Title: Thesis Report

Project Title: k-Tree Bound on Probabilistic Connectivity of Underwater Sensor Network.

Name: Md Asadul Islam

Date: June 28, 2014

Abstract

1 Introduction

Underwater Sensor Networks (UWSNs) is a promising tool because of its remote monitoring and control technology. Application domain of such UWSNs can be military surveillance, disaster prevention, assisted navigation, offshore exploration, tsunami monitoring, oceanographic data collection, to mention a few. Most of the above mentioned applications requires UWSNs nodes move freely with water currents. Thus, node locations at any instant need to be specified probabilistically. When connectivity among some of the sensor nodes is required to perform a given function, the problem of estimating the likelihood that the network achieves such connectivity arises. In this thesis, we formulates some parameterized probabilistic connectivity problem that serves this purpose when the network contains both sensor nodes and relay nodes.

In this chapter, we introduce some of such research problems. We also give an overview of UWSNs, and discuss how such problems can be tackle.

1.1 Introduction

Underwater Sensor Networks (UWSNs) is a prominent network paradigm which consist of variable number of sensor nodes. Primary objective of UWSNs is collaborative monitoring and resource exploration. Underwater communication is using since the World War II, when hydro-phone developed in United States to communicate with submarine [22] but still there are many unexplored area in UWSNs which need to be addressed. Recently UWSNs research is intensified because there are many important applications of underwater sensing networks [2] [14] [7]. According to [14] the application domain of UWSNs can be classified as

- Scientific applications such as observe geological processes on the ocean floor, determine water characteristics e.g. temperature, salinity, oxygen levels, bacterial and other pollutant content, dissolved matter, counting or imaging animal life e.g micro-organisms, fish or mammals,coral reef [6].
- Industrial applications monitor and control commercial activities, such as determine route for underwater cable, underwater equipment related to oil or mineral extraction, underwater pipelines or commercial fisheries. Industrial applications often involve control and actuation components as well.

- Military and homeland security applications involve securing or monitoring port facilities or ships in foreign harbours, de-mining and communication with submarines and divers.
- Humanitarian applications involves search and survey missions e.g. sunken ships, digester prevention e.g. tsunami warning to coastal areas [8], identify hazards on the seabed, locate dangerous rocks or shoals in shallow waters, mooring positions, submerged wrecks etc.

Different types of UWSNs deployment is used in-order to serve above diverse types of applications. According to [14], UWSNs deployment can be classified as static, semi-mobile and mobile.

- Static deployment can be considered as two-dimensional architecture of UWSNs where nodes are attached to docks, buoys, or underwater ground. In this type of network all nodes can communicate with each other, so the network is always connected until battery depleted of some nodes. Typical application of this type of UWSNs can be underwater plates in tectonics monitoring[12], identify hazards on seabed.

- In semi-mobile deployment, nodes are suspended from buoy or they can be anchored to the seabed by strings. So nodes in the semi-mobile UWSNs can move but their movement are limited. The network topology of semi-mobile deployment is static for long duration which promotes connectivity. However the connectivity changes with water currents, subsurface waves, whirls etc. Semi-mobile UWSNs can be considered as static three dimensional architecture of UWSNs. Semi-mobile UWSNs may be used for surveillance applications or monitoring of ocean phenomena e.g. ocean biogeochemical processes, water streams, pollution.

- Mobile UWSN consists of low powered gliders or unpowered drifters. Nodes are subject to large scale movement. Mobility are useful to cover large area of ocean with limited hardware component but it raises many challenges in node localization and network connectivity.

As there are vast diversity of applications of UWSNs and challenges encountered in UWSN design, extensive research work spanning all five layers of the internet protocol stack appears in the literature. Now, we talk about the design of a protocol stack for underwater acoustic communication. For more information you can read research works [9], [14], [2] and [5].

- **Physical layer:** Electromagnetic communication such as radio and optical signal quickly absorbs by water make acoustic communication preferable for UWSNs communication. Path loss due to high attenuation of acoustic wave causes by the distance between transmitter and receiver, surface-bottom reflection and refraction etc. and geometric spreading, man maid noise and ambient noise, multi-path propagation, high delay due to low data rate and Doppler effect are the challenges for physical layer design.

Frequency shift keying(FSK) and non-coherent detection which are very effective for robust communication with low bit rates were using early systems such as for commercial modems (e.g. Telesnor series[13]) and in research prototype (e.g. micro modem development at WHOI[27])

Non-coherent technique need phase tracking, which is a very difficult task in underwater environment. On the contrary coherent modulation techniques does not need such tracking and have been developed for long-range, high-throughput systems [28] (such as phase shift keying (PSK) and quadrature amplitude (QAM)) and are using today's high-speed communications.

- **Data Link Layer:** Multiple access techniques are developed to allow devices to access a common medium, sharing the scarce available bandwidth in an efficient and fair way. Channel Access Control in an underwater sensor network poses additional challenges such as long delays, frequency-dependent attenuation and the relatively long reach of acoustic signals due to the peculiarities of the underwater channel, in particular.

  Time division multiple access (TDMA) and frequency division multiple access (FDMA) were using at the early stage of development of UWSNs. Channel separation in FDMA leads some inefficiency and all user in TDMA need to be synchronised make them unsuitable for UWASs.

  In code division multiple access CDMA signals that coexist in both time and frequency can be separated using specifically designed codes in combination with signal processing techniques, allows multiple devices to transmit simultaneously over the entire frequency band. effectiveness on multi-path fading and not require slot synchronization makes CDMA and attractive choice for UWSNs link layer protocol.

  ALOHA is a class of MAC protocols that do not try to prevent packet collision, but detect collision and retransmit lost packets. In the underwater acoustic environment, ALOHA protocols are affected by low efficiency, mainly due to the slow propagation of the acoustic channel. Additionally, the need for retransmissions increases the power consumption of sensors, and ultimately reduces the network lifetime.

  Carrier sense multiple access (CSMA) protocols are aimed at reducing the packet retransmissions, by monitoring the channel state: if the channel is sensed busy, packet transmission is inhibited so as to prevent collisions with the ongoing transmission. If the channel is sensed free, transmission is enabled. However this approach, although it prevents collisions at the sender, does not avoid collisions at the receiver due to the hidden and exposed terminal problems.

- **Network Layer:** Network layer is responsible for delivering data from source to destination, possibly over multi-hop basis. Recent years tremendous amount of research is going on physical layer and link layer but there are many unexplored area in network layer.

  Early work on routing is vector Based Forwarding [31] where data packets are forwarded along redundant and interleaved paths from the source to sink. This helps in tacking the problems of packet losses and node failures. Forwarding path is nominated by the

routing vector from source to destination. All the nodes receiving the packet computes their positions by measuring its distance to the forwarder. The forwarding path is virtually a routing pipe and the nodes inside this pipe are eligible for packet forwarding. A distributed protocol is proposed by Pompili et al. [31], which is applicable for both delay sensitive and delay-insensitive applications and allow nodes to select the next hop with the objective of minimizing the energy consumption while taking into account the specific characteristics of acoustic propagation as well as the application requirements. A depth based routing is proposed in [32]. When a node wants to send a data packet, it senses own relative current position from the surface and place its value in the header and then broadcasts. The receiving node calculates its own depth position and compares this value with the value embedded. If the value is smaller then forward the packet otherwise discard it. The performance of this protocol worse if the area is sparse and affected by packet loss.

- **Transport layer:** A transport layer protocol is needed in underwater acoustic sensor networks not only to achieve reliable collective transport of event features, but also to perform flow control and congestion control. The primary objective is to save scarce sensor resources and in-crease network efficiency. A reliable transport protocol should guarantee that the applications are able to correctly identify event features estimated by the sensor network. Congestion control is needed to prevent the network from being congested by excessive data with respect to the network capacity, while flow control is needed to avoid that network devices with limited memory are overwhelmed with data transmissions.

  Several solutions have been proposed to address the transport layer problems in UWSNs. For example, Event-to-Sink Reliable Transport (ESRT) protocol is proposed to achieve reliable event detection with minimum energy expenditure. However, the ESRT mechanism relies on spatial correlation among event flows which may not be easily leveraged in underwater acoustic sensor networks. Hence, further investigation is needed to develop efficient transport layer solutions

Connectivity is an important issue for UWSN in-order to perform localization [34], [35],[11], routing [18],[33],[17]. In this thesis, we are interested to measure the connectivity of UWSN. We are considering a semi-mobile and mobile UWSNs. Our interest is to develop methodologies that allows a designer to analyze the likelihood that a network is connected and the connectivity we are interested, can be All node connectivity: every node is connected with sink and connectivity with N_{req}: compute the probability that at least $N_{req} - 1$ nodes is connected with sink node.

The rest of chapter is organized as follows. In section 1.2 we introduce different class of connectivity problems. In section 1.3 we give an overview of some important literature in this area. In section 1.4 we outline thesis contributions and organizations.

1.2 Literature Review

A review of some published research work in this field indicates the following categories of research directions.

a. Connectivity and coverage issue for underwater sensor network.

b. k-tree and partial k-tree.

c. Mobility models for underwater sensor network.

1.2.1 Connectivity and coverage

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

1. Authors in [25] 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. They claim that connectivity is guaranteed regardless of the transmission and sensing range ratio with a coverage very close to a coverage-aware deployment approach.

2. 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 vertically 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. This work improves the coverage significantly but does not guarantee any connectivity. It is only claimed that with a certain transmission range and sensing range ratio, the connectivity can also be ensured. While this has been shown to be the case with a certain number of nodes and ratio, the approach certainly cannot guarantee the connectivity for all cases.

3. Authors in [4] 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.

4. In [3] 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. The paper concludes that truncated octahedrons yield the best results for coverage. Using the same polyhedron, connectivity is guaranteed if the transmission range of the nodes is at least 1.7889 times the sensing range.

1.2.2 k-tree and partial k-tree

1. In [29], authors introduce the class of k-tree, $k \geq 1$, as generalization of trees as follows. The complete graph on k vertices, denoted k_k , is a k-tree. Furthermore, if G is a k-tree then so is the graph obtained from G by adjoining a new vertex, and making it adjacent to every vertex in a complete subgraph on k vertices of G. Partial k-trees are subgraph of k-trees.

2. Recently in [26] authors introduce a problem to find a minimum weight spanning k-tree in a complete weight graph. The above problem is known to be NP-hard problem. They proposes four heuristics to solve above problem. Finally they compare their proposed algorithms with known heuristics and exact algorithms.

1.2.3 Mobility Models

1.3 Network Model and Problem Formulation

In this section, we present a network model that deals with arbitrary topologies where each node is located into a set of regions with probability. We also present four fundamental connectivity problems, denoted by $A-CONN$, $A-CONN$ with $Relays$, $S-CONN$ and $S-CONN$ with $Relays$. In two of our problem formulation we used relay nodes in addition to sensor nodes which do not have the sensing capability but helps to improve overall network performance.

The first problem is all sensor nodes connectivity problem:

Definition (**the** $A-CONN$ **Problem**)**.** Given

- a UWSN G.
- a set of sensor nodes V_{sense}.
- a sink node s. So the set of all node $V = \{V_{sense} \cup s\}$
- a set of regions $R_v = \{r_{(v,1)}, r_{(v,2)}, ...\}$ with probability $p(r_{(v,i)}$ where $i = 1, 2, ...$ for each node $v \in V$
- transmission radius $R_{tr}(v)$ for node $v \in V$.

compute the probability $Conn(G)$ that the network in a state where sink node s can reach all sensor nodes. ■

The second problem is all sensor nodes connectivity problem with the present of relay nodes:

Definition (**the** $A-CONN$ **with** $Relays$ **Problem**)**.** Given

- a UWSN G.
- a set of sensor nodes V_{sense}.
- a sink node s.
- a set of relay nodes V_{relay}. So the set of all nodes $V = \{V_{sense} \cup s \cup V_{relay}\}$.
- a set of regions $R_v = \{r_{(v,1)}, r_{(v,2)}, ...\}$ with probability $p(r_{(v,i)}$ where $i = 1, 2, ...$ for each node $v \in V$.
- transmission radius $R_{tr}(v)$ for node $v \in V$.

compute the probability $Conn(G, V_{relay})$ that the network in a state where sink node s can reach all sensor nodes. ■

The third problem is the subset sensor nodes connectivity problem:

Definition (**the** $S - CONN$ **Problem**)**.** Given

- a UWSN G.
- a set of sensor nodes V_{sense}.
- a sink node s. So the total number of nodes in the network $V = \{V_{sense} \cup s\}$
- an integer $n_{req}, n_{req} \leq |V_{sense}|$
- a set of regions $R_v = \{r_{(v,1)}, r_{(v,2)}, ...\}$ with probability $p(r_{(v,i)}$ where $i = 1, 2, ...$ for each node $v \in V$.
- transmission radius $R_{tr}(v)$ for node $v \in V$.

compute the probability $Conn(G, n_{req})$ that the network in a state where sink node s can reach a subset of sensor nodes having atleast n_{req} sensor nodes. ■

Finally the fourth problem is the subset of sensor nodes connectivity problem with the present of relays nodes in the network:

Definition (**the** $S - CONN$ **with** *Relays* **Problem**)**.** Given

- a UWSN G.
- a set of sensor nodes V_{sense}.
- a set of relay nodes V_{relay}.
- a sink node s. So the total number of nodes in the network $V = \{V_{sense} \cup s \cup V_{relay}\}$
- an integer $n_{req}, n_{req} \leq |V_{sense}|$
- a set of regions $R_v = \{r_{(v,1)}, r_{(v,2)}, ...\}$ with probability $p(r_{(v,i)}$ where $i = 1, 2, ...$ for each node $v \in V$.
- transmission radius $R_{tr}(v)$ for node $v \in V$.

compute the probability $Conn(G, n_{req})$ that the network in a state where sink node s can reach a subset of sensor nodes having atleast n_{req} sensor nodes. ■

We now draw following remarks about the above problems

1. The $A - CONN$ is the core problem which is the modified to formulate $A - CONN$ with *Relays*, $S - CONN$ and $S - CONN$ with *relays* problems.

2. The $A-CONN$ problem is a special case of $A-CONN$ with *Relays* problem. If the input UWSN G consists of only sensor nodes then the connectivity provided by $A-CONN$ will be same as the connectivity provided by $A-CONN$ with *Relays*. But after adding relay nodes the connectivity provided by $A-CONN$ with *Relays* for graph G will be higher than the connectivity provided by $A-CONN$ for graph G.

3. The $A-CONN$ is a important special case of the $S-CONN$ problem where $n_{req} \leq |V_{sense}|$. In the $A-CONN$ problem, the network is operating if the sensor node s can reach all sensor nodes. On the other hand The $S-CONN$ problem, the network is operating if sink s can reach a subset having at least n_{req} sensor nodes.

4. Similarly the $S-CONN$ problem is a special case of $S-CONN$ with *Relays* problems.

1.4 Thesis Contribution and Organization

1.5 Summary

2 Background on k-tree

2.1 Introduction

In this chapter we started by providing some background on k-tree and partial k-tree. We define perfect elimination sequence which is important to construct k-trees from a given set of nodes with their probabilistic locality set. We provide two algorithm in-order to construct 1-tree and 2-tree. Finally, we present some idea on how to construct 3-tree and draw some conclusions.

2.2 Definition of k-tree

In this section, we define clique, k-tree and we present partial k-tree and their classes.

Definition. *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. *Treewidth is a parameter which is use to measure how a graph is "tree like" or "close to being a tree". Treewidth of k tree is k.*

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 a graph which contains all the vertices and subset of edges 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.

Several graph problem that are NP-complete on general graphs have polynomial time algorithms for graphs with treewidth bounded by a constant. Any polynomial time algorithm for graphs of bounded treewidth is a polynomial time algorithm for partial k-tees. So partial k-trees are useful as they might be seen as a tool to gain more insight in graphs of bounded treewidth.

Example 2.1. 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.■

2.3 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 2.2. 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.■

3 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.

3.1 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$Tree(V, E)$

3.2 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.

3.2.1 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's iterates from step 3 to step 10 for each node $v \in V$ and *counter* less than equal to the size of V. The *counter* is going to be always less than the size of V if $T(V, E)$ a partial 2-tree.
- Step 3: it finds the number of edges associated with node n in $T(V, E)$ by simply searching through the edge set of $T(V, E)$. This value indicates the degree deg of v. If $deg(v)$ is equal to one then it is safe to remove the node v and associated edges from $T(V, E)$.
- Step 4: it find the edge $e(v, v_t)$ associated with node v where v_t is the neighbour of v. It removes $e(v, v_t)$ from the edge set E and v from vertices set V. Finally it reset the *counter* to make sure that upto this point it's a partial 2-tree.
- Step 5: if the degree deg of v is two then it is also safe to remove the vertices and associated edges.
- Step 6: so in-order to remove the node v, it's finds the edges $e_1(v, v_1)$ and $e_2(v, v_2$ associated with v by searching the edge set E of $T(V, E)$. Also it finds the neighbours v_1 and v_2 of v. Lastly, it removes $e_1(v, v_1)$ and $e_2(v, v_2)$ from the edge set E and v from the vertices set V. It reset the *counter* to make sure that after upto deleting this node, the tree maintains the property of 2-tree.
- Step 7: it search the edge set E of $T(V, E)$ for an edge between the neighbour's of v i.e. an edge between v_1 and v_2. If there is no edge then it add an edge$e(v_1, v_2)$ to E.

Algorithm 2: Function Check($T(V, E)$)

Input: a UWSN $T(V, E)$ 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 counter = 0 and flag = false
2  foreach node v ∈ V and counter < |V| do
3      if deg(v) == 1 then
4          if e(v, v_t) ∈ E then
               E = E \ e(v, v_t); V = V \ v; counter = 0
           end
       end
5      else if deg(v) == 2 then
6          if e_1(v, v_1) ∈ E and e_1(v, v_2) ∈ E then
               E = E \ {e_1(v, v_1) ∪ e_2(v, v_2)} ;V = V \ v; counter = 0
           end
7          if  e(v_1, v_2) ∉ E then
               E = E ∪ e(v_1, v_2)
           end
       end
       else
8          counter = counter + 1
       end
   end
   if |V| = 0 then
9      set flag=true
   end
10 return flag
```

- Step 8: if step 3 and step 5 are false then it 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 9: after exiting from the loop it check $|V|$ is empty or not. $|V|$ empty means we are able to delete all nodes so $T(V, E)$ is a partial 2-tree and it assign *true* to the $flag$.
- Step 10: Finally it returns the $flag$.

3.2.2 Function TwoTree

We are given $T(V, E)$, a one tree generated by function OneTree() and a edge set E_1 which contains all possible edges between vertices set V excluding E. 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 $T(V, E)$, 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 [10] 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 an edge e from E_1 which has the connectivity value greater than or equal to any other edge in E_1.
- Step 3: use the check function to check after adding this edge to $T(V, E)$, 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 edge-set E of $T(V, E)$.
- Step 5: it remove the edge from the edge-set E_1
- Step 6: it returns the partial 2-tree constructed by this greedy technique.

Algorithm 3: Function TwoTree($T(V, E)$)

Input: $T(V, E)$ is a 1-tree and E_1 is a edge set 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| < 2 * |V| − 3) do
2 |   select a edge e(v1, v2) from E1 which has highest connectivity probability.
3 |   if check(T(V, E ∪ e(v1, v2)) then
4 |   |   E = E ∪ e(v1, v2)
  |   end
5 |   E1 = E1 \ e(v1, v2)
  end
6 return T(V, E)
```

3.3 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.

3.4 Summary

4 $A - CONN$ Problem

4.1 Introduction

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.

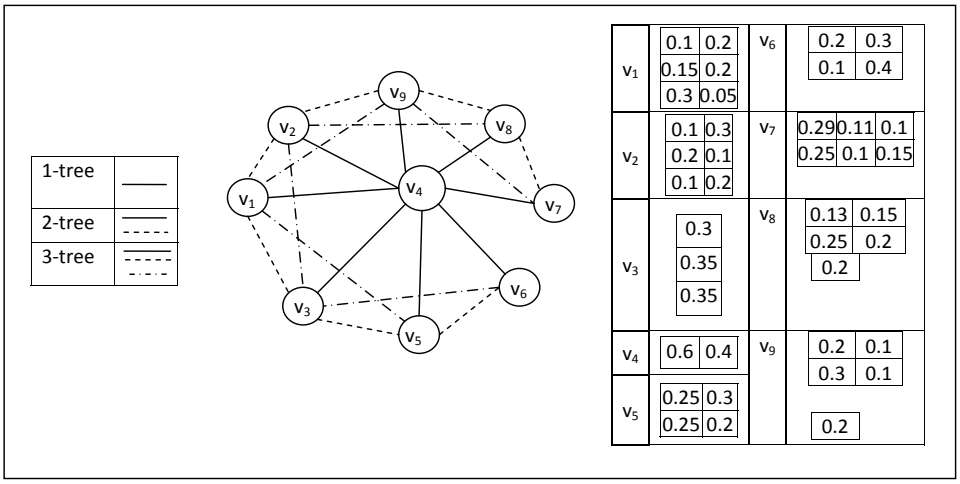


Figure 1: A UWSN modelled by a 3-tree.

4.2 Algorithm for $A - CONN$ Problem

4.2.1 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 4.1. Fig. 1 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 **??**.

4.2.2 Function Main

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

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 4.2.1.
- 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 4.2. One of the PES of the 3-tree depict in fig. 1 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 $< v_2, v_3, v_4 >$ ■

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.

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
        remove node v_i from $Temp$ and assign the result to $T_{(v_i,base)}$
    end
end
return *Prob=$\sum$(All probability for single partition in the remaining table)*

4.2.3 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

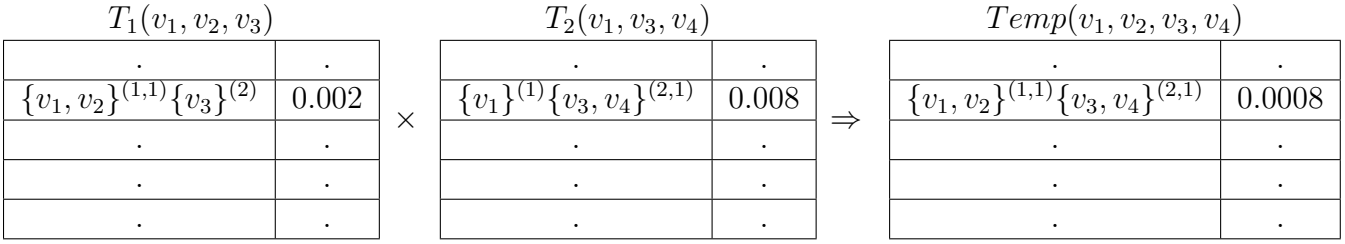


Figure 3 a). Merging two table into $Temp$

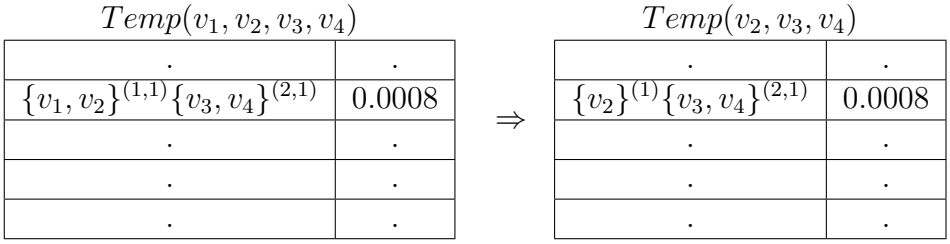


Figure 3 b). Deleting node v_1 from $Temp$

- Step 12 : return table T.

Example 4.3. 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
```

4.2.4 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.

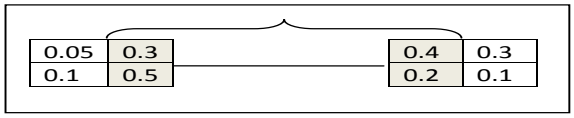


Figure 2: A simple tree with 2 nodes

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
```

4.3 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 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 $0.8 \times 0.6 = 0.48$.

4.4 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.

4.5 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 $= N \times (l_{max}^k \times 2^k)^{k+1}$

4.6 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.

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.

Table 1: 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 2: 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

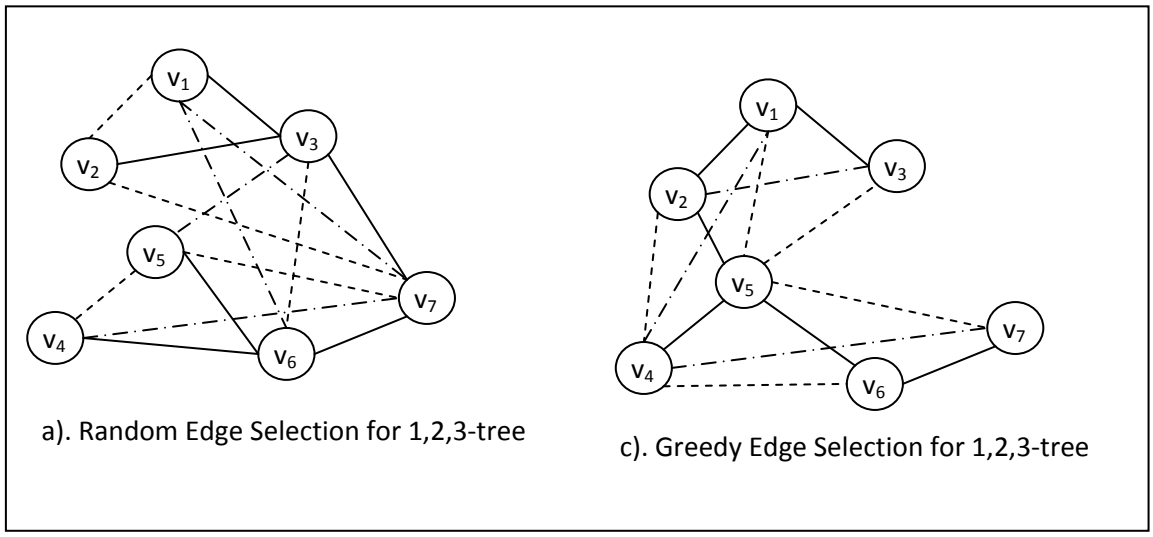


Figure 3: Greedy Vs Random Edge selection

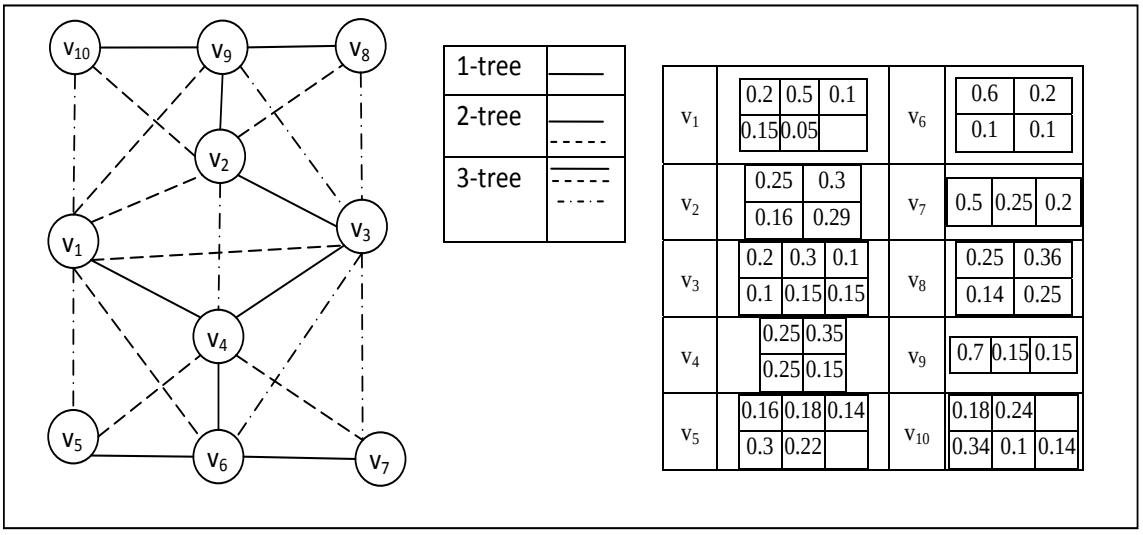
a) k **versus running time** table 1 shows three scenarios running time with respect to k of. In every scenario the running time of the algorithm increases with k. We also observe that difference between the running time 1-tree and 2-tree are 10 to 15 times where the running time between 2-tree and 3-tree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

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 7 illustrates connectivity versus transmission range when the algorithm selects randomly. We have the following observation from fig 7.

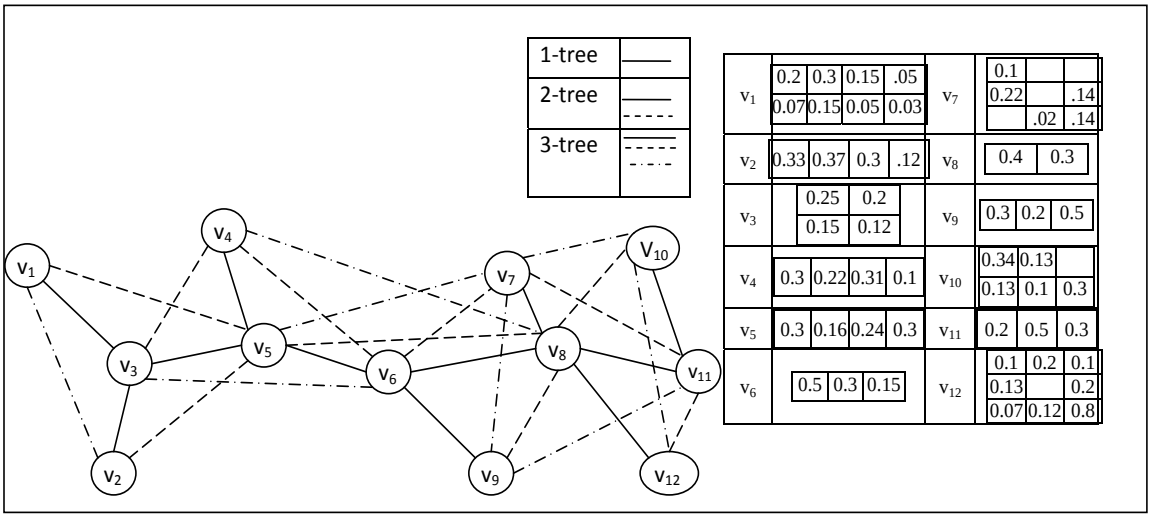
- with increase of transmission range the connectivity is also increasing. This is due to,increase of transmission range the probability of an edge between two nodes also increasing.
- connectivity for 2-tree is always greater than or equal to the connectivity of 1-tree for same transmission range. This 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 8 illustrates connectivity versus transmission



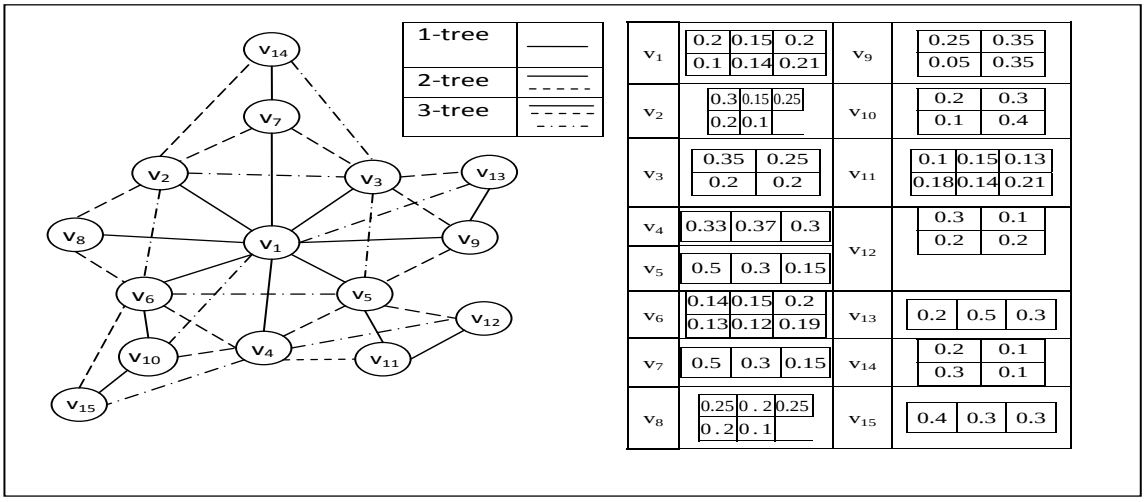
v_1	0.2	0.5	0.1	v_6	0.6	0.2	
	0.15	0.05			0.1	0.1	
v_2	0.25	0.3		v_7	0.5	0.25	0.2
	0.16	0.29					
v_3	0.2	0.3	0.1	v_8	0.25	0.36	
	0.1	0.15	0.15		0.14	0.25	
v_4	0.25	0.35		v_9	0.7	0.15	0.15
	0.25	0.15					
v_5	0.16	0.18	0.14	v_{10}	0.18	0.24	
	0.3	0.22			0.34	0.1	0.14

Figure 4: NetworkI



v_1	0.2	0.3	0.15	.05	v_7	0.1			
	0.07	0.15	0.05	0.03		0.22		.14	
							.02	.14	
v_2	0.33	0.37	0.3	.12	v_8	0.4	0.3		
v_3	0.25	0.2			v_9	0.3	0.2	0.5	
	0.15	0.12							
v_4	0.3	0.22	0.31	0.1	v_{10}	0.34	0.13		
						0.13	0.1	0.3	
v_5	0.3	0.16	0.24	0.3	v_{11}	0.2	0.5	0.3	
v_6	0.5	0.3	0.15		v_{12}	0.1	0.2	0.1	
						0.13		0.2	
						0.07	0.12	0.8	

Figure 5: NetworkII



v_1	0.2	0.15	0.2	v_9	0.25	0.35	
	0.1	0.14	0.21		0.05	0.35	
v_2	0.3	0.15	0.25	v_{10}	0.2	0.3	
	0.2	0.1			0.1	0.4	
v_3	0.35	0.25		v_{11}	0.1	0.15	0.13
	0.2	0.2			0.18	0.14	0.21
v_4	0.33	0.37	0.3	v_{12}	0.3	0.1	
					0.2	0.2	
v_5	0.5	0.3	0.15				
v_6	0.14	0.15	0.2	v_{13}	0.2	0.5	0.3
	0.13	0.12	0.19				
v_7	0.5	0.3	0.15	v_{14}	0.2	0.1	
					0.3	0.1	
v_8	0.25	0.2	0.25	v_{15}	0.4	0.3	0.3
	0.2	0.1					

Figure 6: NetworkIII

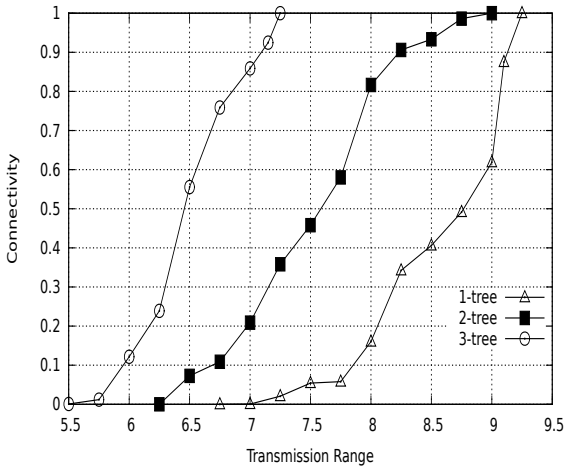


Figure 7: Connectivity versus transmission range with random edge selection.

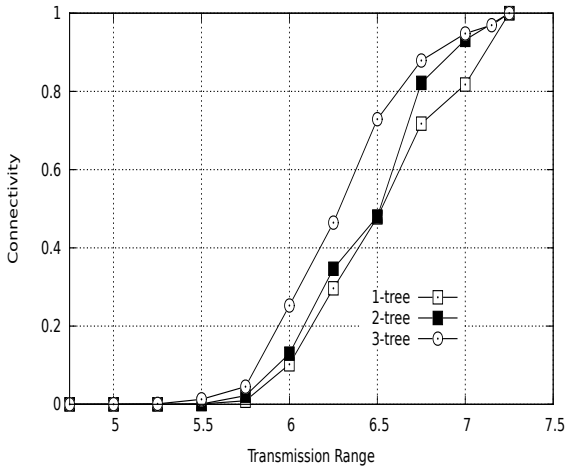


Figure 8: Connectivity versus transmission range with greedy edge selection.

range when we are selecting edges by greedy techniques. In greedy techniques those edges are selected which has a high probable values. Fig 3 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.

4.7 Summary

5 $A-CONN$ with Relays Problem

5.1 Introduction

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.

Example 5.1. Fig 9 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

5.2 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

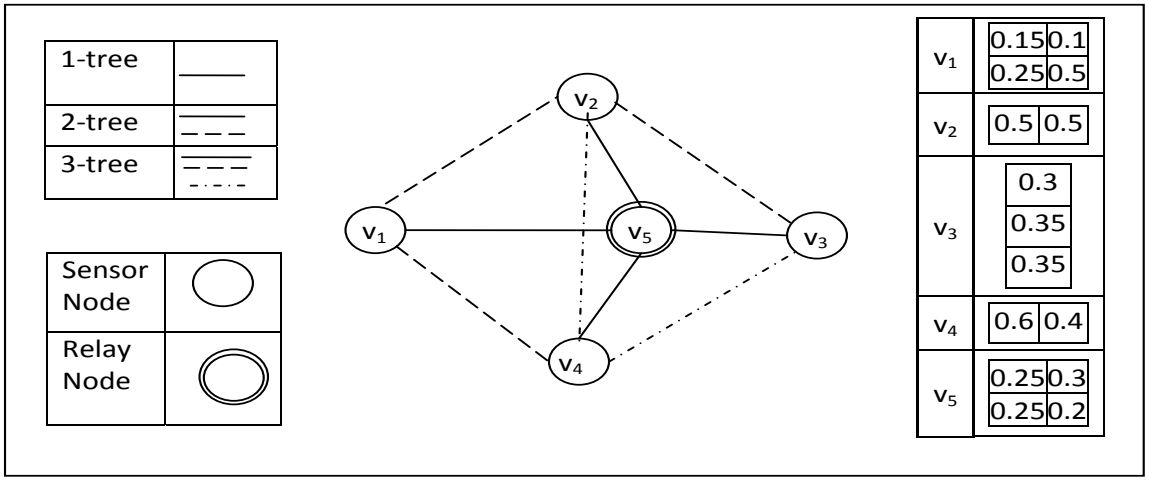


Figure 9: A partial 2-tree with 6 nodes.

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.

5.3 Algorithm for $A - CONN$ with Relays Problem

5.3.1 Key Data Structures

We know from section 4.2.1 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.

$T_1(v_1, v_2, v_3)$

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

$\times$

$T_2(v_1, v_3, v_4)$

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

$\Rightarrow$

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

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

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

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

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

$\Rightarrow$

$Temp(v_2, v_3, v_4)$

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

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

5.3.2 Function Main

- 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)*

5.3.3 Function Merge

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 3: 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.

5.4 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 3 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.10
- 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.10 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.

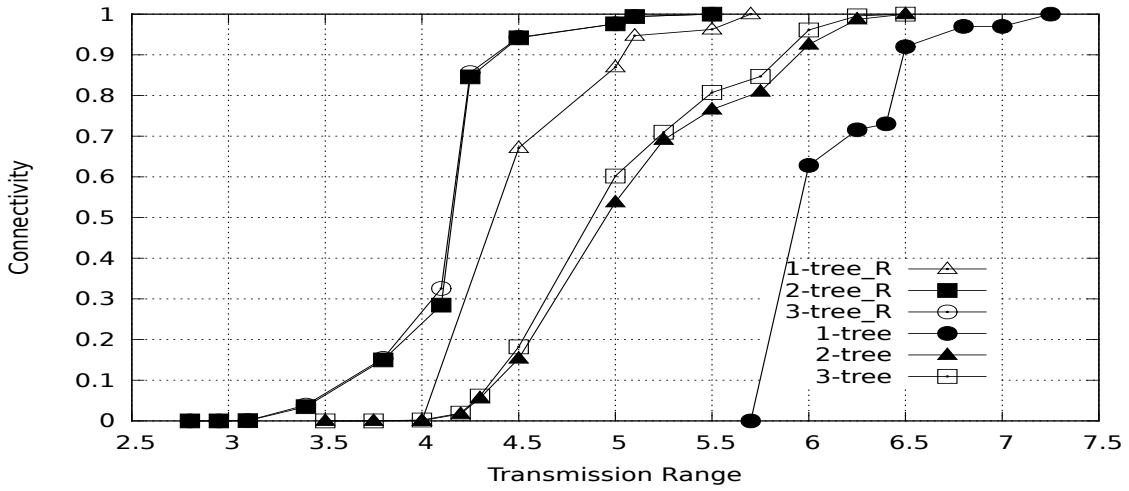


Figure 10: Connectivity Vs Transmission range for network with relays and network without relays

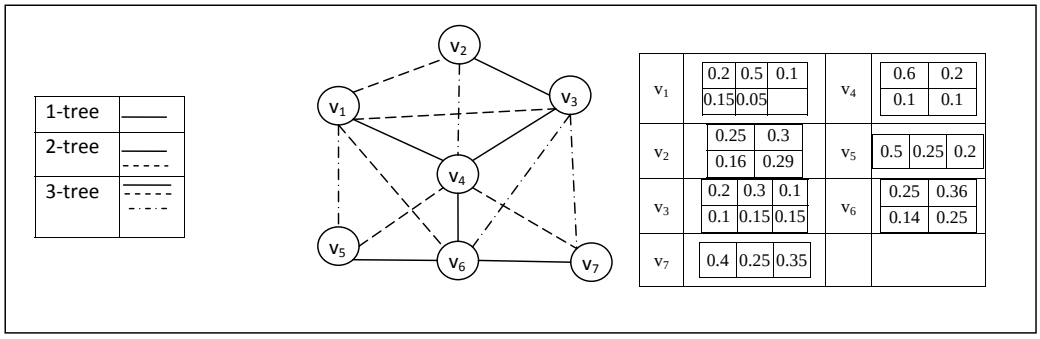


Figure 11: A partial 3-tree with 7 nodes

5.5 Summary

6 $S - CONN$ Problem

6.1 Introduction

6.2 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 6.1. Fig. 11 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. ■

6.3 Algorithms for $S - CONN$

6.3.1 Data Structures

We know from section 4.2.1 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 4.2.4. We modified the $merge()$ function which is used for merging two table described in section 4.2.3 and $Main()$ function which is described in section 4.2.2. 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.

6.3.2 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.

- $Step10$: 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.

6.3.3 Function Merge

In this section steps 6 and 7 are added with the merge function which is mainly described in section 4.2.3

Algorithm 9: 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)*

- *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 10: 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
```

6.4 Simulation Results

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 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.

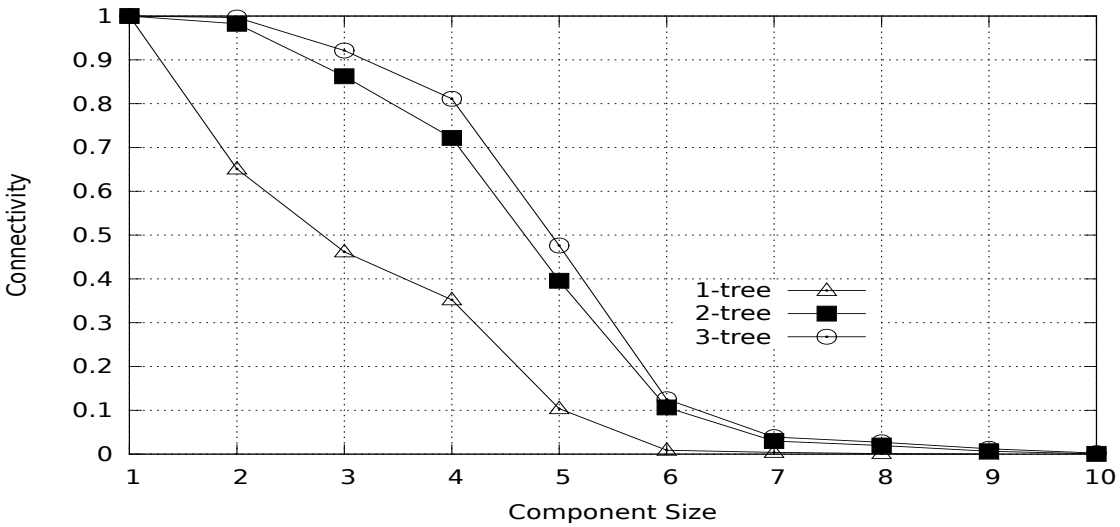


Figure 12: Connectivity versus component size for Newtork I

6.5 Summary

7 $S-CONN$ with Relays Problem

7.1 Introduction

7.2 Algorithms for $S-CONN$ with Relays

7.2.1 Data Structure

7.2.2 Function Main

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 $T_{(v_i,base)}$

7.2.3 Function Merge

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
```

7.3 Simulation Results

7.4 Summary

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] Ian F Akyildiz, Dario Pompili, and Tommaso Melodia. Underwater acoustic sensor networks: research challenges. *Ad hoc networks*, 3(3):257–279, 2005.

[3] 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.

[4] 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.

[5] Bridget Benson, Grace Chang, Derek Manov, Brian Graham, and Ryan Kastner. Design of a low-cost acoustic modem for moored oceanographic applications. In *Proceedings of the 1st ACM international workshop on Underwater networks*, pages 71–78. ACM, 2006.

[6] Matt Bromage, Katia Obraczka, and Donald Potts. Sea-labs: a wireless sensor network for sustained monitoring of coral reefs. In *NETWORKING 2007. Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet*, pages 1132–1135. Springer, 2007.

[7] 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.

[8] Nuno A Cruz and Jose C Alves. Ocean sampling and surveillance using autonomous sailboats. 2008.

[9] Jun-Hong Cui, Jiejun Kong, Mario Gerla, and Shengli Zhou. The challenges of building mobile underwater wireless networks for aquatic applications. *Network, IEEE*, 20(3):12–18, 2006.

[10] 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.

[11] 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.

[12] Lee Freitag, Milica Stojanovic, Matthew Grund, and Sandipa Singh. Acoustic communications for regional undersea observatories. *Proceedings of Oceanology International, London, UK*, 2002.

[13] Dale Green. Acoustic modems, navigation aids, and networks for undersea operations. In *OCEANS 2010 IEEE-Sydney*, pages 1–6. IEEE, 2010.

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] A Quazi and W Konrad. Underwater acoustic communications. *Communications Magazine, IEEE*, 20(2):24–30, 1982.

[23] Joseph Rice, Bob Creber, Chris Fletcher, Paul Baxley, Ken Rogers, Keyko McDonald, Dave Rees, Michael Wolf, Steve Merriam, Rami Mehio, et al. Evolution of seaweb underwater acoustic networking. In *Oceans 2000 MTS/IEEE Conference and Exhibition*, volume 3, pages 2007–2017. IEEE, 2000.

[24] Sumit Roy, Payman Arabshahi, Dan Rouseff, and Warren Fox. Wide area ocean networks: architecture and system design considerations. In *Proceedings of the 1st ACM international workshop on Underwater networks*, pages 25–32. ACM, 2006.

[25] 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.

[26] Roman E Shangin and Panos M Pardalos. Heuristics for minimum spanning k-tree problem. *Procedia Computer Science*, 31:1074–1083, 2014.

[27] Sandipa Singh, Sarah E Webster, Lee Freitag, Louis L Whitcomb, Keenan Ball, John Bailey, and Chris Taylor. Acoustic communication performance of the whoi micromodem in sea trials of the nereus vehicle to 11,000 m depth. In *OCEANS 2009, MTS/IEEE Biloxi-Marine Technology for Our Future: Global and Local Challenges*, pages 1–6. IEEE, 2009.

[28] Milica Stojanovic, J Catipovic, and John G Proakis. Adaptive multichannel combining and equalization for underwater acoustic communications. *The Journal of the Acoustical Society of America*, 94(3):1621–1631, 1993.

[29] Beineke L. W. and Pippert R. E. Properties and characterizations of k -trees. *Mathematika*, 18:141–145, 1971.

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

[31] Peng Xie, Jun-Hong Cui, and Li Lao. Vbf: vector-based forwarding protocol for underwater sensor networks. In *NETWORKING 2006. Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications Systems*, pages 1216–1221. Springer, 2006.

[32] Hai Yan, Zhijie Jerry Shi, and Jun-Hong Cui. Dbr: depth-based routing for underwater sensor networks. In *NETWORKING 2008 Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet*, pages 72–86. Springer, 2008.

[33] 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.

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

[35] 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.