Computer Technology and Application 4 (2013) 625-634

Graph-Theoretic Approach to Network Analysis

Nabil Hassan

Professor and Chairman Electrical Engineering Department, WVS Tubman College, Harper, Maryland County, Liberia

Received: December 03, 2013 / Accepted: December 18, 2013 / Published: December 25, 2013.

Abstract: Networks are a class of general systems represented by their UC-structure. Suppressing the nature of elements the network becomes a weighted graph visualizing the constraints imposed by interconnections rather than the elements themselves. These constraints follow generalized Kirchhoff's laws derived from physical constraints. Once we have a graph; then the working environment becomes the graph-theory. An algorithm derived from graph theory is developed within the paper in order to analyze general networks. The algorithm is based on computing all the spanning trees in the graph G with an associated weight. This weight is the product of admittance's of the edges forming the spanning tree. In the first phase this algorithm computes a depth first spanning tree together with its cotree. Both are used as parents for controlled generation of off-springs. The control is represented in selecting the off-springs that were not generated previously. While the generation of off-springs, is based on replacement of one or more tree edges by cycle edges corresponding to cotree edges. The algorithm can generate a frequency domain analysis of the network.

Key words: UC-structure, network, spanning tree, depth-first search, spanning trees generation algorithm.

1. Introduction

A general system can be defined by the universe of discourse and couplings (UC-structure). That is, a system S is given by a set of elements, their permanent behaviors, and a set of couplings between the elements and environment.

In other words, a system S is defined by the 2-tuple

$$S = (B, C) \quad (1)$$

where $B = \{b_1, b_2, \ldots, b_r\}$ is the behaviors of the elements in the set of elements $A = \{a_0, a_1, \ldots, a_r\}$, a_0 denotes the environment, and $C = \{c_{ij} \mid c_{ij} = A_i \cap A_j, i \neq j\}$ is the set of couplings between elements, and A_i is the set of principle quantities on a_i, that is, a sampling of external quantities or observed quantities on a_i .

A class of UC-structure general systems is the network systems. Many physical and social systems are included in this class: For example, electrical systems, mechanical translation systems, mechanical rotational systems, thermal systems, managerial systems, experiential systems, etc [7, 8]. In a network the element represents a constraint. A second constraint is imposed by the network topology. This later constraint manifests itself in the application of generalized Kirchhoff's laws. That is, nodes in the network are non-accumulating and similarly its loops. The violation of the former law results in excessive continuous storage in one or more nodes, while violation of the later leads to unlimited amplifying loops or at least non observable loops. Both these phenomena are constrained in physical systems.

Topological properties of a network lead to the study of constraints imposed by the coupling laws. To this end, we start by suppressing the nature of individual elements. Therefore, we redraw the network with a line representing each branch of the network. The resulting outcome is the graph of the network.

The term topology is used in two meanings: (1) In modern mathematics, a "space" is a collection of elements together with some kind of mathematical structure governed by axioms. To specify a topological "structure" for a space is to specify a certain special class of subsets of elements, subject to certain axioms. These particular, subsets are called "open sets". The

Corresponding author: Nabil Hassan, professor, doctor, research fields: system and AI. E-mail: profdrnabil63@yahoo.com.

collection of all open sets forms what is called "the topology" of the space; (2) in geometry, topology is its branch studying those properties of spaces which depend only on their topologies and not other structural features of the space.

We are using this term on the second interpretation. Hence, topologically equivalent bodies are equivalent under all elastic deformations. That is, stretching, twisting, squeezing, pulling, or bending. Two networks are equivalent if there is an elastic deformation that leads from one network to the other. In addition, they are equivalent if they differ only in the elements constituting their branches.

A general system represented by its UC-structure may be expressed in the form of the general class of networks. Since the elements can be represented as the interconnection of bases elements in the form of integrators, differentiators, delay elements and multipliers [9]. The coupling between elements is depicted in the network topology. The frequency domain presentation of these elements is $1/j\omega x$, $j\omega x$, $e^{-j\omega T}$ and α, respectively.

A general system with multiple inputs can be analyzed similarly using the superposition principal. That is, the response due to each input is considered separately, and then the complete response is found by summing these partial responses.

A network with suppressed nature of its constituting elements is a weighted graph. Therefore, graph theory becomes the working environment.

2. A Graph-Theoretic Approach

2.1 Terminology

A graph G is an ordered pair of disjoint sets (V, E) such that E is a subset of the set of unordered pairs of V.

When the elements of E are ordered pairs, then G is a directed graph.

The set V is the finite set of vertices, and the set E is the finite set of edges. We shall not consider infinite graphs.

An edge {x, y} joins the vertices x and y, and is denoted by xy. If $xy \in E(G)$ then x and y are adjacent or neighboring vertices of G, and the vertices x and y are incident in the edge xy. The set of vertices adjacent to a vertex x is denoted by $\Gamma(x)$. Two edges are adjacent if they have exactly one common end vertex.

The order of a graph G is the number of vertices n, in G and is denoted by |G|. The size of G is the number of its edges m and denoted by e(G). The degree of a vertex x is the number of adjacent vertices $|\Gamma(x)|$ and is denoted by d(x). The minimum degree of a vertex in a graph is denoted by δ and its maximum degree by Δ. Since each edge has two vertices, then the sum of degrees of all vertices in a graph G is twice the number of edges.

A graph is simple if it has no loops (an edge joining a vertex to it) or multiple edges (two or more edges joining pairs of vertices in G). It is oriented when the positive direction is set for each edge.

If a real number is associated with each edge, then G is network or a weighted graph. In the present paper, we extended this term to include the association of a complex number to each edge.

The adjacency matrix A of a graph G is the n × n matrix having entries a_{ij} defined by:

$$a_{ij} = \begin{cases} 1 & \text{if } v_i v_j \in E(G) \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

The incidence matrix B of a graph G is the n × m matrix having entries b*ij* defined by:

$$b_{ij} = \begin{cases} 1 \text{ if } v_i \text{ is the initial vertex of the edge } e_i \\ -1 \quad \text{if } v_i \text{ is terminal vertex of } e_i \\ o \quad \text{otherwise} \end{cases} \quad (3)$$

In an independent set of vertices, no two elements are adjacent. Similar definition holds for edges. However, a set of paths is independent if for any two paths each vertex belonging to these two paths is an end vertex of both.

A walk W in G is an alternating sequence of vertices and edges, e.g., v0, e1, v1, e2…el, vl, where $e_i = v_i - v_i,\ 0 \le i \le 1$. This walk is denoted by v0, v1…vl, and its length is l.

A path is a walk with distinct vertices, hence it has distinct edges.

A trail is a walk with distinct edges, but not necessarily of distinct vertices. A closed trail (whose

end vertices coincide) is called a circuit. A closed path is a cycle.

A connected graph G is a graph in which there is a path for every pair of vertices $\{v_i, v_j\}$. If in G, there is two paths for each pair of vertices $\{v_i, v_j\}$, then G is bi-connected. If v_i and v_j are connected, then v_j and v_i are also connected.

A subgraph $G^{/}(V^{/}, E^{/})$ of the graph G (V, E) is a graph whose vertex set $V^{/} \subset V$, and edge set $E^{/} \subset E$. Hence we write $G^{/} \subset G$. If $G^{/} = G(V^{/})$, then $G^{/}$ is induced graph of G.

The relation R defined as v_i R v_j if v_i and v_j are connected is an equivalence relation. Hence, R partitions the set V into pair wise disjoint subsets of V. The subgraph induced by any such subset is a maximal connected subgraph. A maximal connected subgraph is a component of G.

If the deletion of a vertex increases the number of components in the graph G, then this vertex is a cut vertex. An edge defined similarly is a bridge.

A graph without any cycle is a forest. A connected forest is a tree. A forest having exactly two components is a thicket. Note that a forest is a disjoint union of trees.

A spanning tree of a graph is the tree that contains each vertex of the graph G. Each connected graph has at least one spanning tree.

Once a spanning tree is defined for a graph G, the remaining edges are called links or chords. Their collection is called a complementary tree or cotree.

A cut set is a minimal set of branches partitioning the vertex set V into two disjoint sets V_1 and V_2.

A fundamental loop is a loop that contains one and only one link. The number of fundamental loops is m – n +1.

A fundamental cut set is a cut set that contains one and only one spanning tree branch.

We define:

1. $\Omega = \{\omega_1, \omega_2, \ldots, \omega_k\}$ to be the set of frequencies at which the network is analyzed;

2. The source node s, and the sink node t;

3. The product of branch admittance's is denoted by w(T) for the spanning tree T of the graph G. We shall call w(T) the weight of T;

4. N is sum of weights of spanning tree in the graph G;

5. N (s, a, b, t) is the sum of weights of spanning trees of graph G in which the edge a b (in this order) occurs in the unique path from s to t;

2.2. The Approach

Several results from graph theory relate closely to network analysis. A cut set satisfies the generalized Kirchhoff's first law. Otherwise the components of a graph become accumulating. Also, the generalized second Kirchhoff's law is satisfied in any loop in the network. A spanning tree encompasses all nodes in the network by connecting them by exactly n – 1 independent branches. Thus, a spanning tree can provide the necessary n – 1 equations in n – 1 independent variables necessary for the network analysis. These variables are rates of displacements, that is, velocity in mechanical translation network, angular velocity in mechanical rotational networks, potential in electrical networks, temperature in thermal net-works, velocity in traffic networks, etc. Analogously, the flow in the m – n +1 links enables the calculation of the flow in each of the n – 1 branches by summing up the flow in the n – 1 fundamental cut sets. The exact meaning of flow is class dependent. It is the force in mechanical translation networks, torque in mechanical rotation networks, current in electrical networks, rate of heat flow in a thermal network, and rate of cars in a traffic network, etc.. If we define loop flows for each of the fundamental loops, we can obtain the n – 1 independent edge flows, since each link is incorporated in exactly one fundamental loop. By applying the second generalized Kirchhoff's law to each fundamental loop we obtain a set of independent equations.

Theorem 1: Assuming a unit flow to the source node s, and out of the sink node t, then the flow in the edge ab

(in this order) in the weighted oriented graph (oriented network) is given by $w_{ab} = \{N(s, a, b, t) - N(s, b, a, t)\}/N$. The assumption of unit flow is not restrictive, since normalization of flow is always possible.

Proof: it is necessary and sufficient to verify the generalized Kirchhoff's laws.

At the source node s for each spanning tree there exists exactly one neighbor v(T) of s that is in the s-t path contained in T; denoted as P(T). Therefore,

$$\sum_{b\in\Gamma(s)} N(s,s,b,t) = N \tag{4}$$

While

$$\sum_{b\in\Gamma(s)} N(s,b,s,t) = 0 \tag{5}$$

Hence,

$$\sum_{b\in\Gamma(s)} w_{sb} = 1 \tag{6}$$

That is, the generalized Kirchhoff's first law is satisfied at s.

By a similar argument, it follows that this law is satisfied at the sink node t. That is:

$$\sum_{a\in\Gamma(t)} w_{at=1} \tag{7}$$

Let us assume no other node is an input or output node. Assume further that each value w is multiplied by N. Then the contribution of a spanning tree T to the current entering vertex v_i then leaving it falls in one of the two cases:

(1) If v_i is not on the s-t path contained in T, P(T), then no contribution of T to either N(s, v_j, v_i, t) or N(s, v_i, v_k , t);

(2) If v_i is on the s-t path P(T) then T contributes w(T) to N(s, v_j, v_i, t) and N(s, v_i, v_k, t) where v_j, v_i, v_k are three consecutive nodes in the path P(T), that is, the contribution of T is w(T) to the current into v_i and the same to current out of v_i. Thus the first law is satisfied.

To verify the second generalized Kirchhoff's law let us de-fine a thicket F as:

$$F = F_s \cup F_t \tag{8}$$

where, F_s is the component that comprises source node s, and F_t is the component with sink node t.

However, the thickets contributing to the rate of displacement in cycle including the vertices a, b are those for which $a \in F_s$ and $b \in F_t$ or vice versa. Then the contribution of a thicket F to the total rate of displacement in a cycle is their algebraic sum over cycle edges. This sums to zero since the rate of displacement over each edge is w_{ab}/D, where D is admittance of the branch and the thicket is contributing to w_{ab} and w_{ba} by the same value of weight. To illustrate the verification of the second generalized Kirchhoff's law, look at Fig. 1.

The contribution of the cycle between a, b vertices to f_t is by D_{eq}. However,

$$f_{T1} = f_T \frac{D_1}{D_{eq}} \quad \text{, and} \quad f_{T2} = f_T \frac{\frac{D_2 D_3}{D_2 + D_3}}{D_{eq}} \tag{9}$$

The rate of displacement on each edge is the corresponding flow over the edge's admittance's. Hence the sum of the rate of displacement around the cycle is zero.

In the theorem 1, it is necessary to compute N, N(s, a, b, t) and N (s, b, a, t) to determine w_{ab}. These are the outcomes of the genetic-like algorithm in Sec. 3. However, a pre-computation of the number of spanning trees and their weights can serve as end criterion or as a verification of the algorithm. This is the topic of the following theorem.

Theorem 2: The number of spanning trees of G is $|\mathbf{B}\mathbf{B}^t|$.

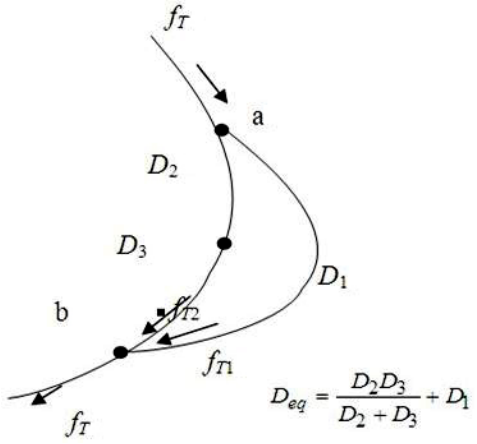

Fig. 1 Illustration of generalized second Kirchoff's law.

Proof: The proof is based on two observations:

The determination of an (n – 1) × (n – 1) submartix of B is "1" if the columns correspond to edges of a spanning tree and "0" otherwise;

The Cauchy-Binet formula of linear algebra, stating that if K is a p × q matrix (p ≤ q) and L is a q × p matrix then:

$$|\mathbf{KL}|=\sum_p |\mathbf{K}_p\mathbf{L}_p| =\sum_p |\mathbf{K}_p||\mathbf{L}_p| \qquad (10)$$

where the summation is over all subsets P of {1, 2, ..., q}, and $\mathbf{K_p}$ is the p ×p submatrix of **K** formed by the columns of **K** indexed with elements of P and $\mathbf{L_p}$ is submatrix of **L** formed by the corresponding rows of **L**.

Theorem 2 is useful in computing N(s, a, b, t) and N(s, a, b, t) and N (s, b, a, t). This requires forming the transition matrix $\mathbf{B_{sa}}$, $\mathbf{B_{at}}$, $\mathbf{B_{bt}}$, $\mathbf{B_{sb}}$ of the components of G comprising s, a; a, t; b, t and s, b. Then N(s, a, b, t) is the lower of the two numbers $|\mathbf{B_{sa}}\ \mathbf{B_{sa}}^t|$ and $|\mathbf{B_{bt}}\ \mathbf{B_{bt}}^t|$. N(s, b, a, t) is found similarly.

3. A Genetic-Like Algorithm

The computation of the following algorithm is to be repeated for each $\omega_i = i$ in the set Ω, i = 1… k. In the first phase of the algorithm a spanning tree is found using the depth first search algorithm (DFS), while preserving enough information about links (chords, or cycle edges).

3.1 The DFS Algorithm

(1) Initialization: k ← 1; j ← 1; i ← 1; V(k) ← v_i;

(2) Test of end: if k = n, then GoTo 7; Else process d(v_k); i ← i+1;

(3) Process tree edge: if v_i, v_kv_i are not labeled, then, label v_{ij}; e(k)←v_kv_i; k ← k+1; V(k) ← v_i; Goto2; Else Goto 4;

(4) Another infant: if v_i is not labeled, v_k v_i is labeled; j < d(v_k) then j← j + 1; Goto3; Else Goto 5;

(5) Backtracking if j = d(v_k); and i ≠ 1, then, k ← k – 1;i ← i – 1; Goto 2; Else Goto 6;

(6) Check of connectedness: If i = 1; k < n (it is a must at this point), then output: "Graph is not connected"; End;

(7) Cycle Edges: ic ← 1; for i = 1, m do: If e(ic) ≠ E(l), then ic ← ic + 1; ec(ic) = l;

The DFS algorithm fails if G is not connected. In this case, an error message is edited. The initialization of i ← 1 is not a must. We may start the algorithm from any vertex other than s. Then instead of i ← i + 1 in (2) we use i ← mode (i + 1, n) + 1, and i ≠ i_0, in (5) and i = i_0 in (6) where $i_0 \in [1, n]$ in the initialization of i ← i_0 in (1).

If the graph G was connected then the resulting acyclic graph of n vertices and n – 1 edges is a spanning tree. This is the initial spanning tree.

The time complexity of the DFS algorithm is ϑ(n + 2m) if the pseudo-statements do need constant time. The basic operation is comparison. Each edge could be at most investigated twice; hence there are at most 2m edge investigations. There are at most n vertices to be labeled. Hence, the above order is an upper limit. Since a graph can have at most C_2^n edges, that is, n(n – 1)/2 edges, then the worst case time complexity is $\vartheta(n^2)$.

The space complexity is ϑ(n) apart from the requirements of the graph representation. The final effect of DFS algorithm; if succeeded, is to partition the edge set E of graph G into two disjoint sets, namely, the set of tree edges, and the set of cycle (chords, or link) edges.

The functions encountered in the DFS algorithm are:

(1) VERTEX: v → [1, n]. Defined by: VERTEX [v] = k ⇔ v is the k'th vertex processed;

(2) TREE_EDGE: E → [1, n – 1]. Defined by, TREE_EDGE (i) = e_i ⇔ e_i is a tree edge, i = 1, n – 1;

(3) CYCLE_EDGE: E → [1, m – n + 1]. Defined by CYCLE_EDGE (i) = e_i ⇔ e_j is a cycle edge, j = 1, m – n + 1;

(4) V_PARENT: v → {True, False}. Defined by

$$\text{V\_PARENT (v)} = \begin{cases}\text{True} & \text{v} \notin \text{VERTEX(v)} \\ \text{False} & \text{otherwise}\end{cases},$$

(5) E_PARENT: E → {True, False}. Defined by:

$$\text{E\_PARENT (e)} = \begin{cases}\text{True} & \text{e} \notin \text{TREE\_EDGE (e)} \\ \text{False} & \text{otherwise}\end{cases}.$$

Only Vertex imposes space requirements. The TREE_EDGE and CYCLE_EDGE functions are marked on the initial graph representations.

In terms of these functions the DFS algorithm may be written as in the following:

```
Proc DFS
begin
k: = i: = j: = 1
VERTEX (k): = v_i
Process test_ END
Process tree_edge
If V_PARENT (v_k) = False and E_PARENT (v_k v_i)
= True and j< dk then
begin
j: = j + 1
Process tree_edge
end;
else
if j = dk and i ≠ 1 then
begin
k: = k – 1
i: = i – 1
process test-end
end;
else
if i = 1 and k = 1 then
begin
output, 'graph is not connected'
end DFS
end

Process test-end
if k = n then
begin
Process End_DFS
end;
else
dk = d(v_k)
Process tree_edge
If V_PARENT (v_k) and E_Parent (v_k v_i) then
begin
Tree_EDGE (k): = v_k v_i
k: = k + 1
V (k): = v_i
Process test_end
end.

Process END_DFS
ic: = 1
for l: = 1 step 1 until m do
if TREE_EDGE (ic) ≠ E(l) then
begin
ic: = ic + 1
CYCL_EDGE (ic) = l
end;
end.
```

A FORTRAN-77 program listing of DFS Subroutine is depicted in the appendix together with associated functions V and NST.

Once an initial spanning tree T1 and a cotree are determined, then we may proceed by generating off-springs from these as parent strings. The addition of a chord (link) to the computed spanning tree produces a fundamental loop. Off-springs can be generated so, by deleting one edge of the cycle (except the link) successively. Thus a number of off-springs equals the length of the cycle minus "1" is generated. It is necessary to prove that the generated off-springs are all different from previously computed spanning trees.

Theorem 3: The addition of a link (cycle edge) to a spanning tree produces a cycle of length, say l. Then it is possible to generate l-1 off-springs, each is a spanning tree, by deleting successively one tree edge from the so formed fundamental loop. Repeating this argument with off-springs as parents and all possible segmentations of the cotree, all the spanning trees of the graph G will be generated.

The inputs to the spanning trees generation algorithm are the initial spanning tree (the output of DFS algorithm), the cotree, and the transition matrix of graph G. After generating a spanning tree its weight

is computed. Then its contribution to N, N(s, a, b, t), and N(s, b, a, t) is determined. Thus the flow in the edge ab is computed by theorem 1.

3.2 Spanning Trees Generation Algorithm

```
Proc STG (i)
BEGIN
KEY: = True; CYCL_LENGTH: = TREE_RANK: =1
EC[CYCLE_LENGTH]: = CYCLE_EDGE (i)
v1: = VER1(CYCLE_EDE(i)); v2: = VER2(CYCL_EDGE));
for l: = v2 step 1 until n – 1 do
CYCL_LENGTH: = CYCL_LENGTH + 1
EC[CYCL_LENGTH]: = TREE_EDGE(l)
if VERTEX (l) = v1 then
begin
Proc TREE;
KEY: = False
end;
if KEY then
begin
CYCL_LENGTH: =1
for l: = v2 step-1 until 1 do
CYCL_LENGTH: = CYCL_LENGTH + 1
EC (CYCL_LENGTH): = TREE_EDGE(;)
if VERTEX(l): = v1 then
begin
proc TREE
end;
end;
end.
w(1): =1
for j: = 1 step 1 until n – 1 do
T1(j): = TREE_EDGE(j)
w(1) = w(1) * C (INV_EdGE (TREE_EDGE [j]))
end;
for i = m – n + 1 step-1 until 1 do
proc STG (i)
end;
N: =0; N1: =0; N2: = 0
for l: =1 step 1 until TREE_RANK do
N = N + w(l)
end;
for j: =1 step1 until TREE_RANK do
for l: =1 step1 until n – 1 do
if Tj (l) = eab and VER1(eab) = a then
begin
N1 = N1 + w(j)
end;
else
If Tj(l) = eab and VER1(eab) = b then
begin
N2 = N2 + w(j)
end;
end;
wab = (N1 – N2)/N
end.
```

Let the number of spanning trees in G be NT. Hence the space complexity of the STG algorithm is NT(n – 1). However, NT = (m – n + 1) * LC_{max}, in the worst case, where LC_{max} is the maximum cycle length. Therefore, in the worst case the space complexity depends on LC_{max} = n. That is, the space complexity is $\vartheta(n^2)$ in the worst case. The time complexity in terms of the basic operations number, that is, comparisons of off-springs is (n – 2) [$LC_1 + LC_2 + \ldots + LC_{n-m+1}$] which is once more $\vartheta(n^2)$. The computation requires NT(n – 1) basis operations, that is, $\vartheta(n^2)$. Hence, the time and space complexity of the algorithm are both of order n^2 in the worst case.

There are two more functions to use within the STG algorithm. These are:

(1) INV_EDGE: [1, n] $\rightarrow$ E. Defined by INV_EDGE[l] = k $\Leftrightarrow$ E(k) = l;

(2) INV_VERTEX: [1, n] $\rightarrow$ V. Defined by INV_VERTEX[l] = k $\Leftrightarrow$ V (k) = l.

The first do loop in the above procedure creates the function (array) EC defining the string of edges in the cycle formed by adding the edge of the cotree. The function VER1(e) returns the initial vertex of the edge e, and VER2(e) returns its end vertex.

3.3 Process Tree

```
TR: = TREE_RANK
for j: = 1 step 1 until TREE_RANK do
TR: = TR + 1
for l: = 1 step 1 until n – 1 do
T_TR (l): = T_j (l)
end;
end;
iv: = INV_VERTEX (v_2)
for j: = 1 step1 until CYCL_LENGTH-1 do
for l: 1 step1 until j do
T_TR (iv – j + CYCL_LENGTH-1+l): = EC(l)
end;
process TEST
TREE_RANK: = TR
w(TR): = 1
for kw: = 1 step1 until n – 1 do
w(TR): = w(TR)*C(INV_EDGE (T_TR [kw])
end;
end.

Process TEST
TR: = TR + 1
for j: =1 step 1 until TR-2 do
if T_TR-1: = T_j then
begin
TR: = TR-2
end;
end.
```

The statement $T_{TR-1} = T_j$ is to be expanded during programming. For example, the criterion for comparison may be chosen as the sum of edges numbers in the tree. Since the sum of n natural numbers is equal if an only if there is a one-to-one correspondence between the elements in each set.

When the weights of edges are complex, then it is necessary to repeat all computation for each element of Ω. This does not require the computation of all spanning trees once more. It just requires computing the weights of the resulting spanning trees for each ω_i.

To illustrate the usefulness of the algorithm we consider the bridge network in Fig. 2.

Direct computation gives flow of –2/21 in the edge ab. The number of spanning trees in the network is NT = 8.The algorithm generates these as shown in Fig. 3.

The computation of flow in the edge ab by the given algorithm results in w_{ab} = -.0.09524 which compares well with the value –2/21 within round-off.

3. Summary and Conclusions

General systems representation is possible using their UC-structure. This structure alone, or combined with other representations, may serve as a system description Networks can serve as an important general class of the system's representations. Subclasses of networks are electrical systems, mechanical transnational systems, mechanical rotational systems, the thermal systems, traffic systems, social systems, etc..

When the nature of elements in a network is suppressed then the network is a weighted graph.

Therefore graph theory may present the working environment for network analysis. In this paper an algorithm based on theorem1 was derived. The nature of the algorithm is similar to genetic algorithm but with rather controlled off-springs generation. The parent strings are generated by a depth-first-search (DFS) algorithm. The output of DFS, if successful, is two strings one is the spanning tree and the second is the cotree. These are used in a spanning tree generation (STG)

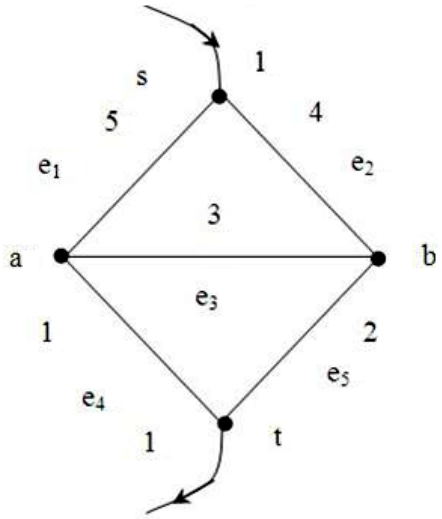

Fig. 2 An example bridge network.

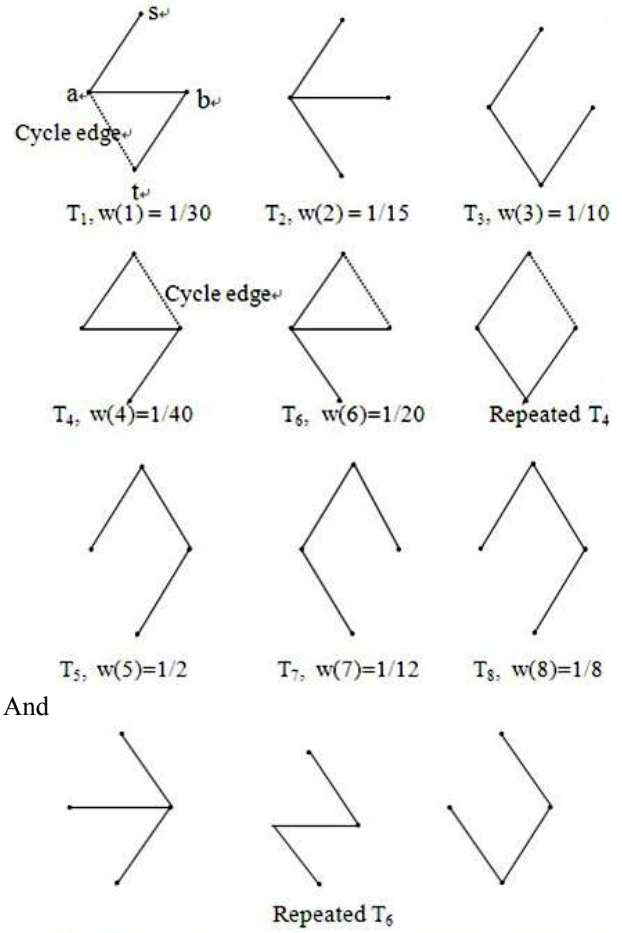

Fig. 3 Generation of spanning tree.

algorithm for generating the spanning trees and performing the requested analysis. The algorithm is applied successfully for the analysis of a number of networks. An example is presented within the paper.

When the weights, that is, the numbers associated with edges, are complex, then the analysis is repeated for each frequency in the set Ω. Hence, the frequency response analysis is performed.

References

[1] A.J. Kfoury, R.N. Moll, M.A. Arbib, A Programming Approach to Computability, Springer Verlag, NewYork, 1982..

[2] A.E. Taylor, General Theory of Functions and Integration, Dover Publications, Inc, New York, 1985.

[3] B. Bollabas, Graph Theory: An introductory Course, Springer Verlag, New York, 1979.

[4] S. Even, R.E. Tarjan, Computing an st-numbering, Theoretic Computer Science (2) (1976) 393-344 and (4) (1977) 123.

[5] G.M. Weinberg, A computer approach to general systems theory, in: George J. Klir (Ed.), Trends in General Systems Theory, Wiley—Interscience, New York, 1972.

[6] J.W. Nilson, Electric Circuits, fourth Ed., Adison Wisely Publishing Company, New York, 1993.

[7] J.J. D'Azzo, G.H. Houpis, Feedback Control Systems Analysis & Synthesis, second Ed., Mc Graw—Hill International, London, 1966.

[8] J.H. Milsum, The hierarchical basis of general living systems, in: George J.Klir (Ed.), Trends in General Systems Theory, Wiley Interscience, New York, 1972.

[9] Prof. Dr. Nabil Hassan, Analog & Digital Systems Analysis and Design, Dar Al Maarefa, Cairo, 1996.

[10] Prof Dr. Nabil Hassan, The Art of Programming in FORTRAN, Dar Al Ketab Al Hadeeth, Cairo, 1996.

[11] R.A. Orchard, On an Approach to General Systems Theory, Wiley—Interscience, New York, 1972.

[12] W. Stallings, Data and Computer Communication, second Ed., Mc millan Publishing Company, New York, 1989.

[13] V.K. Balakrishman, Introductory Discrete Mathematics, Prentice-Hall International, Englewood cliffs, New Jersy, 1991.

Appendix

```
SUBROTINE DFS(IB, NV, ME, LE, K, NT, N, M, MAG, MEG)
DIMENSION IB(N, M), MAG(N1), MEG(N1),
+NV(NT, N), ME (NT, N – 1), LE(2)
* MAG array is a temporal store for neighbor vertices.
* MEG array is a temporal store for neighbor edges.
* Logical function V excludes the previously used edges of current tree.
* Logical variable key is used to extract cycle edges (chords) after the spanning tree is found.
LOGICAL V, KEY
DATA KEY/. TRUE./
DO 40 IU = 1, N – 1
MAG (IU) = 0
40 MEG (IU) = 0
*IT variable determines the rank of current parent.
IV = 1
IT = 1
KK = K
```

```
NV(1,1) = K
2 IF (IV. EQ.N) GOTO 100
* Call of function NST
IJ = NST(MAG,MEG,IB,K,N,M)
* Note:K variable here is used for the current vertex.
DO 1 J = IT, IJ
K = MAG (J)
IF (V(K, NV, NT, N) THEN
*KE variable symbolizes the computed neighbor edge.
* IV variable symbolizes the rank of that edge.
KE = MEG (J)
ME (1, IV) = KE
IV = IV +1
GOTO 2
ENDIF
1 CONTINUE
* Test of connectedness of the graph.
IF (IV.EQ.KK.AND.IV.LT.N) THEN
PRINT *'GRAPH IS NOT CONNECTED!!'
GOTO 100
ENDIF
* Note: K here is the parent in the spanning tree of the, current
vertex.
K = NV (1, IV-1)
IT = IT +1
GOTO 2
IK = 0
* Test of the previous use of the current edge.
DO 3 I2 = 1, M
DO 4 I3 = 1, N – 1
IF (ME(1, I3).NE.I2) GOTO 4
KEY = FALSE.
4CONTINUE
IF (KEY) THEN
IK = IK+1
LE (IK) = I2
ENDIF
KEY = .TRUE.
3CONTINUE
K = KK
RETURN
END
*The logical function V returns true when the input vertices not
included in the list computed spanning tree vertices.
LOGICAL FUNCTION V(K, NV, NT, N)
DIMENSION NV(NT,N)
V = .TRUE.
DO 1 I = 1, N
IF (K.NE.NV (1, I) GOTO 1
V = .FALSE.
GOTO 100
1 CONTINUE
100RETURN
END
* Function NST (Neighbor of current vertex in the Spanning
*Tree) stores spanning tree edges in MEG,
* spanning tree vertices in MAG, and returns. The number of
*determined spanning tree vertices.
FUNCTION NST (AG, MEG, IB, II, N, M)
DIMENSION MAG(N – 1), MEG(N – 1), IB(N, M)
* II variable is the rank input vertex.
* IJ variable is the rank of current neighbor.
IJ = 0
DO 1 I = 1, M
* Row II search in IB matrix for the first neighbor edge.
If (IB(II, II).EQ.0) GOTO 1
* K variable stores the current neighbor vertex.
K = 0
IJ = IJ + 1
MEG (IJ) = I
4K = K + 1
IF (IB(K, I).EQ.0.OR.K.EQ.II) GOTO 2
MAG (IJ) = K
* Test of end of the computing loop.
2IF (K.LT.N) GOTO 4
1CONTINUE
*NST takes the value of IJ, rank of final neighbor.
NST = IJ
RETURN
END
```