Reprinted from JOURNAL OF COMPUTATIONAL PHYSICS
All Rights Reserved by Academic Press, New York and London

Vol. 87, No. 1, March 1990
Printed in Belgium

Note

An Algorithm to Find All Paths between Two Nodes in a Graph

The problem of finding paths connecting two nodes in a given graph is of great interest for several applications in different fields. Whenever complete information on *all* paths (e.g., total number of pahts, length, and cost) is needed, a heuristic search is useless. If the size of a given application is manageable in terms of available CPU time and/or memory limits, an exhaustive search (that is a comprehensive analysis) is still the only way to solve this problem [1].

The algorithm presented in this note has been successfully used to analyze data from Metropolis-Monte Carlo [2a] and molecular dynamics [2b] computer simulations of 125 water molecules interacting through MCY [3] or ST2 [4] potentials. However, it could be easily used virtually without modifications in any case that requires an exhaustive search of an undirected graph.

A schematic flow chart of the algorithm is presented in Fig. 1. The graph to be examined, G, the start, and end nodes are given as input data (block 1). The undirected graph G includes N nodes and it is represented by an adjacency matrix in which the ith row lists the nodes adjacent to node i. The order in which nodes are explored is unessential for our purposes.

This problem is ideally suited for a recursive algorithm. However, since we want to use languages that do not support recursion (e.g., FORTRAN, OCCAM2), we decided to implement backtracking as an alternative approach. The program consists of an exhaustive depth-first graph search for *all* solutions. A stack is used, where the program stores/retrieves the appropriate context, to go one step forward/backward in the graph, by updating a stack index. The program starts pointing to the adjacency list for the start-node (block 2), selects an adjacent node not yet explored (blocks 4–6), tests for solution (i.e., the end-node), a dead end, or a cycle (blocks 7 and 9), and stores intermediate results (block 11). This process is accomplished by pushing the working adjacency matrix in the stack and modifying by zeroing the adjacency just explored. At this point the program is ready to follow a pattern in the graph structure by successive exploration of one of the adjacent nodes, i (block 12), until a solution, a dead-end or a cycle is found. When a solution is found the node list is printed (block 8) and the program goes one step back (block 10) to repeat the cycle for all the previous adjacencies not yet explored. The back travel in the proper way along the graph is granted by retrieving the appropriate adjacency matrix from the stack. Memory requirements are of the order of $N \times K \times L$, where N is the number of nodes, K is the maximum number of

0021-9991/90 $3.00

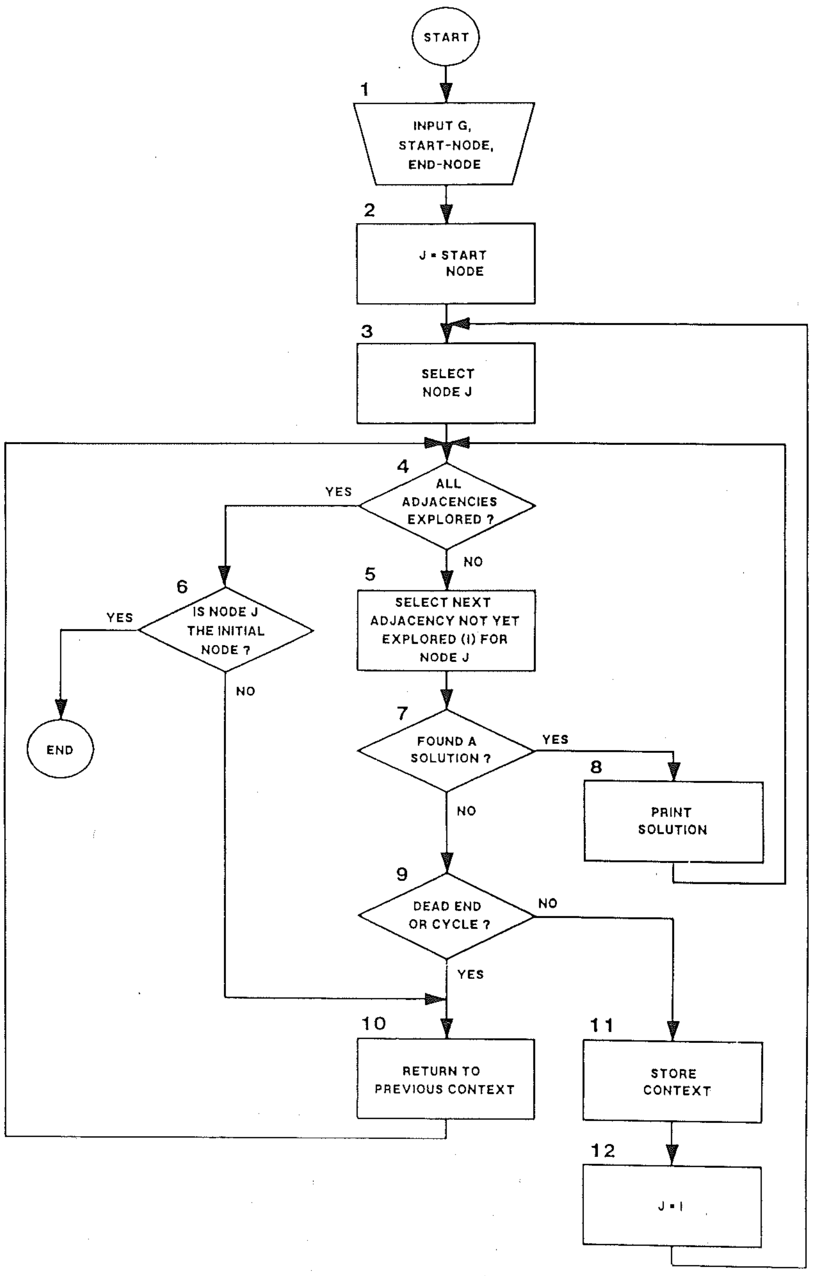
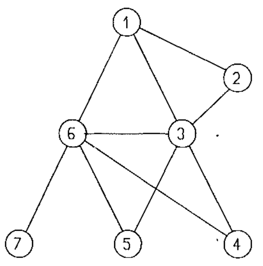

START
1
INPUT G, START-NODE, END-NODE
2
J = START NODE
3
SELECT NODE J
4
ALL ADJACENCIES EXPLORED ?
YES
NO
5
SELECT NEXT ADJACENCY NOT YET EXPLORED (I) FOR NODE J
6
IS NODE J THE INITIAL NODE ?
YES
NO
END
7
FOUND A SOLUTION ?
YES
NO
8
PRINT SOLUTION
9
DEAD END OR CYCLE ?
NO
YES
10
RETURN TO PREVIOUS CONTEXT
11
STORE CONTEXT
12
J = I

FIG. 1. Schematic flowchart of the algorithm.



NODE	ADJACENCIES
1	6 2 3
2	1 3
3	2 1 6 4 5
4	3 6
5	6 3
6	1 3 5 7 4
7	6

FIG. 2. Example of a simple graph and the corresponding adjacency matrix.

adjacencies for each node, and L is the maximum depth to be reached in the exploration of G.

As an illustrative example, let us consider the simple graph shown in Fig. 2 with its adjacency matrix, and suppose we want to find all the pathways connecting node 1 with node 5. In Fig. 3 are schematically shown the steps followed by the program. As one can see, all branchings of a given node are systematically explored each time selecting the deepest node in the graph. Cycles (e.g., 1–6–3–2–1... in Fig. 3) are avoided since the program maintains an updated list of the nodes already visited along the path. Indeed, all the secondary paths to a node already in the list generated during the search process, are identified and ignored. The program exits when the last adjacency of the last adjacent node to the start-node is solved (node 5 from node 3 in the example of Fig. 2).

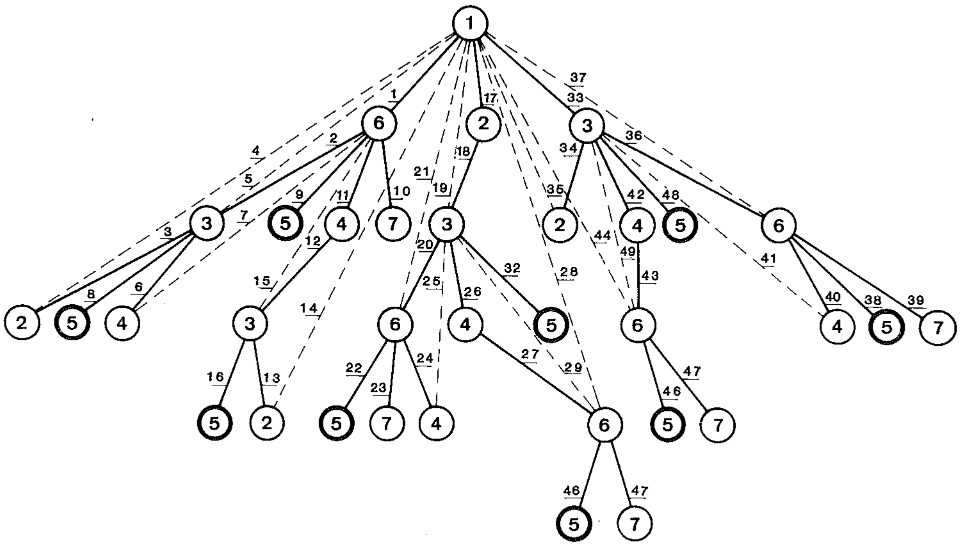


FIG. 3. Diagram of the search process followed by the program. Dashed lines represent adjacencies that generate cycles.

TABLE I

Characteristic Parameters of the Graph Generated from a Monte Carlo Configuration

Number of nodes	125 (water mol.)
Average H-bond/water	2.1
Start node	8
End node	22
Number of paths found	2774
Number of steps	306920
VAX 11/750 with FPU	3360 s
CRAY X-MP/48	6.4 s
1 20 MHz-T800 (FORTRAN)	2040 s
1 20 MHz-T800 (OCCAM2)	95 s

Note. Execution times are of the program running on VAX 11/750 with floating point accelerator, CRAY X-MP/48 and 1-T800 Transputer (FORTRAN and OCCAM2).

In our computer simulations on aqueous systems [5] we are interested in characterizing the *H*-bond pathways between two distant water molecules, in terms of their length, number, and topology. In this work we use the present algorithm to analyze the pathways between two, arbitrarily chosen, water molecules belonging to a Monte Carlo configuration. The adjacency matrix was constructed assuming that two water molecules are "bonded" if their interaction energy is less than −12kJ. At this energy threshold, chosen only for demonstration purposes, the system is well above the percolation limit, therefore most likely a connectivity pathway exists between any given couple of water molecule, and the underlying H-bond network is intricate enough to present virtually all kinds of possible branchings.

In Table I we report CPU time values for a VAX 11/750, CRAY X-MP/48, a Transputer-based system [6], and data referring to the specific snapshot selected

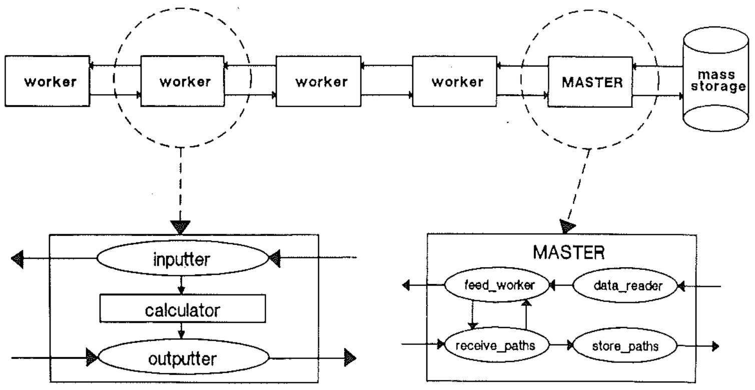


FIG. 4. Schematic representation of the 5-T800 Transputers system.

from our simulation. In this case we have $N = 125$ (the number of water molecules in the simulated sample), $K = 5$ (maximum number of H-bonds per water), $L = 100$ (we are interested in pathways up to 100 molecules long).

As one can see from Table I, the program on CRAY runs over 500 times faster than on VAX without modifications or restructuring, since it is automatically *vectorized* directly by the CRAY FORTRAN compiler. The same FORTRAN program runs on a system which includes one 20MHz-T800 Transputer [7], in almost half of the VAX time. When implemented in OCCAM2 [8], that is the native high-level language for Transputers, it runs in 95 s, that is only 15 times slower than on CRAY (which is more than 1000 times more expensive). The big difference in performance between the FORTRAN and OCCAM2 implementations for Transputers is essentially due to the smart matrix assignment instructions of OCCAM2.

The analysis of a single configuration cannot be efficiently parallelized. In fact, this kind of graph search cannot be efficiently implemented in parallel processes, since information on all nodes and cycles already explored have to be shared in real time among all concurrent processes, and the information exchange through the communication channels would degrade severely the overall system performance. Another way to take advantage of parallelism, is the so-called *processor farm* method [9], where each processor analyzes a different configuration, running the same program on independent sets of data. Thus no communications between processors occur other than those needed to send the pathways list to the master processor. In the present case we used a 5-T800 Transputer array configured as schematically shown in Fig. 4. The overall speed of the farm is essentially independent on the particular network topology, and it grows linearly with the number of processors.

Acknowledgments

We wish to thank Professor S. L. Fornili for many useful discussions, Mr. G. La Gattuta, Mr. M. Lapis, and Mr. S. Pappalardo for technical assistance. The present work has been carried out at IAIF-CNR and supported also by local MPI 60%, CRRNSM, and Prog. Fin. Sist. Inform. e Calc. Parallelo funding.

References

1. For a general treatment of graphs search strategies see: (a) E Rich, *Artificial Intelligence* (McGraw-Hill, New York, 1983), p. 55; (b) N. J. Nilsson, *Principles of Artificial Intelligence* (Tioga, Palo Alto, CA, 1980), p. 53.
2. (a) J. P. Valleau and S. G. Wittington, "A Guide to Monte Carlo for Statistical Mechanics. 1. Highways," *Modern Theoretical Chemistry*, Vol. 5, edited by B. J. Berne (Plenum, New York, 1977), p. 137; (b) J. Kushick and B. J. Berne, "Molecular Dynamics Methods: Continuous Potentials," *Modern Theoretical Chemistry*, Vol. 6, edited by B. J. Berne (Plenum, New York, 1977), p. 41.
3. O. Matsuoka, E. Clementi, and M. Yoshimine, *J. Chem. Phys.* **64**, 1351 (1976).
4. F. H. Stillinger and A. Rahman, *J. Chem. Phys.* **60**, 1545 (1974).

5. R. Noto, M. Migliore, F. Sciortino, and S. L. Fornili, *Mol. Simul.* **1**, 225 (1988); F. Sciortino and S. L. Fornili, *J. Chem. Phys.* **90**, 2786 (1989); F. Bruge', V. Martorana, and S. L. Fornili, *Mol. Simul.* **1**, 309 (1988).
6. INMOS, *Transputer Development System* (Prentice-Hall, London, 1988), p. 129.
7. INMOS, *Technical Note No. 6* (INMOS, Bristol, 1986).
8. D. Pountain, *A Tutorial Introduction to OCCAM Programming* (INMOS, Bristol, 1987), p. 35.
9. C. R. Jesshope, *Comput. Phys. Commun.* **41**, 363 (1986).

Received: December 16, 1988; revised: March 8, 1989

M. Migliore
V. Martorana

C.N.R.-Institute for Interdisciplinary Applications of Physics
via Archirafi 36, I-90123 Palermo, Italy

F. Sciortino

Department of Physics, University of Palermo
via Archirafi, 36, I-90123 Palermo, Italy

Printed by Catherine Press, Ltd., Tempelhof 41, B-8000 Brugge, Belgium