IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Volume-5, Issue-9 E-ISSN: 2347-2693

A novel Approach to Compute Steiner point in Graph: Application for Network Design

M.B.Chandak[1*], S.Bhalotia[2], S.C.Agrawal[3]

[1*]Dept. of CSE, Shri Ramdeobaba College of Engineering and Management, Nagpur, India
[2]Dept. of CSE, Shri Ramdeobaba College of Engineering and Management, Nagpur, India
[3]Dept. of CSE, Shri Ramdeobaba College of Engineering and Management, Nagpur, India

[*]*Corresponding Author: hodcs@rknec.edu, Tel.: +91-0712-2580011[Ext.3200]*

Available online at: www.ijcseonline.org

Received: 09/Aug/2017, Revised: 21/Aug/2017, Accepted: 14/Sep/2017, Published: 30/Sep/2017

***Abstract*—** A Graph has two main components: Vertices and Edges. The vertices are connected using edges. There are two types of graphs, directed and undirected. The major application of graph is representing network on paper. The cost involved in converting paper based network to actual cable based network is majorly controlled by cables required for connection. The cost can be reduced if the length of cable can be reduced. The paper describes the methodology to compute Steiner point. Using Steiner point, it is possible to modify the position of vertices, so as to reduce the cable length, keeping the vertex connectivity intact. The paper describes implementation of Steiner point on graph with number of vertices as 3, 4, 5 and 6. The presented work can be extended for graph with any number of vertices. It is an optimization approach to reduce the cable size and cost of network implementation.

***Keywords*—** Graph, Network, Cable length, Steiner point, data structures

I. INTRODUCTION

A. What is Graph?

A *Graph* G=(V,E) is a structure consisting of set of vertices V={v1,v2,v3,...,} and set of edges E={e1,e2,e3,...,} where each edge connect a pair of vertices.[1,2]

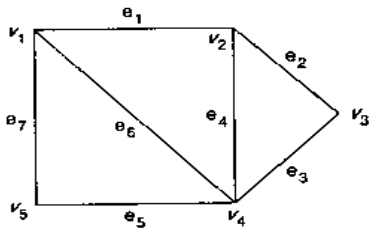

Figure 1. Example of a graph.

In Figure 1, we have V={v1,v2,v3,v4,v5} and E={e1,e2,e3,e4,e5,e6,e7}. Edge e1 connects vertices v1 and v2. Vertices v1 and v2 are *adjacent* to each other as they are connected by a same edge.

There are 2 types of graph:-[1,2]

1. Undirected- An undirected graph is a graph in which edges have no orientation. The edge (x, y) is identical to the edge (y, x), i.e., they are not ordered pairs, but sets {x, y} (or 2-multisets) of vertices.

2. *Directed*- A directed graph or digraph is a graph in which edges have orientations. An arrow (x, y) is considered to be directed from x to y; y is called the head and x is called the tail of the arrow; y is said to be a direct successor of x and x is said to be a direct predecessor of y.

B. Graph as a Data Structure:

In computer science, a *graph* is an abstract data type that is meant to implement the undirected graph and directed graph concepts from mathematics. A graph data structure consists of a finite (and possible mutable) set of vertices or nodes or points, together with a set of unordered pairs of these vertices for an undirected graph or a set of ordered pairs for a directed graph. These pairs are known as edges, arcs, or lines for an undirected graph and as arrows, directed edges, directed arcs, or directed lines for a directed graph. The vertices may be part of the graph structure, or may be external entities represented by integer indices or reference [3,4]. A graph data structure may also associate to each edge some edge value, such as a symbolic label or a numeric attribute (cost, capacity, length, etc.).Many operations can be performed on graphs like adding an edge, removing a vertex, checking if 2 vertices are adjacent, etc.

C. Graph Memory Representation:

There are 2 popular ways to represent graphs.

1. *Adjacency Matrix*- For a simple graph with vertex set V, the adjacency matrix is a square $|V| \times |V|$ matrix A such that its element A_{ij} is one when there is an edge from vertex i to vertex j, and zero when there is no edge. The diagonal elements of the matrix are all zero, since edges from a vertex to itself (loops) are not allowed in simple graphs.
Some graphs can be weighted i.e. they have cost associated with each edge. The cost can be anything depending on the problem and approach user is applying. These graphs are represented by a cost matrix which is similar to adjacency matrix but instead of 0's and 1's, it stores the respective cost of that connection.

2. *Adjacency List*- An adjacency list representation for a graph associates each vertex in the graph with the collection of its neighbouring vertices or edges. Cormen et al. suggest an implementation in which the vertices are represented by index numbers. Their representation uses an array indexed by vertex number, in which the array cell for each vertex points to a singly linked list of the neighbouring vertices of that vertex. In this representation, the nodes of the singly linked list may be interpreted as edge objects; however, they do not store the full information about each edge (they only store one of the two endpoints of the edge) and in undirected graphs there will be two different linked list nodes for each edge (one within the lists for each of the two endpoints of the edge).

D. Graph Applications:

Graphs are used to model many situations of reality, and tasks on graphs model multiple real problems that often need to be resolved. We will give just a few examples:

- *Map of a city* can be modelled by a *weighted oriented graph*. On each street, edge is compared with a length, corresponding to the length of the street, and direction – the direction of movement. If the street is a two-way, it can be compared to two edges in both directions. At each intersection there is a node. In such a model there are natural tasks such as searching for the shortest path between two intersections, checking whether there is a road between two intersections, checking for a loop (if we can turn and go back to the starting position) searching for a path with a minimum number of turns, etc.[3,4]

- *Computer network* can be modelled by an *undirected graph*, whose vertices correspond to the computers in the network, and the edges correspond to the communication channels between the computers. To the edges different numbers can be compared, such as channel capacity or speed of the exchange, etc. Typical tasks for such models of a network are checking for connectivity between two computers, checking for double-connectivity between two points (existence of double-secured channel, which remains active after the failure of any computer), finding a *minimal spanning tree (MST)*, etc. In particular, the Internet can be modelled as a graph, in which are solved problems for routing packets, which are modelled as classical graph problems.[5]

E. Network Representation Using Graphs:

Examples of networks are: an arrangement of intersecting lines, or a group or system of interconnected people or things [1, 2]; a system of computers connected by communications lines (cables); a group of connected radio or television stations; *a network of roads, etc.[6,7]*

Representing a problem as a graph can make a problem much simpler. More accurately, converting a network into graph can provide the appropriate tools for solving the problem.
There are two components to a graph (G= (V,E)):
Nodes and edges

- In graph-like problems, these components have natural correspondences to problem elements
- Entities are nodes and interactions between entities are edges, and the property for which the problem is considered, is taken as the cost of the edges (e.g. distance, traffic, etc.).

Figure 2.1. A graph of connected cities.

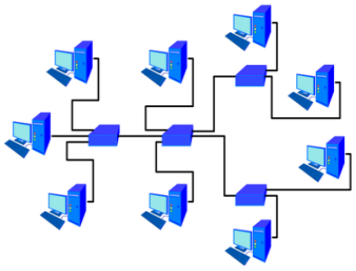

Figure 2.2. A graph of connected computers.

It is common to identify vertices not by name (such as "Audrey," "Boston," or "sweater") but instead by a number. That is, we typically number the |V| |V| |V| vertical bar, V, vertical bar vertices from 0 to |V|−1 |V|-1 |V|−1vertical bar, V, vertical bar, minus, 1[8]. Here is the social network graph with its 10 vertices identified by numbers rather than names:

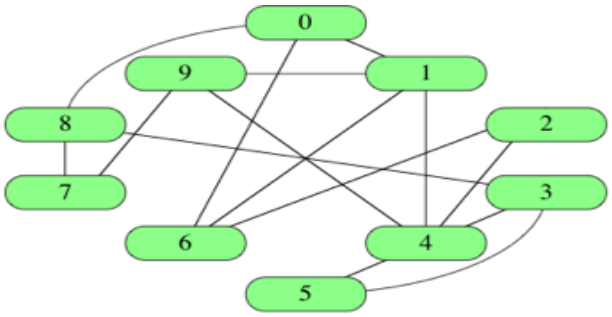

Figure 3.1. A social network graph.

	0	1	2	3	4	5	6	7	8	9
0	0	1	0	0	0	0	1	0	1	0
1	1	0	0	0	1	0	1	0	0	1
2	0	0	0	0	1	0	1	0	0	0
3	0	0	0	0	1	1	0	0	1	0
4	0	1	1	1	0	1	0	0	0	1
5	0	0	0	1	1	0	0	0	0	0
6	1	1	1	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	1	1
8	1	0	0	1	0	0	0	1	0	0
9	0	1	0	0	1	0	0	1	0	0

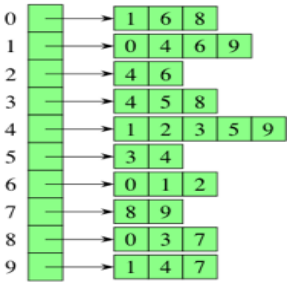

Figure 3.2. Adjacency Matrix and List Representation.

F. Cost Benefit Analysis of Network:

The aim of a cost benefit analysis is to come up with a solution with an optimum cost. A network may consist of various parameters. The parameter to be optimised can be considered as the cost for the edges of the graph (for that network).
Consider a network of five locations with the referenced parameter as the distance of travel from one node to another. The graphical structure for it will be like:

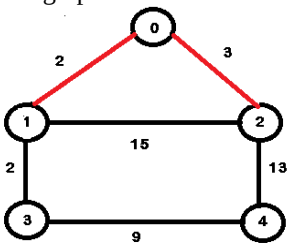

Figure 4.1. A 5-node network.

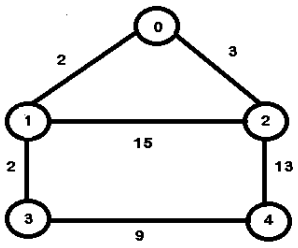

Figure 4.2. Optimum Path.

	0	1	2	3	4
0	0	2	3	0	0
1	2	0	15	2	0
2	3	15	0	0	13
3	0	2	0	0	9
4	0	0	13	9	0

Figure 4.3. Adjacency Matrix Representation

For this network, if we want to find the optimum distance of travel from node 1 to node 2, we need to perform a cost benefit analysis for this network [9,10,11].
Considering different possible paths:

a) 1-> 2 : total cost = 15
b) 1-> 0-> 2 : total cost = 2+3 = 5
c) 1-> 3-> 4-> 2 : total cost = 2+9+13 = 24

Clearly, the optimum path would be (b).
If we can by some means, find another path with a lesser cost than that of (b), that path would then be the optimum path.

G. Cable Length Estimation Method:

One of the important networks is the Cable Network, and reducing the length of the cables for the network is a matter of concern when they span across large areas.
Consider that 4 buildings are to be connected together through cables. Taking distance (in kms) between buildings as cost for the edges, it can be represented as:

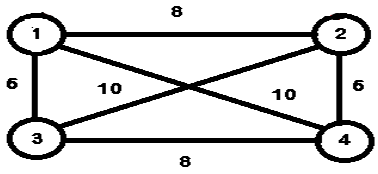

Figure 5.1. A network of 4 buildings.

Total cable length will simply be the sum of all the edges.
For e.g. Cost in this graph = 8+8+6+6+10+10 = 48 km
Similarly, there can be other possible combinations of the edges, and the cable length can be computed accordingly.

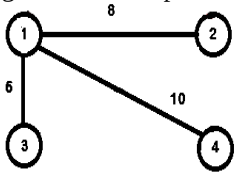

Figure 5.2. Combination (1) of graph edges.

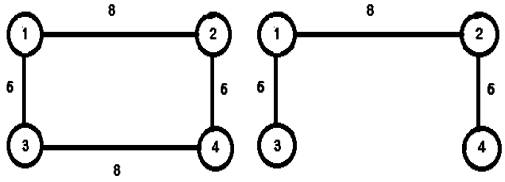

Figure 5.3. Combination (2) and (3) of graph edges.

H. Need For Graph Minimization:

Suppose there are 4 cities A, B, C and D as in Figure 6. The total cost of this graph is 250 km. Now the minimum distance between these cities can be found if we remove the edge with maximum cost (Figure 7). The total cost is now 150 km which is still high. For 4 cities, these numbers seem

very less but when we take a bigger real life problem of 50 cities over few hundred kilometres, then it is very important to minimise the distances to save resources [12,13].

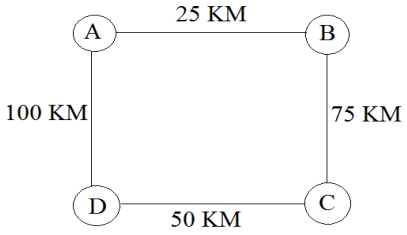

Figure 6. Graph of 4 cities.

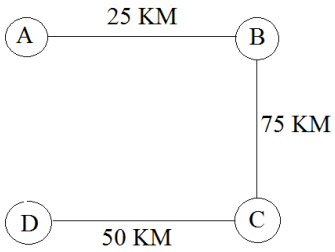

Figure 7. Graph after removing the costliest edge.

***I. Steiner Point Definition*:**

An additional point that reduces the length of the spanning tree is called a Steiner point. The resulting tree is called a Steiner tree. A minimal Steiner tree minimizes total edge length. The minimum Steiner tree is the best possible Steiner tree. A Steiner point cannot be an endpoint; else we could simply delete it and its associated edge, reducing the length of the spanning tree. By the triangular inequality, a Steiner point cannot have degree two. For Figure 6, there will be 2 Steiner points S1 and S2 and joining them with edges will generate a Steiner tree (Figure 8).

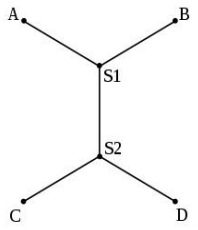

Figure 8. Graph with Steiner points.

Any Steiner Point has 3 properties:-
1. Each Steiner point has 3 edges and each angle is 120 degrees.
2. The no. of Steiner point in tree is 2 less than the total no. of vertices.
3. The total no. of edges in a Steiner tree is 1 less than total no. of vertices summed with total no. of Steiner points [12].

II. METHOD TO COMPUTE STEINER POINT:

Different figures have different properties and methods to calculate Steiner point. We will check each one independently.

There are 3 methods to compute a *Steiner point* of triangle.

1. **Torricelli method**: For a triangle with vertices V1, V2 and V3, construct an equilateral triangle from each edge of triangle V1V2V3. Then construct 3 circles circumscribing each equilateral triangle. The point of intersection of these circles is the Steiner point of that triangle.

2. **Simpson method**: For a triangle with vertices V1, V2 and V3, construct an equilateral triangle from each edge of triangle V1V2V3. Then join the vertex of equilateral triangle not in V1V2V3 to opposite vertex. The point of intersection of these lines is the Steiner point of that triangle.

3. **3-Point Algorithm**: This is combination of Torricelli and Simpson methods. In this method, we select any one edge of that triangle and draw equilateral triangle from connecting vertices. Then draw circle circumscribing the equilateral triangle. Also connect the vertex of equilateral triangle not in original triangle to opposite vertex. The point of intersection of this line and circle is Steiner Point.

For calculating the Steiner Point of rectangle, we need to take a point on mid-point on the smaller edge. Connect it to the mid-point of opposite edge. Then by using property of Steiner point that it needs to have angle 120, we can calculate exactly at what angle the vertices will intersect that line. These intersection points will be Steiner points for that rectangle. Pentagon and hexagon can be resolved by visualising it as a combination of triangle(s) and rectangle.

III. ALGORITHM

algo Steiner() {

// Input the shape (triangle, rectangle, pentagon, hexagon) for which Steiner points are to be computed.

```
if (shape==triangle)
   call triangle_func();     //for finding Steiner point of a
   triangle
else if(shape==rectangle)
   call rectangle_func();    //for finding Steiner point of a
   rectangle
else if(shape==pentagon)
```



```
    call pentagon_func();     //for finding Steiner point of a
    pentagon
else if(shape==hexagon)
    call hexagon_func();      //for finding Steiner point of a
    hexagon
else
    print "wrong input";
}
```

function triangle_func() {

```
  // Input the three vertices of the triangle from the user. Let
  them be P1, P2 & P3.
ST1 = create_triangle_point(P1,P2,P3);   // To calculate the
Steiner point.
  // ST1 is the Steiner point of the triangle, returned by the
  function.
print ST1;
  // Draw appropriate GUI to illustrate all the vertices
  (including the Steiner point) and their edges.
}
```

function rectangle_func() {

```
  // Input the four vertices of the rectangle from the user. Let
  them be P1, P2, P3 & P4.
ST1,ST2 = create_rectangle_Steiner(P1,P2,P3,P4);     // To
calculate the Steiner points.
  // ST1 and ST2 are the Steiner points of the rectangle,
  returned by the function.
print ST1,ST2;
  // Draw appropriate GUI to illustrate all the vertices
  (including the Steiner points) and their edges.
}
```

function pentagon_func() {

```
  // Input all the five vertices of the pentagon from the user.
  Let them be P1, P2, P3, P4 & P5.
  // The vertices of the pentagon is divided into two parts: a
  rectangle and a triangle.
ST1 = create_triangle_point(P2,P4,P3);
  // for calculating Steiner point of the triangle part.
ST2, ST3 = create_rectangle_Steiner(P1,P5,P4,P2);    // For
calculating Steiner point of the  rectangle part.
  // ST1 is the Steiner point of the triangle part(P2,P4,P3) of
  the pentagon.
  // ST2 and ST3 are the Steiner points of the rectangle part
  (P1, P5, P4, P2) of the pentagon.
print ST1,ST2,ST3;
  // Draw appropriate GUI to illustrate all the vertices
  (including the Steiner points) and their edges.
}
```

function hexagon_func() {

```
  // Input all the six vertices of the hexagon from the user. Let
  them be P1, P2, P3, P4, P5 & P6.
  // The vertices of the hexagon is divided into three parts: a
  rectangle,a lower triangle and an //upper triangle.
ST1 = create_triangle_point(P1,P3,P2);     //upper triangle
  ST2,ST3 = create_rectangle_Steiner(P6,P4,P3,P1);
  //middle rectangle
ST4 = create_triangle_point(P6,P4,P5);          //lower triangle
  // ST1 is the Steiner point of the upper triangle
  part(P1,P3,P2) of the hexagon.
  // ST2 and ST3 are the Steiner points of the rectangle part
  (P6,P4,P3,P1) of the hexagon.
  // ST4 is the Steiner point of the lower triangle part
  (P6,P4,P5) of the hexagon.
print ST1,ST2,ST3,ST4;
  // draw appropriate GUI to illustrate all the vertices
  (including the Steiner points) and their edges.
}
```

function create_triangle_point(P2,P4,P3){

```
  //Consider T1 to be the triangle created using P2,P4 & P3.
  //Property of a Steiner Point - None of the interior angles of
  the triangle should be greater than //120 degrees, or else it
  will the minimum distance condition itself, and no Steiner
  point will be needed.
  //Calculate all angles of the triangle. If any angle is greater
  than 120 degrees, print "error" and exit!
  // Let P2[0] be the x-coordinate, and P2[1] be the y-
  coordinate of the Point P2. Same for other points.
  //Applying formula to compute the third point of the
  equilateral triangle its two vertices as P2 & P4 (on the
  opposite side to that of P3)
x = cos60*(P4[0]-P2[0]) - sin60*(P4[1]-P2[1]))
+ P2[0] ;
if (P3[1] > P2[1])
    y= -sin60*(P4[0]-P2[0]) - cos60*(P4[1]-P2[1])) + P2[1];
else
    y= sin60*(P4[0]-P2[0]) - cos60*(P4[1]-P2[1])) + P2[1];

  // Let P7 be the point with x and y as co-ordinates, and T2
  be the equilateral triangle formed by // points (P2,P4,P7).
  // Calculate Centroid of the triangle T2.  Then do the
  following:
S1 = Segment formed by (Centroid & P7);
r= Length (S1);
  // Form a Circle C1 with center as (Centroid), and radius as
  r.  Then do the following:
S2= Segment formed by (P3 & P7);
a1= Intersection of C1 and S1 ;
  // a1 will be a 2x2 float array as their will be two points of
  intersection (Out of the two, one will //be the vertex P7
  which we have to ignore.  Choose the other point. This
  point is the Steiner //point of the triangle (P2,P4,P3).
```



```
x= a1[0,0];   // x-coordinate of the intersection
y= a1[0,1];   // y-coordinate of the intersection
P8=Point(x,y);
  // P8 is the Steiner point and is returned.
return p8;
}
```

function create_rectangle_Steiner(P1,P2,P3,P4) {

```
  // consider lengths of all segments, and choose the larger
  one as length 'l'.
k= (P3[1]+P1[1])/2;       // to find mid-point of the breadth of
the rectangle
  // using a property of Steiner points to compute them. The
  property is that a Steiner point can be maximum connected
  to three vertices, and it subtends an angle of 120 degrees
  from each //of them.
x= (k-p1[1]) / math.tan(60) ;
SP1= Point(p1[0]+x,  k)  ;  //Steiner point-1
SP2= Point(p2[0]-x,  k) ;  //Steiner point-2
  // return the Steiner points of the rectangle.
return SP1,SP2 ;
}
```

IV. RESULTS AND DISCUSSION

***Ex1*:-** Suppose there are 3 towers arranged in triangular geometry. All 3 towers need to be connected to each other for proper communication. Figure 9.1 shows the connections, Figure 9.2 shows its MST and Figure 9.3 shows its Steiner tree representation. The total length of wire used in minimum in case of Steiner tree.

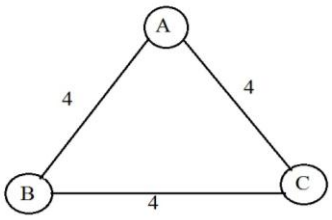

Figure 9.1. A graph of 3 towers.

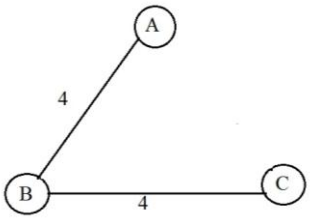

Figure 9.2. Minimum Spanning Tree (MST).

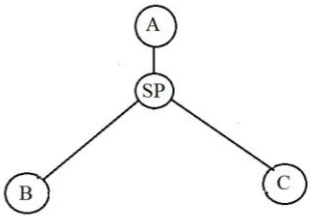

Figure 9.3. Steiner tree Representation of graph.

Total length of Figure 9.1 is 12, Figure 9.2 is 8 and Figure 9.3 is 6.928.

As the figure as only 3 vertices therefore no. of Steiner points will be 1.

***Ex2*:-** Suppose there are 4 towers arranged in rectangular geometry. All 4 towers need to be connected to each other for proper communication. Figure 10.1 shows the connections, Figure 10.2 shows its MST and Figure 10.3 shows its Steiner tree representation. The total length of wire used in minimum in case of Steiner tree. As this is a rectangle, it will have 2 Steiner points.

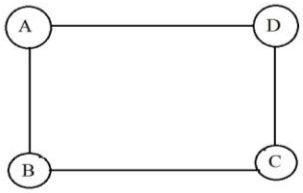

Figure 10.1. A network of 4 towers.

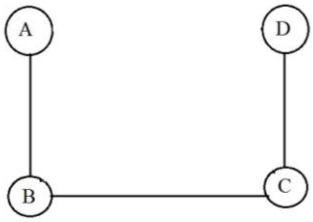

Figure 10.2. MST of towers.

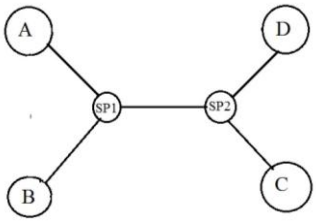

Figure 10.3. Steiner tree Representation.

Total path length of 10.1 is 13, 10.2 is 10 and 10.3 is 9.12

Similarly for pentagon and hexagon, the following figures depict Steiner points.

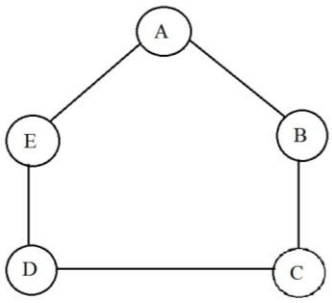

Figure 11.1. A network of 5 towers.

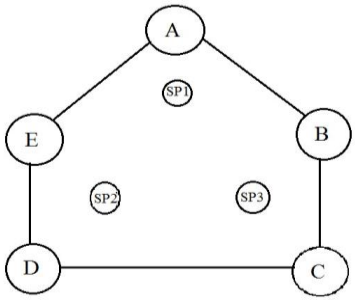

Figure 11.2. Steiner tree Representation of Pentagonal graph.

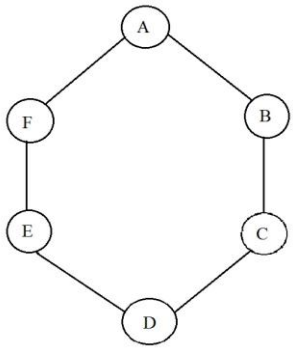

Figure 11.3. A network of 6 towers.

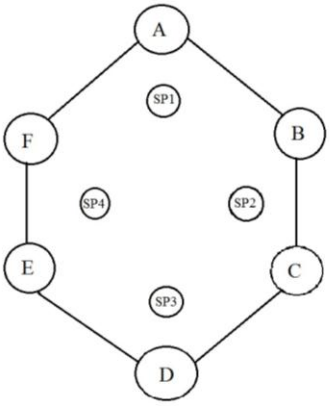

Figure 11.4. Steiner tree Representation of Hexagonal graph

***A. Cable Length Estimation without Steiner Point*:**

Consider a network of 4 cities to be connected through cables. Our aim will be to connect all cities such that the total cable length is minimized. Distances are in kms.

If every node is connected to each other,

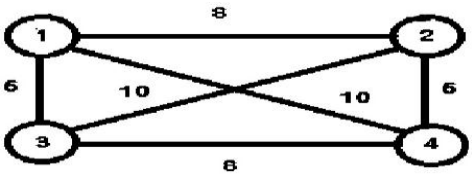

Figure 12. A network of 4 cities.

Total cost (length) = 8+8+6+6+10+10 = 48 km

Taking other possible combinations of the edges, and the cable length can be computed accordingly.

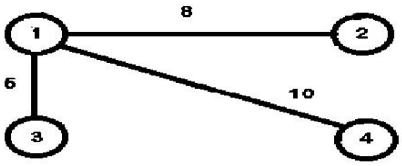

Figure 13.1. Combination (1) of graph edges.

Figure 13.2. Combination (2) of graph edges.

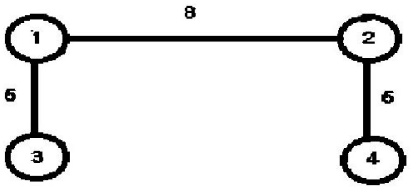

Figure 13.3. Combination (3) of graph edges.

For (Figure 13.1), Total cost = 6+8+10 = 24 km

For (Figure 13.2), Total cost = 6+8+8+6 = 28 km

For (Figure 13.3), i.e., For Minimum Cost Spanning Tree, Total cost = 6+8+6= 20 km

Thus, we get the shortest cable length = 20 km through connection like Figure 13.3.

Even if we consider a node in the center as shown in Figure 13.4:

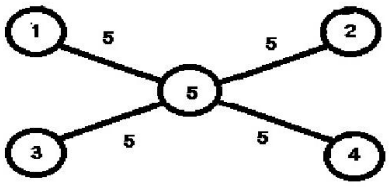

Figure 13.4. Combination (4) of graph edges.

Total cost = 5+5+5+5 = 20 km.

Thus, the minimum cost (length of cable) got up till now is 20 km.

B. Cable Length Estimation with Steiner Point:

Considering the same above problem, we will now show the result of solving the problem using Steiner points (using our algorithm).

Calculating with our algorithm (here, taking up to 4 decimal points),

- dist (s1 - s2) = 4.5359 km.
- dist (s1 - 1) = dist (s1 - 3) = dist (s2 - 2) = dist (s2 - 4) = 3.4641 km.

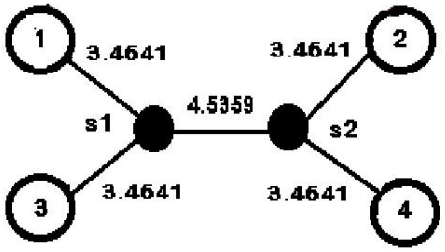

Figure 14. Graph including Steiner points.

Now, Total cost (Cable Length) = 3.4641*4 + 4.5359 = 18.3923 km. (lesser than all previous results).

Thus, we see that by using Steiner points, we got an optimum solution to our problem.

This difference (20 - 18.2923 = 1.6077 km) is significant. If cost of cable is taken as Rs.3/m, then this would imply a cost reduction of 3*1.6077*1000 = Rs.4823.1.

Also, it would imply reduction in attenuation.

V. CONCLUSION

Steiner points are useful to find out the optimum path for any network. They can provide a significant benefit in cost. Be it the cable length problem or any such network, Steiner points prove to be useful.

The significance of the result increases when the range of distance (or cost) increases, i.e., it may not significant for distances of a few kilometres, but it will offer a significant difference for larger distances or ranges. Also, the significance increases with the number of nodes of the network considered. Using our algorithm, Steiner points for networks of shape - triangle, rectangle, pentagon (a triangle and a rectangle part) and hexagon (2 triangle and a rectangle part) - can be calculated effectively. The logic may be extended further to provide solutions for a larger number of nodes (with proper considerations) [8,11,12].

REFERENCES

[1] Germander Soothill (April 27, 2010). The Euclidean Steiner Problem.

[2] Marshall W. Bern and Ronald L. Graham (January 1989). The shortest network problem, Scientific American.

[3] E. N. Gilbert and H. O. Pollak (Volume 16 no. 1, January 1968). SIAM Journal of Applied Mathematics.

[4] M.B.Chandak, R.Dharaskar, "Natural Language processing based context sensitive, content specific architecture and its speech based implementation for smart homes, International Journal of Smart homes, 4(2), 1-10, 2010.

[5] F.Riaz, K.M.Ali, "Application of graph theory in Computer Science" Third International Conference on Computational Intelligence, Communication Systems and Networks.

[6] H.Selim, P.Chopade, "Structural analysis and interactive visualization of large scale big data networks" IEEE 11th International Conference on Networking, Sensing and Control.

[7] N.Lakshmi Prasanna, "Application of Graph labelling in communication networks" Oriental Journal of Computer Science and Technology" ISSN-09724-6471

[8] Dr. S.M. Hegde, "Labeled graphs and Digraphs: Theory and Applications", 12-01-2012 Research Promotion Workshop on IGGA

[9] http://www.britinaca.com/bps/additionalcontent/18/3 3373769/concepts -of-graph-theory-relevant-to-Adhoc- Networks.

[10] T. Suel and J. Yuan. Compressing the graph structure of the web. In Proceedings of the IEEE Data Compression Conference, 2001.

[11] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001), "Section 22.1: Representations of graphs", *Introduction to Algorithms* (Second ed.), MIT Press and McGraw-Hill, pp. 527–531, ISBN 0-262-03293-7

[12] Balakrishnan, V. K. (1997-02-01). Graph Theory (1st ed.). McGraw-Hill. *ISBN 0-07-005489-4*.

[13] Svetlin Nakov, Fundamentals of Computer Programming with C#

Authors Profile

M.B.Chandak, is working as Head, Department of Computer Science and Engineering at Shri Ramdeobaba College of Engineering and Management. He is Ph.D. in Computer Science and Engineering with Natural Language Processing as Specialization. He has good number of publications in international journal of repute.

Mr.S.Balotia, is undergraduate student, studying in Final year B.E at Department of Computer Science and Engineering at Shri Ramdeobaba College of Engineering and Management. His research interest is Big data and Graph Algorithms.

Mr.S.C.Agarwal is undergraduate student, studying in Final year B.E at Department of Computer Science and Engineering at Shri Ramdeobaba College of Engineering and Management. His research interest is Graph Algorithm and Optimization Techniques.

