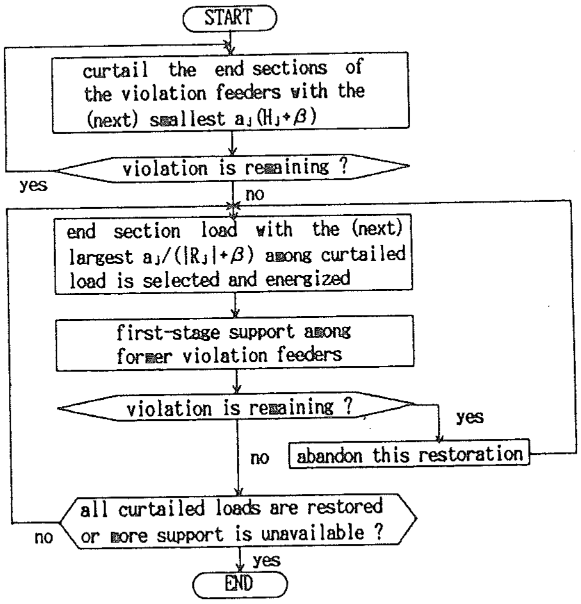


figure 4 (by Aoki) Flowchart of step 3 and 4

3.2 DATASTRUCTURE

In the datastructure six different types are found. These are

1: transformer

2: feeder

3: cable

4: branchpoint,

6: switch

Every type is represented by matrices containing pointers to the closest neighbours and physical data. More details about the data structure is found in the appendix.

3.3 NETWORK REPRESENTATION

There are a few simple rules that must be followed when representing a network in order to make the algorithm work.

- The feeder is only used to provide a useful entrance to the network and to attach the violation vector.
- There must be a switch between the feeder and the first node in the network.
- A cable is used to connect elements of the other types and can therefore not be attached to another cable.
- The switch also represents the circuit breaker in the algorithm in order to reduce the size of the code.

3.4 COMPUTATIONS

The following data is presupposed to be known:

1. The network's topology
2. Cable impedances
3. Node impedances based on a load forecast
4. Transformer voltages

The transmission net was presumed to be strong so that the transformer voltage was kept fairly constant even if the load increased. Series impedances were used as a model of impedances. Knowing this and the fact that the network has a radial structure, the voltage and the current in an arbitrary point in the network could be computed. In figure 5 a radial structure is seen.

First the total impedance must be computed. This is done by traversing the 'tree' of the transformer. When a 'leaf' is found the computation begins. On the way up the computations proceed storing parallell impedances in matrice BRAPH, when a branch point is reached. The search of a new leaf starts.

When the impedance is known the current is easily computed.

Then by applying the laws of Kirchhoff, all currents and voltages are easily computed by once again traversing the tree.

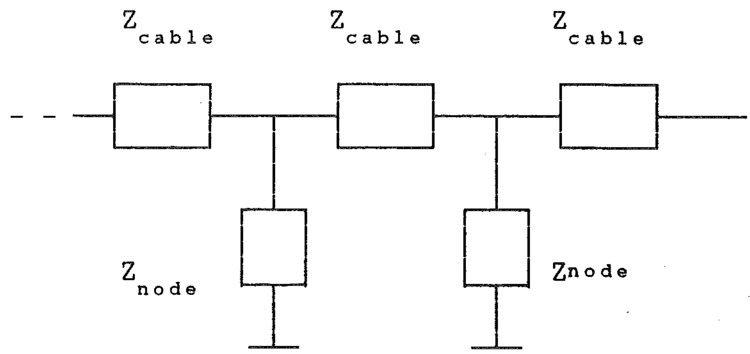


figure 5. Radial structure

This simple move around method seemed to be the best in this case. More sophisticated methods taught in circuit theory seemed to be more difficult to handle in this case because the number of impedances in the local network (the one that is connected to a single transformer) is continuously changing. Too much work would be spent on setting up matrices.

There are some problems concerning numerics. After a leaf node the current often differs from zero but the fault is not big enough for concidering double precision (about 0.1 percent of transformer current). It is necessary to remember that the load forecast probably contains some error.

3.5 IMPROVEMENTS

During the implementation of the algorithm, some possible improvements were discovered.

The multi stage support may be generalised. A third stage support can be performed tolerating violations in the secondary support feeder and a first stage performed on this violated feeder. Aoki mentions this but consider it unnecessary because "the distribution systems are often designed so that all de-energized loads in an island can be restored without creating any serious violations" (Island means a group of de-energized sections). This is certainly the case when the transformer carries a normal load but faults also occur in case of a peak load. A third stage support was implemented.

The measurement given in both the first and second stage supports could be improved. The suggested improvement was

$$\frac{h_j}{\alpha_j(H_j+\beta)} * \frac{1}{a_j}$$

This causes a small load causing small reamaining violation to be selected and this seemed to be an even better measurement than the one by Aoki.

In the load curtailment the priority is discarded by Aoki. This seemed to be wrong. A high priority zone should not be selected for load curtailment if another candidate with lower priority exists. Therefore a possible improvement might be

$$\frac{a_j(H_j + \beta)}{\alpha_j}$$

In this algorithm a small numerical value of alfa indicates a high priority zone

The Aoki algorithm only indicates the switches that must be changed. It never tells you in which order to perform the switching operations. This may be fatal because it can create serious violations due to the fact that first violations are created and then these violations are tried to be reduced. Therefore some kind of indication of order is needed. It is only the switching operations that first reenergizes the isolated zones that can cause a violation, and therefore these must be performed in the end. An improvement of the algorithm would be that switching operations due to fault isolation would be indicated by the value 1. These operations must be performed first. Then the switching operations due to step 3 and 4 are performed so that no violations occur. Then those switching operations caused by the first and second stage support can be performed and eventually the switching operation caused by the reenergizing of isolated nodes.

3.6 TESTRUNS

In order to investigate if the suggested improvements really improved the result, a series of test runs was made on a small network taken from (7). This network is shown in figure 6.

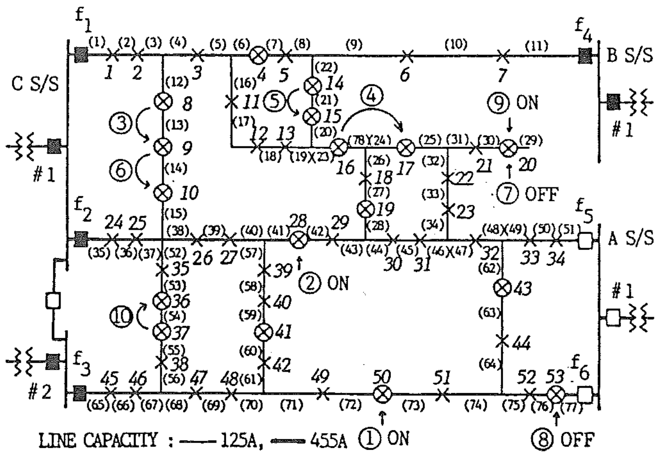


figure 6. Topology of test system (by Aoki)

A number of different qualities were demanded for a good solution.

- The execution time has to be rather short, less than 10 seconds is prefered.

- The number of switching operations need be as low as possible. This is due to the fact that today when almost every switch is manually controlled, it takes some time to perform the switching operations. (mostly due to the distances between the switches in the network)
- The restored load ought to be as big as possible. It may be difficult to measure the restored load because it depends on the node voltage and this can be changed during execution. Different feeders have different loads and therefore different voltage drops.

The plain Aoki algorithm was tested. The test showed that this implementation was equally effective as the Aoki implementation (Execution time ranging from 5 ms to 1 s) The number of switching operations and the amount of restored load seemed to be equivalent. This implementation was then used as a reference.

One sequence of test runs of specially selected faults was run. The faults were severe causing major violations in order to see the differences more clearly.

The third stage support was tested. It was obvious that the execution time rised dramatically to about 5 seconds. This however was not so serious.

It was soon noticed that the third stage support had little effect on the amount of restored load unless the network was unbalanced with all primary and secondary support feeders carrying major loads while other feeders only carried minor to normal load. This was hardly a surprising result.

More important was the number of switching operations. It increased by a factor of approximately 1.5 (In average). When the second stage support has failed nothing is tried in order to reduce the violation until the load curtailment take place.

In the load curtailment process only one single switch is changed. But if the third stage support is used successfully, then for every successful turn of this step six switches are changed. (Two switches connected to the primary support feeder, two to the secondary and two to the tertiary support feeder) The switches changed due to a successful third stage support can be seen in the figure 7.

```
F1 - - - - - - - SW1 - - SW2 - - BP1 - - F2
                                  !
                                 SW3
                                  !
                                 SW4
                                  !
F4 - - BP3 - - SW6 - - SW5 - - BP2 - - F3

   F:Feeder    SW:Switch    BP:Branch Point
```

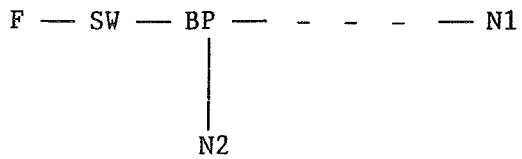
Figure 7. Topology typical for a third stage support.

Between switches and branch points an arbitrary number of sections may be found. The third support is applied to feeder F1 transfering load to the primary support feeder F2(SW1,SW2), causing a violation. The violation is eliminated by a further load transfer to feeder F3 (SW3,SW4), which in its turn has to transfer load to F4 (SW5,SW6) before it can take up any more load. The load transfers in this case are carried out by opening the odd switches and closing the even.

This increased number of switching operations made the suggested improvement less desired because the number of switching operations was already large (about 10-15). The 'optimal' solution must have a chance of being reached before the repair of the faulted zone is finished.

During test runs of the third stage support, the suggested improvement of the measurement in the second and first stage support was detected. When a fault caused serious violations,often the most severe line capacity violation was in the 'root' cable. This could be reduced by transfering any load in the tree, and sometimes a node which had not been deenergized was selected. This cannot be desired. A load transfer should always be made among former deenergized loads in order to best reduce the violation.

It is a known fact that in the beginning (before the reenergizing of isolated sections) the violation feeder was non violated. Therefore it is desired to perform the load transfer in the reenergized sections because these created the violation. The reason for violation must have a greater attention than the size of load and this is done by dividing the proposed measurement (Aoki) by H_j. An example is shown in figure 8.



F stands for feeder, SW for switch, BP for branch point and N for node

Figure 8. Typical topology

The node N1 has been among the reenergized nodes. When it was reenergized, it caused a violation in feeder F. In this case the voltage for N1 is 20 V below the voltage drop limit but N2 is not violated. In the root cable, that is the cable from the feeder to the branchpoint, the current exceeds the line capacity with 40 A. In this case the violation vector would be (20,40) Only these two candidates (N1,N2) for load transfer exist. Both nodes have the same priority. The load of N1 is 60 A and the load of N2 is 40 A. Computing the measurement(by Aoki) for these nodes gives:

N1: $h_j=\sqrt(2000)$,H_j=0 and measurement=0.75

N2: h_j=40 ,H_j=4,7 and measurement=1.0

In this case N2 would have been selected. If the proposed measurement was to be used N1 would be selected instead. The new measurements would be:

N1: 74.0

N2: 0.21

It is obvious that the best choice for reducing the violations is transfering N1. Then both a voltage drop violation and a line capacity violation would be avoided. This was later proved to be an improvement, but not a major one. It reduced the number of switching operations with, in some cases, one or two. (The number of switching operations never increased but for the most cases it was the same.) The amount of restored load was unaffected (in average). The third suggested improvement to include the priority in the load curtailment was also an improvement in the sense that according to priority rules some previously wrong decisions were made right. An example of this is seen in figure 9.

```
              N1
              !
 ----------- BP1
              !
              N2
              !
              N3
```

Figure 9. Typical configuration in case of load curtailment

Here both nodes N1 and N3 are violated with a voltage equally large, 20 V below the voltage drop limit. N1 is a high priority zone (α_{N1}=1) while N2 is a low priority zone (α_{N2}=10). The loads are: N1=10 A , N2=20 A.

Because of the equality in violation both the zones cause equal reductions in the violation vector. In this case the measurement proposed by Aoki will only depend on the size of the load. Therefore the high priority zone will be selected.

This could cause many damages in the society. An important operation in a hospital may have to be interrupted or postponed.

According to the proposed measurement(priority included) the low priority zone would have been selected for load curtailment. Then it is up to the operator to decide the different priorities. In this case the number of switching operations and amount of restored load must be of second interest. Anyway no major changes emerged. This is due to the fact that the service restoration problem is a combinatorial optimization problem. A decision that perhaps is seen as a 'bad' decision in the sence that it restores less load could in fact be better than the decision suggested by the heuristic search approach. It is the combination of decisions taken that lead to the optimal solution.

4. RESULTS

* As a result of an information retrieval two major approaches to the service restoration problem was found (Expert System Approach and Mathematical Approach)
* The mathematical approach is the only one which has been tested in a physical system
* The result from this test system was promising.
* The mathematical approach was selected due to the facts that code was to be written in fortran and the succesful tests that others had done in the field with this approach.
* An algorithm was chosen for implementation, which fulfilled all possible requirements for a good performance such as a short execution time and a small number of switching operations.
* This algorithm was then implemented in fortran.
* The implementation was proven to be as effective as the original implementation.
* Four possible improvements were suggested
* These suggested improvements caused a minor improvement in performance of the algorithm
* An operator aid was added. Not only the switching operations were presented but also a rough indication of the order of these operations was given.
* In a physical system the use of a service restoration algorithm depends on the accuracy of the load forecast.

5. FUTURE

As it has been previously hinted, the service restoration algorithm is only one part of a group of algorithms thought to be used in an automatized distribution network in the future. The reason for the focus on the service restoration algorithm at the moment depends on the possibility to use it with locally controlled switches. It would be too expensive having service crews continuously driving around changing switches in order to improve the performance of the network. In case of an emergency however, it is essential to transfer load. In the future these other algorithms (the load balancing algorithm, the minimum loss and the minimum cost algorithm) as well as the service restoration algorithm will probably all be found in an ordinary automatized distribution network. Due to the high cost of installing communication links, hardly all sectionalizing switches will be remotely controlled. Instead, only those that often will be used for a load transfer will be supplied with remote control.

6. REFERENCES

(1) Ross,D.W. Carson,M Cohen,A.I : "Development of advanced methods for planning electric energy distribution systems" U.S DOE Report ET-73-C-03-1845 feb. 1980

(2) Morelato,A.L. Monticelli,A. "Heuristic search approach to distribution system restoration" IEEE/PES Winter meeting, New York, 1989 paper 89 WM 111-6 PWRD

(3) Lee,S.J. Liu,C.C. Venkata,S.S. "an extended expert system for service restoration of distribution feeders" IFAC symposium power systems.Modelling and applications. Brussels September 5-8 1988

(4) Liu,C.C. Lee,S.J. Venkata,S.S. "An expert system operational aid for restoration and loss reduction of distribution systems" Proc 1987 IEEE PICA conference, Montreal, May 1987.

(5) Castro,C.H Bunch,J.B. Topka,T.M. "Generalized algorithms for distribution feeder deployment and sectionalizing" IEEE Transactions on power apparatus and systems. Vol. PAS-99, No 2 March/April 1980

(6) Kato,S. Naito,T. Kohno,H. Kanawa,H. Shoji,T. "Computer based distribution automation" 1985 PICA conference, pp. 374-380 1985

(7) Aoki,K. Nara,K. Itoh,M. Satoh,H. Kuwabara,H. "A new algorithm for service restoration in distribution systems" IEEE/PES winter meeting, New York 1989 paper 89 WM 085-2 PWRD

(8) Aoki,K. Kuwabara,H. Satoh,S. Kanezashi,M. "Outage state optimal load allocation in distribution systems" IEEE Trans PWRD vol.PWRD-2 No.2, October 1987

(9) Senju,S. Toyoda,Y. "An approach to linear programming with 0-1 variables" Management science, vol. 15, No. 4, 1968

(10) Toyoda,Y. "A simplified algorithm for obtaining approximate solutions to zero-one programming problems" Management science, vol.21, No. 12 1975

7. APPENDIX

7.1 USER'S MANUAL

All the useful files are stored on node SNEV02 at ABB Network Control. The files will be found in directory EXARB ROOT: MBERGSTRAND .

In the file CONSTANTS.FOR are all the necessary constants declared and this file is included in all files in order to facilitate any changes in the structure.

This was the same reason for having the file COMMON.FOR . In this file all global variabels needed were declared . This file is also included whenever it was necessary.

The constants MAXNRT, MAXNRF, MAXNRC, MAXNRB, MAXNRN and MAXNRS have to be changed before the program is used on a new network.(nr of transformers, feeders,cables,branchpoints,nodes and switches) Another constant, BRANCH, may have to be changed. It indicates the maximum allowed branches for any branchpoint in the network.

When this is done all the files RESTORE, OUTFI, MOVING, TESTS, COMP, STEP1, STEP2, FAULT, FETCH and CHG must be recompiled and linked together. The main program is in file RESTORE.FOR.

Then an input file must be made, containing the physical data and the topology of the network.

All the different components (transformers,switches feeders,cables,branch points and nodes) must be numbered.

The program uses formatted input. The procedures of interest with appropriate format is located in the file OUTFI.FOR. The procedures are called WRITEF and INFI. Studies of the structure chart found later in the appendix is essential understanding WRITEF and INFI. The structure chart tells what the different elements in the matrices used stand for.

In the file the different types is in the order

- o Transformers
- o Feeders
- o Cables
- o Branch Points
- o Nodes
- o switches.

The format used is as follows:

- All real values are written using exponents (0.100E+01).
- First two records are needed to define the transformers (common to all the different transformers).
- The first contains the constant MAXNRT (MAXimum NumbeR of Transformers ,integer) written in position 3-8
- The second contains the maximum voltage in the network (real) written in position 3-14

Then one record is used to represent evey single transformer. This record is repeated MAXNRT times. It contains the following data:

- Maximum value of P in pos. 3-14 (real)
- Maximum value of Q in pos. 17-28 (real)
- A pointer to a cable in pos. 31-36 (integer)

The transformers, as well as any other types , have to be in order (Trafo 1,trafo 2...).

FEEDERS come after TRANSFORMERS

- This is followed by one record containing the constant MAXNRF written in pos. 3-14 (integer)

Then one record containing the topology for each single feeder is repeated MAXNRF times. It contains of:

- A pointer to the cable that leads towards the transformer in pos. 3-8
- A pointer to the cable that leads 'down' towards the nodes in pos. 11-16

Both are integer values

The next type in the data file is CABLE.

- First one record containing the constant MAXNRC, written in pos 3-8 must be made. (integer)

That is followed by seven records ,containing the physical data and the topology for each single cable. These seven records are repeated MAXNRC times. The records are shown below:

Record 1. R pos. 3-14 real

Record 2. X pos. 3-14 real

Record 3. Imax pos. 3-14 real

Record 4. Type 1 pos. 3-8 integer

Record 5. Nr 1 pos. 3-8 integer

Record 6. Type 2 pos. 3-8 integer

Record 7. Nr 2 pos. 3-8 integer

Type and Nr are refering to the element, which the cable is connected to.

BRANCH POINTS come after CABLES.

- One record containing the constant MAXNRB in position 3-8 (integer)

Then for each branch point:

- First the actual number of branches, less than the constant BRANCH, is written in one record in pos 3-8
- Then one record containing one pointer to a cable is written in position 3-8 for every branch.

This is repeated MAXNRB times.

An example:

In one network there are 2 branch points. The first branch point has 4 branches and the second has 3. The first branch point is connected to cables 6,7,8 and 9. The second is connected to cables 2,3 and 4. This is how it appears in the data file.

```
2
4
6
7
8
9
3
2
3
4
```

NODES come after BRANCH POINTS.

- One record containing the constant MAXNRN in pos. 3-8
- Then the voltage drop limit is written in one record in pos. 3-14. The voltage drop limit is an actual voltage and not a percentage of the transformer voltage.

Then three records is used to represent one node.
For each node:

```
Record  1. Pointer to cable pos. 3-8 (int)
           Pointer to cable pos. 11-16 (int)
Record  2. R (Resistance)   pos. 3-14 (real)
           X (Inductance)   pos. 17-28 (real)
Record  3. Priority         pos. 3-14 (real)
```

As an example, the physical data and the topology for one node, connected to the cables 3 and 4 is seen below as it must be written in the data file. The node's impedance is (543,j310) and the priority is 1.0.

```
     3       4
0.54300E+03    0.31000E+03
0.10000E+01
```

The above is repeated for each node.

SWITCHES come after NODES.

- One record contains the constant MAXNRS in pos. 3-8.

The topology come before the status.

One record contains the topology.

- A pointer to a cable in pos. 3-8
- A pointer to a cable in pos. 11-16

This is repeated for each switch.

Then follows the status for each switch in one record for each switch.

- Record: Switch status (0,1) in pos. 3-8

This is repeated MAXNRS times.

An example:

One network has two switches. The first switch is connected to cable 10 and 12, and it is opened (Status=0). The second switch is connected to cable 8 and 9, and it is closed (Status=1).

This is how it looks in the data file:

```
  2
 10
 12
  8
  9
  0
  1
```

Following these instructions in order, a correct data file is produced.

All files of the type '.DAT' contains a network, with the name indicating whether the load is minor, medium or major.

During execution of the program all commands and answers must be written in CAPITAL LETTERS.

When the program starts its execution, first a check of the topology is made. Some faults can be detected and messages will be written on the screen telling the user what to do. The faults that will be detected is of the kind: "if you can 'go' to a neighbour but you can not go back to the starting point" it must be wrong in the network topology. Most of the faults are detected by this check but fault can also 'cooperate' so that two faults make it look like no fault. If errors of this kind occur in the network these faults most certainly are detected later causing messages like this: WRONG IN STEPDO.NODE, FORTRAN STOP.

If messages of this type appear, the program provides a way of easily finding these errors. When the first test of the topology is made, the program asks if the operator desires a visible track. If the answer is 'Y' the program will print out all the steps taken in the subroutine init. Then it is easy to follow the program until an error is found. Then the latest point must be remembered. In this point the error is found. Edit the data file and try again If the topology is correct the text INIT PASSED will be written on the screen

Later in this section transcripts of possible test runs are showed with comments added afterwards.

The program will then ask if any changes are to be made in the network (concerning load). If you wish to do so the answer is Y. Then the program will ask if you wish to change the total load. If not, you have the possibility to change a single node impedance. If you choose to change the total load the program will ask for a factor with which every single node impedance in the network will be multiplied. As a guideline you can use 1.1 if you wish to reduce the load with 10 percent ($P = \frac{U^2}{R}$) but the system is not linear.In case of a change in a single node the old value of the impedance is written on the screen and then the new value is read. Then the program asks for a filename. The new load condition is saved for later use. The filename can be seven characters long. All names exept HISTORY,STAGE and CURTAI are allowed. Files with these names are used internally. See also in the transcripts.

Then the program will ask for a point in the network where the fault has occured. It can be any point in the network. A point is indicated by its type and number (both integer values). Available types are

1: Transformers

2: Feeders

3: Cables

4: Branch Points

5: Nodes

6: Switches

Then the algorithm starts working, continuously writing data on the screen. A careful study of decisions taken can be done. In the end a menu is presented containing execution time, the switches that have changed status, the order of these switching operations, deenergized load and restored load. Then the program asks if some more faults (A new test case) is desired and if so the program restarts.

7.2 EXAMPLES

Four different examples are shown in order to demonstrate how the program works.

WHICH FILE IS TO BE READ?

TRAFOS IS OK
FEEDER IS OK

FAULTED NETWORK
BRANCH-POINT NR 1 IS CONNECTED TO CABLE NR 30

CABLE NR 30 IS CONNECTED TO
TYPE= 5 NR= 6
AND
TYPE= 6 NR= 4
FAULT IN BRANCH-POINT(4) OR IN CABLE(3) ?
THE FOLLOWING BRANCHES

1
2
30

THE FAULTING ONE IS (NOT) CONNECTED TO 30
THE BRANCH IS CONNECTED TO CABLE NR ?
BRANCHPOINTS OK
NODES OK
SWITCH OK
SWITCH OK

TRAFOS IS OK
FEEDER IS OK
BRANCHPOINTS OK
NODES OK
SWITCH OK
SWITCH OK

THE NETWORK IS CORRECT
DO YOU WISH TO HAVE A VISIBLE TRACK? Y/N
DO YOU WISH TO MAKE ANY CHANGES IN THE NETWORK?Y/N
INIT PASSED
FAULT IN

1: TRANSFORMER
2: FEEDER (MAIN SUPPORT)
3: CABLE
4: BRANCHINGPOINT
5: NODE

WHICH TYPE OF COMPONENT?

WHICH NODE?
ENTERS STAGE 1 2
ENTERS STAGE*************************************
READS IN STAGE1 ***************************
IMPROVEMENT,FEEDER 1.614994 6
IMPROVEMENT,FEEDER 0.2358537 1
IMPROVEMENT,FEEDER 0.2358537 5
SAVENE**
2 6
SAVENE**

```
LEAVES STAGE1 AFTER A SUCCESFULL SESSION,OK=T
ENTERS STAGE 1          2
ENTERS STAGE************************************
READS IN STAGE1 ***************************
IMPROVEMENT,FEEDER   5.272868                6
IMPROVEMENT,FEEDER  0.2615825                1
IMPROVEMENT,FEEDER  0.2615825                5
SAVENE********************************************
          2            6
SAVENE********************************************
LEAVES STAGE1 AFTER A SUCCESFULL SESSION,OK=T
ENTERS STAGE 1          2
ENTERS STAGE************************************
READS IN STAGE1 ***************************
IMPROVEMENT,FEEDER   6271.001                6
IMPROVEMENT,FEEDER  0.0000000E+00            1
IMPROVEMENT,FEEDER  0.0000000E+00            5
IMPROVEMENT,FEEDER  0.0000000E+00            6
SAVENE********************************************
          2            6
SAVENE********************************************
LEAVES STAGE1 AFTER A SUCCESFULL SESSION,OK=T
ENTERS STAGE 1          2
ENTERS STAGE************************************
READS IN STAGE1 ***************************
IMPROVEMENT,FEEDER   2805.759                6
IMPROVEMENT,FEEDER  0.0000000E+00            1
IMPROVEMENT,FEEDER  0.0000000E+00            5
IMPROVEMENT,FEEDER  0.0000000E+00            6
SAVENE********************************************
          2            6
SAVENE********************************************
LEAVES STAGE1 AFTER A SUCCESFULL SESSION,OK=T
      TOTAL COMPUTATION TIME OF ALGORITM  275 ms

      THE FOLLOWING SWITCHES HAS CHANGED STATUS

                  37    4
                  38    3
                  45    1
                  46    1
                  50    3

      TOTAL DEENERGIZED LOAD DUE TO FAULT
       0.29190E+07

      TOTAL RESTORED LOAD
       0.29190E+07

      ANOTHER FAULT? Y/N
FORTRAN STOP
```

COMMENTS

In the previous example the possibility to discover a single fault was demonstrated.

The error in the topology was identified and then the data concerning the topology for the two possible faulted elements were written on the screen. The operator then identifies the faulted element. (In this case the branch point) All connections for this erroneous branch point are written in the screen and the faulted connection is marked by the text "THE FAULTING ONE IS NOT CONNECTED TO". The program then asks for the number of the cable (the right one) that it is connected to. Then the whole test is done again so that no more faults occured due to this 'new' topology.

WHICH FILE IS TO BE READ?

TRAFOS IS OK
FEEDER IS OK
BRANCHPOINTS OK
NODES OK
SWITCH OK
SWITCH OK

THE NETWORK IS CORRECT
DO YOU WISH TO HAVE A VISIBLE TRACK? Y/N

DO YOU WISH TO MAKE ANY CHANGES IN THE NETWORK?Y/N
POINT IN IS 1 1
POINT OUT FROM STEP DOWN IS 6 57

INDATA NEWDIR (POINT,DIR,BRA)
6 57 0 4

OUTDATA NEWDIR (POINT,DIR,BRA)
6 57 0 4

INDATA TKESTP (POINT,OLDP,DIR,BRA)
6 57 1 1 0 4
POINT IN IS 6 57
POINT OUT FROM STEP DOWN IS 4 1

OUTDATA TKESTP
4 1 6 57 0 4

INDATA NEWDIR (POINT,DIR,BRA)
4 1 0 4

OUTDATA NEWDIR (POINT,DIR,BRA)
4 1 0 4

INDATA TKESTP (POINT,OLDP,DIR,BRA)
4 1 6 57 0 4
POINT IN IS 4 1
POINT OUT FROM STEP DOWN IS 2 1

OUTDATA TKESTP
2 1 4 1 0 4

INDATA NEWDIR (POINT,DIR,BRA)
2 1 0 4

OUTDATA NEWDIR (POINT,DIR,BRA)
2 1 0 4

INDATA TKESTP (POINT,OLDP,DIR,BRA)
2 1 4 1 0 4
POINT IN IS 2 1
POINT OUT FROM STEP DOWN IS 6 54

OUTDATA TKESTP
6 54 2 1 0 4

INDATA NEWDIR (POINT,DIR,BRA)
6 54 0 4

OUTDATA NEWDIR (POINT,DIR,BRA)

```
INDATA NEWDIR (POINT,DIR,BRA)
     5           1           1           5

OUTDATA NEWDIR (POINT,DIR,BRA)
     5           1           1           5

INDATA TKESTP (POINT,OLDP,DIR,BRA)
     5           1           6           1           1           5
POINT IN IS           5           1
CABUP IN NODE IS           2
CABLE UP IS           3
POINT OUT FROM STEPUP IS           6           54

OUTDATA TKESTP
     6           54          5           1           1           5

INDATA NEWDIR (POINT,DIR,BRA)
     6           54          1           5

OUTDATA NEWDIR (POINT,DIR,BRA)
     6           54          1           5

INDATA TKESTP (POINT,OLDP,DIR,BRA)
     6           54          5           1           1           5
POINT IN IS           6           54
POINT OUT FROM STEPUP IS           2           1

OUTDATA TKESTP
     2           1           6           54          1           5

INDATA NEWDIR (POINT,DIR,BRA)
     2           1           1           5

OUTDATA NEWDIR (POINT,DIR,BRA)
     2           1           1           5

INDATA TKESTP (POINT,OLDP,DIR,BRA)
     2           1           6           54          1           5
POINT IN IS           2           1
POINT OUT FROM STEPUP IS           4           1

OUTDATA TKESTP
     4           1           2           1           1           4

INDATA NEWDIR (POINT,DIR,BRA)
     4           1           1           4

OUTDATA NEWDIR (POINT,DIR,BRA)
     4           1           0           5

INDATA TKESTP (POINT,OLDP,DIR,BRA)
     4           1           2           1           0           5
POINT IN IS           4           1
WRONG IN STEPDO.BRA
FORTRAN STOP
```

COMMENTS

In this example the possibility to discover cooperating faults was demonstrated.

```
   1          31         32
 ---- BP1 ---- BP8 ---- SW58
```

Figure 10. One part of the topology.

In branch point 1 (BP1) had a pointer to cable 32 been stored instead of a pointer to cable 31. This error was not detected by the first test. By demanding the program to show how it 'moves' (Answer Y to the question 'DO YOU WISH TO HAVE A VISIBLE TRACK') the last point before the program stops is shown in the screen. Then it is easy to make the proper change in the data file.

In this case the program stoped at point (4,1). That is branch point 1.

Several pages in the middle were left out.

WHICH FILE IS TO BE READ?

TRAFOS IS OK
FEEDER IS OK
BRANCHPOINTS OK
NODES OK
SWITCH OK
SWITCH OK

THE NETWORK IS CORRECT
DO YOU WISH TO HAVE A VISIBLE TRACK? Y/N

DO YOU WISH TO MAKE ANY CHANGES IN THE NETWORK?Y/N

CHANGE TOTAL LOAD? Y/N

```
CHANGE IN TYPE NR?
R IS   1009.800
X IS   408.0000
NEW R IS
NEW X IS

ON WHICH FILE DO YOU WISH TO STORE IT
INIT PASSED
          FAULT IN

             1: TRANSFORMER
             2: FEEDER (MAIN SUPPORT)
             3: CABLE
             4: BRANCHINGPOINT
             5: NODE

         WHICH TYPE OF COMPONENT?


       WHICH NODE?
TOTAL COMPUTATION TIME OF ALGORITM   14 ms

THE FOLLOWING SWITCHES HAS CHANGED STATUS

             41    4
             42    1
             48    1
             49    1
             50    4

TOTAL DEENERGIZED LOAD DUE TO FAULT
 0.21720E+06

TOTAL RESTORED LOAD
 0.21720E+06


ANOTHER FAULT? Y/N
FORTRAN STOP
```

COMMENTS

In the third example the possibility to change a single node impedance was shown. Then a fault causing no violation at all was chosen to show how fast a solution can be reached.

```
LE IS TO BE READ?


       TRAFOS IS OK
       FEEDER IS OK
       BRANCHPOINTS OK
       NODES OK
       SWITCH OK
       SWITCH OK



       THE NETWORK IS CORRECT
       DO YOU WISH TO HAVE A VISIBLE TRACK? Y/N

       DO YOU WISH TO MAKE ANY CHANGES IN THE NETWORK?Y/N

       CHANGE TOTAL LOAD? Y/N
       IMPEDANCES WILL BE MULTPLIED WITH?

       ON WHICH FILE DO YOU WISH TO STORE IT
       INIT PASSED
                FAULT IN

                   1: TRANSFORMER
                   2: FEEDER (MAIN SUPPORT)
                   3: CABLE
                   4: BRANCHINGPOINT
                   5: NODE

                WHICH TYPE OF COMPONENT?


              WHICH TRANSFORMER?
ENTERS STAGE 1           3
ENTERS STAGE************************************
READS IN STAGE1 ***************************
IMPROVEMENT,FEEDER  0.0000000E+00          6
OK=.FALSE.
ENTERS STAGE 2
STAGE2 CALLS STAGE1 ******************************
ENTERS STAGE 1           3
OK IS FALSE
BUSY READING
STAGE1 WITH OK=F HAD AN UNSUCCESSFULL COMPLETION
EXIT STAGE 2 OK= F
ENTERS STAGE 1           5
ENTERS STAGE************************************
READS IN STAGE1 ***************************
IMPROVEMENT,FEEDER   112015.5              6
IMPROVEMENT,FEEDER   13.42432              4
IMPROVEMENT,FEEDER  0.1634084              4
SAVENE******************************************
        5           6
SAVENE******************************************
CALLS RESNET*************
CALLS RESNET*************
IMPROVEMENT,FEEDER   112015.5              6
IMPROVEMENT,FEEDER   13.42432              4
IMPROVEMENT,FEEDER  0.1634084              4
SAVENE******************************************
        5           4
SAVENE******************************************
LEAVES STAGE1 AFTER A SUCCESFULL SESSION,OK=T
ENTERS STEP 3
```

```
STAGE 3 WILL BE CUTTING LOAD IN FEEDER          3
ENTERS STEP 4
ENTERS STAGE 1           3
ENTERS STAGE*************************************
READS IN STAGE1 ***************************
IMPROVEMENT,FEEDER   4902.289              4
IMPROVEMENT,FEEDER   4902.289              5
IMPROVEMENT,FEEDER  0.0000000E+00          6
SAVENE*******************************************
         3           4
SAVENE*******************************************
LEAVES STAGE1 AFTER A SUCCESFULL SESSION,OK=T
      TOTAL COMPUTATION TIME OF ALGORITM  295 ms

      THE FOLLOWING SWITCHES HAS CHANGED STATUS

                  4    3
                  8    3
                  9    3
                 11    3
                 16    4
                 37    4
                 54    1
                 57    1
                 58    1

      TOTAL DEENERGIZED LOAD DUE TO FAULT
       0.99335E+07

      TOTAL RESTORED LOAD
       0.99335E+07


      ANOTHER FAULT? Y/N
FORTRAN STOP
```

COMMENTS

In the fourth example the total load was first changed providing the right answers in the proper places (Y to all but 'DO YOU WISH TO HAVE A VISIBLE TRACK?'). Then a serious fault was chosen (minor load). A fault in transformer 1 occured. The topology of the test system is once again showed in figure 10. A closed switch is marked by 'x' and an open switch is surounded by a circle

The fault is in this case isolated by closing the circuit breakers (switch 54,57,58 or #1,f1,f2 in the figure) Isolated trees are created with roots in nodes 1 and 35. These are marked (1), (35) in figure 11.

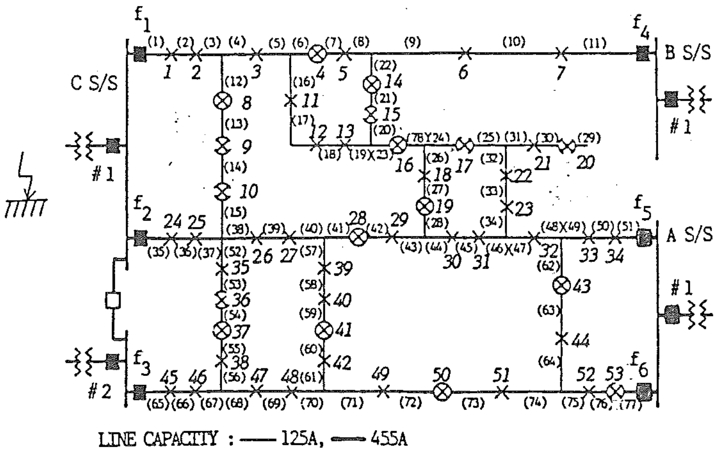


figure 11. Topology

Then for each tree all possible connections are examined.Possible switches for the tree with node 1 as the root are 4,14,16. Note that switch 8 is not a possible connection because this switch leads to another isolated section. Switch 16 is in this case the most suitable switch for reenergizing the 'tree' because feeder 5 has a bigger margin of getting violated than feeder 4. Switch 16 is then opened. The same examination is done for the tree with node 35 as root and in this case switch 37 is opened.

These two switching operations caused violations in feeders 3 and 5. First the violation in feeder 3 is tried to be decreased. The only possible load transfer is transfering node 72 from feeder 3 to feeder 6 (the switches 8,28 and 41 leads all to a violated feeder) but since this load transfer does not reduce the violation at all it is abandoned. There is a voltage drop violation in node 13. This violation would be eliminated by a load transfer of this node, but it is impossible to transfer this node since its neigbour's feeder also is violated.(The neighbour feeder is now feeder 5)

A first stage is then tried on feeder 5. The transcript shows that there are three possible switches for load transfer. These are switch nr: 43, 14, 4. Switch 43 cause the biggest reduction in the violation vector because Switch 43 is opened and switch 33 is closed but then feeder 6 would be violated. Instead the possibility to transfer load by opening switch 11 and closing switch 4 is examined. This load transfer turned out to be succesful. Feeder 5 is not violated any more.

The next step in the algorithm is the load curtailment. Feeder 3 is still violated and therefore node 13 is deenergized by opening switch 9. Then no violations at all exist and the fourth step is performed. Node 13 is reenergized by closing switch 9 and a first stage support is tried. The load transfer succeds and node 13 is transfered to feeder 4. (Node 12 is energized by feeder 4 since the previous first stage support of feeder 5) Then all loads are restored and the algorithm stops.

The switching operations that must be performed is compared with the NON-FAULTED system.So in this case the circuit breakers are alredy changed. They were tripped by the fault.

Realizing this the operator can then command the switching operations to be executed in order.

7.3 OPTIMIZATION

PROBLEM FORMULATION

The service restoration is formulated by Aoki as follows:

$$\text{Maximize} \sum_{i} \sum_{j \in J_{i0}} a_{ij} x_{ij}$$

Subject to

(line capacity constraint)

$$\sum_{j \in J_{ik}} a_{ij} x_{ij} \leqq b_{ik}$$

(transformer capacity constraint)

$$\sum_{j \in J_{t}} a_{ij} x_{ij} \leqq b_{t}$$

(voltage drop constraint)

$$\sum_{l \in T_e} \{ (\sum_{q \in J_{il}} s_{iq} x_{ij}) z_{il} \} \leqq V_{ie} ,$$

Where,

X_{ij}: such 0-1 variable as 0 if section j at feeder i is de-energized and 1 if it is energized. X_{ij} cannot arbitrarily be set to 1. $X_{ij}=1$ only if the adjacent section is energized and the sectionalizing switch connected to it is closed. X_{ij} is a function of the sectionalizing switch status.

a_{ij}: load magnitude of section j at feeder i

b_{ik}: line capacity at k-th point of feeder i

b_t : transformer capacity of transformer t

z_{il}: impedance of section l at feeder i

s_{ik}: $s_{ik} = a_{ik}$ (if $k \neq l$), $s_{ik} = a_{ik}/2$ (if $k=l$) as uniformly distributed load is assumed

v_{ie}: voltage drop limit at the e-th end of feeder i

J_{i0}: index set of de-energized load sections on feeder i

J_{ik}: index set of load sections connected to the leaf side of point k of feeder i

J_{il}: index set of load sections which exist at the leaf side of section l (included) of the feeder i

J_t : index set of load sections ij connected to transformer t

T_e : index set of load sections which exist at the trunk of the tree between bus and section e

EFFECTIVE GRADIENT METHOD

The effective gradient method is used by some economists dealing with problems such as: Which orders will not be served due to a limited resource. The effective gradient method is very simple to understand. It says that if you have to exit from a restricted area with an unlimited number of steps, but wish to stay as close to the border as possible and the distance to the permitted area is not known you should take the shortest step possible in that direction.

7.4 STRUCTURE CHART

The network is represented by the following matrixes where the first index points out the number of the component in the network. The following map shows where the data is stored for every component.

TRAFOS

P_{max}	Q_{max}	P	Q	V	$R_{network}$	$X_{network}$

MSFEED

Trafo Identity	Cable up	Cable down

CABTOP

First neighbour		Second neighbour	
Type	Number	Type	Number

CABPHY

R	X	I_{max}	I

BRAPOI

Actual nr of branches	Branch up	Branch 1	Branch 2	. . .

NODETO

Up	Cable 1	Cable 2	Feeder identity

NODEPH

P	Q	V	R	X

SWITCH

Status	Up	Cable 1	Cable 2	Allowed change

Some comments are required to understand the structure charts.

In up (cable up, branch up) the index pointing out the cable leading in the direction of the transformer is stored.

For open switches, up will be assigned a new value every time the switch will be 'visited'.

Allowed change is used to prevent a load transfer to the faulted zone.

In the program another matrix with three indexes, BRAPH is used to store the impedance one sees if one look down a branch. This matrix have matched the two first indexes with BRAPOI.

```
C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C                  RESTORE.FOR

C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC




        PROGRAM MAIN

C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C       AUTHOUR MAGNUS BERGSTRAND

C       VERSION 1

C       DATE 1989-10-07

C       PURPOSE
C       WHEN A FAULT HAS OCCURRED IN AN ELECTRICAL DISTRIBUTION NETWORK
C       THIS FAULT MUST BE ISOLATED. THE NON  FAULTED ZONES CAN OFTEN BE
C       RESTORED WITHOUT CAUSING ANY VIOLATIONS. THE PROGRAM SUGGEST SUCH
C       A POSSIBLE RESTORATION

C       METHOD
C       THE METHOD IS DISCRIBED IN 'SERVICE RESTORATION IN ELECTRIC
C       DISTRIBUTION NETWORKS' BY MAGNUS BERGSTRAND
C       THE METHOD WAS FIRST DISCRIBED BY AOKI IN
C       'A NEW ALGORITHM FOR SERVICE RESTORATION IN DISTRIBUTION SYSTEMS'
C       IEEE/PES WINTER MEETING, NEW YORK 1989 PAPER 89 WM 085-2 PWRD

C       DESCRIPTION
C       FIRST IS THE FAULTED ZONE ISOLATED AND THE NON FAULTED ZONES
C       CONNECTED TO AN ADJACENT FEEDER. IF THIS CONNECTION DOES NOT
C       CAUSE ANY VIOLATION THE ALGORITHM STOPS AND A SOLUTION IS FOUND.
C       IF VIOLATIONS OCCURED THESE ARE TRIED TO BE DECREASED IN STEP2.
C       STEP2 HAS TWO PRINCIPAL PARTS THE FIRST AND SECOND STAGE SUPPORT.
C       THE FIRST STAGE SUPPORT TRANSFERS LOAD TO AN ADJACENT FEEDER FROM
C       A VIOLATED FEEDER NOT ALLOWING ANY NEW VIOLATIONS.
C       THE SECOND STAGE SUPPORT TRANSFERS FIRST LOAD TO THE PRIMARY SUPPORT
C       FEEDER ALLOWING VIOLATIONS AND THEN TRANSFERS LOAD FROM THE PRIMARY
C       SUPPORT FEEDER TO A SECONDARY SUPPORT FEEDER NOT ALLOWING VIOLATIONS.
C       IF VIOLATIONS STILL EXIST THESE ARE REMOVED BY THE THIRD STEP (THE
C       LOAD CURTAILMENT)
C       AFTER THE LOAD CURTAILMENT THERE IS A POSSIBILITY TO RESTORE LOAD
C       AMONG FORMER VIOLATION FEEDERS AND THIS IS DONE IN THE FOURTH STEP.

C       SEE ALSO
C       FILES   ROUTINES   COMMENTS
C       CONSTANTS          ALL CONSTANTS DECLARED
C       COMMON             ALL COMMON VARIABLES DECLARED
C       OUTFI   INFI       READS THE NETWORK TOPOLOGY AND PHYSICAL DATA FROM
C                          DATA FILE
C       OUTFI   WRITEF     SAVES THE NETWORK TOPOLOGY AND PHYSICAL DATA ON
C                          A DATA FILE
C       CHG     INC        CHANGES TOTAL LOAD
C       CHG     CHANGE     CHANGES PHYSICAL DATA IN A SINGLE ELEMENT
C       TESTS   NETTES     TESTS THE TOPOLOGY OF THE NETWORK
C       TESTS   INIT       INITIATES THE NETWORK
C       COMP    FILPOW     COMPUTES POWER
C       COMP    COMPIM     COMPUTES IMPEDANCE
```

```
C       COMP    FILVOL     COMPUTES AND FILLS IN NODE VOLTAGES AND CABLE CURRENTS
C       FAULT   FLTID      READS THE FAULTED SECTION
C       RESTORE RESULT     PRODUCES THE RESULT MENUE
C       RESTORE GARBAG     'GARBAGE COLLECTOR' CLEANS UP AMONG SWITCHING
C                          OPERATIONS

C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        INCLUDE 'CONSTANTS.FOR'
        INCLUDE 'COMMON.FOR'

        INTEGER FAULT(2)
        INTEGER COU,PA
        CHARACTER*7 CH2
        LOGICAL VIO,READY,WR
        INTEGER TIME,MSCPU,I,A
        CHARACTER DATA

        EXTERNAL MSCPU

 20     WRITE(*,*)'        WHICH FILE IS TO BE READ?'
        READ(*,300)CH2
        CALL INFI(CH2)
        CALL NETTES(CH2)
 10     WRITE(*,*)'        DO YOU WISH TO HAVE A VISIBLE TRACK? Y/N'
        READ(*,300)DATA
        IF (DATA.EQ.'Y') THEN
          WR=.TRUE.
        ELSEIF (DATA.EQ.'N') THEN
          WR=.FALSE.
        ELSE
          GOTO 10
        ENDIF
        READY=.FALSE.
 40     IF (.NOT. READY) THEN
          WRITE(*,100)'DO YOU WISH TO MAKE ANY CHANGES IN THE NETWORK?Y/N'
          READ(*,300)DATA
          IF (DATA.EQ.'Y' .OR. DATA.EQ.'y') THEN
            WRITE(*,100)'CHANGE TOTAL LOAD? Y/N'
            READ(*,300)DATA
            IF (DATA.EQ.'Y' .OR. DATA.EQ.'y') THEN
              CALL INC
            ELSE
              CALL CHANGE
            ENDIF
            WRITE(*,100)'ON WHICH FILE DO YOU WISH TO STORE IT'
            READ(*,300) CH2
            CALL WRITEF(CH2)
          ELSEIF (DATA.NE.'N') THEN
            GOTO 40
          ELSE
            READY=.TRUE.
            GOTO 40
          ENDIF
        ENDIF
        CALL INIT(WR)
        DO 50 COU=1,MAXNRT
          CALL FILPOW(COU)
 50     CONTINUE
        DO 60 COU=1,MAXNRF
          CALL FILVOL(COU,VIO)
 60     CONTINUE

C       THE NETWORK IS NOW INITIATED
C       IDENTIFY FAULT AND START CLOCK
```

```
      CALL FLTID(FAULT)
      TIME=MSCPU()

C     RUN ALGORITHM

      CALL STEP1(FAULT)
      CALL STEP2(VIO)
      IF (VIO) THEN
        CALL STEP3
      ENDIF

C     STOP CLOCK

      TIME=MSCPU()-TIME

C     CLEAN UP AMONG SWITCHING OPERATIONS AND PRESENT RESULT

      CALL GARBAG
      CALL RESULT(TIME)
      WRITE(*,100)'ANOTHER FAULT? Y/N'
      READ(*,300) DATA
      READY=DATA.EQ.'N' .OR. DATA.EQ.'n'
      IF (.NOT. READY) THEN
        GOTO 20
      ENDIF
 100  FORMAT('0',TR6,A)
 200  FORMAT(' ',T10,A)
 300  FORMAT(' ',T1,A)
      STOP
      END



      SUBROUTINE RESULT(TIME)

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     AUTHOUR MAGNUS BERGSTRAND

C     VERSION 1

C     DATE 1989-10-07

C     PURPOSE
C     PRESENTS THE RESULT

C     VARIABLES
C     TIME IS EXECUTION TIME IN ms

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      INCLUDE 'CONSTANTS.FOR'
      INCLUDE 'COMMON.FOR'

      INTEGER TIME,I
      REAL SQRT

      INTRINSIC SQRT

      WRITE(*,100)'TOTAL COMPUTATION TIME OF ALGORITM',TIME,' ms'
      WRITE(*,*)
      WRITE(*,200)'THE FOLLOWING SWITCHES HAS CHANGED STATUS'
      WRITE(*,*)
      DO 10 I=1,MAXNRS
        IF (CHGSWI(I)) THEN
          WRITE(*,300) I,SWINR(I)
```

```
      ENDIF
 10   CONTINUE
      WRITE(*,*)
      WRITE(*,200)'TOTAL DEENERGIZED LOAD DUE TO FAULT'
      WRITE(*,500) SQRT(ISOLAT(1)**2+ISOLAT(2)**2)
      WRITE(*,*)
      WRITE(*,200)'TOTAL RESTORED LOAD'
      WRITE(*,500) SQRT(RESTOR(1)**2+RESTOR(2)**2)
      WRITE(*,*)
 100  FORMAT(' ',TR6,A,I5,A)
 200  FORMAT(' ',TR6,A)
 300  FORMAT(' ',TR15,I5,TR3,I2)
 500  FORMAT(' ',TR6,E12.5)
      RETURN
      END


      SUBROUTINE GARBAG

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     AUTHOUR MAGNUS BERGSTRAND

C     VERSION 1

C     DATE 1989-10-07

C     PURPOSE
C     DECREASE THE NUMBER OF NECESSARY SWITCHING OPERATIONS

C     DESCRIPTION
C     WHEN FOR INSTANCE TWO SECTIONS HAS BEEN DEENERGIZED IN STEP3
C     TWO SWITCHES HAVE CHANGED STATUS. THE SAME RESULT COULD MAYBE
C     BEEN GIVEN IF ONLY THE SWITCH NEAREST THE FEEDER CHANGED STATUS
C     ALL SWITCHES ARE EXAMINED
C     WHEN AN OPEN SWITCH IS FOUND THE SUBROUTINE 'MOVES' TOWARDS
C     THE FEEDER.
C     IF AN OPEN SWITCH IS FOUND ON THE WAY IT IS POSSIBLE TO REDUCE THE NUMBER
C     OF SWITCHING OPERATIONS WITH 1.
C     THE SWITCHING OPERATION PERFORMED ON THE STARTING SWITCH WAS UNNECESSARY

C     SEE ALSO
C     FILES   ROUTINES   COMMENTS
C     STEP2   STEP3      THE LOAD CURTAILMENT
C     MOVING  STEPUP     TAKES ONE STEP UP

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      INCLUDE 'CONSTANTS.FOR'
      INCLUDE 'COMMON.FOR'

      INTEGER P(2)
      INTEGER I
      LOGICAL READY

      DO 10 I=1,MAXNRS
        IF (SWITCH(I,1).EQ.OPEN .AND. CHGSWI(I)) THEN
          P(1)=SWITYP
          P(2)=I
          READY=.FALSE.
 20       IF (.NOT. READY) THEN
            CALL STEPUP(P,.FALSE.)
            READY=(P(1).EQ.SWITYP .OR. P(1).EQ.FEETYP)
            READY=(READY .OR. P(1).EQ.TRATYP)
            GOTO 20
          ENDIF
```

```
          IF (P(1).EQ.SWITYP) THEN
            IF (SWITCH(P(2),1).EQ.OPEN) THEN
              CHGSWI(I)=.NOT. CHGSWI(I)
              SWITCH(I,1)=CLOSED
            ENDIF
          ENDIF
        ENDIF
10    CONTINUE
      RETURN
      END
```

```
C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                CONSTANTS.FOR
C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C       IN THIS FILE ALL CONSTANTS USED ARE DECLARED
C
C       READ MAXNRx AS MAX NUMBER OF x
C
C       BRANCH IS THE MAXIMUM NUMBER OF BRANCHES IN ANY BRANCH POINT IN THE SYSTE
C

        PARAMETER (MAXNRT=4,MAXNRF=6,MAXNRN=78,MAXNRB=18)
        PARAMETER (MAXNRC=177,BRANCH=4,MAXNRS=64)
        PARAMETER (TRATYP=1,FEETYP=2,CABTYP=3)
        PARAMETER (BRATYP=4,NODTYP=5,SWITYP=6)
        PARAMETER (OPEN=0,CLOSED=1,UP=1,DOWN=0)
        PARAMETER (NET=10)
        PARAMETER (BETA=0.00001)
```

```
C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C           COMMON.FOR
C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     AUTHOUR MAGNUS BERGSTRAND
C     VERSION 1
C     DATE 1989-09-30
C     READ ALSO 'SERVICE RESTORATION IN ELECTRICAL DISTRIBUTION NETWORKS'
C     BY MAGNUS BERGSTRAND (APPENDIX SECTION STRUCTURE CHART)
C     DESCRIPTION:
C     IN THIS FILE ALL COOMON VARIABLES IS DECLARED.
C     BELOW FOLLOWS A SHORT DESCRIPTION OF EACH VARIABLE.
C     ALFA IS THE NODE PRIORITY
C
C     IN 'SWINR' THE ORDER OF THE SWITCHING OPERATIONS IS STORED
C     IN ISOLAT AND RESTORE ARE THE AMOUNT OF ISOLATED LOAD AND RESTORED LOAD
C     STORED (W,VAr)
C
C     'TRAFOS' CONTAINS ALL INFORMATION ABOUT THE DIFFERENT TRANSFORMERS IN THE
C     NETWORK EXCEPT THE CABLE TO THE NETWORK WHICH IS STORED IN 'TRANEI'
C
C     MSFEED CONTAINS ALL DATA FOR THE FEEDERS
C
C     CABTOP CONTAINS TOPOLOGY FOR CABLES
C
C     CABPHY=PHYSICAL DATA FOR CABLES. CURRENTS IN AMPERE AND IMPEDANCES
C     IN OHM.
C
C     BRAPOI CONTAINS THE TOPOLOGY FOR BRANCH POINTS
C
C     NODETO CONTAINS THE TOPOLOGY FOR THE NODES
C
C     NODEPH CONTAINS THE PHYSICAL DATA FOR NODES
C     VOLTAGE IN VOLT,IMPEDANCE IN OHM AND POWER IN W
C
C     SWITCH CONTAINS ALL USEFUL INFORMATION ABOUT THE SWITCHES
C
C     IN BRAPH THE IMPEDANCE SEEN LOOKING DOWN A BRANCH IS STORED (OHM)
C
C     A CHART OF WHERE THE DATA IS STORED IN THE FOLLOWING MATRICES:
C     TRAFOS,MSFEED,CABTOP,CABPHY,BRAPOI,NODETO,NODEPH,SWITCH IS FOUND
C     IN THE REPORT 'SERVICE RESTORATION IN ELECTRIC DISTRIBUTION NETWORKS'
C
C     TRAMAX IS THE MAXIMUM VOLTAGE FOR ANY TRANSFORMER IN THE NETWORK
C
C     VIOVEC IS THE VIOLATION VECTOR.
C     THE FIRST INDEX INDICATES THE (VIOLATED) FEEDER
C
C     THE LOGICAL VECTOR CHGSWI INDICATES IF A SWITCH HAS CHANGED STATUS
C     COMPARED WITH THE INITIAL STATE

      IMPLICIT LOGICAL (A-Z)
      REAL ALFA(MAXNRN)
      INTEGER SWINR(MAXNRS)
      REAL ISOLAT(2),RESTOR(2)
      REAL TRAFOS(1:MAXNRT,1:7),TRAMAX
      INTEGER TRANEI(1:MAXNRT)
```

```
INTEGER MSFEED(1:MAXNRF,1:3)
INTEGER CABTOP(1:MAXNRC,1:4)
REAL CABPHY(1:MAXNRC,1:4)
INTEGER BRAPOI(1:MAXNRB,1:BRANCH+3)
INTEGER NODETO(1:MAXNRN,1:4)
REAL NODEPH(1:MAXNRN,1:5),VOLLIM
INTEGER SWITCH(1:MAXNRS,1:5)
REAL BRAPH(1:MAXNRB,3:BRANCH+2,1:2)
REAL VIOVEC(MAXNRF,2)
LOGICAL CHGSWI(MAXNRS)
COMMON TRAFOS,CABPHY,NODEPH
COMMON TRANEI,MSFEED,CABTOP
COMMON BRAPOI,NODETO,SWITCH
COMMON TRAMAX,VOLLIM,RESTOR
COMMON BRAPH,VIOVEC,ISOLAT
COMMON CHGSWI,SWINR,ALFA
```

```
C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C                 CHG.FOR

C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC




        SUBROUTINE CHANGE

C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C       AUTHOUR MAGNUS BERGSTRAND

C       VERSION 1

C       DATE 1989-09-30

C       PURPOSE:
C       THIS PROCEDURE HANDLES THE CHANGE OF PHYSICAL
C       DATA IN A SINGLE ELEMENT

C       SEE ALSO:

C       FILE            ROUTINE          COMMENT
C       CONSTANTS                        ALL CONSTANTS ARE DECLARED
C       COMMON                           ALL COMMON VARIABLES ARE DECLARED
C       CHG             INC              CHANGES TOTAL LOAD

C       VARIABLES:

C       VOLTAGE IN VOLT
C       CURRENT IN AMPERE
C       IMPEDANCE IN OHM

C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        INCLUDE 'CONSTANTS.FOR'
        INCLUDE 'COMMON.FOR'

        EXTERNAL WRITEF

        INTEGER P(2),I

        WRITE(*,100)
        WRITE(*,*) '        CHANGE IN (TYPE , NR) ?'
        READ(*,*) P(1),P(2)

C       THE POINT WHICH WILL BE CHANGED WAS READ

 10     IF (P(1).EQ.TRATYP) THEN
          WRITE(*,*)'        NEW PMAX IS'
          READ(*,*) TRAFOS(P(2),1)
          WRITE(*,*)'        NEW QMAX IS'
          READ(*,*) TRAFOS(P(2),2)
        ELSEIF (P(1).EQ.CABTYP) THEN
          WRITE(*,*)'        NEW R IS'
          READ(*,*) CABPHY(P(2),1)
          WRITE(*,*)'        NEW X IS'
          READ(*,*) CABPHY(P(2),2)
          WRITE(*,*)'        NEW IMAX IS'
          READ(*,*) CABPHY(P(2),3)
        ELSEIF (P(1).EQ.NODTYP) THEN
          WRITE(*,*)'        R IS',NODEPH(P(2),4)
```

```
        WRITE(*,*)'       X IS',NODEPH(P(2),5)
        WRITE(*,*)'       NEW R IS'
        READ(*,*) NODEPH(P(2),4)
        WRITE(*,*)'       NEW X IS'
        READ(*,*) NODEPH(P(2),5)
      ELSE
        GOTO 10
      ENDIF
 100  FORMAT('1')
      RETURN
      END

      SUBROUTINE INC

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     AUTHOUR MAGNUS BERGSTRAND

C     VERSION 1

C     DATE 1989-09-30

C     PURPOSE:
C     READS A FACTOR P WHICH ALL THE NODE IMPEDANCES
C     IN THE NETWORK WILL BE MULTPLIED WITH CAUSING
C     A CHANGE IN THE NETWORK LOAD

C     SEE ALSO:

C     FILE           ROUTINE          COMMENT
C     CONSTANTS                       ALL CONSTANTS ARE DECLARED
C     COMMON                          ALL COMMON VARIABLES ARE DECLARED
C     CHG            CHANGE           CHANGES PHYSICAL DATA OF ANY
C                                     COMPONENT IN THE NETWORK

C     VARIABLES:

C     VOLTAGE IN VOLT
C     CURRENT IN AMPERE
C     IMPEDANCE IN OHM

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      INCLUDE 'CONSTANTS.FOR'
      INCLUDE 'COMMON.FOR'

      REAL P
      INTEGER I,J

      WRITE(*,*)'      IMPEDANCES WILL BE MULTPLIED WITH?'
      READ(*,*) P
      DO 10 I=1,MAXNRN
        NODEPH(I,4)=NODEPH(I,4)*P
        NODEPH(I,5)=NODEPH(I,5)*P
 10   CONTINUE
      RETURN
      END
```

```
C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C             FAULT.FOR

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC



      SUBROUTINE FLTID(FAULT)

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     AUTHOUR MAGNUS BERGSTRAND

C     VERSION 1

C     DATE 1989-10-07

C     PURPOSE
C     READS THE FAULTED POINT

C     SEE ALSO

C     FILES     ROUTINES   COMMENTS
C     RESTORE   MAIN       MAIN PROGRAM

C     VARIABLES
C     FAULT (OUT) IS THE FAULTED POINT (TYPE,NR)

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      INTEGER FAULT(1:2)

 10   WRITE(*,*) '                    FAULT IN'
      WRITE(*,*)
      WRITE(*,*) '                       1: TRANSFORMER'
      WRITE(*,*) '                       2: FEEDER (MAIN SUPPORT)'
      WRITE(*,*) '                       3: CABLE'
      WRITE(*,*) '                       4: BRANCHINGPOINT'
      WRITE(*,*) '                       5: NODE'
      WRITE(*,*)
      WRITE(*,*) '                    WHICH TYPE OF COMPONENT?'
      READ(*,*) FAULT(1)
      IF ((FAULT(1).LE.0).OR.(FAULT(1).GE.6)) GOTO 10
        WRITE(*,*)
        WRITE(*,*)
      IF (FAULT(1).EQ.1) THEN
        WRITE(*,*) '                  WHICH TRANSFORMER?'
      ELSEIF (FAULT(1).EQ.2) THEN
        WRITE(*,*) '                  WHICH FEEDER?'
      ELSEIF (FAULT(1).EQ.3) THEN
        WRITE(*,*) '                  WHICH CABLE?'
      ELSEIF (FAULT(1).EQ.4) THEN
        WRITE(*,*) '                  WHICH BRANCHINGPOINT?'
      ELSEIF (FAULT(1).EQ.5) THEN
        WRITE(*,*) '                  WHICH NODE?'
      ENDIF
      READ(*,*) FAULT(2)
      RETURN
      END
```

```
C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C                 OUTFI.FOR

C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC



        SUBROUTINE WRITEF(N)

C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C       AUTHOUR MAGNUS BERGSTRAND

C       VERSION 1

C       DATE 1989-10-08

C       PURPOSE
C       SAVES THE NETWORK TOPOLOGY AND PHYSICAL DATA ON A FILE

C       DESCRIPTION
C       USES FORMATTED OUTPUT
C       READ ALSO USER'S GUIDE IN 'SERVICE RESRORATION IN ELECTRICAL
C       DISTRIBUTION NETWORKS' BY MAGNUS BERGSTRAND WHERE EXPLANATIONS AND
C       EXAMPLES ARE FOUND.

C       SEE ALSO
C       FILES    ROUTINES    COMMENTS
C       OUTFI    INFI        READS A FILE CONTAINING THE NETWORK TOPOLOGY
C                            AND PHYSICAL DATA FOR A NETWORK USING THE SAME
C                            FORMAT

C       VARIABLES
C       THE STRING N CONTAINS THE NAME OF THE FILE

C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        INCLUDE 'CONSTANTS.FOR'
        INCLUDE 'COMMON.FOR'

        INTEGER I,J
        CHARACTER*7 N

        OPEN(1,FILE=N,STATUS='NEW',ACCESS='SEQUENTIAL',FORM='FORMATTED')
        WRITE(1,600) MAXNRT
        WRITE(1,200) TRAMAX
        DO 10 I=1,MAXNRT
          WRITE(1,300) TRAPOS(I,1),TRAPOS(I,2),TRANEI(I)
 10     CONTINUE
        WRITE(1,600) MAXNRF
        DO 20 I=1,MAXNRF
          WRITE(1,400)MSFEED(I,2),MSFEED(I,3)
 20     CONTINUE
        WRITE(1,600) MAXNRC
        DO 40 I=1,MAXNRC
          DO 50 J=1,3
            WRITE(1,200) CABPHY(I,J)
 50       CONTINUE
          DO 60 J=1,4
            WRITE(1,600) CABTOP(I,J)
 60       CONTINUE
 40     CONTINUE
        WRITE(1,600) MAXNRB
        DO 70 I=1,MAXNRB
          WRITE(1,600) BRAPOI(I,1)
```

```
      DO 80 J=3,(BRAPOI(I,1)+2)
        WRITE(1,600) BRAPOI(I,J)
 80     CONTINUE
 70   CONTINUE
      WRITE(1,600) MAXNRN
      WRITE(1,200) VOLLIM
      DO 90 I=1,MAXNRN
        WRITE(1,400) NODETO(I,2),NODETO(I,3)
        WRITE(1,500) NODEPH(I,4),NODEPH(I,5)
        WRITE(1,200) ALFA(I)
 90   CONTINUE
      WRITE(1,600) MAXNRS
      DO 100 I=1,MAXNRS
        WRITE(1,400) SWITCH(I,3),SWITCH(I,4)
 100  CONTINUE
      DO 110 I=1,MAXNRS
        WRITE(1,600) SWITCH(I,1)
 110  CONTINUE
      CLOSE(1,STATUS='KEEP')
 200  FORMAT(' ',TR2,E12.5)
 300  FORMAT(' ',TR2,2(E12.5,TR2),I6)
 400  FORMAT(' ',2(TR2,I6))
 500  FORMAT(' ',2(TR2,E12.5))
 600  FORMAT(' ',TR2,I6)
      RETURN
      END



      SUBROUTINE INFI(N)
C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     AUTHOUR MAGNUS BERGSTRAND

C     VERSION 1

C     DATE 1989-10-08

C     PURPOSE
C     READS A FILE CONTAINING THE NETWORK TOPOLOGY AND PHYSICAL DATA
C     FOR A NETWORK.

C     DESCRIPTION
C     FORMATTED INPUT IS USED.
C     READ ALSO USER'S GUIDE IN 'SERVICE RESRORATION IN ELECTRICAL
C     DISTRIBUTION NETWORKS' BY MAGNUS BERGSTRAND WHERE EXPLANATIONS AND
C     EXAMPLES ARE FOUND.

C     SEE ALSO
C     FILES    ROUTINES    COMMENTS
C     OUTFI    WRITEF      SAVES A NETWORK ON A FILE USING THE SAME FORMAT

C     VARIABLES
C     THE STRING N IS THE NAME OF THE FILE

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      INCLUDE 'CONSTANTS.FOR'
      INCLUDE 'COMMON.FOR'

      INTEGER I,J,K
      CHARACTER*7 N

      OPEN(1,FILE=N,STATUS='OLD',ACCESS='SEQUENTIAL',FORM='FORMATTED')
      REWIND 1
```

```
      READ(1,600) I
      IF (I.EQ.MAXNRT) THEN
         READ(1,200) TRAMAX
         DO 10 J=1,I
            READ(1,300)TRAFOS(J,1),TRAFOS(J,2),TRANEI(J)
10       CONTINUE
      ELSE
         WRITE(*,*)'        WRONG NR OF TRAFOS',I,MAXNRT
         STOP
      ENDIF
      READ(1,*) I
      IF (I.EQ.MAXNRF) THEN
         DO 20 J=1,I
            READ(1,400) MSFEED(J,2),MSFEED(J,3)
20       CONTINUE
      ELSE
         WRITE(*,*)'        WRONG NR OF FEEDERS',I,MAXNRF
         STOP
      ENDIF
      READ(1,600) I
      IF (I.EQ.MAXNRC) THEN
         DO 30 J=1,I
            DO 40 K=1,3
               READ(1,200) CABPHY(J,K)
40          CONTINUE
            DO 50 K=1,4
               READ(1,600) CABTOP(J,K)
50          CONTINUE
30       CONTINUE
      ELSE
         WRITE(*,*)'        WRONG NR OF CABLES',I,MAXNRC
         STOP
      ENDIF
      READ(1,600) I
      IF (I.EQ.MAXNRB) THEN
         DO 60 J=1,I
            READ(1,600) BRAPOI(J,1)
            DO 70 K=3,(2+BRAPOI(J,1))
               READ(1,600) BRAPOI(J,K)
70          CONTINUE
60       CONTINUE
      ELSE
         WRITE(*,*)'        WRONG NR BRANCHPOINTS',I,MAXNRB
         STOP
      ENDIF
      READ(1,600) I
      IF (I.EQ.MAXNRN) THEN
         READ(1,200) VOLLIM
         DO 80 J=1,I
            READ(1,400) NODETO(J,2),NODETO(J,3)
            READ(1,500) NODEPH(J,4),NODEPH(J,5)
            READ(1,200)ALFA(J)
80       CONTINUE
      ELSE
         WRITE(*,*)'        WRONG NR OF NODES',I,MAXNRN
         STOP
      ENDIF
      READ(1,600) I
      IF (I.EQ.MAXNRS) THEN
         DO 90 J=1,I
            READ(1,400) SWITCH(J,3),SWITCH(J,4)
90       CONTINUE
         DO 100 J=1,I
            READ(1,600) SWITCH(J,1)
100      CONTINUE
      ELSE
```

```
         WRITE(*,*)'WRONG NR OF SWITCHES',I,MAXNRS
         STOP
       ENDIF
  200  FORMAT(' ',TR2,E12.5)
  300  FORMAT(' ',TR2,2(E12.5,TR2),I6)
  400  FORMAT(' ',2(TR2,I6))
  500  FORMAT(' ',2(TR2,E12.5))
  600  FORMAT(' ',TR2,I6)
       CLOSE(1,STATUS='KEEP')
       RETURN
       END



       SUBROUTINE LINES(NR)

C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C      AUTHOUR MAGNUS BERGSTRAND

C      VERSION 1

C      DATE 1989-10-08

C      PURPOSE
C      PRODUCES BLANK ROWS USED IN LAYOUT

C      VARIABLES
C      NR IS THE NUMBER OF BLANK ROWS

C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

       INTEGER NR,I
       DO 10 I=1,NR
         WRITE(*,*)
   10  CONTINUE
       RETURN
       END



       SUBROUTINE TYPTAB

C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C      AUTHOUR MAGNUS BERGSTRAND

C      VERSION 1

C      DATE 1989-10-08

C      PURPOSE
C      PRODUCES A MENUE ON THE SCREEN

C      SEE ALSO
C      FILES     ROUTINES     COMMENTS
C      TESTS     NOTEST       THIS ROUTINE FOR INSTANCE CALLS TYPTAB

C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

       WRITE(*,*)
       WRITE(*,*) '        TYPES IN THE NETWORK'
       WRITE(*,*)
       WRITE(*,*) '          TRAFO          :1'
       WRITE(*,*) '          FEEDER         :2'
       WRITE(*,*) '          CABLE          :3'
```

```
      WRITE(*,*) '          BRANCHPOINT  :4'
      WRITE(*,*) '          NODE         :5'
      WRITE(*,*) '          SWITCH       :6'
      WRITE(*,*)
      WRITE(*,*) '     CABLE CONNECT TO ...?'
      RETURN
      END
```

```
C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                  FETCH.FOR
C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE GETPOI(SWI,P)
C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C     AUTHOUR MAGNUS BERGSTRAND
C     VERSION 1
C     DATE 1989-10-07
C     PURPOSE
C     FETCHES THE POINT (TYPE,NR) ON THE OTHER SIDE OF AN OPEN SWITCH
C     SEE ALSO
C     FILES     ROUTINES  COMMENTS
C     FETCH     GETVOL    GETS THE VOLTAGE ON THE OTHER SIDE OF AN OPEN SWITCH
C     FETCH     GETFEE    GETS THE FEEDER IDENTITY ON THE OTHER SIDE OF AN OPEN
C                         SWITCH
C     FETCH     REALFE    GETS THE FEEDER IDENTITY ON THE OTHER SIDE OF AN OPEN
C                         SWITCH AND TESTS THAT IT IS AN UNBROKEN CONNECTION
C     FETCH     GETPLI    COMPUTES THE MARGIN OF A TRANSFORMER GETTING VIOLATED
C                         (REAL POWER)
C     FETCH     GETQLI    COMPUTES THE MARGIN OF A TRANSFORMER GETTING VIOLATED
C                         (REACTIVE POWER)
C     CONSTANTS           ALL CONSTANTS DECLARED
C     COMMON              ALL COMMON VARIABLES DECLARED
C     VARIABLES
C     SWI IS THE NUMBER OF THE OPEN SWITCH
C     P (OUT) IS THE POINT (TYPE,NR)
C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      INCLUDE 'CONSTANTS.FOR'
      INCLUDE 'COMMON.FOR'
      INTEGER SWI,P(2)
      IF (SWITCH(SWI,2).EQ.3) THEN
        I=SWITCH(SWI,4)
      ELSE
        I=SWITCH(SWI,3)
      ENDIF
      IF (CABTOP(I,1).EQ.SWITYP .AND. CABTOP(I,2).EQ.SWI) THEN
        P(1)=CABTOP(I,3)
        P(2)=CABTOP(I,4)
      ELSE
        P(1)=CABTOP(I,1)
        P(2)=CABTOP(I,2)
      ENDIF
      RETURN
      END

      REAL FUNCTION GETVOL(SWI)
C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C     AUTHOUR MAGNUS BERGSTRAND
```

```
C     VERSION 1

C     DATE 1989-10-07

C     PURPOSE
C     GETS THE VOLTAGE ON THE OTHER SIDE OF AN OPEN SWITCH

C     DESCRIPTION
C     FIRST GETS THE POINT AND THEN THE VOLTAGE

C     SEE ALSO
C     FILES     ROUTINES  COMMENTS
C     CONSTANTS           ALL CONSTANTS DECLARED
C     COMMON              ALL COMMON VARIABLES DECLARED
C     FETCH     GETPOI    GETS THE POINT ON THE OTHER SIDE OF AN OPEN SWITCH
C     FETCH     GETFEE    GETS THE FEEDER IDENTITY ON THE OTHER SIDE OF AN OPEN
C                         SWITCH
C     FETCH     REALFE    GETS THE FEEDER IDENTITY ON THE OTHER SIDE OF AN OPEN
C                         SWITCH AND TESTS THAT IT IS AN UNBROKEN CONNECTION
C     FETCH     GETPLI    COMPUTES THE MARGIN OF A TRANSFORMER GETTING VIOLATED
C                         (REAL POWER)
C     FETCH     GETQLI    COMPUTES THE MARGIN OF A TRANSFORMER GETTING VIOLATED
C                         (REACTIVE POWER)

C     VARIABLES
C     SWI IS THE NUMBER OF THE OPEN SWITCH

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      INCLUDE 'CONSTANTS.FOR'
      INCLUDE 'COMMON.FOR'

      INTEGER P(2)
      INTEGER SWI,I
      REAL VOL

      IF (SWITCH(SWI,1).EQ.CLOSED) THEN
        WRITE(*,*)'       WRONG IN GETVOL'
        STOP
      ELSE
        CALL GETPOI(SWI,P)
        VOL=0.0
 10     IF (P(1).EQ.NODTYP) THEN
          GETVOL=VOL+NODEPH(P(2),3)-VOLLIM
        ELSEIF (P(1).EQ.BRATYP) THEN
          I=BRAPOI(P(2),2)
          I=BRAPOI(P(2),1)
          VOL=VOL-CABPHY(I,4)*SQRT(CABPHY(I,1)**2+CABPHY(I,2)**2)
          CALL STEPUP(P,.FALSE.)
          GOTO 10
        ELSEIF (P(1).EQ.FEETYP) THEN
          GETVOL=TRAMAX
        ELSE
          WRITE(*,*)'       WRONG IN GETVOL'
          GETVOL=0.0
          STOP
        ENDIF
      ENDIF
      RETURN
      END


      INTEGER FUNCTION GETFEE(SWI)

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

```
C       AUTHOUR MAGNUS BERGSTRAND

C       VERSION 1

C       DATE 1989-10-07

C       PURPOSE
C       GETS THE FEEDER IDENTITY ON THE OTHER SIDE OF AN OPEN SWITCH

C       SEE ALSO
C       FILES     ROUTINES  COMMENTS
C       FETCH     GETPOI    GETS THE POINT ON THE OTHER SIDE OF AN OPEN SWITCH
C       FETCH     GETVOL    GETS THE VOLTAGE ON THE OTHER SIDE OF AN OPEN SWITCH
C       FETCH     REALFE    GETS THE FEEDER IDENTITY ON THE OTHER SIDE OF AN OPEN
C                           SWITCH AND TESTS THAT IT IS AN UNBROKEN CONNECTION
C       FETCH     GETPLI    COMPUTES THE MARGIN OF A TRANSFORMER GETTING VIOLATED
C                           (REAL POWER)
C       FETCH     GETQLI    COMPUTES THE MARGIN OF A TRANSFORMER GETTING VIOLATED
C                           (REACTIVE POWER)
C       MOVING    STEPUP    TAKES ONE STEP UP
C       CONSTANTS           ALL CONSTANTS DECLARED
C       COMMON              ALL COMMON VARIABLES DECLARED

C       VARIABLES
C       SWI IS THE NUMBER OF THE OPEN SWITCH

C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        INCLUDE 'CONSTANTS.FOR'
        INCLUDE 'COMMON.FOR'

        INTEGER SWI,P(2)

        IF (SWITCH(SWI,1).EQ.CLOSED) THEN
          WRITE(*,*)'       WRONG IN GETFFE'
          STOP
        ELSE
          CALL GETPOI(SWI,P)
 10       IF (P(1).EQ.NODTYP) THEN
            GETFEE=NODETO(P(2),4)
          ELSEIF (P(1).EQ.BRATYP) THEN
            CALL STEPUP(P,.FALSE.)
            GOTO 10
          ELSEIF (P(1).EQ.FEETYP) THEN
            GETFEE=P(2)
          ELSEIF (P(1).EQ.SWITYP) THEN
            CALL STEPUP(P,.FALSE.)
            GOTO 10
          ELSE
            WRITE(*,*)'       DEGENERATED NET OR WRONG USE OF GETFEE'
            STOP
          ENDIF
        ENDIF
        RETURN
        END


        SUBROUTINE REALFE(SWI,FEE,FOUND)

C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C       AUTHOUR MAGNUS BERGSTRAND

C       VERSION 1
```

```
C     DATE 1989-10-07

C     PURPOSE
C     GETS THE FEEDER IDENTITY ON THE OTHER SIDE OF AN OPEN SWITCH
C     AND TESTS IF IT IS AN UNBROKEN CONNECTION

C     DESCRIPTION
C     FIRST A CALL OF GETFEE IS MADE
C     THEN STARTING FROM THE POINT ON THE OTHER SIDE OF THE SWITCH
C     IT MOVES UPWARDS UNTIL A FEEDER OR AN OPEN SWITCH IS FOUND.
C     IF A FEEDER IS FOUND THEN 'FOUND'=TRUE

C     SEE ALSO
C     FILES     ROUTINES  COMMENTS
C     CONSTANTS           ALL CONSTANTS DECLARED
C     COMMON              ALL COMMON VARIABLES DECLARED
C     FETCH     GETPOI    GETS THE POINT ON THE OTHER SIDE OF AN OPEN SWITCH
C     FETCH     GETVOL    GETS THE VOLTAGE ON THE OTHER SIDE OF AN OPEN SWITCH
C     FETCH     GETFEE    GETS THE FEEDER IDENTITY ON THE OTHER SIDE OF AN OPEN
C                         SWITCH
C     FETCH     GETPLI    COMPUTES THE MARGIN OF A TRANSFORMER GETTING VIOLATED
C                         (REAL POWER)
C     FETCH     GETQLI    COMPUTES THE MARGIN OF A TRANSFORMER GETTING VIOLATED
C                         (REACTIVE POWER)
C     MOVING    STEPUP    TAKES ONE STEP UP

C     VARIABLES
C     SWI IS THE NUMBER OF THE OPEN SWITCH
C     FEE (OUT) IS THE NUMBER OF THE FEEDER
C     FOUND (OUT) INDICATES WHETHER IT WAS AN UNBROKEN CONNECTION OR NOT

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      INCLUDE 'CONSTANTS.FOR'
      INCLUDE 'COMMON.FOR'

      INTEGER SWI,FEE
      INTEGER P(2)
      LOGICAL FOUND,READY
      INTEGER GETFEE

      EXTERNAL GETFEE

      FEE=GETFEE(SWI)
      CALL GETPOI(SWI,P)
      READY=.FALSE.
 10   IF (.NOT. READY) THEN
        CALL STEPUP(P,.FALSE.)
        IF (P(1).EQ.TRATYP) THEN
          WRITE(*,*)'       DEGENERATED NET.WRONG IN REALFE'
          STOP
        ELSEIF (P(1).EQ.FEETYP) THEN
          READY=.TRUE.
        ELSEIF (P(1).EQ.SWITYP) THEN
          IF (SWITCH(P(2),1).EQ.OPEN) THEN
            READY=.TRUE.
          ENDIF
        ENDIF
        GOTO 10
      ENDIF
      FOUND=(FEE.EQ.P(2) .AND. P(1).EQ.FEETYP)
      RETURN
      END

      REAL FUNCTION GETPLI(SWI)
```

```
C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     AUTHOUR MAGNUS BERGSTRAND

C     VERSION 1

C     DATE 1989-10-07

C     PURPOSE
C     GETS THE MARGIN OF A TRANSFORMER GETTING VIOLATED (REAL POWER)

C     SEE ALSO
C     FILES     ROUTINES  COMMENTS
C     CONSTANTS           ALL CONSTANTS DECLARED
C     COMMON              ALL COMMON VARIABLES DECLARED
C     FETCH     GETPOI    GETS THE POINT ON THE OTHER SIDE OF AN OPEN SWITCH
C     FETCH     GETVOL    GETS THE VOLTAGE ON THE OTHER SIDE OF AN OPEN SWITCH
C     FETCH     GETFEE    GETS THE FEEDER IDENTITY ON THE OTHER SIDE OF AN OPEN
C                         SWITCH
C     FETCH     REALFE    GETS THE FEEDER IDENTITY ON THE OTHER SIDE OF AN OPEN
C                         SWITCH AND TESTS THAT IT IS AN UNBROKEN CONNECTION
C     FETCH     GETQLI    COMPUTES THE MARGIN OF A TRANSFORMER GETTING VIOLATED
C                         (REACTIVE POWER)

C     VARIABLES
C     THE MARGIN IN W.
C     SWI IS THE NUMBER OF THE OPEN SWITCH

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      INCLUDE 'CONSTANTS.FOR'
      INCLUDE 'COMMON.FOR'

      INTEGER SWI,P(2)
      INTEGER FEE,TRA

      IF (SWITCH(SWI,1).EQ.CLOSED) THEN
        WRITE(*,*)'        MISUSED GETPLI'
        STOP
      ELSE
        FEE=GETFEE(SWI)
        TRA=MSFEED(FEE,1)
        GETPLI=TRAFOS(TRA,1)-TRAFOS(TRA,3)
      ENDIF
      RETURN
      END


      REAL FUNCTION GETQLI(SWI)

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     AUTHOUR MAGNUS BERGSTRAND

C     VERSION 1

C     DATE 1989-10-07

C     PURPOSE
C     GETS THE MARGIN OF GETTING VIOLATED FOR A TRANSFORMER ON THE OTHER SIDE
C     OF AN OPEN SWITCH (REACTIVE POWER)

C     SEE ALSO
C     FILES     ROUTINES  COMMENTS
C     CONSTANTS           ALL CONSTANTS DECLARED
```

```
C       COMMON                    ALL COMMON VARIABLES DECLARED
C       FETCH     GETPOI          GETS THE POINT ON THE OTHER SIDE OF AN OPEN SWITCH
C       FETCH     GETPOI          GETS THE VOLTAGE ON THE OTHER SIDE OF AN OPEN SWITCH
C       FETCH     GETPOI          GETS THE FEEDER IDENTITY ON THE OTHER SIDE OF AN OPEN
C                                 SWITCH
C       FETCH     REALFE          GETS THE FEEDER IDENTITY ON THE OTHER SIDE OF AN OPEN
C                                 SWITCH AND TESTS THAT IT IS AN UNBROKEN CONNECTION
C       FETCH     GETPLI          COMPUTES THE MARGIN OF A TRANSFORMER GETTING VIOLATED
C                                 (REAL POWER)
C       FETCH     GETQLI          COMPUTES THE MARGIN OF A TRANSFORMER GETTING VIOLATED
C                                 (REACTIVE POWER)

C       VARIABLES
C       THE MARGIN IN VA.
C       SWI IS THE NUMBER OF THE OPEN SWITCH

C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        INCLUDE 'CONSTANTS.FOR'
        INCLUDE 'COMMON.FOR'

        INTEGER SWI,P(2)
        INTEGER FEE,TRA

        IF (SWITCH(SWI,1).EQ.CLOSED) THEN
          WRITE(*,*)'       MISUSED GETQLI'
          STOP
        ELSE
          FEE=GETFEE(SWI)
          TRA=MSFEED(FEE,1)
          GETQLI=TRAFOS(TRA,2)-TRAFOS(TRA,4)
        ENDIF
        RETURN
        END
```

```
C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C                        STEP1.FOR

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC




      SUBROUTINE STEP1(FAULT)

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     AUTHOUR MAGNUS BERGSTRAND

C     VERSION 1

C     DATE 1989-10-07

C     PURPOSE
C     ISOLATES A FAULTED ZONE AND CONNECTS DEENERGIZED NON FAULTED ZONES
C     TO ADJACENT FEEDERS.

C     METHOD
C     THE METHOD IS DESCRIBED IN 'SERVICE RESTORATION IN ELECTRIC
C     DISTRIBUTION NETWORKS' BY MAGNUS BERGSTRAND
C     THE METHOD WAS FIRST DESCRIBED BY AOKI IN
C     'A NEW ALGORITHM FOR SERVICE RESTORATION IN DISTRIBUTION SYSTEMS'
C     IEEE/PES WINTER MEETING, NEW YORK 1989 PAPER 89 WM 085-2 PWRD

C     DESCRIPTION
C     FAULT IS THE FAULTED POINT (TYPE,NR)
C     THERE ARE TWO BASIC TYPES OF FAULTS.
C     FAULTS AMONG THE ZONES OR FAULTS "ABOVE THE FEEDER" (IN A TRANSFORMER)
C     IF A NODE IS FAULTED THIS IS ISOLATED BY FIRST FINDING THE FIRST SWITCH
C     ABOVE THE NODE. CLOSE IT AND THEN CLOSING ALL FIRST SWITCHES BELOW
C     THE FAULTED NODE. IT CAN BE MORE THAN ONE SWITCH DUE TO BRANCH POINTS.
C     WHEN A TRANSFORMER IS FAULTED THE FAULT IS ISOLATED BY THE FAULT ITSELF.
C     ALL THE CIRCUIT BREAKERS ARE TRIPED BY THE FAULT.
C     AFTER THE FAULT HAS BEEN ISOLATED THE ROOTS OF THE ISOLATED TREES
C     ARE WRITTEN ON A FILE AND THEN READ BACK. STARTING FROM THESE ROOTS
C     ALL POSSIBLE CONNECTIONS ARE EXAMINED AND THE ONE WITH THE LEAST CHANCE
C     OF CAUSING ANY VIOLATION IS SELECTED REENERGIZING THE TREE.

C     SEE ALSO
C     FILES   ROUTINES   COMMENTS
C     STEP2   STEP2      SECOND STEP IN THE ALGORITHM
C     STEP2   STEP3      THIRD AND FOURTH STEP IN THE ALGORITHM
C     STEP1   ABOVEF     LOGICAL FUNCTION. IS THE FAULT LOCATED ABOVE ANY FEEDER
C     MOVING  MOVE       TAKES ONE STEP UP OR DOWN
C     MOVING  DIREC      COMPUTES NEW DIRECTIONS
C     MOVING  STEPUP     TAKES ONE STEP UP
C     MOVING  STEPDO     TAKES ONE STEP DOWN
C     COMP    CON        CONNECTS DEENERGIZED ZONES
C     CONSTANTS          ALL CONSTANTS
C     COMMON             ALL COMMON VARIABELS ARE DECLARED

C     VARIABLES
C     FAULT IS THE FAULTED POINT (TYPE,NR)

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      INCLUDE 'CONSTANTS.FOR'
      INCLUDE 'COMMON.FOR'
```

```
      INTEGER FAULT(1:2)
      INTEGER TRAFO,FEEDER,ISOSWI
      INTEGER CURPOI(2),P(2),P1(2),I
      INTEGER DIR,BRA,OLDP(2),Y,J
      LOGICAL READY,WR,CHG,ABOVEF,OK

      PARAMETER(TRASH=1)

      EXTERNAL STEPUP,STEPDO
      EXTERNAL MOVE,DIREC
      EXTERNAL CON,ABOVEF

      OPEN(TRASH,STATUS='SCRATCH',ACCESS='SEQUENTIAL')
      CURPOI(1)=FAULT(1)
      CURPOI(2)=FAULT(2)
      WR=.FALSE.
      FEEDER=0
      ISOSWI=0
      ISOLAT(1)=0.0
      ISOLAT(2)=0.0
      IF (.NOT. ABOVEF(CURPOI)) THEN

C        FAULT AMONG SECTIONS (BELOW FEEDER)

         READY=CURPOI(1).EQ.TRATYP
 10      IF (.NOT. READY) THEN

C           FIND FIRST SWITCH

            IF (CURPOI(1).EQ.CABTYP) THEN
               CALL STEPDO(CURPOI,BRA,WR)
            ENDIF
            CALL STEPUP(CURPOI,WR)
            READY=(CURPOI(1).EQ.SWITYP .OR. CURPOI(1).EQ.TRATYP)
            GOTO 10
         ENDIF
         DIR=DOWN
         BRA=3
         P(1)=CURPOI(1)
         P(2)=CURPOI(2)
         READY=.FALSE.
         CHG=.FALSE.
 20      IF (.NOT. READY) THEN

C           FIND ALL SWITCHES BELOW THE FAULTED ZONE
C           THESE SWITCHES ARE OPENED
C           THEY MUST BE OPENED IN THIS PERHAPS ODD WAY BECAUSE OTHERWISE WILL
C           NEITHER STEPUP NOR STEPDO WORK CORRECTLY

            IF (CHG) THEN
               SWITCH(Y,1)=CLOSED
            ENDIF
            CALL MOVE(P,OLDP,DIR,BRA,WR)
            IF (CHG) THEN
               SWITCH(Y,1)=OPEN
            ENDIF
            READY=(P(2).EQ.CURPOI(2) .AND. P(1).EQ.CURPOI(1))
            IF (READY) THEN
               GOTO 20
            ENDIF
            CHG=.FALSE.
            IF (P(1).EQ.SWITYP) THEN
               I=P(2)
               IF (SWITCH(I,1).EQ.CLOSED) THEN
                  SWITCH(I,1)=OPEN
                  SWITCH(I,5)=0
```

```
            SWINR(I)=1
            CHGSWI(I)=.NOT. CHGSWI(I)
            CHG=.TRUE.
            Y=I
            IF (SWITCH(I,2).EQ.3) THEN
              J=SWITCH(I,4)
            ELSE
              J=SWITCH(I,3)
            ENDIF
C
C           SAVE THE POINT BELOW THE SWITCH THAT HAS ISOLATED THE FAULT
C           THIS POINT IS THEN USED AS ROOT IN A TREE IN ORDER TO DECIDE
C           THE BEST WAY TO REENERGIZE THE NON FAULTED ZONES
C
            IF (CABTOP(J,1).EQ.SWITYP .AND. CABTOP(J,2).EQ.I) THEN
              WRITE(TRASH,*) CABTOP(J,3),CABTOP(J,4)
            ELSEIF (CABTOP(J,3).EQ.SWITYP .AND. CABTOP(J,4).EQ.I) THEN
              WRITE(TRASH,*) CABTOP(J,1),CABTOP(J,2)
            ELSE
              WRITE(*,*)'       WRONG IN STEP 1'
              STOP
            ENDIF
          ELSEIF (SWITCH(I,1).EQ.OPEN) THEN
            SWITCH(I,5)=0
          ELSE
            WRITE(*,*)'       WRONG IN STEP1',SWITCH(I,1),OPEN,CLOSED
            STOP
          ENDIF
        ENDIF
        CALL DIREC(P,OLDP,DIR,BRA)
        GOTO 20
      ENDIF
      IF (CURPOI(1).EQ.SWITYP) THEN
        SWITCH(CURPOI(2),1)=OPEN
        SWITCH(CURPOI(2),5)=0
        SWINR(CURPOI(2))=1
        CHGSWI(CURPOI(2))=.NOT. CHGSWI(CURPOI(2))
      ENDIF
      ENDFILE TRASH
      REWIND TRASH
    ELSE
C
C     THE FAULT WAS IN A TRANSFORMER (ABOVE THE FEEDER)
C     START FROM THE TRANSFORMER MOVING UPWARDS
C     OPENING ALL SWITCHES ON THE WAY UNTIL YOU FIND A FEEDER. STOP.
C     WRITE THE NUMBER OF THE FEEDER ON A FILE
C     DO THIS FOR ALL POSSIBLE BRANCHES
C
      OPEN(3,STATUS='SCRATCH',ACCESS='SEQUENTIAL')
      READY=CURPOI(1).EQ.TRATYP
 100  IF (.NOT. READY) THEN
        IF (CURPOI(1).EQ.CABTYP) THEN
          CALL STEPDO(CURPOI,BRA,WR)
        ENDIF
        CALL STEPUP(CURPOI,WR)
        READY=(CURPOI(1).EQ.SWITYP .OR. CURPOI(1).EQ.TRATYP)
        GOTO 100
      ENDIF
      IF (CURPOI(1).EQ.SWITYP) THEN
        WRITE(3,*)CURPOI(2)
      ENDIF
      DIR=DOWN
      BRA=3
      P(1)=CURPOI(1)
      P(2)=CURPOI(2)
      READY=.FALSE.
```

```
 200    IF (.NOT. READY) THEN
          CALL MOVE(P,OLDP,DIR,BRA,.FALSE.)
          READY=(P(2).EQ.CURPOI(2) .AND. P(1).EQ.CURPOI(1))
          IF (READY) THEN
            GOTO 200
          ENDIF
          IF (P(1).EQ.FEETYP) THEN
            TRAFO=MSFEED(P(2),1)

C           ISOLATED LOAD

            ISOLAT(1)=TRAFOS(TRAFO,3)
            ISOLAT(2)=TRAFOS(TRAFO,4)
            DIR=UP
            P1(1)=FEETYP
            P1(2)=P(2)
            CALL STEPDO(P1,3,.FALSE.)
            IF (P1(1).EQ.SWITYP) THEN
              I=P1(2)
              IF (SWITCH(I,1).EQ.CLOSED) THEN
                SWITCH(I,1)=OPEN
                SWITCH(I,5)=0
                SWINR(I)=1
                CHGSWI(I)=.NOT. CHGSWI(I)
                IF (SWITCH(I,2).EQ.3) THEN
                  J=SWITCH(I,4)
                ELSE
                  J=SWITCH(I,3)
                ENDIF
                IF (CABTOP(J,1).EQ.SWITYP .AND. CABTOP(J,2).EQ.I) THEN
                  WRITE(TRASH,*) CABTOP(J,3),CABTOP(J,4)
                ELSEIF (CABTOP(J,3).EQ.SWITYP .AND. CABTOP(J,4).EQ.I) THEN
                  WRITE(TRASH,*) CABTOP(J,1),CABTOP(J,2)
                ELSE
                  WRITE(*,*)'       WRONG IN STEP 1'
                  STOP
                ENDIF
              ELSEIF (SWITCH(I,1).EQ.OPEN) THEN
                SWITCH(I,5)=0
              ENDIF
            ENDIF
          ELSEIF (P(1).EQ.SWITYP .AND. DIR.EQ.DOWN) THEN
            IF (SWITCH(P(2),1).EQ.CLOSED) THEN
              WRITE(3,*)P(2)
            ENDIF
          ENDIF
          CALL DIREC(P,OLDP,DIR,BRA)
          GOTO 200
        ENDIF
        IF (CURPOI(1).EQ.SWITYP) THEN
          SWITCH(CURPOI(2),1)=OPEN
          SWITCH(CURPOI(2),5)=0
          SWINR(CURPOI(2))=1
          CHGSWI(CURPOI(2))=.NOT. CHGSWI(CURPOI(2))
        ENDIF
        ENDFILE TRASH
        REWIND TRASH
        ENDFILE 3
        REWIND 3
 205    READ(3,*,ERR=210) I

C         READ THE NUMBER OF THE SWITCHES THAT WILL BE OPENED TO ISOLATE THE
C         FAULT

          SWITCH(I,1)=OPEN
          SWITCH(I,5)=0
```

```
            SWINR(I)=1
            CHGSWI(I)=.NOT. CHGSWI(I)
            GOTO 205
 210     ENDIF
 30      READ(TRASH,*,END=40) CURPOI(1),CURPOI(2)

C        READS AND CONNECTS THE DEENERGIZED NODES

         CALL CON(CURPOI,OK)
         IF (.NOT. OK) THEN
            RESTOR(1)=0.0
            RESTOR(2)=0.0
         ENDIF
         GOTO 30
 40      CLOSE(TRASH,STATUS='DELETE')
         IF (OK) THEN
            RESTOR(1)=ISOLAT(1)
            RESTOR(2)=ISOLAT(2)
         ENDIF
         RETURN
         END



         LOGICAL FUNCTION ABOVEF(POI)

C        CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C        AUTHOUR MAGNUS BERGSTRAND

C        VERSION 1

C        DATE 1989-10-07

C        PURPOSE
C        INDICATES IF A FAULT HAS OCCURRED ABOVE A FEEDER

C        DESCRIPTION
C        MOVES UPWARDS UNTIL A FEEDER OR A TRANSFORMER IS FOUND.
C        IF A TRANSFORMER IS FOUND THEN TRUE

C        SEE ALSO
C        FILES     ROUTINES   COMMENTS
C        STEP1     STEP1      ISOLATES THE FAULT AND CONNECTS THE NON FAULTED ZONES.
C                             CALLS ABOVEF
C        MOVING    STEPUP     TAKES ONE STEP UP
C        CONSTANTS            ALL CONSTANTS DECLARED
C        COMMON               ALL COMMON VARIABLES DECLARED

C        VARIABLES
C        POI IS THE FAULTED POINT (TYPE,NR)

C        CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

         INCLUDE 'CONSTANTS.FOR'
         INCLUDE 'COMMON.FOR'

         INTEGER POI(2),P(2)
         LOGICAL READY

         P(1)=POI(1)
         P(2)=POI(2)
         READY=(P(1).EQ.FEETYP .OR. P(1).EQ.TRATYP)
 10      IF (.NOT. READY) THEN
            CALL STEPUP(P,.FALSE.)
```

```
   READY=(P(1).EQ.FEETYP .OR. P(1).EQ.TRATYP)
   GOTO 10
ENDIF
ABOVEF=(P(1).EQ.TRATYP)
RETURN
END
```

```
C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C             MOVING.FOR

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC




      SUBROUTINE STEPUP(POINT,WR)

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     AUTHOUR MAGNUS BERGSTRAND

C     VERSION 1

C     DATE 1989-10-07

C     PURPOSE
C     TAKES ONE STEP UP

C     DESCRIPTION
C     A CENTRAL SUBROUTINE.
C     NOTICE THAT POINT CAN BE OF ANY TYPE EXCEPT CABLE.
C     IN MSFEED,NODETO,BRAPOI AND SWITCH A POINTER UP IS FOUND.
C     IT POINTS OUT AN INDEX TO THE CABLE LEADING TOWARDS THE TRANSFORMER
C     IN CABTOP (CABLE TOPOLOGY) THE TWO END POINTS OF THE CABLE ARE STORED .
C     THIS SUBROUTINE MOVES 'POINT' TO THE OTHER END POINT IN DIRECTION UP

C     SEE ALSO
C     FILES   ROUTINES   COMMENTS
C     MOVING  STEPDO     TAKES ONE STEP DOWN
C     MOVING  MOVE       TAKES ONE STEP UP OR DOWN
C     MOVING  DIREC      COMPUTES NEW DIRECTION
C     CONSTANT           ALL CONSTANTS DECLARED
C     COMMON             ALL COMMON VARIABLES DECLARED

C     VARIABLES
C     POINT IS THE CURRENT POINT (TYPE,NR)
C     IF WR=TRUE THE CURRENT POINT WILL BE PRINTED ON THE SCREEN

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      INCLUDE 'CONSTANTS.FOR'
      INCLUDE 'COMMON.FOR'

      INTEGER POINT(1:2),I
      LOGICAL WR

      IF (WR) THEN
        WRITE(*,*)'        POINT IN IS',POINT(1),POINT(2)
      ENDIF
      IF (POINT(1) .EQ. BRATYP) THEN
        I=BRAPOI(POINT(2),2)
        I=BRAPOI(POINT(2),I)
        IF (CABTOP(I,1).EQ.POINT(1) .AND. CABTOP(I,2).EQ.POINT(2)) THEN
          POINT(1)=CABTOP(I,3)
          POINT(2)=CABTOP(I,4)
        ELSE
          POINT(1)=CABTOP(I,1)
          POINT(2)=CABTOP(I,2)
        ENDIF
      ELSEIF (POINT(1).EQ.NODTYP) THEN
        I=NODETO(POINT(2),1)
```

```
        IF (WR) THEN
          WRITE(*,*)'        CABUP IN NODE IS',I
        ENDIF
        I=NODETO(POINT(2),I)
        IF (WR) THEN
          WRITE(*,*)'        CABLE UP IS',I
        ENDIF
        IF (CABTOP(I,1).EQ.POINT(1) .AND. CABTOP(I,2).EQ.POINT(2)) THEN
          POINT(1)=CABTOP(I,3)
          POINT(2)=CABTOP(I,4)
        ELSE
          POINT(1)=CABTOP(I,1)
          POINT(2)=CABTOP(I,2)
        ENDIF
      ELSEIF (POINT(1).EQ.SWITYP) THEN
        I=SWITCH(POINT(2),2)
        I=SWITCH(POINT(2),I)
        IF (CABTOP(I,1).EQ.POINT(1) .AND. CABTOP(I,2).EQ.POINT(2)) THEN
          POINT(1)=CABTOP(I,3)
          POINT(2)=CABTOP(I,4)
        ELSE
          POINT(1)=CABTOP(I,1)
          POINT(2)=CABTOP(I,2)
        ENDIF
      ELSEIF (POINT(1).EQ.FEETYP) THEN
        I=MSFEED(POINT(2),2)
        IF (CABTOP(I,1).EQ.FEETYP .AND. CABTOP(I,2).EQ.POINT(2)) THEN
          POINT(1)=CABTOP(I,3)
          POINT(2)=CABTOP(I,4)
        ELSE
          POINT(1)=CABTOP(I,1)
          POINT(2)=CABTOP(I,2)
        ENDIF
      ELSE
        WRITE(*,*) '              WRONG TYPE IN (SUBROUTINE STEPUP)'
        STOP
      ENDIF
      IF (WR) THEN
        WRITE(*,*) '        POINT OUT FROM STEPUP IS',POINT(1),POINT(2)
      ENDIF
      RETURN
      END




      SUBROUTINE STEPDO(POINT,BRANMR,WR)

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     AUTHOUR MAGNUS BERGSTRAND

C     VERSION 1

C     DATE 1989-10-07

C     PURPOSE
C     TAKES ONE STEP DOWN

C     DESCRIPTION
C     THIS PROCEDURE TAKES A STEP IN THE DIRECTION "NOT DOWN"
C     IT WORKS FOR ALL TYPES INCLUDING CABLES.
C     WHEN POINT IS A BRANCH POINT STEPDO TAKES A STEP DOWN IN THE
C     BRANCH POINTED OUT BY 'BRANMR'
C     NOTICE THAT NO CHECK IS MADE THAT THIS BRANCH LEADS DOWNWARDS
```

```
C       SEE ALSO
C       FILES    ROUTINES    COMMENTS
C       MOVING   STEPUP      TAKES ONE STEP UP
C       MOVING   MOVE        TAKES ONE STEP UP OR DOWN
C       MOVING   DIREC       COMPUTES NEW DIRECTION
C       CONSTANT             ALL CONSTANTS DECLARED
C       COMMON               ALL COMMON VARIABLES DECLARED

C       VARIABLES
C       POINT IS THE CURRENT POINT (TYPE,NR)
C       BRANMR IS THE NUMBER OF THE DOWNWARDS LEADING BRANCH THAT WILL BE VISITED
C       IF WR=TRUE THEN THE CURRENT POINT WILL BE WRITTEN ON THE SCREEN

C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        INCLUDE 'CONSTANTS.FOR'
        INCLUDE 'COMMON.FOR'

        INTEGER BRANMR,POINT(2)
        LOGICAL WR
        INTEGER I,J,K,L
        IF (WR) THEN
          WRITE(*,*)'       POINT IN IS ',POINT(1),POINT(2)
        ENDIF
        IF (POINT(1).EQ.FEETYP) THEN
          I=MSFEED(POINT(2),3)
          IF (CABTOP(I,1).EQ.FEETYP .AND. CABTOP(I,2).EQ.POINT(2)) THEN
            POINT(1)=CABTOP(I,3)
            POINT(2)=CABTOP(I,4)
          ELSEIF (CABTOP(I,3).EQ.FEETYP .AND. CABTOP(I,4).EQ.POINT(2)) THEN
            POINT(1)=CABTOP(I,1)
            POINT(2)=CABTOP(I,2)
          ELSE
            WRITE(*,*)'       WRONG IN STEPDO.FEEDER'
            STOP
          ENDIF
        ELSEIF (POINT(1).EQ.BRATYP) THEN
          I=BRAPOI(POINT(2),BRANMR)
          IF (CABTOP(I,1).EQ.BRATYP .AND. CABTOP(I,2).EQ.POINT(2)) THEN
            POINT(1)=CABTOP(I,3)
            POINT(2)=CABTOP(I,4)
          ELSEIF (CABTOP(I,3).EQ.BRATYP .AND. CABTOP(I,4).EQ.POINT(2)) THEN
            POINT(1)=CABTOP(I,1)
            POINT(2)=CABTOP(I,2)
          ELSE
            WRITE(*,*)'       WRONG IN STEPDO.BRA'
            STOP
          ENDIF
        ELSEIF (POINT(1).EQ.TRATYP) THEN
          I=TRANEI(POINT(2))
          IF (CABTOP(I,1).EQ.TRATYP .AND. CABTOP(I,2).EQ.POINT(2)) THEN
            POINT(1)=CABTOP(I,3)
            POINT(2)=CABTOP(I,4)
          ELSEIF (CABTOP(I,3).EQ.TRATYP .AND. CABTOP(I,4).EQ.POINT(2)) THEN
            POINT(1)=CABTOP(I,1)
            POINT(2)=CABTOP(I,2)
          ELSE
            WRITE(*,*)'      WRONG INSTEPDO.TRAFO'
            WRITE(*,*)'TRAFO',POINT(2)
            WRITE(*,*)CABTOP(I,1),CABTOP(I,2),CABTOP(I,3),CABTOP(I,4)
            STOP
          ENDIF
        ELSEIF (POINT(1).EQ.CABTYP) THEN

C         THE USED METHOD FOR CABLES
```

```
C     LOOK AT ONE NEIGHBOUR
C     IF CABLE UP IS IDENTICAL WITH THE CURRENT CABLE GO TO THAT NEIGHBOUR
C     OTHERWISE GO TO THE OTHER NEIGHBOUR

      K=POINT(2)
      IF (CABTOP(K,1).EQ.SWITYP .OR. CABTOP(K,3).EQ.SWITYP) THEN
        IF (CABTOP(POINT(2),1).EQ.SWITYP) THEN
          I=1
        ELSE
          I=3
        ENDIF
        J=MOD((I+2),4)
        IF (CABTOP(POINT(2),J).EQ.TRATYP) THEN
          POINT(1)=CABTOP(POINT(2),I)
          POINT(2)=CABTOP(POINT(2),I+1)
        ELSEIF (CABTOP(POINT(2),J).EQ.FEETYP) THEN
          K=CABTOP(POINT(2),J+1)
          IF (POINT(2).EQ.MSFEED(K,2)) THEN
            POINT(1)=FEETYP
            POINT(2)=CABTOP(POINT(2),J+1)
          ELSE
            POINT(1)=SWITYP
            POINT(2)=CABTOP(POINT(2),I+1)
          ENDIF
        ELSEIF (CABTOP(POINT(2),J).EQ.BRATYP) THEN
          K=CABTOP(POINT(2),J+1)
          L=BRAPOI(K,2)
          IF (POINT(2).EQ.BRAPOI(K,L)) THEN
            POINT(1)=BRATYP
            POINT(2)=CABTOP(POINT(2),J+1)
          ELSE
            POINT(1)=SWITYP
            POINT(2)=CABTOP(POINT(2),I+1)
          ENDIF
        ELSEIF (CABTOP(POINT(2),J).EQ.NODTYP) THEN
          K=CABTOP(POINT(2),J+1)
          L=NODETO(K,1)
          IF (POINT(2).EQ.NODETO(K,L)) THEN
            POINT(1)=NODTYP
            POINT(2)=CABTOP(POINT(2),J+1)
          ELSE
            POINT(1)=SWITYP
            POINT(2)=CABTOP(POINT(2),I+1)
          ENDIF
        ENDIF
      ELSEIF (CABTOP(POINT(2),1).EQ.TRATYP) THEN
        POINT(1)=CABTOP(POINT(2),3)
        POINT(2)=CABTOP(POINT(2),4)
      ELSEIF (CABTOP(POINT(2),1).EQ.FEETYP) THEN
        K=CABTOP(POINT(2),2)
        IF (POINT(2).EQ.MSFEED(K,2)) THEN
          POINT(1)=FEETYP
          POINT(2)=CABTOP(POINT(2),2)
        ELSE
          POINT(1)=CABTOP(POINT(2),3)
          POINT(2)=CABTOP(POINT(2),4)
        ENDIF
      ELSEIF (CABTOP(POINT(2),1).EQ.CABTYP) THEN
        WRITE(*,*) '                DEGENERATED NETWORK'
        STOP
      ELSEIF (CABTOP(POINT(2),1).EQ.BRATYP) THEN
        K=CABTOP(POINT(2),2)
        L=BRAPOI(K,2)
        IF (POINT(2).EQ.BRAPOI(K,L)) THEN
          POINT(1)=BRATYP
          POINT(2)=CABTOP(POINT(2),2)
```

```
        ELSE
          POINT(1)=CABTOP(POINT(2),3)
          POINT(2)=CABTOP(POINT(2),4)
        ENDIF
      ELSEIF (CABTOP(POINT(2),1).EQ.NODTYP) THEN
        K=CABTOP(POINT(2),2)
        L=NODETO(K,1)
        IF (POINT(2).EQ.NODETO(K,L)) THEN
          POINT(1)=NODTYP
          POINT(2)=CABTOP(POINT(2),2)
        ELSE
          POINT(1)=CABTOP(POINT(2),3)
          POINT(2)=CABTOP(POINT(2),4)
        ENDIF
      ENDIF

C     HERE THE SECTION CONCEARNING CABLES ENDS

    ELSEIF (POINT(1).EQ.NODTYP) THEN
      IF (NODETO(POINT(2),1).EQ.2) THEN
        I=3
      ELSEIF (NODETO(POINT(2),1).EQ.3) THEN
        I=2
      ELSE
        WRITE(*,*)'       WRONG IN STEPDO.NODE'
        STOP
      ENDIF
      J=NODETO(POINT(2),I)
      IF (CABTOP(J,1).EQ.NODTYP .AND. POINT(2).EQ.CABTOP(J,2)) THEN
        POINT(1)=CABTOP(J,3)
        POINT(2)=CABTOP(J,4)
      ELSEIF (CABTOP(J,3).EQ.NODTYP .AND. POINT(2).EQ.CABTOP(J,4)) THEN
        POINT(1)=CABTOP(J,1)
        POINT(2)=CABTOP(J,2)
      ELSE
        WRITE(*,*)'       WRONG IN STEPDO.NODE'
        STOP
      ENDIF
    ELSEIF (POINT(1).EQ.SWITYP) THEN
      IF (SWITCH(POINT(2),1).EQ.CLOSED) THEN
        IF (SWITCH(POINT(2),2).EQ.3) THEN
          I=4
        ELSE
          I=3
        ENDIF
        J=SWITCH(POINT(2),I)
        IF (CABTOP(J,1).EQ.SWITYP .AND. CABTOP(J,2).EQ.POINT(2)) THEN
          POINT(1)=CABTOP(J,3)
          POINT(2)=CABTOP(J,4)
        ELSEIF (CABTOP(J,3).EQ.6 .AND. CABTOP(J,4).EQ.POINT(2)) THEN
          POINT(1)=CABTOP(J,1)
          POINT(2)=CABTOP(J,2)
        ELSE
          WRITE(*,*)'       WRONG IN STEPDO.SWITCH'
          WRITE(*,*) SWITYP,POINT(2)
          WRITE(*,*)CABTOP(J,1),CABTOP(J,2),CABTOP(J,3),CABTOP(J,4)
          STOP
        ENDIF
      ENDIF
    ENDIF
    IF (WR) THEN
      WRITE(*,*)'       POINT OUT FROM STEP DOWN IS',POINT(1),POINT(2)
    ENDIF
    RETURN
    END
```

```
      SUBROUTINE MOVE(POINT,OLD,DIR,BRA,WR)

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     AUTHOUR MAGNUS BERGSTRAND

C     VERSION 1

C     DATE 1989-10-07

C     PURPOSE
C     TAKES ONE STEP UP OR DOWN. USED IN COOPERATION WITH NEWDIR IT CAN BE
C     USED TO TRAVERSE A TREE.

C     DESCRIPTION
C     DIR INDICATES THE DIRECTION (UP,DOWN)
C     IF DIRECTION IS DOWN AND POINT IS BRANCH POINT THE VARIABLE BRA IS VALID.
C     IT INDICATES THE BRANCH THAT WILL BE 'VISITED'
C     THIS SUBROUTINE WORKS IF THE NETWORK TOPOLOGY IS ALREADY INITIATED.
C     NOTICE THAT THIS PROCEDURE ASSIGNS A NEW VALUE TO 'CABLE UP' IF A
C     CLOSED SWITCH IS FOUND. UP WILL BE ASSIGNED THE DIRECTION TOWARDS OLD.

C     SEE ALSO
C     FILES    ROUTINES    COMMENTS
C     MOVING   STEPUP      TAKES ONE STEP UP
C     MOVING   STEPDO      TAKES ONE STEP DOWN
C     MOVING   DIREC       COMPUTES NEW DIRECTION
C     CONSTANT             ALL CONSTANTS DECLARED
C     COMMON               ALL COMMON VARIABLES DECLARED
C     TESTS    TKESTP      TAKES ONE STEP UP OR DOWN. WORKS IN AN UNINITIATED
C                          NETWORK.

C     VARIABLES
C     POINT IS THE CURRENT POINT (TYPE,NR)
C     OLD IS THE FORMER POINT (TYPE,NR)
C     DIR INDICATES THE DIRECTION (UP,DOWN)
C     BRA IS THE NUMBER OF THE BRANCH
C     IF WR=TRUE THEN THE CURRENT POINT WILL BE WRITTEN ON THE SCREEN

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      INCLUDE 'CONSTANTS.FOR'
      INCLUDE 'COMMON.FOR'

      INTEGER POINT(2),OLD(2)
      INTEGER DIR,BRA
      INTEGER TYP,NR,P(2)
      LOGICAL WR

      TYP=POINT(1)
      NR=POINT(2)
      OLD(1)=POINT(1)
      OLD(2)=POINT(2)
      IF (DIR.EQ.DOWN) THEN
        IF (TYP.EQ.SWITYP) THEN
          IF (SWITCH(NR,1).EQ.CLOSED) THEN

C           IT IS ONLY POSSIBLE TO TAKE ONE STEP DOWN IF THE SWITCH IS CLOSED

            CALL STEPDO(POINT,BRA,WR)
            IF (POINT(1).EQ.SWITYP) THEN
              IF(SWITCH(POINT(2),1).EQ.OPEN) THEN

C             THE RESULT FROM THIS STEP WAS AN OPEN SWITCH
```

```
C           IN THIS SWITCH ASSIGN UP A NEW VALUE TOWARDS OLD.

              I=SWITCH(POINT(2),3)
              IF (CABTOP(I,1).EQ.OLD(1) .AND. CABTOP(I,2).EQ.OLD(2)) THEN
                SWITCH(POINT(2),2)=3
              ELSEIF (CABTOP(I,3).EQ.OLD(1) .AND. CABTOP(I,4).EQ.OLD(2)) THEN
                SWITCH(POINT(2),2)=3
              ELSE
                SWITCH(POINT(2),2)=4
              ENDIF
            ENDIF
          ENDIF
        ENDIF
      ELSE
        CALL STEPDO(POINT,BRA,WR)
        IF (POINT(1).EQ.SWITYP) THEN
          IF(SWITCH(POINT(2),1).EQ.OPEN) THEN

C           THE RESULT FROM THIS STEP WAS AN OPEN SWITCH
C           IN THIS SWITCH ASSIGN UP A NEW VALUE TOWARDS OLD.

            I=SWITCH(POINT(2),3)
            IF (CABTOP(I,1).EQ.OLD(1) .AND. CABTOP(I,2).EQ.OLD(2)) THEN
              SWITCH(POINT(2),2)=3
            ELSEIF (CABTOP(I,3).EQ.OLD(1) .AND. CABTOP(I,4).EQ.OLD(2)) THEN
              SWITCH(POINT(2),2)=3
            ELSE
              SWITCH(POINT(2),2)=4
            ENDIF
          ENDIF
        ENDIF
      ENDIF
      ELSEIF (DIR.EQ.UP) THEN
        CALL STEPUP(POINT,WR)
      ENDIF
      RETURN
      END


      SUBROUTINE DIREC(POINT,OLDP,DIR,BRA)

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     AUTHOUR MAGNUS BERGSTRAND

C     VERSION 1

C     DATE 1989-10-07

C     PURPOSE
C     COMPUTES A NEW DIRECTION

C     DESCRIPTION
C     IS INTENDED TO COOPERATE WITH MOVE. THEN IT CAN BE USED TO TRAVERSE
C     A TREE WITH EITHER AN OPEN SWITCH OR A 'LEAF NODE' (ONLY ONE FEEDING
C     CABLE ATTACHED) AS LEAFS.

C     SEE ALSO
C     FILES    ROUTINES   COMMENTS
C     MOVING   STEPUP     TAKES ONE STEP UP
C     MOVING   STEPDO     TAKES ONE STEP DOWN
C     MOVING   MOVE       TAKES ONE STEP UP OR DOWN
C     TESTS    NEWDIR     COMPUTES A NEW DIRECTION IN A NON INITIATED NETWORK
C     CONSTANT            ALL CONSTANTS DECLARED
C     COMMON              ALL COMMON VARIABLES DECLARED
```

```
C       VARIABLES
C       POINT IS THE CURRENT POINT (TYPE,NR)
C       OLDP IS THE FORMER POINT (TYPE,NR)
C       DIR IS THE DIRECTION
C       BRA IS THE NUMBER OF THE BRANCH

C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        INCLUDE 'CONSTANTS.FOR'
        INCLUDE 'COMMON.FOR'

        INTEGER POINT(2),OLDP(2)
        INTEGER DIR,BRA,NXTBRA
        INTEGER I,C
        LOGICAL READY

        IF (DIR.EQ.DOWN) THEN
          IF (POINT(1).EQ.NODTYP) THEN
            IF (NODETO(POINT(2),2).EQ.0 .OR. NODETO(POINT(2),3).EQ.0) THEN

C             IF THE NODE IS A LEAF NODE CHANGE DIRECTIONS!

              DIR=UP
            ENDIF
          ELSEIF (POINT(1).EQ.SWITYP) THEN
            IF (SWITCH(POINT(2),1).EQ.OPEN) THEN

C             IF AN OPEN SWITCH IS FOUND CHANGE DIRECTION!

              DIR=UP
            ENDIF
          ELSEIF (POINT(1).EQ.BRATYP) THEN

C           IF CABLE UP=4 THEN BRA=4 ELSE BRA=3

            IF (BRAPOI(POINT(2),2).EQ.3) THEN
              BRA=4
            ELSE
              BRA=3
            ENDIF
          ENDIF
        ELSEIF (DIR.EQ.UP) THEN

C         IN THIS CASE THE ONLY TIME WHEN THE DIRECTION IS CHANGED IS WHEN
C         A BRANCH POINT IS REACHED.
C         THEN THE NEXT BRANCH VISITED WILL BE BRA+1 IF THIS BRANCH EXISTS
C         AND IF IT IS NOT THE BRANCH UP

          IF (POINT(1).EQ.BRATYP) THEN
            BRA=2
            I=POINT(2)
            READY=.FALSE.
 10         IF (.NOT. READY) THEN
              BRA=BRA+1
              C=BRAPOI(I,BRA)
              IF (CABTOP(C,1).EQ.OLDP(1) .AND. CABTOP(C,2).EQ.OLDP(2)) THEN
                READY=.TRUE.
              ELSEIF (CABTOP(C,3).EQ.OLDP(1) .AND. CABTOP(C,4).EQ.OLDP(2)) THEN
                READY=.TRUE.
              ENDIF
              GOTO 10
            ENDIF
            NXTBRA=BRA+1
            IF (NXTBRA.LE.(2+BRAPOI(I,1)) .AND. NXTBRA.NE.BRAPOI(I,2)) THEN
              DIR=DOWN
              BRA=NXTBRA
```

```
      ELSEIF (NXTBRA.LT.BRAPOI(I,1)+2 .AND. (NXTBRA+1).NE.BRAPOI(I,2)) THEN
        BRA=BRA+2
        DIR=DOWN
      ENDIF
    ENDIF
  ENDIF
  RETURN
  END
```

```
C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C                       TESTS.FOR

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC



      SUBROUTINE CABFOR(J)

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     AUTHOUR MAGNUS BERGSTRAND

C     VERSION 1

C     DATE 1989-10-06

C     PURPOSE
C     PRODUCES A MENUE

C     SEE ALSO
C     FILES    ROUTINES    COMMENTS
C     CONSTANTS            ALL CONSTANTS DECLARED
C     COMMON               ALL COMMON VARIABLES DECLARED
C     TESTS    TRTEST      TEST OF TOPOLOGY (TRANSFORMERS)
C     TESTS    FETEST      TEST OF TOPOLOGY (FEEDERS)
C     TESTS    BRTEST      TEST OF TOPOLOGY (BRANCH POINTS)
C     TESTS    NOTEST      TEST OF TOPOLOGY (NODES)
C     TESTS    SWTEST      TEST OF TOPOLOGY (SWITCHES)
C     THE ABOVE CALLS CABFOR

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      INCLUDE 'CONSTANTS.FOR'
      INCLUDE 'COMMON.FOR'

      INTEGER J

      WRITE(*,200) 'CABLE NR',J,' IS CONNECTED TO '
      WRITE(*,300) 'TYPE=',CABTOP(J,1),'  NR=',CABTOP(J,2)
      WRITE(*,*)'                  AND'
      WRITE(*,300) 'TYPE=',CABTOP(J,3),'  NR=',CABTOP(J,4)
 200  FORMAT(' ',TR6,A,I5,A)
 300  FORMAT(' ',TR6,A,I2,A,I6)
      RETURN
      END



      SUBROUTINE WRICAB(INDEX)

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     AUTHOUR MAGNUS BERGSTRAND

C     VERSION 1

C     DATE 1989-10-06

C     PURPOSE
C     PRODUCES A MENUE

C     SEE ALSO
C     FILES    ROUTINES    COMMENTS
```

```
C       CONSTANTS               ALL CONSTANTS DECLARED
C       COMMON                  ALL COMMON VARIABLES DECLARED
C       TESTS    TRTEST         TEST OF TOPOLOGY (TRANSFORMERS)
C       TESTS    FETEST         TEST OF TOPOLOGY (FEEDERS)
C       TESTS    BRTEST         TEST OF TOPOLOGY (BRANCH POINTS)
C       TESTS    NOTEST         TEST OF TOPOLOGY (NODES)
C       TESTS    SWTEST         TEST OF TOPOLOGY (SWITCHES)
C       THE ABOVE CALLS WRICAB

C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        INCLUDE 'CONSTANTS.FOR'
        INCLUDE 'COMMON.FOR'
        INTEGER INDEX

        WRITE(*,*)'       CABLE IS CONNECTED TO:     TYPE?  NR?'
        READ(*,*) CABTOP(INDEX,1),CABTOP(INDEX,2)
        WRITE(*,*)'       AND TO:                    TYPE?  NR?'
        READ(*,*) CABTOP(INDEX,3),CABTOP(INDEX,4)
        WRITE(*,*)'       R? X? IMAX?'
        READ(*,*) CABPHY(J,1),CABPHY(J,2),CABPHY(J,3)
        RETURN
        END



        SUBROUTINE TRTEST(OK)

C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C       AUTHOUR MAGNUS BERGSTRAND

C       VERSION 1

C       DATE 1989-10-06

C       PURPOSE
C       TESTS THE TOPOLOGY REGARDING TRANSFORMERS

C       DESCRIPTION
C       IF A POINTS NEIGHBOURS NEIGHBOUR IS NOT THE POINT ITSELF
C       A FAULT IN THE TOPOLOGY IS DETECTED

C       SEE ALSO
C       FILES    ROUTINES       COMMENTS
C       CONSTANTS               ALL CONSTANTS DECLARED
C       COMMON                  ALL COMMON VARIABLES DECLARED
C       TESTS    TRTEST         TEST OF TOPOLOGY (TRANSFORMERS)
C       TESTS    FETEST         TEST OF TOPOLOGY (FEEDERS)
C       TESTS    BRTEST         TEST OF TOPOLOGY (BRANCH POINTS)
C       TESTS    NOTEST         TEST OF TOPOLOGY (NODES)
C       TESTS    SWTEST         TEST OF TOPOLOGY (SWITCHES)
C       TESTS    NETTES         TEST OF NETWORK TOPOLOGY

C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        INCLUDE 'CONSTANTS.FOR'
        INCLUDE 'COMMON.FOR'

        INTEGER I,J,K,TYPE
        LOGICAL OK

        EXTERNAL LINES

        CALL LINES(3)
        DO 10 I=1,MAXNRT
```

```
      J=TRANEI(I)
      K=I
20    IF (CABTOP(J,1).NE.TRATYP .AND. CABTOP(J,2).NE.K) THEN
        IF (CABTOP(J,3).NE.TRATYP .AND. CABTOP(J,4).NE.K) THEN
          WRITE(*,*)'       FAULT IN NETWORK'
          CALL LINES(2)
          OK=.FALSE.
          WRITE(*,100) 'TRAFO NR',I,' IS CONNECTED TO CABLE NR',J
          CALL CABFOR(J)
30        WRITE(*,*)'       WHICH ONE IS THE WRONG ONE? TYPE?'
          READ(*,*) TYPE
          IF (TYPE.EQ.TRATYP) THEN
            WRITE(*,*) '       TRAFO CONNECTED TO WHICH CABLE?'
            READ(*,*) TRANEI(I)
          ELSEIF (TYPE.EQ.CABTYP) THEN
            CALL WRICAB(J)
          ELSE
            WRITE(*,*)'       WRONG TYPE'
            GOTO 30
          ENDIF
          GOTO 20
        ENDIF
      ENDIF
10  CONTINUE
100 FORMAT(' ',TR6,A,I3,A,I5)
    WRITE(*,*)'       TRAFOS IS OK'
    RETURN
    END


    SUBROUTINE FETEST(OK)

C   CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C   AUTHOUR MAGNUS BERGSTRAND

C   VERSION 1

C   DATE 1989-10-06

C   PURPOSE
C   TESTS THE TOPOLOGY REGARDING FEEDERS

C   DESCRIPTION
C   IF A POINTS NEIGHBOURS NEIGHBOUR IS NOT THE POINT ITSELF
C   A FAULT IN THE TOPOLOGY IS DETECTED

C   SEE ALSO
C   FILES    ROUTINES   COMMENTS
C   CONSTANTS           ALL CONSTANTS DECLARED
C   COMMON              ALL COMMON VARIABLES DECLARED
C   TESTS    TRTEST     TEST OF TOPOLOGY (TRANSFORMERS)
C   TESTS    FETEST     TEST OF TOPOLOGY (FEEDERS)
C   TESTS    BRTEST     TEST OF TOPOLOGY (BRANCH POINTS)
C   TESTS    NOTEST     TEST OF TOPOLOGY (NODES)
C   TESTS    SWTEST     TEST OF TOPOLOGY (SWITCHES)
C   TESTS    NETTES     TEST OF NETWORK TOPOLOGY

C   CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

    INCLUDE 'CONSTANTS.FOR'
    INCLUDE 'COMMON.FOR'

    LOGICAL OK
    INTEGER I,J,K,TY1,TY2,NR1,NR2,TYPE
```

```
      EXTERNAL LINES

      DO 10 I=1,MAXNRF
        DO 20 J=2,3
          K=MSFEED(I,J)
 30       IF (CABTOP(K,1).NE.FEETYP .AND. CABTOP(K,2).NE.I) THEN
            IF (CABTOP(K,3).NE.FEETYP .AND. CABTOP(K,4).NE.I) THEN
              TY1=CABTOP(K,1)
              TY2=CABTOP(K,3)
              NR1=CABTOP(K,2)
              NR2=CABTOP(K,4)
              OK=.FALSE.
              CALL LINES(3)
 40           WRITE(*,*)'       FAULTED NETWORK'
              WRITE(*,200)'FEEDER NR',I,'IS CONNECTED TO CABLE NR',K
              WRITE(*,*)
              CALL CABFOR(K)
              WRITE(*,*)
              WRITE(*,*)'       FAULT IN (TYPE) ?'
              READ(*,*) TYPE
              IF (TYPE.EQ.FEETYP) THEN
                WRITE(*,*) '       TO WHICH CABLE IS THE FEEDER CONNECTED?'
                READ(*,*) MSFEED(I,J)
              ELSEIF (TYPE.EQ.CABTYP) THEN
                CALL WRICAB(K)
              ELSE
                WRITE(*,*) '       WRONG TYPE!'
                GOTO 40
              ENDIF
              GOTO 30
            ENDIF
          ENDIF
 20     CONTINUE
 10   CONTINUE
 200  FORMAT(' ',TR6,A,I3,A,I4)
      WRITE(*,*) '       FEEDER IS OK'
      RETURN
      END



      SUBROUTINE BRTEST(OK)

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     AUTHOUR MAGNUS BERGSTRAND

C     VERSION 1

C     DATE 1989-10-06

C     PURPOSE
C     TESTS THE TOPOLOGY REGARDING BRANCH POINTS

C     DESCRIPTION
C     IF A POINTS NEIGHBOURS NEIGHBOUR IS NOT THE POINT ITSELF
C     A FAULT IN THE TOPOLOGY IS DETECTED

C     SEE ALSO
C     FILES    ROUTINES    COMMENTS
C     CONSTANTS            ALL CONSTANTS DECLARED
C     COMMON               ALL COMMON VARIABLES DECLARED
C     TESTS    TRTEST      TEST OF TOPOLOGY (TRANSFORMERS)
```

```
C       TESTS    FETEST      TEST OF TOPOLOGY (FEEDERS)
C       TESTS    BRTEST      TEST OF TOPOLOGY (BRANCH POINTS)
C       TESTS    NOTEST      TEST OF TOPOLOGY (NODES)
C       TESTS    SWTEST      TEST OF TOPOLOGY (SWITCHES)
C       TESTS    NETTES      TEST OF NETWORK TOPOLOGY

C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        INCLUDE 'CONSTANTS.FOR'
        INCLUDE 'COMMON.FOR'

        LOGICAL OK
        INTEGER I,J,K,L,TYPE

        EXTERNAL LINES

        DO 10 I=1,MAXNRB
          DO 20 J=3,(BRAPOI(I,1)+2)
 30         K=BRAPOI(I,J)
            IF (CABTOP(K,1).NE.BRATYP .AND. CABTOP(K,2).NE.I) THEN
              IF (CABTOP(K,3).NE.BRATYP .AND. CABTOP(K,4).NE.I) THEN
                OK=.FALSE.
                CALL LINES(3)
 40             WRITE(*,*)'      FAULTED NETWORK'
                WRITE(*,200)'BRANCH-POINT NR',I,' IS CONNECTED TO CABLE NR',K
                WRITE(*,*)
                CALL CABFOR(K)
                WRITE(*,*)'      FAULT IN BRANCH-POINT(4) OR IN CABLE(3) ?'
                READ(*,*) TYPE
                IF (TYPE.EQ.BRATYP) THEN
                  WRITE(*,*)'      THE FOLLOWING BRANCHES'
                  WRITE(*,*)
                  DO 50 L=3,(2+BRAPOI(I,1))
                    WRITE(*,*)'          ',BRAPOI(I,L)
 50               CONTINUE
                  WRITE(*,*)
                  WRITE(*,*)'      THE FAULTING ONE IS (NOT) CONNECTED TO',BRAPOI
                  WRITE(*,*)'      THE BRANCH IS CONNECTED TO CABLE NR ?'
                  READ(*,*) BRAPOI(I,J)
                ELSEIF(TYPE.EQ.CABTYP) THEN
                  CALL WRICAB(K)
                ELSE
                  WRITE(*,*)'     WRONG TYPE!'
                  GOTO 40
                ENDIF
                GOTO 30
              ENDIF
            ENDIF
 20       CONTINUE
 10     CONTINUE
 200    FORMAT(' ',TR6,A,I3,A,I4)
        WRITE(*,*)'       BRANCHPOINTS OK'
        RETURN
        END



        SUBROUTINE NOTEST(OK)

C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C       AUTHOUR MAGNUS BERGSTRAND

C       VERSION 1
```

```
C     DATE 1989-10-06

C     PURPOSE
C     TESTS THE TOPOLOGY REGARDING NODES

C     DESCRIPTION
C     IF A POINTS NEIGHBOURS NEIGHBOUR IS NOT THE POINT ITSELF
C     A FAULT IN THE TOPOLOGY IS DETECTED

C     SEE ALSO
C     FILES     ROUTINES    COMMENTS
C     CONSTANTS             ALL CONSTANTS DECLARED
C     COMMON                ALL COMMON VARIABLES DECLARED
C     TESTS     TRTEST      TEST OF TOPOLOGY (TRANSFORMERS)
C     TESTS     FETEST      TEST OF TOPOLOGY (FEEDERS)
C     TESTS     BRTEST      TEST OF TOPOLOGY (BRANCH POINTS)
C     TESTS     NOTEST      TEST OF TOPOLOGY (NODES)
C     TESTS     SWTEST      TEST OF TOPOLOGY (SWITCHES)
C     TESTS     NETTES      TEST OF NETWORK TOPOLOGY

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      INCLUDE 'CONSTANTS.FOR'
      INCLUDE 'COMMON.FOR'

      LOGICAL OK
      INTEGER I,J,K,TYPE

      EXTERNAL LINES

      DO 10 I=1,MAXNRN
        J=NODETO(I,2)
        K=NODETO(I,3)
        IF (.NOT. (J.EQ.0)) THEN
 20       IF (CABTOP(J,1).NE.NODTYP .AND. CABTOP(J,2).NE.I) THEN
            IF (CABTOP(J,3).NE.NODTYP .AND. CABTOP(J,4).NE.I) THEN
              OK=.FALSE.
              CALL LINES(3)
              WRITE(*,*)'       FAULTED NETWORK'
              WRITE(*,*)
              WRITE(*,200)'NODE NR',I,'IS CONNECTED TO CABLE NR',J
              WRITE(*,*)
              CALL CABFOR(J)
              WRITE(*,*)
              CALL CABFOR(K)
 30           WRITE(*,*)
              WRITE(*,*)'       FAULT IN NODE (5) OR IN CABLE (3)'
              READ(*,*) TYPE
              IF (TYPE.EQ.NODTYP) THEN
                WRITE(*,*)'       TO WHICH CABLE IS THE NODE CONNECTED?'
                READ(*,*) NODETO(I,2)
              ELSEIF (TYPE.EQ.CABTYP) THEN
                CALL WRICAB(J)
              ELSE
                WRITE(*,*)'       WRONG TYPE!'
                GOTO 30
              ENDIF
              GOTO 20
            ENDIF
          ENDIF
        ENDIF
        IF (.NOT. (K.EQ.0)) THEN
 40       IF (CABTOP(K,1).NE.NODTYP .AND. CABTOP(K,2).NE.I) THEN
            IF (CABTOP(K,3).NE.NODTYP .AND. CABTOP(K,4).NE.I) THEN
              OK=.FALSE.
              CALL LINES(3)
```

```
            WRITE(*,*)'      FAULTED NETWORK'
            WRITE(*,*)
            WRITE(*,200)'NODE NR',I,'IS CONNECTED TO CABLE NR',K
            WRITE(*,*)
            CALL CABFOR(J)
            WRITE(*,*)
            CALL CABFOR(K)
 50         WRITE(*,*)
            WRITE(*,*)'      FAULT IN NODE (5) OR IN CABLE (3)'
            READ(*,*) TYPE
            IF (TYPE.EQ.NODTYP) THEN
              WRITE(*,*)'      TO WHICH CABLE IS THE NODE CONNECTED?'
              READ(*,*) NODETO(I,3)
            ELSEIF (TYPE.EQ.CABTYP) THEN
              CALL WRICAB(K)
            ELSE
              WRITE(*,*)'      WRONG TYPE!'
              GOTO 50
            ENDIF
            GOTO 40
          ENDIF
        ENDIF
      ENDIF
 10   CONTINUE
 100  FORMAT('1')
 200  FORMAT(' ',TR6,A,I4,A,I4)
      WRITE(*,*)'      NODES OK'
      RETURN
      END



      SUBROUTINE SWICAB(A,OK)

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     AUTHOUR MAGNUS BERGSTRAND

C     VERSION 1

C     DATE 1989-10-06

C     PURPOSE
C     TESTS THE TOPOLOGY REGARDING 'A HALF SWITCH'

C     DESCRIPTION
C     IF A POINTS NEIGHBOURS NEIGHBOUR IS NOT THE POINT ITSELF
C     A FAULT IN THE TOPOLOGY IS DETECTED

C     SEE ALSO
C     FILES    ROUTINES   COMMENTS
C     CONSTANTS           ALL CONSTANTS DECLARED
C     COMMON              ALL COMMON VARIABLES DECLARED
C     TESTS    TRTEST     TEST OF TOPOLOGY (TRANSFORMERS)
C     TESTS    FETEST     TEST OF TOPOLOGY (FEEDERS)
C     TESTS    BRTEST     TEST OF TOPOLOGY (BRANCH POINTS)
C     TESTS    NOTEST     TEST OF TOPOLOGY (NODES)
C     TESTS    SWTEST     TEST OF TOPOLOGY (SWITCHES)
C     TESTS    NETTES     TEST OF NETWORK TOPOLOGY

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      INCLUDE 'CONSTANTS.FOR'
      INCLUDE 'COMMON.FOR'

      INTEGER A,I,J,TYPE
```

```
      LOGICAL OK

      EXTERNAL LINES

      DO 10 I=1,MAXNRS
        J=SWITCH(I,A)
 20     IF (CABTOP(J,1).NE.SWITYP .AND. CABTOP(J,2).NE.I) THEN
          IF (CABTOP(J,3).NE.SWITYP .AND. CABTOP(J,4).NE.I) THEN
            OK=.FALSE.
            CALL LINES(3)
 30         WRITE(*,*)'       FAULTED NETWORK'
            WRITE(*,*)
            WRITE(*,200)'SWITCH NR',I,'IS CONNECTED TO CABLE NR',J
            WRITE(*,*)
            CALL CABFOR(J)
            WRITE(*,*)
            WRITE(*,*)'       FAULT IN SWITCH (6) OR IN CABLE (3) ?'
            READ(*,*) TYPE
            IF (TYPE.EQ.SWITYP) THEN
              WRITE(*,*)'       TO WHICH CABLE IS THE SWITCH CONNECTED?'
              READ(*,*) SWITCH(I,A)
            ELSEIF (TYPE.EQ.CABTYP) THEN
              CALL WRICAB(J)
            ELSE
              WRITE(*,*)'       WRONG TYPE!'
              GOTO 30
            ENDIF
            GOTO 20
          ENDIF
        ENDIF
 10   CONTINUE
 200  FORMAT(' ',TR6,A,I4,A,I4)
      WRITE(*,*)'       SWITCH OK'
      RETURN
      END


      SUBROUTINE SWTEST(OK)

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     AUTHOUR MAGNUS BERGSTRAND

C     VERSION 1

C     DATE 1989-10-06

C     PURPOSE
C     TESTS THE TOPOLOGY REGARDING SWITCHES

C     DESCRIPTION
C     IF A POINTS NEIGHBOURS NEIGHBOUR IS NOT THE POINT ITSELF
C     A FAULT IN THE TOPOLOGY IS DETECTED

C     SEE ALSO
C     FILES     ROUTINES    COMMENTS
C     CONSTANTS             ALL CONSTANTS DECLARED
C     COMMON                ALL COMMON VARIABLES DECLARED
C     TESTS     TRTEST      TEST OF TOPOLOGY (TRANSFORMERS)
C     TESTS     FETEST      TEST OF TOPOLOGY (FEEDERS)
C     TESTS     BRTEST      TEST OF TOPOLOGY (BRANCH POINTS)
C     TESTS     NOTEST      TEST OF TOPOLOGY (NODES)
C     TESTS     SWTEST      TEST OF TOPOLOGY (SWITCHES)
C     TESTS     NETTES      TEST OF NETWORK TOPOLOGY

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

```
      INCLUDE 'CONSTANTS.FOR'
      INCLUDE 'COMMON.FOR'

      LOGICAL OK

      CALL SWICAB(3,OK)
      CALL SWICAB(4,OK)
      RETURN
      END



      SUBROUTINE NETTES(CH2)

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     AUTHOUR MAGNUS BERGSTRAND

C     VERSION 1

C     DATE 1989-10-06

C     PURPOSE
C     TEST THE NETWORK TOPOLOGY

C     DESCRIPTION
C     ANY FAULLT THAT WILL BE DETECTED WILL BE CORRECTED

C     SEE ALSO
C     FILES    ROUTINES    COMMENTS
C     CONSTANTS            ALL CONSTANTS DECLARED
C     COMMON               ALL COMMON VARIABLES DECLARED
C     TESTS    TRTEST      TEST OF TOPOLOGY (TRANSFORMERS)
C     TESTS    FETEST      TEST OF TOPOLOGY (FEEDERS)
C     TESTS    BRTEST      TEST OF TOPOLOGY (BRANCH POINTS)
C     TESTS    NOTEST      TEST OF TOPOLOGY (NODES)
C     TESTS    SWTEST      TEST OF TOPOLOGY (SWITCHES)

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      CHARACTER*7 CH2
      LOGICAL OK

      EXTERNAL LINES

 10   OK=.TRUE.
      CALL TRTEST(OK)
      CALL FETEST(OK)
      CALL BRTEST(OK)
      CALL NOTEST(OK)
      CALL SWTEST(OK)
      IF (.NOT. OK) THEN
        CALL WRITEF(CH2)
        GOTO 10
      ENDIF
      CALL LINES(3)
      WRITE(*,*)'        THE NETWORK IS CORRECT'
      RETURN
      END



      SUBROUTINE TKESTP(POINT,OLDP,DIR,BRA,WR)

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

```
C     AUTHOUR MAGNUS BERGSTRAND

C     VERSION 1

C     DATE 1989-10-06

C     PURPOSE
C     CAN BE USED TO TRAVERSE A NETWORK IN COOPERATION WITH NEWDIR

C     DESCRIPTION
C     THIS SUBROUTINE IS ONLY TO BE CALLED FROM INIT
C     TKESTP MANAGES TO TRAVERSE AN UNINITIATED NETWORK
C     IT INITIATES THE NETWORK AT THE SAME TIME

C     SEE ALSO
C     FILES    ROUTINES    COMMENTS
C     CONSTANTS            ALL CONSTANTS DECLARED
C     COMMON               ALL COMMON VARIABLES DECLARED
C     MOVING   STEPUP      TAKES ONE STEP UP
C     MOVING   STEPDO      TAKES ONE STEP DOWN
C     MOVING   MOVE        TAKES ONE STEP UP OR DOWN (IN AN INITIATED NETWORK)
C     MOVING   DIREC       COMPUTES NEW DIRECTION
C     TESTS    NEWDIR      COMPUTES NEW DIRECTION. IS INTENDED TO COOPERATE
C                          WITH TKESTP

C     VARIABLES
C     POINT IS THE CURRENT POINT. (TYPE,NR)
C     OLDP IS THE FORMER POINT. (TYPE,NR)
C     DIR IS THE DIRECTION (UP OR DOWN)
C     BRA IS THE NUMBER OF BRANCH IN CASE OF A BRANCH POINT
C     IF WR=TRUE THEN TEXT IS WRITTEN ON THE SCREEN INDICATING THE CURRENT
C     POINT

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      INCLUDE 'CONSTANTS.FOR'
      INCLUDE 'COMMON.FOR'

      INTEGER POINT(2),OLDP(2),DIR,BRA
      INTEGER IN,I,J
      LOGICAL WR

      IF (WR) THEN
        WRITE(*,*)
        WRITE(*,*)'        INDATA TKESTP (POINT,OLDP,DIR,BRA)'
        WRITE(*,*)POINT(1),POINT(2),OLDP(1),OLDP(2),DIR,BRA
      ENDIF
      IF (DIR.EQ.UP) THEN
        OLDP(1)=POINT(1)
        OLDP(2)=POINT(2)
        CALL STEPUP(POINT,WR)

C       ONE STEP UP WAS TAKEN

        IF (POINT(1).EQ.BRATYP) THEN

C         FIND OUT IN WHICH BRANCH OLDP IS.

          DO 20 J=3,(2+BRAPOI(POINT(2),1))
            I=BRAPOI(POINT(2),J)
            IF (CABTOP(I,1).EQ.OLDP(1) .AND. CABTOP(I,2).EQ.OLDP(2)) THEN
              BRA=J
            ELSEIF (CABTOP(I,3).EQ.OLDP(1) .AND. CABTOP(I,4).EQ.OLDP(2)) THEN
              BRA=J
            ENDIF
```

```
 20         CONTINUE
         ENDIF
      ELSEIF (DIR.EQ.DOWN) THEN
         IF (POINT(1).EQ.BRATYP) THEN
            I=POINT(2)
            IF (BRAPOI(I,2).EQ.0) THEN

C           'UP' IS NOT KNOWN
C           UP IS IN THE DIRECTION OF OLDP!

               DO 30 J=3,BRAPOI(I,1)+2
                  K=BRAPOI(I,J)
                  IF (CABTOP(K,1).EQ.OLDP(1) .AND. CABTOP(K,2).EQ.OLDP(2)) THEN
                     BRAPOI(I,2)=J
                  ELSEIF (CABTOP(K,3).EQ.OLDP(1) .AND. CABTOP(K,4).EQ.OLDP(2)) THEN
                     BRAPOI(I,2)=J
                  ENDIF
 30            CONTINUE
               IF (BRAPOI(I,2).EQ.3) THEN

C                 SINCE THE DIRECTION IS DOWN

                  BRA=4
               ELSE
                  BRA=3
               ENDIF
            ENDIF
         ENDIF
         OLDP(1)=POINT(1)
         OLDP(2)=POINT(2)
         CALL STEPDO(POINT,BRA,WR)

C        ONE STEP DOWN WAS TAKEN
C        HEREAFTER UP WILL ASSIGNED A VALUE IN CASE OF SWITCH OR NODE

         IF (POINT(1).EQ.SWITYP) THEN
            J=POINT(2)
            I=SWITCH(J,3)
            IF (CABTOP(I,1).EQ.OLDP(1) .AND. CABTOP(I,2).EQ.OLDP(2)) THEN
               SWITCH(J,2)=3
            ELSEIF(CABTOP(I,3).EQ.OLDP(1) .AND. CABTOP(I,4).EQ.OLDP(2)) THEN
               SWITCH(J,2)=3
            ELSE
               SWITCH(J,2)=4
            ENDIF
         ELSEIF (POINT(1).EQ.NODTYP) THEN
            I=NODETO(POINT(2),2)
            IF (CABTOP(I,1).EQ.OLDP(1) .AND. CABTOP(I,2).EQ.OLDP(2)) THEN
               NODETO(POINT(2),1)=2
            ELSEIF(CABTOP(I,3).EQ.OLDP(1) .AND. CABTOP(I,4).EQ.OLDP(2)) THEN
               NODETO(POINT(2),1)=2
            ELSE
               NODETO(POINT(2),1)=3
            ENDIF
         ENDIF
      ENDIF
      IF (WR) THEN
         WRITE(*,*)
         WRITE(*,*)'        OUTDATA TKESTP'
         WRITE(*,*)POINT(1),POINT(2),OLDP(1),OLDP(2),DIR,BRA
      ENDIF
      RETURN
      END


      SUBROUTINE NEWDIR(POINT,DIR,BRA,WR)
```

```
C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     AUTHOUR MAGNUS BERGSTRAND

C     VERSION 1

C     DATE 1989-10-06

C     PURPOSE
C     COMPUTES THE NEW DIRECTION.

C     DESCRIPTION
C     IS USED IN COOPERATION WITH TKESTP IN ORDER TO TRAVERSE AN
C     UNINTIALIZED NETWORK

C     SEE ALSO
C     FILES     ROUTINES    COMMENTS
C     CONSTANTS             ALL CONSTANTS DECLARED
C     COMMON                ALL COMMON VARIABLES DECLARED
C     MOVING    STEPUP      TAKES ONE STEP UP
C     MOVING    STEPDO      TAKES ONE STEP DOWN
C     MOVING    MOVE        TAKES ONE STEP UP OR DOWN (IN AN INITIATED NETWORK)
C     MOVING    DIREC       COMPUTES NEW DIRECTION
C     TESTS     TKESTP      TAKES ONE STEP UP OR DOWN. IS INTENDED TO COOPERATE
C                           WITH NEWDIR

C     VARIABLES
C     POINT IS THE CURRENT POINT. (TYPE,NR)
C     DIR IS THE DIRECTION (UP OR DOWN)
C     BRA IS THE NUMBER OF BRANCH IN CASE OF A BRANCH POINT
C     IF WR=TRUE THEN TEXT IS WRITTEN ON THE SCREEN INDICATING THE CURRENT
C     POINT

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      INCLUDE 'CONSTANTS.FOR'
      INCLUDE 'COMMON.FOR'

      INTEGER POINT(2)
      INTEGER DIR,BRA
      INTEGER I
      LOGICAL WR

      IF (WR) THEN
        WRITE(*,*)
        WRITE(*,*)'        INDATA NEWDIR (POINT,DIR,BRA)'
        WRITE(*,*) POINT(1),POINT(2),DIR,BRA
      ENDIF
      IF (POINT(1).EQ.BRATYP) THEN
        IF (DIR.EQ.UP) THEN

C         IN THIS CASE THE ONLY POSSIBLE CHANGE OF DIRECTION IS WHENEVER
C         A BRANCH POINT IS REACHED. THEN THE NEW DIRECTION MAY BE DOWN
C         IN THE NEXT BRANCH

          I=POINT(2)
          IF (BRAPOI(I,1).GT.(BRA-2) .AND. BRAPOI(I,2).NE.(BRA+1)) THEN
            BRA=BRA+1
            DIR=DOWN
          ELSEIF (BRAPOI(I,2).EQ.(BRA+1)) THEN
            IF (BRA+2.LE.2+BRAPOI(I,1)) THEN
              BRA=BRA+2
              DIR=DOWN
            ENDIF
          ENDIF
```

```
        ENDIF
      ELSEIF (POINT(1).EQ.SWITYP) THEN

C         THE DIRECTION IS DOWN

C         IF AN OPEN SWITCH IS FOUND CHANGE DIRECTION

        IF (SWITCH(POINT(2),1).EQ.OPEN) THEN
          DIR=UP
        ENDIF
      ELSEIF (POINT(1).EQ.NODTYP) THEN

C         IF A NODE IS FOUND AND THIS NODE HAS ONLY ONE FEEDING CABLE
C         CHANGE DIRECTION

        IF (NODETO(POINT(2),2).EQ.0) THEN
          DIR=UP
        ELSEIF (NODETO(POINT(2),3).EQ.0) THEN
          DIR=UP
        ENDIF
      ENDIF
      IF (WR) THEN
        WRITE(*,*)
        WRITE(*,*)'       OUTDATA NEWDIR (POINT,DIR,BRA)'
        WRITE(*,*) POINT(1),POINT(2),DIR,BRA
      ENDIF
      RETURN
      END



      SUBROUTINE FILLIN(POINT,DIR,BRA,TRA,FEE)

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     AUTHOUR MAGNUS BERGSTRAND

C     VERSION 1

C     DATE 1989-10-06

C     PURPOSE
C     ASSIGNS FEEDER AND TRANSFORMER IDENTITY.
C     COMPUTES IMPEDANCES.

C     DESCRIPTION
C     IS ONLY TO BE CALLED FROM INIT
C     THE IMPEDANCE WILL BE COMPUTED BY A CALL OF UPDATE

C     SEE ALSO
C     FILES   ROUTINES   COMMENTS
C     CONSTANTS          ALL CONSTANTS DECLARED
C     COMMON             ALL COMMON VARIABLES DECLARED
C     TESTS   INIT       INITIATES THE NETWORK
C     TESTS   UPDATE     COMPUTES IMPEDANCE
C     MOVING  STEPUP     TAKES ONE STEP UP

C     VARIABLES
C     POINT IS THE CURRENT POINT (TYPE,NUMBER)
C     DIR IS THE DIRECTION (UP OR DOWN)
C     BRA IS THE NUMBER OF BRANCH
C     TRA IS THE TRANSFORMER IDENTITY
C     FEE IS THE FEEDER IDENTITY

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

```
      INCLUDE 'CONSTANTS.FOR'
      INCLUDE 'COMMON.FOR'

      INTEGER POINT(2),P(2)
      INTEGER DIR,BRA,TRA,FEE
      INTEGER OLD(2)
      REAL CUR
      LOGICAL FIRST

      IF (DIR.EQ.DOWN) THEN
         IF (POINT(1).EQ.FEETYP) THEN
            FEE=POINT(2)
            MSFEED(POINT(2),1)=TRA
         ELSEIF (POINT(1).EQ.SWITYP) THEN
            SWITCH(POINT(2),5)=1
         ELSEIF (POINT(1).EQ.NODTYP) THEN
            I=POINT(2)
            NODETO(I,4)=FEE
            CUR=(NODEPH(I,1)**2+NODEPH(I,2)**2)/(NODEPH(I,4)**2+NODEPH(I,5)**2)
            FIRST=.TRUE.
            P(1)=POINT(1)
            P(2)=POINT(2)
            OLD(1)=0
            OLD(2)=0
 10         IF (P(1).NE.TRATYP) THEN
               CALL UPDATE(P,OLD,FIRST,CUR,NODEPH(POINT(2),4),NODEPH(POINT(2),5))
               OLD(1)=P(1)
               OLD(2)=P(2)
               CALL STEPUP(P,.FALSE.)
               GOTO 10
            ENDIF
            CALL UPDATE(P,OLD,FIRST,CUR,1.0,1.0)
         ENDIF
      ENDIF
      RETURN
      END



      SUBROUTINE UPDATE(POINT,OLD,FIRST,CUR,NODER,NODEX)

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     AUTHOUR MAGNUS BERGSTRAND

C     VERSION 1

C     DATE 1989-10-06

C     PURPOSE
C     COMPUTES IMPEDANCES

C     DESCRIPTION
C     COMPUTES IMPEDANCES WHILE MOVING FROM A NODE TO A TRANSFORMER
C     COMPUTATIONS ARE BASED ON THE FACT THAT THE NETWORK HAS A RADIAL
C     STRUCTURE.
C     THE RESULT Z=(RES,j*REA) ARE SAVED BETWEEN CALLS
C     THE PARALLELL IMPEDANCE BETWEEN THE OLD VALUE AND A NEW NODE IS COMPUTED
C     THEN IS THE CABLE IMPEDANCE ADDED

C     SEE ALSO
C     FILES     ROUTINES    COMMENTS
C     CONSTANTS             ALL CONSTANTS DECLARED
C     COMMON                ALL COMMON VARIABLES DECLARED
C     MOVING    STEPUP      TAKES ONE STEP UP
C     COMP      COMPIM      COMPUTES IMPEDANCE. INITIATED NETWORK
```

```
C       COMP    BRAPAR     COMPUTES EQUIVALENT PARALLELL IMPEDANCE FOR A BRANCH
C                          POINT
C       COMP    PARALL     COMPUTES EQUIVALENT PARALLELL IMPEDANCE

C       VARIABLES
C       POINT IS THE CURRENT POINT (TYPE,NR)
C       OLD IS THE FORMER POINT (TYPE,NR)
C       FIRST INDICATES FIRST CALL.
C       CUR IS THE CURRENT (AMPERE)
C       NODER IS NODE RESISTANCE (OHM)
C       NODEX IS NODE REACTANCE (OHM)

C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        INCLUDE 'CONSTANTS.FOR'
        INCLUDE 'COMMON.FOR'

        INTEGER POINT(2),OLD(2)
        INTEGER I,J,K
        LOGICAL FIRST
        REAL NODER,NODEX,CUR,F(2),G(2)
        REAL RES,REA,SUMR,SUMX

        SAVE RES,REA

        IF (FIRST) THEN
          RES=NODEPH(POINT(2),4)
          REA=NODEPH(POINT(2),5)
          FIRST=.FALSE.
        ELSEIF (.NOT. FIRST) THEN
          IF (POINT(1).EQ.TRATYP) THEN
            TRAFOS(POINT(2),6)=RES
            TRAFOS(POINT(2),7)=REA
          ELSEIF (POINT(1).EQ.FEETYP) THEN
            I=MSFEED(POINT(2),3)
            RES=RES+CABPHY(I,1)
            REA=REA+CABPHY(I,2)
          ELSEIF (POINT(1).EQ.BRATYP) THEN
            I=POINT(2)
            DO 30 J=3,(BRAPOI(I,1)+2)
              K=BRAPOI(I,J)
              IF (CABTOP(K,1).EQ.OLD(1) .AND. CABTOP(K,2).EQ.OLD(2)) THEN
                BRAPH(I,J,1)=RES
                BRAPH(I,J,2)=REA
              ELSEIF (CABTOP(K,3).EQ.OLD(1) .AND. CABTOP(K,4).EQ.OLD(2)) THEN
                BRAPH(I,J,1)=RES
                BRAPH(I,J,2)=REA
              ENDIF
 30         CONTINUE
            CALL BRAPAR(POINT(2),RES,REA)
            K=BRAPOI(POINT(2),2)
            K=BRAPOI(POINT(2),K)
            RES=CABPHY(K,1)+RES
            REA=CABPHY(K,2)+REA
          ELSEIF (POINT(1).EQ.NODTYP) THEN
            F(1)=RES
            F(2)=REA
            G(1)=NODEPH(POINT(2),4)
            G(2)=NODEPH(POINT(2),5)
            CALL PARALL(F,G,RES,REA)
            I=NODETO(POINT(2),1)
            I=NODETO(POINT(2),I)
            RES=CABPHY(I,1)+RES
            REA=CABPHY(I,1)+REA
          ELSEIF (POINT(1).EQ.SWITYP) THEN
            I=SWITCH(POINT(2),2)
```

```
        I=SWITCH(POINT(2),I)
        RES=RES+CABPHY(I,1)
        REA=REA+CABPHY(I,2)
      ELSE
        WRITE(*,*)'        WRONG TYPE IN UPDATE!'
        WRITE(*,*)'        THE POINT IS',POINT(1),POINT(2)
        STOP
      ENDIF
    ENDIF
    RETURN
    END

    SUBROUTINE INIT(WR)

C   CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C   AUTHOUR MAGNUS BERGSTRAND

C   VERSION 1

C   DATE 1989-10-06

C   PURPOSE
C   INITIATES THE NETWORK

C   DESCRIPTION
C   FIRST ALL PARALLELL IMPEDANCES IN BRANCH POINTS ARE ASSIGNED A
C   LARGE VALUE
C   THEN FOR EACH TRANSFORMER THE TOPOLOGY IS INITIATED AND IMPEDANES COMPUTE
C   BY A CALL OF FILLIN

C   SEE ALSO
C   FILES     ROUTINES    COMMENTS
C   CONSTANTS             ALL CONSTANTS DECLARED
C   COMMON                ALL COMMON VARIABLES DECLARED
C   TESTS     FILLIN      INITIATE THE NETWORK AND COMPUTES IMPEDANCES
C   TESTS     TKESTP      TAKES A STEP UP OR DOWN IN A NONINITIATED NETWORK
C   TESTS     NEWDIR      COMPUTES NEW DIRECTION IN A NONINITIATED NETWORK
C   MOVING    STEPDO      TAKES ONE STEP DOWN

C   VARIABLES
C   IF 'WR'=TRUE THEN TEXT SHOWING THE CURRENT POINT WILL BE WRITTEN
C   ON THE SCREEN

C   CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

    INCLUDE 'CONSTANTS.FOR'
    INCLUDE 'COMMON.FOR'

    INTEGER POINT(2),OLDP(2),DIR,BRA
    INTEGER TRA,FEE
    INTEGER I,J,K
    LOGICAL WR


    DO 10 I=1,MAXNRB
      BRAPOI(I,2)=0
      DO 20 J=3,(2+BRAPOI(I,1))
        BRAPH(I,J,1)=1.0E+20
        BRAPH(I,J,2)=0.0
20    CONTINUE
10  CONTINUE
    DO 30 I=1,MAXNRS
      CHGSWI(I)=.FALSE.
30  CONTINUE
    DO 50 I=1,MAXNRT
```

```
      TRA=I
      TRAFOS(TRA,6)=1.001E+06
      TRAFOS(TRA,7)=1.001E+06
      POINT(1)=TRATYP
      POINT(2)=TRA
      OLDP(1)=TRATYP
      OLDP(2)=TRA
      DIR=DOWN
      BRA=4
      CALL STEPDO(POINT,BRA,WR)
      IF (POINT(1).EQ.SWITYP) THEN
        K=SWITCH(POINT(2),3)
        IF (CABTOP(K,1).EQ.OLDP(1) .AND. CABTOP(K,2).EQ.OLDP(2)) THEN
          SWITCH(POINT(2),2)=3
        ELSEIF (CABTOP(K,3).EQ.OLDP(1) .AND. CABTOP(K,4).EQ.OLDP(2)) THEN
          SWITCH(POINT(2),2)=3
        ELSE
          SWITCH(POINT(2),2)=4
        ENDIF
      ELSE
        WRITE(*,*)'      DEGENERATED NET.SWITCH EXPECTED'
        STOP
      ENDIF
      CALL FILLIN(POINT,DIR,BRA,TRA,FEE)
      CALL NEWDIR(POINT,DIR,BRA,WR)
60    IF (.NOT. (POINT(1).EQ.TRATYP .AND. POINT(2).EQ.TRA)) THEN
        CALL TKESTP(POINT,OLDP,DIR,BRA,WR)
        CALL FILLIN(POINT,DIR,BRA,TRA,FEE)
        CALL NEWDIR(POINT,DIR,BRA,WR)
        GOTO 60
      ENDIF
50  CONTINUE
    WRITE(*,*)'       INIT PASSED'
    RETURN
    END
```

```
C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                    COMP.FOR
C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC



      SUBROUTINE CON(POINT,OK)
C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C     AUTHOUR MAGNUS BERGSTRAND
C     VERSION 1
C     DATE 1989-09-30
C     PURPOSE:
C     CONNECTS A GROUP OF SECTIONS TO THE FEEDER
C     WITH THE LARGEST MARGIN OF GETTING VIOLATED
C     DESCRIPTION:
C     STARTS A SEARCH FROM THE ROOT OF A TREE AND
C     EXAMINES ALL POSSIBLE LOAD TRANSFERS. WHEN
C     THE BEST SWITCH FOR LOAD TRANSFER IS FOUND
C     THIS SWITCH IS OPENED
C     IF HOWEVER NO POSSIBLE CONNECTION IS AVAILABLE
C     OK EQUALS FALSE.
C     VARIABLES:
C     POINT IS THE ROOT OF THE TREE (TYPE,NR)
C     SEE ALSO:
C     FILE        ROUTINE     COMMENT

C     CONSTANTS               ALL CONSTANTS DECLARED
C     COMMON                  ALL COMMON VARIABLES DECLARED
C     FETCH       GETPOI      GETS A POINT ON THE OTHER SIDE OF SWITCH
C     FETCH       GETVOL      GETS THE VOLTAGE ON THE OTHER SIDE
C     FETCH       GETFEE      GETS THE NUMBER OF THE FEEDER ON THE OTHER
C                             SIDE OF THE SWITCH
C     FETCH       GETPLI      GETS THE TRANSFORMERS MARGIN OF GETTING
C                             VIOLATED REGARDING ACTIVE POWER
C     FETCH       GETQLI      GETS THE TRANSFORMERS MARGIN OF GETTING
C                             VIOLATED REGARDING REACTIVE POWER
C     MOVING      MOVE        TAKES ONE STEP UP OR DOWN
C     MOVING      DIREC       COMPUTES NEW DIRECTION
C     COMP        FORBAC      CHANGES DIRECTIONS IN THE TOPOLOGY DUE
C                             TO THE LOAD TRANSFER
C     COMP        COMPIM      COMPUTES NEW IMPEDANCE FOR A FEEDER
C     COMP        FILPOW      COMPUTES NEW POWER FOR A TRANSFORMER
C     COMP        FILVOL      COMPUTES ALL NODE VOLTAGES AND CABLE
C                             CURRENTS FOR A FEEDER
C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      INCLUDE 'CONSTANTS.FOR'
      INCLUDE 'COMMON.FOR'

      INTEGER POINT(2),P1(2),OLD(2)
      LOGICAL READY,NEG
      INTEGER FEEDER,GETFEE,FEE
```

```
      INTEGER TRAFO,DIR,BRA,SWI
      LOGICAL VIO
      REAL BEST,TEMP,VOL,P,Q
      REAL ABS,GETVOL
      REAL GETPLI,GETQLI
      LOGICAL FOUND,VAL,OK

      EXTERNAL GETPOI,GETVOL,GETFEE
      EXTERNAL GETPLI,GETQLI
      EXTERNAL MOVE,DIREC

      VAL=.FALSE.
      P1(1)=POINT(1)
      P1(2)=POINT(2)
      READY=.FALSE.
      DIR=DOWN
      IF (P1(1).EQ.BRATYP) THEN
        IF (BRAPOI(P1(1),2).EQ.3) THEN
          BRA=4
        ELSE
          BRA=3
        ENDIF
      ENDIF
      BEST=-1.0E38
      IF (P1(1).EQ.NODTYP) THEN
        IF (NODETO(P1(2),2).EQ.0 .OR. NODETO(P1(2),3).EQ.0) THEN
          GOTO 20
        ENDIF
      ENDIF
 10   IF (.NOT. READY) THEN
        IF (P1(1).EQ.NODTYP .AND. DIR.EQ.DOWN) THEN

C     ADDS UP THE MAGNITUDE OF ISOLATED LOAD

          ISOLAT(1)=ISOLAT(1)+NODEPH(P1(2),1)
          ISOLAT(2)=ISOLAT(2)+NODEPH(P1(2),2)
        ENDIF
        CALL MOVE(P1,OLD,DIR,BRA,.FALSE.)
        CALL DIREC(P1,OLD,DIR,BRA)
        IF (P1(1).EQ.SWITYP) THEN
          IF (SWITCH(P1(2),1).EQ.OPEN) THEN
            CALL REALFE(P1(2),FEE,FOUND)
            IF (FOUND) THEN

C     A FEEDER THAT CAN TAKE UP LOAD IS FOUND
C     IT IS POSSIBLE TO MAKE A CONNECTION
C     BELOW FOLLOWS COMPUTATIONS IN ORDER TO
C     DECIDE THE BEST CONNECTION

              VAL=.TRUE.
              VOL=GETVOL(P1(2))
              P=GETPLI(P1(2))
              Q=GETQLI(P1(2))
              NEG=(VOL.LT.0.0 .OR. P.LT.0.0)
              NEG=(NEG .OR. Q.LT.0.0)
              IF (NEG) THEN
                TEMP=-(ABS(VOL)+ABS(P)+ABS(Q))
              ELSE
                TEMP=VOL+P+Q
              ENDIF
              IF (TEMP.GT.BEST) THEN

C      THE BEST CONNECTION IS STORED FOR LATER USE

                BEST=TEMP
                FEEDER=FEE
```

```
              SWI=P1(2)
            ENDIF
          ENDIF
        ENDIF
      ENDIF
      READY=(P1(1).EQ.POINT(1) .AND. P1(2).EQ.POINT(2) .AND. DIR.EQ.UP)
      GOTO 10
    ENDIF
    OK=VAL
 20 IF (VAL) THEN

C   A CONNECTION IS FOUND AND THE BEST SWITCHING OPERATION IS PERFORMED

      SWINR(SWI)=4
      SWITCH(SWI,1)=CLOSED
      CHGSWI(SWI)=.NOT. CHGSWI(SWI)
      TRAFO=MSFEED(FEEDER,1)
      CALL FORBAC(SWI,TRAFO,FEEDER)
      CALL COMPIM(FEEDER)
      CALL FILPOW(TRAFO)
      CALL FILVOL(FEEDER,VIO)
    ELSE

C   NO POSSIBLE CONNECTION EXIST AND NO LOAD WAS RESTORED

      WRITE(*,*)'       IT IS NOT POSSIBLE TO MAKE ANY CONNECTION'
      RESTOR(1)=0.0
      RESTOR(2)=0.0
    ENDIF
    RETURN
    END

    SUBROUTINE UDAT(SWI,TRAFO,FEEDER)

C   CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C   AUTHOUR MAGNUS BERGSTRAND

C   VERSION 1

C   DATE 1989-09-30

C   PURPOSE:
C   FILLS IN FEEDER AND TRANSFORMER IDENTITY IN PROPER PLACES
C   WHEN A LOAD TRANSFER HAS CHANGED THE TOPOLOGY

C   DESCRIPTION:
C   IS ONLY TO BE CALLED BY SUBROUTINE FORBAC
C   SIMPLE ASSIGNEMENT OF VARIABLES TRAFO AND FEEDER

C   VARIABLES:
C   TRAFO,INTEGER,THE NUMBER OF THE 'NEW' TRANSFORMER OF THE FEEDER
C   FEEDER,INTEGER, THE NUMBER OF THE 'NEW' FEEDER A NODE BELONGS TO

C   SEE ALSO:

C   FILE        ROUTINE     COMMENT
C   CONSTANTS               ALL CONSTANTS DECLARED
C   COMMON                  ALL COMMON VARIABLES DECLARED
C   COMP        FORBAC      CHANGES THE 'DIRECTIONS' OF THE TOPOLOGY
C                           DUE TO A LOAD TRANSFER
C   MOVING      MOVE        TAKES ONE STEP UP OR DOWN
C   MOVING      DIREC       COMPUTES NEW DIRECTION

C   CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

```
      INCLUDE 'CONSTANTS.FOR'
      INCLUDE 'COMMON.FOR'

      INTEGER P(2),OLD(2)
      LOGICAL READY
      INTEGER DIR,BRA

      P(1)=SWITYP
      P(2)=SWI
      READY=.FALSE.
      DIR=DOWN
      BRA=3
 10   IF (.NOT. READY) THEN
        CALL MOVE(P,OLD,DIR,BRA,.FALSE.)
        CALL DIREC(P,OLD,DIR,BRA)
        IF (DIR.EQ.UP) THEN
          IF (P(1).EQ.NODTYP) THEN
            NODETO(P(2),4)=FEEDER
          ELSEIF (P(1).EQ.FEETYP) THEN
            MSFEED(P(2),1)=TRAFO
          ENDIF
        ENDIF
        READY=(P(1).EQ.SWITYP .AND. P(2).EQ.SWI)
        GOTO 10
      ENDIF
      RETURN
      END


      SUBROUTINE FORBAC(SWI,TRAFO,FEEDER)

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     AUTHOUR MAGNUS BERGSTRAND

C     VERSION 1

C     DATE 1989-09-30

C     PURPOSE:
C     CHANGES DIRECTIONS.
C     WHEN A LOAD TRANSFER IS MADE THE FORMER DIRECTION 'DOWN'
C     WILL BECOME 'UP'

C     DESCRIPTION:
C     STARTING AT THE SWITCH THAT WILL BE OPENED THIS ROUTINE
C     MOVES TOWARDS THE SWITCH THAT WILL BE CLOSED CHANGING
C     DIRECTIONS. THE ACTUAL CHANGE IN POINTERS IS DONE BY A
C     CALL OF SUBROUTINE CHGDIR

C     VARIABLES:
C     SWI IS THE NUMBER OF THE SWITCH THAT WILL BE OPENED
C     TRAFO IS THE NUMBER OF THE 'NEW' TRANSFORMER THAT A
C     FEEDER POSSIBLY WILL BE ATTACHED TO
C     FEEDER IS THE NUMBER OF THE 'NEW' FEEDER THAT A NODE
C     WILL BE ATTACHED TO

C     SEE ALSO:
C     FILE          ROUTINE      COMMENT

C     CONSTANTS                  ALL CONSTANTS DECLARED
C     COMMON                     ALL COMMON VARIABLES DECLARED
C     COMP          UDAT         ASSIGNS NEW VALUES OF TRAFO,FEEDER
C     COMP          CHGDIR       CHANGES POINTERS DUE TO A LOAD TRANSFER
C     MOVING        STEPUP       TAKES ONE STEP UP
C     MOVING        STEPDO       TAKES ONE STEP DOWN
```

```
C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      INCLUDE 'CONSTANTS.FOR'
      INCLUDE 'COMMON.FOR'

      INTEGER TRAFO,FEEDER,I
      INTEGER SWI,DIR,BRA
      INTEGER POI(2),OLD(2)
      LOGICAL READY

      POI(1)=SWITYP
      POI(2)=SWI
      OLD(1)=SWITYP
      OLD(2)=SWI
      READY=.FALSE.
      CALL STEPUP(POI,.FALSE.)
      I=SWITCH(SWI,3)
      IF (CABTOP(I,1).EQ.POI(1) .AND. CABTOP(I,2).EQ.POI(2)) THEN
        SWITCH(SWI,2)=4
      ELSEIF (CABTOP(I,3).EQ.POI(1) .AND. CABTOP(I,4).EQ.POI(2)) THEN
        SWITCH(SWI,2)=4
      ELSE
        SWITCH(SWI,2)=3
      ENDIF
 10   IF (.NOT. READY) THEN
        CALL CHGDIR(POI,OLD,BRA,TRAFO,FEEDER)
        OLD(1)=POI(1)
        OLD(2)=POI(2)
        CALL STEPDO(POI,BRA,.FALSE.)
        IF (POI(1).EQ.SWITYP) THEN
          IF (SWITCH(POI(2),1).EQ.OPEN) THEN
            READY=.TRUE.
            I=SWITCH(POI(2),3)
            IF (CABTOP(I,1).EQ.OLD(1) .AND. CABTOP(I,2).EQ.OLD(2)) THEN
              SWITCH(POI(2),2)=3
            ELSEIF (CABTOP(I,3).EQ.OLD(1) .AND. CABTOP(I,4).EQ.OLD(2)) THEN
              SWITCH(POI(2),2)=3
            ELSE
              SWITCH(POI(2),2)=4
            ENDIF
            I=SWITCH(POI(2),2)
            I=SWITCH(POI(2),I)
          ENDIF
        ENDIF
        GOTO 10
      ENDIF
      CALL UDAT(SWI,TRAFO,FEEDER)
      RETURN
      END


      SUBROUTINE CHGDIR(POI,OLD,BRA,TRAFO,FEEDER)

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     AUTHOUR MAGNUS BERGSTRAND

C     VERSION 1

C     DATE 1989-09-30

C     PURPOSE:
C     WHEN A LOAD TRANSFER IS PERFORMED CHANGES MUST BE MADE IN THE
C     TOPOLOGY. THIS SUBROUTINE CHANGES POINTER ACCORDING TO THE
C     NEW TOPOLOGY
```

```
C       DESCRIPTION:
C       IT IS ONLY TO BE CALLED BY SUBROUTINE FORBAC
C       STANDING AT THE POINT POI CHANGES IS MADE ACCORDING TO THE TYPE
C       OF THIS POINT.

C       VARIABLES:
C       POI IS THE CURRENT POINT IN THE NETWORK
C       OLD IS THE PRVIOUSLY VISITED POINT IN THE NETWORK
C       BRA IS AN OUTPARAMETER. IT INDICATES THE OLD VALUE OF UP.
C       TRAFO IS THE NUMBER OF THE TRANSFORMER WHICH ALL OTHER
C       ELEMENTS ARE ATTACHED TO.
C       FEEDER IS THE NUMBER OF THE FEEDER ENERGIZING THE TRANSFERED LOAD

C       SEE ALSO:

C       FILE          ROUTINE       COMMENT
C       CONSTANTS                   ALL CONSTANTS DECLARED
C       COMMON                      ALL COMMON VARIABLES DECLARED
C       COMP          FORBAC        CHANGES THE NETWORK TOPOLOGY

C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        INCLUDE 'CONSTANTS.FOR'
        INCLUDE 'COMMON.FOR'

        INTEGER TRAFO,FEEDER,BRA
        INTEGER POI(2),OLD(2)
        INTEGER I,J

        IF (POI(1).EQ.SWITYP) THEN
          I=SWITCH(POI(2),3)
          IF (CABTOP(I,1).EQ.OLD(1) .AND. CABTOP(I,2).EQ.OLD(2)) THEN
            SWITCH(POI(2),2)=3
          ELSEIF (CABTOP(I,3).EQ.OLD(1) .AND. CABTOP(I,4).EQ.OLD(2)) THEN
            SWITCH(POI(2),2)=3
          ELSE
            SWITCH(POI(2),2)=4
          ENDIF
        ELSEIF (POI(1).EQ.NODTYP) THEN
          I=NODETO(POI(2),2)
          IF (CABTOP(I,1).EQ.OLD(1) .AND. CABTOP(I,2).EQ.OLD(2)) THEN
            NODETO(POI(2),1)=2
          ELSEIF (CABTOP(I,3).EQ.OLD(1) .AND. CABTOP(I,4).EQ.OLD(2)) THEN
            NODETO(POI(2),1)=2
          ELSE
            NODETO(POI(2),1)=3
          ENDIF
          I=NODETO(POI(2),1)
          I=NODETO(POI(2),1)
          NODETO(POI(2),4)=FEEDER
        ELSEIF (POI(1).EQ.BRATYP) THEN
          BRA=BRAPOI(POI(2),2)
          DO 10 J=3,(2+BRAPOI(POI(2),1))
            I=BRAPOI(POI(2),J)
            IF (CABTOP(I,1).EQ.OLD(1) .AND. CABTOP(I,2).EQ.OLD(2)) THEN
              BRAPOI(POI(2),2)=J
            ELSEIF (CABTOP(I,3).EQ.OLD(1) .AND. CABTOP(I,4).EQ.OLD(2)) THEN
              BRAPOI(POI(2),2)=J
            ENDIF
 10       CONTINUE
        ELSEIF (POI(1).EQ.FEETYP) THEN
          I=MSFEED(POI(2),2)
          MSFEED(POI(2),2)=MSFEED(POI(2),3)
          MSFEED(POI(2),3)=I
          MSFEED(POI(2),1)=TRAFO
```

```
      ELSE
         WRITE(*,*)'       WRONG TYPE IN CHGDIR',POI(1),POI(2)
         STOP
      ENDIF
      OLD(1)=POI(1)
      OLD(2)=POI(2)
      RETURN
      END


      SUBROUTINE COMPIM(FEEDER)

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     AUTHOUR MAGNUS BERGSTRAND

C     VERSION 1

C     DATE 1989-09-30

C     PURPOSE:
C     COMPUTES THE IMPEDANCE FOR FEEDER 'FEEDER'

C     METHOD:
C     COMPUTATIONS IS BASED ON THE FACT THAT THE DISTRIBUTION NETWORK
C     HAS A RADIAL STRUCTURE. THEREFORE PARALLELL IMPEDANCES WITH CABLE
C     IMPEDANCES WILL BE COMPUTED. SEE ALSO 'SERVICE RESTORATION IN
C     ELECTRIC DISTRIBUTION NETWORKS' BY MAGNUS BERGSTRAND (SECTION
C     COMPUTATIONS)

C     DESCRIPTION:
C     TRAVERSES THE TREE WITH 'FEEDER' AS ROOT.
C     STORES PARALLEL IMPEDANCES IN EVERY BRANCH POINT
C     COMPUTATIONS IS ONLY MADE WHEN MOVING TOWARDS THE TOP
C     WHEN THE ROOT IS REACHED THE SUBROUTINE CONTINUES COMPUTING
C     IMPEDANCES UNTIL THE TOP IS REACHED

C     VARIABLES:
C     FEEDER IS THE NUMBER OF THE FEEDER FOR WICH THE IMPEDANCE IS COMPUTED
C     IMPEDANCES IN OHM

C     SEE ALSO:

C     FILE        ROUTINE     COMMENT
C     CONSTANTS               ALL CONSTANTS DECLARED
C     COMMON                  ALL COMMON VARIABLES DECLARED
C     MOVING      MOVE        TAKES ONE STEP UP OR DOWN
C     MOVING      DIREC       COMPUTES NEW DIRECTION
C                             THE ABOVE ARE USED TO TRAVERSE THE TREE
C     COMP        PARALL      COMPUTES THE EQUIVALENT TO TWO PARALLELL
C                             IMPEDANCES
C     COMP        BRAPAR      COMPUTES THE EQUIVALENT IMPEDANCE FOR A
C                             BRANCH POINT

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      INCLUDE 'CONSTANTS.FOR'
      INCLUDE 'COMMON.FOR'

      INTEGER POI(2),OLD(2)
      LOGICAL READY,FIRST
      INTEGER A,DIR,BRA,I,J
      REAL SUMR,SUMX,SUM,RES,REA,F(2),G(2)

      POI(1)=FEETYP
      POI(2)=FEEDER
```

```
      READY=.FALSE.
      OLD(1)=POI(1)
      OLD(2)=POI(2)
      DIR=DOWN
      BRA=3
 10   IF (.NOT. READY) THEN
        CALL MOVE(POI,OLD,DIR,BRA,.FALSE.)
        CALL DIREC(POI,OLD,DIR,BRA)
        IF (DIR.EQ.UP) THEN
          IF (POI(1).EQ.NODTYP) THEN
            IF (FIRST) THEN

C      A LEAF IS FOUND. COMPUTATIONS START

              RES=NODEPH(POI(2),4)
              REA=NODEPH(POI(2),5)
              FIRST=.FALSE.
            ELSE

C      THE NODE IS NOT A LEAF. PARALLELL IMPEDANCE IS COMPUTED INCLUDING
C      CABLE IMPEDANCE

              G(1)=RES
              G(2)=REA
              F(1)=NODEPH(POI(2),4)
              F(2)=NODEPH(POI(2),5)
              CALL PARALL(F,G,RES,REA)
              I=NODETO(POI(2),1)
              I=NODETO(POI(2),I)
              RES=CABPHY(I,1)+RES
              REA=CABPHY(I,2)+REA
            ENDIF
          ELSEIF (POI(1).EQ.SWITYP) THEN
            IF (.NOT. FIRST) THEN

C     ADDS UP CABLE IMPEDANCE

              I=SWITCH(POI(2),2)
              I=SWITCH(POI(2),I)
              RES=RES+CABPHY(I,1)
              REA=REA+CABPHY(I,2)
            ELSE
              RES=1.0E+30
              REA=1.0E+30
            ENDIF

          ELSEIF (POI(1).EQ.TRATYP) THEN

C     STORE IMPEDANCE FOR LATER USE

            TRAFOS(POI(2),6)=RES
            TRAFOS(POI(2),7)=REA

          ELSEIF (POI(1).EQ.BRATYP) THEN

C     COMPUTES THE PARALLEL IMPEDANCE AND SAVES IT IN BRAPH FOR LATER USE

            DO 20 I=3,(2+BRAPOI(POI(2),1))
              J=BRAPOI(POI(2),I)
              IF (CABTOP(J,1).EQ.OLD(1) .AND. CABTOP(J,2).EQ.OLD(2)) THEN
                BRAPH(POI(2),I,1)=RES
                BRAPH(POI(2),I,2)=REA
              ELSEIF (CABTOP(J,3).EQ.OLD(1) .AND. CABTOP(J,4).EQ.OLD(2)) THEN
                BRAPH(POI(2),I,1)=RES
                BRAPH(POI(2),I,2)=REA
              ENDIF
```

```
 20         CONTINUE
            CALL BRAPAR(POI(2),RES,REA)
            I=BRAPOI(POI(2),2)
            I=BRAPOI(POI(2),I)
            RES=RES+CABPHY(I,1)
            REA=REA+CABPHY(I,2)

          ELSEIF (POI(1).EQ.FEETYP) THEN
            I=MSFEED(POI(2),2)
            RES=RES+CABPHY(I,1)
            REA=REA+CABPHY(I,2)
          ELSE
            WRITE(*,*)'       WRONG TYPE IN COMPIM',POI(1),POI(2)
            STOP
          ENDIF
        ELSEIF (DIR.EQ.DOWN) THEN

C       IF A BRANCH POINT IS REACHED AND A NEW DIRECTION DOWN HAS BEEN
C       COMPUTED RES AND REA CONTAINING THE IMPEDANCE OF THE BRANCH THAT
C       HAS BEEN VISITED MUST BE SAVED FOR LATER USE (Z=(RES,j*REA))

          IF (POI(1).EQ.BRATYP) THEN
            DO 30 I=3,(2+BRAPOI(POI(2),1))
              A=BRAPOI(POI(2),I)
              IF (CABTOP(A,1).EQ.OLD(1) .AND. CABTOP(A,2).EQ.OLD(2)) THEN
                J=I
              ELSEIF (CABTOP(A,3).EQ.OLD(1) .AND. CABTOP(A,4).EQ.OLD(2)) THEN
                J=I
              ENDIF
 30         CONTINUE
            IF (J.NE.BRAPOI(POI(2),2)) THEN
              BRAPH(POI(2),J,1)=RES
              BRAPH(POI(2),J,2)=REA
            ENDIF
          ENDIF
          FIRST=.TRUE.
        ENDIF
        READY=POI(1).EQ.FEETYP
        GOTO 10
      ENDIF
      READY=.FALSE.
 35   IF (.NOT. READY) THEN

C     HERE THE SAME COMPUTATIONS AS THE ABOVE IS PERFORMED UNTIL
C     THE TRANSFORMER IS REACHED

        OLD(1)=POI(1)
        OLD(2)=POI(2)
        CALL STEPUP(POI,.FALSE.)
        IF (POI(1).EQ.SWITYP) THEN
          IF (.NOT. FIRST) THEN
            I=SWITCH(POI(2),2)
            I=SWITCH(POI(2),I)
            RES=RES+CABPHY(I,1)
            REA=REA+CABPHY(I,2)
          ENDIF
        ELSEIF (POI(1).EQ.TRATYP) THEN
          TRAFOS(POI(2),6)=RES
          TRAFOS(POI(2),7)=REA
        ELSEIF (POI(1).EQ.BRATYP) THEN
          DO 40 I=3,(2+BRAPOI(POI(2),1))
            J=BRAPOI(POI(2),I)
            IF (CABTOP(J,1).EQ.OLD(1) .AND. CABTOP(J,2).EQ.OLD(2)) THEN
              BRAPH(POI(2),I,1)=RES
              BRAPH(POI(2),I,2)=REA
            ELSEIF (CABTOP(J,3).EQ.OLD(1) .AND. CABTOP(J,4).EQ.OLD(2)) THEN
```

```
            BRAPH(POI(2),I,1)=RES
            BRAPH(POI(2),I,2)=REA
          ENDIF
 40     CONTINUE
        CALL BRAPAR(POI(2),RES,REA)
        I=BRAPOI(POI(2),2)
        I=BRAPOI(POI(2),I)
        RES=RES+CABPHY(I,1)
        REA=REA+CABPHY(I,2)
      ENDIF
      READY=POI(1).EQ.TRATYP
      GOTO 35
    ENDIF
    RETURN
    END




    SUBROUTINE FILPOW(TRAFO)

C   CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C   AUTHOUR MAGNUS BERGSTRAND

C   VERSION 1

C   DATE 1989-09-30

C   PURPOSE:
C   COMPUTES AND STORES THE POWER OF THE TRANSFORMER 'TRAFO'

C   DESCRIPTION:
C   FIRST THE IMPEDANCE FOR THE TRANSFORMER MUST BE COMPUTED
C   USING COMPIM
C   THEN THE POWER IS COMPUTED USING I=I/Z,P=Z*I**2

C   VARIABLES:
C   TRAFO IS THE NUMBER OF THE TRANSFORMER
C   POWER IN VA
C   VOLTAGE IN VOLT
C   IMPEDANCE IN AMPERE

C   SEE ALSO:

C   FILE        ROUTINE     COMMENT
C   CONSTANTS               ALL CONSTANTS DECLARED
C   COMMON                  ALL COMMON VARIABLES DECLARED
C   COMP        COMPIM      COMPUTES THE IMPEDANCE FOR ONE FEEDER

C   CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC


    INCLUDE 'CONSTANTS.FOR'
    INCLUDE 'COMMON.FOR'

    INTEGER TRAFO
    REAL CUR

    CUR=TRAMAX/SQRT(TRAFOS(TRAFO,6)**2+TRAFOS(TRAFO,7)**2)
    TRAFOS(TRAFO,3)=TRAFOS(TRAFO,6)*CUR**2
    TRAFOS(TRAFO,4)=TRAFOS(TRAFO,7)*CUR**2
    RETURN
    END
```

```
      SUBROUTINE COMPVO(R,X,CUR,CURFI,VOLT,FI)

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     AUTHOUR MAGNUS BERGSTRAND

C     VERSION 1

C     DATE 1989-09-30

C     PURPOSE:
C     COMPUTES NODE VOLTAGES

C     DESCRIPTION:
C     USES THE FORMULA V2=V1-Z*I

C     VARIABLES:
C     THE IMPEDANCE Z IS REPRESENTED BY (R,j*X) BOTH IN OHMS
C     THE CURRENT I IS EXPONENTIAL FORM I=CUR*EXP(CURFI)
C     CUR IN AMPERE,CURFI IN RADIANS
C     THE VOLTAGE ALSO IN EXPONENTIAL FORM (BOTH IN AND OUT)
C     VOLT IN VOLTS AND FI IN RADIANS

C     SEE ALSO:

C     FILE          ROUTINE      COMMENT
C     CONSTANTS                  ALL CONSTANTS DECLARED
C     COMMON                     ALL COMMON VARIABLES DECLARED
C     COMP          COMPCU       COMPUTES THE CABLE CURRENT

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      REAL R,X,CUR,CURFI,VOLT,FI
      REAL ZABS,ZFI
      REAL REA,IM

      ZABS=SQRT(R**2+X**2)
      ZFI=ATAN(X/R)
      REA=VOLT*COS(FI)-CUR*ZABS*COS(CURFI+ZFI)
      IM=VOLT*SIN(FI)-CUR*ZABS*SIN(CURFI+ZFI)
      VOLT=SQRT(REA**2+IM**2)
      FI=ATAN(IM/REA)
      RETURN
      END



      SUBROUTINE COMPCU(BRAFI,BRACUR,CURREN,FI,POI,BRA)

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     AUTHOUR MAGNUS BERGSTRAND

C     VERSION 1

C     DATE 1989-09-30

C     PURPOSE:
C     COMPUTES THE CABLE CURRENT IN TWO CASES:
C     1. AFTER A NODE
C     2. THE CURRENT DOWN IN THE BRANCH 'BRA' IF 'POI' IS A BRANCH POINT

C     DESCRIPTION:
C     1. SINCE THE VOLTAGE IS KNOWN THE CURRENT IS DECREMENTED BY U/Z
```

```
C       2. ALL IMPEDANCES FOR THE DIFFERENT BRANCHES ARE KNOWN AS WELL
C          AS THE VOLTAGE THE CURRENT IS EASILY COMPUTED USING THE LAWS
           OF KIRCHHOFF

C       VARIABLES:
C       BRACUR IN AMPERE (CURRENT FROM FEEDER TO BRANCHPOINT)
C       BRAFI IN RADIAN (PHASE)
C       CURREN IN AMPERE (1. IN AND OUT      2. ONLY OUT)
C       FI IN RADIAN (1. IN AND OUT       2. ONLY OUT)

C       SEE ALSO:
C       FILE           ROUTINE      COMMENT

C       CONSTANTS                   ALL CONSTANTS DECLARED
C       COMMON                      ALL COMMON VARIABLES DECLARED
C       COMP           COMPIM       COMPUTES IMPEDANCE OF A FEEDER
C       COMP           COMPVO       COMPUTES VOLTAGE
C       COMP           FILVOL       COMPUTES AND FILLS IN NODE VOLTAGES
C                                   AND CABLE CURRENTS FOR ONE FEEDER

C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        INCLUDE 'CONSTANTS.FOR'
        INCLUDE 'COMMON.FOR'

        REAL BRAFI,BRACUR
        REAL CURREN,FI
        INTEGER POI(2),BRA
        COMPLEX ZP,Z,CUR
        REAL RES,REA,VO
        REAL ALFA1,ZABS

        IF (POI(1).EQ.BRATYP) THEN
          CALL BRAPAR(POI(2),RES,REA)
          ZP=CMPLX(RES,REA)
          Z=CMPLX(BRAPH(POI(2),BRA,1),BRAPH(POI(2),BRA,2))
          CUR=CMPLX(BRACUR*COS(BRAFI),BRACUR*SIN(BRAFI))
          CUR=CUR*ZP/Z
          CURREN=ABS(CUR)
          FI=ATAN((AIMAG(CUR))/(REAL(CUR)))
        ELSEIF (POI(1).EQ.NODTYP) THEN
          VO=NODEPH(POI(2),3)
          RES=NODEPH(POI(2),4)
          REA=NODEPH(POI(2),5)
          ALFA1=ATAN(REA/RES)-FI
          ZABS=SQRT(RES**2+REA**2)
          CURREN=CURREN-COS(ALFA1)*VO/ZABS
          FI=FI+ATAN(VO*SIN(ALFA1)/CURREN/ZABS)
        ELSE
          WRITE(*,*)'WRONG TYPE IN COMPCU'
          STOP
        ENDIF
        RETURN
        END


        SUBROUTINE FILVOL(FEEDER,VIO)

C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C       AUTHOUR MAGNUS BERGSTRAND

C       VERSION 1

C       DATE 1989-09-30
```

```
C       PURPOSE:
C       COMPUTES AND FILLS IN NODE VOLTAGES AND CABLE CURRENTS FOR ONE FEEDER

C       DESCRIPTION:
C       TRAVERSES THE TREE WITH 'FEEDER' AS ROOT.
C       WHEN MOVING DOWNWARDS COMPUTING NEW CABLE CURRENTS AND NODE VOLTAGES
C       IF ANY VIOLATIONS OCCUR 'VIO' BECOMES TRUE AND THE COMMON DECLARED
C       VIOVEC (VIOLATION VECTOR) DIFFERS FROM ZERO
C       ONLY THE BIGGEST LINE CAPACITY VIOLATION IS STORED IN THE FIRST ELEMENT
C       THE SUM OF VOLTAGE DROP CONSTRAINTS ARE STORED IN SECOND ELEMENT

C       VARIABLES:
C       FEEDER IS THE NUMBER OF THE FEEDER
C       VIO INDICATES IF A VIOLATION OCCURED

C       SEE ALSO:

C       FILE           ROUTINE       COMMENT
C       CONSTANTS                    ALL CONSTANTS DECLARED
C       COMMON                       ALL COMMON VARIABLES DECLARED
C       MOVING         MOVE          TAKES ONE STEP UP OR DOWN
C       MOVING         DIREC         COMPUTES NEW DIRECTION
C       MOVING         STEPUP        TAKES ONE STEP UP
C       MOVING         STEPDO        TAKES ONE STEP DOWN
C       COMP           COMPVO        COMPUTES NODE VOLTAGE
C       COMP           COMPCU        COMPUTES CURRENT

C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        INCLUDE 'CONSTANTS.FOR'
        INCLUDE 'COMMON.FOR'

        INTEGER FEEDER
        REAL R,X,VOL,BRAVOL(MAXNRB),RSUM,XSUM,SUM,CURREN
        INTEGER I,J,K,A,B
        INTEGER WAY(MAXNRB)
        INTEGER BRA,DIR,POI(2),OLD(2)
        LOGICAL READY,VIO,ENDNOD
        REAL CUR,BRACUR(MAXNRB)
        REAL SQRT
        REAL NODFI(MAXNRN),BRAFI(MAXNRB)
        REAL CURFI,VOLFI,BCURFI(MAXNRB)

        INTRINSIC SQRT

        VIO=.FALSE.
        VIOVEC(FEEDER,1)=0.0
        VIOVEC(FEEDER,2)=0.0
        POI(1)=FEETYP
        POI(2)=FEEDER
        READY=.FALSE.
        I=1
 10     IF (.NOT. READY) THEN

C       MOVES TO TRANSFORMER MEMORIZING PATH

          OLD(1)=POI(1)
          OLD(2)=POI(2)
          CALL STEPUP(POI,.FALSE.)
          IF (POI(1).EQ.BRATYP) THEN
            DO 20 J=3,(2+BRAPOI(POI(2),1))
              K=BRAPOI(POI(2),J)
              IF (CABTOP(K,1).EQ.OLD(1) .AND. CABTOP(K,2).EQ.OLD(2)) THEN
                WAY(I)=J
                I=I+1
```

```
          ELSEIF (CABTOP(K,3).EQ.OLD(1) .AND. CABTOP(K,4).EQ.OLD(2)) THEN
            WAY(I)=J
            I=I+1
          ENDIF
 20     CONTINUE
        ENDIF
        READY=(POI(1).EQ.TRATYP)
        GOTO 10
      ENDIF
      READY=.FALSE.

C     INITIATE

      R=0.0
      X=0.0
      VOL=TRAMAX
      VOLFI=0.0
      DO 25 J=1,MAXNRB
        BRAVOL(J)=0.0
        BRACUR(J)=0.0
        BRAFI(J)=0.0
        BCURFI(J)=0.0
 25   CONTINUE

C     STARTS COMPUTING ON THE WAY BACK TO FEEDER

      CUR=TRAMAX/SQRT(TRAFOS(POI(2),6)**2+TRAFOS(POI(2),7)**2)
      CURFI=ATAN(TRAFOS(POI(2),7)/TRAFOS(POI(2),6))
      BRA=4
 30   IF (.NOT. READY) THEN
        CALL STEPDO(POI,BRA,.FALSE.)
        IF (POI(1).EQ.BRATYP) THEN
          I=I-1
          BRA=WAY(I)
          J=BRAPOI(POI(2),2)
          J=BRAPOI(POI(2),J)
          R=R+CABPHY(J,1)
          X=X+CABPHY(J,2)
          IF (CUR.GT.CABPHY(J,3)) THEN
            VIO=.TRUE.
            IF ((CUR-CABPHY(J,3)).GT.VIOVEC(FEEDER,1)) THEN
              VIOVEC(FEEDER,1)=VIOVEC(FEEDER,1)+CUR-CABPHY(J,3)
            ENDIF
          ENDIF
          CABPHY(J,4)=CUR
          BRACUR(POI(2))=CUR
          BCURFI(POI(2))=CURFI
          CALL COMPVO(R,X,CUR,CURFI,VOL,VOLFI)
          BRAVOL(POI(2))=VOL
          BRAFI(POI(2))=VOLFI
          IF (VOL.LT.VOLLIM) THEN
            VIO=.TRUE.
          ENDIF
          R=0.0
          X=0.0
          CALL COMPCU(BCURFI(POI(2)),BRACUR(POI(2)),CUR,FI,POI,BRA)
        ELSEIF (POI(1).EQ.SWITYP) THEN
          J=SWITCH(POI(2),2)
          J=SWITCH(POI(2),J)
          R=R+CABPHY(J,1)
          X=X+CABPHY(J,2)
          CABPHY(J,4)=CUR
          IF (CUR.GT.CABPHY(J,3)) THEN
            VIO=.TRUE.
            IF ((CUR-CABPHY(J,3)).GT.VIOVEC(FEEDER,1)) THEN
              VIOVEC(FEEDER,1)=VIOVEC(FEEDER,1)+CUR-CABPHY(J,3)
```

```
          ENDIF
        ENDIF
      ELSEIF (POI(1).EQ.FEETYP) THEN
        J=MSFEED(POI(2),2)
        R=R+CABPHY(POI(2),1)
        X=X+CABPHY(POI(2),2)
      ELSE
        WRITE(*,*)'       DEGENERATED NETWORK.FILVOL'
        STOP
      ENDIF
      READY=(POI(1).EQ.FEETYP .AND. POI(2).EQ.FEEDER)
      GOTO 30
    ENDIF

C     NOW THE FEEDER IS REACHED
C     FROM HERE ON THE TREE IS TRAVERSED COMPUTING CURRENTS AND VOLTAGES

      READY=.FALSE.
      DIR=DOWN
 40   IF (.NOT. READY) THEN
        CALL MOVE(POI,OLD,DIR,BRA,.FALSE.)
        IF (DIR.EQ.DOWN) THEN
          ENDNOD=.TRUE.
          IF (POI(1).EQ.SWITYP) THEN
            IF (SWITCH(POI(2),1).EQ.CLOSED) THEN
              J=SWITCH(POI(2),2)
              J=SWITCH(POI(2),J)
              R=R+CABPHY(J,1)
              X=X+CABPHY(J,2)
              IF (CUR.GT.CABPHY(J,3)) THEN
                VIO=.TRUE.
                IF ((CUR-CABPHY(J,3)).GT.VIOVEC(FEEDER,1)) THEN
                  VIOVEC(FEEDER,1)=VIOVEC(FEEDER,1)+CUR-CABPHY(J,3)
                ENDIF
              ENDIF
              CABPHY(J,4)=CUR
            ENDIF
          ELSEIF (POI(1).EQ.NODTYP) THEN
            J=NODETO(POI(2),1)
            J=NODETO(POI(2),J)
            R=R+CABPHY(J,1)
            X=X+CABPHY(J,2)
            CABPHY(J,4)=CUR
            CALL COMPVO(R,X,CUR,CURFI,VOL,VOLFI)
            NODEPH(POI(2),3)=VOL
            IF (VOL.LT.VOLLIM) THEN
              VIO=.TRUE.
            ENDIF
            IF (CUR.GT.CABPHY(J,3)) THEN
              VIO=.TRUE.
              IF ((CUR-CABPHY(J,3)).GT.VIOVEC(FEEDER,1)) THEN
                VIOVEC(FEEDER,1)=VIOVEC(FEEDER,1)+CUR-CABPHY(J,3)
              ENDIF
            ENDIF
            CURREN=VOL/SQRT(NODEPH(POI(2),4)**2+NODEPH(POI(2),5)**2)
            NODEPH(POI(2),1)=NODEPH(POI(2),4)*CURREN**2
            NODEPH(POI(2),2)=NODEPH(POI(2),5)*CURREN**2
            CALL COMPCU(1.0,1.0,CUR,CURFI,POI,BRA)
            R=0.0
            X=0.0
          ELSEIF (POI(1).EQ.BRATYP) THEN
            J=BRAPOI(POI(2),2)
            J=BRAPOI(POI(2),J)
            R=R+CABPHY(J,1)
            X=X+CABPHY(J,2)
            CALL COMPVO(R,X,CUR,CURFI,VOL,VOLFI)
```

```
            BRAVOL(POI(2))=VOL
            BRAFI(POI(2))=VOLFI
            IF (CUR.GT.CABPHY(J,3)) THEN
              VIO=.TRUE.
              IF ((CUR-CABPHY(J,3)).GT.VIOVEC(FEEDER,1)) THEN
                VIOVEC(FEEDER,1)=VIOVEC(FEEDER,1)+CUR-CABPHY(J,3)
              ENDIF
            ENDIF
            CABPHY(J,4)=CUR
            BCURFI(POI(2))=CURFI
            BRACUR(POI(2))=CUR
          ELSE
            WRITE(*,*)'       WRONG IN NETWORK.FILVOL'
            WRITE(*,*)'       POINT IS',POI(1),POI(1)
            STOP
          ENDIF
        ENDIF
        CALL DIREC(POI,OLD,DIR,BRA)
        IF (DIR.EQ.DOWN .AND. POI(1).EQ.BRATYP) THEN
          CALL COMPCU(BCURFI(POI(2)),BRACUR(POI(2)),CUR,CURFI,POI,BRA)
          VOL=BRAVOL(POI(2))
          VOLFI=BRAFI(POI(2))
          R=0.0
          X=0.0
        ENDIF
        IF (DIR.EQ.UP .AND. ENDNOD) THEN
          IF (POI(1).EQ.NODTYP) THEN

C      THIS IS THE ONLY PLACE NECESSARY TO CHECK THE VOLTAGE DROP CONSTRAINT
C      ANY VIOLATION WILL ALWAYS OCCUR IN A LEAF NODE

            IF (NODEPH(POI(2),3).LT.VOLLIM) THEN
              VIOVEC(FEEDER,2)=VIOVEC(FEEDER,2)+VOLLIM-NODEPH(POI(2),3)
            ENDIF
            ENDNOD=.FALSE.
          ENDIF
        ENDIF
        READY=(POI(1).EQ.FEETYP)
        GOTO 40
      ENDIF
      RETURN
      END


      SUBROUTINE PARALL(A,B,RES,REA)

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     AUTHOUR MAGNUS BERGSTRAND

C     VERSION 1

C     DATE 1989-09-30

C     PURPOSE:
C     COMPUTES THE IMPEDANCE (RES,j*REA)=A//B

C     VARIABLES:
C     A AND B ARE VECTORS (OHM,j*OHM)
C     RES IS RESISTANCE (OUT) IN OHM
C     REA IS REACTANCE (OUT) IN OHM

C     SEE ALSO:

C     FILE          ROUTINE      COMMENT
C     CONSTANTS                  ALL CONSTANTS DECLARED
```

```
C       COMMON                      ALL COMMON VARIABLES DECLARED
C       COMP        BRAPAR          COMPUTES PARALLELL IMPEDANCE WHEN POINT
C                                   IS A BRANCH POINT

C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        REAL A(2),B(2),RES,REA
        COMPLEX A1,B1,C1,C2

        A1=CMPLX(A(1),A(2))
        B1=CMPLX(B(1),B(2))
        C1=CMPLX(1,0)
        C2=C1/A1+C1/B1
        C2=C1/C2
        RES=REAL(C2)
        REA=AIMAG(C2)
        RETURN
        END



        SUBROUTINE BRAPAR(NR,RES,REA)

C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C       AUTHOUR MAGNUS BERGSTRAND

C       VERSION 1

C       DATE 1989-09-30

C       PURPOSE:
C       COMPUTES THE PARALLELL IMPEDANCE FOR BRANCH POINT 'NR'

C       VARIABLES:
C       THE IMPEDANCE (RES,j*REA) IN OHMS (OUT)

C       SEE ALSO:

C       FILE        ROUTINE         COMMENT
C       CONSTANTS                   ALL CONSTANTS DECLARED
C       COMMON                      ALL COMMON VARIABLES DECLARED
C       COMP        PARALL          COMPUTES THE PARALLELL IMPEDANCE BETWEEN
C                                   TWO VECTORS

C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        INCLUDE 'CONSTANTS.FOR'
        INCLUDE 'COMMON.FOR'

        INTEGER NR
        REAL RES,REA
        INTEGER I,J
        COMPLEX A,B,C
        LOGICAL READY

        IF (BRAPOI(NR,2).EQ.3) THEN
          I=4
        ELSE
          I=3
        ENDIF
        A=CMPLX(BRAPH(NR,I,1),BRAPH(NR,I,2))
        B=CMPLX(1,0)
        A=B/A
 10     IF (I.LT.(BRAPOI(NR,1)+2)) THEN
```

```
   I=I+1
   IF (I.NE.BRAPOI(NR,2)) THEN
      C=CMPLX(BRAPH(NR,I,1),BRAPH(NR,I,2))
      A=A+B/C
   ENDIF
   GOTO 10
ENDIF
A=B/A
RES=REAL(A)
REA=AIMAG(A)
RETURN
END
```

```
C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                      STEP2.FOR
C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC



        SUBROUTINE STAGE(FEEDER,OK)
C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C       AUTHOUR MAGNUS BERGSTRAND
C       VERSION 1
C       DATE 1989-10-05
C       PURPOSE
C       WHEN A LOAD TRANSFER IS TO BE PERFORMED THIS SUBROUTINE
C       EXAMINE DIFFERENT POSSIBILITYS AND MAKES A FILE 'STAGE'
C       CONTAINING DATA ABOUT POSSIBLE SWITCHES
C       DESCRIPTION
C       TRAVERSES THE TREE WITH THE FEEDER 'FEEDER' AS ROOT.
C       WHEN A CLOSED SWITCH IS FOUND THE ESTIMATED REDUCTION
C       OF THE VIOLATION VECTOR IS COMPUTED AND STORED IN THE
C       FILE 'STAGE'. ALSO THE NUMBERS OF THE SWITCHES INVOLVED
C       , THE NODES PRIORITY AS WELL AS THE NUMBER OF THE FEEDER
C       THAT WILL TAKE UP ANY LOAD WILL BE STORED.
C       THIS SUBROUTINE IS ONLY TO BE CALLED FROM 'STAGE1'
C       SEE ALSO
C       FILES        ROUTINES     COMMENTS
C       CONSTANTS                 ALL CONSTANTS DECLARED
C       COMMON                    ALL COMMON VARIABLES DECLARED
C       STEP2        STAGE1       FIRST STAGE SUPPORT. TRANSFERS LOAD TO
C                                 AN ADJACENT FEEDER.
C       STEP2        STAGE2       SECOND STAGE SUPPORT. TRANSFERS LOAD TO
C                                 AN ADJACENT FEEDER WICH IN ITS TURN
C                                 TRANSFERS LOAD TO A SECONDARY SUPPORT
C                                 FEEDER.
C       STEP2        IMP          COMPUTES THE ESTIMATED REDUCTION OF THE
C                                 VIOLATION VECTOR
C       MOVING       MOVE         TAKES ONE STEP UP OR DOWN
C       MOVING       DIREC        COMPUTES NEW DIRECTION
C       FETCH        GETFEE       GETS THE NUMBER OF THE FEEDER ON THE OTHER
C                                 SIDE OF AN OPEN SWITCH
C       VARIABLES
C       FEEDER IS THE NUMBER OF THE FEEDER THAT WILL TRANSFER LOAD
C       OK INDICATES IF ANY POSSIBLE LOAD TRANSFERS WERE FOUND
C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        INCLUDE 'CONSTANTS.FOR'
        INCLUDE 'COMMON.FOR'

        INTEGER FEEDER
        LOGICAL OK
        INTEGER I
        INTEGER POI(2),OLD(2)
        INTEGER DIR,BRA,SWI2,FEE
```

```
      REAL SUM,LOAD,ALPHA
      INTEGER GETFEE
      LOGICAL VIOLTE

      EXTERNAL GETFEE
      EXTERNAL VIOLTE

      WRITE(*,*)'ENTERS STAGE*************************************'
      OPEN(1,FILE='STAGE',STATUS='NEW',ACCESS='SEQUENTIAL')
      POI(1)=FEETYP
      POI(2)=FEEDER
      DIR=DOWN
      BRA=3
      CALL STEPDO(POI,BRA,.FALSE.)
      OK=.FALSE.
 10   IF (POI(1).NE.FEETYP) THEN

C     TRAVERSES THE TREE

        CALL MOVE(POI,OLD,DIR,BRA,.FALSE.)
        CALL DIREC(POI,OLD,DIR,BRA)
        IF (POI(1).EQ.SWITYP) THEN
          IF (SWITCH(POI(2),1).EQ.OPEN .AND. SWITCH(POI(2),5).EQ.1) THEN

C     A POSSIBLE LOAD TRANSFER IS FOUND

            FEE=GETFEE(POI(2))
            IF (.NOT. VIOLTE(FEE)) THEN

C     THE FEEDER THAT WILL TAKE UP LOAD IS NOT VIOLATED
C     COMPUTE THE IMPROVEMENT AND STORE DATA

              OK=.TRUE.
              CALL IMP(POI(2),SWI2,SUM,LOAD,ALPHA)
              WRITE(1,*) POI(2),SWI2,SUM,FEE,LOAD,ALPHA
            ENDIF
          ENDIF
        ENDIF
        GOTO 10
      ENDIF
      CLOSE(1,STATUS='KEEP')
      RETURN
      END



      SUBROUTINE IMP(SWI,SWI2,SUM,LOAD,ALPHA)
C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     AUTHOUR MAGNUS BERGSTRAND

C     VERSION 1

C     DATE 1989-10-05

C     PURPOSE
C     COMPUTES THE ESTIMATED REDUCTION OF THE VIOLATION VECTOR IF
C     THE SWITCH 'SWI' WOULD BE CLOSED AND 'SWI2' OPENED.

C     DESCRIPTION
C     STARTING AT SWITCH 'SWI' THIS SUBROUTINE MOVES TO SWITCH 'SWI2'
C     CHECKING VIOLATIONS ON THE WAY.

C     SEE ALSO

C     FILES         ROUTINES      COMMENTS
```

```
C     CONSTANTS                          ALL CONSTANTS DECLARED
C     COMMON                             ALL COMMON VARIABLES DECLARED
C     COMP            FILVOL             COMPUTES NODE VOLTAGES AND CABLE CURRENTS
C                                        ASSIGNS THE VIOLATION VECTOR
C     STEP2           MKESUM             COMPUTES THE IMPROVEMENT MEASURE SUM
C     STEP2           EVAL               EVALUATES THE MEASSURE IN THE LOAD CURTAILMENT
C     STEP2           IMPROV             COMPUTES THE ESTIMATED REDUCTION IN THE
C                                        LOAD CURTAILMENT
C     MOVING          STEPUP             TAKES ONE STEP UP
C     MOVING          STEPDO             TAKES ONE STEP DOWN
C     MOVING          MOVE               TAKES ONE STEP UP OR DOWN
C     MOVING          DIREC              COMPUTES NEW DIRECTION

C     VARIABLES
C     SWI IS THE OPEN SWITCH THAT WILL BE CLOSED DUE TO A LOAD TRANSFER
C     SWI2 IS THE CLOSED SWITCH THAT WILL BE OPENED DUE TO A LOAD TRANSFER
C     SUM (OUT) IS THE ESTIMATED IMPROVEMENT
C     LOAD (OUT) IS THE POWER CONSUMTION FOR THE THE NODES TRANSFERED VA.
C     ALPHA (OUT) IS THE NODE PRIORITY FOR TRANSFERED NODES.

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      INCLUDE 'CONSTANTS.FOR'
      INCLUDE 'COMMON.FOR'

      INTEGER SW1,SWI2
      REAL SUM,CUR,LOAD,ALPHA
      INTEGER POI(2),OLD(2),P(2)
      INTEGER DIR,BRA
      REAL SQRT
      LOGICAL ENDNOD,FIRST

      INTRINSIC SQRT

      ALPHA=100
      POI(1)=SWITYP
      POI(2)=SWI
      CALL STEPUP(POI,.FALSE.)
 10   IF (POI(1).NE.SWITYP) THEN
        CALL STEPUP(POI,.FALSE.)
        GOTO 10
      ENDIF
      DIR=DOWN
      SWI2=POI(2)

C     SWI2 IS FOUND
C     ONE HAVE TO START FROM THIS SWITCH BECAUSE OF THE REDUCTION
C     DEPENDS ON POSSIBLE BRANCH POINTS. ONE SWITCHING OPERATION
C     CAN TRANSFER LARGE LOADS IN DIFFERENT BRANCHES

      SUM=0.0
      BRA=3
      FIRST=.TRUE.
      P(1)=POI(1)
      P(2)=POI(2)
      OLD(1)=0
      OLD(2)=0
      ENDNOD=.TRUE.
      CALL STEPDO(POI,BRA,.FALSE.)
 20   IF (POI(1).NE.P(1) .OR. POI(2).NE.P(2)) THEN

C     THIS LITTLE TREE WITH SWI2 AS ROOT IS TRAVERSED COMPUTING
C     THE ESTIMATED REDUCTION OF THE VIOLATION VECTOR

        IF (DIR.EQ.UP) THEN
          CALL MKESUM(POI,SUM,LOAD,ENDNOD,FIRST,ALPHA)
```

```
          ELSE
            ENDNOD=.TRUE.
          ENDIF
          CALL MOVE(POI,OLD,DIR,BRA,.FALSE.)
          CALL DIREC(POI,OLD,DIR,BRA)
          GOTO 20
        ENDIF
        ENDNOD=.FALSE.
 30     IF (POI(1).NE.FEETYP) THEN

C       THE BIGGEST LINE CAPACITY VIOLATION CAN BE IN THIS CABLE

          CALL STEPUP(POI,.FALSE.)
          CALL MKESUM(POI,SUM,LOAD,ENDNOD,FIRST,ALPHA)
          GOTO 30
        ENDIF
        RETURN
        END


        SUBROUTINE MKESUM(POI,SUM,LOAD,ENDNOD,FIRST,ALPHA)
C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C       AUTHOUR MAGNUS BERGSTRAND

C       VERSION 1

C       DATE 1989-10-05

C       PURPOSE
C       COMPUTES THE MEASURE SUM

C       DESCRIPTION
C       IF THE CURRENT POINT IS A NODE A CHECK IS DONE DONE TO SEE IF
C       THE NODE VOLTAGE IS BELOW THE MAXIMUM VOLTAGE DROP
C       THEN A CHECK OF THE LINE CAPACITY CONSTRAINTS IS PERFORMED.
C       IF ANY VIOLATIONS IS FOUND SUM WILL BE INCREASED.

C       SEE ALSO

C       FILES          ROUTINES       COMMENTS
C       CONSTANTS                     ALL CONSTANTS DECLARED
C       COMMON                        ALL COMMON VARIABLES DECLARED
C       STEP2          IMP            COMPUTES THE ESTIMATED  REDUCTION OF THE
C                                     VIOLATION IN CASE OF FIRST OR SECOND STAGE
C                                     SUPPORT BY A CALL OF MKESUM
C       STEP2          EVAL           COMPUTES THE ESTIMATED REDUCTION OF THE
C                                     VIOLATION IN CASE OF LOAD CURTAILMENT
C                                     BY A CALL OF IMPROV.
C       STEP2          IMPROV         COMPUTES THE REDUCTION OF THE VIOLATION
C                                     VECTOR IN CASE OF A LOAD CURTAILMENT

C       VARIABLES
C       POI IS THE CURRENT POINT (TYPE,NR)
C       SUM (OUT) IS THE ESTIMATED REDUCTION
C       LOAD IS THE MAGNITUDE OF TRANSFERED LOAD IN VA.
C       ENDNOD INDICATES THAT A NODE IS A LEAF
C       FIRST INDICATES FIRST CALL FOR A FEEDER
C       ALPHA IS THE PRIORITY

C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC


        INCLUDE 'CONSTANTS.FOR'
        INCLUDE 'COMMON.FOR'
```

```
      REAL SUM,CUR,SQRT,MIN,VOLSUM,LOAD,LOAD1,ALPHA
      INTEGER POI(2),I
      LOGICAL ENDNOD,FIRST

      INTRINSIC SQRT,MIN

C     CUR (REDUCTION IN LINE CAPACITY VIOLATIONS),VOLSUM (REDUCTION IN
C     VOLTAGE DROP VIOLATIONS) AND LOAD1 (MAGNITUDE OF TRANSFERED LOAD)
C     ARE SAVED BETWEEN CALLS

      SAVE CUR
      SAVE VOLSUM
      SAVE LOAD1

      IF (FIRST) THEN

C     INITIATE

        CUR=0.0
        VOLSUM=0.0
        FIRST=.FALSE.
        LOAD1=0.0
      ENDIF

C     THE INDEX OF THE CABLE LEADING TOWARDS THE FEEDER IS STORED IN 'I'

      IF (POI(1).EQ.SWITYP) THEN
        I=SWITCH(POI(2),2)
        I=SWITCH(POI(2),I)
      ELSEIF (POI(1).EQ.NODTYP) THEN
        I=NODETO(POI(2),1)
        I=NODETO(POI(2),I)
        IF (ENDNOD) THEN
          LOAD1=LOAD1+SQRT(NODEPH(POI(2),1)**2+NODEPH(POI(2),2)**2)
          IF (VOLLIM.GT.NODEPH(POI(2),3)) THEN

C     VOLTAGE DROP VIOLATION

            VOLSUM=VOLSUM+VOLLIM-NODEPH(POI(2),3)
            ALPHA=MIN(ALFA(POI(2)),ALPHA)
          ENDIF
          ENDNOD=.FALSE.
        ENDIF
      ELSEIF (POI(1).EQ.BRATYP) THEN
        I=BRAPOI(POI(2),2)
        I=BRAPOI(POI(2),I)
      ENDIF
      IF (CABPHY(I,4).GT.CABPHY(I,3)) THEN

C     LINE CAPACITY VIOLATION

        IF ((CABPHY(I,4)-CABPHY(I,3)).GT.CUR) THEN
          CUR=CABPHY(I,4)-CABPHY(I,3)
        ENDIF
      ENDIF
      LOAD=LOAD1
      SUM=SQRT((CUR**2)+(VOLSUM**2))
      RETURN
      END


      LOGICAL FUNCTION VIOLTE(FEE)
C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     AUTHOUR MAGNUS BERGSTRAND
```

```
C      VERSION 1

C      DATE 1989-10-05

C      PURPOSE
C      CHECKS IF THE FEEDER 'FEE' IS VIOLATED

C      DESCRIPTION
C      IF THE FEEDER 'FEE' IS VIOLATED THEN
C      THIS LOGICAL FUNCTION EQUALS TRUE

C      VARIABLES
C      FEE IS THE NUMBER OF THE FEEDER

C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC



       INCLUDE 'CONSTANTS.FOR'
       INCLUDE 'COMMON.FOR'

       VIOLTE=(VIOVEC(FEE,1).GT.0.0 .OR. VIOVEC(FEE,2).GT.0.0)
       RETURN
       END


       SUBROUTINE STAGE1(FEE,OK,FIRST,OUTFEE)
C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C      AUTHOUR MAGNUS BERGSTRAND

C      VERSION 1

C      DATE 1989-10-05

C      PURPOSE
C      TRANSFERS LOAD TO AN ADJACENT NON VIOLATED FEEDER.

C      METHODE
C      THE METHODE IS DESCRIBED IN 'SERVICE RESTORATION IN ELECTRIC
C      DISTRIBUTIONAL NETWORKS' BY MAGNUS BERGSTRAND
C      THE METHOD WAS FIRST DESCRIBED BY AOKI IN
C      'A NEW ALGORITHM FOR SERVICE RESRORATION IN DISTRIBUTION SYSTEMS'
C      IEEE/PES WINTER MEETING,NEW YORK 1989 PAPER 89 WM 085-2 PWRD

C      DESCRIPTION
C      STARTING AT A VIOLATED FEEDER ALL POSSIBLE LOAD TRANSFERS TO
C      A NON VIOLATED FEEDER IS EXAMINED. THE LOAD TRANSFER THAT WILL
C      REDUCE THE VIOLATION VECTOR THE MOST WILL BE TRIED FIRST. IF THIS
C      LOAD TRANSFER CAUSES A VIOLATION IT IS ABANDONED AND THE NEXT LOAD
C      TRANSFER IS TRIED. IF NO LOAD TRANSFER IS POSSIBLE WITHOUT ANY NEW
C      VIOLATIONS OK EQUALS FALSE.
C      STAGE1 HAS ANOTHER FUNCTION. WHEN A FIRST STAGE SUPPORT HAS FAILED
C      A SECOND STAGE SUPPORT IS TRIED. FIRST IN THIS SECOND STAGE A CALL
C      OF STAGE1 IS MADE WITH OK.EQ..FALSE. . THIS CAUSES THE FIRST STAGE
C      TO ALLOW VIOLATIONS IN THE PRIMARY SUPPORT FEEDER.

C      SEE ALSO

C      FILES          ROUTINES       COMMENTS
C      CONSTANTS                     ALL CONSTANTS DECLARED
C      COMMON                        ALL COMMON VARIABLES DECLARED
C      STEP2          STEP2          IS THE SECOND STEP OF THE ALGORITHM
C                                    HANDLES THE CALLS OF STAGE1 AND STAGE2
C      STEP2          STAGE          MAKES A FILE CONTAINING DATA ABOUT THE
C                                    DIFFERENT LOAD TRANSFERS.
```

```
C     STEP2        STAGE2        THE SECOND STAGE SUPPORT. TRANSFERS FIRST LOAD
C                                TO A PRIMARY SUPPORT FEEDER ALLOWING VIOLATIONS
C                                THESE VIOLATIONS ARE THEN REMOVED BY A LOAD
C                                TRANSFER TO A SECONDARY SUPPORT FEEDER.

C     VARIABLES
C     FEE IS THE NUMBER OF VIOLATED FEEDER (IN)
C     IF OK IS FALSE A VIOLATION IN THE SUPPORT FEEDER IS ALLOWED
C     FIRST INDICATES IF IT IS THE FIRST TIME A LOAD TRANSFER IS TRIED.
C     IF A PREVIOUS LOAD TRANSFER HAS FAILED THE FILE CONTAINING DATA OF
C     POSSIBLE LOAD TRANSFERS IS KEPT.
C     OUTFEE IS AN OUTPARAMETER INDICATING THE NUMBER OF THE SWITCH NOW
C     VIOLATED DUE TO A LOAD TRANSFER. IT IS ONLY VALID WHEN THE CALL WAS
C     MADE WITH OK=FALSE.

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      INCLUDE 'CONSTANTS.FOR'
      INCLUDE 'COMMON.FOR'

      LOGICAL FIRST
      INTEGER FEE,TRAFO,OUTFEE
      LOGICAL VIO,OK
      INTEGER SWI,SWI2,FEEDER
      INTEGER BSWI,BSWI2,BFEE
      REAL SUM,BSUM,SLASK,VIOSUM,LOAD
      REAL MAX,MAX2

      SAVE MAX,MAX2

      EXTERNAL VIOSUM

      WRITE(*,*)'ENTERS STAGE 1',FEE
      IF (OK) THEN
        CALL STAGE(FEE,OK)
        IF (OK) THEN
          WRITE(*,*)'READS IN STAGE1 **************************'
          OPEN(1,FILE='STAGE',STATUS='OLD',ACCESS='SEQUENTIAL')
          BSUM=0.0
          MAX=1.0E+30
          BSWI=MAXNRS+1
 10       READ(1,*,ERR=20) SWI,SWI2,SUM,FEEDER,LOAD,ALPHA

C     READS,COMPUTES AND DECIDES WHICH LOAD TRANSFER IS THE BEST
C     THIS LOAD TRANSFER IS SAVED

            SLASK=0.01+VIOSUM(FEE)-SUM
            SLASK=SUM/(SLASK+BETA)/LOAD/ALPHA
            WRITE(*,*)'IMPROVEMENT,FEEDER',SLASK,FEEDER
            IF (SLASK.GT.BSUM .AND. SLASK.LT.MAX) THEN
              BSUM=SLASK
              BSWI=SWI
              BSWI2=SWI2
              BFEE=FEEDER
            ENDIF
            GOTO 10
 20       IF (BSWI.LT.MAXNRS+1) THEN

C       THEN A POSSIBLE LOAD TRANSFER WAS FOUND
C       THE NETWORKS STATE IS SAVED

            WRITE(*,*)'SAVENE****************************************'
            CALL SAVENE(BFEE)
            WRITE(*,*)FEE,BFEE
            WRITE(*,*)'SAVENE****************************************'
            CALL SAVENE(FEE)
```

```
C        THE LOAD TRANSFER IS PERFORMED

            SWITCH(BSWI2,1)=OPEN
            SWITCH(BSWI,1)=CLOSED
            SWINR(BSWI)=3
            CHGSWI(BSWI2)=.NOT. CHGSWI(BSWI2)
            SWINR(BSWI2)=3
            CHGSWI(BSWI)=.NOT. CHGSWI(BSWI)
            TRA=MSFEED(BFEE,1)
            CALL FORBAC(BSWI,TRA,BFEE)

C        THE NEW STATE IS COMPUTED

            CALL COMPIM(BFEE)
            IF (SWITCH(BSWI2,2).EQ.3) THEN
              SWITCH(BSWI2,2)=4
            ELSE
              SWITCH(BSWI2,2)=3
            ENDIF
            CALL COMPIM(FEE)
            CALL FILPOW(TRA)
            TRA=MSFEED(FEE,1)
            CALL FILPOW(TRA)
            CALL FILVOL(FEE,VIO)
            CALL FILVOL(BFEE,VIO)
            IF (VIO) THEN

C      SINCE A VIOLATION OCCURED THE OLD STATE MUST BE READ BACK

              REWIND 1
              WRITE(*,*)'CALLS RESNET************'
              REWIND 1
              CALL RESNET
              WRITE(*,*)'CALLS RESNET************'
              CALL RESNET

C       TAKE NEXT LOAD TRANSFER AND TRY AGAIN

              MAX=BSUM
              BSUM=0.0
              BSWI=MAXNR5+1
              GOTO 10
            ELSEIF (.NOT. VIO) THEN

C       DELETE FILE AFTER A SUCCESFUL LOAD TRANSFER

              CLOSE(1,STATUS='DELETE')
              WRITE(*,*)'LEAVES STAGE1 AFTER SUCCESFULL SESSION,OK=T'
            ENDIF
          ELSE

C       NO POSSIBLE LOAD TRANSFER WAS FOUND.
C       SAVE FILE FOR LATER USE IN THE SECOND STAGE SUPPORT

            CLOSE(1,STATUS='KEEP')
            WRITE(*,*)'OK=.FALSE.'
            OK=.FALSE.
          ENDIF
        ENDIF
      ELSEIF (.NOT. OK) THEN

C     ALLOWS VIOLATION IN THE SUPPORT FEEDER

        WRITE(*,*)'OK IS FALSE'
        IF (FIRST) THEN
```

```
         BSUM=0.0
         MAX2=1.0E+30
       ELSE

C      A PREVIOUS UNSUCCESFUL SECOND STAGE HAS BEEN MADE.
C      THE OLD STATE MUST BE READ BACK

         CALL RESNET
         CALL RESNET
       ENDIF
       BSWI=MAXNRS+1

C      READ,COMPUTE AND IDENTIFY THE BEST LOAD TRANSFER

       OPEN(1,FILE='STAGE',STATUS='OLD',ACCESS='SEQUENTIAL')
 25    READ(1,*,ERR=30) SWI,SWI2,SUM,FEEDER,LOAD,ALPHA
         SLASK=0.01+VIOSUM(FEE)-SUM
         SLASK=SUM/(SLASK+BETA)/LOAD/ALPHA
         IF (SLASK.GT.BSUM .AND. SLASK.LT.MAX2) THEN
           BSUM=SLASK
           BSWI=SWI
           BSWI2=SWI2
           BFEE=FEEDER
         ENDIF
         GOTO 25
 30    IF (BSWI.LT.MAXNRS+1) THEN

C      A POSSIBLE LOAD TRANSFER WAS FOUND
C      SAVE OLD STATE

         WRITE(*,*)'SAVENE*****************************************'
         CALL SAVENE(BFEE)
         WRITE(*,*)FEE,BFEE
         WRITE(*,*)'SAVENE*****************************************'
         CALL SAVENE(FEE)

C      PERFORM LOAD TRANSFER

         SWITCH(BSWI2,1)=OPEN
         SWITCH(BSWI2,5)=0
         SWITCH(BSWI,1)=CLOSED
         SWINR(BSWI2)=3
         SWINR(BSWI)=3
         CHGSWI(BSWI2)=.NOT. CHGSWI(BSWI2)
         CHGSWI(BSWI)=.NOT. CHGSWI(BSWI)
         TRA=MSFEED(BFEE,1)
         CALL FORBAC(BSWI,TRA,BFEE)

C      COMPUTE NEW STATE

         CALL COMPIM(BFEE)
         CALL COMPIM(FEE)
         CALL FILPOW(TRA)
         TRA=MSFEED(FEE,1)
         CALL FILPOW(TRA)
         CALL FILVOL(FEE,VIO)
         CALL FILVOL(BFEE,VIO)
         OK=.TRUE.
         MAX2=BSUM
         CLOSE(1,STATUS='KEEP')
         OUTFEE=BFEE
       ELSE

C      NO POSSIBLE LOAD TRANSFER WAS FOUND

         WRITE(*,*)'STAGE1 WITH OK=F HAD AN UNSUCCESSFULL COMPLETION'
```

```
            OK=.FALSE.
            CLOSE(1,STATUS='DELETE')
          ENDIF
        ENDIF
        RETURN
        END




        SUBROUTINE SAVENE(FEEDER)
C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C       AUTHOUR MAGNUS BERGSTRAND

C       VERSION 1

C       DATE 1989-10-05

C       PURPOSE
C       SAVES THE FORMER STATE IN THE NETWORK WHEN A LOAD TRANSFER IS
C       TO BE MADE

C       DESCRIPTION
C       TRAVERSES A  TREE WITH THE FEEDER 'FEEDER' AS ROOT SAVING EVERYTHING
C       IN THE NETWORK TOPOLOGY THAT COULD BE CHANGED DUE TO A LOAD TRANSFER
C       IN THE FILE 'HISTORY.DAT'
C       A STACK OF BACKUP FILES IS CREATED

C       SEE ALSO

C       FILES          ROUTINES        COMMENTS
C       CONSTANTS                      ALL CONSTANTS DECLARED
C       COMMON                         ALL COMMON VARIABLES DECLARED
C       MOVING         STEPUP          TAKES ONE STEP UP
C       MOVING         STEPDO          TAKES ONE STEP DOWN
C       MOVING         MOVE            TAKES ONE STEP UP OR DOWN
C       MOVING         DIREC           COMPUTES NEW DIRECTION
C       STEP2          RESNET          READS BACK THE FORMER STATE FROM THE FILE
C                                      'HISTORY.DAT'
C       STEP2          STAGE1          CREATES A STACK OF BACKUP FILES BY CALLS
C                                      OF SAVENET.

C       VARIABLES
C       'FEEDER' IS THE NUMBER OF THE FEEDER THAT WILL BE SAVED.

C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        INCLUDE 'CONSTANTS.FOR'
        INCLUDE 'COMMON.FOR'

        INTEGER FEEDER
        INTEGER POI(2),OLD(2)
        INTEGER I,J
        INTEGER DIR,BRA
        LOGICAL READY

        OPEN(2,FILE='HISTORY',STATUS='NEW',ACCESS='SEQUENTIAL')
        POI(1)=FEETYP
        POI(2)=FEEDER
        WRITE(2,*)FEEDER,VIOVEC(FEEDER,1),VIOVEC(FEEDER,2)
        DIR=DOWN
        BRA=3
        OLD(1)=POI(1)
        OLD(2)=POI(2)
        READY=.FALSE.
```

```
 10   IF (.NOT. READY) THEN

C     TRAVERSES THE TREE

        CALL MOVE(POI,OLD,DIR,BRA,.FALSE.)
        CALL DIREC(POI,OLD,DIR,BRA)
        IF (DIR.EQ.UP) THEN

C       FIRST IS THE POINT WRITTEN.
C       THEN DIFFERENT DATA THAT MAY BE CHANGED ARE WRITTEN DEPENDING
C       ON THE TYPE OF THE POINT

          WRITE(2,*)POI(1),POI(2)
          IF (POI(1).EQ.SWITYP) THEN
            WRITE(2,*)SWITCH(POI(2),1),SWITCH(POI(2),2),SWITCH(POI(2),5)
            I=SWITCH(POI(2),2)
            I=SWITCH(POI(2),I)
          ELSEIF (POI(1).EQ.NODTYP) THEN
            WRITE(2,*)NODETO(POI(2),1),NODETO(POI(2),4)
            WRITE(2,*)NODEPH(POI(2),1),NODEPH(POI(2),2),NODEPH(POI(2),3)
            I=NODETO(POI(2),1)
            I=NODETO(POI(2),I)
          ELSEIF (POI(1).EQ.BRATYP) THEN
            WRITE(2,*) BRAPOI(POI(2),2)
            DO 20 J=3,(2+BRAPOI(POI(2),1))
              WRITE(2,*)BRAPH(POI(2),J,1),BRAPH(POI(2),J,2)
 20         CONTINUE
            WRITE(2,*)BRAPOI(POI(2),2)
            I=BRAPOI(POI(2),2)
            I=BRAPOI(POI(2),I)
          ELSEIF (POI(1).EQ.FEETYP) THEN
            I=MSFEED(POI(2),2)
          ELSEIF (POI(1).EQ.TRATYP) THEN
            WRITE(2,*)TRAFOS(POI(2),3),TRAFOS(POI(2),4)
            WRITE(2,*)TRAFOS(POI(2),5),TRAFOS(POI(2),6)
            WRITE(2,*)TRAFOS(POI(2),7)
          ELSE
            WRITE(*,*)'        WRONG TYPE IN SAVENE'
            STOP
          ENDIF
          IF (POI(1).NE.TRATYP) THEN
            WRITE(2,*)CABPHY(I,4)
          ENDIF
        ENDIF
        READY=(POI(1).EQ.FEETYP)
        GOTO 10
      ENDIF
      DIR=UP
      READY=.FALSE.
 30   IF (.NOT. READY) THEN

C     EVEN THE TOPOLOGY ABOVE THE FEEDER IS SAVED

        CALL STEPUP(POI,.FALSE.)
        IF (DIR.EQ.UP) THEN
          WRITE(2,*)POI(1),POI(2)
          IF (POI(1).EQ.SWITYP) THEN
            WRITE(2,*)SWITCH(POI(2),1),SWITCH(POI(2),2),SWITCH(POI(2),5)
            I=SWITCH(POI(2),2)
            I=SWITCH(POI(2),I)
          ELSEIF (POI(1).EQ.BRATYP) THEN
            WRITE(2,*) BRAPOI(POI(2),2)
            DO 40 J=3,(2+BRAPOI(POI(2),1))
              WRITE(2,*)BRAPH(POI(2),J,1),BRAPH(POI(2),J,2)
 40         CONTINUE
            WRITE(2,*)BRAPOI(POI(2),2)
```

```
            I=BRAPOI(POI(2),2)
            I=BRAPOI(POI(2),I)
         ELSEIF (POI(1).EQ.FEETYP) THEN
            I=MSFEED(POI(2),2)
         ELSEIF (POI(1).EQ.TRATYP) THEN
            WRITE(2,*)TRAFOS(POI(2),3),TRAFOS(POI(2),4)
            WRITE(2,*)TRAFOS(POI(2),5),TRAFOS(POI(2),6)
            WRITE(2,*)TRAFOS(POI(2),7)
         ELSE
            WRITE(*,*)'       WRONG TYPE IN SAVENE'
            STOP
         ENDIF
         IF (POI(1).NE.TRATYP) THEN
            WRITE(2,*)CABPHY(I,4)
         ENDIF
       ENDIF
       READY=POI(1).EQ.TRATYP
       GOTO 30
     ENDIF
     CLOSE(2,STATUS='KEEP')
     RETURN
     END


     SUBROUTINE RESNET
C    CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C    AUTHOUR MAGNUS BERGSTRAND

C    VERSION 1

C    DATE 1989-10-05

C    PURPOSE
C    READS BACK A FORMER STATE WHEN A LOAD TRANSFER HAS FAILED.

C    DESCRIPTION
C    OPENS AND READS THE FILE 'HISTORY.DAT' (ALWAYS THE TOP OF THE STACK
C    OF BACKUP FILES)

C    SEE ALSO

C    FILES        ROUTINES      COMMENTS
C    CONSTANTS                  ALL CONSTANTS DECLARED
C    COMMON                     ALL COMMON VARIABLES DECLARED
C    STEP2        SAVENE        SAVES A FORMER STATE IN THE FILE 'HISTORY.DAT'

C    CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

     INCLUDE 'CONSTANTS.FOR'
     INCLUDE 'COMMON.FOR'

     INTEGER POI(2),I,J

     OPEN(2,FILE='HISTORY',STATUS='OLD',ACCESS='SEQUENTIAL')
     READ(2,*)I,VIOVEC(I,1),VIOVEC(1,2)
 10  READ(2,*,ERR=40)POI(1),POI(2)
     IF (POI(1).EQ.SWITYP) THEN
       J=SWITCH(POI(2),1)
       READ(2,*)SWITCH(POI(2),1),SWITCH(POI(2),2),SWITCH(POI(2),5)
       I=SWITCH(POI(2),2)
       I=SWITCH(POI(2),I)
       IF (J.NE.SWITCH(POI(2),1)) THEN
         CHGSWI(POI(2))=.NOT. CHGSWI(POI(2))
       ENDIF
     ELSEIF (POI(1).EQ.NODTYP) THEN
```

```
         READ(2,*)NODETO(POI(2),1),NODETO(POI(2),4)
         READ(2,*)NODEPH(POI(2),1),NODEPH(POI(2),2),NODEPH(POI(2),3)
         I=NODETO(POI(2),1)
         I=NODETO(POI(2),I)
      ELSEIF (POI(1).EQ.BRATYP) THEN
         READ(2,*) BRAPOI(POI(2),2)
         J=BRAPOI(POI(2),2)
         J=BRAPOI(POI(2),J)
         I=J
         DO 20 J=3,(2+BRAPOI(POI(2),1))
           READ(2,*)BRAPH(POI(2),J,1),BRAPH(POI(2),J,2)
 20      CONTINUE
         READ(2,*)BRAPOI(POI(2),2)
         I=BRAPOI(POI(2),2)
         I=BRAPOI(POI(2),I)
      ELSEIF (POI(1).EQ.FEETYP) THEN
         I=MSFEED(POI(2),2)
      ELSEIF (POI(1).EQ.TRATYP) THEN
         READ(2,*)TRAFOS(POI(2),3),TRAFOS(POI(2),4)
         READ(2,*)TRAFOS(POI(2),5),TRAFOS(POI(2),6)
         READ(2,*)TRAFOS(POI(2),7)
      ELSE
         WRITE(*,*)'       WRONG TYPE IN RESNET'
         STOP
      ENDIF
      IF (POI(1).NE.TRATYP) THEN
         READ(2,*)CABPHY(I,4)
      ENDIF
      GOTO 10
 40   CLOSE(2,STATUS='DELETE')
      RETURN
      END


      SUBROUTINE STAGE2(FEE,OK,FIRST,OUTFEE,FEE2,I)
C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     AUTHOUR MAGNUS BERGSTRAND

C     VERSION 1

C     DATE 1989-10-05

C     PURPOSE
C     TRANSFERS LOAD TO A SECONDARY SUPPORT FEEDER VIA A PRIMARY SUPPORT
C     FEEDER.

C     METHOD
C     THE METHOD IS DESCRIBED IN 'SERVICE RESTORATION IN ELECTRIC
C     DISTRIBUTION NETWORKS' BY MAGNUS BERGSTRAND.
C     THE METHOD WAS FIRST DESCRIBED BY AOKI IN
C     'A NEW ALGORITHM FOR SERVICE RESRORATION IN DISTRIBUTION SYSTEMS'
C     IEEE/PES WINTER MEETING,NEW YORK 1989 PAPER 89 WM 085-2 PWRD

C     DESCRIPTION
C     LOAD IS TRANSFERED TO A PRIMARY SUPPORT FEEDER CAUSING A VIOLATION.
C     THIS VIOLATION IS THEN REMOVED BY TRANSFERED LOAD TO A SECONDARY
C     SUPPORT FEEDER

C     SEE ALSO

C     FILES         ROUTINES      COMMENTS
C     CONSTANTS                   ALL CONSTANTS DECLARED
C     COMMON                      ALL COMMON VARIABLES DECLARED
C     STEP2         STEP2         PROVIDES THE FRAMEWORK TO STAGE1 AND
C                                 STAGE2
```

```
C      STEP2          STAGE1         FIRST STAGE SUPPORT. CAN BE FORCED TO ALLOW
C                                    VIOLATION IN THE SUPPORT FEEDER.
C      STEP2          DELHIS         READS BACK THE FORMER STATE

C      VARIABLES
C      FEE IS THE NUMBER OF THE VIOLATED FEEDER
C      OK=FALSE INDICATES THAT VIOLATION IS ALLOWED IN THE SECONDARY SUPPORT
C      FEEDER. (THE ALGORITHM CAN BE GENERALIZED WITH A THIRD,FOURTH...
C      STAGE SUPPORT)
C      FIRST INDICATES FIRST CALL FOR FEEDER 'FEE'
C      OUTFEE (OUTPARAMETER) IS THE NUMBER OF THE VIOLATED PRIMARY
C      SUPPORT FEEDER. IT IS ONLY VALID WHEN OK=F.
C      FEE2 (OUTPARAMETER) IS THE NUMBER OF THE VIOLATED SECONDARY
C      SUPPORT FEEDER. IT IS ONLY VALID WHEN OK=F.
C      'I' IS THE NUMBER OF SUCCESFUL CALLS OF FIRST STAGE SUPPORT.
C      2*I IS THE NUMBER OF BACKUP FILES IN THE STACK.

C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

       INCLUDE 'CONSTANTS.FOR'
       INCLUDE 'COMMON.FOR'

       INTEGER FEE,OUTFEE,I
       LOGICAL OK,FIRST
       INTEGER FEE2,SLASK
       LOGICAL READY,FIRST1,FIRST2,VIOLTE

       EXTERNAL VIOLTE

       WRITE(*,*)'ENTERS STAGE 2'
       IF (OK) THEN

C      NO VIOLATION IS ALLOWED IN THE SECONDARY SUPPORT FEEDER

         FIRST1=.TRUE.
         WRITE(*,*)'STAGE2 CALLS STAGE1 ******************************'

C      ALLOWS VIOLATION IN THE PRIMARY SUPPORT FEEDER

         OK=.FALSE.
         CALL STAGE1(FEE,OK,FIRST1,FEE2)
         FIRST2=.FALSE.
         FIRST1=.FALSE.
         READY=.NOT. OK
         I=0
 10      IF (.NOT. READY) THEN

C        DURING THIS LOOP THE VIOLATION IN THE PRIMARY SUPPORT FEEDER
C        IS TRIED TO BE ELIMINATED
C        THE VARIABEL I IS INCREMENTED FOR EACH SUCCESFUL LOAD TRANSFER
C        TO A SECONDARY SUPPORT FEEDER.

           WRITE(*,*)'LOOP IN STAGE 2 **********'
           OK=.TRUE.
           CALL STAGE1(FEE2,OK,FIRST2,SLASK)
           IF (.NOT. OK) THEN

C        IT WAS NOT POSSIBLE TO TRANSFER LOAD FROM THE PRIMARY SUPPORT FEEDER
C        TO ANY SECONDARY SUPPORT FEEDER
C        READ BACK THE FORMER STATE
C        TRY A NEW PRIMARY SUPPORT FEEDER

             CALL DELHIS(I)
             I=0
             OPEN(1,FILE='STAGE',STATUS='OLD',ACCESS='SEQUENTIAL')
             CLOSE(1,STATUS='DELETE')
```

```
            CALL  STAGE1(FEE,OK,FIRST1,FEE2)
            READY=(.NOT. OK)
          ELSE
            READY=.NOT. VIOLTE(FEE2)
            I=I+1
          ENDIF
          GOTO 10
        ENDIF
      ELSEIF (FIRST) THEN

C     OK IS FALSE. VIOLATIONS ARE ALLOWED IN THE SECONDARY SUPPORT FEEDER

        FIRST=.FALSE.
        FIRST1=.TRUE.
        OK=.FALSE.
        CALL STAGE1(FEE,OK,FIRST1,FEE2)

C       VIOLATIONS ARE ALLOWED IN THE PRIMARY SUPPORT FEEDER

        FIRST2=.TRUE.
        FIRST1=.FALSE.
        READY=(.NOT. OK)
        I=0
 40     IF (.NOT. READY) THEN
          WRITE(*,*)'LOOP IN STAGE 2 ***********'
          OK=.TRUE.
          CALL STAGE1(FEE2,OK,FIRST2,OUTFEE)

C         VIOLATIONS IN THE SECONDARY SUPPORT FEEDER WAS ALLOWED

          IF (.NOT. OK) THEN

C           NO POSSIBLE LOAD TRANSFER TO ANY SECONDARY SUPPORT FEEDER WAS FOUND
C           READ BACK FORMER STATE
C           FIND A NEW POSSIBLE PRIMARY SUPPORT FEEDER

            READY=.TRUE.
            OK=.FALSE.
            CALL STAGE1(FEE2,OK,FIRST2,OUTFEE)
            IF (.NOT. OK) THEN
              CALL DELHIS(I)
              I=0
              OPEN(1,FILE='STAGE',STATUS='OLD',ACCESS='SEQUENTIAL')
              CLOSE(1,STATUS='DELETE')
              CALL STAGE1(FEE,OK,FIRST1,FEE2)

C             A NEW VIOLATED SECONDARY SUPPORT FEEDER WAS TRIED

              READY=(.NOT. OK)
            ELSE
              I=I+1
            ENDIF
          ELSE

C         THE ROUTINE MANAGED TO TRANSFER LOAD TO A NOW VIOLATED SECONDARY
C         SUPPORT FEEDER

            READY=.NOT. VIOLTE(FEE2)
            I=I+1
          ENDIF
          GOTO 40
        ENDIF
      ELSE

C       THERE HAVE BEEN AN UNSUCCESFUL THIRD STAGE SUPPORT
C       TRY TO TRANSFER LOAD TO A NEW SECONDARY SUPPORT FEEDER
```

```
      FIRST2=.FALSE.
      FIRST1=.FALSE.
      READY=.FALSE.
      I=0
 50   IF (.NOT. READY) THEN
        OK=.FALSE.
        CALL STAGE1(FEE,OK,FIRST2,SLASK)

C       LOAD WAS TRANSFERED TO A PRIMARY SUPPORT FEEDER

        FIRST2=.FALSE.
        IF (.NOT. OK) THEN

C         THE LOAD TRANSFER TO ANY PRIMARY SUPPORT FEEDER WAS
C         UNSUCCESFUL.

          READY=.TRUE.
        ELSE
          CALL STAGE1(SLASK,OK,FIRST1,FEE2)
          IF (OK) THEN

C           LOAD WAS TRANSFERED TO THE SECONDARY SUPPORT FEEDER

            READY=.TRUE.
          ELSE

C         LOAD COULD NOT BE TRANSFERED TO ANY SECONDARY SUPPORT FEEDER
C         READ BACK OLD STATE AND TRY AGAIN

            CALL DELHIS(I)
            I=0
          ENDIF
          GOTO 50
        ENDIF
      ENDIF
    ENDIF
    WRITE(*,*)'EXIT STAGE 2 OK=',OK
    RETURN
    END



    SUBROUTINE STAGE3(FEE,OK,FIRST,OUTFEE)
C   CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C   AUTHOUR MAGNUS BERGSTRAND

C   VERSION 1

C   DATE 1989-10-05

C   PURPOSE
C   TRANSFERS LOAD TO TERTIARY SUPPORT FEEDER

C   DESCRIPTION
C   FIRST A FIRST STAGE SUPPORT IS TRIED ALLOWING VIOLATIONS
C   IN THE SUPPORT FEEDER. THEN IS SECOND STAGE SUPPORT APPLIED
C   TO THIS VIOLATED SUPPORT FEEDER.

C   SEE ALSO

C   FILES          ROUTINES      COMMENTS
C   CONSTANTS                    ALL CONSTANTS DECLARED
C   COMMON                       ALL COMMON VARIABLES DECLARED
```

```
C       STEP2           STAGE1          FIRST STAGE SUPPORT
C       STEP2           STAGE2          SECOND STAGE SUPPORT

C       VARIABLES
C       FEE IS THE NUMBER OF THE VIOLATED FEEDER
C       OK INDICATES THAT NO VIOLATION IS ALLOWED IN THE TERTIARY SUPPORT
C       FEEDER
C       FIRST INDICATES WETHER A FORMER CALL WITH FEEDER=FEE HAS BEEN MADE
C       OUTFEE IS THE NUMBER OF THE TERTIARY SUPPORT FEEDER
C       THESE LAST VARIABLES ARE WITHOUT MEANING IF NOT A FOURTH STAGE
C       SUPPORT IS IMPLEMENTED

C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC


        INCLUDE 'CONSTANTS.FOR'
        INCLUDE 'COMMON.FOR'

        INTEGER FEE,OUTFEE
        LOGICAL OK,FIRST

        IF (OK) THEN
          FIRST1=.TRUE.
          OK=.FALSE.
          CALL STAGE2(FEE,OK,FIRST1,OUTF,FEE2,NR)
 10       IF (.NOT. READY) THEN
            CALL STAGE1(OUTF,OK,.TRUE.,SLASK)
            IF (.NOT. OK) THEN
              CALL DELHIS(NR)
              NR=0
              OPEN(1,FILE='STAGE',STATUS='OLD',ACCESS='SEQUENTIAL')
              CLOSE(1,STATUS='DELETE')
              CALL STAGE2(FEE,FIRST1,OUTF,FEE2,NR)
              READY=(.NOT. OK)
            ENDIF
            GOTO 10
          ENDIF
        ENDIF
        RETURN
        END


        SUBROUTINE DELHIS(I)
C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C       AUTHOUR MAGNUS BERGSTRAND

C       VERSION 1

C       DATE 1989-10-05

C       PURPOSE
C       READS BACK THE STATE OF NETWORK AS IT WAS BEFORE AN UNSUCCESFULL
C       LOAD TRANSFER

C       SEE ALSO

C       FILES           ROUTINES        COMMENTS
C       CONSTANTS                       ALL CONSTANTS DECLARED
C       COMMON                          ALL COMMON VARIABLES DECLARED
C       STEP2           RESNET          READS THE FILE 'HISTORY.DAT' CONTAINING
C                                       PARTS OF THE STATE

C       VARIABLES
C       'I' INDICATES THE SIZE OF THE STACK OF BACK UP FILES.
C       2*I IS THE NUMBER OF FILES
```

```
C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      INTEGER I,J

      DO 10 J=1,I
        CALL RESNET
        CALL RESNET
 10   CONTINUE
      RETURN
      END



      SUBROUTINE STEP2(VIO)
C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     AUTHOUR MAGNUS BERGSTRAND

C     VERSION 1

C     DATE 1989-10-05

C     PURPOSE
C     PROVIDES THE FRAMEWORK FOR THE FIRST AND SECOND STAGE SUPPORT

C     METHOD
C     THE ALGORITHM IS DESCRIBED IN 'SERVICE RESTORATION IN ELECTRIC
C     DISTRIBUTION NETWORKS' BY MAGNUS BERGSTRAND
C     THE METHOD WAS FIRST DESCRIBED BY AOKI IN
C     'A NEW ALGORITHM FOR SERVICE RESRORATION IN DISTRIBUTION SYSTEMS'
C     IEEE/PES WINTER MEETING,NEW YORK 1989 PAPER 89 WM 085-2 PWRD

C     DESCRIPTION
C     ALL FEEDERS ARE CHECKED IN ORDER IF THEY ARE VIOLATED
C     FOR ANY VIOLATED FEEDER:
C     FIRST STAGE SUPPORT
C     IF FAILURE: SECOND STAGE SUPPORT
C     IF SUCCES REPEAT IT

C     SEE ALSO

C     FILES         ROUTINES      COMMENTS
C     CONSTANTS                   ALL CONSTANTS DECLARED
C     COMMON                      ALL COMMON VARIABLES DECLARED
C     STEP2         STAGE1        FIRST STAGE SUPPORT
C     STEP2         STAGE2        SECOND STAGE SUPPORT
C     STEP2         VIOLTE        LOGICAL FUNCTION. CHECKS IF A FEEDER IS VIOLATEI

C     VARIABLES
C     VIO IS AN OUTPARAMETER. IT INDICATES WHETHER ANY FEEDER IS VIOLATED.

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      INCLUDE 'CONSTANTS.FOR'
      INCLUDE 'COMMON.FOR'

      LOGICAL OK,VIO
      INTEGER FEE,OUTFEE
      INTEGER I,NR,J
      LOGICAL VIOLTE

      EXTERNAL VIOLTE

      VIO=.FALSE.
      DO 10 I=1,MAXNRF
```

```
         FEE=I
         IF (VIOLTE(FEE)) THEN
C
           A VIOLATED FEEDER IS FOUND

           OK=.TRUE.
 20        IF (OK) THEN
             CALL STAGE1(FEE,OK,.TRUE.,OUTFEE)
             IF (.NOT. OK) THEN

C              FIRST STAGE SUPPORT FAILED
C              TRY A SECOND STAGE SUPPORT

               OK=.TRUE.
               CALL STAGE2(FEE,OK,.TRUE.,OUTFEE,NR)
               IF (OK) THEN
                 DO 30 J=1,2*NR

C                  REMOVE USED FILES

                   OPEN(3,FILE='HISTORY',STATUS='OLD',ACCESS='SEQUENTIAL')
                   CLOSE(3,STATUS='DELETE')
 30              CONTINUE
                 OPEN(3,FILE='STAGE',STATUS='OLD',ACCESS='SEQUENTIAL')
                 CLOSE(3,STATUS='DELETE')
               ENDIF
             ELSE

C              REMOVE USED FILES

               OPEN(1,FILE='HISTORY',STATUS='OLD',ACCESS='SEQUENTIAL')
               CLOSE(1,STATUS='DELETE')
               OPEN(1,FILE='HISTORY',STATUS='OLD',ACCESS='SEQUENTIAL')
               CLOSE(1,STATUS='DELETE')
             ENDIF
             OK=(OK .AND. VIOLTE(FEE))
             GOTO 20
           ENDIF
         ENDIF
         VIO=VIO .OR. VIOLTE(FEE)
 10    CONTINUE
       RETURN
       END



       SUBROUTINE IMPROV(FEE,SWI)
C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C      AUTHOUR MAGNUS BERGSTRAND

C      VERSION 1

C      DATE 1989-10-05

C      PURPOSE
C      EXAMINES DIFFERENT POSSIBLE SWITCHES SUITABLE FOR A LOAD TRANSFER
C      THE BEST SWITCH, THAT IS THE ONE THAT WILL CAUSE A SMALL REDUCTION
C      OF THE VIOLATION VECTOR WILL BE CHOSEN

C      DESCRIPTION
C      TRAVERSES THE TREE WITH FEEDER 'FEE' AS ROOT.
C      WHEN A LEAF IS FOUND A CALL OF EVAL IS PERFORMED.

C      SEE ALSO
```

```
C       FILES           ROUTINES        COMMENTS
C       CONSTANTS                       ALL CONSTANTS DECLARED
C       COMMON                          ALL COMMON VARIABLES DECLARED
C       MOVING          MOVE            TAKES ONE STEP UP OR DOWN
C       MOVING          DIREC           COMPUTES NEW DIRECTION
C       STEP2           EVAL            EVALUATES THE NUMBER OF SWITCH THAT IS
C                                       THE MOST SUITABLE TO PERFORM A LOAD CURTAILMENT

C       VARIABLES
C       'FEE' IS THE NUMBER OF THE VIOLATED FEEDER CAUSING THE LOAD CURTAILMENT
C       SWI (OUTPARAMETER) IS THE NUMBER OF THE SWITCH TO OPEN FOR THE
C       BEST LOAD CURTAILMENT

C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        INCLUDE 'CONSTANTS.FOR'
        INCLUDE 'COMMON.FOR'

        INTEGER FEE,SWI,P(2),OLD(2)
        INTEGER DIR,BRA
        LOGICAL READY,ENDS,FIRST

        P(1)=FEETYP
        P(2)=FEE
        DIR=DOWN
        BRA=3
        FIRST=.TRUE.
        READY=.FALSE.
 10     IF (.NOT. READY) THEN
          IF (DIR.EQ.DOWN) THEN
            ENDS=.TRUE.
          ENDIF
          CALL MOVE(P,OLD,DIR,BRA,.FALSE.)
          CALL DIREC(P,OLD,DIR,BRA)
          IF (DIR.EQ.UP .AND. ENDS) THEN
            CALL EVAL(P,SWI,FIRST,FEE)
            ENDS=.FALSE.
          ENDIF
          READY=P(1).EQ.FEETYP
          GOTO 10
        ENDIF
        RETURN
        END


        SUBROUTINE EVAL(P,SWI,FIRST,FEE)
C       CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C       AUTHOUR MAGNUS BERGSTRAND

C       VERSION 1

C       DATE 1989-10-05

C       PURPOSE
C       COMPUTES THE NUMBER OF THE SWITCH MOST SUITABLE FOR A LOAD
C       CURTAILMENT SO FAR

C       DESCRIPTION
C       STARTING FROM A LEAF THE SUBROUTINE FINDS THE FIRST SWITCH
C       TOWARDS THE TOP. WHEN A SWITCH IS FOUND THIS IS USED AS A ROOT IN
C       A TREE. THE TREE IS TRAVERSED AND IN THE MEAN TIME THE SUBROUTINE
C       COMPUTES THE MAGNITUDE OF THE WITHDRAWAL OF THE VIOLATION VECTOR.
C       BEGINING AT THE ROOT IS NECESSARY DUE TO THE EXISTANCE OF BRANCH POINTS
C       LOAD IN MORE THAN ONE BRANCH MAY BE CUT OFF
C       THE BEST MAGNITUDE OF VIOLATIONS WITHDRAWAL AND THE
```

```
C      NUMBER OF THE SWITCH CAUSING THE WITHDRAWAL ARE SAVED
C      BETWEEN CALLS

C      SEE ALSO

C      FILES         ROUTINES      COMMENTS
C      CONSTANTS                   ALL CONSTANTS DECLARED
C      COMMON                      ALL COMMON VARIABLES DECLARED
C      MOVING        STEPUP        TAKES ONE STEP UP
C      MOVING        STEPDO        TAKES ONE STEP DOWN
C      MOVING        MOVE          TAKES ONE STEP UP OR DOWN
C      MOVING        DIREC         COMPUTES NEW DIRECTION
C      STEP2         MKESUM        DECIDES THE NUMBER OF THE SWITCH DUE TO
C                                  A STAGE SUPPORT
C      VARIABLES
C      P IS THE LEAF (TYPE,NR)
C      SWI (OUTPARAMETER) IS THE NUMBER OF THE SWITCH SO FAR
C      FIRST INDICATES FIRST CALL IN NEW STATE
C      FEE IS THE NUMBER OF THE VIOLATED FEEDER

C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

       INCLUDE 'CONSTANTS.FOR'
       INCLUDE 'COMMON.FOR'

       INTEGER POI(2),P(2),SWI,FEE,TEMP,DIR,BRA,OLD(2)
       LOGICAL FIRST,READY
       REAL BEST,VOLSUM,SUM,CURSUM,LOAD,SLASK
       REAL VIOSUM,SQRT,Z,MIN,ALPHA
       REAL CUR1

       INTRINSIC SQRT,MIN

       EXTERNAL VIOSUM

       SAVE TEMP
       SAVE BEST

       IF (FIRST) THEN
         BEST=1.0E+30
         FIRST=.FALSE.
         TEMP=0
       ENDIF
       VOLSUM=0.0
       CURSUM=0.0
       CUR1=0.0
       LOAD=0.0
       POI(1)=P(1)
       POI(2)=P(2)
       READY=.FALSE.
 5     IF (.NOT. READY) THEN
         CALL STEPUP(POI,.FALSE.)
         READY=POI(1).EQ.SWITYP
         GOTO 5
       ENDIF

C      THE ROOT IS FOUND

       TEMP=POI(2)
       DIR=DOWN
       BRA=3
       READY=.FALSE.
 10    IF (.NOT. READY) THEN
         IF (DIR.EQ.UP) THEN

C            THE NUMBER OF A CABLE IS STORED
```

```
          IF (POI(1).EQ.SWITYP) THEN
            I=SWITCH(POI(2),2)
            I=SWITCH(POI(2),I)
          ELSEIF (POI(1).EQ.NODTYP) THEN
            I=NODETO(POI(2),1)
            I=NODETO(POI(2),I)

C           NODE PRIORITY,CURRENT REDUCTION,LOAD REDUCTION AND
C           REDUCTION IN VOLTAGE DROP IS COMPUTED

            ALPHA=ALFA(POI(2))
            Z=SQRT((NODEPH(POI(2),4)**2)+(NODEPH(POI(2),5)**2))
            CUR1=CUR1+NODEPH(POI(2),3)/Z
            LOAD=LOAD+SQRT(NODEPH(POI(2),1)**2+NODEPH(POI(2),2)**2)
            IF (VOLLIM.GT.NODEPH(POI(2),3)) THEN
              VOLSUM=VOLSUM+VOLLIM-NODEPH(POI(2),3)
            ENDIF
          ELSEIF (POI(1).EQ.BRATYP) THEN
            I=BRAPOI(POI(2),2)
            I=BRAPOI(POI(2),I)
          ENDIF
          IF (CABPHY(I,4).GT.CABPHY(I,3)) THEN

C           CHECK OF LINE CAPACITY CONSTRAINTS

            IF ((CABPHY(I,4)-CABPHY(I,3)).GT.CURSUM) THEN
               CURSUM=MIN(CUR1,(CABPHY(I,4)-CABPHY(I,3)))
            ENDIF
          ENDIF
        ENDIF

C       TRAVERSES THE TREE

        CALL MOVE(POI,OLD,DIR,BRA,.FALSE.)
        CALL DIREC(POI,OLD,DIR,BRA)
        IF (POI(1).EQ.SWITYP .AND. POI(2).EQ.TEMP) THEN
          READY=.TRUE.
        ENDIF
        GOTO 10
      ENDIF
      READY=.FALSE.
 20   IF (.NOT. READY) THEN

C       THE SAME TEST IS THEN MADE UP TO THE FEEDER

        IF (POI(1).EQ.SWITYP) THEN
          I=SWITCH(POI(2),2)
          I=SWITCH(POI(2),I)
        ELSEIF (POI(1).EQ.NODTYP) THEN
          I=NODETO(POI(2),1)
          I=NODETO(POI(2),I)
          ALPHA=ALFA(POI(2))
        ELSEIF (POI(1).EQ.BRATYP) THEN
          I=BRAPOI(POI(2),2)
          I=BRAPOI(POI(2),I)
        ENDIF
        IF (CABPHY(I,4).GT.CABPHY(I,3)) THEN
          IF ((CABPHY(I,4)-CABPHY(I,3)).GT.CURSUM) THEN
             CURSUM=MIN(CUR1,(CABPHY(I,4)-CABPHY(I,3)))
          ENDIF
        ENDIF
        CALL STEPUP(POI,.FALSE.)
        IF (POI(1).EQ.FEETYP) THEN
          READY=.TRUE.
        ENDIF
```

```
        GOTO 20
      ENDIF

C     COMPUTE AND DECIDE

      SUM=SQRT((CURSUM**2)+(VOLSUM**2))
      SLASK=0.01+VIOSUM(FEE)-SUM
      SLASK=LOAD*(SLASK+BETA)*ALPHA
      IF (SLASK.LT.BEST .AND. SLASK.GT.0.0) THEN
        BEST=SLASK
        SWI=TEMP
      ENDIF

C     FINAL REMARK
C     BEST AND SWI IS SAVED

      RETURN
      END



      SUBROUTINE STEP3
C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     AUTHOUR MAGNUS BERGSTRAND

C     VERSION 1

C     DATE 1989-10-05

C     PURPOSE
C     WHEN NO MORE LOAD TRANSFER IS POSSIBLE LOAD MUST BE CUT OFF.
C     AFTER A LOAD CURTAILMENT THERE IS A POSSIBILITY TO RESTORE
C     LOAD AMONG FORMER VIOLATION FEEDERS.

C     METHODE
C     THE METHODE IS DESCRIBED IN 'SERVICE RESTORATION IN ELECTRIC
C     DISTRIBUTION NETWORKS' BY MAGNUS BERGSTRAND.
C     THE METHOD WAS FIRST DESCRIBED BY AOKI IN
C     'A NEW ALGORITHM FOR SERVICE RESGRORATION IN DISTRIBUTION SYSTEMS'
C     IEEE/PES WINTER MEETING,NEW YORK 1989 PAPER 89 WM 085-2 PWRD

C     DESCRIPTION
C     THIS SUBROUTINE HANDLES BOTH STEP3 AND STEP4 IN THE ALGORITHM
C     (LOAD CURTAILMENT AND RESTORATION OF CURTAILED LOADS)
C     THE SUBROUTINE WILL SEARCH FOR A VIOLATED FEEDER.
C     WHEN A VIOLATED FEEDER IS FOUND A CALL OF IMPROV IS MADE, THUS
C     PRODUCING THE NUMBER OF THE BEST SWITCH FOR A LOAD CURTAILMENT.
C     SWITCHING OPERATION IS PERFORMED AND THE NEW STATE WILL BE COMPUTED
C     THE NUMBER OF THE SWITCH IS SAVED ON THE FILE 'CURTAI.DAT' FOR LATER
C     USE
C     A DESCRIPTION OF THE FOURTH STEP IN THE ALGORITHM IS FOUND FURTHER DOWN

C     SEE ALSO

C     FILES         ROUTINES      COMMENTS
C     CONSTANTS                   ALL CONSTANTS DECLARED
C     COMMON                      ALL COMMON VARIABLES DECLARED
C     MOVING        STEPUP        TAKES ONE STEP UP
C     MOVING        STEPDO        TAKES ONE STEP DOWN
C     MOVING        MOVE          TAKES ONE STEP UP OR DOWN
C     MOVING        DIREC         COMPUTES NEW DIRECTION
C     STEP2         IMPROV        COMPUTES THE NUMBER OF THE SWITCH THAT WILL BE
C                                 OPENED DUE TO A LOAD CURTAILMENT
C     COMP          COMPIM        COMPUTES IMPEDANCE
C     COMP          FILVOL        COMPUTES AND FILLS IN NODE VOLTAGES AND
```

```
C                                 CABLE CURRENTS
C      COMP          FILPOW       COMPUTES POWER
C      STEP2         SAVENE       SAVES THE STATE OF THE NETWORK
C      STEP2         RESNET       READS BACK FORMER STATE
C      STEP2         STAGE1       FIRST STAGE SUPPORT

C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

       INCLUDE 'CONSTANTS.FOR'
       INCLUDE 'COMMON.FOR'

       INTEGER I,FEE,OUTFEE,SWI,SWI2
       LOGICAL VIOLTE
       LOGICAL OK,FIRST,READY
       LOGICAL VIO
       REAL S1,S2,BSUM,SUM,MINI,SLASK,VIOSUM,LOAD
       INTEGER BSW1,BSW12,BFEE,FEEDER,TRA
       INTEGER POI(2),P(2),OLD(2),DIR,BRA

       EXTERNAL VIOSUM
       EXTERNAL VIOLTE

       WRITE(*,*)'ENTERS STEP 3'
       OPEN(2,FILE='CURTAI',STATUS='NEW',ACCESS='SEQUENTIAL')
       DO 10 I=1,MAXNRF
         FEE=I
         OK=.TRUE.
         FIRST=.TRUE.
         READY=.NOT. VIOLTE(FEE)
 15      IF (.NOT. READY) THEN

C          A VIOLATED FEEDER IS FOUND

           WRITE(*,*)'STAGE 3 WILL BE CUTTING LOAD IN FEEDER',FEE
 30        CALL IMPROV(FEE,SWI)

C          PERFORM SWITCHING OPERATIONS

           SWITCH(SWI,1)=OPEN
           SWINR(SWI)=2
           CHGSWI(SWI)=.NOT. CHGSWI(SWI)

C          SAVE NUMBER OF THE SWITCH ON 'CURTAI.DAT'

           WRITE(2,*) SWI

C          COMPUTE NEW STATE

           CALL COMPIM(FEE)
           TRA=MSFEED(FEE,1)
           CALL FILPOW(TRA)
           CALL FILVOL(FEE,VIO)
           READY=.NOT. VIO
           GOTO 15
         ENDIF
 10    CONTINUE
       ENDFILE 2
       REWIND 2

C      DESCRIPTION
C      HERE STEP 4, THE RESTORATION OF CURTAILED LOADS, BEGINS.
C      ONE AT A TIME THE CURTAILED LOADS ARE REENRGIZED IF IT IS
C      POSSIBLE. ONE CURTAILED SECTION MAY LACK ENERGIZED
C      NEIGHBOURS DUE TO REPEATED LOAD CURTAILMENTS IN THE SAME
C      BRANCH.
C      THEN A FIRST STAGE SUPPORT IS TRIED. THE LOAD
```

```
C     WILL ONCE AGAIN BE CUT OFF IF THIS FAILS

      S1=0.0
      S2=0.0
 40   READ(2,*,ERR=60)SWI

C       READ IN 'CURTAI.DAT'

        POI(1)=SWITYP
        POI(2)=SWI

C       THEN A SEARCH FOR THE FIRST SWITCH ABOVE THE READ SWITCH STARTS
C       IF THIS NEW SWITCH IS CLOSED IT IS POSSIBLE TO REENERGIZE THE
C       FORMER DEENERGIZED ZONE

        CALL STEPUP(POI,.FALSE.)
        READY=POI(1).EQ.SWITYP
 50     IF (.NOT. READY) THEN
          CALL STEPUP(POI,.FALSE.)
          READY=POI(1).EQ.SWITYP
          GOTO 50
        ENDIF
        IF (SWITCH(POI(2),1).EQ.CLOSED) THEN
          READY=.FALSE.
          DIR=DOWN
          P(1)=POI(1)
          P(2)=POI(2)
 55       IF (.NOT. READY) THEN
            CALL MOVE(P,OLD,DIR,BRA,.FALSE.)
            IF (P(1).EQ.NODTYP .AND. DIR.EQ.UP) THEN
              FEEDER=NODETO(P(2),4)
              S1=S1+NODEPH(P(2),1)
              S2=S2+NODEPH(P(2),2)
            ENDIF
            CALL DIREC(P,OLD,DIR,BRA)
            READY=(P(1).EQ.POI(1) .AND. P(2).EQ.POI(2))
            GOTO 55
          ENDIF

C         SAVE OLD STATE

          CALL SAVENE(FEEDER)

C         PERFORM SWITCHING OPERATION

          TRA=MSFEED(FEEDER,1)
          SWITCH(SWI,1)=CLOSED
          CHGSWI(SWI)=.NOT. CHGSWI(SWI)

C         COMPUTE NEW STATE

          CALL COMPIM(FEEDER)
          CALL FILPOW(TRA)
          CALL FILVOL(FEEDER,VIO)
          OK=.TRUE.
          FIRST=.TRUE.

C         TRY A FIRST STAGE SUPPORT

          CALL STAGE1(FEEDER,OK,FIRST,OUTFEE)
          IF (.NOT. OK) THEN

C           READ BACK OLD STATE

            CALL RESNET
            RESTOR(1)=RESTOR(1)-S1
```

```
            RESTOR(2)=RESTOR(2)-S2
          ENDIF
        ELSE

C         THE SWITCH WAS CLOSED
C         FIND OUT HOW BIG THE LOAD WAS THAT COULD NOT BE RESTORED

          READY=.FALSE.
          DIR=DOWN
          P(1)=POI(1)
          P(2)=POI(2)
 58       IF (.NOT. READY) THEN
            CALL MOVE(P,OLD,DIR,BRA,.FALSE.)
            IF (P(1).EQ.NODTYP .AND. DIR.EQ.UP) THEN
              FEEDER=NODETO(P(2),4)
              S1=S1+NODEPH(P(2),1)
              S2=S2+NODEPH(P(2),2)
            ENDIF
            CALL DIREC(P,OLD,DIR,BRA)
            READY=(P(1).EQ.POI(1) .AND. P(2).EQ.POI(2))
            GOTO 58
          ENDIF
          RESTOR(1)=RESTOR(1)-S1
          RESTOR(2)=RESTOR(2)-S2
        ENDIF
        GOTO 40
 60   CLOSE(2,STATUS='DELETE')
      RETURN
      END


      REAL FUNCTION VIOSUM(FEE)

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

C     AUTHOUR MAGNUS BERGSTRAND

C     VERSION 1

C     DATE 1989-10-05

C     PURPOSE
C     THIS FUNCTION COMPUTES THE EFFECTIVE LENGTH OF THE VIOLATION

C     FILES        ROUTINES     COMMENTS
C     STEP2        VIOLTE       INDICATES WHETHER A FEEDER IS VIOLATED OR NOT

C     VARIABLES
C     'FEE' IS THE NUMBER OF THE (PERHAPS) VIOLATED FEEDER

C     CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC


      INCLUDE 'CONSTANTS.FOR'
      INCLUDE 'COMMON.FOR'

      INTEGER FEE
      REAL SQRT

      INTRINSIC SQRT

      VIOSUM=SQRT(VIOVEC(FEE,1)**2+VIOVEC(FEE,2)**2)
      RETURN
      END
```