Testing the Connectivity of Networks

Taha Sochi*

2011

*University College London, Department of Physics & Astronomy, Gower Street, London, WC1E 6BT. Email: t.sochi@ucl.ac.uk.

Abstract

In this article we discuss general strategies and computer algorithms to test the connectivity of unstructured networks which consist of a number of segments connected through randomly distributed nodes.

Keywords: connectivity; unstructured network; topology; computer algorithm; node mapping; segment mapping.

1 Introduction

The network, as an abstract concept, consists of segments joined at randomly distributed nodes, as presented schematically in Figure 1. The segments can represent routes, ducts, pipes, paths, streets, wires, elements in the finite difference or finite element methods, etc., while the nodes can represent pores, junctions, crossroads, pylons, airports, terminals, and so on. In many scientific and engineering applications, networks are used as model input data to describe the topology, and might even the geometry, of certain physical or theoretical objects which are under investigation. Examples of network include real and model porous media [1–3], vascular networks of blood vessels [4], electronic circuits [5], fluid transport pipe systems [6], city traffic routes [7], computing and communication networks [8], the World Wide Web [9], electric power grids [10], railway nets [11], decay routes of excited quantum systems such as atomic and molecular species [12], and so on.

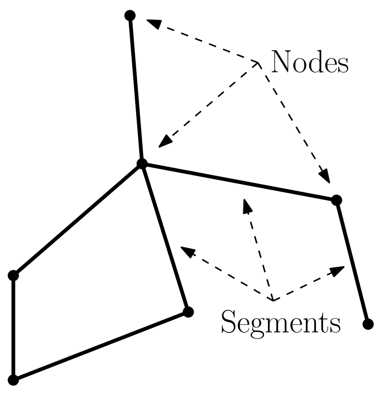

Figure 1: A simple network.

One condition that is commonly required in these networks is total connectivity, that is the network must be a single joined entity with no cut or separation. Strictly speaking, the total connectivity of a network means that any junction in the network can be reached from any other junction following the routes inside the network without jump. The network is partially connected if it consists of a number of totally connected sections.

As most of these networks, especially the large ones, are produced through automated computational processes, the connectivity of the network cannot be guaranteed. In fact even the manually built networks can suffer from disconnectivities due to human errors. Because direct tests by human inspection (e.g. by tracking the routes and building connectivity tables) is not practical except for the very small networks, automated computer algorithms based on certain strategies are required to test the connectivity of these networks.

In this article, unless stated otherwise, we deal with general networks without specific condition on their structure. In this context, some terminology may provide helpful clarification. The connectivity index, c, of a node is the number of segments connected to that node. The connectivity index, in general, can take the values $c = 0$ for a singular non-connected node, $c = 1$ for a boundary node which is connected to a single segment, $c = 2$ for a bridge node connecting only two segments, and $c = n > 2$ for branching nodes. The network is unstructured if it has variable node connectivity index with random node distribution and indexing. The connectivity index may also be attributed to the network as a whole, in which case it means the average node connectivity index and is usually computed as twice the total number of segments divided by the total number of connected nodes which excludes the singular ones.

The totally-connected network (as well as each section of a partially connected network) can be open where each node can be reached from any other node through a unique non-retraced path, or closed where each node can be reached from any other node through a number of distinct paths, or semi-closed where some nodes can be reached through unique paths while others can be reached through multiple paths. A schematic representation of these three types of network are presented in Figure 2. The assumption here is that each segment in the network has exactly two nodes which define its ends. For the open networks, these two nodes uniquely identify the segment and distinguish it from all other segments. In the following we assume that there is no singular nodes in the network as they serve no purpose in the context of connectivity and hence can be removed from the network.

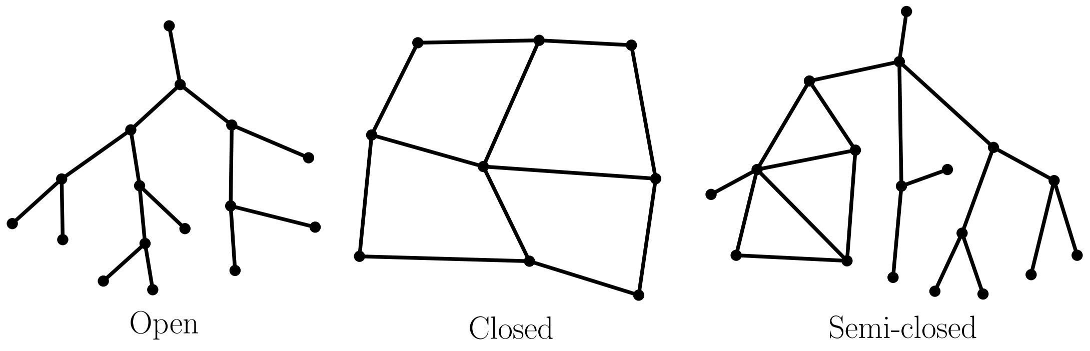

Figure 2: Schematic representation of network types.

2 Methods

There are many methods for testing the connectivity of networks. In the following we outline some of these methods with proposed algorithms and graphic illustrations. As we do not assume a specific topology for the networks, the algorithms and conclusions are general.

2.1 Direct Inspection

Direct connectivity inspection method is the most straight test to verify network connectivity since it is derived from the definition of total connectivity, that is in a totally connected network each node is connected to all other nodes in the network and hence any node can be reached from any other node within the network routes. The requirement, therefore, of this simple method is to find out through a systematic route inspection if all other nodes can be reached from each node in the network. This can be achieved, for example, by constructing a two-dimensional square array where both dimensions represent the list of all nodes in the network (e.g rows for origin nodes and columns for destination nodes) and the entries in the array can represent Boolean flags for finding a connecting route between the corresponding nodes representing the row and column of the particular entry. The network is then totally connected if all the entries, after a systematic route inspection, are found to be true.

However, because connectivity is a reflexive relation (i.e. each node is connected to itself) the diagonal entries are redundant. Moreover, because connectivity is a symmetric relation (i.e. if node A is connected to node B then node B is connected to node A) the connectivity can be tested once for each pair of nodes and hence only half of the possible route inspections are required. Consequently, what is needed is only to verify that node n is connected to all nodes m $(m > n)$ where $n = 1, 2, \dots, (N - 1)$ with N being the total number of nodes. Hence, the square array reduces to a lower or upper triangular array.

In fact even this is a stronger condition than is needed because a necessary and sufficient condition for total connectivity is that a single randomly-selected node (let us call it X) is connected to all other nodes. The reason is that connectivity is a transitive relation (i.e. if A is connected to B and A is connected to C then B is connected to C) and hence under this reduced connectivity condition any two nodes in the network are connected to each other at least through node X. The triangular array then reduces to a one-dimensional array whose entries represent connectivity status of node X with respect to all other nodes in the network (e.g. one row representing node X versus $N - 1$ columns representing all other nodes).

However, for an unstructured network even this reduced route inspection is not an easy task in its simple and direct realization, and hence other methods which are implicitly based on this method are easier in implementation and more efficient in execution, as we will see for example in the following node mapping method.

2.2 Node Mapping

The idea of node mapping, whose essence is the reduced route inspection as outlined in the last section, is that instead of testing the connectivity between two pre-selected nodes, which is very difficult and demanding since there are many possibilities for the routes and junctions between these two particular nodes, the search is focused on finding all the connections between random nodes with a start from a single randomly-selected node. By compiling these random connections into a list of connected nodes, a connected section, which could incorporate the entire network, can be constructed. The starting point in this method is to build a node-to-neighbors mapping (i.e. a nodes list that maps each node to its immediate neighbor nodes) and provide it as an input. This can take the form of a structured indexed array whose first entry is a node index while its second entry is a vector of the indices of all neighbor nodes. An implicit node index can be used for the sake of efficiency and reduced memory storage and hence the first entry is redundant. It is very easy and efficient to assemble such a mapping from the network segments list where the index of each node of a segment is added to the list of connected nodes in the entry of the other node.

Node mapping starts from a randomly-selected node by adding this node and all its neighbors to a connected section list with the removal of the node itself from the list of network nodes. The process then goes on recursively by adding the neighbor nodes of each one of the previously found neighbors with the removal of these neighbor nodes from the network nodes list as soon as their immediate neighbors are added to the neighbor list and hence to the current connected section. The process ends either with the complete consumption of the nodes list if the network is totally connected, or with the failure of finding any new connected node in one iteration which marks the identification of a complete connected section. In the latter case, this operation can be repeated, to find the other sections, until the network nodes list is exhausted.

The use of two 1D Boolean arrays, one to mark the removal of the nodes from the network list and the other to mark the affiliation of the nodes to the current section, can make this process highly efficient in terms of memory, especially if implicit node indices (represented by the ordinal indices of the array entries) are used for accessing the entries of the Boolean arrays, with an added advantage of ease of implementation. The direct access of the Boolean entries of the nodes by the prompt use of their indices will speed up the whole operation and make it highly efficient in the CPU time as well. A flow chart of a possible algorithmic implementation based on the node mapping method with the use of two 1D Boolean arrays is presented in Figure 3. Other computational techniques, such as the actual removal of the processed nodes from the nodes list directly, can also be used for implementing this method. However, some of these techniques may incur a high computational cost in terms of CPU time and memory consumptions and could complicate the implementation.

The use of direct access through the use of node indices may be complicated by the existence of missing node indices as the network indexing is assumed to be random. This may also be caused by the absence of some indices due, for example, to the removal of singular nodes. However, this can be easily managed by re-indexing the nodes orderly to remove the missing indices, at least temporarily while performing the connectivity test. The nodes can be re-indexed back to their original indices after completing the connectivity test. This difficulty can also be overcome through the use of large storage with a vacant entries for the missing indices. These vacant entries can be marked (e.g. by filling them with a certain invalid value) so that they are excluded from the involvement in the connectivity test. The latter may incur an unnecessary large storage especially for very large networks if the missing indices create large indices gaps.

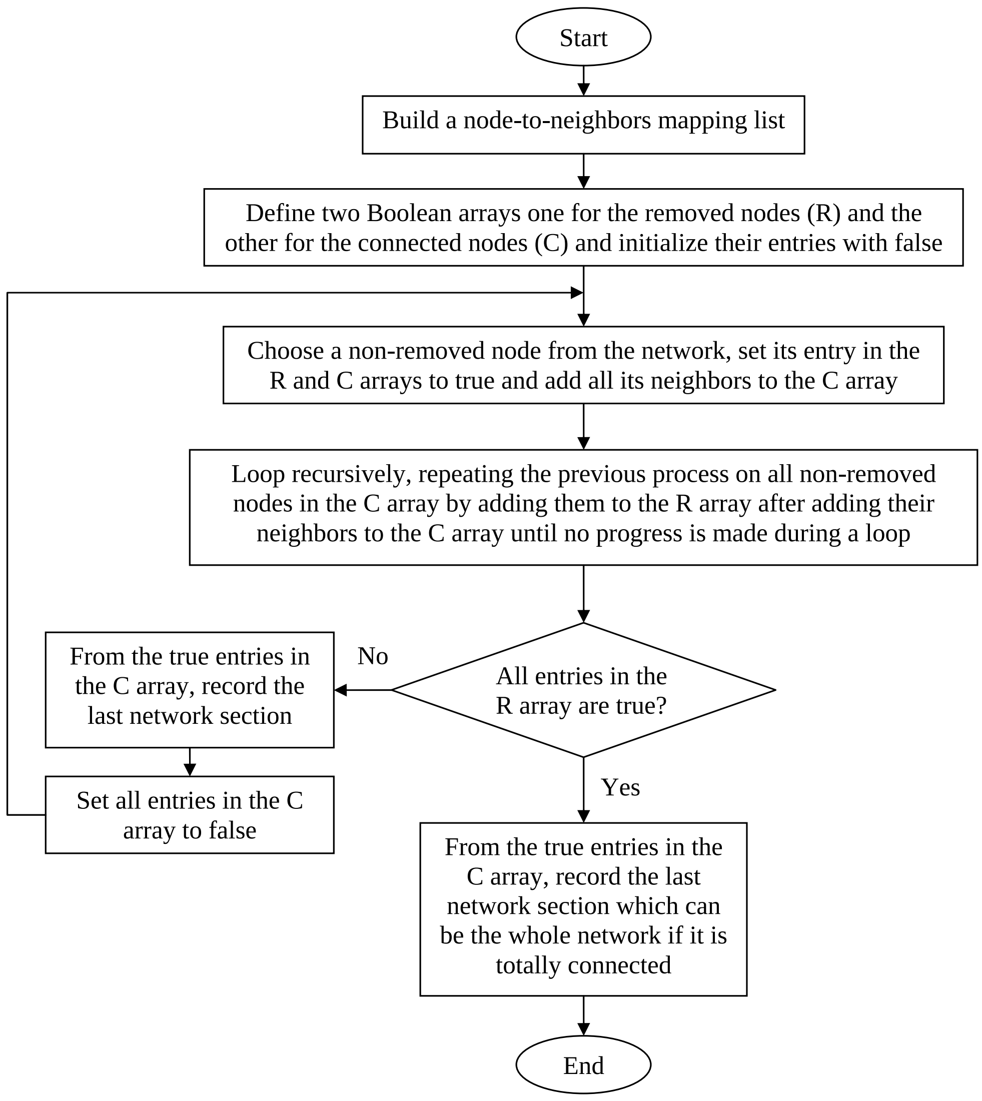
Start
Build a node-to-neighbors mapping list
Define two Boolean arrays one for the removed nodes (R) and the other for the connected nodes (C) and initialize their entries with false
Choose a non-removed node from the network, set its entry in the R and C arrays to true and add all its neighbors to the C array
Loop recursively, repeating the previous process on all non-removed nodes in the C array by adding them to the R array after adding their neighbors to the C array until no progress is made during a loop
All entries in the R array are true?
No
From the true entries in the C array, record the last network section
Set all entries in the C array to false
Yes
From the true entries in the C array, record the last network section which can be the whole network if it is totally connected
End

Figure 3: Generic flow chart of an algorithm based on the node mapping method.

2.3 Segment Mapping

In this highly efficient method, the search for connectivity starts from seeding a list of connected nodes by the two nodes of a randomly selected segment. By going through the remaining segments and adding the node of any segment whose other node is found on the connected nodes list, a connected section, which possibly comprises the whole network, will gradually build up. All segments whose nodes are added to the list are removed from the segments list either directly or by the use of a labeling mechanism such as a 1D Boolean array to mark the status of the segments as being removed or not. The inspection of the segments list is repeated until the segments list is empty (in which case the network is totally connected) or the inspection of all the remaining segments in the list in one of the iteration cycles returns no new nodes to be added to the connected nodes list (in which case the network is dismembered and partially connected). In the latter case, this inspection process can be repeated iteratively to identify all the sections of the network until the exhaustion of the entire segments list.

The input data required for this method is a list of the network segments where each segment is identified by the indices of its two end nodes. The efficiency of this method can be enhanced by employing a direct access technique to the nodes status (as being included in the current section or not) through possible use of an indexed array without need for searching a nodes list. The use of a 1D Boolean array, whose implicit cell index can be used to indicate the node index, can therefore facilitate the marking of the nodes if they are connected to the current section or not. All the entries of this array may be initially set to false at the start of each section search, and the entries of the connected nodes can be set to true as the search goes on. By the end of each section search, the nodes that belong to that section (which possibly include the whole network) can be identified from the implicit indices of the true cells. A similar technique can also be used to identify all the segments that belong to each section. A flow chart of a possible computational algorithmic implementation based on the segment mapping method with the use of a 1D Boolean array for labeling the nodes and recording their status with respect to the current section is presented in Figure 4.

Other techniques for recording the nodes connected to the section, such as adding these nodes and their segments to a network section array, can also be used instead of the use of a 1D Boolean array. These techniques may be used to provide more detailed information about the section, such as the segments of the section as well as its nodes, with minimum post processing requirements although they should require more memory consumption and processing time.

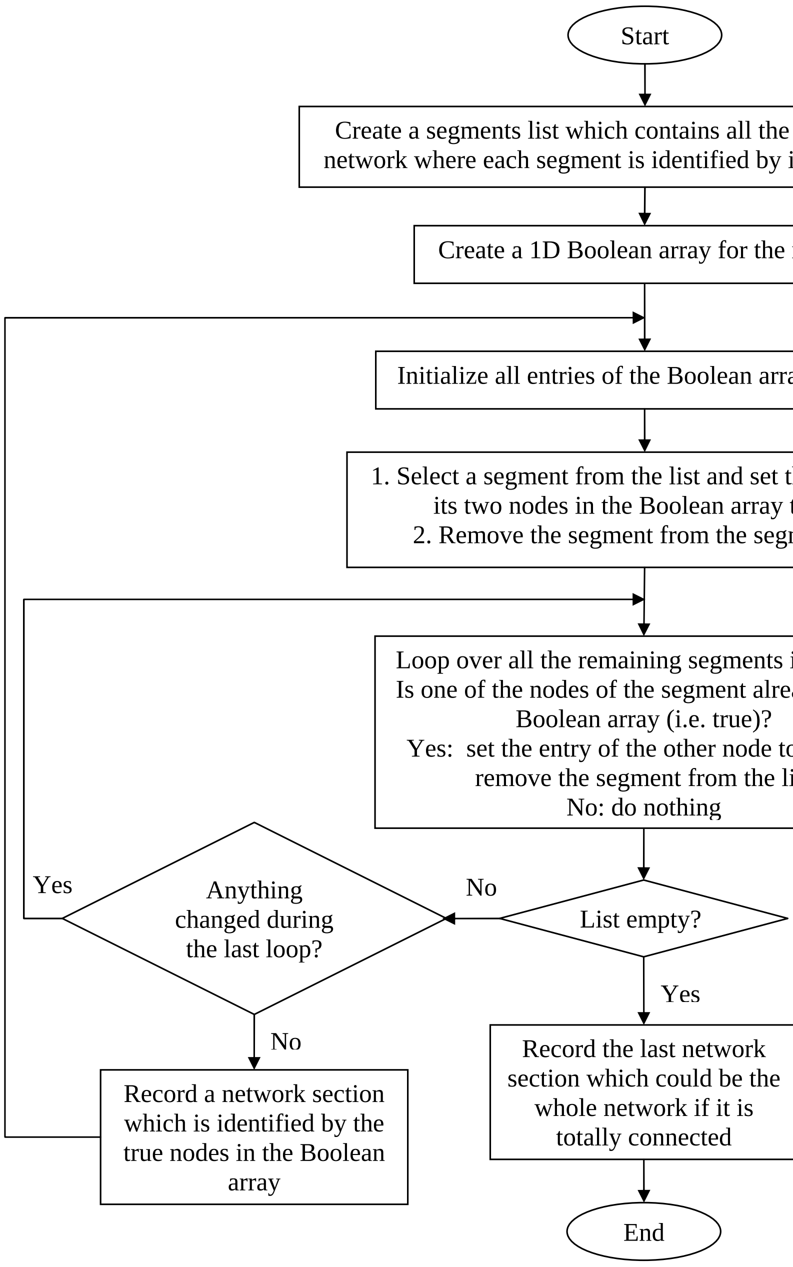
Start
Create a segments list which contains all the segments in the network where each segment is identified by its two end nodes
Create a 1D Boolean array for the nodes
Initialize all entries of the Boolean array to false
1. Select a segment from the list and set the entries of its two nodes in the Boolean array to true
2. Remove the segment from the segment list
Loop over all the remaining segments in the list: Is one of the nodes of the segment already in the Boolean array (i.e. true)?
Yes: set the entry of the other node to true and remove the segment from the list
No: do nothing
List empty?
No
Yes
Anything changed during the last loop?
Yes
No
Record a network section which is identified by the true nodes in the Boolean array
Record the last network section which could be the whole network if it is totally connected
End

Figure 4: Generic flow chart of an algorithm based on the segment mapping method.

3 Comparison

In this section, we present a general comparison between the node mapping and segment mapping methods and their associated algorithms with their advantages and shortcomings. However, we would like to insist that most of the conclusions derived from these comparisons are dependent on the particular implementation of these algorithms.

The node mapping and segment mapping algorithms, as outlined in the flow charts of Figures 3 and 4, were implemented in a C++ computer program. Many tests have been carried out; some of these are presented in Figures 5, 6 and 7. In these figures, the average execution time from several runs has been taken to smooth out fluctuations. In all these tests, a Visual Studio 6.0 compiler on a normal laptop computer with a 1.99 GHz processor and 1.87 GB of RAM memory running under Windows XP operating system was used. All the networks utilized in these tests are computer generated using a stochastic computational procedure based on establishing random connections between randomly selected nodes with certain constraints on the number of nodes, connectivity index and number of sections.

As seen in Figures 5, 6 and 7, although the segment mapping has a better performance with respect to the network size (as quantified by the number of nodes and number of segments which is related to the average connectivity index) when the network is totally connected, the performance of the node mapping becomes superior for partitioned networks.

The memory requirement of the node mapping, as outlined in the flow chart of Figure 3, is $2N$ Boolean storage plus Nc_{av} integer storage where N is the total number of nodes and c_{av} is the average connectivity index of the network. For the segment mapping, the memory requirement, as outlined in the flow chart of Figure 4, is N Boolean storage plus $2M$ integer storage where M is the number of segments. Because $2M = Nc_{av}$, the segment mapping requires less memory according to the implementation of Figure 4. However, if the removal of segments from the segments list is achieved not directly but through the use of a segments Boolean array to mark the removed segments, then an extra M Boolean storage is required for the segment mapping, and hence its memory requirement will exceed the requirement of the node mapping when $c_{av} > 2$, which is typically the more common case. Anyway, the memory costs for both algorithms are affordable with modest computational resources even for very large networks.

An advantage of the segment mapping is that more detailed information about the network sections (that is information about both nodes and segments of sections) can be obtained directly with no need for extra computational effort. In node mapping, what is found directly is the nodes of each section, and hence extra effort is required to obtain information about the segments in these sections.

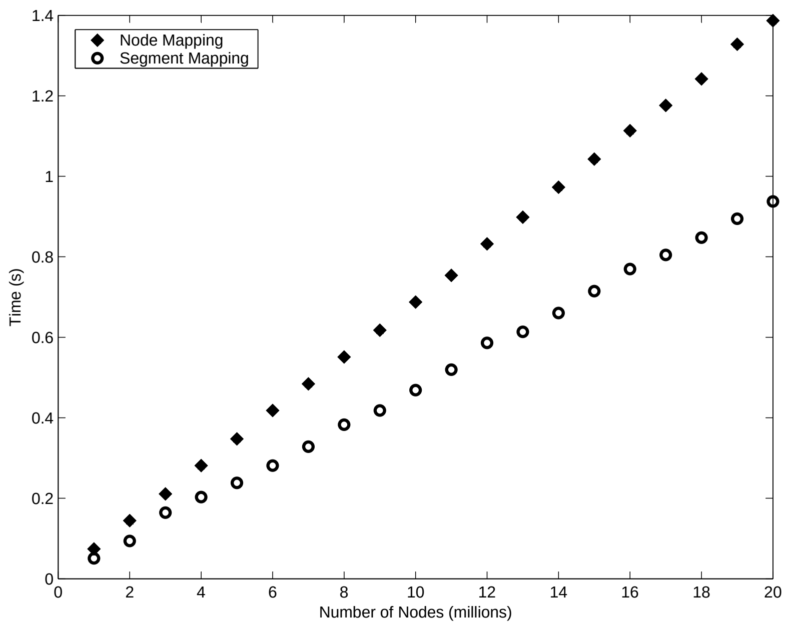

Figure 5: Execution time in seconds versus the number of network nodes in millions for the node mapping and segment mapping algorithms. All networks used in these tests have a single section with an average connectivity index of approximately 5.

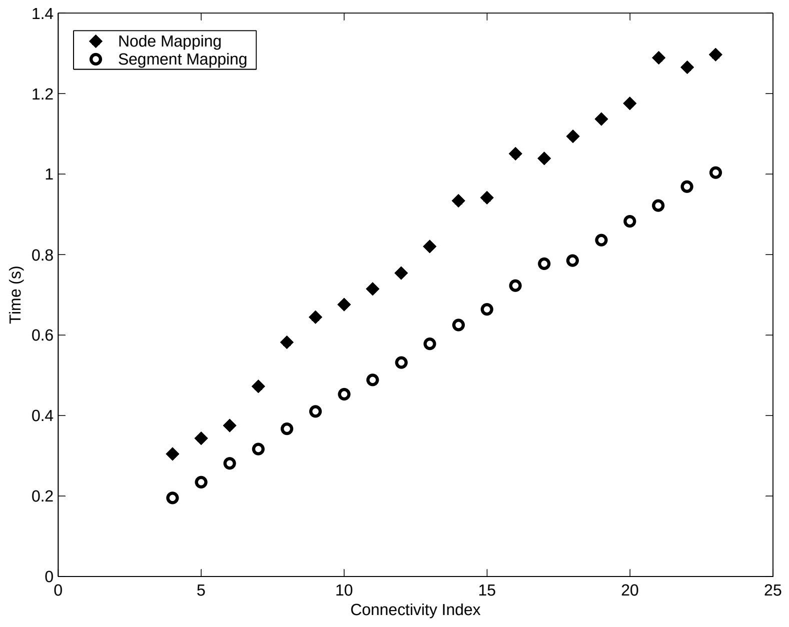

Figure 6: Execution time in seconds versus the average connectivity index of the network for the node mapping and segment mapping algorithms. All networks used in these tests have a single section with five million nodes.

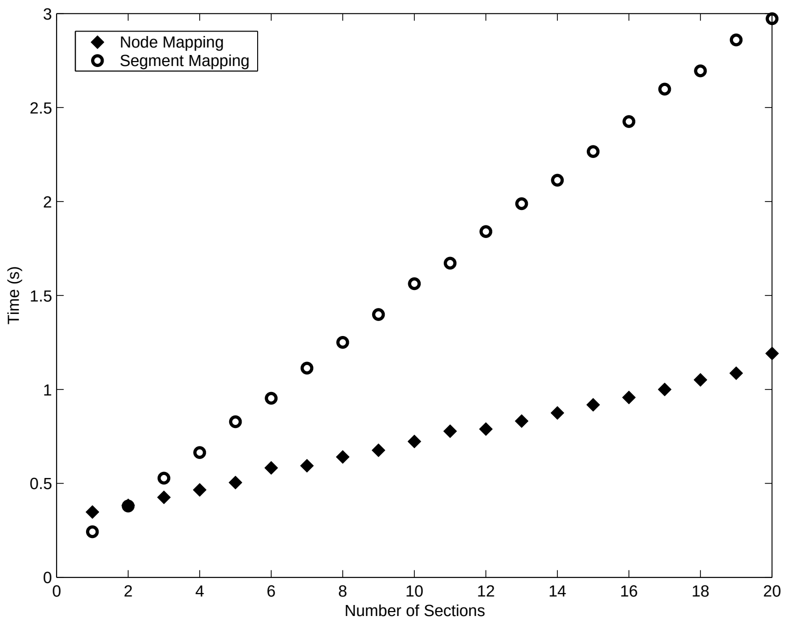

Figure 7: Execution time in seconds versus the number of network sections for the node mapping and segment mapping algorithms. All networks used in these tests have about five million nodes with an average connectivity index of approximately 5.

4 Conclusions

The importance of networks in many scientific and engineering applications cannot be overstated. Therefore, testing and validation of network connectivity is crucial to obtain valid computational results. In this article, two highly optimized methods, node mapping and segment mapping, for verifying network connectivity with their associated computer algorithms have been presented. Both methods have their merits. However, the main advantage of both is that they can be easily implemented and used for testing the connectivity and acquiring the network sections.

As implemented, segment mapping is superior in terms of speed of execution for totally connected networks, while node mapping is superior for partitioned networks. The superiority of these algorithms with regard to the memory storage requirement depends on the network characteristics, such as average connectivity index, as well as the particular algorithmic implementation. Other factors that affect the performance of these methods include the type of network and its topology, the type of iteration used to loop over the nodes and segments, connectivity index distribution, and the order of storing the nodes and segments in their arrays which depends on the indexing.

According to the author's implementation of the node mapping and segment mapping algorithms, both memory and time of execution scale linearly with the size of the network as quantified by the number of nodes and number of segments which is correlated to the average connectivity index of the network (refer to Figures 5 and 6). The time of execution also scales linearly with the number of sections (refer to Figure 7).

References

[1] P.E. Øren; S. Bakke; O.J. Amtzen. Extending Predictive Capabilities to Network Models. *SPE Annual Technical Conference and Exhibition, San Antonio, Texas, SPE 38880*, 1997. 3

[2] T. Sochi. *Pore-Scale Modeling of Non-Newtonian Flow in Porous Media*. PhD thesis, Imperial College London, 2007. 3

[3] T. Sochi; M.J. Blunt. Pore-scale network modeling of Ellis and Herschel-Bulkley fluids. *Journal of Petroleum Science and Engineering*, 60(2):105–124, 2008. 3

[4] S.J. Sherwin; V. Franke; J. Peiró; K. Parker. One-dimensional modelling of a vascular network in space-time variables. *Journal of Engineering Mathematics*, 47(3-4):217–250, 2003. 3

[5] S. Roux; A. Hansen. A new algorithm to extract the backbone in a random resistor network. *Journal of Physics A*, 20:L1281–L1285, 1987. 3

[6] N. Ratnayake; I.N. Jayatilake. Study of transport of contaminants in a pipe network using the model EPANET. *Water Science and Technology*, 40(2):115–120, 1999. 3

[7] R.W. Bentley; T.A. Lambe. Assignment of traffic to a network of signalized city streets. *Transportation Research Part A: General*, 14(1):57–65, 1980. 3

[8] S. Sarvotham; R. Riedi; R. Baraniuk. Network and user driven alpha-beta on-off source model for network traffic. *Computer Networks*, 48(3):335–350, 2005. 3

[9] B.C. Soh; S. Young. Network system and world wide web security. *Computer Communications*, 20(16):1431–1436, 1998. 3

[10] P. Crucitti; V. Latora; M. Marchiori. A topological analysis of the Italian electric power grid. *Physica A: Statistical Mechanics and its Applications*, 338(1-2):92–97, 2004. 3

[11] L.E. Meester; S. Muns. Stochastic delay propagation in railway networks and phase-type distributions. *Transportation Research Part B: Methodological*, 41(2):218–230, 2007. 3

[12] T. Sochi. Emissivity: A program for atomic transition calculations. *Communications in Computational Physics*, 7(5):1118–1130, 2010. 3