FINDING SPANNING TREE MINIMIZING THE MAXIMUM EDGE LOAD

A thesis submitted
to Kent State University in partial
fulfillment of the requirements for the
degree of Masters of Science

by
Siddharth K Raina
December 2006
Thesis written by
Siddharth K Raina
B.S, Nagpur University, 1999
M.S, Kent State University, 2006

Approved by

Dr. Feodor F. Dragan____________, Advisor

Dr. Robert A. Walker___________, Chair, Department of Computer Science

Dr. John R. D. Stalvey__________, Dean, College of Arts and Sciences
TABLE OF CONTENTS

TABLE OF CONTENTS ................................................................................................ iii
LIST OF FIGURES .......................................................................................................... v
CHAPTER 1  INTRODUCTION ................................................................................ 1
  1.1 Graph ................................................................................................................. 2
  1.2 Spanning Tree ................................................................................................... 4
CHAPTER 2  PROBLEM FORMULATION .............................................................. 6
  2.1 Edge Load ......................................................................................................... 6
  2.2 Motivation ......................................................................................................... 8
  2.3 Significance of largest loaded edge ................................................................. 11
  2.4 Objective ......................................................................................................... 12
  2.5 Effect of Topology .......................................................................................... 12
CHAPTER 3  PROPOSED HEURISTICS ................................................................. 14
  3.1 Observations .................................................................................................. 14
  3.2 Max-leaf Tree Heuristic (MLTH) .................................................................... 15
    3.2.1 Motivation ............................................................................................... 15
    3.2.2 Algorithm ............................................................................................... 15
    3.2.3 Proposed Heuristic .................................................................................. 26
  3.3 Best Shortest Path Heuristic (BSPH) ............................................................... 28
    3.3.1 Motivation ............................................................................................... 28
    3.3.2 Algorithm ............................................................................................... 29
    3.3.3 Proposed Heuristic .................................................................................. 38
  3.4 Minimum Routing Cost Tree Heuristic (MRCH) .............................................. 40
    3.4.1 Motivation ............................................................................................... 40
    3.4.2 Algorithm ............................................................................................... 40
LIST OF FIGURES

FIG. 1 DIRECTED GRAPH ...........................................................................................................................2
FIG. 2 UNDIRECTED GRAPH ......................................................................................................................3
FIG. 3 BI-PARTITE GRAPH .........................................................................................................................3
FIG. 4 SPANNING TREE ..............................................................................................................................4
FIG. 5 SINGLE SPANNING TREE ...............................................................................................................7
FIG. 6 SUBTREES AFTER REMOVAL OF AN EDGE ...............................................................................7
FIG. 7 MRCT - TREE A .................................................................................................................................8
FIG. 8 NOT AN MRCT - TREE B .................................................................................................................9
FIG. 9 TREE A - STAR ...............................................................................................................................12
FIG. 10 TREE B ............................................................................................................................................13
FIG. 11 INPUT GRAPH ...............................................................................................................................17
FIG. 12 FOREST OF SINGLETONS ...........................................................................................................17
FIG. 13 FOREST AFTER ITERATION FOR VERTEX 1 ..........................................................................18
FIG. 14 FOREST AFTER ITERATION FOR VERTEX 2 ..........................................................................20
FIG. 15 FOREST AFTER ITERATION FOR VERTEX 3 ..........................................................................21
FIG. 16 FOREST AFTER ITERATION FOR VERTEX 4 ..........................................................................22
FIG. 17 FOREST AFTER ITERATION FOR VERTEX 5 ..........................................................................23
FIG. 18 FOREST AFTER ITERATION FOR VERTEX 6, 7 AND 8 ..........................................................24
FIG. 19 TREE GENERATED AFTER STEP 2 OF 3-APPROX MAXIMUM LEAF ALGORITHM...............25
FIG. 20 INPUT GRAPH ...............................................................................................................................27
FIG. 21 RECURSIVE RE-GENERATION OF TREE USING MLTH ..........................................................27
FIG. 22 ITERATION OF MLTH ON G SHOWN IN FIG. 20 .................................................................28
FIG. 23 INPUT GRAPH ...............................................................................................................................30
FIG. 24 STEPWISE GENERATION OF SHORTEST PATH TREE WITH VERTEX 1 AS SOURCE.....37
FIG. 25 SPT FOR DIFFERENT VERTICES AS SOURCE AND CALCULATION OF $L(E_{MAX})$ ........39
FIG. 26 SPT FOR VERTEX 8 AS SOURCE RESULTS IN 2-APPROXIMATION MRCT ......................42
FIG. 27 SPT FOR DIFFERENT SOURCE AND CALCULATION OF SHORTEST PATH FOR EACH .42
FIG. 28 TREE A WITH $L_a(E_{MAX}) = 30$ & TREE B WITH $L_b(E_{MAX}) = 36$...............................43
FIG. 29 INPUT BI-PARTITE GRAPH .................................................................44
FIG. 30 POSSIBLE MATCH FOR INPUT BI-PARTITE GRAPH .........................44
FIG. 31 GRAPH AFTER INITIAL SEMI MATCHING ........................................45
FIG. 32 GENERATION OF INITIAL SEMI MATCHING .....................................46
FIG. 33 TREE GENERATED BY BSPH ..........................................................47
FIG. 34 TREE WITH VERTICES LABELED WITH THEIR HEIGHTS..................47
FIG. 35 BI-GRAPH FORMED BY TAKING EDGES BETWEEN VERTICES AT HEIGHT 2 AND 1 ...48
FIG. 36 OUTPUT OF INITIAL_SEMI_MATCHING ON BI-PARTITE GRAPH SHOWN IN FIG. 35 ...48
FIG. 37 BI-GRAPH FORMED BY TAKING EDGES BETWEEN VERTICES AT HEIGHT 1 AND 0 ...48
FIG. 38 OUTPUT OF INITIAL_SEMI_MATCHING ON BI-GRAPH SHOWN IN FIG. 37........48
FIG. 39 TREE GENERATED BY UNION OF TREES SHOWN IN FIG. 38 AND FIG. 36 ..........49
FIG. 40 MAIN SCREEN ..............................................................................51
FIG. 41 ADDING VERTICES TO CREATE GRAPH ...........................................52
FIG. 42 CONNECTING VERTICES TO FORM AN EDGE ....................................52
FIG. 43 FORMING A GRAPH ........................................................................53
FIG. 44 SELECTING A HEURISTIC TO RUN ..................................................53
FIG. 45 LOAD ON EMAX SHOWN AS A LINK ON TOP LEFT CORNER OF SCREEN ..........54
FIG. 46 TREE GENERATED BY SELECTED HEURISTIC ................................54
FIG. 47 SELECTION GRAPH MODEL TO SIMULATE ......................................55
FIG. 48 ENTER INPUT PARAMETERS FOR RANDOM GRAPHS ......................56
FIG. 49 ENTER INPUT PARAMETERS FOR WAXMAN GRAPHS ......................56
FIG. 50 ENTER INPUT PARAMETERS FOR POWER-LAW GRAPHS ..................57
FIG. 51 SELECTING PARAMETERS AS SELECTION CRITERIA .........................58
FIG. 52 FOR INPUT GRAPHS, $L(E_{MAX})$ FROM EACH HEURISTIC IS SHOWN ALONG Y-AXIS ....59
CHAPTER 1

INTRODUCTION

Graphs theory is an old subject with many modern applications. Graphs are used to solve problems in many fields. In problems arising in computer science, mathematics, engineering, and many other disciplines we often need to represent arbitrary relationship among data objects. Graph is a mathematical object that perfectly models any binary relationships.

Computer network is a system of communication between computers. The growth in numbers and types of users has led to deployment of complex networks and need to optimize its performance. An analogy between graphs and networks is easy to identify. On one hand where Graph is a collection of vertices linked to each other by set of edges, a network is set of nodes linked to each other by a medium. Because of its close analogy, a network is often modeled as graph and various algorithms can be used to optimize its certain characteristics.

Many problems where a network of communicating hosts is represented as a graph, solution bears a spanning tree. For example, a switched network should have only one path to any destination active at any one point in time, this is called a loop free topology. If more than one open path were to exist then data frames could loop endlessly (known as a broadcast storm) crippling the network. The minimum spanning tree algorithm ensures that only one path to a destination is available at any one time by detecting loops and blocking switch ports as required.
1.1 Graph

A Graph is defined as set of vertices joined by set of links called edges. Depending on application, edges can be directed. A directed graph is an ordered pair \( G = (V, A) \) with \( V \), set of vertices and \( A \), set of ordered pairs of vertices \( (u, v) \in A \), called directed edges represented as \( e(u, v) \), where \( u \) is the origin and \( v \) the destination. Since, the edges are ordered pairs \( e(u, v) \neq e(v, u) \). Fig. 1 shows a directed graph \( G_1 = (V, A) \) where \( V = \{1, 2, 3, 4, 5, 6, 7, 8\} \) and \( A = \{(1, 3), (2, 1), (2, 7), (3, 2), (4, 2), (4, 5), (5, 6), (6, 4), (8, 4), (8, 7)\} \).

A undirected graph is an unordered pair \( G = (V, E) \) with \( V \), set of vertices and \( E \), set of unordered pairs of vertices of vertices \( (u, v) \in E \), called undirected edges represented as \( e(u, v) \), where \( u, v \in V \). Since, the edges are unordered pairs \( (u, v) = (v, u) \). Fig. 2 shows an undirected graph \( G_2 = (V, E) \) where \( V = \{1, 2, 3, 4, 5, 6, 7, 8\} \) and \( E = \{(1, 2), (1, 3), (2, 3), (2, 7), (2, 4), (4, 5), (4, 6), (4, 8), (5, 6), (7, 8)\} \).

![Fig. 1 Directed Graph](image1.png)

A graph is said to be connected if there is a path from any vertex to any other vertex in it. A connected graph can further be classified as dense or sparse, depending on its density of edges. In literature we do not have any definite definition of the density of a graph. For graph \( G = (V, E) \)
let $|e|$ denote number of edges in graph and $n$ denote number of vertices, then to keep graph connected we have, $(n - 1) \leq |e| \leq n \times (n - 1) / 2.$

![Fig. 2 Undirected Graph](image)

In this thesis we divide the range given above into 3 equal parts & assign each range a density level. We calculate number of edges $\hat{\partial}_i$ at each density level $i$ with following upper and lower bounds:

$$(n - 1) + (i - 1) \left\lceil \frac{(n - 1) \times (n - 2)}{6} \right\rceil < \hat{\partial}_i \leq i \left\lceil \frac{(n - 1) \times (n - 2)}{6} \right\rceil,$$

where $i = \{1, 2, 3\}.$

![Fig. 3 Bi-partite Graph](image)
A bi-partite graph, also called a bi-graph, is a set of graph vertices decomposed into two disjoint sets such that no two graph vertices within the same set are adjacent. Formally, a bi-partite graph is a graph $G = (V \cup U, E)$ in which $E \subseteq U \times V$. Fig. 3 below shows an example of bi-partite graph $G = (V \cup U, E)$ where $V = \{1, 2, 3, 4\}$ and $U = \{5, 6, 7\}$ and $E = \{(1, 6), (1, 7), (2, 5), (3, 6), (3, 7), (4, 5), (4, 7)\}$.

In this thesis, unless otherwise specified a Graph would mean undirected, connected graph without weights.

### 1.2 Spanning Tree

A spanning tree for a given graph is its sub-graph that is a tree and contains all of its vertices. Hence, a graph $G = (V, E)$ has a spanning tree $T = (V, E)$ where

- $V = V$
- $E \subseteq E$
- There exists a unique path between any two vertices of the tree $T$.
- Fig. 4 shows a spanning tree $T$ for graph $G_2$ shown in Fig. 2

![Fig. 4 Spanning Tree](image-url)
A forest is a set of disconnected trees, hence an acyclic graph. A forest with \( k \) components and \( n \) nodes has \( n - k \) graph edges. In this thesis, unless otherwise specified a Tree would mean undirected, un-weighted connected tree.

In the next chapter we shall define load of edge in a tree, maximum edge of a tree and discuss effect of tree topology on load of maximum edge. We further continue with definition of the problem we address in this thesis.
CHAPTER 2

PROBLEM FORMULATION

With different aspects, there are different measurements of the goodness of a spanning tree. As an example, *Minimum Spanning Tree (MST)* is an optimal solution for networks with low building cost i.e. cost to install a link where as *Minimum Routing Cost Tree (MRCT)* is optimal solution for networks with low routing cost i.e. cost to traverse it after the link is installed. In this chapter we discuss the aspect of minimizing load on maximum edge as measurement of a good spanning tree.

2.1 Edge Load

Let \( T = (V, E) \) be a tree and \( e \in E \). Assume \( T_1 \) and \( T_2 \) are the two sub graphs that result by removing \( e \) from \( T \). Then load on edge \( e(u, v) \) is defined as \( l(e(u, v)) = | T_1 | \times | T_2 | \), where \( | T_1 | \) represents number of vertices in sub-graph \( T_1 \) and \( | T_2 | \) represents number of vertices in sub-graph \( T_2 \) [13].

In terms of network of communicating nodes, a link between neighboring nodes can be represented as an edge. Load on any such link can be thought of as number of times it would be used for communication between any pair of nodes in the routing tree. For example, consider the spanning tree \( T \) as shown in Fig. 5, \( T = (V, E) \) where \( V = \{1, 2, 3, 4, 5, 6, 7, 8\} \) and \( E = \{e(1, 2), \ldots, e(7, 8)\} \).
\( e(2, 3), e(2, 4), e(2, 7), e(4, 5), e(4, 6), e(7, 8) \). To calculate load of edge \( e(2, 7) \), we assume to remove edge \( e(2, 7) \) from \( T \), resulting 2 sub graphs \( T_1 \) (nodes with black background) and \( T_2 \) (nodes with white background) as shown in Fig. 6.

By definition, \( l(e(2, 7)) = 6 \times 2 = 12 \). As seen from Fig. 6, edge \( e(2, 7) \) is used once for every communication between 6 vertices of sub-graph \( T_1 \), and 2 vertices in sub-graph \( T_2 \), thus forming 12 communication channels using \( e(2, 7) \).
2.2 Motivation

Consider the following problem in network design: given a graph with routing cost on the edges, the goal is to find a spanning tree such that the average routing cost of communicating between any pair using the tree is minimized. In general, the routing cost of the tree itself would be sum over all pairs of the routing cost for the pair in this tree i.e. $C(T) = \sum_{u,v} d_{T}[u, v]$ where $C(T)$ is the total routing cost of the spanning tree $T$ and $d_{T}[u, v]$ is the distance between vertex $u$ and $v$ in spanning tree $T$.

Minimizing the average routing cost is equivalent to minimizing the total routing cost between all pairs of vertices using the tree. MRCT is the one with minimum routing cost among all possible spanning trees for a given graph. For example, below we show 2 out of all the possible spanning trees of graph in Fig. 2. The one shown in Fig.7 is MRCT for the given graph where as the one shown in Fig. 8 is not.

![Fig. 7 MRCT - Tree A](image-url)
We show calculation of routing cost of both the trees in Table 1. Now let’s calculate load on each edge of both the trees. In tree A, we have $e(2, 4)$ has the largest load, $l(e(2, 4)) = 16$, where as in tree B largest loaded edge is $e(2, 7)$ with load, $l(e(2, 7)) = 15$. In this thesis, we try to find a tree (similar to tree B) from a given graph which has lowest value of largest loaded edge among all the possible spanning trees.
<table>
<thead>
<tr>
<th>Communicating Pairs</th>
<th>Routing cost in tree $A$</th>
<th>Routing cost in tree $B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e(1, 2)$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$e(1, 3)$</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$e(1, 4)$</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$e(1, 5)$</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>$e(1, 6)$</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>$e(1, 7)$</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$e(1, 8)$</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>$e(2, 3)$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$e(2, 4)$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$e(2, 5)$</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$e(2, 6)$</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$e(2, 7)$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$e(2, 8)$</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$e(3, 4)$</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$e(3, 5)$</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>$e(3, 6)$</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>$e(3, 7)$</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$e(3, 8)$</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>$e(4, 5)$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$e(4, 6)$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$e(4, 7)$</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$e(4, 8)$</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
Table 1: Calculation of routing cost of Tree A and Tree B

2.3 Significance of largest loaded edge

The above example shows that Tree A though has better routing cost compared to that of tree B, but keeping in mind analogy between trees and networks lets consider failure of link representing largest loaded edge. Failure of such a link between node 2 and 4 in tree A would leave us with 2 disconnected sub-trees, both of 4 nodes which can not communicate, i.e. it would affect 16 communication channels. Whereas in case of tree B, failure of link between node 2 and 4 would leave us with 2 disconnected sub-trees, one with 3 nodes and another one with 5 which can no more communicate, i.e. it would affect 15 communication channels. Thus by selecting a spanning tree with minimum value of largest loaded edge, we make sure that failure of such a link would affect less number of communication channels in the network.
2.4 Objective

“In this thesis, our primary goal is to present and compare different heuristics of finding a spanning tree $T = (V, U)$ for a given graph $G = (V, E)$, where $U \subseteq E$, such that largest edge load of $T$ is minimized among all possible trees for the given graph $G$.”

For a graph $G$, we call such an edge as maximum edge, $e_{\text{max}}$, and represent load on maximum edge in resulting tree $T$ as $l_T(e_{\text{max}})$.

2.5 Effect of Topology

Topology of the resulting spanning tree plays a very important factor in the value of the largest loaded edge it has. Next example illustrates the impact of topology by considering two extreme cases.

Fig. 9 Tree A - Star
Suppose for a given graph $G$ with $n$ vertices, we have spanning trees $A$ and $B$ as shown in Fig. 9 and Fig. 10 respectively. For tree $A$, the load on each of its edge is $(n - 1)$, since each edge is incident with a leaf. Hence $l_A(e_{\text{max}}) = (n - 1)$. Whereas for tree $B$ removing an edge $e(V_i, V_{i+1})$ will result in two components of $i$ and $(n - i)$ vertices. Hence $l_B(e_{\text{max}})$ would be calculated as given below.

$$l_T(e_{\text{max}}) = \frac{(n - 1) \times (n + 1)}{4} = \frac{n^2 - 1}{4}, \text{where } n \text{ is odd} \ldots \ldots \ldots \ldots \ldots (1)$$

$$= \frac{n^2}{4}, \text{where } n \text{ is even.}$$

From the above example we see that topology of tree $A$, a star is a favorable topology for a tree with least possible value of maximum edge, $e_{\text{max}}$ for a given graph.

$MRCT$ problem is proven to be NP-complete [3]. As discussed in Section 2.2, the problem we define is close to the problem of finding $MRCT$ for a given graph. For this reason, in this thesis no attempt is made to prove NP-completeness of the problem of minimizing load on the largest loaded edge. Instead, we propose heuristics based on some well known algorithms and run them on graphs based on well known topologies such as Waxman and Power-law to compare results of these heuristics.

We see that, $\text{Bi-partite Semi-matching Heuristic (BSH)}$ outperforms other heuristics considered and $\text{Max-Leaf Tree Heuristic (MLTH)}$ on an average fetches worst result.
CHAPTER 3

PROPOSED HEURISTICS

In this chapter we discuss some properties of a tree that we could target to generate spanning tree with the desired topology. We give the heuristics and describe the algorithms on which they are based.

3.1 Observations

In Chapter 2 we discussed crucial role topology of the generated tree plays in the value of largest loaded edge of the generated tree. We also saw that, ideally star is the desired topology. A few observations about tree with such a topology are:

1. It has maximum number of leaves possible,
2. Length of shortest path from source to every other vertex is 1.

From (1), we see that if an edge divides a spanning tree into two sub-trees of same size i.e. \( n / 2 \), then value of largest loaded edge would be in \( O(n^2) \). Hence, in our heuristic we try to avoid such a situation.
3.2 Max-leaf Tree Heuristic (*MLTH* )

### 3.2.1 Motivation

As mentioned, one of the properties of tree that we want to target in order to generate desired topology is a tree with maximum number of leaves. Hence, in this heuristic we make use of approximation algorithm that generates tree with maximum leaves.

### 3.2.2 Algorithm

Given an undirected graph, finding a spanning tree of with the maximum number of leaves is NP-complete [1]. Lu & Ravi [2] gave a 3-approximation maximum leaf spanning tree algorithm. The algorithm computes an approximation of maximum leaf spanning tree in two steps.

First it constructs a maximal leafy forest $F$. The forest consists of several leafy trees and there may be some vertices not in the forest, which are viewed as trees of singleton. In the second step, the leafy trees and the trees of singleton are combined into a spanning tree by adding edges across the trees. The following is the algorithm for constructing a maximal leafy forest.

3_Appox_Max_Leaf(Graph $G$) [2]

Let $F$ be an empty set

For every vertex $v$ in $G$ do

$$S(v) = \{v\}$$
\[ d(v) = 0 \]

For every vertex \( v \) in \( G \) do

\[ S' = \emptyset \]

\[ d' = 0 \]

For every vertex \( u \) that is adjacent to \( v \) in \( G \) do

If \( u \not\in S(v) \) and \( S(u) \not\in S' \) then

\[ d' = d' + 1 \]

Insert \( S(u) \) into \( S' \)

If \( d(v) + d' \geq 3 \) then

For every \( S(u) \) in \( S' \) do

Add \( e(u, v) \) to \( F \)

\[ S(u) = S(v) = S(u) \cup S(v) \]

Update \( d(u) = d(u) + 1 \) and \( d(v) = d(v) + 1 \)

The second step of the whole algorithm is to connect all the leafy trees and the singleton trees to form a spanning tree. In first step, we use \( S(w) \) to denote the sub-tree of \( F \) that contains vertex \( w \). The degree of node \( w \) in \( F \) is kept in \( d(w) \). The variable \( d' \) is the maximal number of edges adjacent to vertex \( v \) that could be added to \( F \) without creating cycles. If there exists an edge \( e(u, v) \) in \( d' \), then \( S(u) \) is stored in the set \( S' \). If \( d' + d(u) \) is greater than or equal to three, then we add those \( d \) edges to \( F \) and union \( S(v) \) with those \( d \) sub-trees \( S(u) \).
Using the graph shown in Fig. 11, we give iterations of 3\_Approx\_Max\_Leaf algorithm. After iteration for each vertex we show status of forest with all the sub-trees shown separately.

**Step 1:**

\[ F = \emptyset \]

\[ d(1) = d(2) = d(3) = d(4) = d(5) = d(6) = d(7) = d(8) = 0 \]

\[ S(1) = \{1\}, S(2) = \{2\}, S(3) = \{3\}, S(4) = \{4\}, S(5) = \{5\}, S(6) = \{6\}, S(7) = \{7\}, S(8) = \{8\} \]
Iteration for vertex 1

\( S' = \phi \)

d' = 0

for vertex 2:

\[ 2 \notin S(1) \text{ and } S(2) \notin S' \Rightarrow \text{True} \]

d' = 1, \( S' = \{S(2)\} \)

for vertex 7:

\[ 7 \notin S(1) \text{ and } S(7) \notin S' \Rightarrow \text{True} \]

d' = 2, \( S' = \{S(2), S(7)\} \)

if \([d(1) + d' = 2] \geq 3 \Rightarrow \text{False}\)

---

**Fig. 13 Forest after iteration for vertex 1**

---

Iteration for vertex 2

\( S' = \phi \)

d' = 0
for vertex 1:

\[ 1 \not\in S(2) \text{ and } S(1) \not\in S' \Rightarrow \text{True} \]

\[ d' = 1, \ S' = \{S(1)\} \]

for vertex 3:

\[ 3 \not\in S(2) \text{ and } S(3) \not\in S' \Rightarrow \text{True} \]

\[ d' = 2, \ S' = \{S(1), S(3)\} \]

for vertex 4:

\[ 4 \not\in S(2) \text{ and } S(4) \not\in S' \Rightarrow \text{True} \]

\[ d' = 3, \ S' = \{S(1), S(3), S(4)\} \]

if \( [d(2) + d' = 3] \geq 3 \Rightarrow \text{True} \)

for \( S(1) \)

\[ F = \{e(1, 2)\} \]

\[ S(2) = S(1) = \{e(1, 2)\} \]

\[ d(2) = 1, \ d(1) = 1 \]

for \( S(3) \)

\[ F = \{e(1, 2), e(2, 3)\} \]

\[ S(2) = S(1) = S(3) = \{1, 2, 3\} \]

\[ d(2) = 2, \ d(3) = 1 \]

for \( S(4) \)

\[ F = \{e(1, 2), e(2, 3), e(2, 4)\} \]

\[ S(2) = S(1) = S(3) = S(4) = \{1, 2, 3, 4\} \]

\[ d(2) = 3, \ d(4) = 1 \]
Iteration for vertex 3

$S' = \emptyset$

$d' = 0$

for vertex 2:

$2 \not\in S(3)$ and $S(2) \not\in S' \Rightarrow \text{False}$

for vertex 8:

$8 \not\in S(3)$ and $S(8) \not\in S' \Rightarrow \text{True}$

$d' = 1, S' = \{S(8)\}$

if $[d(3) + d' = 2] \geq 3 \Rightarrow \text{False}$
Iteration for vertex 4

\[ S' = \emptyset \]
\[ d' = 0 \]

for vertex 2:

\[ 2 \not\in S(4) \text{ and } S(2) \not\in S' \rightarrow \text{False} \]

for vertex 5:

\[ 5 \not\in S(4) \text{ and } S(5) \not\in S' \rightarrow \text{True} \]

\[ d' = 1, S' = \{S(5)\} \]

for vertex 6:

\[ 6 \not\in S(4) \text{ and } S(6) \not\in S' \rightarrow \text{True} \]

\[ d' = 2, S' = \{S(5), S(6)\} \]

if \[ d(4) + d' \geq 3 \rightarrow \text{True} \]

for \( S(5) \)

\[ F = \{e(1, 2), e(2, 3), e(2, 4), e(4, 5)\} \]
\[ S(2) = S(1) = S(3) = S(4) = S(5) = \{1, 2, 3, 4, 5\} \]
\[ d(4) = 2, \; d(5) = 1 \]

for \( S(6) \)

\[ F = \{e(1,2), \; e(2,3), \; e(2,4), \; e(4,5), \; e(4,6)\} \]
\[ S(2) = S(1) = S(3) = S(4) = S(5) = S(6) = \{1, 2, 3, 4, 5, 6\} \]
\[ d(4) = 3, \; d(6) = 1 \]

Fig. 16 Forest after iteration for vertex 4

**Iteration for vertex 5**

\[ S' = \emptyset \]
\[ d' = 0 \]

for vertex 4:

\[ 4 \not\in S(5) \text{ and } S(4) \not\in S' \Rightarrow \text{False} \]

for vertex 6:

\[ 6 \not\in S(5) \text{ and } S(6) \not\in S' \Rightarrow \text{False} \]
for vertex 7:

\[ 7 \not\in S(5) \text{ and } 7 \not\in S' \Rightarrow \text{True} \]

\[ d' = 1, S' = \{S(7)\} \]

if \[ d(5) + d' = 2 \geq 3 \Rightarrow \text{False} \]

Fig. 17 Forest after iteration for vertex 5

Iteration for vertex 6

\[ S' = \emptyset \]

\[ d' = 0 \]

for vertex 4:

\[ 4 \not\in S(6) \text{ and } 4 \not\in S' \Rightarrow \text{False} \]

for vertex 5:

\[ 5 \not\in S(6) \text{ and } 5 \not\in S' \Rightarrow \text{False} \]

for vertex 8:

\[ 8 \not\in S(6) \text{ and } 8 \not\in S' \Rightarrow \text{True} \]
\[ d' = 1, S' = \{S(8)\} \]

if \([d(6) + d' = 2] \geq 3 \Rightarrow \text{False}\]

**Iteration for vertex 7**

\[ S' = \emptyset \]

\[ d' = 0 \]

for vertex 1:

\[ 1 \notin S(7) \text{ and } S(1) \notin S' \Rightarrow \text{True} \]

\[ d' = 1, S' = \{S(1)\} \]

for vertex 5:

\[ 5 \notin S(7) \text{ and } S(5) \notin S' \Rightarrow \text{False} \]

for vertex 8:

\[ 8 \notin S(7) \text{ and } S(8) \notin S' \Rightarrow \text{True} \]

\[ d' = 2, S' = \{S(1), S(8)\} \]

if \([d(7) + d' = 2] \geq 3 \Rightarrow \text{False}\]

![Fig. 18 Forest after iteration for vertex 6, 7 and 8](image-url)
Iteration for vertex 8

\[ S' = \emptyset \]

\[ d' = 0 \]

for vertex 2:

\[ 2 \notin S(8) \text{ and } S(2) \notin S' \Rightarrow \text{True} \]

\[ d' = 1, S' = \{S(2)\} \]

for vertex 6:

\[ 6 \notin S(8) \text{ and } S(6) \notin S' \Rightarrow \text{False} \]

for vertex 7:

\[ 7 \notin S(8) \text{ and } S(7) \notin S' \Rightarrow \text{True} \]

\[ d' = 2, S' = \{S(2), S(7)\} \]

if \[ d(8) + d' \geq 3 \Rightarrow \text{False} \]

Step 2: Here we connect all the sub-tress & singletons in the forest to form a tree.

Fig. 19 Tree generated after Step 2 of 3-Approx Maximum Leaf Algorithm
3.2.3 Proposed Heuristic

Each iteration of this heuristic generates a new graph $G''$ which has lesser vertices compared to graph generated in previous iteration $G'$. Input graph $G''$ goes through following steps.

1. generating 3 approximation of Maximum Leaf tree, for the graph which is input to the current iteration,
2. removing the leaf edges, resulting in tree $T'$, the leaf edges are added to the output tree $T$ of the heuristic,
3. populating $T'$ with edges from the original graph, resulting in graph $G'$ which becomes input for next iteration

The iterations continue till there is more than one vertex root in the $G'$, and ends by adding edge from root to itself i.e. $e(root, root)$. For input graph $G$ the heuristic is as follows:

$$
T = \emptyset
$$

$$
G' = G
$$

$$
T = T \cup MLSH(G')
$$

MLSH(Graph $G'$)

if($|G'| > 1$)

$$
T' = 3_{Appox\_Max\_Leaf}(G')
$$

$$
e' = Clip\_Leaves(T')
$$

$$
G'' = Add\_Edges\_From(G, T')
$$

return $e' \cup MLSH(G'')$

else

return $e(root, root)$
In \textit{Clip\_Leaves}(T') we clip all the leaf edges of tree T'. We can do this by first removing the vertices with only 1 neighbor and then updating neighbor list of rest of the vertices. \textit{Add\_Edges\_From}(G, T') populates T' with edges from graph G. Consider G as shown in the Fig. 20. Sample iteration of the proposed heuristic used to find spanning tree with minimum heaviest loaded edge in shown in Fig. 22.

![Input Graph](image)

**Fig. 20 Input Graph**

Fig. 21 shows recursive generation of the tree with minimum value of largest-loaded edge using \textit{MLTH} heuristic after single node is left in iteration 3.

![Recursive re-generation of tree using MLTH](image)

**Fig. 21 Recursive re-generation of tree using MLTH**
3.3 Best Shortest Path Heuristic (BSPH)

3.3.1 Motivation

In this heuristic we make use of Dijkstra’s single source shortest path algorithm (SSSP), as it was observed that the star topology also has the shortest path from root to every other node of the tree. Hence, by making use of SSSP, we try to find shortest path tree (SPT) for each vertex as source and choose the one which yields smallest value of largest loaded edge.
3.3.2 Algorithm

In this section we describe the single source shortest path algorithm. This algorithm is used in this heuristic and the one which uses 2-Approximation of MRCT discussed in the following section.

In graph theory, the single-source shortest path problem is the problem of finding a path between two vertices such that the sum of the weights of its constituent edges is minimized. Dijkstra’s algorithm (SSSP) computes the shortest paths from a given start point $s$ to all other nodes. The shortest path algorithm for a given graph $G = (V, E)$ and source vertex $s$ is as given below.

SSSP(Graph $G$, vertex $s$)

For every node $v$ in $G$
\[ d[v] = \infty, \text{previous}[v] = \text{undefined} \]
\[ d[s] = 0 \]
\[ S = \phi, T = \phi \]
\[ Q = V \]
While $Q \neq \phi$
\[ u = \text{extractMin}(Q) \]
\[ S = S \cup \{u\} \]
For every node $v$ which is neighbor of $u$
\[ \text{if } d[v] > d[u] + w(u, v) \]
\[ d[v] = d[u] + w(u, v) \]
\[ \text{previous}[v] = u \]
\[ T \cup e(u, v) \]
return T

The algorithm works by keeping for each vertex \( v \), the cost \( d[v] \) of the shortest path found so far between source, \( s \) and \( v \). Basic operation of Dijkstra’s algorithm is edge relaxation, where if there is an edge from \( u \) to \( v \), then the shortest known path from \( s \) to \( u \) (\( d[u] \)) can be extended to a path from \( s \) to \( v \) by adding \( e(u, v) \) at the end. This path will have length \( d[u] + w(u, v) \). If this is less than the current \( d[v] \), we can replace the current value of \( d[v] \) with the new value. Edge relaxation is applied until all values \( d[v] \) represent the cost of the shortest path from \( s \) to \( v \).

The algorithm is organized so that each edge \( e(u, v) \) is relaxed only once, when \( d[u] \) has reached its final value. The function \( extractMin(Q) \), searches for node \( u \) in the vertex set \( Q \) that has least \( d[u] \) value. The node is then removed from the set \( Q \) and returned.

Consider the graph shown in Fig. 23 to be the input, we iterate through the algorithm to generate the shortest path tree. Consider vertex 1 as source.

For every node

\[
\]

\[
previous[1] = previous[2] = \ldots = previous[9] = \text{undefined}
\]
\(d[1] = 0\)

\(S = \emptyset, T = \emptyset\)

\(Q = V = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}\)

while \(Q \neq \emptyset\) \(\Rightarrow\) True

\(u = extractMin(Q) = 1 \Rightarrow Q = \{2, 3, 4, 5, 6, 7, 8, 9\}\)

\(S = \{1\}\)

for vertex 2

\(if d[2] > d[1] + w(1, 2) \Rightarrow True\)

\(d[2] = d[1] + w(1, 2) = 1\)

\(previous[2] = 1\)

\(T = \{e(1,2)\}\)

for vertex 8

\(if d[8] > d[1] + w(1, 8) \Rightarrow True\)

\(d[8] = d[1] + w(1, 8) = 1\)

\(previous[8] = 1\)

\(T = \{e(1, 2), e(1, 8)\}\)

for vertex 9

\(if d[9] > d[1] + w(1, 9) \Rightarrow True\)

\(d[9] = d[1] + w(1, 9) = 1\)

\(previous[9] = 1\)

\(T = \{e(1, 2), e(1, 8), e(1, 9)\}\)

while \(Q \neq \emptyset\) \(\Rightarrow\) True

\(d[1] = 0\)


\[ u = extractMin(Q) = 2 \implies Q = \{3, 4, 5, 6, 7, 8, 9\} \]

\[ S = \{1, 2\} \]

for vertex 1

\[ if \; d[1] > d[2] + w(2, 1) \implies False \]

for vertex 8

\[ if \; d[8] > d[2] + w(2, 8) \implies False \]

for vertex 3

\[ if \; d[3] > d[2] + w(2, 3) \implies True \]

\[ d[3] = d[2] + w(2, 3) = 2 \]

\[ previous[3] = 2 \]

\[ T = \{e(1, 2), e(1, 8), e(1, 9), e(2, 3)\} \]

while \( Q \neq \emptyset \implies True \]

\[ d[1] = 0 \]


\[ d[3] = 2 \]


\[ u = extractMin(Q) = 8 \implies Q = \{3, 4, 5, 6, 7, 9\} \]

\[ S = \{1, 2, 8\} \]

for vertex 1

\[ if \; d[1] > d[8] + w(8, 1) \implies False \]

for vertex 2

\[ if \; d[2] > d[8] + w(8, 2) \implies False \]

for vertex 9
if $d[9] > d[8] + w(8, 9) \Rightarrow False$

for vertex 3

if $d[3] > d[8] + w(8, 3) \Rightarrow False$

for vertex 7

if $d[7] > d[8] + w(8, 7) \Rightarrow True$

$d[7] = d[8] + w(8, 7) = 2$

$previous[7] = 8$

$T = \{ e(1, 2), e(1, 8), e(1, 9), e(2, 3), e(8, 7) \}$

while $Q \neq \phi \Rightarrow True$

d[1] = 0


$u = extractMin(Q) = 9 \Rightarrow Q = \{ 3, 4, 5, 6, 7 \}$

$S = \{ 1, 2, 8, 9 \}$

for vertex 1

if $d[1] > d[9] + w(9, 1) \Rightarrow False$

for vertex 8

if $d[8] > d[9] + w(9, 8) \Rightarrow False$

while $Q \neq \phi \Rightarrow True$

d[1] = 0


\[ u = \text{extractMin}(Q) = 3 \Rightarrow Q = \{4, 5, 6, 7\} \]
\[ S = \{1, 2, 8, 9, 3\} \]

for vertex 2
\[
\text{if } d[2] > d[3] + w(3, 2) \Rightarrow \text{False}
\]

for vertex 7
\[
\text{if } d[7] > d[3] + w(3, 7) \Rightarrow \text{False}
\]

for vertex 8
\[
\text{if } d[8] > d[3] + w(3, 8) \Rightarrow \text{False}
\]

for vertex 4
\[
\text{if } d[4] > d[3] + w(3, 4) \Rightarrow \text{True}
\]
\[ d[4] = d[3] + w(3, 4) = 3 \]
\[ \text{previous}[4] = 3 \]
\[ T = \{e(1, 2), e(1, 8), e(1, 9), e(2, 3), e(8, 7), e(3, 4)\} \]

for vertex 5
\[
\text{if } d[5] > d[3] + w(3, 5) \Rightarrow \text{True}
\]
\[ \text{previous}[5] = 3 \]
\[ T = \{e(1, 2), e(1, 8), e(1, 9), e(2, 3), e(8, 7), e(3, 4), e(3, 5)\} \]

while \( Q \neq \emptyset \Rightarrow \text{True} \)
\[ d[1] = 0 \]
\[ d[3] = 2, d[7] = 2 \]
\[ d[6] = \infty \]
\( u = \text{extractMin}(Q) = 7 \Rightarrow Q = \{4, 5, 6\} \)

\( S = \{1, 2, 8, 9, 3, 7\} \)

for vertex 3

if \( d[3] > d[7] + w(7, 3) \) \( \Rightarrow \) False

for vertex 8

if \( d[8] > d[7] + w(7, 8) \) \( \Rightarrow \) False

for vertex 5

if \( d[5] > d[7] + w(7, 5) \) \( \Rightarrow \) False

for vertex 6

if \( d[6] > d[7] + w(7, 6) \) \( \Rightarrow \) False

\[ d[6] = d[7] + w(7, 6) = 3 \]

\[ \text{previous}[6] = 7 \]

\[ T = \{e(1, 2), e(1, 8), e(1, 9), e(2, 3), e(8, 7), e(3, 4), e(3, 5), e(7, 6)\} \]

while \( Q \neq \phi \) \( \Rightarrow \) True

\[ d[1] = 0 \]


\[ d[3] = 2, d[7] = 2 \]


\( u = \text{extractMin}(Q) = 4 \Rightarrow Q = \{5, 6\} \)

\( S = \{1, 2, 8, 9, 3, 7, 4\} \)

for vertex 3

if \( d[3] > d[4] + w(4, 3) \) \( \Rightarrow \) False

for vertex 6

if \( d[6] > d[4] + w(4, 6) \) \( \Rightarrow \) False
while $Q \neq \emptyset \Rightarrow \text{True}$

d[1] = 0


$u = \text{extractMin}(Q) = 5 \Rightarrow Q = \{6\}$

$S = \{1, 2, 8, 9, 3, 7, 4, 5\}$

for vertex 3

\[
\text{if } d[3] > d[5] + w(5, 3) \Rightarrow \text{False}
\]

for vertex 6

\[
\text{if } d[6] > d[5] + w(5, 6) \Rightarrow \text{False}
\]

for vertex 7

\[
\text{if } d[7] > d[5] + w(5, 7) \Rightarrow \text{False}
\]

while $Q \neq \emptyset \Rightarrow \text{True}$

$u = \text{extractMin}(Q) = 6 \Rightarrow Q = \emptyset$

$S = \{1, 2, 8, 9, 3, 7, 4, 5, 6\}$

for vertex 4

\[
\text{if } d[4] > d[6] + w(6, 4) \Rightarrow \text{False}
\]

for vertex 5

\[
\text{if } d[5] > d[6] + w(6, 5) \Rightarrow \text{False}
\]
for vertex 7

\[
\text{if } d[7] > d[6] + w(6, 7) \Rightarrow \text{False}
\]

Output = \( T = \{e(1, 2), e(1, 8), e(1, 9), e(2, 3), e(8, 7), e(3, 4), e(3, 5), e(7, 6)\} \)

Fig. 24 shows stepwise generation of shortest path tree for the graph shown in Fig. 23 with vertex 1 as source.

Fig. 24 Stepwise generation of Shortest path tree with vertex 1 as source
3.3.3 Proposed Heuristic

In this section we present the heuristic that uses Dijkstra’s single shortest path algorithms (SSSP) to find tree with least value of largest-loaded edge. In the given pseudo-code, each iteration generates a shortest path tree $T_v$ rooted at a particular vertex $v$. Following which we calculate the largest loaded edge $l_{T_v}(e_{\text{max}})$. The heuristic returns the tree $T_i$ such that $l_{T_i}(e_{\text{max}})$ is minimum among all possible trees.

**BSPH(Graph G)**

Tree $M = \emptyset$

For each vertex $v$ in $G$ do

$T_v = \text{SSSP}(G, v)$

if ($M \neq \emptyset$)

if ($l_{T_v}(e_{\text{max}}) < l_{M}(e_{\text{max}})$)

$M = T_v$

else

$M = T_v$

return $M$

Fig. 25 shows single source shortest path trees for different vertices as source and the load of the max edge $e_{\text{max}}$ for each tree. As seen, as per the heuristic the tress with source as vertex 3 and 4 are the desirable result since $l_{T_3}(e_{\text{max}}) = l_{T_4}(e_{\text{max}}) = 14$. 
Fig. 25 SPT for different vertices as source and calculation of $l(e_{\text{max}})$
3.4 Minimum Routing Cost Tree Heuristic (MRCH)

3.4.1 Motivation

In general, when the cost on an edge represents a price for routing messages between its endpoints (such as delays), the routing cost for a pair of vertices in a given spanning tree is defined as the sum of costs of the edges in the unique tree path between them. Routing cost of the tree itself is the sum over all pairs of vertices of the routing costs for the pair in this tree. Minimum Routing Cost Spanning Tree (MRCT) is the one with minimum routing cost among all possible spanning trees.

For trees where cost of each edge is same, Tree with large routing cost may have comparatively larger number of longer paths. As the path length increases, so does load on the inner most edge of the path. In this way, maximum edge spanning tree seems to be a special case of Minimum routing cost spanning tree, where cost of all edges are same. Hence, in the following heuristic we use MRCT.

3.4.2 Algorithm

David. S. Johnson et al. [4] showed that MRCT problem on a general graph is NP-hard. Wu and Chao [3] showed that there exists a shortest-path tree which is a 2-approximation of an MRCT. Such a shortest-path tree is rooted at the median of the given graph.

Median of a graph $G = (V, E)$ is a vertex $v \in V$, such that the aggregate of shortest distances between $v$ and all other vertices in $G$ is minimum among all the vertices of $G$, i.e. $v$ is median if it has minimum $\sum_{w=1}^{n} S_{d}(v, w)$ where $v, w \in V$ and $S_{d}(v, w)$ is shortest path between $v$
and \( w \) in \( G \). In order to calculate shortest path between a source and remaining vertices we make use of Dijkstra’s single source shortest path algorithm.

### 3.4.3 Proposed Heuristic

This heuristic is similar to the BSPT except that the comparison of the trees generated is on the sum of edges instead of the value of the largest-loaded edge in it.

\[
\text{MRCH(Graph } G \text{)}
\]

Tree \( M = \phi \)

For each vertex \( v \) in \( G \) do

\[
T_v = \text{SSSP}(G, v)
\]

if (\( M \neq \phi \))

if (\( \text{totalSum}(T_v) < \text{totalSum}(M) \))

\[
M = T_v
\]

else

\[
M = T_v
\]

return \( M \)

For graph shown in Fig. 23, Fig. 27 shows single source shortest path trees for different vertices as source and calculation of \( \sum_{w=1..n} S_v[w] \) for each vertex as the source. As seen, the trees with source as vertex 3 and 8 result in minimum value of \( \sum_{w=1..n} S_v[w] \), hence we can consider either to be one which is 2-approximation of an MRCT.
Fig. 26 SPT for vertex 8 as source results in 2-Approximation MRCT

Considering tree with vertex 8 as shown in Fig. 26, we have $l_{T8}(e_{max}) = 18$. 

Fig. 27 SPT for different source and calculation of shortest path for each
3.5 Bi-Partite Semi-Matching Tree Heuristic (BSH)

3.5.1 Motivation

Consider the trees as shown in the Fig. 28. Tree A has edge with largest load 30 and Tree B has edge with largest load of 36. Addition of edge to sub-tree $S_2$ in Tree B resulted in extra load on edge between sub-tree $S_2$ and root of the tree. This leads us to believe that by evenly matching available vertices at depth $h$ to those at depth $h-1$ would give better results.

![Fig. 28 Tree A with $l_A(e_{max}) = 30$ & Tree B with $l_B(e_{max}) = 36$](image)

3.5.2 Algorithm

For a Bi-partite graph $G = (V \cup U, E)$ where $E \subseteq U \times V$, we define matching to be subset of edges $M \subseteq E$ such that for all vertices $v \in V$, at most one edge of $M$ is incident on $v$, as given in [5]. We say that a vertex $v \in V$ is matched by matching $M$ if some edge of $M$ is incident on $v$; otherwise $v$ is unmatched.

Consider a Bi-partite graph $G$ shown in Fig. 29. Let $V \cup U$ be the set of vertices, where $V$ and $U$ are disjoint and all edges $E$ go between $V$ and $U$. In the Fig. 29, we have shown $V$ and $U$ in different colors for simplicity. Fig. 30 shows possible matching for it.
$M \subseteq E$ is a semi-matching if each vertex in $U$ is incident with exactly one edge in $M$. For sake of understanding, if we refer vertices in $U$ as tasks and to vertices in $V$ as machines, a semi-matching would give an assignment of each task to a machine that is capable of processing it.

In their paper Harvey, Ladner, Lovász and Tamir [6], they consider problem of fairly matching vertices of $U$ and $V$ and refer to this problem as optimal semi-matching. They present and analyze two algorithms, in one of which the first step is to find initial semi-matching. In this
heuristic, we make use of this first step described below. Input is a bi-partite graph $G$ and output is a matching $M$.

**Initial_Semi_Matching(Graph $G$)**

$M = \emptyset$

Sort $U$ by increasing degree

For each $u \in U$

- Let $S$ be the set of neighbors of $u$ with minimum load i.e. number of edges in semi-matching
- Let $v$ be a member of $S$ such that $\deg(v)$ is minimum
- Add $e(u, v)$ to $M$

End

The Fig. 31 shows the output of the *Initial_Semi_Matching* algorithm given above for the input graph shown in Fig. 29.

![Fig. 31 Graph after initial semi matching](image-url)
Fig. 32 shows step by step execution of the Initial_Semi_Matching algorithm for the graph shown in Fig. 29.

![Fig. 32 Generation of Initial semi matching](image)

### 3.5.3 Proposed Heuristic

First step in this heuristic is to find Best Shortest Path tree for the given graph $G$, for which we make use of $BSPH$ we proposed earlier. With height of tree as $h$, next step involves series of iteration $i$, for $0 \leq i < h$, where we consider bi-partite graph $G'$ formed by set of vertices at height $[h-i]$ and $[h-(i+1)]$ and edges between these vertices from $G$, and find $Initial_Semi_Matching$ for it. The heuristic is given below.

**BSH(Graph G)**

$T = \phi$
\[ T' = BSPH(G) \]

\[ h = \text{height}(T') \]

for \( i = 0 \) to \( h \)

\[ U = \text{Set of vertices at height } [h - i] \]

\[ V = \text{Set of vertices at height } [h - (i + 1)] \]

Create Bi-partite graph \( G' = (U \cup V, E') \), where \( E' = \{ e(u, v): u \in U, v \in V \} \)

\[ M = \text{Initial\_Semi\_Matching}(G') \]

\[ T = T \cup M \]

Since this heuristic uses \( BSPH \), we make use of one of the trees constructed by \( BSPH \) as shown in Fig. 33 with the input as the graph shown in Fig. 23 to explain generation of tree with least value of maximum edge using \( BSII \).

---

**Fig. 33 Tree generated by BSPH**

---

**Fig. 34 Tree with vertices labeled with their heights**
Considering vertices at height 2 and at height 1, heuristic generates bi-partite graph as shown in Fig. 35, and the output of \textit{Initial\_Semi\_Matching} for it is shown in Fig. 36.

![Fig. 35 Bi-graph formed by taking edges between vertices at height 2 and 1](image)

![Fig. 36 Output of Initial\_Semi\_Matching on Bi-partite graph shown in Fig. 35](image)

Now in similar way considering vertices at height 1 and at height 0 we have bi-partite graph and its resulting \textit{Initial\_Semi\_Matching} shown in Fig. 37 and Fig. 38.

![Fig. 37 Bi-graph formed by taking edges between vertices at height 1 and 0](image)

![Fig. 38 Output of Initial\_Semi\_Matching on Bi-graph shown in Fig. 37](image)
The tree shown in Fig. 39, generated by the heuristic is union of the semi-matchings shown in Fig. 38 and Fig. 36. For the tree shown in Fig. 39 we have $l(e_{\text{max}}) = 14$.

![Fig. 39 Tree generated by union of trees shown in Fig. 38 and Fig. 36](image)

In this chapter we discussed various algorithms used in the heuristics we propose in this thesis. In the following chapter we continue with discussion on results of these heuristics when the input is a large graph based on models representing network topologies generated by popular models like Waxman and Power law.
CHAPTER 4

EXPERIMENT SETUP AND CONCLUSION

In previous chapters we defined maximum edge of a tree. We also proposed and discussed heuristics we use in this thesis to find spanning tree minimizing the maximum edge load. In this chapter, we describe our experiment environment and results we obtain from the proposed heuristics. Our experiments are based on simulations, which are based on network topologies generated using Waxman [12] and Power-law [11] models and on random topologies.

4.1 Software Description

The simulation software takes as input parameters of the graph which is to be constructed and generates spanning tree using the heuristics suggested. It’s developed using C#.NET, and uses MS-Access as backend to store its results. The topologies generated and trees resulting by running the heuristics on those topologies are stored in respective xml files. In this section we go through series of screen shots, and intend to present navigability of this application.

4.1.1 Main Screen

The main screen presents the user with a client area where the user can create a graph and see output of the heuristic or heuristics selected. The menu bar of the main screen has different
options such as “Show”, “Generate Trees For” and “Run”. Each of these options is described ahead in this section.

4.1.2 User Input

The client area becomes the drawing board for the user. A mouse click would generate a labeled vertex. If previous mouse click was on a vertex, and the user clicks a different vertex, then an edge is added between the two.
The screen shot above shows the vertices generated by clicking on the client area. The next screen shot user adds edge to the above screen.

Fig. 41 Adding vertices to create graph

Fig. 42 Connecting vertices to form an edge
Continuing this way, the user can generate whole graph as shown in Fig 43. Once the graph is generated, the user may want to run a particular heuristic as shown in Fig 44.

Fig. 43 Forming a graph

Fig. 44 Selecting a heuristic to run
A link appears at the top left corner of the client area, describing the output.

![Image](image1.png)

*Fig. 45 Load on emax shown as a link on top left corner of screen*

Once the user clicks on it the resulting tree is shown below.

![Image](image2.png)

*Fig. 46 Tree generated by selected heuristic*
User can use ‘Clear Screen’ option under ‘Show’ menu option, to clear the user work area.

4.1.3 Simulation of Graph Models

The menu bar contains option labeled “Generate Trees For”, which lists all the graph topologies we have considered in this thesis. The screen shot below shows them all.

![Fig. 47 Selection graph model to simulate](image)

Clicking on either of them opens a window which asks to enter parameters used to generate selected graph type. For example screen below asks user to input vertex count and density of the random graphs the application would generate, and run the heuristics on.
Fig. 48 Enter input parameters for Random graphs

Similarly, below we show input screens for topologies based on Waxman model and Power-law model.

Fig. 49 Enter input parameters for Waxman graphs
4.1.4 Output Comparison Charts

Once the application processes all the generated topologies as per the user input, the user can view a statistical graph for comparing output from different heuristic. In order to do so, the user can click on the menu option “Comparison Charts” under the main menu option “Show”. This opens the window as shown in the following figure, in which the user can make appropriate selections to make comparisons using the statistical.
As an example, in the following figure we view such a chart, that shows bar graph with ids of input graphs along its x-axis and load on largest-loaded edge of resulting tree generated by each heuristic along its y-axis. Each colored bar represents output for a particular heuristic, as represented in the legend.

The charts are created using a third party component ChartDirector [10].
4.2 Random Model

In this model, different topologies are generated by varying \( n \), number of nodes in a graph and desired density \( d \), of the network graph. Edges are added between randomly selected pairs of nodes until the desired density is achieved.

For our experimental purposes, we generate 10 graphs of varying size starting with \( 40 < n \leq 80 \), with density of \( d = 1, 2 \) and 3.

4.2.1 Experimental Results

For Random Graphs of size \( n = 80 \) and density \( d = 1 \).
For Random Graphs of size $n = 80$ and density $d = 2$.

For Random Graphs of size $n = 80$ and density $d = 3$. 
Observation:

1. *BSH* outperforms other heuristics.

2. Results from *BSPH* & *MRCH* are comparable, though among the two *BSPH* always performs better.

3. From Appendix A we see that on an average *MLTH*, *MRCH* and *BSPH* fetches result that is 2.57 times, 1.67 times and 1.49 times poor as compared to *BSH*.

4. On an average, with increase in density the result as returned by all the heuristics show improvement.

For Random Graphs of size $n = 70$ and density $d = 1$. 
For Random Graphs of size $n = 70$ and density $d = 2$.

For Random Graphs of size $n = 70$ and density $d = 3$. 
Observation:

1. BSH outperforms other heuristics.

2. Results from BSPH & MRCH are comparable, though among the two BSPH always performs better.

3. From Appendix A we see that on an average MLTH, MRCH and BSPH fetches result that is 2.66 times, 1.61 times and 1.38 times poor as compared to BSH.

4. On an average, with increase in density the results returned by all the heuristics show improvement.

For Random Graphs of size $n = 60$ and density $d = 1$. 
For Random Graphs of size $n = 60$ and density $d = 2$.

For Random Graphs of size $n = 60$ and density $d = 3$.
**Observation:**

1. *BSH* outperforms other heuristics.

2. Results from *BSPH* & *MRCH* are comparable, though among the two *BSPH* always performs better.

3. From Appendix A we see that on an average *MLTH, MRCH* and *BSPH* fetches result that is 2.61 times, 1.57 times and 1.45 times poor as compared to *BSH*.

4. On an average, with increase in density the results returned by all the heuristics show improvement.

For Random Graphs of size $n = 50$ and density $d = 1$. 
For Random Graphs of size $n = 50$ and density $d = 2$.

For Random Graphs of size $n = 50$ and density $d = 3$. 
Observation:

1. BSH outperforms other heuristics.
2. Results from BSPH & MRCH are comparable, though among the two BSPH always performs better.
3. From Appendix A we see that on an average MLTH, MRCH and BSPH fetches result that is 2.29 times, 1.53 times and 1.39 times poor as compared to BSH.
4. On an average, with increase in density the results returned by all the heuristics show improvement.

4.2.2 Analysis

For a given Random graph, BSH always outperforms other heuristics. On the other hand MLTH on an average always performs worse. Increasing the density of input graph increases the number of edges in the graph. For same set of vertices, with increase in number of edges, each
heuristic has better chance to form desirable better output tree. Hence, we see improvement in the average result we get from each heuristic by varying density of the input graph.

4.3 Waxman Model

The networking literature contains variety of random methods to model inter-networks. All are variations of the same basic method: a set of vertices distributed in a plane, and an edge is added between each pair of vertices with some probability. Topologies generated using Waxman model, add edge from \( u \) to \( v \) based on the probability given by: 
\[
P(u, v) = \alpha e^{-d / (\beta \cdot L)}
\]
where \( 0 < \alpha, \beta \leq 1 \), \( d \) is Euclidean distance from \( u \) to \( v \) and \( L \) is maximum distance between any two nodes. Increase in \( \alpha \) increases number of edges in the graph, where as increase in \( \beta \) increases ratio of long edges to short edges [8].

For our experimental purposes, we start with the values \( \alpha = 0.2 \) and \( \beta = 0.2 \), we fix the value of \( \alpha \) and increase the value of gradually to simulate dense network with average low degree. Then, we repeat the same process but by fixing the value of \( \beta \) and increasing the value of \( \alpha \) to simulate networks with short links and average high degree. We generate graphs of different size \( n \) by varying \( \alpha \) and \( \beta \) among values 0.2, 0.5 and 0.8.

4.3.1 Experimental Results

For Waxman graphs of size \( n = 80 \), \( \alpha = 0.2 \) and \( \beta = 0.2 \)
For Waxman graphs of size $n = 80$, $\alpha = 0.2$ and $\beta = 0.5$

For Waxman graphs of size $n = 80$, $\alpha = 0.2$ and $\beta = 0.8$
For Waxman graphs of size $n = 80$, $\alpha = 0.5$ and $\beta = 0.2$

For Waxman graphs of size $n = 80$, $\alpha = 0.5$ and $\beta = 0.5$
For Waxman graphs of size $n = 80$, $\alpha = 0.5$ and $\beta = 0.8$

For Waxman graphs of size $n = 80$, $\alpha = 0.8$ and $\beta = 0.2$
For Waxman graphs of size $n = 80$, $\alpha = 0.8$ and $\beta = 0.5$

For Waxman graphs of size $n = 80$, $\alpha = 0.8$ and $\beta = 0.8$
Observation:

1. BSH outperforms other heuristics.
2. Results from BSPH & MRCH are comparable, though among the two BSPH always performs better.
3. From Appendix B we see that on an average MLTH, MRCH and BSPH fetches result that is 2.57 times, 1.61 times and 1.41 times poor as compared to BSH.
4. Stepwise increase in values of $\alpha$ and $\beta$ does not show any proportionate impact on output of the heuristics.

For Waxman graphs of size $n = 70$, $\alpha = 0.2$ and $\beta = 0.2$
For Waxman graphs of size $n = 70$, $\alpha = 0.2$ and $\beta = 0.5$

For Waxman graphs of size $n = 70$, $\alpha = 0.2$ and $\beta = 0.8$
For Waxman graphs of size $n = 70$, $\alpha = 0.5$ and $\beta = 0.2$

For Waxman graphs of size $n = 70$, $\alpha = 0.5$ and $\beta = 0.5$
For Waxman graphs of size $n = 70$, $\alpha = 0.5$ and $\beta = 0.8$

For Waxman graphs of size $n = 70$, $\alpha = 0.8$ and $\beta = 0.2$
For Waxman graphs of size $n = 70$, $\alpha = 0.8$ and $\beta = 0.5$

For Waxman graphs of size $n = 70$, $\alpha = 0.8$ and $\beta = 0.8$
**Observation:**

1. BSH outperforms other heuristics.
2. Results from BSPH & MRCH are comparable, though among the two BSPH always performs better.
3. From Appendix B we see that on an average MLTH, MRCH and BSPH fetches result that is 2.66 times, 1.68 times and 1.48 times poor as compared to BSH.
4. Stepwise increase in values of $\alpha$ and $\beta$ does not show any proportionate impact on output of the heuristics.

For Waxman graphs of size $n = 60$, $\alpha = 0.2$ and $\beta = 0.2$
For Waxman graphs of size $n = 60$, $\alpha = 0.2$ and $\beta = 0.5$

For Waxman graphs of size $n = 60$, $\alpha = 0.2$ and $\beta = 0.8$
For Waxman graphs of size $n = 60$, $\alpha = 0.5$ and $\beta = 0.2$

For Waxman graphs of size $n = 60$, $\alpha = 0.5$ and $\beta = 0.5$
For Waxman graphs of size $n = 60$, $\alpha = 0.5$ and $\beta = 0.8$

For Waxman graphs of size $n = 60$, $\alpha = 0.8$ and $\beta = 0.2$
For Waxman graphs of size \( n = 60 \), \( \alpha = 0.8 \) and \( \beta = 0.5 \)

For Waxman graphs of size \( n = 60 \), \( \alpha = 0.8 \) and \( \beta = 0.8 \)
Observation:

1. **BSH** outperforms other heuristics.

2. Results from **BSPH** & **MRCH** are comparable, though among the two **BSPH** always performs better.

3. From Appendix B we see that on an average **MLTH**, **MRCH** and **BSPH** fetches result that is 2.53 times, 1.62 times and 1.43 times poor as compared to **BSH**.

4. Stepwise increase in values of $\alpha$ and $\beta$ does not show any proportionate impact on output of the heuristics.

For Waxman graphs of size $n = 50$, $\alpha = 0.2$ and $\beta = 0.2$
For Waxman graphs of size $n = 50$, $\alpha = 0.2$ and $\beta = 0.5$

For Waxman graphs of size $n = 50$, $\alpha = 0.2$ and $\beta = 0.8$
For Waxman graphs of size $n = 50$, $\alpha = 0.5$ and $\beta = 0.2$

For Waxman graphs of size $n = 50$, $\alpha = 0.5$ and $\beta = 0.5$
For Waxman graphs of size $n = 50$, $\alpha = 0.5$ and $\beta = 0.8$

For Waxman graphs of size $n = 50$, $\alpha = 0.8$ and $\beta = 0.2$
For Waxman graphs of size $n = 50$, $\alpha = 0.8$ and $\beta = 0.5$

For Waxman graphs of size $n = 50$, $\alpha = 0.8$ and $\beta = 0.8$
Observation:

1. BSH outperforms other heuristics.

2. Results from BSPH & MRCH are comparable, though among the two BSPH always performs better.

3. From Appendix B we see that on an average MLTH, MRCH and BSPH fetches result that is 2.42 times, 1.58 times and 1.42 times poor as compared to BSH.

4. Stepwise increase in values of $\alpha$ and $\beta$ does not show any proportionate impact on output of the heuristics.

4.3.2 Analysis

For a given Waxman graph, BSH always outperforms other heuristics. On the other hand MLTH on an average always performs worse. Stepwise increase in values of $\alpha$ and $\beta$ does not show any proportionate impact on output of the heuristics. Increase in value of $\alpha$ might make one
to expect better result from each heuristic because of increase in number of edges, but it’s not the case because many combinations of $\alpha$ and $\beta$ are shown to give same number of edges [11].

4.4 Power-law Model

Power-law model can be used to generate router-level topologies [7]. For our experimental purposes, we generate Power-law topologies using BRITE [9], which is the best known Power-law-based topology generator. BRITE generates different topologies by changing the values of the following parameters [14]:

1. $HS$, Size of one side of the plane,
2. $LS$, Size of one side of a high level square,
3. $NP$, Node Placement,
4. $m$, Number of links added per new node and
5. $IG$, Incremental Growth.

For experimental purposes, we generate set of 10 graphs for varying $m$ and $n$. Keeping rest of the parameters same $m$ takes values of 2, 4 and 6 as described in Appendix C.

4.4.1 Experimental Results

For Power-law Graphs of size $n = 80$, $HS = 20$, $LS = 20$, $NP = 2$ and $m = 2$. 
For Power-law Graphs of size $n = 80$, $HS = 20$, $LS = 20$, $NP = 2$ and $m = 4$.

For Power-law Graphs of size $n = 80$, $HS = 20$, $LS = 20$, $NP = 2$ and $m = 6$. 

90
Observation:

1. BSH outperforms other heuristics.

2. Results from BSPH & MRCH are comparable, though among the two BSPT always performs better.

3. From Appendix C we see that on an average MLTH, MRCH and BSPH fetches result that is 4.24 times, 1.47 times and 1.38 times poor as compared to BSH.

4. Unlike other heuristics with increase in $m$, the relative performance of MLTH compared to BSH tends to worsen.

For Power-law Graphs of size $n = 70$, $HS = 20$, $LS = 20$, $NP = 2$ and $m = 2$. 
For Power-law Graphs of size \( n = 70, HS = 20, LS = 20, NP = 2 \) and \( m = 4 \).

For Power-law Graphs of size \( n = 70, HS = 20, LS = 20, NP = 2 \) and \( m = 6 \).
**Observation:**

1. Bi-partite semi matching heuristic outperforms other heuristics.

2. Results from BSPT & MRCT heuristic are comparable, though among the two BSPT always performs better.

3. From Appendix C we see that on an average *MLTH, MRCH* and *BSPH* fetches result that is 3.88 times, 1.43 times and 1.32 times poor as compared to *BSH*.

4. Unlike other heuristics with increase in m, the relative performance of *MLTH* compared to *BSH* tends to worsen.

For Power-law Graphs of size $n = 60$, $HS = 20$, $LS = 20$, $NP = 2$ and $m = 2$
For Power-law Graphs of size $n = 60$, $HS = 20$, $LS = 20$, $NP = 2$ and $m = 4$.

For Power-law Graphs of size $n = 60$, $HS = 20$, $LS = 20$, $NP = 2$ and $m = 6$. 
Observation:

1. Bi-partite semi matching heuristic outperforms other heuristics.

2. Results from BSPT & MRCT heuristic are comparable, though among the two BSPT always performs better.

3. From Appendix C we see that on an average MLTH, MRCH and BSPH fetch results that is 3.64 times, 1.44 times and 1.32 times poor as compared to BSH.

4. Unlike other heuristics with increase in m, the relative performance of MLTH compared to BSH tends to worsen.

For Power-law Graphs of size $n = 50$, $HS = 20$, $LS = 20$, $NP = 2$ and $m = 2$. 
For Power-law Graphs of size $n = 50$, $HS = 20$, $LS = 20$, $NP = 2$ and $m = 4$.

For Power-law Graphs of size $n = 50$, $HS = 20$, $LS = 20$, $NP = 2$ and $m = 6$. 

96
Observation:

1. Bi-partite semi matching heuristic outperforms other heuristics.

2. Results from BSPT & MRCT heuristic are comparable, though among the two BSPT always performs better.

3. From Appendix C we see that on an average $MLTH$, $MRCH$ and $BSPH$ fetches result that is 3.42 times, 1.58 times and 1.41 times poor as compared to $BSH$.

4. Unlike other heuristics with increase in $m$, the relative performance of $MLTH$ compared to $BSH$ tends to worsen.

4.4.2 Analysis

As mentioned earlier Power-law model is used to generate router level topology. Inherently graphs based on Power-law model are not much dense because of which increase in $m$ does not have significant effect on result from $MLTH$. Whereas result from other heuristics mostly
keeps getting better i.e. unlike other heuristics with increase in $m$, the relative performance of MLTH tends to worsen.

4.5 Conclusion

In this thesis we first propose a new kind of problem of finding spanning tree minimizing the maximum edge load. We show effect of topology of resulting tree on its resulting largest loaded edge. We propose four different heuristics for our problem such as Max-Leaf Tree Heuristic (MLTH), Best Shortest Path Heuristic (BSPH), MRCT Heuristic (MRCH) and Bi-partite Semi Matching Heuristic (BSH) and discuss motivation behind each approach.

Comparing the results obtained we see that BSH outperforms other heuristics and MLTH on an average fetches worst result. We also see that on the average, increase in number of edges of input graph i.e. density, improves the output of each heuristic.

Future work may concentrate on making enhancements to current heuristics and exploring other ways solving the problem. BSH generates tree by adding edges between vertices at different heights of a BSPT for a given graph. We can improve the performance by making decision based on weight of sub-trees rooted at vertices which are lower in height.

In our work, we have not made any attempt to prove NP-completeness of the problem. Presenting such a proof and designing approximation algorithms for it would be worth exploring.
APPENDIX A:
Average maximum edge load with Random graph as input

<table>
<thead>
<tr>
<th>Column Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Average of maximum edge load of the spanning tree resulting using 3 – Approx Max leaf Algorithm</td>
</tr>
<tr>
<td>B</td>
<td>Average of maximum edge load of the spanning tree resulting using 2 – Approx MRCT</td>
</tr>
<tr>
<td>C</td>
<td>Average of maximum edge load of the spanning tree resulting using Best Shortest Path Heuristic</td>
</tr>
<tr>
<td>D</td>
<td>Average of maximum edge load of the spanning tree resulting using Bi-partite Heuristic</td>
</tr>
</tbody>
</table>

Each row in the following table gives average of the result each heuristic fetches over set of 10 input graphs for given vertex count and density. Heuristics are labeled A, B, C and D as described in table above.
<table>
<thead>
<tr>
<th>Vertex Count</th>
<th>Density</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>1</td>
<td>376.1</td>
<td>282.7</td>
<td>251.1</td>
<td>197</td>
</tr>
<tr>
<td>40</td>
<td>2</td>
<td>356.9</td>
<td>219.7</td>
<td>191.2</td>
<td>143.4</td>
</tr>
<tr>
<td>40</td>
<td>3</td>
<td>216.3</td>
<td>89</td>
<td>85.7</td>
<td>68.4</td>
</tr>
<tr>
<td>50</td>
<td>1</td>
<td>598.4</td>
<td>435.1</td>
<td>398.4</td>
<td>296.5</td>
</tr>
<tr>
<td>50</td>
<td>2</td>
<td>467.5</td>
<td>336.4</td>
<td>300</td>
<td>200.4</td>
</tr>
<tr>
<td>50</td>
<td>3</td>
<td>278.3</td>
<td>158.7</td>
<td>147</td>
<td>108.9</td>
</tr>
<tr>
<td>60</td>
<td>1</td>
<td>846.5</td>
<td>536.6</td>
<td>453.3</td>
<td>313.4</td>
</tr>
<tr>
<td>60</td>
<td>2</td>
<td>729.5</td>
<td>505.2</td>
<td>486.3</td>
<td>313.4</td>
</tr>
<tr>
<td>60</td>
<td>3</td>
<td>434.7</td>
<td>219.8</td>
<td>209.8</td>
<td>153.3</td>
</tr>
<tr>
<td>70</td>
<td>1</td>
<td>1182.5</td>
<td>702.8</td>
<td>630.4</td>
<td>455.9</td>
</tr>
<tr>
<td>70</td>
<td>2</td>
<td>1095.2</td>
<td>795.7</td>
<td>612.7</td>
<td>456.9</td>
</tr>
<tr>
<td>70</td>
<td>3</td>
<td>632</td>
<td>327.7</td>
<td>304.2</td>
<td>210.6</td>
</tr>
<tr>
<td>80</td>
<td>1</td>
<td>1529.2</td>
<td>947.7</td>
<td>795.5</td>
<td>538.7</td>
</tr>
<tr>
<td>80</td>
<td>2</td>
<td>1378.9</td>
<td>1021.6</td>
<td>916.9</td>
<td>619.7</td>
</tr>
<tr>
<td>80</td>
<td>3</td>
<td>799.5</td>
<td>489.8</td>
<td>470.3</td>
<td>301.6</td>
</tr>
</tbody>
</table>
APPENDIX B:
Average maximum edge load with Waxman graph as input

<table>
<thead>
<tr>
<th>Column Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Average of maximum edge load of the spanning tree resulting using 3 – Approx Max leaf Algorithm</td>
</tr>
<tr>
<td>B</td>
<td>Average of maximum edge load of the spanning tree resulting using 2 – Approx MRCT</td>
</tr>
<tr>
<td>C</td>
<td>Average of maximum edge load of the spanning tree resulting using Best Shortest Path Heuristic</td>
</tr>
<tr>
<td>D</td>
<td>Average of maximum edge load of the spanning tree resulting using Bi-partite Heuristic</td>
</tr>
</tbody>
</table>

Each row in the following table gives average of the result each heuristic fetches over set of 10 input graphs for given vertex count, $\alpha$ and $\beta$. Heuristics are labeled A, B, C and D as described in table above.
<table>
<thead>
<tr>
<th>Vertex Count</th>
<th>$\alpha$</th>
<th>$\beta$</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>0.2</td>
<td>0.2</td>
<td>480.3</td>
<td>281.7</td>
<td>237.1</td>
<td>164.4</td>
</tr>
<tr>
<td>50</td>
<td>0.2</td>
<td>0.5</td>
<td>529.2</td>
<td>366.1</td>
<td>314.1</td>
<td>225.5</td>
</tr>
<tr>
<td>50</td>
<td>0.2</td>
<td>0.8</td>
<td>523.4</td>
<td>350</td>
<td>305.4</td>
<td>223.4</td>
</tr>
<tr>
<td>50</td>
<td>0.5</td>
<td>0.2</td>
<td>503.3</td>
<td>326.3</td>
<td>299.9</td>
<td>207</td>
</tr>
<tr>
<td>50</td>
<td>0.5</td>
<td>0.5</td>
<td>451.8</td>
<td>305.4</td>
<td>262.3</td>
<td>181.6</td>
</tr>
<tr>
<td>50</td>
<td>0.5</td>
<td>0.8</td>
<td>492.8</td>
<td>301.3</td>
<td>278.2</td>
<td>211.6</td>
</tr>
<tr>
<td>50</td>
<td>0.8</td>
<td>0.2</td>
<td>554.5</td>
<td>367.6</td>
<td>327.8</td>
<td>223.4</td>
</tr>
<tr>
<td>50</td>
<td>0.8</td>
<td>0.5</td>
<td>433.8</td>
<td>274.9</td>
<td>260.9</td>
<td>182</td>
</tr>
<tr>
<td>50</td>
<td>0.8</td>
<td>0.8</td>
<td>480.7</td>
<td>350.4</td>
<td>330.5</td>
<td>224.7</td>
</tr>
<tr>
<td>60</td>
<td>0.2</td>
<td>0.2</td>
<td>695.1</td>
<td>424.3</td>
<td>382</td>
<td>293.5</td>
</tr>
<tr>
<td>60</td>
<td>0.2</td>
<td>0.5</td>
<td>784.4</td>
<td>534.4</td>
<td>471.8</td>
<td>294</td>
</tr>
<tr>
<td>60</td>
<td>0.2</td>
<td>0.8</td>
<td>715.7</td>
<td>386.9</td>
<td>355.9</td>
<td>260.8</td>
</tr>
<tr>
<td>60</td>
<td>0.5</td>
<td>0.2</td>
<td>699.1</td>
<td>459.6</td>
<td>377.5</td>
<td>261.2</td>
</tr>
<tr>
<td>60</td>
<td>0.5</td>
<td>0.5</td>
<td>703.4</td>
<td>468.7</td>
<td>418.6</td>
<td>297.7</td>
</tr>
<tr>
<td>60</td>
<td>0.5</td>
<td>0.8</td>
<td>637.4</td>
<td>477.3</td>
<td>415.1</td>
<td>297.4</td>
</tr>
<tr>
<td>60</td>
<td>0.8</td>
<td>0.2</td>
<td>738.9</td>
<td>448</td>
<td>409</td>
<td>275.9</td>
</tr>
<tr>
<td>60</td>
<td>0.8</td>
<td>0.5</td>
<td>637</td>
<td>416.5</td>
<td>379.6</td>
<td>257.1</td>
</tr>
<tr>
<td>60</td>
<td>0.8</td>
<td>0.8</td>
<td>735.6</td>
<td>469.1</td>
<td>385.9</td>
<td>276.7</td>
</tr>
<tr>
<td>70</td>
<td>0.2</td>
<td>0.2</td>
<td>1000.2</td>
<td>542.7</td>
<td>509.6</td>
<td>351.1</td>
</tr>
<tr>
<td>70</td>
<td>0.2</td>
<td>0.5</td>
<td>1018.7</td>
<td>617.6</td>
<td>563.5</td>
<td>388.2</td>
</tr>
<tr>
<td>70</td>
<td>0.2</td>
<td>0.8</td>
<td>990.2</td>
<td>702.8</td>
<td>624.9</td>
<td>415.2</td>
</tr>
<tr>
<td>70</td>
<td>0.5</td>
<td>0.2</td>
<td>951.2</td>
<td>577.2</td>
<td>508.6</td>
<td>346</td>
</tr>
<tr>
<td>70</td>
<td>0.5</td>
<td>0.5</td>
<td>970.2</td>
<td>621.7</td>
<td>542.5</td>
<td>380.6</td>
</tr>
<tr>
<td>70</td>
<td>0.5</td>
<td>0.8</td>
<td>950.9</td>
<td>577.4</td>
<td>516.3</td>
<td>329.3</td>
</tr>
<tr>
<td>70</td>
<td>0.8</td>
<td>0.2</td>
<td>968.9</td>
<td>658.7</td>
<td>546.9</td>
<td>358.4</td>
</tr>
<tr>
<td>70</td>
<td>0.8</td>
<td>0.5</td>
<td>1078.6</td>
<td>745.6</td>
<td>618.7</td>
<td>424.8</td>
</tr>
<tr>
<td>70</td>
<td>0.8</td>
<td>0.8</td>
<td>1071.3</td>
<td>667.8</td>
<td>605.2</td>
<td>400.5</td>
</tr>
<tr>
<td>80</td>
<td>0.2</td>
<td>0.2</td>
<td>1331.6</td>
<td>796.3</td>
<td>693.9</td>
<td>537.1</td>
</tr>
<tr>
<td>80</td>
<td>0.2</td>
<td>0.5</td>
<td>1330.6</td>
<td>927.3</td>
<td>805.3</td>
<td>533.8</td>
</tr>
<tr>
<td>80</td>
<td>0.2</td>
<td>0.8</td>
<td>1359.9</td>
<td>756.6</td>
<td>661.2</td>
<td>445.3</td>
</tr>
<tr>
<td>80</td>
<td>0.5</td>
<td>0.2</td>
<td>1457.6</td>
<td>918.3</td>
<td>798.4</td>
<td>606.7</td>
</tr>
<tr>
<td>80</td>
<td>0.5</td>
<td>0.5</td>
<td>1153.7</td>
<td>738.6</td>
<td>662.6</td>
<td>476.8</td>
</tr>
<tr>
<td>80</td>
<td>0.5</td>
<td>0.8</td>
<td>1269.7</td>
<td>835.9</td>
<td>723.4</td>
<td>486.5</td>
</tr>
<tr>
<td>80</td>
<td>0.8</td>
<td>0.2</td>
<td>1310.2</td>
<td>918.9</td>
<td>803</td>
<td>567.9</td>
</tr>
<tr>
<td>80</td>
<td>0.8</td>
<td>0.5</td>
<td>1417.9</td>
<td>809.1</td>
<td>743.6</td>
<td>529.3</td>
</tr>
<tr>
<td>80</td>
<td>0.8</td>
<td>0.8</td>
<td>1293.8</td>
<td>787.5</td>
<td>687.7</td>
<td>476.8</td>
</tr>
</tbody>
</table>
APPENDIX C:
Average maximum edge load with Power-law graph as input

<table>
<thead>
<tr>
<th>Column Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Average of maximum edge load of the spanning tree resulting using 3 – Approx Max leaf Algorithm</td>
</tr>
<tr>
<td>B</td>
<td>Average of maximum edge load of the spanning tree resulting using 2 – Approx MRCT</td>
</tr>
<tr>
<td>C</td>
<td>Average of maximum edge load of the spanning tree resulting using Best Shortest Path Heuristic</td>
</tr>
<tr>
<td>D</td>
<td>Average of maximum edge load of the spanning tree resulting using Bi-partite Heuristic</td>
</tr>
</tbody>
</table>

Each row in the following table gives average of the result each heuristic fetches over set of 10 input graphs varying all but the number of links added per new node. Heuristics are labeled A, B, C and D as described in table above. For all the input graphs, $HS = 20$, $LS = 20$ and $NP = 2$. 
<table>
<thead>
<tr>
<th>Vertex Count</th>
<th>$m$</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>2</td>
<td>598.11</td>
<td>345.58</td>
<td>323.82</td>
<td>274.82</td>
</tr>
<tr>
<td>50</td>
<td>4</td>
<td>589.55</td>
<td>240.33</td>
<td>220</td>
<td>150.55</td>
</tr>
<tr>
<td>50</td>
<td>6</td>
<td>587.5</td>
<td>244.4</td>
<td>203.7</td>
<td>149.6</td>
</tr>
<tr>
<td>60</td>
<td>2</td>
<td>858.3</td>
<td>446</td>
<td>426.5</td>
<td>355.9</td>
</tr>
<tr>
<td>60</td>
<td>4</td>
<td>842.2</td>
<td>320.6</td>
<td>293</td>
<td>213.4</td>
</tr>
<tr>
<td>60</td>
<td>6</td>
<td>873.8</td>
<td>244.2</td>
<td>224</td>
<td>186.9</td>
</tr>
<tr>
<td>70</td>
<td>2</td>
<td>1130.5</td>
<td>594.3</td>
<td>554.2</td>
<td>458.6</td>
</tr>
<tr>
<td>70</td>
<td>4</td>
<td>1191.6</td>
<td>412.2</td>
<td>385.9</td>
<td>257.5</td>
</tr>
<tr>
<td>70</td>
<td>6</td>
<td>1147.7</td>
<td>354.3</td>
<td>324.8</td>
<td>251.4</td>
</tr>
<tr>
<td>80</td>
<td>2</td>
<td>1447.5</td>
<td>695.9</td>
<td>660.1</td>
<td>521.4</td>
</tr>
<tr>
<td>80</td>
<td>4</td>
<td>1490</td>
<td>495.8</td>
<td>449.1</td>
<td>318</td>
</tr>
<tr>
<td>80</td>
<td>6</td>
<td>1528.4</td>
<td>441</td>
<td>427.8</td>
<td>289.4</td>
</tr>
</tbody>
</table>
BIBLIOGRAPHY


