Approved:

Curtis R. Cook

Curtis R. Cook, Associate Professor
Department of Computer Science
in Charge of Major

Date presented: August, 1980

An Iteration Algorithm for Graph Orientation

by

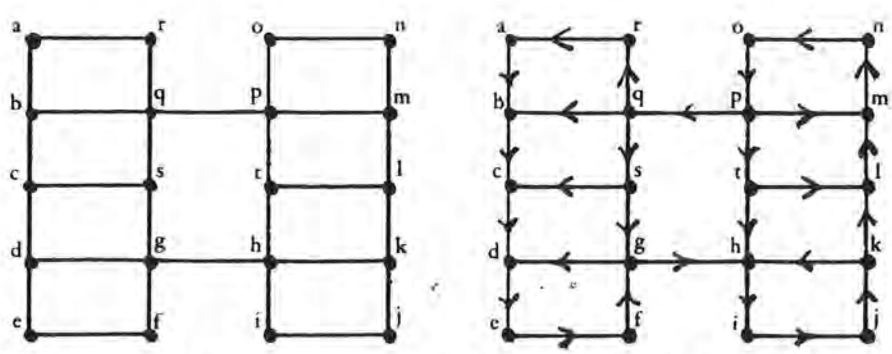
Shiang-Meei Heh

August, 1980

Table of Contents

I. Introduction

Consider the following traffic control problem: Given a network of two-way streets in a city, under what circumstances can we convert each street into a one-way street in such a way that it is possible to travel from any location to any other? Translating this problem into a graph theory problem: Can we give each edge of an undirected graph a direction (or an orientation), in such a way that in the resulting directed graph there is a directed path from each vertex u to every other vertex v? Restated, our question asks: under what circumstances does the graph have a strongly connected orientation? Robbins (9) proved that a graph has a strong orientation if and only if it is connected and has no bridge. A bridge in a connected graph is an edge whose removal results in a disconnected graph. Robbins' proof constructed a strongly connected directed graph but did not consider the efficiency of the orientation. For example, Fig. 1(a) shows an undirected, connected, bridgeless graph and Fig. 1(b) is a strongly connected, but inefficient, orientation of (a). If a person at location "a" wants to go to location "r", he must travel in a roundabout way. This assignment (Fig. 1(b)) meets the criteria, namely it gives a strongly connected orientation; but it does not give an efficient one.



(a) (b)

Fig. 1

We will follow the standard graph-theoretical notation and terminology (see Harary (6) or Deo (3)). An undirected graph $G=(V,E)$ consists of a set of objects $V=\{v_1,v_2,v_3,\ldots,v_n\}$ called vertices, and another set $E=\{e_1,e_2,e_3,\ldots,e_m\}$ whose elements are called edges, such that each edge e_k is identified with an unordered pair (v_i,v_j) of vertices. An edge is incident with both its end-vertices. A **path** p in an undirected graph G, is a sequence of vertices and edges, beginning and ending with vertices, such that each edge is incident with the vertices preceding and following it and no vertex appears more than once in a path. A directed graph (or digraph for short) G consists of a set of vertices $V=\{v_1,v_2,v_3,\ldots,v_n\}$, a set of edges $E=\{e_1,e_2,e_3,\ldots,e_m\}$, and a mapping f that maps every edge onto some ordered pair of vertices (u,v). An undirected graph G is connected if there is a path between every pair of vertices. In a digraph there are two different types of connectedness. A digraph is said to be strongly connected if there is at least one directed path from every vertex to every other vertex. A digraph is said to be weakly connected if its corresponding undirected graph is connected but G is not strongly connected. The number of incident edges directed out of vertex v is called the out-degree of v. The number of edges directed into the vertex v is called the in-degree of vertex v. For any undirected graph G, we can assign each edge of G some arbitrary direction. The resulting digraph is called an orientation of G. From now on whenever we mention the term "orientation", it means a strongly connected orientation unless otherwise specified. In an undirected (resp. directed) graph G, the distance from vertex u to vertex v, denoted by dist(u,v;G), is the number of edges in the shortest path (resp. directed path) from u to v. We also postulate dist(u,u;G)=0. The <u>eccentricity</u> of vertex v, E(v), in a graph G is the distance from v to the vertex farthest from v in G; that is, E(v)= max{dist(v,u;G): u belongs to G}. The <u>diameter</u> of a graph G is the maximal eccentricity in G (i.e., dia(G) =max{E(v): v belongs to G}); the <u>radius</u> of G is the minimal eccentricity of G (i.e., rad(G)=min{E(v): v belongs to G}). The diameter of an undirected graph G is at least the radius and at most twice the radius of G. Note that the diameter and the radius are defined only for connected undirected graphs and for strongly connected directed graphs. A <u>minimal strongly connected graph</u> is a strongly connected digraph such that if any edge is removed the remaining graph is not strongly connected. A <u>superfluous edge</u> is a directed edge between two distinct vertices such that if the edge is removed the remaining graph is still strongly connected. A certain class of problems is called NP-complete. It consists all the problems for which no polynomial-bounded algorithm has been discovered, nor has it been possible to show that polynomial-bounded algorithms do not exist for these problems.

II. Metrics

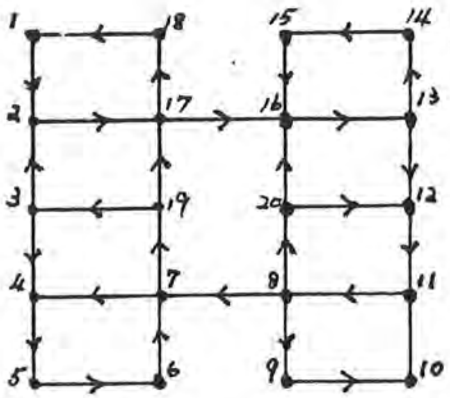
Recall that the original problem was to find an efficient conversion of the two-way streets into one-way streets so that it is still possible to travel between any two locations. In this section we will develop a reasonable and easily computed measure for comparing two orientations of an undirected graph. Three possible metrics considered were the diameter, the sum of the eccentricities of all vertices, and the sum of all distances between pairs of vertices. We chose the latter.

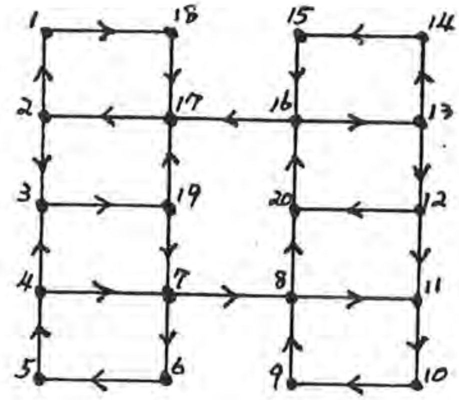
One naturally suggested measure is the diameter of the orientation. It would seem that the smaller the diameter, the better the orientation. This does not follow because the diameter only indicates the greatest distance between a pair of vertices and not the number of pairs of vertices this distance apart. In an extreme case two orientations could have the same diameter, but in one orientation most of the vertex pairs are this distance apart and in the other orientation only one pair of vertices is this distance apart. Hence the first orientation is much worse than the second. It is also possible that for two different orientations of the same graph, one has larger diameter but smaller average distance than the other. This is shown in Fig. 2. Thus we must conclude that the diameter is not a good measure.

Another candidate for the measure is the sum of the eccentricities of all vertices. The eccentricity of a vertex v is the maximum distance from v to any other vertex. This measure is like the diameter except it measures the longest distance for each vertex instead of just one vertex. Fig. 2 shows two orientations, one with smaller sum of eccentricities but larger total sum of distances of all vertices than the other. This measure only gives an estimated value of the worst case for each vertex and is not a good measure.

The last suggested measure is the sum of all distances between all pairs of vertices. This measure is easily computed from the distance matrix and if we divided the total sum of all distances of all vertices by (n^2), n is the number of vertices in the graph, we will obtain the average distance between each pair of vertices. Since an efficient orientation should have a short as possible distance between any pair of vertices, one with a smallest average distance would seem to be the most efficient. Hence the smaller the sum of all distances, the smaller the average.

From a computation point of view, the diameter, the sum of eccentricities, and the sum of all distances can immediately be determined from the distance matrix. The (i,j) entry of the distance matrix gives the distance from vertex i to the vertex j. The distance matrix can be computed in $O(n^3)$ time where n is the number of vertices (Floyd (5)). Thus the sum of all distances is a reasonable and easily computed measure of the efficiency of an orientation.





(a)

diameter = 12

sum of all distances=1972

E(1)=11	E(11)=6
E(2)=10	E(12)=7
E(3)=9	E(13)=8
E(4)=12	E(14)=11
E(5)=11	E(15)=10
E(6)=10	E(16)=9
E(7)=9	E(17)=9
E(8)=5	E(18)=11
E(9)=8	E(19)=8
E(10)=7	E(20)=8

sum of eccentricities=179

(b)

diameter = 13

sum of all distances=1936

E(1)=12	E(11)=13
E(2)=9	E(12)=10
E(3)=8	E(13)=11
E(4)=7	E(14)=10
E(5)=8	E(15)=9
E(6)=9	E(16)=8
E(7)=7	E(17)=10
E(8)=10	E(18)=11
E(9)=11	E(19)=7
E(10)=12	E(20)=7

sum of eccentricities=189

Fig. 2

III. Initial orientation algorithm and heuristic

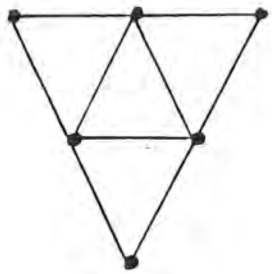
Our orientation algorithm is very similar to many iterative numerical methods such as the Newton-Raphson method. These algorithms work as follows: first an initial guess of a solution is made. Then a loop where the process of obtaining a new and improved guess from the previous guess is repeated until the improvement is below a certain value. Since it has been proven that the numerical method will converge to the solution, once the loop stops we know that the guess is very close to the solution.

Our orientation algorithm also has two parts: a good initial orientation and an iterative process in which the previous orientation is modified to obtain a better orientation. One major difference between our algorithm and the iterative numerical methods is that it has been proven that the numerical method will converge to the solution and our method only "converges" to a local minimum. The reason for this is that our algorithm would have to try all possible orientations in order to state that it converged to the minimum. To avoid having to try all possibilities, a heuristic is applied to the orientation to obtain a better orientation as computed by our measure, the sum of the distances of all pairs of vertices. Unlike the numerical method in which the modification of the guess yields a better guess, we cannot guarantee that the application of the heuristic will yield a better orientation.

The problem of finding the best orientation has been shown to be NP-complete (Chvátal (2)). Our algorithm, which is a combination of a good initial orientation and the iterative application of a heuristic, will lead to an orientation that is relatively close to the best orientation in very few iterations.

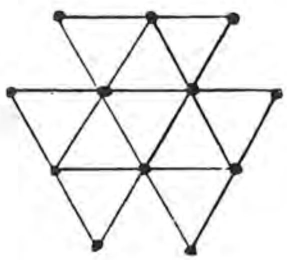
In this chapter we will develop an algorithm for finding a good initial orientation and a general heuristic for an efficient orientation. Our choice of algorithm is based on the comparison of several algorithms on the graphs in Fig. 3. Each vertex in each of these graphs was used as the starting vertex for each algorithm. Our test data, although not representing random, undirected, connected, bridgeless graphs, were selected as difficult test cases for the algorithm tested. All of the resulting orientations were compared using the sum of all distances measure developed in Chapter 2 (Table 1). Our heuristic resulted from the observation that in the best orientations the in- and out-degree were equal or nearly equal.

In general the orientation algorithms we tested were variations of the Depth-First Search that used by Roberts (10) to prove that an undirected graph has a strongly connected orientation if and only if it has no bridges. The Depth-First Search (DFS) or backtracking on a graph was first formalized and used by Hopcroft and Tarjan (7) and was subsequently studied in some depth by Tarjan (11).

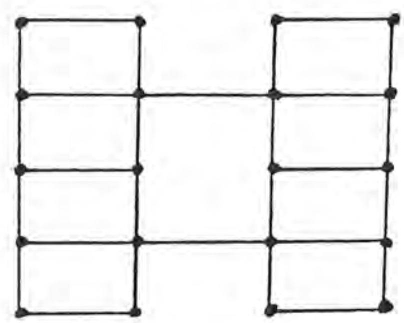


DATA 1

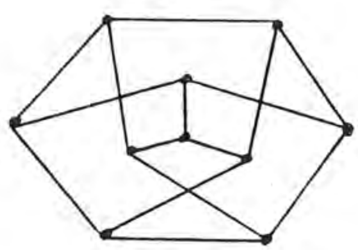
DATA 2



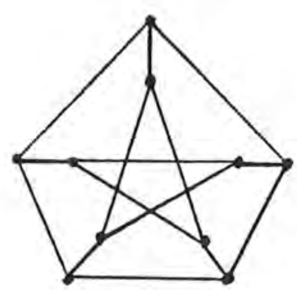
DATA 3



DATA 4

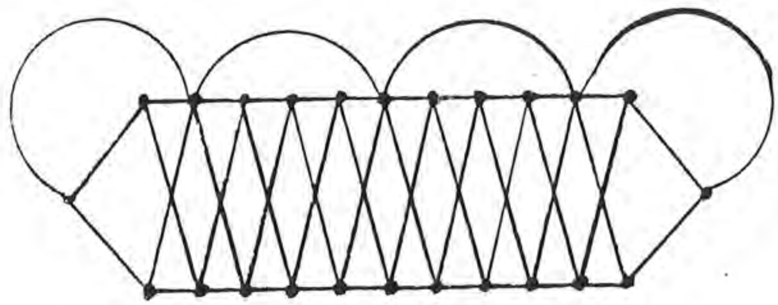


DATA 5

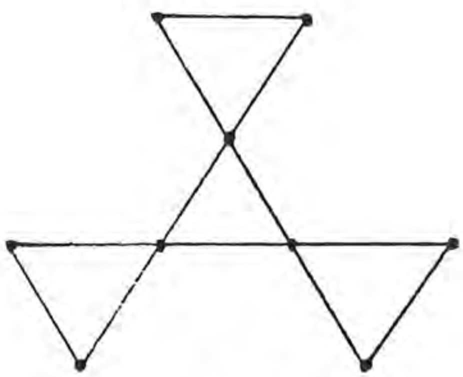


DATA 6

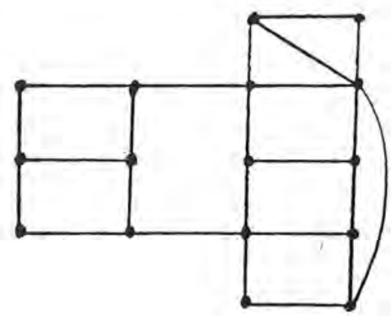
Fig. 3 test graphs



DATA 7

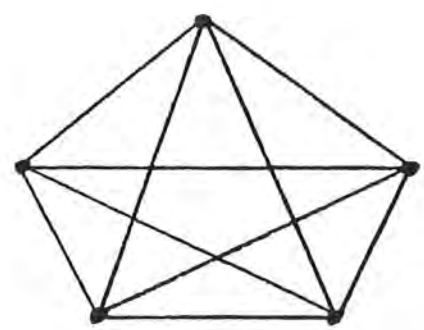


DATA 8



DATA 9

DATA 10



DATA 11

Fig. 3-continue test graphs

Description of Depth-First Search (DFS) algorithm:

Step 1. Pick any vertex as the initial starting vertex and label it 1.

Step 2. Randomly choose an adjacent, un-explored vertex, label it 2, and orient the edge from 1 to 2.

Step 3. Stand at the vertex v, labeled i, chosen by previous step, pick an adjacent, un-explored vertex v', assign next label, then direct the edge from v to v'.

If there is no other adjacent, un-explored vertex, go back to the vertex labeled i-1.

Step 4. Repeat previous step until all vertices have been traveled.

Step 5. Orient all remaining edges from the vertex with a higher label to the one with a lower label.

Note that if the undirected graph is disconnected then we will stop before we label all vertices.

Selecting the next unvisited vertex and directing the remaining edges after all vertices have been visited are the two parts of the Depth-First Search algorithm that are candidates for modification. The algorithm (Roberts (10)) chose a randon unvisited vertex during its labeling phase and directed all of the remaining edges from a higher number vertex to a lower numbered one. Two modifications of the vertex labeling phase are to choose the vertex with maximum or minimum degree instead of a random vertex; our modification of the directing all remaining edges phase is to attempt to balance the in- and out-degree of each vertex. Six algorithms that were all possible combinations of the original DFS algorithm and our modifications were compared.

Before we state our algorithms, we need some definitions.

(a) MAX(v)={u: u is a vertex of G such that u is adjacent to v and has largest vertex-degree}.

(b) BAL(v)=out-degree(v) - in-degree(v).
Initially, BAL(v) is set to zero.

(c) PASS(v) is the vertex-explored status;
PASS(v)=0 if vertex v has only in-edges or only out-edges.
PASS(v)=1 if vertex v has both in-edges and out-edges.

A. Modified Vertex-Degree Balance (MVDB) Algorithm

(* Choose un-explored vertex with maximum degree during labeling process and direct remaining edges according to vertex degree *)

Step 1. Choose a vertex v which has maximum degree as the initial starting vertex and label it 1.

Step 2. Choose v'=MAX(v), label it 2, and orient the edge from v to v'. Set PASS(v)=1
BAL(v)=BAL(v)+1
BAL(v')=BAL(v')-1.

Step 3. Stand at the vertex k which just been labeled i, check for MAX(k). If there exists one, say k', label it i=1 and direct the edge from k to k'. Set PASS(k)=1, BAL(v) =BAL(v)+1 and BAL(v')=BAL(v')-1. If there is no adjacent, un-explored vertex then go back to previous labeled vertex.

Step 4. Repeat step 3 until all vertices have been traveled.

Step 5. Check the end-vertex v (one with BAL=-1 and PASS=0)

i. if BAL(v)=-1 and PASS(v)=0 and there is an undirected edge between v and the starting vertex (one that has label 1) then direct the edge from v to the starting vertex.
BAL(v)=BAL(v)+1
PASS(v)=1

ii. if BAL(v)=-1 and PASS(v)=0 but there is no undirected edge between v and the starting vertex then we search for the vertex v' which has lowest label and adjacent to v (that is, there is an undirected edge between v and v') once we find v' then we direct the edge from v to v'.
BAL(v)=BAL(v)+1
PASS(v)=1
BAL(v')=BAL(v')-1
Repeat this step until all end-vertices have been adjusted.

Step 6. Stand at the vertex k with highest label, check to see if there is an undirected edge incident to k. If there exists one which is incident to k and k' then

i. if BAL(k) < BAL(k') then direct edge from k to k'.

BAL(k)=BAL(k)+1

BAL(k')=BAL(k')-1

ii. if BAL(k) > BAL(k') then direct edge from k' to k.

BAL(k)=BAL(k)-1

BAL(k')=BAL(k')+1

iii. if BAL(k)=BAL(k') then

if label of k is higher than the label of k' then direct the edge from k to k'.

BAL(k)=BAL(k)+1

BAL(k')=BAL(k')-1

else direct the edge from k' to k.

BAL(k)=BAL(k)-1

BAL(k')=BAL(k')+1

If there is no undirected edge incident to k then we go to the vertex with label one less than k.

Repeat this step until all remaining edges been oriented.

Remark 1. During the labeling phase, if there are more than one vertex have same MAX value then we randomly choose next node.

Remark 2. Since the original undirected graph is bridgeless, each vertex must have at least two incident edges.

B. Modified Depth-First Search (MDFS) Algorithm

(* Choose max-degree in labeling phase and orient remaining edges from higher label to lower *)

Step 1. - Step 4. Same as the step 1 - 4 in MVDB algorithm.

Step 5. Same as the step 5 in DFS algorithm.

C. Depth-First Vertex Degree Balance (DVDB) algorithm

(* Choose next vertex randomly in labeling phase and orient remaining edges according to vertex degree and vertex status *)

Step 1. - Step 4. Similar to step 1 - 4 in DFS algorithm, it also keep track of BAL- and PASS-values.

Step 5. - Step 6. Same as in MVDB algorithm.

D. Minimal Modified Vertex Degree Balance (MINMVDB) Algorithm

(* Similar to MVDB algorithm. This algorithm choose the min-degree in the labeling phase and orient remaining edges according to the vertex degree and vertex status *)

MIN(v)= u: u is a vertex of G such that u is adjacent to v and has minimal degree .

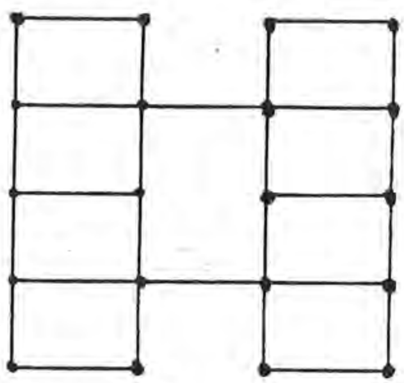
Step 1. - Step 6. Same as in MVDB algorithm.

E. Minimal Modified Depth-First Search (MINMDFS) Algorithm

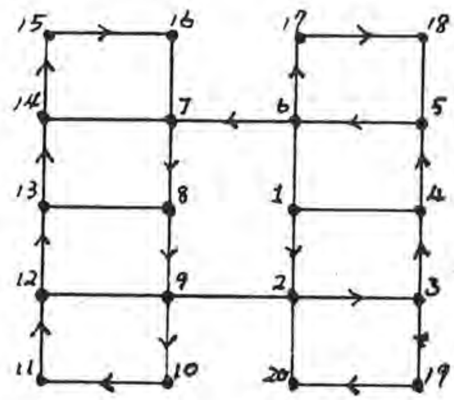
(* Choose min-degree in the labeling phase and orient remaining edges from higher label to lower *)

Step 1. - Step 6. Same as in MDFS algorithm.

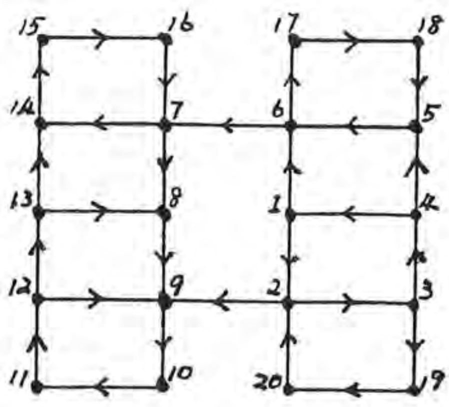
The DFS algorithm first appeared in Tarjan's "Depth-First Search and Linear Graph Algorithms" (11), and it was used in the proof in Roberts (10) that the DFS algorithm will always generate a strongly connected orientation. MDFS algorithm and MINMDFS algorithm are special cases of DFS algorithm, so they will also generate a strongly connected orientation. The DVDB and MINMVDB algorithms, however, will not always produce a strongly connected orientation. To illustrate the non-strongly connected case of DVDB algorithm, let us examine the graph in Fig. 4.



(a) undirected graph



(b) depth-first spanning tree

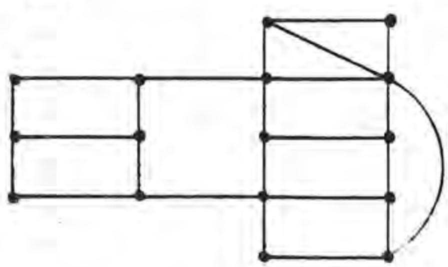


(c) non-strongly connected orientation (by DVDB algorithm)

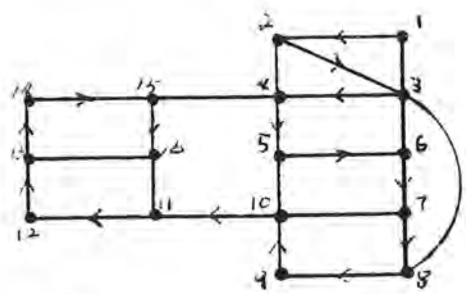
Fig. 4

Fig. 4(a) is our test graph 4, it is connected and bridgeless. Fig. 4(b) shows the depth-first spanning tree generated by DVDB algorithm, note that this spanning tree was created by choosing next adjacent vertex randomly. Fig. 4(c) shows a completed orientation generated by DVDB algorithm, note that during the stage we orient the remaining edges, the DVDB algorithm step 5 will force the edge (2,9) directed from 2 to 9, this cause the orientation not strongly connected.

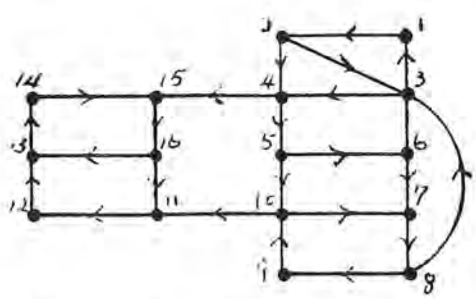
Fig. 5 shows a non-strongly connected orientation generated by MINMVDB algorithm, again, Fig. 5(a) is an undirected, connected, bridgeless graph. Fig. 5(b) is the depth-first spanning tree generated by the MINMVDB algorithm. Note that during the time we label all vertices, we choose MIN(v) to be the next point which creates a very long path and results in a non-strongly connected orientation.



(a) undirected graph



(b) spanning tree



(c) orientation (by MINMVDB algorithm)

Fig. 5

Table 1. Sum of all distances of all vertices

Data	DFS*	MVDB	MDFS	DVDB**	MINMVDB**	MINMDFS
			Algorithms			
1	65	58	65	58	63	66
2	348	348	368	337	348	360
3	385	336	385	337	363	477
4	1988	1818	2012	2240	1850	2102
5	288	290	290	288	288	288
6	288	290	290	290	277	288
7	2958	2292	2976	2267	2411	3091
8	216	216	216	216	216	216
9	1107	1025	1049	992	1044	1129
10	236	220	286	216	220	268
11	34	30	34	30	30	34

* The DFS algorithm and DVDB algorithm were run over all vertices of data as starting vertex (see appendix B), then we took the best value to compare with other Four algorithms.

** The DVDB algorithm and MINMVDB algorithm may generate a non-strongly connected orientation (see appendix B for detail list).

Table 1 shows the results from our six algorithms. Our program (Appendix A) was designed for CDC Cyber 720 and the program needs little change from one algorithm to another. From Table 1 it seems the DVDB algorithm is the best one among these six algorithms, however, the DVDB algorithm sometimes produces non-strongly connected orientation (Fig. 4), so it can not be chosen to generate our initial orientation. The DFS and DVDB algorithms were tested with every vertex as the starting point, this is because these two algorithms were required to use the random number generator and we wanted to see if we obtained different results if we used a different starting point. We also found some interesting properties in Table 1.

1. The DFS algorithm will generate an orientation which has one vertex (initial starting vertex) with all but one in-edge and some vertices (end vertex of each path) with all but one out-edge. This characteristic makes the orientation very undesirable, because it can lead to very long distances between vertices.

2. The MINMVDB algorithm sometimes will also generate a non-strongly connected orientation. This is because during the time we generate the spanning tree, we are required to select an adjacent vertex with minimum degree which, in fact, will increase the length of path of the spanning tree. Then when we finish the spanning tree and try to orient those undirected edges, the step 4 of MINMVDB algorithm will make wrong decision and cause the orientation to be not strongly connected (Fig. 5).

3. During the time when MVDB algorithm, or MDFS algorithm, or MINMDFS algorithm generates its spanning tree, if there are more than one vertex adjacent to vertex v and have same MAX(v) value (or MIN(v) value) it will choose next vertex randomly. This "random" choice will cause some different orientation. For example, the data 5 and data 6 (Peterson graph) have all vertices with same vertex-degree, so the way MVDB, MINMDFS, and MDFS algorithms generate the spanning tree will be the same as DFS and DVDB algorithms.

The DVDB and MINMVDB algorithms will not always generate strongly connected orientation, the MINMDFS algorithm seems to generate the worst orientation, the DFS and MDFS algorithms will generate results worse than the MVDB algorithm. All these seem that the MVDB algorithm is the best algorithm among all six algorithms tested. The algorithm that attempted to balance the in- and out-degree generally produce better results than those that did not. This suggests that attempt to balance the in- and out-degree is a good general rule or heuristic for orienting a graph.

IV. Iteration algorithm

In this chapter we describe an iteration algorithm that obtains a good orientation. The algorithm uses the MVDB algorithm of Chapter 3 for an initial orientation. The iteration process involves applying a heuristic to the orientation to obtain a hopefully better orientation. In Chapter 3 we observed that in most good orientations, each vertex had approximately the same in- and out-degree. Attempting to balance the in- and out-degree is the heuristic used in our iteration algorithm. First we remove a maximal set of edges that still leaves the graph minimally strongly connected; then we re-insert the edges in the maximal set in such a way as to balance the vertices.

A minimally strongly connected graph is a digraph such that if any edge is removed the resulting graph is not strongly connected. Fig. 8 shows a minimally strongly connected graph. As we can see if any edge is removed the graph is not strongly connected. If the initial orientation is a minimally strongly connected graph, we know we cannot improve it. But if the graph is not minimal, we can certainly rearrange some edges' direction and balance the vertex degree.

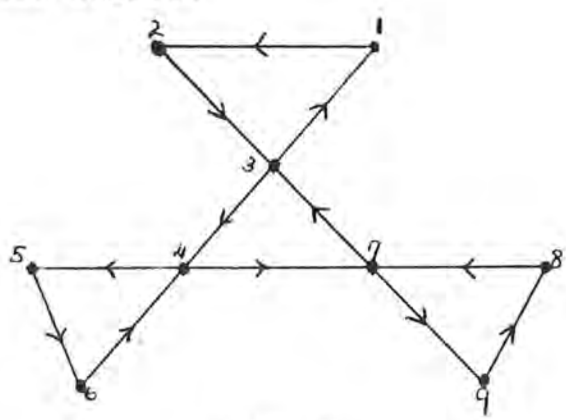


Fig. 8

Our algorithm for finding a minimally strongly connected graph is similar to one in Hsu (8). It finds a set of superfluous edges, whose removal leaves the resulting graph strongly connected.

Description of Minimally Strongly Connected (MSC) Method:

Start at the vertex v_k which was last-visited during the labeling phase.

a. Remove the out-edge which is incident with v_k and vertex v_j.
b. Calculate the distance matrix.
c. If the distance from v_k to v_j is equal to 999 (that is the value we used in our program to indicate no path between a certain pair of vertices), then we put that edge back and check next out-edge.
   If the distance between v_k and v_j is not equal to 999, we have found a superfluous edge. Record the edge, set $BAL(v_k)=BAL(v_k)-1$, $BAL(v_j)=BAL(v_j)+1$ and then check for the next out-edge.
d. If we have checked all out-edges then we go to next lower labeled vertex. Go to step a.

After we obtain the minimally strongly connected graph, we are now ready to put those removed edges back and in a manner that balances the vertex degree.

Description of Re-insertion Method:

a. Pick the edge (v_k, v_j) which was first removed during the time we generated the minimally strongly connected graph.

b. Compare $BAL(v_k)$ to $BAL(v_j)$.

   If $BAL(v_k) > BAL(v_j)$ then we put this edge back and direct it from v_j to v_k. Set $BAL(v_k)=BAL(v_k)-1$, $BAL(v_j)=BAL(v_j)+1$. Go to c.

   If $BAL(v_k) < BAL(v_j)$ then we put this edge back and direct it from v_k to v_j. Set $BAL(v_k)=BAL(v_k)+1$, $BAL(v_j)=BAL(v_j)-1$. Go to c.

   If $BAL(v_k)=BAL(v_j)$ then we compute the eccentricities of both v_j and v_k.

   If $E(v_k) >= E(v_j)$ then we put this edge back and direct it from v_k to v_j. Set $BAL(v_k)=BAL(v_k)+1$, $BAL(v_j)=BAL(v_j)-1$. Go to c.

   Otherwise, put edge back and direct it from v_j to v_k. Set $BAL(v_k)=BAL(v_k)-1$, $BAL(v_j)=BAL(v_j)+1$.

c. Pick next edge which been removed by MSC algorithm, go to b. Repeat until all removed edges have been put back.

Table 2 shows the initial orientations from MVDB algorithm and the results of five iterations of the combination of MSC and Re-insert methods. As we can see from Table 2, four out of eleven orientations were improved and the rest remained unchanged or oscillated. The iteration scheme did improve some but not all of the orientations.

Since MSC method requires $O(n^5)$ run-time, where n is the number of vertices of the graph, and from the

Table 2 Iterative results (initial orientation generated by MVDB)

Data	Initial Orientation	Iterations 1	2	3	4	5	Final Results
1	58	58	58	58	58	58	58
2	348	348	348	348	348	348	348
3	336	341	336	336	336	336	336
4	1818	1796	1818	1796	1818	1796	1796
5	290	278	268	262	262	262	262
6	290	268	262	268	262	268	262
7	2292	2239	2248	2239	2248	2239	2239
8	216	216	216	216	216	216	216
9	1025	967	967	967	967	967	967
10	220	226	220	226	220	226	220
11	30	30	30	30	30	30	30

test results of Table 2 indicate that if an orientation can not be improved it will remain unchanged or oscillated. This provides a stopping condition for our iteration algorithm. We will repeat the iteration until the orientation is worse than the previous one or until two consecutive iterations yield same value. We describe the iteration algorithm as following:

Step 1. Apply MVDB algorithm to obtain an initial orientation.

Step 2. Repeat

(a) Apply MSC method to remove all superfluous edges and to obtain the minimally strongly connected graph.

(b) Apply Re-insert method to put those removed edges back into the graph.

Until

(a) The result is worse than the previous result.

(b) The result oscillates over two consecutive iterations.

Step 3. Output the final orientation.

V. Summary

We have developed an iteration algorithm for producing an "efficient" graph orientation. This algorithm requires $O(n^5)$ run-time where n is the number of vertices of the given undirected graph.

The iteration algorithm which uses MVDB algorithm to generate an initial orientation then applies MSC and Re-insert methods repeatedly to produce the final orientation. It is very similar to iterative numerical methods. They both have an initial guess and a repeated process. However, unlike the numerical method which will converge to its solution, we cannot prove our algorithm will converge to the best orientation. Table 3 shows the results from our iteration algorithm by choosing different initial orientation. We can see that the initial orientation generated by MVDB algorithm produces better results.

For the future investigations we should consider the followings:

1. Other measures of the efficiency of an orientation.
2. Improvement of the MSC method.
3. Other heuristics for re-inserting superfluous edges.
4. Other stopping conditions.

The weighted graph which is not included in this paper may also use our iteration algorithm to generate an orientation.

Table 3 Total distances (with iteration method)

Data	Algorithms DFS	MVDB	MDFS	MINMDFS
1	64	58	58	58
2	343	348	348	348
3	343	336	358	364
4	1878	1796	1924	1850
5	262	262	262	268
6	283	262	274	262
7	2257	2239	2257	2472
8	216	216	216	216
9	966	967	995	975
10	238	220	224	220
11	30	30	30	30

References

1. Aho, A. V., J. E. Hopcroft, and J. D. Ullman. 1974 The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading, MASS.

2. Chvátal, V., and C. Thomassen. 1978 Distance in orientation of graphs. Journal of Combinatorial Theory, Series B. 24:61-75.

3. Deo, Narsingh. 1974 Graph Theory with Applications to Engineering and Computer Science. Prentice-Hall, Englewood Cliffs, N.J.

4. Edmonds, J. 1965 Paths, trees, and flowers. Canad. J. Math. 17:449-467.

5. Floyd, Robert W. 1962 Shortest path-algorithm 97. Comm. ACM 5:345.

6. Harary, F. 1969 Graph Theory. Addison-Wesley, Reading, MASS.

7. Hopcroft, J. E., and R. E. Tarjan. 1971 Planarity testing in V(log(V)) steps. Computer Science Technical Report No. 201, Stanford University, Stanford, Calif., March 1971. Also in proc. IFIP Congress, Ljubljana, Booklet Ta-2, Aug. 1971, 18-23.

8. Hsu, Harry T. 1975 An algorithm for finding a minimal equivalence graph of a digraph. ACM 22(1):11-16.

9. Robbins, H. E. 1939 A theorem on graphs with an application to a problem of traffic control. Amer. Math. Monthly 46:281-283.

10. Roberts, Fred S. 1976 Discrete Mathematical Models, with Applications to Social, Biological, and Environmental Problems. p.93-98. Prentice-Hall, Englewood Cliffs, N.J.

11. Tarjan, Robert. 1972 Depth-first search and linear graph algorithms. SIAM J. Comput. 1(2):146-160.

Appendix A

Fortran program
for graph orientation

```
      PROGRAM GRAPH(INPUT,OUTPUT)
C-----------------------------------------------------------------------
C
C     THIS PROGRAMS USE DIFFENENT ALGORITHMS TO FIND
C THE STRONGLY CONNECTED ORIENTATION OF THE GIVEN GRAPH.
C
C  THIS PROGRAM DESCRIBE MVDB(MODIFIED VERTEX-DEGREE BALANCE) ALGORITHM
C  HOWEVER, CHANGE TO DFS, DVDB, OR MDFS ALGORITHM IS VERY EASY.
C  (FOR HOW TO CHANGE TO OTHER ALGORITHM, PLEASE SEE COMMENTS INSIDE
C  THE PROGRAM.)
C
C     SUBROUTINES USED.
C          ORIENT(N,M,NUM,START,ADJACN) : THIS SUBROUTINE GENERATES THE
C                INITIAL SPANNING TREE.
C          VDBM(N,M,BAL,PASS,IN,START) : THIS SUBROUTINE DIRECTS THOSE
C                UNDIRECTED EDGES AND COMPLETE THE ORIENTATION.
C          DIR(PASS,BAL,M,I,J) : THIS SUBROUTINE DETERMINE THE EDGE
C                 <I,J> GOES FROM I TO J. (CALLED BY VDBM SUBROUTINE)
C          COMPL(M,N) : THIS SUBROUTINE DO THE LAST STEP, COMPUTE THE
C                  DIRECTED DISTANCE MATRIX AND REPORT THE TOTAL SUM,
C                  DIAMETER, AND RADIUS.
C          DFS(M,N,IN) : THIS SUBROUTINE ORIENTS THE UN-DIRECTED EDGE
C                        FROM VERTEX WITH HIGHER LABEL TO THE ONE WITH
C                        LOWER LABEL.
C          PATH(M,N) : THIS SUBROUTINE ACTUALLY COMPUTE THE DISTANCE
C                 MATRIX.
C
C
C     VARIABLES USED.
C
C          DEGREE(30) .......... THE NUMBER OF EDGES ADJACENT TO EACH
C                                VERTEX. (VERTEX DEGREE)
C          ADJACN(30,30) ....... ARRAY FOR THE ADJACENT VERTEX LABEL.
C          G(30,30) ............ THE ADJACENCY MATRIX OF THE GIVEN
C                                UN-DIRECTED GRAPH.
C          START ............... THE STARTING VERTEX.
C
C-----------------------------------------------------------------------
C
C
C
      DIMENSION DEGREE(30),ADJACN(30,30),G(30,30)
      INTEGER ADJACN,G,DEGREE,START
C
C
C. ..... PROGRAM START.   INITIALIZE VARIABLES.
C          FIRST READ IN THE NUMBER OF VERTICES THEN
C     READ IN THE VERTEX-DEGREE AND THE LABEL OF ADJACENT
C     VERTICES.
C
C
      ICOUNT=0
```

```
      DO 20 I=1,30
        DO 10 J=1,30
C
C..... SET G(I,J)=999 TO INDICATE "NO EDGE BETWEEN VERTICES I AND J".
C
          G(I,J)=999
          ADJACN(I,J)=0
   10   CONTINUE
        G(I,I)=0
   20 CONTINUE
C
C..... INPUT THE TOTAL NUMBER OF VERTICES AND ADJACENT LIST.
C
      READ *,N
      DO 50 K=1,N
        READ*,DEGREE(K)
        NN=DEGREE(K)
        READ*,(ADJACN(K,J),J=1,NN)
C
C..... SET UP ADJACENCY MATRIX.
C
C
        DO 40 LL=1,NN
          G(K,ADJACN(K,LL))=1
   40   CONTINUE
   50 CONTINUE
C
C
C..... FIND THE STARTING VERTEX.
C
C
      MAX=-999
      DO 60 I=1,N
        IF(DEGREE(I) .GT. MAX) MAX=I
   60 CONTINUE
      START=MAX
C
C
C..... IF WE WANT TO USE DFS (OR DVDB) ALGORITHM THEN USE
C      START=IFIX(RANF(X)*N+1.0)
C
C..... CALL SUBROUTINE ORIENT TO GENERATE THE SPANNING TREE.
C
C
        CALL ORIENT(N,G,DEGREE,START,ADJACN)
      STOP
      END
```

```
      SUBROUTINE ORIENT(N,M,NUM,START,ADJACN)
C
C-----------------------------------------------------------------------
C
C     THIS SUBROUTINE GENERATES THE SPANNING TREE BY CHOOSING THE
C  ADJACENT VERTEX WHO HAS MAXIMUM VERTEX-DEGREE.
C
C     SUBROUTINE USED.
C          CHOOSE(I,NUM,ADJACN,CHOICE) : CHOOSE THE NEXT VERTEX.
C          VDBM(N,M,BAL,PASS,IN,START) : COMPLETE THE ORIENTATION.
C
C
C     VARIABLES USED.
C          M(30,30) ........ THE ADJACENCY MATRIX.  INPUT VALUE.
C          N ............... THE TOTAL NUMBER OF VERTICES.
C          IN(30) .......... ARRAY FOR THE ORDER OF SPANNING TREE.
C          START ........... THE STARTING VERTEX.
C          ADJACN .......... ARRAY FOR THE ADJACENT VERTICES LABELS.
C          PREV(30) ........ ARRAY FOR BACKTRACKING THE VERTICES ORDER.
C          BAL(30) ......... ARRAY FOR IN OR OUT EDGES.
C          PASS(30) ........ ARRAY TO STORE THE STATUS OF VERTICES.
C                            PASS(I)=0 MEANS THE VERTEX I HAS ONLY IN-EDGE
C                            OR OUT-EDGE.
C                            PASS(I)=1 MEANS VERTEX I HAS BOTH IN-EDGE
C                            AND OUT-EDGE.
C          CHOICE .......... THE NEXT VERTEX LABEL.
C
C-----------------------------------------------------------------------
C
C
      DIMENSION M(30,30),IN(30),NUM(30),PREV(30),BAL(30)
      INTEGER ADJACN(30,30),PREV,BAL,START,CHOICE,PASS(30)
C
C..... INITIALIZE THE VARIABLES.
C
      ICOUNT=0
      DO 10 K=1,30
      PREV(K)=IN(K)=BAL(K)=PASS(K)=0
   10 CONTINUE
      IFIRST=1
C
C..... SET UP THE STARTING VERTEX.
C
      I=START
      IN(IFIRST)=START
C
C..... IF NUM(I)=0 MEANS NO  UN-EXPLORED VERTEX ADJACENT TO VERTEX I.
C       GO BACK TO PREVIOUS VERTEX.
C
  555 IF(NUM(I) .NE. 0) GO TO 1000
      I=PREV(I)
```

```
C..... IF WE  STANDING AT THE STARTING VERTEX AND WE ALREADY TRAVELED
C       ALL VERTICES, IT IS TIME TO STOP.
C
      IF((I .EQ. START) .AND. (ICOUNT .GE. (N-1))) GO TO 999
      GO TO 555
 1000 BAL(I)=BAL(I)+1
      PASS(I)=1
C
C..... CHOOSE NEXT ADJACENT VERTEX.
C
      CALL CHOOSE(I,NUM,ADJACN,CHOICE)
C
C..... IF DFS OR DVDB ALGORITHM THEN USE
C       CHOICE=IFIX(RANF(X)*NUM(I)+1.0)
C
      NEXT=ADJACN(I,CHOICE)
      IFIRST=IFIRST+1
C
C..... RECORD THE ORDER.
C
      IN(IFIRST)=NEXT
C
C
C..... DELETE THE VERTEX WHICH ALREADY BEEN EXPLORED.
C
      DO 1005 INDEX=1,N
        NN=NUM(INDEX)
        DO 1001 KK=1,NN
          IF(ADJACN(INDEX,KK) .NE. I) GO TO 1001
          DO 1002 KKK=KK,NN
            ADJACN(INDEX,KKK)=ADJACN(INDEX,KKK+1)
 1002     CONTINUE
          NUM(INDEX)=NUM(INDEX)-1
 1001   CONTINUE
 1005 CONTINUE
C
C
C..... ORIENT THE EDGE AND RECORD THE VERTEX-STATUS.
C
      M(NEXT,I)=999
      BAL(NEXT)=BAL(NEXT)-1
 4000 PREV(NEXT)=I
      ICOUNT=ICOUNT+1
      I=NEXT
C
C
C..... CHECK IF THERE IS ANY UN-EXPLORED VERTEX.
C
      IF(NUM(I) .NE. 0) GO TO 555
```

```
C
C..... IF NO MORE UN-EXPLORED VERTEX ADJACENT TO CURRENT VERTEX THEN
C      WE DELETE CURRENT VERTEX FROM ADJACENT-LIST AND GO BACK TO
C      PREVIOUS VERTEX.
C
  610 DO 612 IJ=1,N
        INDEX2=NUM(IJ)
        DO 613 IK=1,INDEX2
          IF(ADJACN(IJ,IK) .NE. I) GO TO 613
          DO 614 IL=IK,INDEX2
            ADJACN(IJ,IL)=ADJACN(IJ,IL+1)
  614     CONTINUE
          NUM(IJ)=NUM(IJ)-1
  613   CONTINUE
  612 CONTINUE
      I=PREV(I)
      GO TO 555
C
C
C..... CALL SUBROUTINE VDBM TO FINISH THE ORIENTATION.
C
 999  CALL VDBM(N,M,BAL,PASS,IN,START)
C
C..... IF DFS OR MDFS ALGORITHM THEN USE
C      CALL DFS(M,N,IN)
C
      RETURN
      END
```

```
---------------------------------------------------------------------

    THIS SUBROUTINECOMPARE VERTEX STATUS TO DETERMINE EDGE DIRECTION.

---------------------------------------------------------------------

    SUBROUTINE VDBM(N,M,BAL,PASS,IN,START)

..... THIS SBROUTINE ISCALLE DBY ORIENT ROUTINE.
      THE MAIN PURPOSE OF THISROUTINE IS TO DETERMINE THE DIRECTION OF
      EDGES AFTER WE OBTAIN THE SPANNING TREE.

..... N : TOTAL NUMBER OF VERTICES.
      M(30,30) : ADJACENCY MATRIX.
                 BAL(V)=(NUMBER OF OUT-EDGES OF V - NUMBER OF IN-EDGES OF V)
      PASS(30) : VERTEX STATUS,
                  PASS8V)=0  IF VERTEX HAS ONLY IN-EDGES, OR ONLY OUT-EDGES,
                             OR NOT YET EXPLORED.
                        =1  IF VERTEX HAS BOTH IN-EDGES AND OUT-EDGES.
      IN(30) : LIST OF ORDER THAT VERTICES EXPLORED.
      START : THE INITIAL STARTING VERTEX.
      IM(30,30) : TEMPORARY SCRATCH ARRAY.


      INTEGER M(30,IM(30,30),30),BAL(30),PASS(30),IN(30),START

     DO 10 K=1,N
       IF(M(K,START) .NE. 1 .OR. M(START,K) .NE. 1 .OR. PASS(K) .NE. 0
     .     .OR. BAL(K) .GE. 0) GO TO 10
       M(START,K)=999
       BAL(K)=BAL(K)+1
       BAL(START)=BAL(START)-1
       PASS(K)=1
  10 CONTINUE


*****CHECH OTHER END-NODE.


     DO 30 K=1,N
       KK=IN(N-K+1)
       IF(PASS(KK) .NE. 0) GO TO 30
       IF(BAL(KK) .GE. 0) GO TO 30
       DO 20 I=1,N
         II=IN(I)
         IF(M(KK,II) .NE. 1 .OR. M(II,KK) .NE. 1) GO TO 20
       DO 15 IJ=1,N
           IF(IN(IJ) .EQ. KK) KKI=IJ
           IF(IN(IJ) .EQ. II) III=IJ
  15       CONTINUE
         IF(KKI .LT. III) GO TO 30
```

```
          M(II,KK)=999
          BAL(KK)=BAL(KK)+1
          BAL(II)=BAL(II)-1
          PASS(KK)=1
          KK=II
          IF(KK .EQ. START) GO TO 30
   20  CONTINUE
    30 CONTINUE
       DO 70 I=1,N
         DO 60 J=I,N
          IF(M(I,J) .NE. 1 .OR. M(J,I) .NE. 1) GO TO 60
          IF(BAL(I) .GT. BAL(J)) GO TO 50
          IF(BAL(I) .EQ. BAL(J)) GO TO 40
          CALL DIR(PASS,BAL,M,I,J)
          GO TO 60
   40    DO 45 K=1,N
          IF(IN(K) .EQ. I) KI=K
          IF(IN(K) .EQ. J) KJ=K
   45  CONTINUE
         IF(KI .LE. KJ) GO TO 50
         CALL DIR(PASS,BAL,M,I,J)
         GO TO 60
   50  CALL DIR(PASS,BAL,M,J,I)
   60  CONTINUE
   70 CONTINUE

 .....CALL SUBROUTINE 'COMPL' TOCOMPUTE THE DISTANCE BETWEEN ANY PAIR
        OF VERTICES, THE DIAMETER OF ORIENTATION, THE RADIUS OF ORIENTATION,
        AND THE DISTANCE MATRIX.

      IF WE ARE USING THE ITERATIVE METHOD THEN WE SHOULD USE
      THE LOOP 140

       CALL COMPL(M,N)


   140 DO 130 I=1,5
         DO 120 J=1,N
          DO 120 K=1,N
            IM(J,K)=M(J,K)
   120   CONTINUE
         CALL MEG(IM,N,IN)
         CALL COMPL(IM,N)
   130 CONTINUE


        RETURN
        END
```

```
C
C
C------------------------------------------------------------------
C
C     SUBROUTINE DFS(M,N,IN) IS USED AFTER WE OBTAINED THE SPANNING
C  TREE.
C
C     THE MAIN PURPOSE OF THIS SUBROUTINE IS TO ORIENT THOSE UNDIRECTED
C  EDGE FROM VERTEX WITH HIGHER LABEL TO VERTEX WITH LOWER LABEL.
C
C---------------------------------------------------------------------
C
C
      SUBROUTINE DFS(M,N,IN)
C
C..... M(30,30) IS THE IMCOMPLETED ADJACENT MATRIX.
C      N IS THE NUMBER OF VERTICES.
C      IN(30) IS THE LIST OF ORDER OF ORIENTED VERTICES.
C
      DIMENSION M(30,30),IN(30)
C
C..... SET UP VARIABLES.
C
      K=NN=N
C
C..... ORIENTS THOSE UN-DIRECTED EDGES ACCORDING TO THE VERTEX LABEL.
C
      DO 3 I=1,N
        KK=IN(K)
        DO 2 J=1,NN
          IF((M(KK,J) .EQ. 1) .AND. (M(J,KK) .EQ. 1)) M(J,KK)=999
    2   CONTINUE
        K=K-1
    3 CONTINUE
C
C..... ORIENTATION COMPLETE, CALL SUBROUTINE "COMPL" TO COMPUTES THE
C      TOTAL DISTANCES, DIAMETER, RADIUS AND GENERATES THE DISTANCE MATRIX.
C
      CALL COMPL(M,N)
      RETURN
      END
```

```
C
C
C-----------------------------------------------------------------------
C
C      SUBROUTINE DIR(PASS,BAL,M,L1,L2) IS CALLED BY VDBM SUBROUTINE
C   THE MAIN PURPOSE OF THIS SUBROUTINE IS TO ORIENT THE EDGE BETWEEN
C   VERTEX L1 AND VERTEX L2 ACCORDING TO THEIR STATUS.
C
C----------------------------------------------------------------------
C
C
      SUBROUTINE DIR(PASS,BAL,M,L1,L2)
C
C..... WHEN THIS SUBROUTINE IS CALLED IT SHOULD ORIENT THE EDGE FROM L1
C       TO L2 AND CHANGE THEIR STATUS.
C
      INTEGER M(30,30),BAL(30),PASS(30)
      M(L2,L1)=999
      PASS(L1)=1
      BAL(L1)=BAL(L1)+1
      BAL(L2)=BAL(L2)-1
      RETURN
      END
```

```
C
C
C-----------------------------------------------------------------------
C
C   SUBROUTINE PATH(M,N) USED TO COMPUTE THE DISTANCE BETWEEN ANY PAIR
C  OF VERTICES.(IN OTHER WORDS, IT GENERATES THE DISTANCE MATRIX.
C
C REMARK. THIS SUBROUTINE IS BASE ON ROBERT W. FLOYD, SHORTEST PATH----
C        ALGORITHM 97. [ ]
C          THIS ALGORITHM REQUIRE o(N*N*N) RUN-TIME.
C
C-----------------------------------------------------------------------
C
C
      SUBROUTINE PATH(M,N)
C
C..... IF THERE IS NO PATH FROM VERTEX I TO J THEN WE USE "999" TO
C        INDICATE THIS CASE
C
      DIMENSION M(30,30)
      DO 20 I=1,N
        DO 30 J=1,N
          IF(M(J,I) .GE. 999) GO TO 30
          DO 40 K=1,N
            IF(M(I,K) .GE. 999) GO TO 40
            IS=M(J,I)+M(I,K)
            IF(IS .LT. M(J,K)) M(J,K)=IS
   40     CONTINUE
   30   CONTINUE
   20 CONTINUE
      RETURN
      END
```

```
C
C
C------------------------------------------------------------------------
C
C      SUBROUTINE CHOOSE(I,NUM,ADJACN,NCH) COMPARE THE VERTEX-DEGREE AND
C   CHOOSE THE ONE WITH MOST VERTEX-DEGREE AS THE NEXT POINT.
C
C------------------------------------------------------------------------
C
C
      SUBROUTINE CHOOSE(I,NUM,ADJACN,NCH)
C
C..... ADJACN(30,30) : ARRAY OF ADJACENT VERTEX LABELS.
C       NUM(30) : ARRAY OF VERTEX-DEGREE.
C       TEMP(30) : ARRAY OF TEMPORARY STORAGE AREA.
C       COUNT : COUNTER OF VERTICES WHICH HAS SAME VERTEX-DEGREE.
C
      INTEGER ADJACN(30,30),NUM(30),TEMP(30),COUNT
C
C..... INITIALIZATION.
C
      DO 100 K=1,30
  100 TEMP(K)=0
      COUNT=1
      NO=NUM(I)
C
C..... INITIALIZE THE MAXIMUM VALUE.
C
      MAX=NUM(ADJACN(I,1))
C
      NCH=1
C
C..... CHECK TO SEE HOW MANY UN-EXPLORED VERTEX ADJACENT TO VERTEX I.
C
      IF(NO .EQ. 1) RETURN
C
C..... THERE ARE MORE THAN ONE UN-EXPLORED VERTICES ADJACENT TO I, WE
C       NEED TO FIND THE ONE WITH MOST VERTEX-DEGREE.
C
      DO 3 J=2,NO
        IF(NUM(ADJACN(I,J))-MAX) 3,1,2
    1   TEMP(COUNT)=NCH
        COUNT=COUNT+1
        NCH=J
        GO TO 3
    2   MAX=NUM(ADJACN(I,J))
        COUNT=1
        NCH=J
        GO TO 3
    3 CONTINUE
      IF(COUNT .EQ. 1) RETURN
```

```
C
C..... IF MORE THAN ONE VERTICES HAS SAME MOST-DEGREE THAN WE RANDOMLY
C      CHOOSE ONE.
C
      KILL=IFIX(RANF(X)*COUNT+1.0)
C
C..... IF THE RANDOM NUMBER EQUALS TO THE ONE LAST FOUND THEN RETURN.
C
      IF(KILL .EQ. COUNT) RETURN
C
C..... ELSE FIND THE VERTEX FROM THE TEMPORARY STORAGE AREA.
C
      NCH=TEMP(KILL)
      RETURN
      END
```

```
C
C
C-----------------------------------------------------------------------
C
C       SUBROUTINE COMPL(M,N) IS USED TO COMPUTE THEDISTANCE BETWEEN ANY
C    PAIR OF VERTICES, THE DIAMETER OF ORIENTATION, THERADIUS OF
C    ORIENTATION, AND THE DISTANCE MATRIX.
C
C-----------------------------------------------------------------------
C
C
      SUBROUTINE COMPL(M,N)
C
C..... MAXD(30) : MAXIMUM VALUE OF EACH ROW IN THE DISTANCE MATRIX.
C       IND(30) : INDEX FOR OUTPUT.
C       ISUM(30) : TOTAL SUM OF EACH ROW(COLUMN).
C
C
      DIMENSION MAXD(30)
      DIMENSION M(30,30),IN(30),IND(30),ISUM(30)
C
C..... INITIALIZATION.
C
      DO 10 I=1,N
        IND(I)=I
        ISUM(I)=0
   10 CONTINUE
C
C..... TITLE.
C
      PRINT 11
   11 FORMAT(///,5X,10(1H*),' DIRECTED ADJACENT MATRIX ',10(1H*),/)
      PRINT 12,(IND(I),I=1,N)
   12 FORMAT(//,11X,30I4)
      PRINT 45
C
C
C..... OUTPUT THE DIRECTED ADJACENCY MATRIX.
C
      DO 13 I=1,N
        PRINT 50,I,(M(I,J),J=1,N)
   13 CONTINUE
C
C
C..... COMPUTETHEDISTANCE BETWEEN ANY PAIR OF VERTICES.
C
      CALL PATH(M,N)
```

```
C
C
C..... COMPUTETHE TOTAL SUMS.
C
      DO 20 I=1,N
        DO 15 J=1,N
          ISUM(I)=ISUM(I)+M(I,J)
   15   CONTINUE
   20 CONTINUE
C
C
C..... TITLE.
C
      PRINT 30
   30 FORMAT(///,10(1H*),' DISTANCE MATRIX AND SUM ',10(1H*),/)
      PRINT 40,(IND(K),K=1,N)
   40 FORMAT(//11X,30(I4))
      PRINT 45
   45 FORMAT(2X,120(1H-),/)
C
C
C..... OUTPUT THE DISTANCE MATRIX AND TOTAL SUMS.
C
      DO 60 I=1,N
        PRINT 50,I,(M(I,J),J=1,N),ISUM(I)
   50   FORMAT(2X,I4,2X,1H:,2X,30(I4))
   60 CONTINUE
C
C..... COMPUTE THE COLUMN TOTAL AND THE GRAND TOTAL(TOTAL SUM).
C
      ITOTAL=0
      DO 80 I=1,N
        IND(I)=0
        DO 75 K=1,N
          IND(I)=IND(I)+M(K,I)
   75   CONTINUE
        ITOTAL=ITOTAL+ISUM(I)
   80 CONTINUE
C
C..... PRINT COLUMN TOTALS AND GRAND TOTAL.
C
      PRINT 85,(IND(K),K=1,N),ITOTAL
   85 FORMAT(11X,30(I4))
C
C..... COMPUTE THE DIAMETER AND RADIUS.
C      FIRST GET THE MAXIMUM VALUES FOR EACH ROW.
C
      DO 101 II=1,N
        MAXD(II)=-999
        DO 102 JJ=1,N
          IF(M(II,JJ).GT.MAXD(II))MAXD(II)=M(II,JJ)
  102   CONTINUE
  101 CONTINUE
```

```
C
C..... GET DIAMETER AND RADIUS.
C
      MAD=-999
      MIR=999
      DO 103 KK=1,N
        IF(MAXD(KK).GT.MAD)MAD=MAXD(KK)
        IF(MAXD(KK).LT.MIR)MIR=MAXD(KK)
  103 CONTINUE
C
C..... OUTPUT DIAMETER AND RADIUS.
C
      PRINT 105,MAD,MIR
  105 FORMAT(/,2X,'DIA ',I5,5X,'RAD ',I5)
   88 RETURN
      END
```

```
C
C
C-----------------------------------------------------------------------
C
C     THIS SUBROUTINE FINDS THE MINIMAL STRONGLY CONNECTED DIGRAPH.
C
C-----------------------------------------------------------------------
C
C
      SUBROUTINE MEG(M,N,IN)
C
C.... M(30,30) : DIRECTED ADJACENCY MATRIX.
C     IN(30) : LIST OF ORDER OF VERTICES TRAVELED.
C     DIS(30,30) : TEMPORARY STORAGE AREA.
C     SP(30,30) : TEMPORARY STORAGE AREA.
C
      DIMENSION M(30,30),IN(30),DIS(30,30),SP(30,30)
      INTEGER DIS,SP
C
C
C.... INITIALIZE SP ARRAY.
C
      DO 20 I=1,N
        DO 10 J=1,N
          SP(I,J)=999
   10   CONTINUE
   20 CONTINUE
C
C
C.... ELIMINATES THE SUPERFLUOUS EDGES.
C       THE IDEAL OF THIS SUBROUTINE IS:  IF REMOVAL OF EDGE CAUSE
C       THE GRAPH NOT STRONGLY CONNECTED THEN WE PUT THE EDGE BACK,
C       OTHERWISE, WE FOUND A SUPERFLUOUS EDGE.  REMOVE ALL SUPERFLUOUS
C       EDGES, WE WILL OBTAIN A MINIMAL STRONGLY CONNECTED DIGRAPH.
C
C
      DO 80 ISA=1,N
        I=IN(N-ISA+1)
        DO 70 J=1,N
C
C
C.... WE START AT THE VERTEX WHICH WAS LAST VISITED.
C
C
C
C.... IF M(I,J) = 1 THAT MEANS NO EDGE GOES FROM VERTEX I TO VERTEX J
C     WE WILL NOT CONSIDER THIS CASE.
C
          IF(M(I,J) .NE. 1) GO TO 70
          SP(I,J)=1
C
C
C.... GET A COPY OF CURRENT ADJACENCY MATRIX OF THE DIGRAPH.
C
          DO 50 K=1,N
            DO 40 L=1,N
              DIS(K,L)=M(K,L)
   40       CONTINUE
   50     CONTINUE
```

```
              DIS(I,J)=999
C
C
C.... COMPUTE THE DISTANCE MATRIX.
C
              CALL PATH(DIS,N)
C
C
C.... IF DIS(I,J) NOT EQUALS TO 999 THAT MEANS WE FOUND A SUPERFLUOUS EDGE.
C
              IF(DIS(I,J) .NE. 999) GO TO 60
              SP(I,J)=999
              GO TO 70
   60         M(I,J)=999
C
C
C.... CHECK DIS(J,I) TO MAKE SURE IT IS A SUPERFLUOUS EDGE.
C
              IF(DIS(J,I) .NE. 999) GO TO 70
              SP(I,J)=9990
              M(I,J)=1
   70   CONTINUE
   80 CONTINUE
C
C
C.... AFTER WE OBTAINED THE MINIMAL STRONGLY CONNECTED DIGRAPH,
C     WE CALL SUBROUTINE EDGE TO PUT THOSE REMOVED EDGES BACK AND
C     ADJUST THE DISTANCES BETWEEN ALL PAIRS OF VERTICES.
C
C
      CALL EDGE(N,M,SP)
      RETURN
      END
```

```
C
C
C-----------------------------------------------------------------------
C
C     SUBROUTINE EDGE(N,M,MS) WILL PUT THOSE REMOVED EDGES BACK TO THE
C     MINIMAL STRONGLY CONNECTED DIGRAPH ACCORDING TO THE VERTEX STATUS
C     AND/OR THE DISTANCE BETWEEN EACH PAIR OF VERTICES.
C
C-----------------------------------------------------------------------
C
C
      SUBROUTINE EDGE(N,M,MS)
C
C.... M(30,30) : ADJACENT MATRIX OF THE MINIMAL STRONGLY CONNECTED DIGRAPH.
C     MS(30,30) : TEMPORARY SCRATCH ARRAY.
C     MM(30,30) : TEMPORARY SCRATCH ARRAY.
C     ROW(30) : VERTEX STATUS(ROW(1)=[NUMBER OF OUT-EDGES - NUMBER OF IN-EDG
C
C
      DIMENSION M(30,30),MS(30,30),MM(30,30),ROW(30)
      INTEGER ROW
C
C
C.... FIND THE VERTEX WITH MOST OUT-DEGREE.
C
      DO 20 I=1,N
        ROW(I)=0
        DO 10 J=1,N
          IF(M(I,J) .NE. 1) GO TO 5
          ROW(I)=ROW(I)+1
    5     IF(M(J,I) .NE. 1) GO TO 10
          ROW(I)=ROW(I)-1
   10   CONTINUE
   20 CONTINUE
C
C
C.... PUT THE REMOVED EDGE BACK.
C
      DO 40 I=1,N
        DO 30 J=1,N
C
C.... IF MS(I,J)=1 INDICATES WE FOUND A REMOVED EDGE.
C
          IF(MS(I,J) .NE. 1) GO TO 30
C
C.... COMPARE THE VERTEX STATUS IN ORDER TO DETERMINE THE DIRECTION.
C
          IF(ROW(I) .GT. ROW(J)) GO TO 25
          IF(ROW(I) .EQ. ROW(J)) GO TO 21
          ROW(I)=ROW(I)+1
          ROW(J)=ROW(J)-1
          M(I,J)=1
          MS(I,J)=999
          MS(J,I)=999
          M(J,I)=999
          GO TO 30
```

```
.... IF TWO VERTICES BOTH HAS SAME VERTEX-STATUS THEN WE NEED TO
     COMPUTE THE ECCENTRICITIES OF THOSE VERTICES.

  21      DO 22 MK=1,N
            DO 22 NK=1,N
              MM(MK,NK)=M(MK,NK)
  22      CONTINUE
          CALL PATH(MM,N)
            MAXI=-999
            MAXJ=-999
          DO 23 KKK=1,N
            IF(MAXI .LE. MM(I,KKK)) MAXI=MM(I,KKK)
            IF(MAXJ .LE. MM(J,KKK)) MAXJ=MM(J,KKK)
  23      CONTINUE

.... COMPARE THE ECCENTRICTIES TO DETERMINE THE EDGE DIRECTION.

          IF(MAXI .GE. MAXJ) GO TO 24
          ROW(J)=ROW(J)+1
          ROW(I)=ROW(I)-1
          M(J,I)=1
          M(I,J)=999
          MS(I,J)=999
          MS(J,I)=999
          GO TO 30
  24      ROW(I)=ROW(I)+1
          ROW(J)=ROW(J)-1
          M(I,J)=1
          M(J,I)=999
          MS(I,J)=999
          MS(J,I)=999
          GO TO 30
  25      ROW(I)=ROW(I)-1
          ROW(J)=ROW(J)+1
          M(I,J)=999
          M(J,I)=1
          MS(I,J)=999
          MS(J,I)=999
  30    CONTINUE
  40 CONTINUE
     RETURN
     END
```

Appendix B

Test results of DFS, DVDB, and MINMDFS algorithms

Appendix B. Tables of tested results of DFS and DVDB algorithms.

(1) Total sum of DFS algorithm

Starting vertex	Data 1	2	3	4	5	6	7	8	9	10	11
1	66	350	409	2070	288	288	3230	216	1107	286	34
2	66	375	443	2036	292	288	3195	216	1137	295	34
3	66	375	431	1988	290	288	3138	216	1122	268	34
4	65	375	396	2402	290	288	3153	216	1110	254	34
5	68	368	451	2012	290	288	3103	216	1183	236	34
6	65	375	439	2272	288	292	3598	216	1035	241	
7		375	431	2202	288	290	3371	216	1129	255	
8		375	399	2206	290	292	3065	216	1213	236	
9		368	395	2288	292	288	3437	216	1183	275	
10		348	402	2206	290	290	3298		1125	252	
11		348	396	2044			3280		1149		
12			385	2030			3138		1234		
13				2232			3130		1193		
14				2298			3157		1183		
15				2094			3249		1215		
16				2604			3297		1175		
17				2106			3023				
18				2428			2958				
19				2064			3260				
20				2246			3220				
21							2964				
22							3153				
23							3160				
24							3486				
Best value	65	348	385	1988	288	288	2958	216	1035	236	34

Note: These results were run over all eleven data and corresponded to every vertex as starting vertex.

(2) Diameters of DFS algorithm

Starting vertex	Data 1	2	3	4	5	6	7	8	9	10	11
1	5	7	7	13	9	9	20	6	13	7	4
2	5	8	10	14	8	9	19	6	13	8	4
3	5	8	8	12	8	9	15	6	12	9	4
4	4	8	7	18	8	9	19	6	12	7	4
5	5	7	10	12	8	9	17	6	13	6	4
6	5	8	9	16	9	8	21	6	11	6	
7		8	9	16	9	8	21	6	13	8	
8		8	7	15	8	8	15	6	14	6	
9		8	6	14	8	9	19	6	13	9	
10		7	7	14	8	8	20		12	8	
11		7	7	14			21		13		
12			5	13			19		14		
13				14			18		13		
14				16			15		14		
15				14			16		12		
16				18			17		14		
17				14			16				
18				16			15				
19				14			19				
20				15			16				
21							15				
22							15				
23							20				
24							22				
Best value	4	7	5	12	8	8	15	6	11	6	4

Note: These results were run over all eleven data and corresponded to every vertex as starting vertex.

(3) Total sum of DVDB algorithm

Starting vertex	Data 1	2	3	4	5	6	7	8	9	10	11
1	58	337	369	*	290	290	2422	216	*	220	30
2	58	367	365	1798	292	292	2290	216	997	220	30
3	58	348	*	1908	290	292	2348	216	992	216	30
4	58	348	343	1900	290	290	2365	216	1095	216	30
5	58	348	349	2010	290	292	2304	216	1021	226	30
6	58	368	*	*	288	290	2428	216	1023	217	
7		348	337	1848	288	290	2307	216	1061	222	
8		348	370	1800	290	292	2264	216	1041	232	
9		348	*	2006	292	292	2380	216	1111	231	
10		348	*	1884	290	290	2240		1023	216	
11		348	*	2124			2351		1065		
12			368	1830			2318		*		
13				2012			2373		*		
14				1834			2382		1108		
15				1916			2289		1047		
16				*			2337		1063		
17				*			2350				
18				1852			2307				
19				*			2316				
20				1878			2355				
21							2295				
22							2318				
23							2271				
24							2300				
Best value	58	337	337	1798	288	290	2240	216	992	216	30

* A non-strongly connected orientation.

Note: These results were run over all eleven data and corresponded to every vertex as starting vertex.

(4) Diameters of DVDB algorithm

Starting vertex	Data 1	2	3	4	5	6	7	8	9	10	11
1	3	6	6	*	8	8	14	6	*	5	2
2	3	8	6	11	8	9	12	6	10	5	2
3	3	7	*	12	8	9	12	6	9	4	2
4	3	7	5	11	8	8	14	6	12	4	2
5	3	7	5	14	8	9	12	6	11	5	2
6	3	8	*	*	9	8	10	6	11	4	
7		7	4	11	9	8	10	6	11	5	
8		7	6	11	8	9	10	6	11	6	
9		7	*	13	8	9	11	6	13	6	
10		7	*	12	8	8	12		10		
11		7	*	14			12		11		
12			5	11			12		*		
13				12			12		*		
14				11			10		11		
15				14			10		10		
16				*			13		11		
17				*			12				
18				12			12				
19				*			12				
20				11			13				
21							11				
22							11				
23							12				
24							13				
Best value	3	6	4	11	8	8	10	6	9	4	2

* A non-strongly connected orientation.

Note: These results were run over all eleven data and corresponded to every vertex as starting vertex.

(5) Total sum of MINMDFS algorithm

Starting vertex	Data 1	2	3	4	5	6	7	8	9	10	11
1	68	375	477	2202	288	288	3233	216	1215	331	34
2	68	368	485	2324	288	288	3197	216	1183	331	34
3	66	368	485	2102	288	288	3224	216	1183	288	34
4	68	361	485	2324	288	288	3092	216	1203	274	34
5	68	368	485	2200	288	288	3124	216	1201	268	34
6	66	368	485	2266	288	288	3268	216	1201	270	
7		367	477	2324	288	288	3204	216	1129	277	
8		375	485	2322	288	288	3594	216	1207	277	
9		361	485	2322	288	288	3124	216	1231	290	
10		360	485	2202	288	288	3144		1217	310	
11		382	477	2324			3224		1224		
12			485	2096			3197		1163		
13				2324			3233		1203		
14				2200			3197		1203		
15				2256			3290		1210		
16				2322			3479		1211		
17				2322			3235				
18				2272			3348				
19				2400			3233				
20				2432			3208				
21							3091				
22							3479				
23							3271				
24							3227				
Best value	66	360	477	2102	288	288	3091	216	1129	268	34

Note: These results were run over all eleven data and corresponded to every vertex as starting vertex.

(6) Diameters of MINMDFS algorithm

Starting vertex	Data 1	2	3	4	5	6	7	8	9	10	11
1	5	8	11	16	9	9	23	6	15	9	4
2	5	8	11	19	9	9	23	6	15	9	4
3	5	8	11	16	9	9	23	6	15	9	4
4	5	7	11	19	9	9	18	6	14	9	4
5	5	7	11	16	9	9	18	6	14	9	4
6	5	7	11	19	9	9	20	6	15	9	
7		8	11	16	9	9	20	6	12	9	
8		8	11	16	9	9	23	6	15	9	
9		8	11	19	9	9	18	6	14	9	
10		8	11	16	9	9	18		15	9	
11		8	11	19			23		14		
12			11	16			23		14		
13				19			23		14		
14				16			23		14		
15				16			23		15		
16				16			23		15		
17				16			20				
18				19			19				
19				17			16				
20				18			20				
21							20				
22							23				
23							22				
24							22				
Best value	5	7	11	16	9	9	16	6	12	9	4

Note: These results were run over all eleven data and corresponded to every vertex as starting vertex.