AN EFFICIENT ORDERING ALGORITHM TO IMPROVE SPARSE VECTOR METHODS

A. Gómez
Member IEEE
Dep. de Ingeniería
Eléctrica

L.G. Franquelo
Member IEEE
Dep. de Ingeniería
Electrónica, de Sistemas
y Automática

Univ. de Sevilla
Spain

Keywords: Sparse Techniques, Ordering Algorithms, Linear Equations.

Abstract.

This paper presents a new node ordering algorithm to enhance sparse vector methods. The proposed technique locally minimizes the number of non-zero elements of the inverse of the table of factors. It uses the cardinality of the set of nodes which precede each node in the path graph as a tie-break criterion in the minimum degree elimination process. Test results are included showing that the method performs better than previously published methods.

INTRODUCTION.

A great deal of Sparse Matrix problems still represent a challenge for Electrical Engineers, especially if the large scale of these problems and real-time requirements are considered. Nowadays, advances in computer hardware and development of efficient algorithms permit to afford certain problems that only a few years ago did not seem to have a practical solution.

Sparse Matrix techniques introduced in the sixties [1] had a great transcendence in the efficient solution of the power flow problem. More recently, Sparse Vector techniques were introduced to speed up a number of important power system problems [2]. Two kinds of papers can be found in the literature related with these topics: Basic developments and applications. Among these, reference [3] can be mentioned, where two Partial Refactorization methods are applied to Newton Load Flow, Fast Decoupled Load Flow and Security Analysis. Other recent papers related to applications deal with the following problems: Parallel Inversion of Sparse Matrices [4], Fault Analysis [5] and Linear Contingency Analysis [6]. The first kind of papers are devoted to the development of algorithms and basic concepts that permit to get better performances when these techniques are applied to particular problems [2, 7-9].

This paper falls under this category because it presents a new ordering algorithm of general application.

REVIEW OF SPARSE VECTOR CONCEPTS.

Consider the solution of the equation:

$$\mathbf{Ax=b} \qquad (1)$$

where **A** is sparse. If we restrict ourselves, for simplicity, to the case in which **A** is symmetric and positive definite (e.g. the matrix **B'** in the Fast Decoupled Load Flow), then **A** could be factorized as $\mathbf{U^tU}$. Once this requisite has been fulfilled, the standard forward and backward operations on the independent vector give the desired result, **x**.

In this paper the attention is focused on two kinds of problems:

a) Solution of (1) when either vector **b** has only a few non-zero elements or a small number of elements in the unknown vector **x** are needed (Sparse Vector problem).

b) Repeated solution of (1) when matrix **A** is slightly modified (Partial Matrix Refactorization problem).

It is a common practice to use an undirected graph associated to matrix **A** so that the elimination process can be easily visualized. Each time a row in **A** is eliminated, the graph must be updated by deleting the corresponding vertex and incident arcs and adding the required fill-ins, giving the reduced graph at this stage. Also, a filled graph may be associated to the matrix $\mathbf{U+U^t}$ which depends, obviously, on the ordering adopted during the elimination process, i.e., on the number of fill-ins generated.

As an example, consider the 10-node system shown in Figure 1. The structure of the resultant **U** matrix is shown in Figure 2, when the Minimum Degree strategy (MD) is adopted. The leftmost column indicates the order in which the nodes have been eliminated.

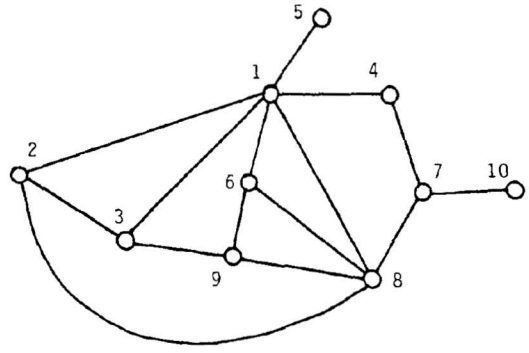

Fig. 1. 10-Node System.

```
      1234567890
 5    1X    X          1
10    2 X X            1
 4    3  XX X          2
 7    4   X O  X       2
 2    5    XXX X       3
 1    6     XXXX       3
 3    7      XOOX      3
 6    8       XXX      2
 8    9        XX      1
 9   10         X      0
```

X : Original element
O : Fill-in element

Fig. 2. Structure of U.

For both problems formerly mentioned it is interesting to introduce, moreover, other concepts, following [2].

First of all, a singleton is a vector with only one non-zero element (e.g. in location **k**). A path for a singleton is an ordered list of rows of **U** defined as follows:

1) Include **k** in the path. If **k** is the last row of **U**, exit.
2) Replace **k** with the first non-zero column in row **k** of **U**, and go to step 1).

If **b** is a singleton, then only the rows of its path are needed during the forward elimination process. Analogously, if only the **k**-th entry of **x** is wanted, only the rows of this path are strictly required during the backward substitution process. In this case the path is swept in the reverse order. These processes are called the Fast Forward (FF) and Fast Backward (FB) processes respectively. Taking into account the analogy between factorization and forward elimination processes, it is clear that only the rows involved in the appropriate path should be updated in **U** when the **k**-th row of **A** is modified (Partial Matrix Refactorization, or simply PMR).

A path graph may be built by union of all possible singleton paths. Figure 3 shows the path graph for matrix **U** in Figure 2.

The appropriate subset of this tree determines the rows involved in FF, FB or PMR processes when **b** has several non-zero elements, more than one element is wanted in **x**, or various rows are modified in **A** (provided in this case that the structure of **U** remains the same). Each row of **U** with d_i non-zero elements contributes with d_i mult-adds in the FF/FB process and $d_i(d_i+1)/2$ in PMR.

There is an interesting relationship between this path graph and the structure of U^{-1} which is the key for the development of more efficient ordering algorithms. The non-zero columns of the **k**-th row of U^{-1} give the path for the node at **k**-th position.

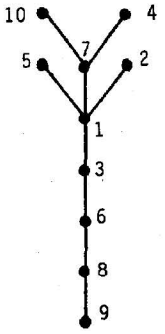

Fig. 3. Path Graph of the 10-Node System.

Conversely, the non-zero rows of the **k**-th column of U^{-1} give the subset of the tree which precedes the node at this position. Figure 4 shows the structure of U^{-1} for the same earlier example, and visualizes the mentioned relationship between U^{-1} and the path graph for the case of node 1 which is numbered at 6^{th} position.

```
      1234567890
 5    1X  * XXXXX
10    2 X X XXXXX
 4    3  XX XXXXX
 7    4   X XXXXX
 2    5    XXXXXX
 1    6     XXXXX
 3    7      XXXX
 6    8       XXX
 8    9        XX
 9   10         X
```

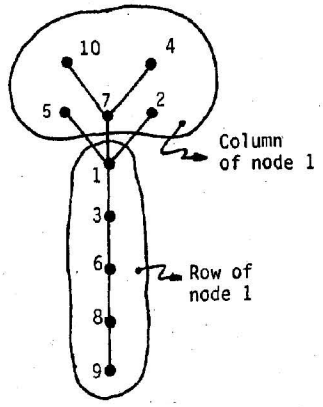

Fig. 4. Structure of U^{-1} and Relationship with the Path Graph.

From the preceding comments it is clear that the number of non-zero elements of U^{-1} is a measure of the average singleton path length. Actually, the number of non-zero elements in U^{-1} (including the diagonal) divided by the number of nodes gives the average singleton path length. Also the number of non-zero elements of **U** is a measure of the average number of mult-adds required to perform the FF/FB or the PMR process. Hence, what is required is that both **U** and U^{-1} remain as sparse as possible. Sometimes, a less sparse **U** is permissible if this is counteracted by the saving achieved in U^{-1}.

MOTIVATION AND DESCRIPTION OF THE ALGORITHM.

In order to allow comparisons the MD algorithm (or Tinney's second scheme) will be first described. This description will be somewhat conventional as the actual computer implementation will depend strongly on the storage-accessing scheme adopted (e.g., steps 4 and 5 below could be probably merged into only one step).

For this purpose the next vectors will be used:

f(i) : Position of node **i** in the final ordering. Initially set to zero; **f(i)=0** means that node **i** has not yet been considered.
D(i) : Degree of node **i** in the reduced graph. Initially set to the degree in the original graph.

Then, for a graph with **N** nodes:

Minimum Degree Algorithm (MD):

1) Let **k=1**.
2) Pick up a node **i** with **f(i)=0** such that **D(i)** is minimum. The ties are broken arbitrarily, although for programming convenience, the first or the last eligible node in the natural order is usually chosen. Set **f(i)=k**.
3) If **k** equals **N** then stop.
4) For each **j** adjacent to **i** such that **f(j)=0**, set **D(j)=D(j)-1**.
5) For each pair of nodes **m**, **n** adjacent to **i** but not adjacent to each other, such that **f(m)=f(n)=0**, create a new edge joining **m** and **n**, and increase **D(m)**, **D(n)** in one unit.
6) Set **k=k+1** and go to step 2).

The main weakness of this algorithm is the large number of ties which appear in step 2). The way these ties are broken may not have a strong influence on the

sparse structure of U [10], but this is not the case when U^{-1} or, equivalently, the average path length is considered. For instance, the average path length for the 118-node IEEE test system is 9.46 when the first minimum-degree node encountered in the natural order is chosen. This value rises to 11.77 when the last eligible node is taken. Examples could be found where just the opposite would happen. That is, the behaviour of the MD strategy, as far as the efficiency of sparse vector methods is concerned, depends strongly on the natural order of the nodes, which is not a desirable feature (see Comparative Results).

Recognizing this fact three strategies were proposed in reference [7] intended to enhance the sparsity of U^{-1} without sacrificing the sparsity of U, by simply using a second criterion to break the ties which appear during the execution of MD algorithm. Among these strategies, Algorithm 1 (A1) seemed to be the best one. When a tie appears, it numbers in the first place the node with fewer adjacent nodes which have been already eliminated. More precisely, when more than one node has a minimum degree in the reduced graph, the one with a minimum degree in the filled graph is first eliminated. As an example consider Figure 5. In the reduced graph there are 2 nodes with a degree of 3. Node b is first chosen by A1 because it has 3 adjacent nodes in the set of eliminated nodes, against node a which has 4 of such nodes.

More recently [8], a new algorithm has been proposed which tries to improve the MD strategy in a similar way. This algorithm, which is called Minimum Degree, Minimum Length (MD-ML), uses the concept of the depth (or the length) of a node in the path graph, which is updated very easily. The MD-ML algorithm is also described in the discussion and closure of [7] where a preliminary and brief comparison is made with A1. Figure 6 shows the path graph for the 10-node system when the logic of A1 is used, while Figure 7 shows the same example following the MD-ML strategy. In this particular case A1 gives a shorter average path (3.2) than MD-ML does (3.7), though, in general, the MD-ML strategy performs better than A1.

Once these two algorithms have been revised, the rest of the section will be devoted to the new algorithm proposed in this paper. Consider once more the 10-node system when nodes 1, 8 and 9 remain to be eliminated and the other nodes were eliminated in the order 5, 10, 4, 7, 2, 6, 3. Figure 8 shows the filled graph under this situation. Three fill-ins have been generated. Figures 9.a, 9.b and 9.c show how the path graph would grow if nodes 1, 8 and 9 respectively were eliminated at this point (the degree of all of them is 2). In this situation the MD-ML strategy would not be useful since the length is 2 in all three cases. The squared nodes are the set of already-eliminated nodes which are adjacent to the node in hand, i.e., the cardinality of this set is used as a tie-break criterion by A1. Hence, node 9 should be chosen firstly if this strategy were adopted. Notice that this set of nodes represents the non-zero rows in the corresponding column of U (see, for instance, the last three columns in Figure 6). However, the nodes inside the closed

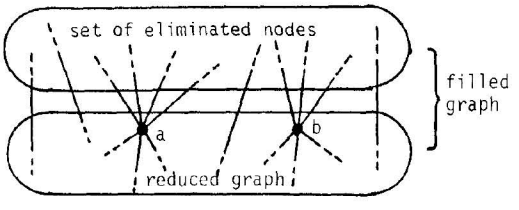

Fig. 5. Sample Illustrating the Logic of A1.

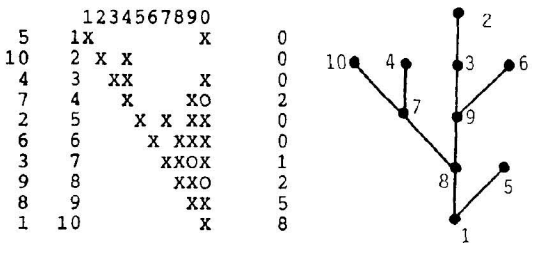

Fig. 6. Structure of U and Path Graph Following A1.

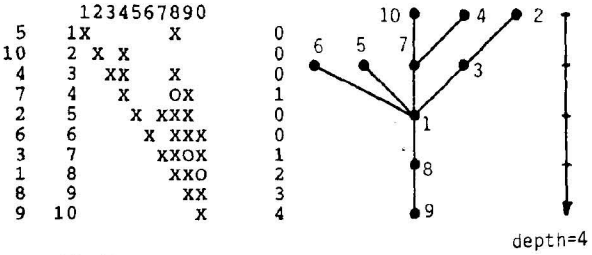

Fig. 7. Structure of U and Path Graph Following MD-ML.

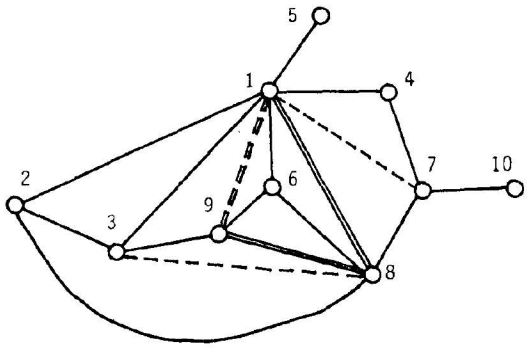

Fig. 8. Filled Graph Before Nodes 1, 8 and 9 Have Been Eliminated.

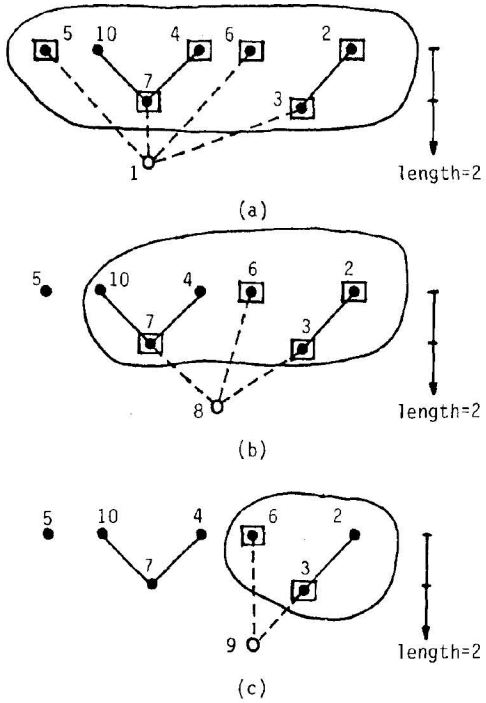

Fig. 9. Growth of the Path Graph under Three Different Hypothesis.

curve represent the non-zero rows in the same column of U^{-1} (remember Figure 4). Obviously, the squared nodes are always included within the closed curve, as the structure of **U** is a subset of the structure of U^{-1}. What we are proposing is to use the cardinality of this new set, i.e., the set of nodes which precede the node under consideration in the path graph, as a tie-break criterion. This is equivalent to locally minimize the number of non-zero elements in U^{-1} and, hence, better results are expected. The only drawback of this method is that this set is more difficult to compute or update than the parameters used by competitive algorithms.

Following this new strategy, node 9 is also the first to be eliminated. The resulting path graph is shown in Figure 10 where the preceding sets for nodes 1 and 8 are also outlined. It is clear that the next node to be chosen is node 8, leaving node 1 at the bottom (see the final result in Figure 6).

A careful analysis of the changes produced from Figure 9 to Figure 10 will provide us with a method to update the preceding set each time a node is eliminated. Let us define the following vectors:

P(i) : Number of nodes in the path graph which precede node i (plus one to include i itself). Initially set to one.
F(i) : Boolean variable. It is 'true' if node i is the last node added to any connected component of the path graph, i.e., if node i is a bordering (or frontier) node. Initially set to 'false'.

The importance of bordering nodes is that it is only required to know P(k) for such nodes. For instance, in Figure 9 nodes 3, 5, 6 and 7 are bordering nodes and P(3)=2, P(5)=1, P(6)=1, P(7)=3. It is immediate to see that:

P(1)=P(3)+P(5)+P(6)+P(7)+1=8
P(8)=P(3)+P(6)+P(7)+1=7
P(9)=P(3)+P(6)+1=4

Let us now return to Figure 10 where node 9 is a new bordering node and nodes 3 and 6 have left this set of nodes. Observe that the new values of P(1) and P(8) can be updated as follows:

P(1)=P(1)+P(9)-P(3)-P(6)=9
P(8)=P(8)+P(9)-P(3)-P(6)=8

These considerations lead finally to the following algorithm (see Appendix for a practical implementation). For a graph with **N** nodes:

Minimum Degree, Minimum Number of Predecessors Algorithm (MD-MNP).

1) Let k=1.
2) Pick up a node i with f(i)=0 such that D(i) is minimum. Break the ties which appear in this process selecting the node with minimum P(i). Set f(i)=k, F(i)='true'.
3) If k equals **N** then stop.
4) For each j adjacent to i in the filled graph do the following:
  4.1) If f(j)=0 then set P(j)=P(j)+P(i), D(j)=D(j)-1.
  4.2) If F(j)='true' then set F(j)='false' and for each **m** adjacent to **j** such that f(**m**)=0 do P(**m**)=P(**m**)-P(j).
5) For each pair of nodes **m**, **n** adjacent to i but not adjacent to each other, such that f(**m**)=f(**n**)=0, create a new edge joining **m** and **n**, and increase D(**m**), D(**n**) in one unit.
6) Set k=k+1 and go to step 2).

Figure 11 shows the results of applying this algorithm to the 10-node system. The same average path length as in A1 is obtained though the resultant ordering is different.

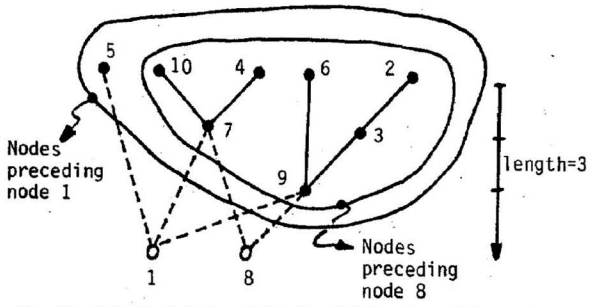

Fig. 10. Path Graph Before Nodes 1 and 8 Have Been Eliminated.

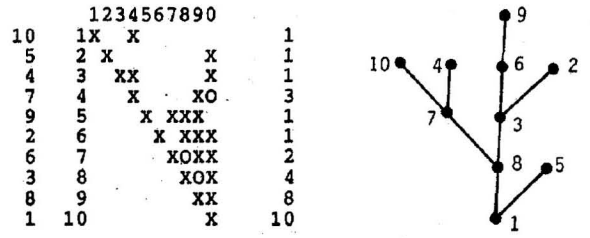

Fig. 11. Structure of U and Path Graph Following MD-MNP.

OTHER POSSIBLE VERSIONS.

In view of the proposed algorithm, some modifications may be intuitively introduced which should lead eventually to further improvements. For instance, instead of using P(i) as a merit factor, S(i) could be used, where S(i) is defined as the addition of all possible path lengths from the nodes which precede node i, like in Figure 12.a. Figure 12.b shows how node **b** should be chosen following this new criterion while node **a** is preferable from the point of view of MD-MNP strategy. This version which may be called Minimum Degree, Minimum Total Path Length (MD-MTP) differs from the main version in what follows: The ties in step 2) are broken by means of S(i) which is initially set to zero. In step 4), each time P is updated, S must be also recomputed, depending on the case, as follows:

4.1) S(j)=S(j)+(P(i)+S(i))
4.2) S(**m**)=S(**m**)-(P(j)+S(j))

That is, this version uses P only as a means of easily updating S.

By using any of the algorithms described or revised until now, a large number of ties still appear in the second criterion. In order to decrease such a large number of draws, it is possible to use the mean path, computed as the real value S(i)/P(i), in step 2).

Another way of decreasing the number of ties could be by using a third criterion. Among others, the following ones can be mentioned: MD-MNP-ML, MD-MNP-MTP

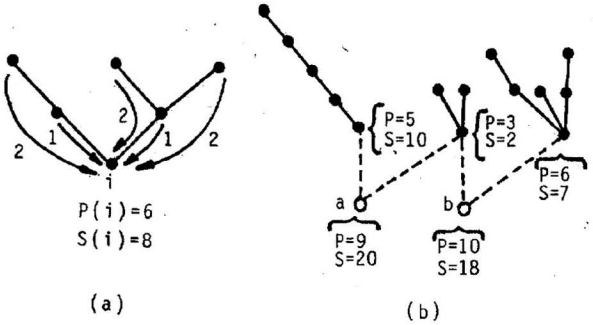

Fig. 12. Illustration of the MD-MTP Strategy.

and MD-MNP-Maximum distance appart from the centre of the graph.

Unfortunately, these more complex strategies have not yielded good enough results, when they are applied to our networks, to be further considered.

Finally, a refinement of the MD-MNP strategy stands out which is based on the following reasoning: Assume you are interested in minimizing the total mult-adds required in the FF/FB process. When node **k** is involved in a singleton path it contributes with D(**k**) mult-adds. Considering that node **k** appears in P(**k**) paths, its overall contribution is D(**k**).P(**k**) mult-adds. So it seems logical to use the product P.D as a merit factor. However, using this product as the main ordering criterion causes a large number of fill-ins in **U** because the MD strategy is perturbed.

It would be interesting, then, to find a compromise solution between this strategy and MD-MNP. In this sense, MD-MNP could be modified so that a small difference in D is less important than a big difference in P. Formally: once a minimum degree node **k** is considered as a candidate in step 2), choose a node **j** which satisfies $D(j) \leq D(k)+h$ and P(j).D(j) is as small as possible. When **h** ranges from 1 to 4 better results are obtained as may be seen in the next section. Typically **h**=3 gives the best results.

COMPARATIVE RESULTS.

Appart from the tutorial examples used in previous sections, other real systems, including the IEEE test systems and several larger Spanish networks, have been tested [9].

Tables I, II and III summarize some of the results obtained with MD, MD-ML and MD-MNP algorithms respectively for four selected networks. Each table contains the following items from left to right: Size of the system (nodes), number of non-zero off-diagonal elements of **U** and $\mathbf{U}^{-1}$, mult-adds required in the full factorization process, ordering execution time and, finally, the average and standard deviation, for every possible singleton, of the path length, mult-adds in the FF/FB and mult-adds in the PMR process. Actually, the results refer to **B'** matrix and, therefore, the slack node is not included.

Figures 13 and 14 compare the savings obtained by A1, MD-ML and MD-MNP with respect to MD when they are applied to these systems. The figures show clearly that the method proposed in this paper (MD-MNP) is superior to MD-ML which is the best algorithm published up to date.

Nod	U	-1 U	Opr. Fac.	Time sec.	PATH Mean	PATH SD	OPER.FF/FB Mean	OPER.FF/FB SD	OPER. PMR Mean	OPER. PMR SD
118	253	990	425	0.15	9.46	2.6	21.11	7.2	40.09	15.0
265	549	3622	972	0.63	14.72	3.8	48.84	12.4	132.97	36.1
448	1189	10667	2850	1.96	24.86	6.9	140.68	41.3	563.70	162.4
661	1851	17638	4972	4.18	27.72	8.6	189.62	72.1	908.90	367.7

Table I. Results obtained with MD (T-2) method.

Nod	U	-1 U	Opr. Fac.	Time sec.	PATH Mean	PATH SD	OPER.FF/FB Mean	OPER.FF/FB SD	OPER. PMR Mean	OPER. PMR SD
118	251	893	419	0.18	8.63	2.1	18.04	6.2	33.37	14.2
265	549	3002	983	0.80	12.37	3.5	37.74	15.6	99.15	48.2
448	1180	9475	2755	2.30	22.20	6.7	108.82	37.1	378.53	130.0
661	1830	15746	4778	4.92	24.86	6.8	152.18	49.4	781.08	189.7

Table II. Results obtained with MD-ML method.

Nod	U	-1 U	Opr. Fac.	Time sec.	PATH Mean	PATH SD	OPER.FF/FB Mean	OPER.FF/FB SD	OPER. PMR Mean	OPER. PMR SD
118	251	805	419	0.20	7.88	2.1	15.93	6.5	29.01	14.8
265	548	2870	977	0.83	11.87	2.8	33.71	10.9	83.46	38.7
448	1179	7667	2761	2.35	18.15	4.6	83.87	23.7	289.52	86.2
661	1799	14634	4514	4.83	23.17	5.7	129.14	38.4	518.60	174.9

Table III. Results obtained with MD-MNP method.

In table IV the average savings achieved with a larger number of networks are presented. Again, it can be seen that the proposed algorithm yields better results than previous methods for the three items considered. Notice that even a slight improvement can be attained by using the refined version described earlier. Observe also that the largest savings are

Method	Mean Path	Mean Number Operations FF/FB	Mean Number Operations Part. Refact.
A1 ref. [7]	5.63	12.75	17.28
MD-ML ref. [8]	12.33	20.89	26.60
MD-MNP	19.70	32.01	39.71
Refined MD-MNP	25.66	35.56	40.88

Table IV. Savings with respect to MD (%).

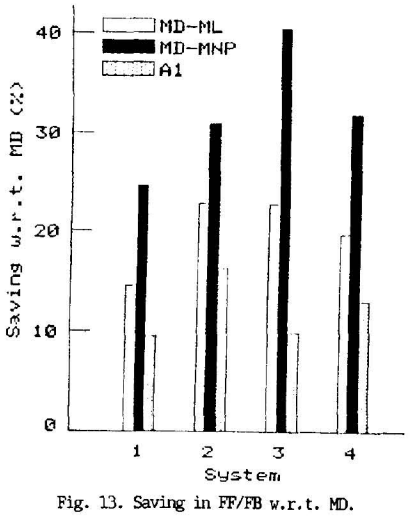

Fig. 13. Saving in FF/FB w.r.t. MD.

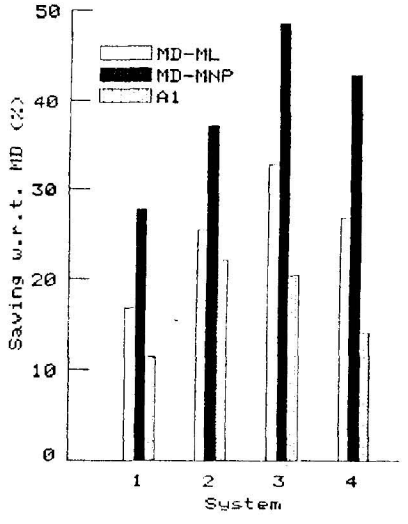

Fig. 14. Saving in PMR w.r.t. MD.

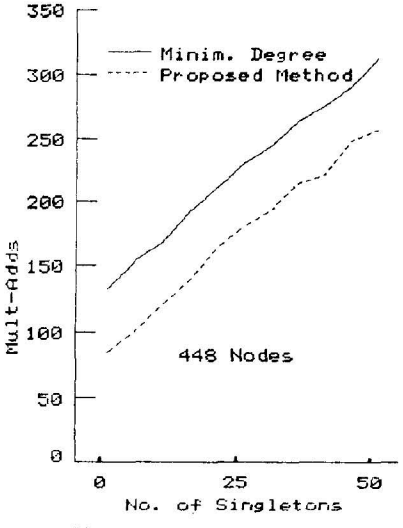

Fig. 15. Mult-Adds in FF/FB.

obtained in the PMR process (the rightmost column) as may be expected, since the non-zero elements of any row involved in the path contribute quadratically to the overall operations count of this process while only linearly to the FF/FB process.

As the number of singletons grows, the advantage of using sparse vector methods decreases. When this number tends to the size of the system, the average number of mult-adds required in the FF/FB approach the second column of the tables (i.e., the number of non-zero elements in U). This fact may be seen in Figure 15, where the proposed algorithm (MD-MNP) is compared to MD for as many singletons as approximately a 10% of the network's size (in this case the 448-node system). Analogously, as the number of modified rows in the matrix increases, the use of PMR techniques is less useful. Now, the required mult-adds tend to the fourth column of the tables (operations count in the full process). Figure 16 illustrates how the MD-MNP method behaves compared to MD for the same network. In both figures, the operations count was computed from a total of 100 trials except, logically, for one singleton.

As was commented in a previous section, the behaviour of the MD strategy, as far as the path length is concerned, has a strong dependence on the natural order of the buses. Consequently, the performance of this method may be, in certain cases, almost as good as that of the proposed algorithm and very poor in other cases. However, those methods which use a second criterion have a more regular behaviour. To prove this assertion a simple test has been done consisting in generating randomly four different node orderings for several networks. Figure 17 shows the results of this test for the 118-node system. The black bar refers to the natural order while the white ones correspond to the four random orders. It is apparent that the proposed method (MD-MNP) is quite insensitive to the way the nodes are ordered whereas, in the other extreme, MD produces notably disperse results in spite of the small size of the test case considered.

Execution times of the proposed algorithm (Table III) are only slightly larger than those of the MD algorithm (Table I) though they depend on the efficiency of the implementations (see Appendix). These times were obtained on a μVAX-II.

Finally, though the aim of this paper is not to present specific applications, it may be interesting to point out that about a 15% reduction in the execution time has been achieved when MD-MNP is applied to the parallel solution of linear equations (e.g. those which appear in the Fast Decoupled Load Flow).

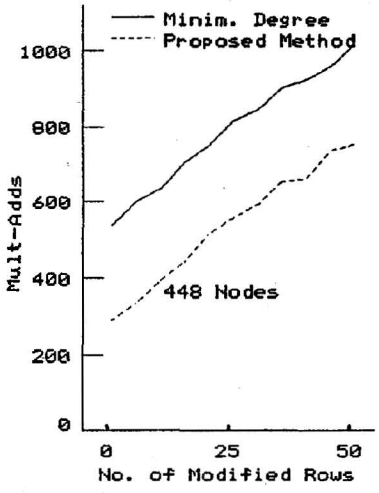

Fig. 16. Mult-Adds in PMR.

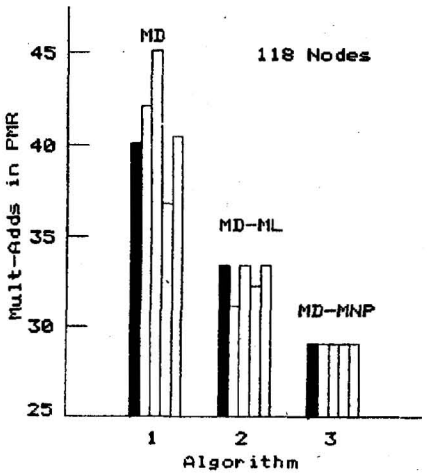

Fig. 17. Mult-Adds in PMR for Different Orderings.

CONCLUSIONS.

In this paper a node ordering algorithm to enhance sparse vector methods is presented. The method proposes a way to break the draws which appear during the choice of the pivot in the elimination process when the minimum degree strategy is adopted. A vector that contains the number of predecessors for each node in the path graph is used as a second criterion, yielding a local minimization of the number of non-zero elements in U^{-1} or, equivalently, of the mean path length. A way to compute this vector which requires a more complex logic than other simpler methods is suggested. However, the promising results obtained with the test networks (above a 35% saving in operations count w.r.t. minimum degree) justify its use.

Further work is necessary to better understand how local minimum fill-in strategies interact with local minimum path length strategies in a global sense. While minimum fill-in strategies (like MD) may eventually lead to satisfactory results from the point of view of path length, the opposite is not true. Using MNP as the main criterion gives an excessive number of fill-ins which, in turn, produce longer paths. This is why MNP is only used as a tie-break criterion. Is it possible, however, that other combinations of MD and MNP strategies lead to better results? Some data are presented in the paper which suggest that the answer is affirmative.

REFERENCES.

[1] - Tinney W.F., Walker J.W., Direct Solutions of Sparse Network Equations by Optimally Ordered Triangular Factorization. Procee. IEEE vol. 55 pp. 1801-1809, 1967.

[2] - Tinney W.F., Brandwajn V., Chan S.M., Sparse Vector Methods. IEEE Trans. on PAS-104, pp. 295-301, 1985.

[3] - Chan S.M., Brandwajn V., Partial Matrix Refactorization. IEEE Trans. on PWRS-1, pp. 193-200, 1986.

[4] - Betancourt R., Alvarado F.L., Parallel Inversion of Sparse Matrices. IEEE Trans. on PWRS-1, pp. 74-81, 1986.

[5] - Brandwajn V., Tinney W.F., Generalized Method of Fault Analysis. IEEE Trans. on PAS-104, pp. 1301-1306, 1985.

[6] - Brandwajn V., Efficient Bounding Method for Linear Contingency Analysis. IEEE/PES 1987 Winter Meeting, New Orleans. 1987.

[7] - Gómez A., Franquelo L.G., Node Ordering Algorithms for Sparse Vector Method Improvement. IEEE/PES 1987 Winter Meeting.

[8] - Betancourt R., An Efficient Heuristic Ordering Algorithm for Partial Matrix Refactorization. IEEE/PES 1987 Summer Meeting.

[9] - Gómez A., Franquelo L.G., A Node Ordering Algorithm to Speed up the Solution of Sparse Matrix and Sparse Vector Linear Equation Systems. AMSE Int. Conf. Sept. 1986.

[10] - Duff I.S., Erisman A.M., Reid J.K., Direct Methods for Sparse Matrices. Oxford University Press, N.Y., 1986, Chapt. VII.

APPENDIX: Practical implementation.

There may be several alternatives to actually implement both MD and MD-MNP algorithms, depending mainly on the way the graph's structure is stored and dealt with. An implementation is proposed here which is not necessarily the most efficient one but it is only intended to show that MD-MNP is not so complex compared to MD as it seems at first glance.

The usual linked data structure is adopted to store, for each node, the set of adjacent nodes. Vector **FIRST** points to the beginnings of these sets. **CREATE** is a routine which inserts a pair of symmetric elements into the chain and updates **D** (degree) provided the elements did not already exist. **NEXADJ** gives the currently-pointed adjacent node back and updates the pointer (only two statements are required). **MINIMUM** is a function which computes the minimum degree node according to step 2 in the main text. Following the notation introduced in the paper and assuming all vectors and the data structure are properly initialized, the MD-MNP algorithm might be as follows:

```
BEGIN
k=1
while (k<=N) do
 begin
  i=MINIMUM(f,D,P)                    $
  f(i)=k
  F(i)='true'                         *
  point1=FIRST(i)
  while (point1<>0) do
   begin
    NEXADJ(point1,n)
    if (f(n)=0) then
     begin
      D(n)=D(n)-1
      P(n)=P(n)+P(i)                  *
      point2=point1
      while (point2<>0) do
       begin
        NEXADJ(point2,m)
        if (f(m)=0) CREATE(n,m,D)
       end
     end
    else                              *
     begin                            *
      if (F(n)='true') then           *
       begin                          *
        F(n)='false'                  *
        point2=FIRST(n)               *
        while (point2<>0) do          *
         begin                        *
          NEXADJ(point2,m)            *
          if (f(m)=0) P(m)=P(m)-P(n)  *
         end                          *
       end                            *
     end                              *
   end
  k=k+1
 end
END
```

Note that step 4 has been embedded into step 5 in order to avoid duplicated accesses to the structure. By omitting **P** in the line marked with '$' and all of the lines marked with '*' the MD algorithm is obtained. Since the MD algorithm is very well known, its computational complexity is not going to be discussed here [10]. Rather it will serve as a reference to compare the proposed method.

Each time a tie appears in the searching of the minimum degree node an additional comparison is required to break it. The resultant number of comparisons is unknown "a priori" but they are peculiar (inherent) to any method which exploits a second criterion.

The other major difference appears in the logic that updates vectors **F** (frontier) and **P** (predecessors). Every node becomes a bordering node when it is eliminated. Later, it leaves forever the boundary when any of its adjacent nodes is eliminated. Hence, at the end of the ordering process the condition **F(n)='true'** has been realised N-1 times, once for each node (except the last one). This means that only one extra swept of the graph's overall structure is required, compared to MD. In addition, **2b** updatings of **F** and **P** are carried out, where **b** is the number of nonzero elements in **U**.

It is clear from the above paragraph that the overhead introduced is not very significant. Furthermore, since the proposed method gives usually less fill-in, the resultant time is even better than was expected, mainly because **CREATE**, which has an involved logic, is called fewer times. Other refined versions commented in the paper maybe, however, more time-consuming. Due to space limitations they will not be analyzed here.

ACKNOWLEDGEMENTS.

The authors acknowledge support of this work by the Spanish Comisión Interministerial de Ciencia y Tecnología (PA86-0233).

Antonio Gómez Expósito was born in Andújar, Jaén, Spain, on August 26, 1957. He received the Ingeniero Industrial and the Doctor Ingeniero Industrial degrees from the Universidad de Sevilla, Spain, in 1983 and 1985, respectively. Since 1982 he has been an Assistant and Associated Professor at the Departamento de Ingeniería Eléctrica in the Escuela Superior de Ingenieros Industriales of the Universidad de Sevilla. His current research interests lie in sparse matrices and reactive power control.

Leopoldo García Franquelo was born in Málaga, Spain, on April 14, 1954. He received the Ingeniero Industrial and the Doctor Ingeniero Industrial degrees from the Universidad de Sevilla, Spain, in 1977 and 1980, respectively. Since 1978 he has been Assistant, Associated and Professor at the Departamento de Ingeniería Electrónica, de Sistemas y Automática in the Escuela Superior de Ingenieros Industriales of the Universidad de Sevilla. His current interests include mathematical modelling of circuits, computer applications and programming techniques.