Title: Technical Report

Project Title: Connectivity and coverage of underwater sensor networks

Name: Md Asadul Islam

Date: June 21, 2014

Abstract

1 Introduction

- Interest of monitoring underwater environment is growing day by day. Sensor network is a promising tool for underwater environment monitoring because of its remote monitoring and control technology.
- Underwater Sensor networking is using for military surveillance[29],[52],[38],[51],[55],[7],[6],[39],[21] disaster prevention, assisted navigation, offshore exploration, tsunami monitoring and oceanographic data collection.
- Routing for UWSN is an important aspect. Authors [46],[42],[48],[15],[4] providing much attention for UWSN.
- Coverage and connectivity are two important aspect of underwater sensor network[22]. One can determine the quality of surveillance of a underwater wireless sensor network from network coverage.
- Many to one data flow from a set of sources to a common sink over a tree based routing topology, is a fundamental communication architecture of a underwater sensor networks.
- Some physical layer parameters for UWSN is illustrated by table 2.

Electromagnetic wave and acoustic wave are two main medium of wireless communication [31][41],[54],[23],[11],[10],[35][36]. Both types of communication are using extensively for terrestrial and underwater communication. Optical communication are special form of electromagnetic communication which is also using for underwater wireless communication. In this section we are going to provide an overview of above three types of communication for UWSN.

- Acoustic Communication:
  - Sound wave can travel both in air and water similarly. The speed of sound in water is almost 4.3 times faster than the speed of sound in air which makes acoustic communication attractive medium for underwater data transfer[45][44] [37]
  - Acoustic communication is attractable media because attenuation for acoustic wave is much smaller than electromagnetic wave. Attenuation increases linearly with frequency for acoustic communication[47].

Model	Distance (m)	Data Rate (Kbps)	Frequency (KHz)	Power consumption (Watt)	Weight (Kg)
Link Quest UWM2200 [26]	1000	35.7	53.55-89.25	6	3
Link Quest UWM3000H[26]	6000	5	7.5-12.5	12	4.1
Link Quest UWM4000[26]	4500	8.5	12.75-21.75	7	8.2
Link Quest UWM10000[26]	10000	5	7.5-12.5	40	21
Evo Logics S2C$M$48/78[27]	2000	28	48-78	10-100	12
Evo Logics S2C$M$7/17[27]	8000	3.5	48-78	10-100	7.8
HERMES[5]	120	150	310	32	

Table 1: Comparison of various underwater acoustic modems

  - Sound with higher frequency attenuate much faster than sound with lower frequency. This presents the trade-off between distance and data transfer rate. That is higher frequency sound wave allows to transfer data at higher rate where lower frequency acoustic signal can travel long distance.
  - On the contrary the performance of acoustic communication in shallow water is very poor.
  - Also acoustic transmission is adversely affected by water temperature, salinity, depth and turbidity[9].
  - It is easily visible from table 1 that the data transfer rate of the fastest modem is only a couple of hundred kilo bits. Although acoustic modems are the only means of sending data for long distance in underwater environment, it is not suitable for short distance data communication with high speed.

- Radio Communication:
  - Radio waves are faster than acoustic waves. Even in the water the radio wave can travel at the speed of $3 \times 10^8 ms^{-1}$ which is about 2×10^5 faster than acoustic waves.
  - Radio wave is unaffected by salinity, temperature and depth.
  - Radio wave is attractive choice for data communication in shallow water. Attenuation of radio wave is less in comparison with acoustic wave[47][18]. Noise arises for data communication with radio wave is less in shallow water in comparison with acoustic waves.
  - Radio waves is able to provide high bandwidth (up to 100MBps) for very short distance [9].
  - On the contrary radio waves is susceptible to electromagnetic interference. That is radio wave attenuate highly in water and signal cannot go further.

- Optical Wave:

- Underwater optical communications is capable of transferring data at higher rate in comparison with acoustic communication with lower power consumption and simpler computational complexity[24][12] [13].
- The cost of optical communication is much cheaper than acoustic communication.
- Optical communication is recently using for robust real time video streaming[14].
- On contrary in optical communication nodes need to tightly aligned. Also the communication range is very short.

Parameter Name	Value	Description
Standard	IEEE 802.11	Can be used as medium access control protocol (MAC) for underwater acoustic communication with modification[20].
Range	1m-10km [25]	Short range modem provides very high bandwidth (typically MHz or more) where long range modem provides low bandwidth(several bps)
Data Rates	bps - MBps	Varies from standard to standard from 5 bps to 19200 bps.
Energy Source	battery, external sources	Primarily battery is used as a power source but for most of the standard external power supply can be used as a power supply.
Well-known models	LinQuest Inc, Hydro International	UWM1000,UWM2000,S1510 Radio Modem,Digital Underwater Modem UM 30
Path loss model		Path loss of underwater acoustic communication channel depends on the distance between the transmitter and receiver and signal frequency[43].

Table 2: Physical Layer Parameters

Connectivity is an important issue for UWSN in-order to perform localization [56], [57],[19], routing [34],[53],[30]. In this task we are interested to measure the connectivity of UWSN. We are considering a network consists of anchored sensor nodes or free moving sensor nodes illustrated in fig.1. The connectivity we are interested, can be

- All node connectivity: every node is connected with sink.
- 2-terminal connectivity: is the connectivity that two given vertices, called the source and the sink, can communicate.
- k-terminal connectivity: compute the probability that every operational pair of sites in k-can communicate with each other.

2 Literature Review

In this section, we are going to describe some literature discuss the connectivity and coverage issue of underwater wireless sensor network and some approximation algorithm.

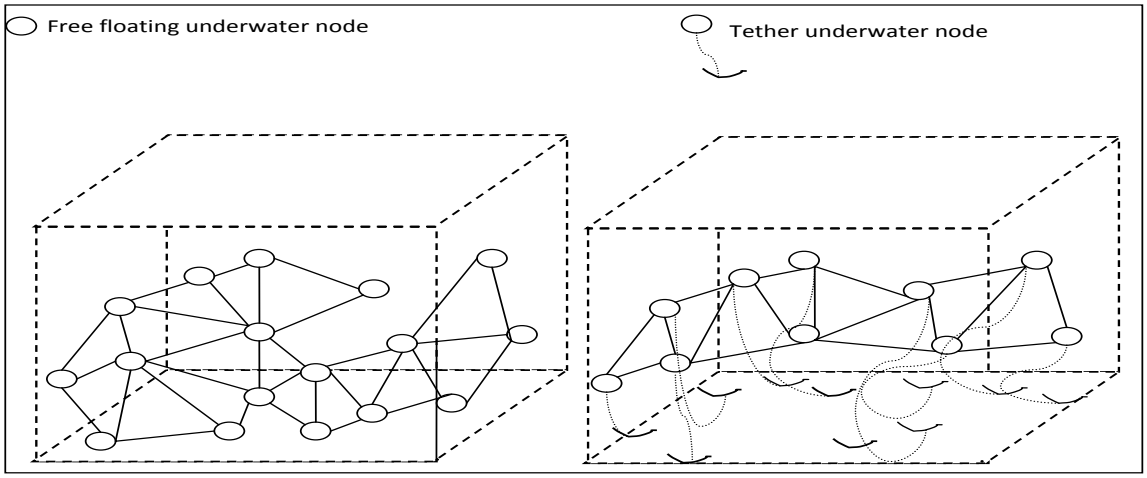

Figure 1: A example of UWSN where nodes are tether and free floating.

- [40] proposes a UWSN distributed node deployment strategy where they drop the nodes on the water surface. Maximize coverage and guaranteed connectivity from initial deployment was their primary goal. They finds the connected dominating set of initial deployment by using using a distributed leader election algorithm at 2-D plane. After that they adjust the depth of all dominatee which maintain the connectivity with dominant node by stretched along z axis as much as possible until their sensing ranges do not overlap to maximize coverage.

- Authors in [17] present a simple mobility model for adhoc routing. They investigate the use of a simple probabilistic model for low speed mobile hosts. Also they examine the combinatorial aspect of accessing the likelihood that a given end-to-end route exists and computes a heuristic solution to an optimization problem that calls for selecting a most probable route that balance the nodal loads. Finally they devise efficient solutions for the above two problem.

- In [1] authors propose a distributed node deployment scheme which can increase the initial network coverage in an iterative basis. They assuming that the nodes are initially deployed at the bottom of the water and can only move in vertical direction in 3-D space. The idea is to relocate the nodes at different depths based on a local agreement in order to reduce the sensing overlaps among the neighbouring nodes. Redundancy is observe by one of the node called Leader. It utilizes vertex colouring problem formulation in-order to determine coverage overlap. The nodes continue to adjust their depths until there is no room for improving their coverage. They consider both tether and and untethered architecture for node deployment.

- Authors in [3] analyse the connectivity and k-coverage issues in 3D WSNs, where each point is covered by at least k sensors. They proposes the Reuleaux tetrahedron model to characterize k-coverage of a $3D$ field and investigate the corresponding minimum sensor spatial density. They compute the connectivity of 3D k-covered WSNs. They prove that $3D$ k-covered WSNs can sustain a large number of sensor failures based

on conditional connectivity and forbidden faulty sensor set. Finally they provide a measure of connectivity and fault tolerance of $3D$ WSNs based on an accurate characterization of k-coverage of $3D$ fields.

- The work of [57] highlighted the localization of a sensor node. According to their model, the moving speed of underwater node changes continuously and shows semi-periodic properties. They predict the future mobility pattern from the past mobility information. The nodes maintain connectivity in-order to provide accurate localization information.

- In [8] authors proposes Meandering Current Mobility Model (MCM) which is able to capture the physical movement of the sensor nodes with ocean currents. In MCM, nodes are moving by the effect of meandering sub-surface currents and vortices. According to MCM, there is a strong correlations between nearby sensors. Vertical movements in ocean are negligible with respect to horizontal ones. Thus, in their model they neglect vertical displacement which makes mobility in 2D. This model is more realistic than other mobility models [28][57] for UWSNs since nodes are drifted according to the movement of the ocean. They studied dynamic coverage and connectivity as a function of time under the MCM model. They also consider the effect of different deployment strategies on network coverage and connectivity.

- Connectivity and coverage are important issue for UWSN. In [2] authors tackling the problem of determining how to deploy minimum number of sensor nodes so that all points inside the network is within the sensing range of at least one sensor and all sensor nodes can communicate with each other, possibly over a multi-hop path. They used sphere-based communication and sensing model. They place a node at the center of each virtual cell created by truncated octahedron-based tessellation. They provide solutions for both limited and full communication redundancy requirements.

- The work of [32], [33] extends [8] and proposes a ring like motion to capture the basic of sea flow. According to this model, there are two types of mobility, uncontrollable mobility which breaks the coverage of sensor network and controllable mobility which restore the coverage of sensor network. In the ring-like model [32], they consider local variety and main circulation in-order to capture some characteristics of water bodies. They use probability in-order to determine the next position of a node.

- In [49], [50] author proposed a polynomial algorithm to solve Steiner tree problem. The Steiner tree problem in an undirected graph is the problem of finding a tree spanning a pre-specified set of vertices at minimum cost where the cost of a tree is equal to the sum of the cost of its edges.

We device a scheme where we approximate the UWSN by using a special types of graph which is called partial k-tree. According to our scheme each node can be located into a set of regions with certain probabilities which is known as probabilistic locality set. The locality set of a node at a certain time instant is known from the initial location of the node and underwater current.

We introduce a problem called the Connectivity in Underwater Sensor Network (CUSN) problem for Underwater Wireless Sensor Network (UWSN). We have a set of nodes and we have a probabilistic locality set of each node. We would like to compute the probability that the network is connected.

3 System Model and Problem Formulation

In the section we are going to describe notation, connectivity and coverage model and node connectivity model. Finally we present problem formulation for $Conn(G, \mathbf{R})$.

3.1 Notation

In this section we are going to describe some of the notations, we used throughout this paper.

- $G = (V, E)$: the underlying connectivity graph of a given UWSN.
- $R_{tr}(v)$: the transmission range for node v.
- $R_v = \{r_{(v,1)}, r_{(v,2)}, ...\}$: the locality set for node v.
- $\mathbf{R} = \{R_v : v \in V\}$: the set of all the locality sets of all node in G.
- $p(r_{(v,i)})$: the probability that node v is in region $r_{(v,i)}$.
- PES : an ordering $(v_1, v_2, ..., v_{n-k})$ of the nodes of G for every $i \in \{1, 2, .., n-k\}$:, the node v_i is simplicial in the subgraph of G.
- $\{K_{(v,1)}, K_{(v,2)}, ..., K_{(v,k)}\}$: are k-cliques associated with vertex v in a k-tree, where v is a simplicial vertex.
- $K_{(v,base)}$: the base clique to which v is attached and it is formed by all adjacent vertices of v.
- $\{T_{(v,1)}, T_{(v,2)}, ..., T_{(v,k)}\}$: tables associated with clique $\{K_{(v,1)}, K_{(v,2)}, ..., K_{(v,k)}\}$ respectively.
- $V(G')$: the nodes of a subgraph G'.
- $Reach(r_{(x,i)}, r_{(y,j)}) = 1$ if node x in region $r_{(x,i)}$ can reach node y in region $r_{(y,j)}$ else 0.
- $Conn(G, \mathbf{R})$: The probability that the nodes of G are in a state where all nodes are connected $= \sum Pr[S : S$ is a connected state of nodes in $G]$.

3.2 Connectivity and Coverage Model

Underwater Wireless Sensor Network (UWSN) is modelled by a graph G of (V, E) where $V = (v_1, v_2, v_3, .., v_n)$ is a set of vertices and E is a set of edges. The transmission range for node $v \in V$ is $R_{tr}(v)$. One node can transmit data to other node if they are within transmission range of each other.

3.3 Location Probability Model

We are considering each sensor node $v \in V$ can be located into a set of regions, $R_v = \{r_{(v,1)}, r_{(v,2)}, ...\}$ with certain probabilities, $p(r_{(v,1)}), p(r_{(v,2)}), ...$ which are called location probabilities. Mathematical sum of a node's location probabilities are 1 but in some cases we are considering a set of regions with high probable values where sum of locations probabilities

can be less than 1. Location probability of a node in a specific region, is independent of location probability of that node located into other regions. Also location probabilities of one node is independent of the location probabilities of other nodes. Also we are considering the probable locations of a node can be contiguous as well as non-contiguous.

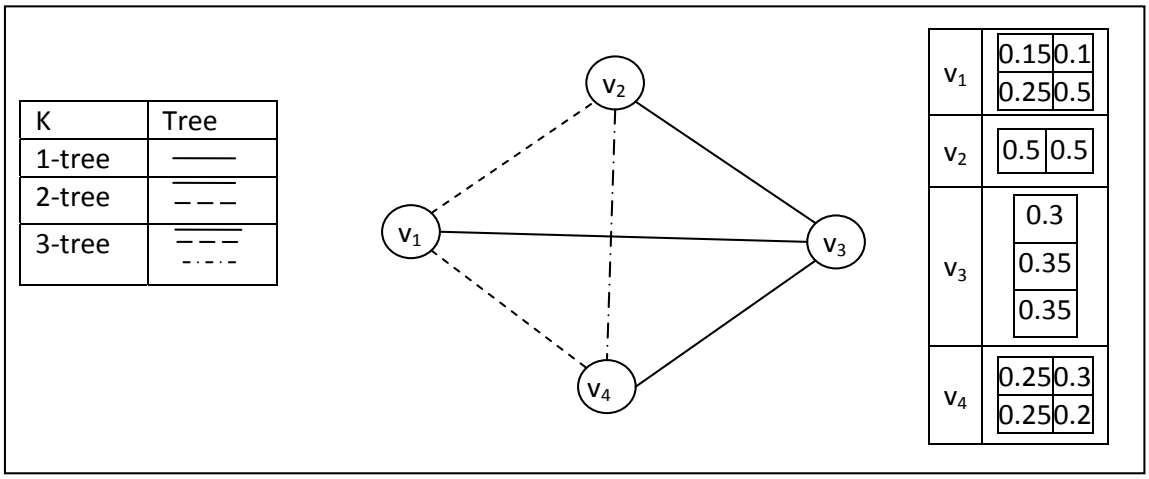

Figure 2: A partial 3-tree with 4 nodes.

Example 3.1. Fig. 1 illustrates a network of 4 nodes. Locality set of node v_1, v_2, v_3 and v_4 has $4, 2, 3$ and 4 regions. We are also considering v_4 as a sink node. Transmission range $R_{tr} = 8.5$ unit. By using this transmission range node v_1 from any of its probable regions can communicate with all probable regions of node v_2 and v_3 but not with node v_4. Similarly node v_2 from all probable regions can communicate with all probable regions of node v_4 and node v_1 but for node v_3 only the region with probability value 0.3. Node v_3 can communicate with node v_4 when v_4 is located in the regions with probability values 0.25 and 0.2. ■

3.4 Node Connectivity Model

If one or more regions in the locality set of node $x \in V$ is within the transmission radius $R_{tr}(y)$ of one or more regions in the locality set of node $y \in V$ and vice-versa then node x is reachable from node y.

A network state, S arises when each node is assigned to a specific region from it's locality set. A connected state, S_{conn} is a network state where each node can communicate with others.

3.5 Problem Statement

Now we formally define the problem.

Definition (The $Conn(G, \mathbf{R})$ Problem)**.** Let G is a UWSN where each node $v \in V$ can be located into a set of regions $R_v = \{r_{(v,1)}, r_{(v,2)}, ...\}$ with probability $p(r_{(v,i)}$ where $i = 1, 2, ...$

Also $R_{tr}(v)$ is the transmission radius for node $v \in V$. we would like to find the probability $Conn(G, \mathbf{R})$ that each node is connected. ■

An exhaustive approach to compute $Conn(G, \mathbf{R})$ can be

- compute $Reach(r_{(x,i)}, r_{(y,j)})$ for every pair of region $r_{(x,i)}, r_{(y,j)}$ where $(x, y) \in V$
- find all network states $\mathbf{S}$ for the given graph G.
- find all connected states $\mathbf{S}_{conn} \subset \mathbf{S}$
- calculate $Conn(G, \mathbf{R}) = \sum Pr[S : S \in \mathbf{S}_{conn}]$ where $Pr[S]$ can be computed by multiplying all nodes specific regional probability in state S.

Running time of the exhaustive algorithm can be calculated as follows

- let $f(n, \mathbf{R})$ is the time to find all network states $\mathbf{S}$ for the given graph G.
- let $g(n, \mathbf{R})$ the time to find all connected states $S \subset \mathbf{S}$
- in-order to calculate $Conn(G, \mathbf{R})$ the upper bound can be $|R_1| \times |R_2| \times .. \times |R_n|$. Also let v is the node located maximum number of regions then we can calculate it $|R_1| \times |R_2| \times .. \times |R_n| = \mathcal{O}(|R_v|^n)$
- so the total running time will be $\mathcal{O}(|R_v|^n)$ as $f(n, \mathbf{R})$ and $g(n, \mathbf{R})$ minor in comparison.

4 k-trees

In this section, we will define clique, k-tree and perfect elimination sequence which is used in the main algorithm.
A clique is a set of vertices that induce a complete subgraph of a graph G. A clique with k vertices is considered to be a k-tree.

Definition. The class of k-tree can be defined recursively as follows:

- The complete graph on k vertices is a k-tree.
- Lets G_n is a k-tree with n vertices where $n \geq k$. Then we can construct a k-tree G_{n+1} of $n + 1$ vertices by adding a vertex adjacent to exactly k vertices, namely all vertices of a k clique of G_n. ■

A partial k-tree is any subgraph of a k-tree . partial k-trees are rich class of graph. Forest is an example of partial 1-tree. Series-parallel graphs and chordal graphs are subfamily of partial 2-trees. Also Halin graphs, Nested SAT and IO-graphs are subclasses of partial 3-trees.

Example 4.1. Fig. 1 depict a partial 2-tree. The complete graph of 2 vertices namely v_1 and v_2 is a 2-tree. Then we added vertex V_3 which is adjacent to both v_1 and v_2 is a 2-tree of 3 vertices. Finally a vertices s is added to the clique v_1v_2 to form the 2-tree with 4 vertices.■

4.1 Perfect Elimination Sequence

A perfect elimination sequence (PES) in a graph is an ordering of the vertices of the graph such that, for each vertex v, v and the neighbors of v that occur after v in the order form a clique. In-order to find PES we need simplicial vertex.

Definition (Simplicial Vertex)**.** A simplicial vertex of a graph G is a vertex v such that the neighbours of v form a clique in G. Clearly, if G has a PES, then the last vertex in it is simplicial in G.■

Example 4.2. In fig. 1 we show that there are two simplicial vertices, v_1 and s. But s is our sink node so we are not eliminating s. So After elimination of v_1, we will be able to eliminate either v_2 or v_3. So the PES will be either v_1, v_2 or v_1, v_3.■

5 Extracting a k-tree

In this section, we are going to describe how we construct a k-tree from a set of given UWSN node where each node can be located into a set of regions.

5.1 Extracting 1-tree

We describe how we construct 1-tree for given a set of nodes.

We are given a set of nodes V where each nodes is located into a set of regions. The function OneTree() construct a greedy 1-tree, $Tree(V, E)$ which has the heighest connectivity.

- Step 1: N is a set which is initialize by $\{V \setminus sink\}$ and T is another set which is initialized by $sink$ node. E is a empty edge set.
- Step 2: repeat the following steps until N is empty.
- Step 3: if $p \in N$ and $q \in T$ are two nodes which has a connectivity probability greater than or equal to any other remaining pair. For pairs we indicate one node from set N and another node from set T
- Step 4: add the edge $e(p, q)$ to E.
- Step 5: delete node p from set N and add p to T.
- Step 6: Finally the algorithms return the tree $Tree(V, E)$

Algorithm 1: Function OneTree(V)

Input: V is a set of nodes where each node is located into a set of probable regions
Output: A 1-tree $Tree(V, E)$

1 **Initialization:** N and T are two sets of nodes where N is initialized by all given nodes in V except the sink node and T is initialized by the sink node and E is an empty edge set.
2 **while** $N! = \emptyset$ **do**
3 | $p \in N$ is a node and $q \in T$ is another node where the connectivity probability between node p and q is higher or equal to another pair.
4 | add (p, q) to E.
5 | delete node p from N and add p to T
end
6 Return$Tree(V, E)$

5.2 Extracting 2-tree

In-order to extract 2-tree, we use the 1-tree skeleton. We use two functions check($Tree$) and TwoTree($Tree$,$Tree_1$) to convert 1-tree to 2-tree.

5.2.1 Function Check

The check function take a tree as a input and check that whether it's a partial 2-tree or not. If the input tree is a partial 2-tree then it returns *true* otherwise it returns $false$.

- Step 1: it initializes a *counter* by 0 which help to terminate infinite looping and $flag$ by $false$.
- Step 2: it extracts all the nodes from given0 $Tree$ and put it into a queue, Q.
- Step 3: it's iterates from step 4 to step 14 until Q is not empty and *counter* less than equal to the size of Q. The *counter* is going to be always less than the size of Q if $Tree$ a partial 2-tree.
- Step 4: pops a item from from Q and assign it to n.
- Step 5: it finds the number of edges associated with node n in $Tree$ by simply searching through the edge set of $Tree$. This value indicates the degree deg of n.
- Step 6: if deg is equal to one then it is safe to remove the node n and associated edges from $Tree$.
- Step 7: it find the edge e associated with node n and remove e from the edge set E and n from vertices set V.
- Step 8: it reset the *counter* to make sure that upto this point it's a partial 2-tree.
- Step 9: if the degree deg of n is two then it is also safe to remove the vertices and associated edges.
- Step 10: so in-order to remove the node n, it's finds the edges e_1 and e_2 associated with n by searching the edge set E of $Tree$. Also it finds the neighbours n_1 and n_2 of n.
- Step 11: it removes e_1 and e_2 from the edge set E and n from the vertices set V.
- Step 12: it search the edge set E of $Tree$ for an edge between the neighbour of n i.e. an edge between n_1 and n_2. If there is no edge then it add an edge to E.
- Step 13: it reset the *counter* to make sure that after upto deleting this node, the tree maintains the property of 2-tree.
- Step 14: if step 6 and step 9 are false then it push n to the Q and increment the value of *counter* by one. If each node has a degree more than 2 then the tree is not going to be a partial 2-tree. We are incrementing the *counter* so that in above situation it will terminate from infinite looping.
- Step 15: after exiting from the loop it check Q is empty or not. Queue empty means we are able to delete all nodes so $Tree$ a partial 2-tree. It's holds the property of partial 2-tree. So it assign *true* to the $flag$.
- Step 16: otherwise it assign $false$ to indicate that the $Tree$ is not a partial 2-tree.
- Step 17: Finally it returns the $flag$.

5.2.2 Function TwoTree

We are given two trees $Tree(V, E)$, the one tree generated by function OneTree() and another tree $Tree1(V, E_1)$ where E_1 contains all possible edges excluding E that can be generated by vertices set V. That is $E \cap E_1 = \emptyset$. The function TwoTree() select high probable edges and from E_1 and check whether or not after adding that edge to $Tree(V, E)$, it hold the property of 2-tree. If so then it add that edge to E and delete it from E_1.

- Step 1: in [16] authors shows that if G is a k-tree on n vertices and m edges then $m = kn - k(k+1)/2$ and for 2-tree $m = 2n - 3$. We iterates until the size of E less than $2 * |V| - 3$
- Step 2: select the an edge from E_1 which has the connectivity value greater than or equal to any other edge.
- Step 3: use the check function to ckeck after adding this edge to $Tree$, it's still going to be a partial 2-tree.
- Step 4: if the above condition hold then it will add the edge to the edgeset E of $Tree$.
- Step 5: it remove the edge from the edgeset E_1 of $Tree1$
- Step 6:it returns the partial 2-tree constructed by this greedy technique.

Algorithm 2: Function TwoTree($Tree, Tree1$)

Input: $Tree(V, E)$ is a 1-tree and $Tree_1(V, E_1)$ is another tree where $E \cup E_1$ is the set of all possible edges between vertices $V = \{v_1, v_2, ..., v_n\}$ and $E \cap E_1 = \emptyset$

Output: A partial 2-tree.

```
1 while (E.size < 2 * |V| − 3) do
2 |   select the edge e from Tree1 which has highest connectivity probability.
3 |   if check(Tree(V, E_1 ∪ e) then
4 |   |   add e to the edgeset E of Tree
  |   end
5 |   remove e from the edgeset E_1 of Tree1
  end
6 return Tree
```

5.3 Extracting 3-tree

In-order to construct 3-tree, we use 2-tree skeleton. We added edges to 2-tree so the the following condition holds

- After adding edges it should hold the property of 3-tree.
- We select those edges with high probable values in-order to construct 3-tree from a greedy 2-tree.

Algorithm 3: Function Check($Tree$)

Input: a UWSN $Tree = (V, E)$ is a partial 2-tree where each node, $v \in V$ can be located into a set of regions $R_v = \{r_{(v,1)}, r_{(v,2)}...\}$ with probability, $\{p(r_{(v,1)}), p(r_{(v,2)})...\}$ and $(x, y) \in E$ if $x \in V$ can be located one of it's locality set and $y \in V$ can be located one of it's locality set, so that they reach each other.

Output: $flag$

```
1  Initialization: set the counter to 0 and flag to false.
2  Extract all the nodes from Tree and put into queue, Q .
3  while Q! = Empty and counter < Q.size do
4  |  Remove the front node n, form the Q
5  |  find the degree, deg of n in the tree, Tree.
6  |  if deg == 1 then
7  |  |  find and remove edge e from Tree associated with node n
8  |  |  reset the counter
   |  end
9  |  else if deg == 2 then
10 |  |  find the edges e1 and e2 associated with node n and neighbour nodes of n,
   |  |  n1, n2
11 |  |  remove e1 and e2 from Tree
12 |  |  if there is no edge between n1 and n2 in Tree then add a edge between them.
13 |  |  reset the counter
   |  end
   |  else
14 |  |  add n to the end of the queue,Q and increment the counter by one
   |  end
   end
   if Q is empty then
15 |  set flag=true
   end
   else
16 |  set the flag = false
   end
17 return flag
```

6 Main Algorithm

In this section we present an algorithm to compute the exact solution for $Conn(G, \mathbf{R})$ problem. More specially, the algorithm takes as input a UWSN network G where each node has a probabilistic locality set, a PES and compute Prob, a solution to the input $Conn(G, \mathbf{R})$ instance. The function uses a dynamic programming framework to solve $Conn(G, \mathbf{R})$ problem.

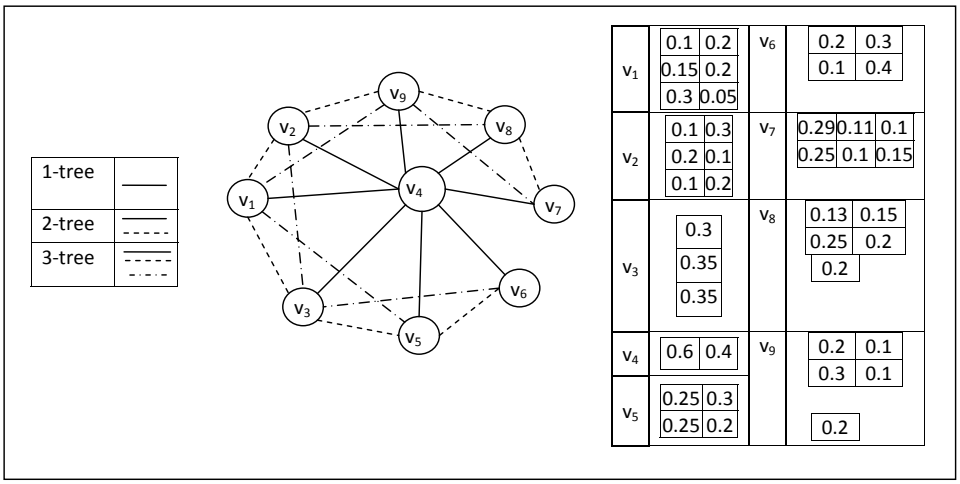

Figure 3: A UWSN modelled by a 3-tree.

6.1 Key Data-structures

Each clique associated with a table. Each row in the table defines key-value mapping.

- A key is a set of sets namely partition(s) of nodes along with corresponding regions in that particular clique.
- A value is a probability which is the multiplication of regional probability of nodes for the clique.

Example 6.1. Fig. 3 illustrates a graph G which is a 3-tree with 9 nodes and their corresponding probabilistic locality set. There are 19 triangles so there are 19 cliques associated with this 3-tree. For each clique the algorithm maintain a table. For every table there are some rows as key-value mapping. For example the clique $< v_1, v_2, v_3 >$ can be partitioned 5 different ways including $\{v_1, v_2, v_3\}, \{v_1, v_2\}\{v_3\}, \{v_1, v_3\}\{v_2\}, \{v_1\}\{v_2, v_3\}$ and $\{v_1\}\{v_2\}\{v_3\}$. For each partition there are $6 \times 6 \times 4 = 144$ rows as key-value mappings because the locality set of node v_1, v_2 and v_3 are $6, 6$ and 4 respectively. One of the row is $\{v_1, v_2, v_3\}^{(1,1,1)}0.004$ where v_1, v_2 and v_3 are in same partition with region $1, 1$ and 1 respectively and the probability 0.004 is multiplication of corresponding regional probabilities.■

When the algorithm starts elimination node by the order of PES there will be table merging and details is presented in section 6.2.

6.2 Algorithms Description

We now explain the main steps of function main, merge and merge partitions.

Algorithm 4: Function Main(G, $\mathbf{R}$, $p(r_{(v,i)})$, PES)

Input: a UWSN $G = (V, E)$ is a partial k-tree where each node, $v \in V$ can be located into a set of regions $R_v = \{r_{(v,1)}, r_{(v,2)}...\}$ with probability, $\{p(r_{(v,1)}), p(r_{(v,2)})...\}$ and $(x, y) \in E$ if $x \in V$ can be located one of it's locality set and $y \in V$ can be located one of it's locality set, so that they reach each other. PES is a perfect elimination sequence $(v_1, v_2, ..., v_{n-k})$ of G.

Output: Prob, a solution to the input instance.

Notation: $Temp$ is a map from keys to probabilities.

1 **Initialize** every clique by a table.
2 **for** $i = 1, 2, ..., n - k$ **do**
3 node v_i is associated with k-cliques, $K_{(v_i,1)}, K_{(v_i,2)}, .., K_{(v_i,k)}$
//$T_{(v_i,1)}, T_{(v_i,2)}, .., T_{(v_i,k)}$ are the tables associated with cliques $K_{(v_i,1)}, K_{(v_i,2)}, .., K_{(v_i,k)}$ respectively
4 $Temp = T_{(v_i,1)}$
5 **for** $j = 2, 3.., k$ **do**
6 $Temp = merge(Temp, T_{(v_i,j)})$
end
// clique $K_{(v_i,base)}$ is the base clique and table $T_{(v_i,base)}$ is the base table of node v_i
7 $Temp = merge(Temp, T_{(v_i,base)})$
8 **if** *v_i is in a partition of Temp by itself* **then**
9 ignore the row containing that partition.
end
10 **else**
remove node v_i from $Temp$ and assign the result to $T_{(v_i,base)}$
end
end
11 **return** *Prob=$\sum$(All probability for single partition in the remaining table)*

6.2.1 Function Main

Main function eliminates every node according to the PES by merging all cliques associated with that node and updating the result to base clique. The last remaining clique associates with the sink node. Finally main function calculates the connectivity from the last clique by adding probabilities for those row which is associate with single partition.

In more details,

- Step 1 : initialize each clique by a table describe in subsection 6.1.
- Step 2 : processes each node v_i in PES. Every processing node, v_i is associated with k cliques.
- Step 3 : finds all those cliques, $K_{(v_i,1)}, K_{(v_i,2)}, .., K_{(v_i,k)}$.
- Step 4 : assign table $Temp$ with table $T_{(v_i,1)}$ associated with clique $K_{(v_i,1)}$.

- Steps 5-6 : iteratively merge all associated tables of node v_i and assign the result to table $Temp$ by using $merge$ function.
- Step 7 : merge $Temp$ table with base table of node v_i $T_{(v_i,base)}$ by using $merge$ function and assign the result to $Temp$.
- Step 8 : check whether v_i is in a partition by itself i.e. bad set.
- Step 9 : if there is a bad partition in $Temp$ then the algorithm simply ignore the row along with bad partition.
- Step 10 : remove the vertex v_i and it's locality set from $Temp$ and assign the result to $T_{(v_i,base)}$.
- Step 11 : calculate and return connectivity for the network.

Example 6.2. One of the PES of the 3-tree depict in fig. 3 is $< v_1, v_6, v_5, v_7, v_8, v_9 >$. In-order to eliminate v_1, the algorithm merge three cliques $< v_1, v_3, v_4 >$, $< v_1, v_2, v_3 >$ and $< v_1, v_2, v_4 >$ to $< v_1, v_2, v_3, v_4 >$. Next step the algorithm find the base clique $< v_2, v_3, v_4 >$ and merge with $< v_1, v_2, v_3, v_4 >$. Finally it deletes v_1 from merged clique and update the result to $< v_2, v_3, v_4 >$ ■

$T_1(v_1, v_2, v_3)$

.	.
$\{v_1, v_2\}^{(1,1)}\{v_3\}^{(2)}$	0.002
.	.
.	.
.	.

$\times$

$T_2(v_1, v_3, v_4)$

.	.
$\{v_1\}^{(1)}\{v_3, v_4\}^{(2,1)}$	0.008
.	.
.	.
.	.

$\Rightarrow$

$Temp(v_1, v_2, v_3, v_4)$

.	.
$\{v_1, v_2\}^{(1,1)}\{v_3, v_4\}^{(2,1)}$	0.0008
.	.
.	.
.	.

Figure 3 a). Merging two table into $Temp$

$Temp(v_1, v_2, v_3, v_4)$

.	.
$\{v_1, v_2\}^{(1,1)}\{v_3, v_4\}^{(2,1)}$	0.0008
.	.
.	.
.	.

$\Rightarrow$

$Temp(v_2, v_3, v_4)$

.	.
$\{v_2\}^{(1)}\{v_3, v_4\}^{(2,1)}$	0.0008
.	.
.	.
.	.

Figure 3 b). Deleting node v_1 from $Temp$

6.2.2 Function merge

The primary task of merge function is to merge two table T_1, T_2, update keys and values of the newly created table T and return the table T.

In more details,

- Step 1 : finds the common vertices between two tables and assign the common nodes probability 1.
- Step 2 : check if whether or not the common vertex set C is empty. If it is empty then the algorithm returns otherwise go to next step.
- Steps 3-5 : performs the row-wise merging by using function $mpar$ and create another row.

- Steps 6-7 : updates the locality set for every vertex v of newly created row.
- Steps 8-9 : calculates the regional probability of common nodes.
- Step 10 : updates the newly created row probability by multiplying row-wise probability and dividing by the sum of common nodes regional probabilities.
- Step 11 : insert the row into table T and
- Step 12 : return table T.

Example 6.3. Fig 3 a). illustrates merging row $\{v_1, v_2\}^{\{1,1\}}\{v_3\}^{\{2\}}$ 0.002 of table T_1 with row $\{v_1\}^{\{1\}}\{v_3, v_4\}^{\{2,1\}}$ 0.008 of table T_2 which is fundamental operation for merging two tables. The function merge uses $pMerge$ function which takes two partitions $\{v_1, v_2\}\{v_3\}$ and $\{v_1\}\{v_3, v_4\}$ and returned $\{v_1, v_2\}\{v_3, v_4\}$ because $\{v_1, v_2\} \cup \{v_1\} = \{v_1, v_2\}$ and $\{v_3\} \cup \{v_3, v_4\} = \{v_3, v_4\}$. The merge function updates the locality set of each node of the merged partition $\{v_1, v_2\}^{\{1,1\}}\{v_3, v_4\}^{\{2,1\}}$ from the locality set of nodes in merging partitions. There are two common nodes v_1 and v_3 located in region 1 and 2 respectively with probability 0.1 and 0.2 respectively in the merging partitions. It update the probability of newly created partition 0.0008 by multiplying row-wise probabilities 0.002×0.008 and dividing the probability by product of the common nodes corresponding regional probabilities 0.1×0.2.

Fig 3 b). shows the deletion of node v_1 from $Temp$. The function main simply look for the node and remove it and it's locality set from key.■

Algorithm 5: Function merge(T_1, T_2)

Input: Two tables T_1 and T_2 that share at least one common vertex
Output: A table T
Notation C is a set of vertices and Obj is a row of table T and $Prob\_C$ is a double variable

```
1  set C = the set of common vertices between T1 and T2 , set Prob_C = 1
2  if C ≠ ∅ then
3  |  foreach row r in T1 do
4  |  |  foreach row s in T2 do
5  |  |  |  Obj.par =pMerge(r.par,s.par)
6  |  |  |  foreach vertex v_i in Obj.par where i = 1, 2, .., k + 1 do
7  |  |  |  |  Obj.loc[v_i] = r.loc[v_i]||s.loc[v_i]
   |  |  |  end
8  |  |  |  foreach vertex v ∈ C do
9  |  |  |  |  Prob_C = Prob_C * s.loc[v]
   |  |  |  end
10 |  |  |  Obj < Obj.par : Obj.loc >= Prob[r]×Prob[s] / Prob_C
11 |  |  |  Insert Obj in T as a row.
   |  |  end
   |  end
   end
12 return Table T
```

6.2.3 Function Merge Partitions

The merging partitions is mainly done using the union operation by mapr function. The mpar function takes as input two partitions P_1, P_2 and merge them into one partition P.

In more details,

- Steps 1-2 : adds all the sets of P_2 to P_1.
- Steps 3-4 : selects two sets s^* and t^* from partition P_1
- Step 5: checks whether or not they are disjoint . If they are not disjoint then it will go to step 6 otherwise it will return to step 4.
- Step 6 : delete s^* from P_1.
- Step 7 : computes the union of s^* and t^* and insert it at the beginning of partition P_1.
- Step 8 : delete t^* from P_1.
- Steps 9-10 : set the iterator to the beginning of P_1 and return to step 5.
- Step 11 : assign P_1 to P and finally the algorithm return P.

Algorithm 6: function pMerge(P_1, P_2)

Input: Two partitions P_1 and P_2
Output: A partition P
Notation: s and t are two set iterators and their corresponding set are indicated by s^* and t^*.

```
1  foreach set s* in P2 do
2  |  P1.push_back(s*)
   end
3  for (s = P1.begin(); s ≠ P1.end(); ++s) do
4  |  for (t = s.next(); t ≠ P1.end(); ++t) do
5  |  |  if s* ∩ t* ≠ ∅ then
6  |  |  |  P1.push_front(s* ∪ t*)
7  |  |  |  P1.delete(s*)
8  |  |  |  P1.delete(t*)
9  |  |  |  s = P1.begin()
10 |  |  |  break
   |  |  end
   |  end
   end
11 set P = P1
   return P
```

6.3 Verification Cases

In this subsection we are going to present some cases by using these one can verify the correctness of our algorithm.

- For a partial k-tree, there should be more than one PES. In our algorithm, we tried all elimination sequence and we got same result.
- The algorithm exhaustively consider all possible configuration to calculate the connectivity. Consider a small example with two nodes illustrates in fig **??** where v_1 and v_2

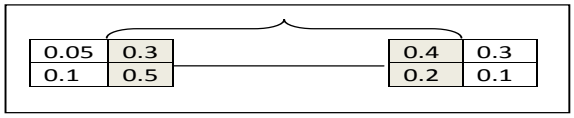

Figure 4: A simple tree with 2 nodes

has the locality set of four for both nodes. In-order to find connectivity the algorithm consider all possible configuration and there are 16 possible configurations. But v_1 can reach node v_2 when v_1 is in regions with probability 0.3 and 0.5 and v_2 is in regions with probability 0.4 and 0.2. So the connectivity for this small network is $0.8 \times 0.6 = 0.48$.

6.4 Correctness

In this section we prove the correctness of our algorithm. It is an exhaustive algorithm which takes care all possible configuration. Our algorithm consists of two fundamental operation, partition merge and table merge. First we prove that partition merge and table merge are correct then we prove that our algorithm is correct.
The partition merge perform core operations of the algorithm. The partition merge is correct because it is merging two partitions into one given that they share at least one node. The table merge is correct because of the following reason. The table merge is merging each row of one table with each row of another table by holding the condition that they share atleast one node with same regional position. It is using partition merge in-order to merge two rows which is correct. The main algorithm is elimination a node by merging all clique associated with that node and updating the merged clique with base clique of that node. Each clique is associated with table so cliques are merged by using table merge operation which is correct. That concludes that the algorithm is correct.

6.5 Running time

- The number of nodes in a table for k-tree $= k$.
- The number of ways k nodes can be partitioned $= 2^k$.
- locality set of a node which is largest in comparison with other nodes $= l_{max}$.
- The number of ways each partition reappear $= l_{max}^k$.
- So the length of a table $= l_{max}^k \times 2^k$.
- In-order to merge two tables total number of operation$= (l_{max}^k \times 2^k)^2$.
- When we are elimination a node the number of clique associated with a node for k-tree$= k$ and there is one base clique associated with every clique. So total number of clique associated with a node $= k + 1$.
- Total number of operation to eliminate one node $= (l_{max}^k \times 2^k)^{k+1}$
- Total number of node we are eliminating $= N$.
- So the total cost $= N \times (l_{max}^k \times 2^k)^{k+1}$

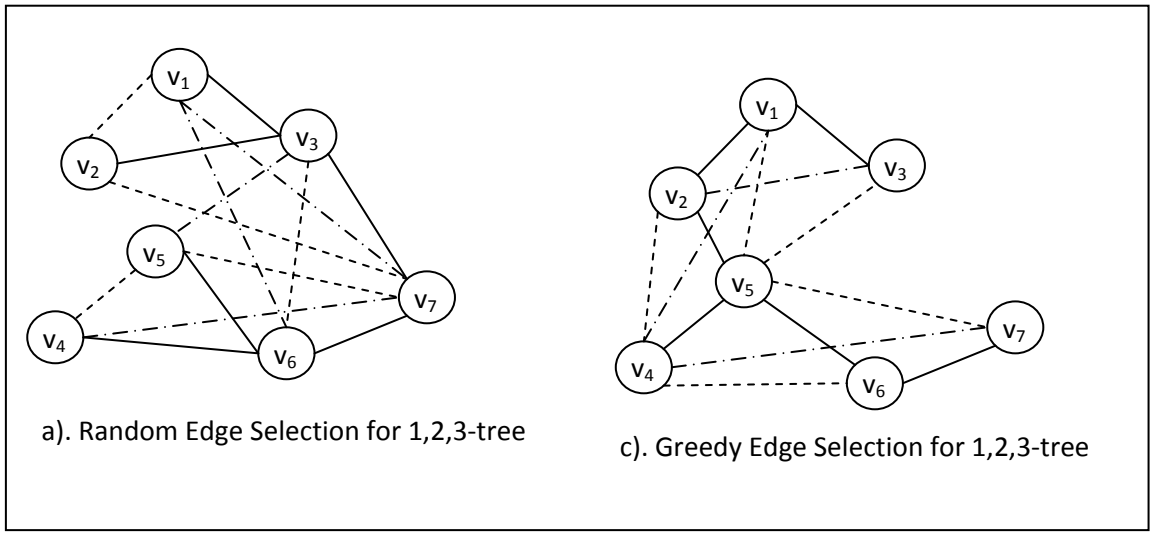

Figure 5: Greedy Vs Random Edge selection

7 Simulation Results

In this section we present simulation results that aim to investigate the following aspect

(a) complexity of our algorithm increases with increasing the value of k for partial k-tree.

(b) influence of k on accuracy where $k = 1, 2, 3, ...$

(c) effect of choosing different k-trees.

Table 3: Running time (RT) with respect to k

k	Network I	Network II	Network III
1	30	30	20
2	290	380	380
3	85000	875000	940000

Table 4: Accuracy with respect to k

k	Network I	Network II	Network III
1	62	32.96	81.8
2	71.2	36.15	98.95
3	75	37.31	100

We used three networks in-order represent running time and accuracy. Network I consists of 7 nodes with locality set of each node range from 4 to 8. Network II consists of 9 nodes where each node can be located from 3 to 6 locations.Network III consists of 12 nodes with locality set of each node range from 2 to 8.

a) k **versus running time** table 1 shows three scenarios running time with respect to k of. In every scenario the running time of the algorithm increases with k. We also observe that difference between the running time 1-tree and 2-tree are 10 to 15 times where the running time between 2-tree and 3-tree are more than 1000 times. This is because of complexity of 3-tree is much more than 2-tree.

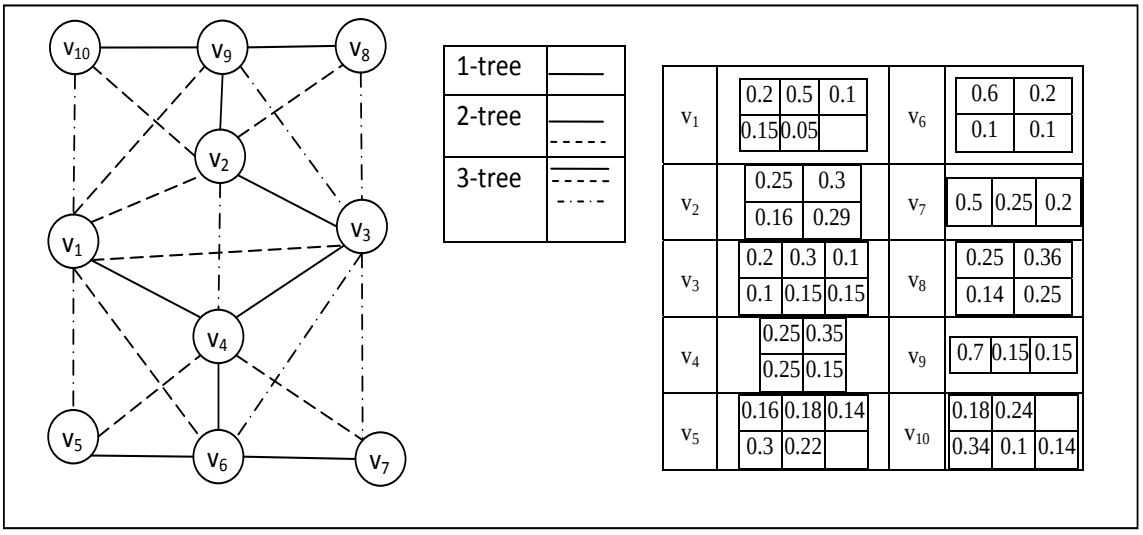

Figure 6: NetworkI

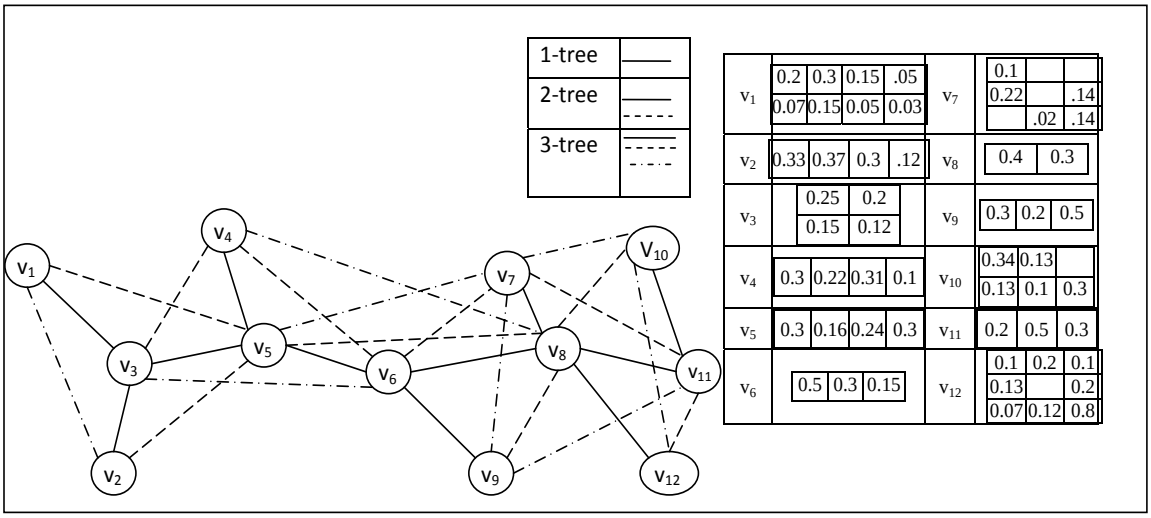

Figure 7: NetworkII

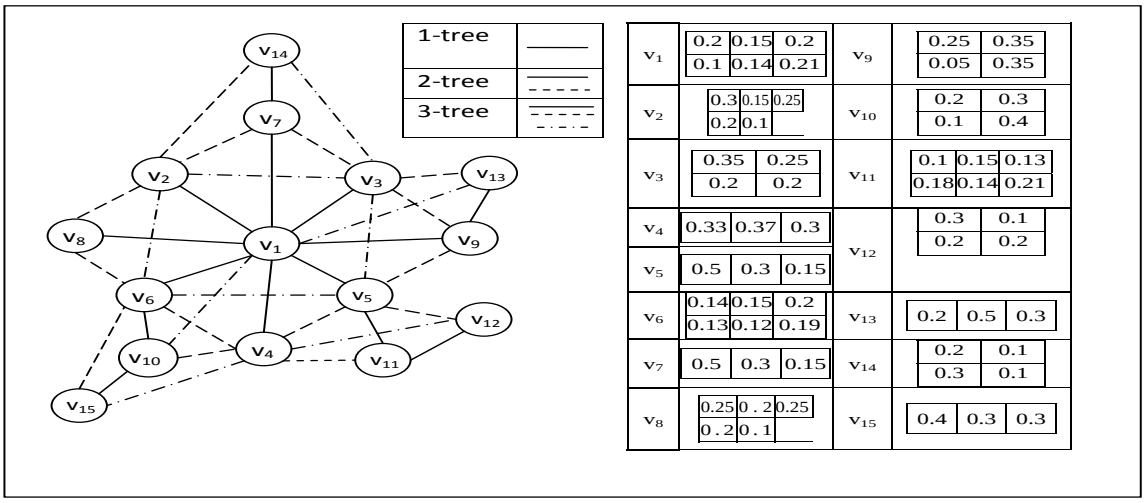

Figure 8: NetworkIII

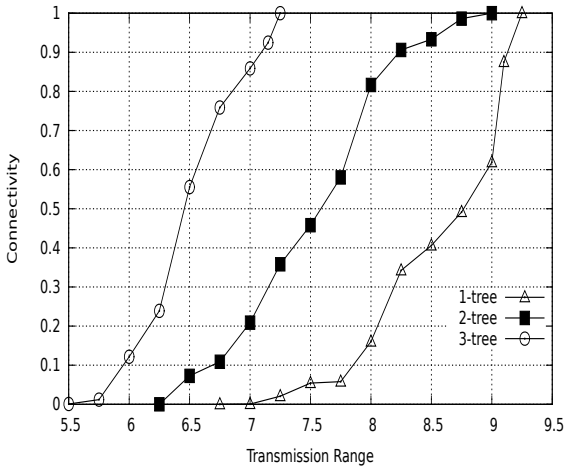

Figure 9: Connectivity versus transmission range with random edge selection.

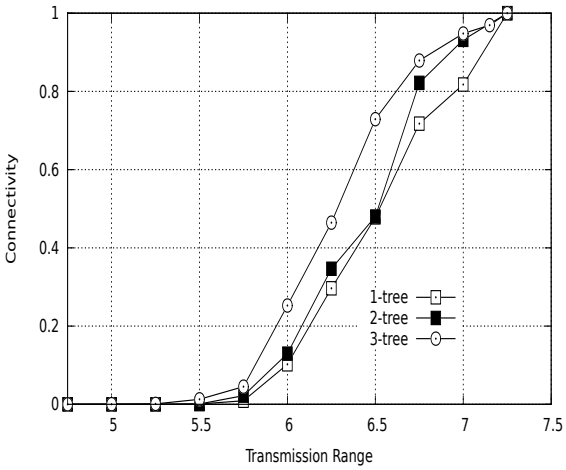

Figure 10: Connectivity versus transmission range with greedy edge selection.

b) k **versus accuracy** table 2 illustrates accuracy increases with k. The increase of accuracy is large for 1-tree to 2-tree than for 2-tree to 3-tree.

c1) **Effect of random edge selection.** Fig 9 illustrates connectivity versus transmission range when the algorithm selects randomly. We have the following observation from fig 9.

- with increase of transmission range the connectivity is also increasing. This is due to,increase of transmission range the probability of an edge between two nodes also increasing.
- connectivity for 2-tree is always greater than or equal to the connectivity of 1-tree for same transmission range. This is because in 2-tree there are more edges in comparison with 1-tree.
- The above is also true for 2-tree to 3-tree.

c2) **Effect of greedy edge selection.** Fig 10 illustrates connectivity versus transmission range when we are selecting edges by greedy techniques. In greedy techniques those edges are selected which has a high probable values. Fig 5 illustrates greedy edge selection vs random edge selection scenario. It is observable from figure that the network is fully connected with a smaller transmission range for all tree in comparison with random edge selection strategy.

8 Network with Relays

In this section we add relay nodes to our network. So our data structures changes as a result main algorithm and merge function also changes in some aspect. We describe the updates and show the results after adding relays.
We added relays, $Rel \subset V$ to the network in addition to sensor nodes which is referring target $Tar \subset V$ node in this section.

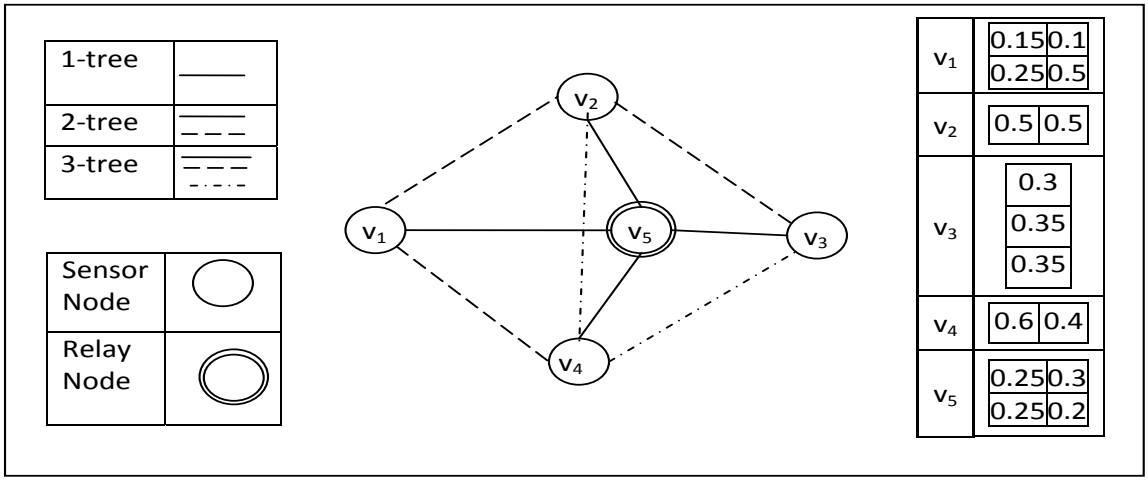

Figure 11: A partial 2-tree with 6 nodes.

Example 8.1. Fig 11 illustrates a network of 6 nodes where v_1, v_2, v_3 and v_4 are target nodes and v_5, v_6 are relays. v_5 is enhancing the connectivity of the network but v_6 is not increasing the connectivity. So we can simply ignore v_6 and measure the connectivity from v_1 to v_5

8.1 Problem Statement

In this section we define the problem.

Definition (The $Conn(G, Tar)$ Problem)**.** We are given, V the set of all nodes where each node $v \in V$ is located in a set of regions, $R_v = \{r_{(v,1)}, r_{(v,2)}, ...\}$ with probability $p(r_{(v,i)}$ where $i = 1, 2, ...$ $Rel \in V$ is the set of relay nodes and $Tar \in V$ is the set of target nodes where $V = Rel \cup Tar$. Also, if $x \in V$ and $y \in V$ are two nodes where x can be located into one of it's locality set and y can be located into one of it's locality set that they can communicate with each other then there is a link between x and y, denoted by $(x, y) \in E$. Here we use relay nodes to enhance the performance of the network but we are interested to find out the probability $Conn(G, Tar)$ that all target nodes Tar are connected.

8.2 Key Data Structures

We know from section 6.1 that a typical row in a table is key-value mapping. A key consists of i). partitions ii). regional set iii). target node attach. We explain them this section.

i) partitions: There can be more than one partition associated with each row. Each partition consists with one or more node. We use braces to distinguish each partition. Two or more nodes are in the same partition means they can communicate each other when they are in regions indicated by regional set. For example in the key $\{v_1, v_2\}^{(1,1)}_{(1)} v_3{}^{(2)}_{(0)}$, there are two partitions including $\{v_1, v_2\}$ and $\{v_3\}$. Also node v_1 can reach node 2 when they are both in region 1 of their corresponding locality set but neither node v_1 nor node v_2 from region 1 can reach node 3 when node v_3 is in region 2 of it's locality set.

ii). regional set: The regional set of a partition consists of the position of corresponding node in the locality set. The regional set is indicated as a superscript of partition and is surrounded by parentheses. For example in the key $\{v_1, v_2\}^{(1,1)}_{(1)} v_3{}^{(2)}_{(0)}$, there are two regional sets $(1, 1)$ and (2) associated with two partitions $\{v_1, v_2\}$ and $\{v_3\}$ respectively. More specifically the partition-regional set pair $\{v_1, v_2\}^{(1,1)}$ indicates that node v_1 and v_2 are both located in region 1 of their corresponding locality set.

iii). Target node attach: The target node attach associates with every partition. If there is one or more target node in a partition then the value of target node attach is 1 otherwise it is 0. In the above example we indicate the target node attach as a subscript of the partition. First partition in the above key v_1 and v_2 are target nodes so as a subscript we put 1 and for second partition we simply put 0 to indicate that v_3 is a relay node.

$T_1(v_1, v_2, v_3)$

.	.
$\{v_1, v_2\}^{(1,1)}_{(1)}\{v_3\}^{(2)}_{(0)}$	0.002
.	.
.	.
.	.

$\times$

$T_2(v_1, v_3, v_4)$

.	.
$\{v_1\}^{(1)}_{(1)}\{v_3, v_4\}^{(2,1)}_{(1)}$	0.008
.	.
.	.
.	.

$\Rightarrow$

$Temp(v_1, v_2, v_3, v_4)$

.	.
$\{v_1, v_2\}^{(1,1)}_{(1)}\{v_3, v_4\}^{\{2,1\}}_{(1)}$	0.0008
.	.
.	.
.	.

Figure 3 a). Merging two table into $Temp$

$Temp(v_1, v_2, v_3, v_4)$

.	.
$\{v_1, v_2\}^{(1,1)}_{(1)}\{v_3, v_4\}^{(2,1)}_{(1)}$	0.0008
.	.
.	.
.	.

$\Rightarrow$

$Temp(v_2, v_3, v_4)$

.	.
$\{v_2\}^{(1)}_{(1)}\{v_3, v_4\}^{(2,1)}_{(1)}$	0.0008
.	.
.	.
.	.

Figure 3 b). Deleting node v_1 from $Temp$

8.3 Main function

- Step 8: check whether or not the node v_i, we are going to eliminate is a relay node. If v_i is relay node, It goes to next step otherwise it goes to step 10.
- Step 9: search every partition in every row of the table $Temp$ for node v_i. It deletes node v_i from every partition that contains node v_i. It also updates the regional sets of corresponding partitions from which v_i was deleted by removing the corresponding regions of v_i.
- Step 10: If v_i is sensor node then it goes to next step.
- Step 11: search every partition in every row of table $Temp$ for v_i. If the partitions that contains v_i consists of more than one node then remove v_i from that partitions. It also removes the location information of v_i from the regional sets corresponding to those

partitions. If a partition contains v_i itself this kind of partition is called bad partition. The algorithm simply ignore bad partition.

Algorithm 7: Function Main(G, $\mathbf{R}$, Tar, Rel, $p(r_{(v,i)})$, PES)

Input: a UWSN $G = (V, E)$ is a partial k-tree where each node, $v \in V$ can be located into a set of regions $R_v = \{r_{(v,1)}, r_{(v,2)}...\}$ with probability, $\{p(r_{(v,1)}), p(r_{(v,2)})...\}$ and $(x, y) \in E$ if $x \in V$ can be located one of it's locality set and $y \in V$ can be located one of it's locality set, so that they reach each other. A set of target nodes $Tar = \{v_1, v_2, ...\}$ where $Tar \subset V$. A set of relay nodes $Rel = \{v_1, v_2, ...\}$ where $Rel \subset V$. PES is a perfect elimination sequence $(v_1, v_2, ..., v_{n-k})$ of G.

Output: Prob, a solution to the input instance.

Notation: $Temp$ is a map from keys to probabilities.

Initialize every clique by a table.
for $i = 1, 2, ..., n - k$ **do**
    node v_i is associated with k-cliques, $K_{(v_i,1)}, K_{(v_i,2)}, .., K_{(v_i,k)}$
    $//T_{(v_i,1)}, T_{(v_i,2)}, .., T_{(v_i,k)}$ are the tables associated with cliques $K_{(v_i,1)}, K_{(v_i,2)}, .., K_{(v_i,k)}$ respectively
    $Temp = T_{(v_i,1)}$
    for $j = 2, 3.., k$ **do**
        $Temp = merge(Temp, T_{(v_i,j)})$
    end
    // clique $K_{(v_i,base)}$ is the base clique and table $T_{(v_i,base)}$ is the base table of node v_i
    $Temp = merge(Temp, T_{(v_i,base)})$
    if *v_i is a relay node* **then**
        remove node v_i from $Temp$ and assign the result to $T_{(v_i,base)}$ after updating the regional set.
    end
    else
        Remove v_i from all the partitions of $Temp$ except the partition with v_i itself and assign the result to $T_{(v_i,base)}$ after updating the regional set.
        //We simply ignore the row where there is a partition of v_i itself.
    end
end
return *Prob=$\sum$(All probability for single partition in the remaining table)*

8.4 Merge Function

Algorithm 8: Function merge(T_1, T_2)

Input: Two tables T_1 and T_2 that share at least one common vertex
Output: A table T
Notation C is a set of vertices and Obj is a row of table T and $Prob\_C$ is a double variable

```
1  set C = the set of common vertices between T1 and T2 , set Prob_C = 1
2  if C ≠ ∅ then
3      foreach row r in T1 do
4          foreach row s in T2 do
5              Obj.par =pMerge(r.par,s.par)
6              foreach vertex v_i in Obj.par where i = 1, 2, .., k + 1 do
7                  Obj.locmap[v_i] = r.locmap[v_i]||s.locmap[v_i]
               end
8              foreach node v in every partition P in Obj.par do
9                  foreach node u in every partition Q in s.par do
10                     if v == u then
11                         Obj.tAttach[P] = max(Obj.tAttach[P], s.tAttach[Q])
                       end
                   end
12                 foreach node w in every partition T in r.par do
13                     if v == w then
14                         Obj.tAttach[P] = max(Obj.tAttach[P], r.tAttach[T])
                       end
                   end
               end
15             foreach vertex v ∈ C do
16                 Prob_C = Prob_C * s.loc[v]
               end
17             Obj < Obj.par : Obj.loc >= Prob[r]×Prob[s] / Prob_C
18             Insert Obj in T as a row.
           end
       end
   end
19 return Table T
```

- Step 8: selects every node v of every partition P in the newly created row Obj.
- Step 9: iterates every node u of every partition Q in the row s.
- Step 10: check whether u and v are the same same node or not. If they are same then it goes to next step otherwise it goes to step 9.
- Step 11: updates the target attach entry of partition P by taking the max between the target attach entry of partition P and target attach entry of partition Q.

k	Network I	Network IR	Network II	NeworkIIR	NetworkIII	Network IIIR
1	30.23	86.96	5.23	22.27	15.99	28.44
2	53.72	96.72	17.55	59.43	80.23	89.3
3	60.20	97.60	20.96	60.5	83.3	92.3

Table 5: Accuracy with respect to k

- Step 12: iterates every node w of every partition T in the row r.
- Step 13: check whether w and v are the same same node or not. If they are same then it goes to next step otherwise it goes to step 9.
- Step 11: updates the target attach entry of partition P by taking the max between the target attach entry of partition P and target attach entry of partition T.

8.5 Simulation Results

This section illustrates some simulation results when we introduce relays in the network. We also compare this simulation results with the result we obtained for network without relays.

- Table 5 compares accuracy for network with relays and without relays with respect to different values of k.
- It is obvious from the table that after adding relays accuracy increased significantly in comparison with the network without relays.
- Transmission range may vary from one network to another network but we used same transmission range for network with relay and network without relays.
- After adding relay nodes to the Network I the connectivity improves significantly which is illustrated by fig.12
- The performance of 3-tree is superior in comparison with 2-tree and which is also true for 2-tree to simple tree. Similarly the performance of 2-tree with relays in comparison with tree with relays are better.
- We can see from fig.12 that for 2-tree with relays gains 100% connectivity with transmission range less than 5.2 for Network I. On the other hand, for tree with relays needs the transmission range 5.75 in-order to achieve full connectivity.
- Also performance of 3-tree with relays is almost same as 2-tree with relays but in some cases 3-tree outperforms 2-tree. So tree with relays shows same characteristics as trees without relays.

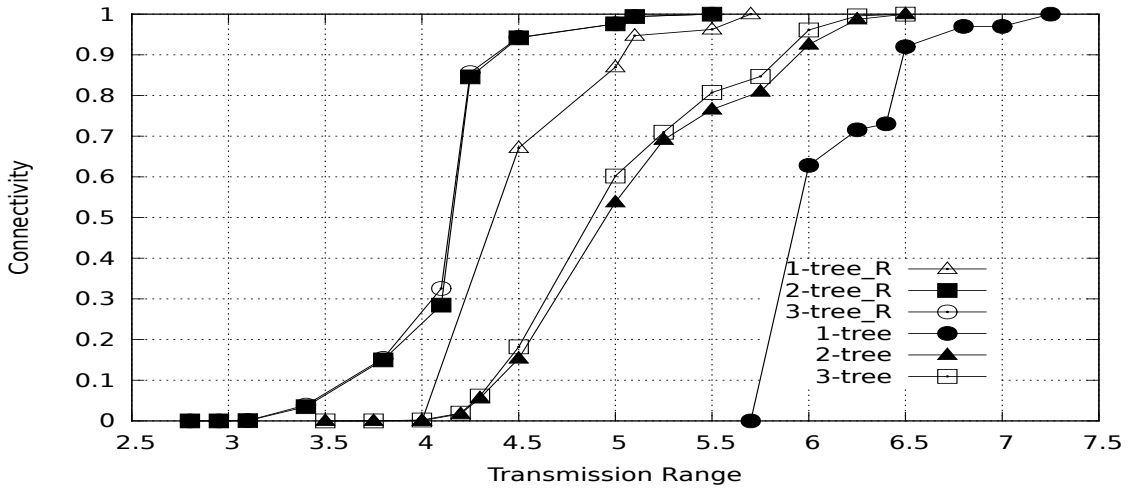

Figure 12: Connectivity Vs Transmission range for network with relays and network without relays

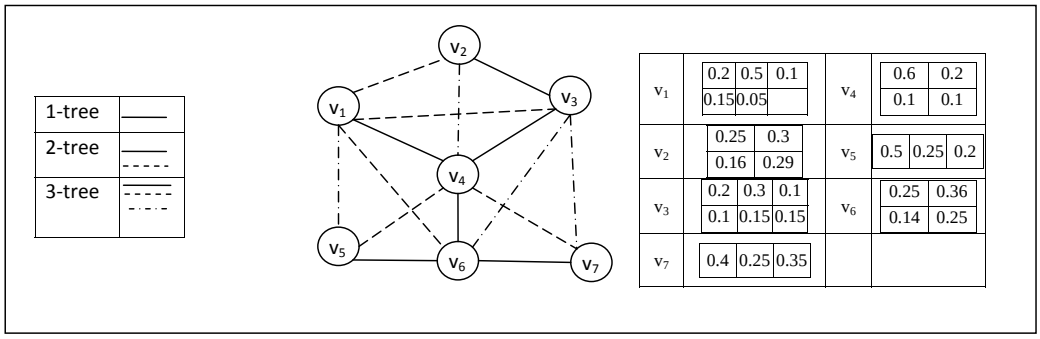

Figure 13: A partial 3-tree with 7 nodes

9 Network with N_{req}

9.1 Problem Definition

Definition (The $Conn(G, \mathbf{R}, N_{req})$ Problem)**.** Let G is a UWSN where each node $v \in V$ can be located into a set of regions $R_v = \{r_{(v,1)}, r_{(v,2)}, ...\}$ with probability $p(r_{(v,i)}$ where $i = 1, 2, ...$ Also $R_{tr}(v)$ is the transmission radius for node $v \in V$ and N_{req} is a set of nodes where $N_{req} \subset V$. we would like to find the probability $Conn(G, \mathbf{R}, N_{req})$ that N_{req} node(s) is connected with the sink node. ■

Example 9.1. Fig. 13 illustrates a network of 7 nodes. Locality set of nodes $v_1, v_2, v_3, v_4, v_5, v_6$ and v_7 has $5, 4, 6, 3, 4, 3$ and 4 regions. We are also considering v_4 as a sink node. Transmission range $R_{tr} = 8.5$ unit. By using this transmission range, we are interested to find out what is the probability that N_{req} number of nodes including the sink node is connected. For example we are interested to find out what is the probability that 5 nodes including he sink node is connected. ■

9.2 Algorithms for Network with N_{req}

9.2.1 Data Structures

We know from section 6.1 that a typical row in a table is key-value mapping. A key consists of i). partitions ii). regional set iii). target node attach and value is the probability. In algorithms for networks with N_{req}, we added a new data structure called component size, $Csize$. We are going to explain component size in this section.

Component Size: Component size is associated with every partition. Component size of a partition indicates the number of nodes connected with that partition. Component size of a partition is incremented by one when a node is deleted from that partition. For example if $\{v_1, v_2, v_4\}^{\{l_1,l_3,l_5\}}_{(tnodeAttach)}[Csize]$ is a typical key where the partition is $\{v_1, v_2, v_4\}$, regional set and target node attach are shown as a superscript and subscript of the partition and we use square brackets. So if we remove v_1 from the partition the new key will be $\{v_2, v_4\}^{\{l_3,l_5\}}_{(tnodeAttach)}[Csize + 1]$. Also when two or more partitions merged into one partition, the component size of the newly created partition will be the mathematical sum of the component size of merged partitions. For example consider two keys $\{v_1, v_2, v_4\}^{\{l_1,l_3,l_5\}}_{(tnodeAttach_1)}[Csize_1]$ and $\{v_1, v_3, v_5\}^{\{l_4,l_1,l_2\}}_{(tnodeAttach_2)}[Csize_2]$. After merging these two key the newly formed key will be $\{v_1, v_2, , v_3, v_4, v_5\}^{\{l_1,l_3,l_1,l_5,l_2\}}_{(tnodeAttach_1 or tnodeAttach_2)}[Csize_1 + Csize_2]$

We use three functions $Main()$, $merge()$ and $pMerge()$ for Network with N_{req} in-order to calculate connectivity. For merging two rows we use $pMerge()$ function described in section 6.2.3. We modified the $merge()$ function which is used for merging two table described in section 6.2.2 and $Main()$ function which is described in section 6.2.1. We added component size, $Csize$ with each partition.In this section we are going to describe the steps we added in the $Main()$ function and $merge()$ function.

9.2.2 Function merge

In this section steps 6 and 7 are added with the merge function which is mainly described in section 6.2.2

- *Step*6 : iterates through each partition, Par_i of newly created array row, Obj where $i = 1, 2, 3, ...$ Row Obj is created from row r and row s.
- *Step*7 : search for a partition(s) Par_1 in row r and Par_2 in row s where $Par_1 \subseteq Par_i$ and $Par_2 \subseteq Par_i$. Finally it calculates the component size for partition Par_i by adding the component size of Par_1 and Par_2.

Algorithm 9: Function merge(T_1, T_2)

Input: Two tables T_1 and T_2 that share at least one common vertex
Output: A table T
Notation C is a set of vertices and Obj is a row of table T and $Prob\_C$ is a double variable

```
1  set C = the set of common vertices between T1 and T2 , set Prob_C = 1
2  if C ≠ ∅ then
3      foreach row r in T1 do
4          foreach row s in T2 do
5              Obj.par =pMerge(r.par,s.par)
6              foreach partition Par_i in Obj.par where i = 1, 2, 3, ... do
7                  Obj.Csize[Par_i] = r.Csize[Par1] + s.Csize[Par2]
                   //If there is partition Par1 in r and Par2 in s where Par1 ⊆ Par_i and
                   Par2 ⊆ Par_i
               end
8              foreach vertex v_i in Obj.par where i = 1, 2, .., k + 1 do
9                  Obj.loc[v_i] = r.loc[v_i]||s.loc[v_i]
               end
10             foreach vertex v ∈ C do
11                 Prob_C = Prob_C * s.loc[v]
               end
12             Obj < Obj.par : Obj.loc >= Prob[r]×Prob[s] / Prob_C
13             Insert Obj in T as a row.
           end
       end
   end
14 return Table T
```

9.2.3 Function Main

In this section we are going to show when and how we incremented the component size. We added step 10 to the $Main$ function which increment the component size when a node is deleted from a partition.

- *Step*10 : when a node is deleted from a partition the component size of that partition is incremented by one. The component size indicates the number of nodes connected with that partition.

9.3 Simulation Results for Network with N_{req}

In this section we used Network I depicted in section **??**. Network I has 10 nodes and each node has a regional set range from 3 to 6. Figure **??** depict how connectivity changes with component size.We used $Tx = 8$ for transmission range. Also we compare the performance of 1-tree, 2-tree and 3-tree. The graph shows that the connectivity for all trees declining

Algorithm 10: Function Main(G, $\mathbf{R}$, $p(r_{(v,i)})$, PES)

Input: a UWSN $G = (V, E)$ is a partial k-tree where each node, $v \in V$ can be located into a set of regions $R_v = \{r_{(v,1)}, r_{(v,2)}...\}$ with probability, $\{p(r_{(v,1)}), p(r_{(v,2)})...\}$ and $(x, y) \in E$ if $x \in V$ can be located one of it's locality set and $y \in V$ can be located one of it's locality set, so that they reach each other. PES is a perfect elimination sequence $(v_1, v_2, ..., v_{n-k})$ of G.

Output: Prob, a solution to the input instance.

Notation: $Temp$ is a map from keys to probabilities.

Initialize every clique by a table.
for $i = 1, 2, ..., n - k$ **do**
    node v_i is associated with k-cliques, $K_{(v_i,1)}, K_{(v_i,2)}, .., K_{(v_i,k)}$
    $//T_{(v_i,1)}, T_{(v_i,2)}, .., T_{(v_i,k)}$ are the tables associated with cliques $K_{(v_i,1)}, K_{(v_i,2)}, .., K_{(v_i,k)}$ respectively
    $Temp = T_{(v_i,1)}$
    for $j = 2, 3.., k$ **do**
        $Temp = merge(Temp, T_{(v_i,j)})$
    end
    // clique $K_{(v_i,base)}$ is the base clique and table $T_{(v_i,base)}$ is the base table of node v_i
    $Temp = merge(Temp, T_{(v_i,base)})$
    if *v_i is in a partition of $Temp$ by itself* **then**
        ignore the row containing that partition.
    end
    else
        increment the component size $Csize$ from the partition containing v_i from $Temp$ and remove node v_i from that partition.
        assign the result to $T_{(v_i,base)}$
    end
end
return *Prob=$\sum$(All probability for single partition in the remaining table)*

with increasing of component size. Incrementing the component size means we wanted more node should be connected with the sink. As a result the connectivity is declining. Also we can see that connectivity for 1-tree is declining quickly in comparison with other 2 trees.

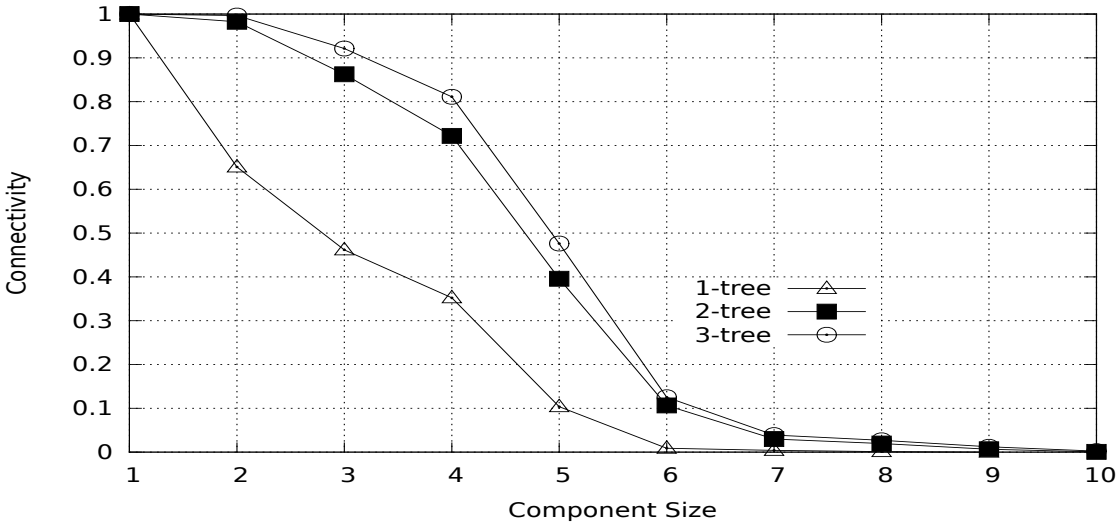

Figure 14: Connectivity versus component size for Newtork I

10 Network with Relays and N_{req}

Algorithm 11: Function Main(G, **R**, Tar, Rel, $p(r_{(v,i)})$, PES)

Input: a UWSN $G = (V, E)$ is a partial k-tree where each node, $v \in V$ can be located into a set of regions $R_v = \{r_{(v,1)}, r_{(v,2)}...\}$ with probability, $\{p(r_{(v,1)}), p(r_{(v,2)})...\}$ and $(x, y) \in E$ if $x \in V$ can be located one of it's locality set and $y \in V$ can be located one of it's locality set, so that they reach each other. A set of target nodes $Tar = \{v_1, v_2, ...\}$ where $Tar \subset V$. A set of relay nodes $Rel = \{v_1, v_2, ...\}$ where $Rel \subset V$. PES is a perfect elimination sequence $(v_1, v_2, ..., v_{n-k})$ of G.

Output: Prob, a solution to the input instance.

Notation: $Temp$ is a map from keys to probabilities.

Initialize every clique by a table.
for $i = 1, 2, ..., n - k$ **do**
    node v_i is associated with k-cliques, $K_{(v_i,1)}, K_{(v_i,2)}, .., K_{(v_i,k)}$
    $//T_{(v_i,1)}, T_{(v_i,2)}, .., T_{(v_i,k)}$ are the tables associated with cliques $K_{(v_i,1)}, K_{(v_i,2)}, .., K_{(v_i,k)}$ respectively
    $Temp = T_{(v_i,1)}$
    for $j = 2, 3.., k$ **do**
        $Temp = merge(Temp, T_{(v_i,j)})$
    end
    // clique $K_{(v_i,base)}$ is the base clique and table $T_{(v_i,base)}$ is the base table of node v_i
    $Temp = merge(Temp, T_{(v_i,base)})$
    if *v_i is in a partition by itself* **then**
        ignore the row containing that partition
    end
    else if *v_i is a relay node* **then**
        Remove v_i from all the partitions of $Temp$ and assign the result to $T_{(v_i,base)}$ after updating the regional set.
    end
    else
        Remove v_i from all the partitions of $Temp$ and increment the component size of corresponding partition by one.
        assign the result to $T_{(v_i,base)}$ after updating the regional set.
    end
end

Algorithm 12: Function merge(T_1, T_2)

Input: Two tables T_1 and T_2 that share at least one common vertex
Output: A table T
Notation C is a set of vertices and Obj is a row of table T and $Prob\_C$ is a double variable

```
1  set C = the set of common vertices between T1 and T2 , set Prob_C = 1
2  if C ≠ ∅ then
3      foreach row r in T1 do
4          foreach row s in T2 do
5              Obj.par =pMerge(r.par,s.par)
6              foreach partition Par_i in Obj.par where i = 1, 2, 3, ... do
7                  Obj.Csize[Par_i] = r.Csize[Par1] + s.Csize[Par2]
                   //If there is partition Par1 in r and Par2 in s where Par1 ⊆ Par_i and
                   Par2 ⊆ Par_i
               end
8              foreach vertex v_i in Obj.par where i = 1, 2, .., k + 1 do
9                  Obj.locmap[v_i] = r.locmap[v_i]||s.locmap[v_i]
               end
10             foreach node v in every partition P in Obj.par do
11                 foreach node u in every partition Q in s.par do
12                     if v == u then
13                         Obj.tAttach[P] = max(Obj.tAttach[P], s.tAttach[Q])
                       end
                   end
14                 foreach node w in every partition T in r.par do
15                     if v == w then
16                         Obj.tAttach[P] = max(Obj.tAttach[P], r.tAttach[T])
                       end
                   end
               end
17             foreach vertex v ∈ C do
18                 Prob_C = Prob_C * s.loc[v]
               end
19             Obj < Obj.par : Obj.loc >= Prob[r]×Prob[s] / Prob_C
20             Insert Obj in T as a row.
           end
       end
   end
21 return Table T
```

References

[1] Kemal Akkaya and Andrew Newell. Self-deployment of sensors for maximized coverage in underwater acoustic sensor networks. *Computer Communications*, 32(7):1233–1244, 2009.

[2] S. M. Nazrul Alam and Zygmunt J. Haas. Coverage and connectivity in three-dimensional underwater sensor networks. *Wireless Communication and Mobile Computing (WCMC)*, 8(8):995–1009, 2008.

[3] Habib M Ammari and Sajal K Das. A study of k-coverage and measures of connectivity in 3d wireless sensor networks. *Computers, IEEE Transactions on*, 59(2):243–257, 2010.

[4] Muhammad Ayaz, Imran Baig, Azween Abdullah, and Ibrahima Faye. A survey on routing techniques in underwater wireless sensor networks. *Journal of Network and Computer Applications*, 34(6):1908–1927, 2011.

[5] P-PJ Beaujean, Edward A Carlson, John Spruance, and Dion Kriel. Hermes-a high-speed acoustic modem for real-time transmission of uncompressed image and status transmission in port environment and very shallow water. In *OCEANS 2008*, pages 1–9. IEEE, 2008.

[6] Amy S Bower, H Thomas Rossby, and John L Lillibridge. The gulf stream-barrier or blender? *Journal of Physical Oceanography*, 15(1):24–32, 1985.

[7] AS Bower and T Rossby. Evidence of cross-frontal exchange processes in the gulf stream based on isopycnal rafos float data. *Journal of Physical Oceanography*, 19(9):1177–1190, 1989.

[8] Antonio Caruso, Francesco Paparella, Luiz Filipe M Vieira, Melike Erol, and Mario Gerla. The meandering current mobility model and its impact on underwater mobile sensor networks. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 221–225. IEEE, 2008.

[9] Xianhui Che, Ian Wells, Gordon Dickers, Paul Kear, and Xiaochun Gong. Re-evaluation of rf electromagnetic communication in underwater sensor networks. *Communications Magazine, IEEE*, 48(12):143–151, 2010.

[10] K. Chen, M. Ma, E. Cheng, F. Yuan, and W. Su. A survey on mac protocols for underwater wireless sensor networks. *Communications Surveys Tutorials, IEEE*, PP(99):1–15, 2014.

[11] Salvador Climent, Antonio Sanchez, Juan Vicente Capella, Nirvana Meratnia, and Juan Jose Serrano. Underwater acoustic wireless sensor networks: Advances and future trends in physical, mac and routing layers. *Sensors*, 14(1):795–833, 2014.

[12] Marek Doniec, Michael Angermann, and Daniela Rus. An end-to-end signal strength model for underwater optical communications. 2013.

[13] Marek Doniec, Iuliu Vasilescu, Mandar Chitre, Carrick Detweiler, Matthias Hoffmann-Kuhnt, and Daniela Rus. Aquaoptical: A lightweight device for high-rate long-range underwater point-to-point communication. In *OCEANS 2009, MTS/IEEE Biloxi-Marine Technology for Our Future: Global and Local Challenges*, pages 1–6. IEEE, 2009.

[14] Marek Doniec, Anqi Xu, and Daniela Rus. Robust real-time underwater digital video streaming using optical communication. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 5117–5124. IEEE, 2013.

[15] Xiu-juan DU, Ke-jun HUANG, Sheng-lin LAN, Zhen-xing FENG, and Fan LIU. Lb-agr: level-based adaptive geo-routing for underwater sensor network. *The Journal of China Universities of Posts and Telecommunications*, 21(1):54–59, 2014.

[16] Ehab S. Elmallah and Charles J. Colbourn. Partitioning the edges of a planar graph into two partial k-trees. In *In Proc. 19th Southeastern Conf. on Combinatorics, Graph Theory, and Computing*, pages 69–80, 1988.

[17] E.S. Elmallah, H.S. Hassanein, and H.M. AboElFotoh. On the use of a simple mobility model in ad hoc routing. In *Parallel Processing Workshops, 2001. International Conference on*, pages 479–484, 2001.

[18] Ali Elrashidi, Abdelrahman Elleithy, Majed Albogame, and Khaled Elleithy. Underwater wireless sensor network communication using electromagnetic waves at resonance frequency 2.4 ghz. In *Proceedings of the 15th Communications and Networking Simulation Symposium*, page 13. Society for Computer Simulation International, 2012.

[19] Melike Erol-Kantarci, Sema Oktug, Luiz Vieira, and Mario Gerla. Performance evaluation of distributed localization techniques for mobile underwater acoustic sensor networks. *Ad Hoc Networks*, 9(1):61–72, 2011.

[20] Alan C. Farrell and Jun Peng. Performance of IEEE 802.11 MAC in underwater wireless channels. *Procedia Computer Science*, 10(0):62 – 69, 2012. {ANT} 2012 and MobiWIS 2012.

[21] Glenn R Flierl. Particle motions in large-amplitude wave fields. *Geophysical & Astrophysical Fluid Dynamics*, 18(1-2):39–74, 1981.

[22] Amitabha Ghosh and Sajal K Das. Coverage and connectivity issues in wireless sensor networks: A survey. *Pervasive and Mobile Computing*, 4(3):303–334, 2008.

[23] Seongwon Han, Youngtae Noh, Uichin Lee, and Mario Gerla. M-fama: A multi-session mac protocol for reliable underwater acoustic streams. In *INFOCOM, 2013 Proceedings IEEE*, pages 665–673. IEEE, 2013.

[24] Frank Hanson and Stojan Radic. High bandwidth underwater optical communication. *Applied Optics*, 47(2):277–283, 2008.

[25] John Heidemann, Milica Stojanovic, and Michele Zorzi. Underwater sensor networks: applications, advances and challenges. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 370(1958):158–175, 2012.

[26] LinkQuest Inc. SoundLink Underwater Acoustic Modems. `http://www.link-quest.com/html/uwm_hr.pdf`, 2008. [Online; accessed 03 April 2014].

[27] Hydro International. Evologic Underwater Acoustic Modems. `http://www.hydro-international.com/files/productsurvey_v_pdfdocument_15.pdf`, 2007. [Online; accessed 03 April 2014].

[28] Jules Jaffe and Curt Schurgers. Sensor networks of freely drifting autonomous underwater explorers. In *Proceedings of the 1st ACM international workshop on Underwater networks*, pages 93–96. ACM, 2006.

[29] Jiejun Kong, Jun-hong Cui, Dapeng Wu, and Mario Gerla. Building underwater ad-hoc networks and sensor networks for large scale real-time aquatic applications. In *Military Communications Conference, 2005. MILCOM 2005. IEEE*, pages 1535–1541. IEEE, 2005.

[30] Uichin Lee, Paul Wang, Youngtae Noh, FLM Vieira, Mario Gerla, and Jun-Hong Cui. Pressure routing for underwater sensor networks. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.

[31] Jun Li and Mylène Toulgoat. Em channel characteristics and their impact on mac layer performance in underwater surveillance networks. *Wireless Communications and Mobile Computing*, 2014.

[32] Ji Luo, Dan Wang, and Qian Zhang. Double mobility: coverage of the sea surface with mobile sensor networks. In *INFOCOM 2009, IEEE*, pages 118–126. IEEE, 2009.

[33] Ji Luo, Dan Wang, and Qian Zhang. On the double mobility problem for water surface coverage with mobile sensor networks. *Parallel and Distributed Systems, IEEE Transactions on*, 23(1):146–159, 2012.

[34] Youngtae Noh, Uichin Lee, Paul Wang, Brian Sung Chul Choi, and Mario Gerla. Vapr: Void-aware pressure routing for underwater sensor networks. *Mobile Computing, IEEE Transactions on*, 12(5):895–908, 2013.

[35] Roald Otnes, Alfred Asterjadhi, Paolo Casari, Michael Goetz, Thor Husøy, Ivor Nissen, Knut Rimstad, Paul van Walree, and Michele Zorzi. *Underwater acoustic networking techniques*. Springer, 2012.

[36] Jim Partan, Jim Kurose, and Brian Neil Levine. A survey of practical issues in underwater networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 11(4):23–33, 2007.

[37] John G Proakis, Ethem M Sozer, Joseph A Rice, and Milica Stojanovic. Shallow water acoustic networks. *Communications Magazine, IEEE*, 39(11):114–119, 2001.

[38] Lina Pu, Yu Luo, Haining Mo, Zheng Peng, Jun-Hong Cui, and Zaihan Jiang. Comparing underwater mac protocols in real sea experiment. In *IFIP Networking Conference, 2013*, pages 1–9. IEEE, 2013.

[39] Lloyd Regier and Henry Stommel. Float trajectories in simple kinematic flows. *Proceedings of the National Academy of Sciences*, 76(10):4760–4764, 1979.

[40] Fatih Senel, Kemal Akkaya, and Turgay Yilmaz. Autonomous deployment of sensors for maximized coverage and guaranteed connectivity in underwater acoustic sensor networks. In *Local Computer Networks (LCN), 2013 IEEE 38th Conference on*, pages 211–218. IEEE, 2013.

[41] Ghalib Asadullah Shah. A survey on medium access control in underwater acoustic sensor networks. In *Advanced Information Networking and Applications Workshops, 2009. WAINA'09. International Conference on*, pages 1178–1183. IEEE, 2009.

[42] Jian Shen, Jin Wang, Jianwei Zhang, and Shunfeng Wang. A comparative study on routing protocols in underwater sensor networks. In *Advanced Technologies, Embedded and Multimedia for Human-centric Computing*, pages 593–602. Springer, 2014.

[43] Milica Stojanovic. On the relationship between capacity and distance in an underwater acoustic communication channel. *ACM SIGMOBILE Mobile Computing and Communications Review*, 11(4):34–43, 2007.

[44] Milica Stojanovic. Underwater acoustic communications: Design considerations on the physical layer. In *Wireless on Demand Network Systems and Services, 2008. WONS 2008. Fifth Annual Conference on*, pages 1–10. IEEE, 2008.

[45] Milica Stojanovic and James Preisig. Underwater acoustic communication channels: Propagation models and statistical characterization. *Communications Magazine, IEEE*, 47(1):84–89, 2009.

[46] Nguyen Thanh Tung and Nguyen Sy Minh. Power-aware routing for underwater wireless sensor network. In *Context-Aware Systems and Applications*, pages 97–101. Springer, 2014.

[47] Carlos Uribe and Walter Grote. Radio communication model for underwater wsn. In *New Technologies, Mobility and Security (NTMS), 2009 3rd International Conference on*, pages 1–5. IEEE, 2009.

[48] Abdul Wahid, Sungwon Lee, Dongkyun Kim, and Kyung-Shik Lim. Mrp: A localization-free multi-layered routing protocol for underwater wireless sensor networks. *Wireless Personal Communications*, pages 1–16, 2014.

[49] Joseph A Wald and Charles J Colbourn. Steiner trees in probabilistic networks. *Microelectronics Reliability*, 23(5):837–840, 1983.

[50] Joseph A Wald and Charles J Colbourn. Steiner trees, partial 2–trees, and minimum ifi networks. *Networks*, 13(2):159–167, 1983.

[51] Ping Wang, Lin Zhang, Bhaskar Krishnamachari, and Victor OK Li. Token-based data collection protocols for multi-hop underwater acoustic sensor networks: short paper. In *Proceedings of the Fourth ACM International Workshop on UnderWater Networks*, page 10. ACM, 2009.

[52] Yang Xiao. *Underwater acoustic sensor networks*. CRC Press, 2010.

[53] Lei Ying, Sanjay Shakkottai, Aneesh Reddy, and Shihuan Liu. On combining shortest-path and back-pressure routing over multihop wireless networks. *IEEE/ACM Transactions on Networking (TON)*, 19(3):841–854, 2011.

[54] Farizah Yunus, Sharifah HS Ariffin, and Yasser Zahedi. A survey of existing medium access control (mac) for underwater wireless sensor network (uwsn). In *Mathematical/Analytical Modelling and Computer Simulation (AMS), 2010 Fourth Asia International Conference on*, pages 544–549. IEEE, 2010.

[55] Wenyi Zhang, Milica Stojanovic, and Urbashi Mitra. Analysis of a simple multihop underwater acoustic network. In *Proceedings of the third ACM international workshop on Underwater Networks*, pages 3–10. ACM, 2008.

[56] Zhong Zhou, Jun-Hong Cui, and Shengli Zhou. Efficient localization for large-scale underwater sensor networks. *Ad Hoc Networks*, 8(3):267–279, 2010.

[57] Zhong Zhou, Zheng Peng, Jun-Hong Cui, Zhijie Shi, and Amvrossios C Bagtzoglou. Scalable localization with mobility prediction for underwater sensor networks. *Mobile Computing, IEEE Transactions on*, 10(3):335–348, 2011.