An Optimal Algorithm to Detect Balancing in Common-edge Sigraph

Deepa Sinha
South Asian University,
Akbar Bhawan, Chanakyapuri,
New Delhi – 110 021.

Anshu Sethi
Center for Mathematical Sciences,
Banasthali University,
Banasthali – 304 022

ABSTRACT
A signed graph (or sigraph in short) S is a graph G in which each edge x carries a value $s(x) \in \{+1, -1\}$ called its *sign* denoted specially as $S = (G, s)$. Given a sigraph S, a new sigraph $C_E(S)$, called the common-edge sigraph of S is that sigraph whose vertex-set is the set of pairs of adjacent edges in S and two vertices of $C_E(S)$ are adjacent if the corresponding pairs of adjacent edges of S have exactly one edge in common, and the sign of the edge is the sign of the common edge. If all the edges of the sigraph S carry + sign then S is actually a graph and the corresponding common-edge sigraph is termed as the common-edge graph. In this paper, algorithms are defined to obtain a common-edge sigraph and detect whether it is balanced or not in $O(n^3)$ steps which will be optimal in nature.

Keywords
Algorithm, sigraph, common-edge graph, common-edge sigraph, balanced signed graph.

1. INTRODUCTION
For standard *terminology* and notation in graph theory, except for those that are specifically defined here, the reader is referred to West [33] and for algorithms, refer to Coreman [12]. Throughout the text, finite, undirected graph with no loops or multiple edges are considered. A graph having n vertices and e edges; is denoted by (n, e) where n is called the order and e is called the size of G. In computers, any graph G is observed as network by computer scientist where vertices are taken to be nodes and edges to be taken as links.

In the spirit of a study of graph-valued functions, obtaining the line graph $L(G)$ of a given graph $G = (V, E)$ may be treated as a mapping L that operates on G to give rise to $L(G)$ as the graph whose vertices are the edges of G with two of these vertices joined to each other (or, adjacent) whenever the edges of G they represent have a common vertex in G or equivalently the two edges form a P_3 in G. H is called line graph if and only if ∃ a graph G such that H ≅ L(G).
.
Broersma and Hoede [9] defined in general path graphs $P_k(G)$ of G for any positive integer k as follows: $P_k(G)$ has for its vertex-set the set P_k^G of all distinct paths in G having k vertices, and two vertices in $P_k(G)$ are adjacent if they represent two paths $P, Q \in P_k^G$ whose union forms either a path P_{k+1} or a cycle C_k in G. Some improvement of their paper was subsequently given by [27, 7, 28].

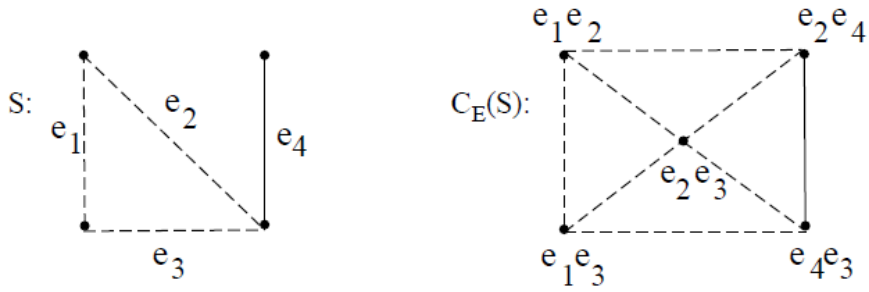
Much earlier, making independently the same observation as above on the formation of a line graph $L(G)$ of a given graph G, Kulli [32] had defined the common-edge graph $C_E(G)$ of G as the intersection graph of the family $P_3(G)$ of 2-paths (i.e., paths of length two) each member of which is treated as a set of edges of the corresponding 2-path; as shown by him, it is not difficult to see that
$C_E(G) \cong L^2(G)$
for any isolate-free graph G, where $L(G) := L^1(G)$ and $L^t(G)$ denotes the t-th iterated line graph of G for any integer $t \geq 2$.

The notion of $L(G)$ has been extended to the realm of *signed graph* (or *sigraph* in short) [8]. As in [9] (also see, [7]) by a sigraph S we mean a 2 graph $G = (V, E)$ called the underlying graph of S and denoted by S^u, in which each edge x carries a value $s(x) \in \{+1, -1\}$ called its sign; an edge x is positive or negative according to whether $s(x) = +1$ or $s(x) = -1$. The set of positive edges of S is denoted by $E^+(S)$ and $E^-(S) = E(G) - E^+(S)$ is the set of negative edges of S. Graphs themselves regarded as sigraphs in which every edge is positive. Given

a graph G, let $\varphi(S)$ denote the set of all sigraphs whose underlying graph is G. In general, a subgraph S' of a sigraph S is said to be *all-positive*(*all-negative*) if all the edges of S' are positive (negative). A sigraph is said to be *homogeneous* if it is either all-positive or all-negative and *heterogeneous* otherwise. *Cliques* are defined as the complete subgraphs of the graph.

For a sigraph S, Behzad and Chartrand [8] defines its common-edge sigraph, $C_E(S)$ as the sigraph whose vertex-set is the set of pairs of adjacent edges in S and two vertices of $C_E(S)$ are adjacent if the corresponding pairs of adjacent edges of S have exactly one edge in common, with the same sign as that of common edge.

A sigraph *S* and its common-edge sigraph $C_E(S)$ is shown in Figure 1.



The number of positive (negative) edges incident at vertex v, denoted by $d^+(v)(d^-(v))$ is called *positive*(*negative*) degree of the vertex v in S. The *total degree* $d(v)$ of the vertex v in S is the sum $d(v) = d^+(v) + d^-(v)$.

A *cycle* in a signed graph S is said to be *positive* if the product of the signs of its edges is positive or, equivalently, if the number of negative edges in it is even. A cycle which is not positive is said to be *negative*. A signed graph is said to be *balanced* if every cycle in it is positive.

Theorem 1. [19] *A signed graph is balanced if and only if there exists a partition of its vertex set into two subsets, one of them possibly empty, such that every positive edge joins two vertices in the same subset and every negative edge joins two vertices from different subsets.*

Based on concept *balancing of sigraphs* and *characterization of common-edge sigraphs* a computer-oriented approach is obtained to detect whether a given common-edge sigraph is balanced or not.

Since, for a network use *nodes* in case of vertices and *link* for edges, so while giving algorithms for the characterization, nodes and links are used.

As an example, the nodes of S will be 1, 2, 3, 4, . . . , n and the node of H corresponding to the edge of S joining node 1 and node 2 will be called "1-2". A node of H will henceforth be a pair of numbers written in increasing order.

2. BALANCED COMMON-EDGE SIGRAPHS

The following result gives a characterization of sigraphs whose common-edge sigraphs $C_E(S)$ is balanced:

Theorem 2. [6] *For any sigraph S, $C_E(S)$ is balanced if and only if S is a balanced sigraph and*
(a) if $d(v_i) > 3$ then $d^-(vi) = 0$;
(b) if $d(v_i) = 3$ then $d^-(v_i) = 0$; or $d^-(v_i) = 2$; and
(c) for every x - y path $P_4 = (x, v,w, y)$ of length three, vw is a positive edge in S.

As an example, vertices of S will be 1, 2, 3.., n and edges will be in the form of adjacency matrix of order n x n with entries 1 for positive edge, 0 representing no edge and -1 for negative edge.
Following procedure is implemented to obtain a common-edge sigraph from a given sigraph and check whether this common-edge sigraph is balanced or not:

Enter the number of nodes i.e. n. Input n x n adjacency matrix with respect to given sigraph. The adjacency matrix takes the entries as 0, 1 and -1 for no edge, positive edge and negative edge respectively. To find the common-edge sigraph of this sigraph, find first non zero entry of the adjacency matrix, say (i, j^{th}) entry, then search for non zero entries in row i, column i and column j. For each such non zero entry, say (i, k^{th}) entry in i^{th} row, there corresponds a vertex in C_E(S), in this case edges (i, k) and (k, j) in S would be vertex of $C_E(S)$. Now sign of the vertex depends on the sign of edge (i, j) in S. If edge (i, j) is positive then corresponding vertex would be positive otherwise it would be negative. This way new matrix of $C_E(S)$ is computed.

To check whether a given common-edge sigraph is balanced or not, following 4 conditions need to be checked:
(i.) S is balanced or not
(ii.) if $d(v_i) > 3$ then $d^-(v_i) = 0$;
(iii.) if $d(v_i) = 3$ then $d^-(v_i) = 0$; or $d^-(v_i) = 2$;
(iv.) for every x - y path $P_4 = (x, v, w, y)$ of length three, vw is a positive edge in S.

If all the conditions are satisfied, then common-edge sigraph is balanced and will also print the common-edge sigraph of S.

To check (i.) condition, i.e. balancing of the sigraph, partition vertex set $V(S)$ into two subsets V_1 and V_2 (one of them possibly empty) such that every negative edge of S joins a vertex of V_1 with one of V_2 while no positive edge does so. Start with the first vertex say $i = 1$. Initially the first vertex is assigned the Group1. Now look all edges incident to it. Maintain two groups Group1 and Group2. Group1 contains all the vertices with positive edges and Group2 with negative edge. Now look in first row, if the (i, j^{th}) entry is -1, then j will be assigned Group2 else if 1 then Group1. Same procedure is repeated for all vertices. If any vertex belong to both the groups then the sigraph is not balanced else balanced.

To check (ii.) condition, count number of non-zero entry and number of -1 in each row for each vertex and check if the given condition is satisfied or not. If it satisfies the given condition then check (iii.) condition otherwise terminate the procedure and say $C_E(S)$ is not balanced.

To check (iii.) condition, check if the given condition is satisfied or not from the count already calculated in (ii.) condition. If it satisfies the given condition then check (iv.) condition otherwise terminate the procedure and say $C_E(S)$ is not balanced.

To check (iv.) condition, find every path x - v - w - y of length 3 in S, and check if (v, w) is positive or not. If it is positive

then the given condition is satisfied else say $C_E(S)$ is not balanced. If all the above 4 conditions are satisfied we can say that $C_E(S)$ is balanced.

3. ALGORITHM TO CONVERT A SIGRAPH TO COMMON-EDGE SIGRAPH

3.1 Part – 1:

Step1. Enter number of vertices i.e. *n* of sigraph *S* whose common-edge sigraph *H* is required.

Step2. Enter lower triangular (or upper triangular) part of adjacency matrix *vertex* [*i*] [*j*] and sign matrix *signver* [*i*] [*j*] of sigraph *S*.

Step3. Instead of 1 assign distinct numbers at all those positions of adjacency matrix where there is a non-zero entry.

Step4. Now search for non-zero entries in adjacency matrix.

(i.) For each such non-zero entry, say (i, j^{th}) entry, search for non-zero entries in row *i*, column *i* and column *j*. Now for each such nonzero entry, say (i, k^{th}) entry in row *i*, there corresponds a vertex in common-edge graph. In this case (i, j) − (i, k) will be a vertex in common edge graph *H*. Prepare a vertex matrix *comver* [*i*] [*j*] for common-edge graph.

(ii.) Now sign of this vertex depends on sign of edge (i, j) in *S*. If edge (i, j) is positive then corresponding vertex would be positive otherwise it would be negative. Prepare sign matrix *signver1* [*i*] [*j*] for vertices.

Step5. Instead of 1 assign distinct numbers at all those positions of vertex matrix where there is a non-zero entry.

Step6. Now search for non-zero entries in vertex matrix c*omver* [*i*] [*j*].

(i.) For each such non-zero entry, say (i, j^{th}) entry, search for non-zero entries in row *i*, column *i* and column *j*. Now for each such nonzero entry, say (i, k^{th}) entry in row *i*, there corresponds an edge in $C_E(G)$.

(ii.) Now sign of this edge depends on sign of common vertex. In this case if vertex *i* is negative then edge would be negative and if vertex *i* is positive then edge is also positive.

Step7. Thus the sigraph so produced is required common-edge sigraph *H* of *S*.

Complexity of computation involved in above algorithm

In Step3, we have to assign distinct numbers to all the non zero entries in adjacency matrix. Since we have entered lower triangular matrix, thus we need to check $n(n\text{-}1)/2$ entries.

Hence complexity for this step is $O(n^2)$.

Then in Step4, first we have to search for non zero entries in adjacency matrix and then corresponding to each such entry, say (i, j^{th}) entry, we have to search for non zero entries in row *i*, column *i* and column *j*. Also we have to check signs of these entries.

Thus complexity of this step = $O(n^2 \times n) = O(n^3)$.

In Step5, we assign distinct numbers to non zero entries in vertex matrix with complexity $O(n^2)$. In Step6 we repeat the procedure of Step4, thus complexity of this step would be $O(n^3)$.

Thus complexity of computation involved in above algorithm is $O(n^3)$, where *n* is number of vertices in *S*.

3.2 Part – 2:

Step1. Enter the number of vertices *n* of input sigraph *S* whose common-edge sigraph $C_E(S)$ is required.

Step2. Enter the *n* x *n* adjacency matrix i.e. *vertex* [*i*] [*j*] with enteries 0, 1 and -1 of sigraph *S* where *i* = 1 to *n* and *j* = 1 to *n*.

Step3. Set EdgeIndexCtr = 0

Step4. Repeat Step4 to Step11 for *i* = 1 to *n*

Step5. Repeat Step5 to Step11 for *j* = *i*+1 to *n*

Step6. Check if (*vertex* [*i*] [*j*] != 0) // There is an edge
If yes, find an adjacent edge

Step7. Repeat Step7 to Step9 for *k* = *j*+1 to *n*

Step8. Check if (*vertex* [*j*] [*k*] != 0), if yes,
Set CEVertex &CurVertex = CeVertices[CurCEVertexIndex]
CurVertex.e1[0] = i
CurVertex.e1[1] = j
CurVertex.e2[0] = j
CurVertex.e2[1] = k
CurVertex.Index = CurCEVertexIndex++

Step9. Check if (*vertex*[*i*][*k*] != 0), if yes,
Set CEVertex &CurVertex = CeVertices[CurCEVertexIndex]
CurVertex.e1[0] = i
CurVertex.e1[1] = j
CurVertex.e2[0] = i
CurVertex.e2[1] = k
CurVertex.Index = CurCEVertexIndex++

Step10. Repeat Step11 for *k* = *i*+1 to *j*

Step11. Check if (*vertex* [*j*] [*k*] != 0), if yes,
Set CEVertex &CurVertex = CeVertices[CurCEVertexIndex]
CurVertex.e1[0] = i
CurVertex.e1[1] = j
CurVertex.e2[0] = k
CurVertex.e2[1] = j
CurVertex.index = CurCEVertexIndex++

Step12. // Create common-edge graph
Repeat Step 12 to Step17 for *i* = 1 to CurCEVertexIndex

Step13. Repeat Step 13 to Step17 for *j* = *i*+1 to CurCEVertexIndex

Step14. Set CEVertex &c1 = CeVertices[i]
&c2 = CeVertices[j]

Step15. Set sign = 0

Step16. // Find if there is a common edge
Check if (c1.e1[0] == c2.e1[0] &&c1.e1[1] == c2.e1[1]) // c1e1 == c2e1

Set sign = vertex[c1.e1[0]][c1.e1[1]]; // Sign of common edge
else check if (c1.e1[0] == c2.e2[0] && c1.e1[1] == c2.e2[1])
Set sign = vertex[c1.e1[0]][c1.e1[1]];
else check if (c1.e2[0] == c2.e1[0] && c1.e2[1] == c2.e1[1])
Set sign = vertex[c1.e2[0]][c1.e2[1]];
else check if (c1.e2[0] == c2.e2[0] && c1.e2[1] == c2.e2[1])
Set sign = vertex[c1.e2[0]][c1.e2[1]];
CeGraph[i][j] = sign; CeGraph[j][i] = sign;

Step17. Set CeGraph[i][j] = sign;
CeGraph[j][i] = sign;

Step18. Print CeGraph i.e common-edge sigraph.

Complexity of computation involved in above algorithm

In Step2, we have entered n x n matrix, thus complexity for this step is $O(n^2)$.

Then in Step4 first we have to search for non zero entries in adjacency matrix and then corresponding to each such entry, say (i, j^{th}) entry, we have to search for non zero entries in row i, column i and column j as in Step5. Then, for each such non zero entry, say (i, k^{th}) entry in i^{th} row,we have to find vertex in $C_E(S)$ as in Step10. That way common-edge graph is computed and for sign of the edge we need to again traverse the matrix as in Step16.

Thus complexity of this step = $O(n^3 \text{ x } n) = O(n^4)$.

Hence complexity of computation involved in above algorithm is $O(n^4)$, where n is number of vertices in S.

3.3 Conclusion

Thus, common-edge sigraph from a given sigraph can be implemented in two ways. First part takes input in the form of two matrices and has complexity $O(n^3)$ whereas second part takes only one matrix as input and with complexity $O(n^4)$. There is a trade-off between space and time complexity. Depending on nature of the problem, implementation of the above algorithm can be done in any of the way.

4. ALGORITHM TO DETECT BALANCING OF A SIGRAPH

Harary and Kabell [19] have proposed an algorithm to detect balance in signed graphs in $O(n)$ steps. But since matrix is used as an input to the signed graph, code named "BALANCE" is defined, to check balancing of a given sigraph, where vertices can also be termed as nodes from algorithmic point of view in $O(n^2)$ steps:

Step1. Enter the number of vertices i.e. n of sigraph S.

Step2. Enter the n x n adjacency matrix i.e. *vertex* [i] [j] with enteries 0, 1 and -1 of sigraph S where i = 1 to n and j = 1 to n.

Step3. Print the given sigraph.

Step4. Set IsBalanced = true.

Step5. Repeat Step5 for i = 1 to n && IsBalanced
5.1 if (!IsVisited [i])
5.1.1 Set IsVisited[i] = true
IsGroup[i] = true
Push back i and Group1
5.2 While ((!queue.empty()) && IsBalanced)
5.2.1 Set CurGroup = queue.front() and
Set g = Cur.second
5.2.2 Repeat Step5.2.2 for j = 1 to n && IsBalanced
5.2.2.1 Check if vertex [Cur.first [j] == 1)
If No, Goto 5.2.2.2
If yes, check if (g.Othergroup !
Curgroup[j] == true)
If yes, Set IsBalanced = false and Print
j is in both groups
else check if (!IsVisited [j])
Set g.Othergroup ! Curgroup[j] = true
IsVisited[j] = true
Push back j and Othergroup
5.2.2.2 Check if vertex [Cur.first [j] == -1)
If No, Goto Step6
If yes, check if (g.Curgroup[j]== true)
If yes, Set IsBalanced = false and
Print j is in both groups
else check if (!IsVisited [j])
Set (g.Othergroup!Curgroup[j] = true)
IsVisited[j] = true
Push back j and Othergroup

Step6. If (IsBalanced) Print "Sigraph is balanced"
else Print "Sigraph not balanced"

Step7. Exit

Complexity of computation

In Step2, we have to assign distinct numbers to all the non-zero entries in adjacency matrix. Since we have entered n x n matrix, thus we need to check n^2 entries.

Hence complexity for this step is $O(n^2)$.

In Step5 we visit each vertex and for each vertex we find adjacent edges incident to it and groups are assigned depending on whether the entry is positive or negative assign group to each non-zero entry. Thus 5.2.1 and 5.2.2 are repeated till queue is not empty. Since there are n2 entries,

Complexity for this step is $O(n^2)$.

Thus, Total complexity involved = $O(n^2) + O(n^2) = O(n^2)$.

Hence complexity of computation involved in above algorithm is $O(n^2)$ which is optimal in nature, where n is number of vertices in S.

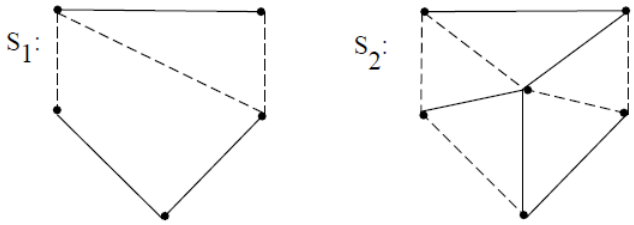


Figure 2: S_1 is balanced and S_2 is unbalanced

5. ALGORITHM TO DETECT BALANCING IN COMMON-EDGE SIGRAPHS

The algorithm to detect balancing in common-edge sigraph is based on the characterization given by Acharya & Sinha[6].
It uses following functions:
MAX denotes maximum number of vertices
n = number of vertices
struct CEVertex // Represents 2 edges
int e1[2] // Edge 1
e2[2] // Edge 2
index - Index in new graph
Step1. Enter the number of vertices i.e. *n* of sigraph *S*.

Step2. Enter the *n* x *n* adjacency matrix i.e. *vertex* [i] [j] with enteries 0, 1 and -1 of sigraph *S* where *i* = 1 to *n* and *j* = 1 to *n*.

Step3. Implement algorithm defined in *Section 3* to obtain $C_E(S)$.

Step4. Implement algorithm defined in *Section 4* to check whether *S* is balanced. // First condition check

Step5. If *S* is balanced, Goto Step6 else Goto Step18.

Step6. // Checking degree requirement
Repeat Step6 to Step16 for *v* = 1 to *n* && IsCommonbal

Step7. Set NbPositiveEdge = 0
NbNegativeEdge = 0
NbTotalEdge = 0

Step8. Repeat Step8 to Step9 for *j* = 1 to *n*

Step9. if (*vertex* [i] [j] == 1) NbPositiveEdge++
if (*vertex* [*i*] [*j*] == -1) NbNegativeEdge++
if ((*vertex* [*i*] [*j*] == 1) || (*vertex* [*i*] [*j*] == -1)) NbTotalEdge++

Step10. if (NbtotalEdge > 3), if yes,
check if(NbNegativeEdge !=0), if yes,
Print "Degree Requirement not satisfied, i.e condition 'a' fails"
Print "Hence, $C_E(S)$ is not balanced" and Goto Step25

Step11. if (NbTotalEdge == 3), if yes
check if((NbNegativeEdge != 0) || (NbNegativeEdge != 2)),
if yes,
Print "Degree requirement not satisfied, i.e condition 'b' fails"
Print "Hence, $C_E(S)$ is not balanced" and Goto Step25

Step12. //check condition 'c' with complexity $O(n^4)$
if (NbTotalEdge ≥ 3) , if yes,
Print "Vertex where degree is greater than equal to 3 = " *v*

Step13. Repeat Step13 to Step16 for *x* = 1 to *n*

Step14. Repeat Step14 to Step16 for *w* = 1 to *n*

Step15. Repeat Step15 to Step16 for *y* = 1 to *n*

Step16. Check
if((vertex[x][v]!=0)&&(vertex[y][w]!=0)&&(vertex[v][w]==-1)&&(v!=y)&&(y!=x)&&(if yes, print "Third condition does not satify at " x - v - w – y
"Therefore, Common-edge Sigraph is not balanced"
Set isCommonbal=false

Step17. //check condition 'c' with complexity $O(n^3)$

Step18. Repeat Step18 to Step23 for v = 1 to n.

Step19. Repeat Step19 to Step23 for w = 1 to n.

Step20. Check if (vertex[w][v] ==1) , If yes, Goto Step21

Step21. Repeat Step21 for x = 1 to n.

Check if (vertex[x][v] ==-1), If yes, Set isedgewithv = 1

Step22. Repeat Step22 for y = 1 to n.
Check if (vertex[y][w] ==-1), If yes, Set isedgewithw = 1

Step23. Check if ((isedgewithv!=0) && (isedgewithw!=0)),
If yes, print "Third condition does not satify at " x - v - w - y
"Therefore, Common-edge Sigraph is not balanced"
Set isCommonbal=false

Step24. Print "$C_E(S)$ is balanced"

Step25. Exit

Complexity of computation involved in above algorithm
In Step2, since we input a graph of order *n* x *n*, complexity of this step = $O(n^2)$.
In Step3, we compute $C_E(S)$ as defined in *Section 3*, therefore,
Complexity of this step = $O(n^4)$ or $O(n^3)$.
In Step4, we check whether *S* is balanced or not as defined in *Section 4*, therefore,
Complexity of this step = $O(n^2)$.

In Step7 and Step9, since we have to traverse each node of the graph and calculate number of positive, negative and total edges of the graph and for this we traverse each row and each column,

Thus complexity of this step = $O(n^2)$.

In Step10, we check 'a' condition for each vertex, therefore, Complexity of this step = $O(n)$.

In Step11, we check 'b' condition for each vertex, therefore, Complexity of this step = $O(n)$.

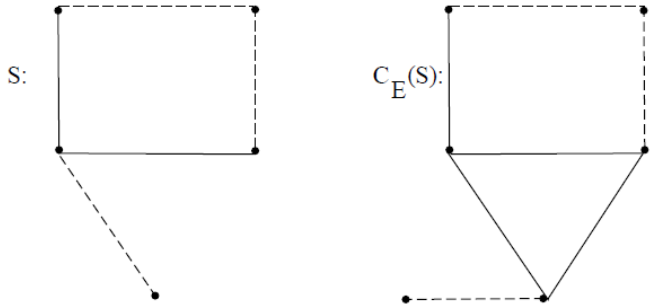


Figure 3: A Sigraph *S* such that $C_E(S)$ is balanced

In Step12, we check condition 'c', for this we have to traverse the graph and find path of length 3, i.e. Step13, Step14 and Step15 are repeated '*n*' times and for each vertex, Complexity of this step = $O(n^3) \times n = O(n^4)$.

Further, to improve the complexity in condition 'c', we implement Step17 to Step23, where we find first positive edge *vw*. Then we find adjacent edges from vertex *v* and *w* and if there is any edge adjacent we set value equal to 1.

Complexity of this step = $O(n^2) \times n = O(n^3)$.

Total maximum complexity = $O(n^2) + O(n^4) + O(n^2) + O(n^2) + O(n) + O(n) + O(n^4) = O(n^4)$.

Optimal complexity = $O(n^2) + O(n^3) + O(n^2) + O(n^2) + O(n) + O(n) + O(n^3) = O(n^3)$.

Hence, complexity of computation involved in above algorithm is $O(n^3)$, where *n* is number of vertices in *S*.

6. CONCLUSION AND SCOPE

In this paper, an algorithmic approach is depicted to convert signed graph *S* into common-edge sigraph $C_E(S)$ and detect whether it is balanced or not in $O(n^4)$ steps. Further complexity is reduced to $O(n^3)$ steps. This way, algorithm optimality is proved. We can extend our algorithm to detect common-edge sigraph and output its root common-edge sigraph. Also, algorithm can be developed to obtain iterated common-edge sigraph and *S*-consistent common-edge sigraphs. These problems are important, at least from the socio-psychological point of view that they might help understand how interacting dyads in a social network tend to change the social structure in a way prescribed by the structure of common-edge sigraph of the initial social network; structural evolution of social networks has been a topic of current research interest (e.g., see, Holland & Leinhardt [20]; Acharya [1]; Acharya & Acharya [3, 4]; Doreian [14, 15, 16]; Kovchegov [24, 25, 26]).

7. ACKNOWLEDGMENTS

The authors express gratitude to Mr. Dhananjay Kulkarni who was always there in prior discussion and helping in writing algorithms and finding complexity and to the referees who made extensive and constructively critical comments on the first version of the paper.

8. REFERENCES

[1] Acharya B .D, 1980, (1980b) Applications of sigraphs in behavioral sciences, MRI Technical Report DST/HCS..

[2] Acharya B. D, 1983, A characterization of consistent marked graphs, Nat. Acad. Sci. -Letters, India.

[3] Acharya, B. D and Acharya, M, 1983, A graph-theoretical model for the analysis of intergroup stability in a social system, Manuscript. In: A mathematical bibliography of signed and gain graphs and allied areas, VII Edition.

[4] Acharya, B. D and Acharya, M, 1986, New algebraic models of a social system, Indian Journal of Pure and Applied Mathematics.

[5] Acharya, M and Sinha, D, 2005, Characterizations of line sigraphs, Nat. Acad. Sci. –Letters.

[6] Acharya, M and Sinha, D, 2006, Common-edge sigraphs, AKCE Int. J. Graphs Comb.

[7] Balbuena, C, Garcia-Vazquez, P. 2004, Edge-connectivity in P_k - path graphs, Discrete Mathematics.

[8] Behzad, M. and Chartrand, G. T , 1969, Line coloring of signed graphs, Elem. Math.

[9] Broersma, H. J., HOEDE, C., 1989, Path graphs, Journal of Graph Theory.

[10] Cartwright, D. and Harary, F., 1956, Structural Balance: A generalization of Heider's Theory, Psych. Rev.

[11] Chartrand, G. T., 1977, Graphs as Mathematical Models, Prindle, Weber and Schmidt, Inc., Boston, Massachusetts.

[12] Cormen, Thomas, Leiserson, Charles., Rivest, Ronald., Stein, Clifford., 2011, Introduction to algorithm, Third Edition, PHI Learning Private Limited.

[13] Deo, Narasing., 1995, Graph theory with application to Engineering and Computer Science, Prentice Hall India.

[14] Doreian, P., (1979/80), On the evolution of group and network structure, Social Networks.

[15] Doreian, P., and Mrvar, A., 1996, A partitioning approach in structural balance, Social Networks.

[16] Doreian, P., 2002, Event sequences as generators of social network evolution, Social Networks.

[17] Harary, F., Norman, R.Z., Cartwright, R. W., 1965, Structural Models: An Introduction to the Theory of Directed Graphs, Wiley Inter Science, Inc., New York.

[18] Harary, F., 1969, Graph Theory, Addison-Wesley Publ. Comp., Reading, Massachusetts.

[19] Harary, F., and Kabell, J. A., 1980/81, A simple algorithm to detect balance in signed graphs, Math. Soc. Sci.

[20] Holland, L. W., and Leinhardt, S., 1977, Social structure as a network process, Zeitschrift f¨ur Soziologi´e.

[21] Horowitz, Ellis., Sahni, Sartaj., 2004, Computer Algorithm, Galgotia Publications, 2004 Edition.

[22] Kanetkar, Yashavant., 2004, "Let Us C", Fifth Edition, BPB Publications.

[23] Kanetkar, Yashavant., 2008, "Graphics under C", Fifth Edition, BPB publications.

[24] Kovchegov, V.B., 1993, A model of dynamics of group structure of human institutions, Journal of Mathematical Sociology.

[25] Kovchegov, V.B., 1994, A principle of nonergodicity for modeling of the human groups by nets of probability automata, Proceeding of the 14th IMACS World Conference on Computational and Applied Mathematics.

[26] Kovchegov, V.B., 2004, Application of the theory of locally interacting and product potential networks of automata to modelling balance in social groups, Preprint.[27] Li, H., Lin, Y., 1993, On the characterization of path graphs, Journal of Graph Theory.

[28] Li, X., Zhao, B., 2004, Isomorphisms of P_k – graphs for $k \geq 4$, Discrete Mathematics.

[29] Lehot, P.G.H., 1974, An optimal algorithm to detect a line graph and output its root graph, Journal of the Association for Computing Machinery.

[30] Sinha, D., 2005, New frontiers in the theory of signed graph, Ph.D. Thesis, University of Delhi (Faculty of Technology).

[31] Sinha, D., Upadhayaya, S., Kataria, P., 2013, Characterization of Common edge signed graphs, Applied Discrete Mathematics.

[32] Kulli, V.R., 1973, On common-edge graphs, The Karnatak University Journal: Science XVIII.

[33] West, D.B., 1996, Introduction to Graph Theory, Prentice-Hall of India Pvt. Ltd.

[34] Zasalavsky, T., 1981, Characterizations of signed graphs, J. Graph Theory.

[35] Zasalavsky, T., 1982, Signed graphs, Discrete Appl. Math