University of Alberta

Master Thesis

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

Author:
Md Asadul Islam

Supervisor:
Ehab S. Elmallah

June 2014

Contents

List of Figures

List of Tables

‘Figure1.pdf’ n

Chapter 1

Introduction

Underwater Sensor Networks (UWSNs) is one of the enabling technologies for the development of ocean-observation systems. UWSNs is an attractive choice to researchers and industry people because of its remote monitoring and control technology. Application domain of 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.

The chapter starts by presenting some applications domain of UWSNs, different UWSNs deployment strategies and design of a protocol stack for underwater acoustic communication . Next, we provide an overview of some related research problem tackles by researchers 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 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 [1] 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] [3] [4]. According to [3] 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 [5].
- 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 [6], 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 [3], 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 [7], 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 bio-geochemical 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 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 [8], [3], [2], [9], [10], [4], [11] and [12]. Also there are some real-world research include [13], [14] [15] [16] [17].

- **Physical layer:** Electromagnetic communication such as radio and optical signal quickly absorbs by water makes 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., geometric spreading, man made noise and ambient noise, multi-path propagation, high delay due to low data rate and Doppler effects 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 [18]) and for research prototype (e.g. micro modem development at WHOI [19]).

  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 [20] (such as phase shift keying (PSK) and quadrature amplitude (QAM)) and are using today's 'high-speed' communications.

- **Data Link Layer:** Multiple access is the techniques which allow devices to access a common medium, sharing the limited available bandwidth in an efficient and fair way. Long delays, frequency-dependent attenuation and the relatively long reach of acoustic signals due to the peculiarities of the underwater channel are some challenges for link layer design.

  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.

  Code division multiple access (CDMA) is a technique where signals coexist in both time and frequency. In-order to separate the signal a specially design code is used. That specifically designed code along with signal processing techniques allows multiple devices transmit simultaneously over entire frequency band. Effectiveness on multi-path fading and no need of slot synchronization makes CDMA an attractive choice for UWSNs link layer protocol.

  ALOHA is a class of Medium Access Control (MAC) protocols that detect collision and retransmit lost packets without trying to prevent the collision. Due to the slow propagation of the acoustic channel ALOHA protocols are affected by low efficiency in Underwater Acoustic communication. 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 [21] 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 [22]. 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 increase 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 [23], [24],[25], routing [26],[27],[28]. 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 with and without relays: every node is connected with sink in the presence or absence of relay nodes and connectivity with N_{req} with and without relays: compute the probability that at least $N_{req} - 1$ nodes is connected with sink node in the presence or absence of relay nodes.

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

Recent years has seen tremendous advances in underwater acoustic communication, node localization, routing, connectivity and coverage issue. In this section we present a overview of some research related to our area of interest. We can categorize them in the following 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 [29] proposes a UWSNs 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 [30] 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 [31] analyse the connectivity and k-coverage issues in 3D UWSNs, 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 UWSNs. They prove that 3D k-covered UWSNs 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 UWSNs based on an accurate characterization of k-coverage of 3D fields.

4. In [32] 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 [33], 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 [34] 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

Several experimental and analytical results in oceanography literature have shaped our current understanding of mobility for underwater sensor networks. Of the vast literature existing in the field, we recall the following early landmark results.

1. In [35], the authors report on several observations collected in the Gulf Stream using thirty-seven RAFOS drifters launched off Cape Hatteras. Mobility of the free floating drifters are tracked for 30 or 45 days. In [36], the author describes a 2-dimensional kinematic model of a meandering jet. The model captures the striking patterns of cross-stream and vertical motion associated with meanders observed in [35].

2. Investigations and results obtained in the above directions have been valuable for networking researchers approaching the challenge of modelling the mobility of underwater sensor networks. In [37], the authors adopt a kinematic model for capturing the effect of meandering sub-surface currents and

vortices of free floating sensor nodes. The model, called the meandering current mobility model, is useful for large coastal environments that span several kilometers. It captures the strong correlations in mobility of nearby sensor nodes. Using simulation, the authors investigate several network connectivity, coverage, and localization aspects.

3. In [38], [39], the authors consider sensor nodes with movement capability. Each node incurs both uncontrollable and controllable mobility (abbreviated as U-mobility and C-mobility). Using a grid layout that divides a geographic area into cells, the authors adopt a probabilistic U-mobility model. The model takes into consideration two types of effects: local variety effects (caused by reefs, turbulence), and main circulation effects (caused by wind, salinity). Using such model, the authors present an energy efficient approach for satisfying network coverage requirements.

In this thesis, we adopt a simple probabilistic locality model where the geographic area under consideration is partitioned into disjoint regions (rectangles) using a grid layout. In our model, the use of a high grid resolution (i.e., a layout with small regions) can potentially give accurate results at the expense of decreased solution efficiency. We assume that one can utilize a physical model of underwater currents to compute the probability that a sensor node is located at a given region during some time interval of interest. Using such probability distribution, one obtains a probabilistic graph model of the network.

Our work here formalizes four problems, denoted $A-CONN$, $AR-CONN$, $S-CONN$, and $SR-CONN$ that call for determining the likelihood that a probabilistic graph is entirely or partially connected. Our main contribution is an efficient dynamic programming algorithm to solve above four problems on tree-like networks. The algorithm can be used to derive lower bounds on the solution of any arbitrary probabilistic network. Our devised algorithm extends a result in [40] to compute the probability that a given sequence of nodes in a probabilistic network forms a simple connected path. To the best of our knowledge, the problems formulation and devised algorithm are novel aspects of the paper.

In the next sections we outline the system model and problem formulations.

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 $A-CONN$, $AR-CONN$, $S-CONN$ and $SR-CONN$. 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.

1.3.1 Node Locality Sets

We denote by $V = V_{sense} \cup V_{relay}$ the set of nodes in a given UWSN G where V_{sense} is a subset of sensor nodes that can perform both sensing and data communication, and V_relay is a subset of relay only nodes that do not perform sensing. We assume that V_{sense} has a distinguished *sink* node, denoted s, that performs network wide command and control functions. After some time interval T from network deployment time, each node x can be in some location determined by water currents causing node movement.

To simplify analysis, approaches in the literature typically divide the geographic area containing nodes into rectangles of a superimposed grid layout. Thus, at time T , each node x can be in any one of a possible set of grid rectangles denoted $R_x = \{r_{(x,1)}, r_{(x,1)}, ...\}$. We call R_x the locality set of x (for simplicity, we omit the dependency on T from the notation). Depending on the mobility model induced by water currents, node x can be in any possible grid rectangle $r_{(x,i)}$ with a certain probability, denoted $p(r_{(x,i)})$. Computing such probability distribution is outside the scope of the thesis. We assume, however, that such distribution is computable given enough information on the dynamical aspects of the water currents.

Henceforth, we use $r_{(x,i)}$ to refer to node x at the ith location index. For brevity, we also refer to $r_{(x,i)}$ as the location of x (rather than the grid rectangle containing x) at an instant of interest. To gain efficiency in solving large problem instances with large locality sets, it may be convenient to truncate some locality sets to include only locations of high occurrence probability, and ignore the remaining locations. In such cases, we get $\sum_{r_{(x,i)} \in R_x} p(r_{(x,i)}) \leq 1$, if R_x is truncated.

1.3.2 Node Reachability

At any instant, node x can reach node y if the acoustic signal strength from x to y (and vice versa) exceeds a certain threshold value. In acoustic UWSN, the directions of water currents play an important role in signal delay (see, e.g., [17]). For simplicity, we assume that given the exact locations of x and y , we can determine if x and y can reach each other, and if so, we set the link indicator $Reach(x, y) = 1$. Else, if no satisfactory communication can take place then we set $Reach(x, y) = 1$.

Our general objective in this paper is to develop effective methodologies for computing lower bounds on the likelihood that the network is totally, or partially, connected. To this end, we adopt the following rule: we set $Reach(r_{(x,i)}, r_{(y,j)}) = 1$ if and only if the two nodes x and y can reach each other if they are located anywhere in their respective rectangles $r_{(x,i)}$ and $r_{(y,j)}$.

The above rule implies that connectivity between x and y is ignored if they can reach each other at some (but not all) pairs of points in their respective rectangles. As can be seen, ignoring connectivity in such cases results in computing lower bounds on the network connectivity, as required. Our devised algorithm presented below is exact with respect to the given relation *Reach* that defines the input network G.

1.3.3 Problem Definition

The first problem is all sensor nodes connectivity problem:

Definition 1.1 (**the** $A - CONN$ **Problem**)**.** Given

- a UWSN G.
- a set of sensor nodes V_{sense}.
- a sink node s and $s \in V_{sense}$.
- 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 $Conn(G)$ 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 1.2 (the $AR - CONN$ Problem). Given

- a UWSN G.
- a set of sensor nodes V_{sense}.
- a sink node s and $s \in V_{sense}$.
- a set of relay nodes V_{relay}. So the set of all nodes $V = \{V_{sense} \cup V_{relay}\}$.
- 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 $Conn(G)$ 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 1.3 (the $S - CONN$ Problem). Given

- a UWSN G.
- a set of sensor nodes V_{sense}.
- a sink node s and $s \in V_{sense}$.
- 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 1.4 (the $SR - CONN$ Problem). Given

- a UWSN G.
- a set of sensor nodes V_{sense}.

- a set of relay nodes V_{relay}. So the total number of nodes in the network $V = \{V_{sense} \cup V_{relay}\}$
- a sink node s and $s \in V_{sense}$.
- 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 $AR - CONN$, $S - CONN$ and $SR - CONN$ problems.

2. The $A - CONN$ problem is a special case of $AR - CONN$ 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 $AR - CONN$. But after adding relay nodes with the network the connectivity provided by $AR - CONN$ for graph G will be higher than the connectivity provided by $A - CONN$ for the same graph.

3. The $A - CONN$ is a special case of the $S - CONN$ problem where $n_{req} \leq |V_{sense}|$. In the $A - CONN$ problem, the network is operating if the *sink* 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. When $n_{req} = V_{sense}$ then both problem are same.

4. Similarly the $S - CONN$ problem is a special case of $SR - CONN$. Without relay nodes both provides same connectivity for same graph. In the presence of relay node the connectivity provided by $SR - CONN$ will be superior that the connectivity provided by $S - CONN$ because relay nodes enhance connectivity.

We next remark that the above problems share some basic aspects with the class of network reliability problems discussed in [41]. In particular, all such problems are defined over some type of probabilistic graphs where each node (and/or link) can be in any one of two, or more, states. In case of network reliability problems, a node or link can either be operating or failed, whereas in our present context, a node can be in any one of a possible set of locations.

Events of interest on such probabilistic graphs occur when the given network is in some particular network states. In our present context, a network state S of G arises when each node $x \in V$ is located at some specific location in its respective locality set R_x. Thus, if $V = \{v_1, v_2, ..., v_n\}$ then a state S of V can be specified by $\{r_{(v_1,i_1)}, r_{(v_2,i_2)}, ..., r_{(v_n,i_n)}\}$ where each $r_{(v_\alpha,i_\alpha)} \in R_{v_\alpha}$. Two states S_1 and S_2 are different if they differ in the location of at least one node. Assuming node locations are independent of each other, we have $Pr(S) = \prod_{v_\alpha \in V} p(r_{(v_\alpha,i_\alpha)})$.

In the $A-CONN$ and $AR-CONN$ problem, a state is operating if the *sink* s can reach all sensor nodes in V_{sense} . Likewise, in the $S-CONN$ and $SR-CONN$ problem, a state S is operating if the *sink* node s can reach a subset having at least n_{req} sensor nodes. Let S be the set of all operating states S of a given problem then the required solution is given by $\sum_{S \in \mathbf{S}} Pr(S)$.

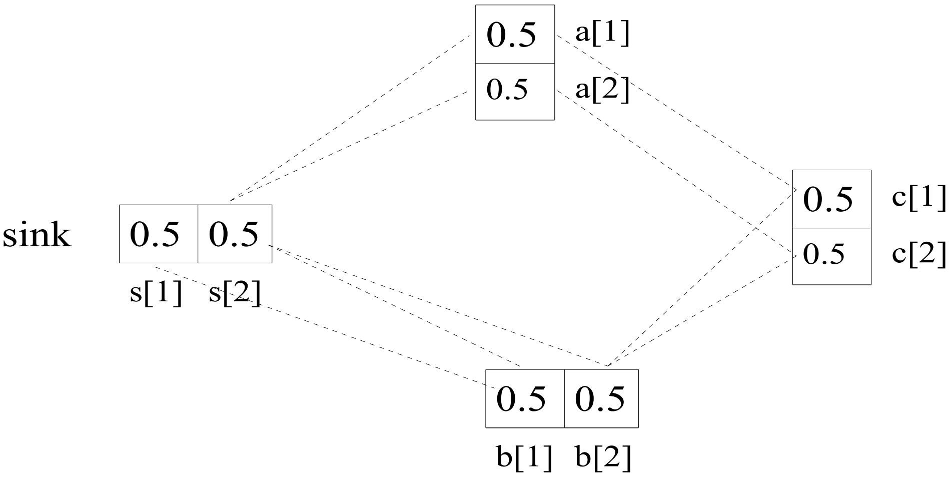


Figure 1.1: An example network with locality sets.

Example 1.1. *Fig 1.1 illustrates a probabilistic graph on 4 nodes where* $V = \{s, a, b, c\}$ *and locality set of each node has 2 locations. So the network has* 2^4 *states. For* $A-CONN$ *problem, state* $S_1 = \{r_{(s,2)}, r_{(a,2)}, r_{(b,2)}, r_{(c,2)}\}$ *is operating and state* $S_1 = \{r_{(s,1)}, r_{(a,2)}, r_{(b,2)}, r_{(c,2)}\}$ *is failed.*

1.4 Thesis Contribution and Organization

Main contribution of this thesis is as follows

1. In chapter 2, we present some background on k-tree. We also present two greedy algorithm which construct 1-tree and 2-tree from given set of nodes. This constructed trees are used by four problems as a input in-order to find out connectivity among the given set of nodes.
2. In chapter 3, we investigate the $A-CONN$ problem. We design an efficient dynamic program to provide an exact solution for $A-CONN$ problem. we investigate about the correctness, running time and verification cases of the algorithm. Finally we provide some simulation results.
3. In chapter 4, we modified the algorithm used to solve $A-CONN$ problem and use to solve $AR-CONN$ problem. We verified our claim that relays enhance the performance of algorithm by comparing the result obtained by solving $AR-CONN$ problem and $A-CONN$ problem.
4. In chapter 5, again we modified our algorithm we used to solve $A-CONN$ problem in-order to solve $S-CONN$ problem. We verified that when $n_{req} = V_{sense}$, the connectivity provided by $A-CONN$ and $S-CONN$ are same in the simulation result section.
5. In chapter 6, we modified the algorithm used to solve $S-CONN$ problem so that we are able to add relay nodes with the network. We show that the performance of $SR-CONN$ is better than $S-CONN$ in-order to assure connectivity of a network by simulation results.

The rest of this thesis is organized as follows. Chapter 2 provides a background about k-tree, partial k-tree and show how to construct k-tree by greedy technique. Chapter 3 to 6 present the four problems, their solutions and results.

1.5 Summary

In this chapter we present some applications of UWSNs, different types of deployment strategy used in UWSNs, five layers of internet protocol stack used in UWSNs. Then we provide a literature survey on connectivity and coverage, k-tree and partial k-tree and on mobility models. After that we provide system models and we formulate four problems. finally we present thesis contribution and organization.

Chapter 2

Background on k-tree

In this chapter, we start 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.

2.1 Introduction

The concept of k-tree was first introduced by Beineke and Pippert [42], [43]. A clique with k vertices is considered to be a k-tree. A k-treecan be reduced to a k-complete graph by repeatedly removing vertices of degree k. The order in which the vertices are eliminated are called one of the perfect elimination order (PES). A partial k-trees are subgraph of k-tree with same number of vertices but possibly less number of edges.

According to [44] the application domain of partial k-tree are reliability of communication networks with the presence of line failure, concurrent broadcasting in a common medium network, reliability evaluation in complex systems and evaluation of queries in relational database systems. Next section we are define k-tree.

2.2 Definition of k-tree

In this section, we define clique, k-tree,Treewidth, partial k-tree and we present different classes of k-tree.

Definition 2.1. A clique is a set of vertices that induce a complete subgraph of a graph G. ■

Definition 2.2. 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 2.3. 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 and some liner algorithms for graphs with treewidth bounded by a constant [45]. 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. 2.1 depict a partial k-tree where $k = 1, 2, 3$. The side of the k-tree represent each node with locality set with their corresponding probability. For example the locality set of node v_1 and v_6 are 4 and 6 respectively. The solid lines with nodes represent 1-tree. Dashed lines with the solid lines with nodes represent 2-tree and dot-dashed lines, dashed lines and solid lines represent 3-tree.* ■

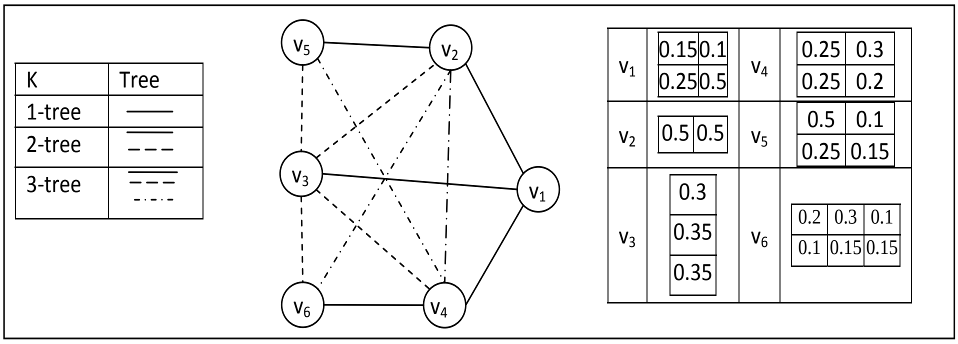


Figure 2.1: partial k-tree of 6 nodes with locality sets.

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 2.4 (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. Fig 2.1 depict a k-tree of 6 nodes. There are three simplicial vertices in the k-tree namely v_1, v_5 and v_6. v_1 is our sink node so we are not eliminating v_1. So we can start eliminate either v_5 or v_6. So one of the PES will be $\{v_5, v_6, v_3\}$. ■

2.4 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.

2.4.1 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 $Tree(V, E)$
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.
while $N! = \emptyset$ do
    $p \in N$ 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.
    add $(p, q)$ to $E$.
    delete node $p$ from $N$ and add $p$ to $T$
end
Return$Tree(V, E)$
```

2.4.2 Extracting 2-tree

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

2.4.2.1 Function Check

Authors in [46] shows that the recognition of partial k-tree for an arbitary k is NP-hard problem but it can be done in polynomial time for fixed k. Here we are using the check function which takes a tree as a input and check that whether

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 $v \in V$ and $counter < |V|$ do
3      if $deg(v) == 1$ then
4          if $e(v, v_t) \in E$ then
               $E = E \setminus e(v, v_t)$; $V = V \setminus v$; $counter = 0$
           end
       end
5      else if $deg(v) == 2$ then
6          if $e_1(v, v_1) \in E$ and $e_1(v, v_2) \in E$ then
               $E = E \setminus \{e_1(v, v_1) \cup e_2(v, v_2)\}$ ;$V = V \setminus v$; $counter = 0$
           end
7          if $e(v_1, v_2) \notin E$ then
               $E = E \cup e(v_1, v_2)$
           end
       end
       else
8          $counter = counter + 1$
       end
   end
   if $|V| = 0$ then
9      set $flag$=$true$
   end
10 return $flag$
```

it's a partial 2-tree or not. If the input tree is a partial 2-tree then it return *true* otherwise it return $false$.

- Step 1: it initializes a *counter* by 0 which terminates 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 v 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.
- 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 $flag$.

2.4.2.2 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 [47] 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 $e(v_1, v_2)$ from $E_1$ which has highest connectivity probability.
3     if check($T(V, E \cup e(v_1, v_2))$) then
4         $E = E \cup e(v_1, v_2)$
      end
5     $E_1 = E_1 \setminus e(v_1, v_2)$
  end
6 return $T(V, E)$
```

2.4.3 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.

2.5 Summary

In this chapter we present some background on k-tree and partial k-tree. We define perfect elimination sequence and why it is necessary for our algorithm.Finally we present two greedy algorithm to construct 1-tree and 2-tree from a given set of nodes.

Chapter 3

All node connectivity $A - CONN$ Problem

In this chapter, we present an algorithm to compute an exact solution for $A - CONN$ 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 G instance. The function uses a dynamic programming framework to solve $A - CONN$ problem. Then we present some verification cases, correctness and running time of our presented exact algorithm. Finally we conclude the chapter after presenting some simulation results.

3.1 Introduction

In semi-mobile and mobile architecture of UWSNs, nodes appear and disappear with time. As a result connectivity is going to change continuously and it is very challenging to measure connectivity of such type of networks. Connectivity of this type of dynamic graph depends on node placement, water currents, salinity, temperature etc.

We device a scheme where we approximate the UWSN by 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 time instant is known from the initial location of the node and underwater current.

We present an exact solution for all node connectivity $A-CONN$ problem and we use dynamic programming framework for implementation purpose. Next section we present an overview of algorithms in-order to solve $A - CONN$ problem.

3.2 Overview of Algorithms

The function main in Fig. 11 along with function merge and function pMerge employs dynamic programming approach. It takes as input an instance G of $A - CONN$ problem, a tree $T = (V, E)$ and PES, a perfect elimination sequence $(v_1, v_2, ..., v_{n-k})$ of T. The function computes the exact solution $Conn(G)$ of the given instance.

We consider $T(V, E)$ as a partial k-tree ($k = 1, 2, 3, ...$) where each node, $v \in V$ is 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.

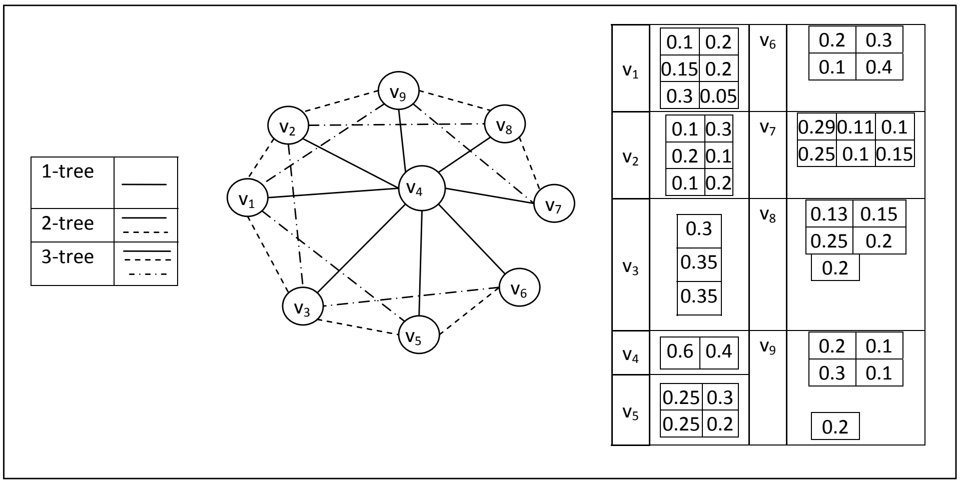
The key variables and data structures in three functions are as follows.

- $\{T_{(v,1)}, T_{(v,2)}, ..., T_{(v,k)}\}$: tables associated with clique $\{K_{(v,1)}, K_{(v,2)}, ..., K_{(v,k)}\}$ respectively.
- $\{K_{(v,1)}, K_{(v,2)}, ..., K_{(v,k)}\}$: are k-cliques associated with vertex v in a k-tree, where v is a simplicial vertex. Each clique associated with a table. Each row in the table defines key-value mapping. A key consists of i). partitions, *par*, ii).regional sets for $A - CONN$ problem and a value is a probability which is the multiplication of regional probability of nodes for the clique.

  i). partitions, *par*: There can be more than one partition associated with each row. Each partition consists with one or more nodes. 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 $\{v_1, v_2\}^{(1,1)}{v_3}^{(2)}$, there are two partitions including $\{v_1, v_2\}$ and $\{v_3\}$. Also node v_1 can reach node v_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 v_3 when node v_3 is in region 2 of it's locality set.

ii). regional set, *locmap*: The regional set of a partition consists of position(s) of corresponding node(s) in their locality set(s). A regional set is indicated as a superscript of a partition and is surrounded by parentheses. In-order to maintain regional set we use a map called location map, *locmap*. In *locmap*, we use every node in the partition as a key and corresponding location of that node in it's locality set as value. For example in the key $\{v_1, v_2\}^{(1,1)} v_3{}^{(2)}$, there are two regional sets $(1, 1)$ and (2) along 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.

- $K_{(v,base)}$: the base clique to which v is attached and it is formed by all adjacent vertices of v.
- $p(r_{(v,i)})$: the probability that node v is in region $r_{(v,i)}$.
- $Reach(r_{(x,i)}, r_{(y,j)}) = 1$ if node x in region $r_{(x,i)}$ can reach node y in region $r_{(y,j)}$ else 0.
- PES : an ordering $(v_1, v_2, ..., v_{n-k})$ of the nodes of $T(V, E)$ for every $i \in \{1, 2, .., n-k\}$:, the node v_i is simplicial in the subgraph of T.
- $Conn(G)$: 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 $G]$.



v_1	0.1 0.2 0.15 0.2 0.3 0.05	v_6	0.2 0.3 0.1 0.4
v_2	0.1 0.3 0.2 0.1 0.1 0.2	v_7	0.29 0.11 0.1 0.25 0.1 0.15
v_3	0.3 0.35 0.35	v_8	0.13 0.15 0.25 0.2 0.2
v_4	0.6 0.4	v_9	0.2 0.1 0.3 0.1 0.2
v_5	0.25 0.3 0.25 0.2		

FIGURE 3.1: A UWSN modelled by a 3-tree.

Example 3.1. Fig. 3.1 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 $\{v_1, v_2, v_3\}^{(1,4,2)}0.004$ where $\{v_1, v_2, v_3\}$ indicates all three nodes are in same partition, superscript $(1, 4, 2)$ indicates node v_1, v_2 and v_3 are in region $1, 4$ and 2 respectively of their corresponding locality set 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 3.2.2.

3.2.1 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.
- Steps 2-10 form the main loop of the $Main$ function. The loop iteratively deletes a node v_i according to PES after merging all the cliques associates with node v_i. Processing a node v_i done as follows.
- Step 3 finds all cliques, $K_{(v_i,1)}, K_{(v_i,2)}, .., K_{(v_i,k)}$ associate with node v_i and their corresponding table $T_{(v_i,1)}, T_{(v_i,2)}, .., T_{(v_i,k)}$.
- 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 table $Temp$ with base table of node v_i, $T_{(v_i,base)}$ by using $merge$ function and assign the result to $T_{(v_i,base)}$.

```
Algorithm 4: Function Main($G$, $T(V, E)$, $PES$)
Input: An instance of $A - CONN$ problem where $G$ has a $k$-tree topology $T(V, E)$
Output: $Conn(G)$
Notation: $Temp$ is a temporary table. Each row of $Temp$ is a key-value mapping.
1  Initialize every clique by a table.
2  for $i = 1, 2, ..., |V| - k$, according to the order of $PES$ 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      $T_{(v_i,base)} = merge(Temp, T_{(v_i,base)})$
8      if $v_i$ is in a partition of table $T_{(v_i,base)}$ by itself then
9          delete the row containing that partition.
       end
10     else
           remove node $v_i$ and it's corresponding location from every row of $T_{(v_i,base)}$
       end
   end
11 return $Pr = \sum$(All probability for single partition in the remaining table)
```

- Step 8-9 finds all rows in table $T_{(v_i,base)}$ where v_i is in a partition by itself i.e. bad set then the function simply removes all rows.
- Step 10 remove the vertex v_i and it's locality set from $T_{(v_i,base)}$.
- After exiting from the main loop, the current tree T contains only one clique associate with sink s. Step 11 computes the solution $Conn(G)$ from the remaining table associate with sink node.

Example 3.2. One of the PES of the 3-tree depict in fig. 3.1 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 $< v_2, v_3, v_4 >$ ■

$T_1(v_1, v_2, v_3)$

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

×

$T_2(v_1, v_3, v_4)$

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

⇒

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

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

⇒

$Temp(v_2, v_3, v_4)$

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

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

3.2.2 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 3.3. 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 $\{v_1, v_2\}\{v_3, v_4\}$. The partition $\{v_1\}$ merge with partition $\{v_1, v_2\}$ because there is a common node v_1 between these two partition. So after merging the result will be $\{v_1, v_2\} \cup \{v_1\} = \{v_1, v_2\}$. $\{v_1\}$ is not going to merge with $\{v_3, v_4\}$ because there is no common nodes between these two partition. Similarly partition $\{v_3\}$ merge with partition $\{v_3, v_4\}$ and the result is $\{v_3, v_4\}$. 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
set $C$ = the set of common vertices between $T_1$ and $T_2$ , set $Prob\_C = 1$
if $C \neq \emptyset$ then
    foreach row $r$ in $T_1$ do
        foreach row $s$ in $T_2$ do
            $Obj.par$ =pMerge($r.par$,$s.par$)
            foreach vertex $v_i$ in $Obj.par$ where $i = 1, 2, .., k+1$ do
                $Obj.locmap[v_i] = r.locmap[v_i] || s.locmap[v_i]$
            end
            foreach vertex $v \in C$ do
                $Prob\_C = Prob\_C * p(r_{(v,s.locmap[v])})$
            end
            $Obj < Obj.par : Obj.locmap >= \frac{Prob[r] \times Prob[s]}{Prob\_C}$
            Insert $Obj$ in $T$ as a row.
        end
    end
end
return Table $T$
```

3.2.3 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 $P_2$ do
2      $P_1.push\_back(s^*)$
   end
3  for ($s = P_1.begin(); s \neq P_1.end(); ++s$) do
4      for ($t = s.next(); t \neq P_1.end(); ++t$) do
5          if $s^* \cap t^* \neq \emptyset$ then
6              $P_1.push\_front(s^* \cup t^*)$
7              $P_1.delete(s^*)$
8              $P_1.delete(t^*)$
9              $s = P_1.begin()$
10             break
           end
       end
   end
11 set $P = P_1$
   return $P$
```

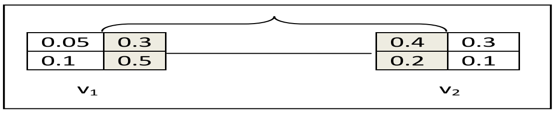


Figure 3.2: A simple tree with 2 nodes

3.3 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 3.2 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 $0.8 \times 0.6 = 0.48$.

3.4 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.

3.5 Running time

- The number of nodes in a table for k-tree $= n$.
- The number of ways n nodes can be partitioned $= 2^n$.
- Maximum number of locations in the locality set of any node $= l_{max}$.
- The number of ways each partition reappear $= l_{max}^n$.
- So the length of a table $= l_{max}^n \times 2^n$.
- In-order to merge two tables, total number of operation$= (l_{max}^n \times 2^n)^2$.
- 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 $= (l_{max}^n \times 2^n)^{k+1}$
- Total number of node we are eliminating $= n$.
- So the total cost $= n \times (l_{max}^n \times 2^n)^{k+1}$

3.6 Simulation Results

In this section we present simulation results that explore the following performance aspects of our devised algorithm:

(a) the execution time of the algorithm.
(b) influence of k on accuracy where $k = 1, 2, 3, ...$
(c) effect of choosing different k-trees.

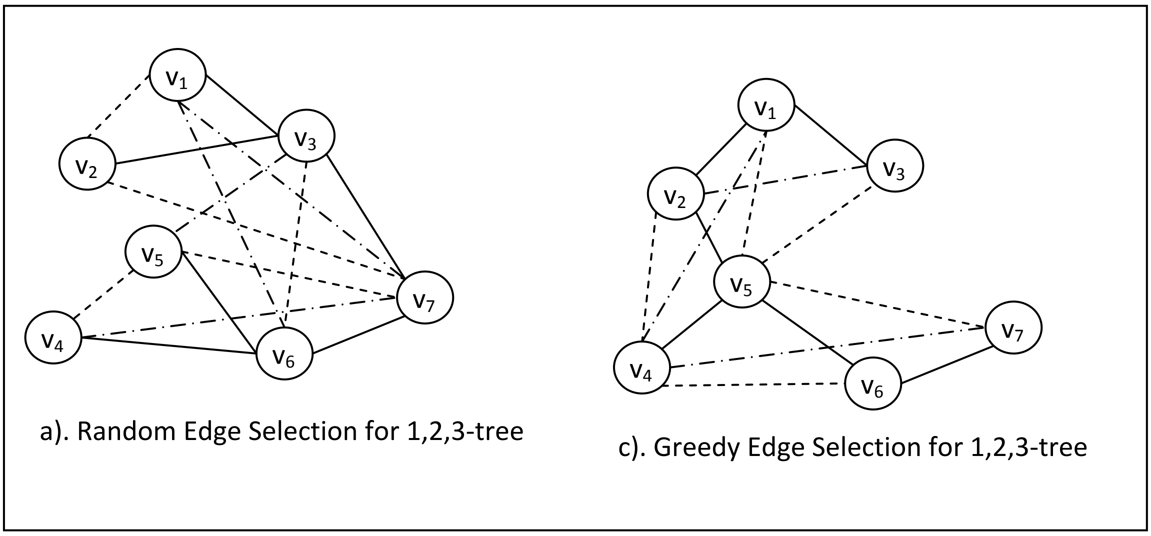
Our devised k-tree algorithm are implemented in C++ with the use of STL (standard Template Library) container classes.

Table 3.1: 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 3.2: Accuracy with respect to k

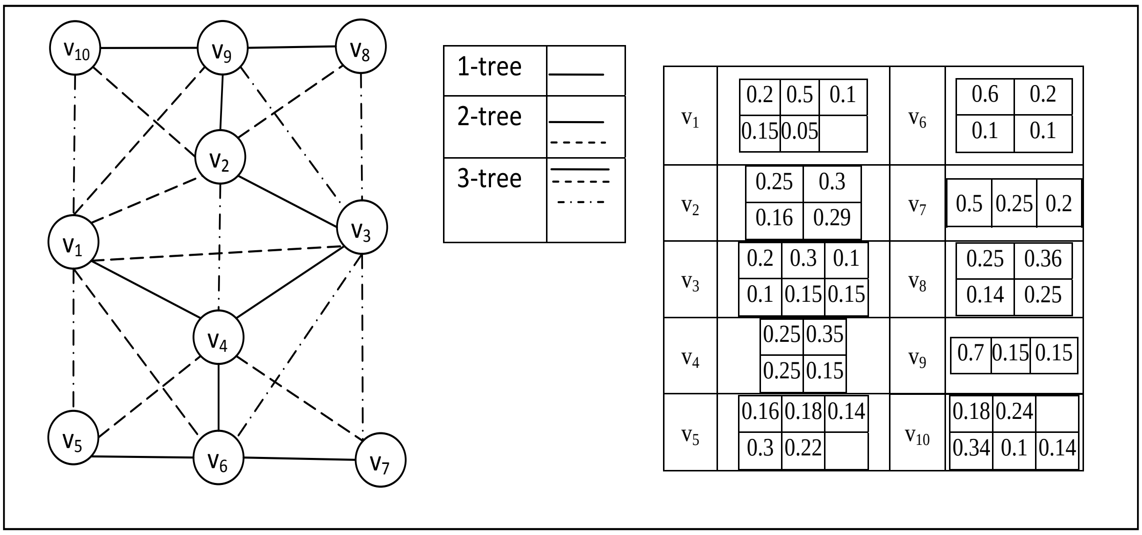
k	Network I	Network II	Network III
1	62	32.96	81.8
2	71.2	36.15	98.95
3	75	37.31	100



a). Random Edge Selection for 1,2,3-tree

c). Greedy Edge Selection for 1,2,3-tree

Figure 3.3: Greedy Vs Random Edge selection



v1	0.2 0.5 0.1 0.15 0.05	v6	0.6 0.2 0.1 0.1
v2	0.25 0.3 0.16 0.29	v7	0.5 0.25 0.2
v3	0.2 0.3 0.1 0.1 0.15 0.15	v8	0.25 0.36 0.14 0.25
v4	0.25 0.35 0.25 0.15	v9	0.7 0.15 0.15
v5	0.16 0.18 0.14 0.3 0.22	v10	0.18 0.24 0.34 0.1 0.14

Figure 3.4: Network I

3.6.1 Test Networks:

For simplicity of constructing test networks and analyzing the obtained results, we assume that all nodes have the same transmission results R_{tr}, and we set the *Reach* relation according to the Euclidean distance between involved nodes.

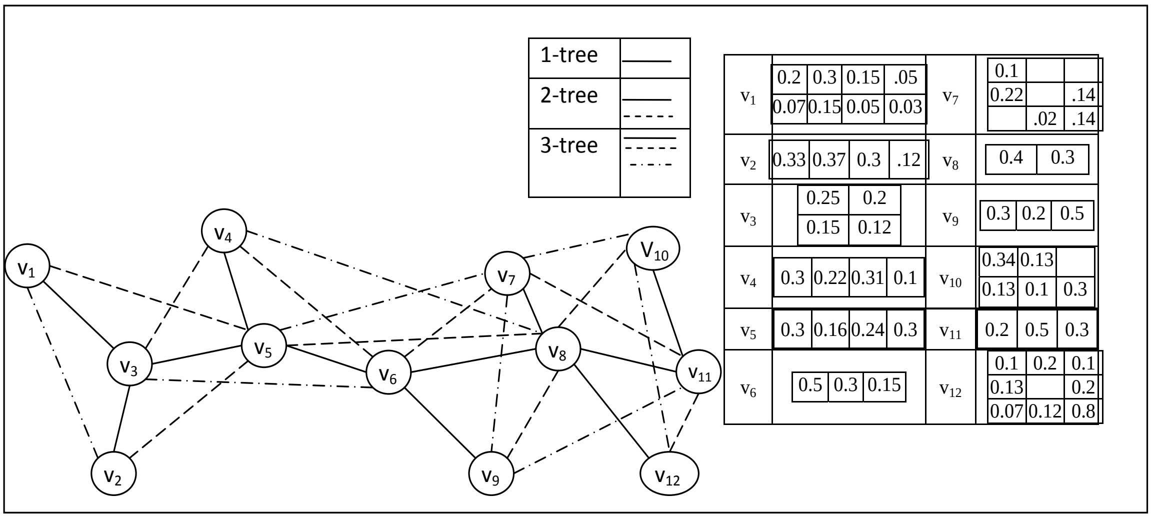


Figure 3.5: Network II

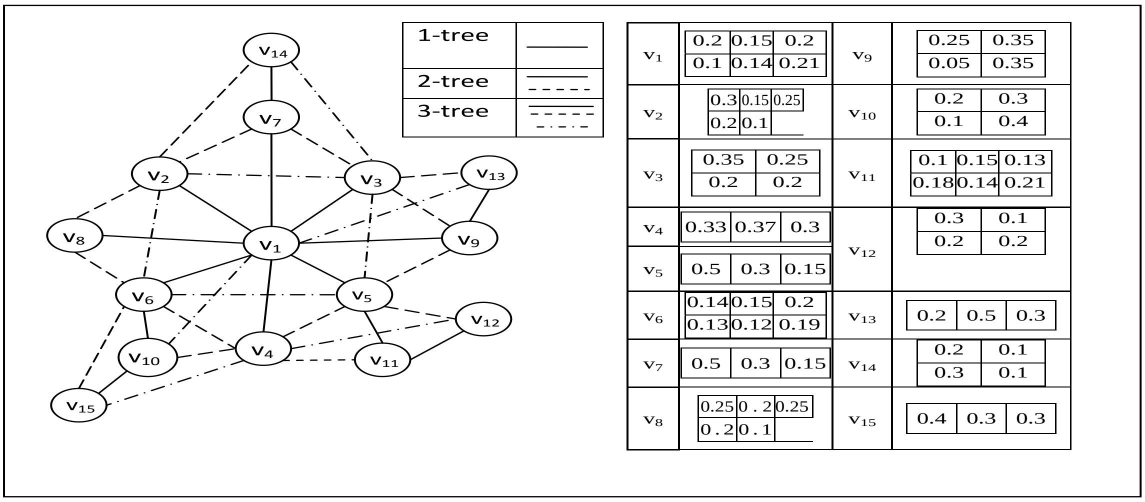


Figure 3.6: Network III

We have experimented with networks of different sizes in range $[7-18]$ nodes where each node has a locality set in range $[2-8]$ rectangles.Here, we use three networks in-order represent running time and accuracy. Network I in Fig. 3.4, consists of 10 nodes with locality set of each node range $[4-6]$. Network II in Fig. 3.5, consists of 12 nodes where each node can be located from 3 to 8 locations. Network III in Fig. 3.6, consists of 15 nodes with locality set of each node range from 2 to 6.

a) k **versus running time** table 3.1 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

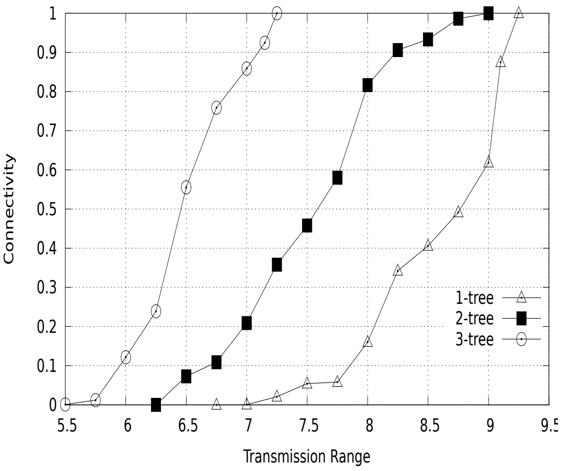


FIGURE 3.7: Connectivity versus transmission range with random edge selection.

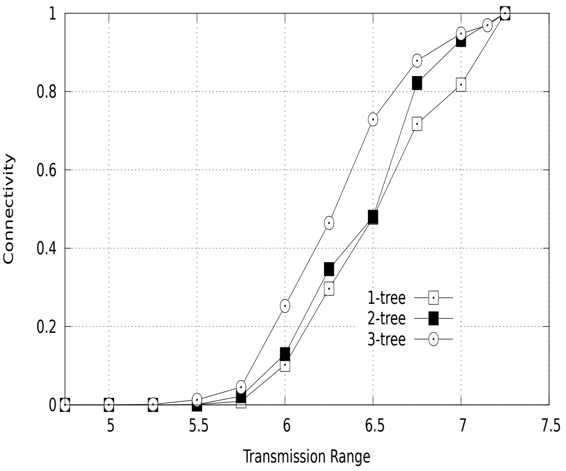


FIGURE 3.8: Connectivity versus transmission range with greedy edge selection.

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 3.7 illustrates connectivity versus transmission range when the algorithm selects randomly. We have the following observation from fig 3.7.

- 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 3.8 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 3.3 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.

3.7 Summary

In this chapter, we present the $A-CONN$ problem with possible data structures and variables require to provide an exact solution using dynamic programming approach. We present the correctness, verification cases and running time of the presented algorithm. Finally we present some simulation results on different types of network scenario.

Chapter 4

$AR - CONN$ Problem

In this chapter, we are going to present the $AR - CONN$ problem and it's exact solution. We added relay nodes to our $A - CONN$ problem inorder to formulate $AR - CONN$ problem. So our data structures changes as a result main algorithm and merge function also changes in some aspect. We describe the updates and compare the results after adding relays.

4.1 Introduction

Relay nodes plays an important role in terresterial communication as well as underwater communication.Relay nodes does not have the sensing capabilities but enhace the overall connectivity.

4.2 Problem Statement

In this section we define the $AR - CONN$ problem and we present an example scenario where we are seeking all sensor nodes connectivity with the presence of relay nodes.

Definition 4.1 (The $AR - CONN$ Problem)**.** We are given, V the set of all nodes where each node $v \in V$ is located into a set of regions, $R_v = \{r_{(v,1)}, r_{(v,2)}, ...\}$ with probability $p(r_{(v,i)})$ where $i = 1, 2, ...$ $V_{relay} \in V$ is the set of relay nodes and

$V_{sense} \in V$ is the set of sensor nodes where $V = V_{sense} \cup V_{relay}$. Sink $s \in V_{sense}$ is a special type of sensor node which is performing network wide commands and control function. 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 $(x, y) \in E$. We use relay nodes to enhance the performance of the network. Here, we are interested to find out the probability $Conn(G)$ that all sensor nodes V_{sense} are connected.

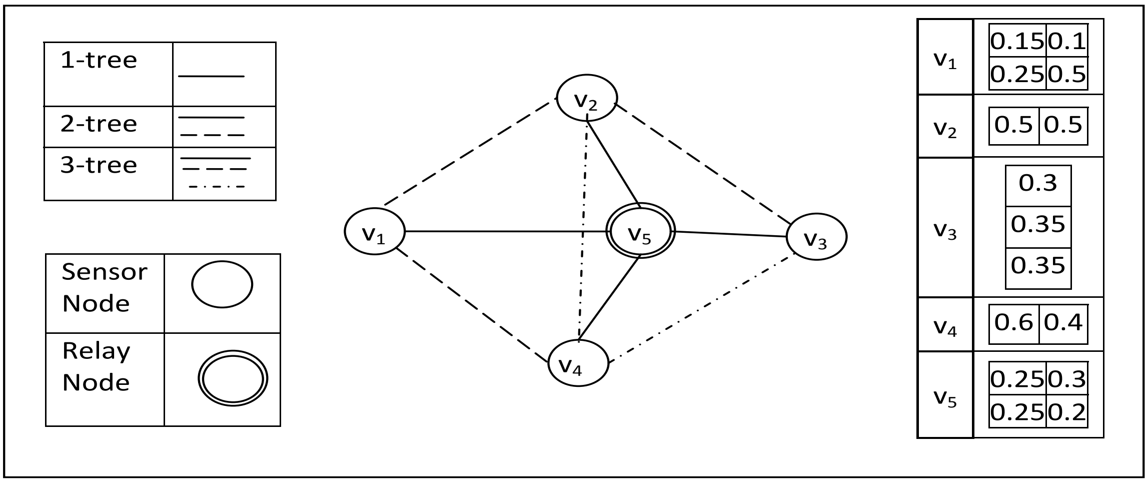


Figure 4.1: A partial 2-tree with 6 nodes.

Example 4.1. Fig 4.1 illustrates a network of 5 nodes where v_1, v_2, v_3 and v_4 are target nodes and v_5 is relay node. v_5 is enhancing the connectivity of the network without performing sensing function. v_3 is the our sink node for this network. In $AR - CONN$ problem, we are interested to find out the probability that v_1, v_2 and v_4 are connected with sink v_3.

4.3 Algorithm for $AR - CONN$ Problem

4.3.1 Key Data Structures

The algorithm (Function Main in Alg. **??**, Function merge in Alg. 8 and Function pMerge in Alg. 6)employs a dynamic programming approach. It takes as input an instance G of the $AR-CONN$ problem and a k-tree, $T(V, E)$ and PES, a perfect elimination sequence. The algorithm computes the exact solution $Conn(G)$ of the given instance.

Section 3.2 present 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 i). partitions and ii). regional set in section 3.2. In this section we are going to explain iii). target node attach.

- 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. For example in the key $\{v_1, v_2\}^{(1,1)}_{(1)}\{v_5\}^{(2)}_{(0)}$, there are two partitions including $\{v_1, v_2\}$ and $\{v_5\}$. 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_5 is a relay node.

$T_1(v_1, v_2, v_5)$

.	.
$\{v_1, v_2\}^{(1,1)}_{(1)}\{v_5\}^{(2)}_{(0)}$	0.002
.	.
.	.
.	.

×

$T_2(v_1, v_4, v_5)$

.	.
$\{v_1\}^{(1)}_{(1)}\{v_4, v_5\}^{(2,1)}_{(1)}$	0.008
.	.
.	.
.	.

⇒

$Temp(v_1, v_2, v_4, v_5)$

.	.
$\{v_1, v_2\}^{(1,1)}_{(1)}\{v_4, v_5\}^{\{2,1\}}_{(1)}$	0.0008
.	.
.	.
.	.

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

$Temp(v_1, v_2, v_4, v_5)$

.	.
$\{v_1, v_2\}^{(1,1)}_{(1)}\{v_4, v_5\}^{(2,1)}_{(1)}$	0.0008
.	.
.	.
.	.

⇒

$Temp(v_2, v_4, v_5)$

.	.
$\{v_2\}^{(1)}_{(1)}\{v_4, v_5\}^{(2,1)}_{(1)}$	0.0008
.	.
.	.
.	.

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

4.3.2 Function Main

Main steps of function main is described in section 3.2.1. After adding relay nodes main function is changed in some aspect. In this section we are going to describe only those changes after adding relay nodes.

- In Step 8, we are checking the node v_i is in a partition by itself and node v_i is relay node. If node v_i is a partition by itself and the node v_i is a relay node but there are some other node sensor was connected with v_i previously. In that case the *tAttach* for v_i will be 1 and we are unable to delete node v_i because the information about previously deleted sensor node is going to be eliminated.

- 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, $T(V, E)$, PES)

```
Input: An instance of $AR - CONN$ problem where $G$ has a $k$-tree topology
       $T(V, E)$
Output: $Conn(G)$
Notation: $Temp$ is a temporary table. Each row of $Temp$ is a key-value mapping.
Initialize every clique by a table.
for $i = 1, 2, ..., |V| - k$, according to the order of $PES$ 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 $v_i$
    $T_{(v_i,base)} = merge(Temp, T_{(v_i,base)})$
    if $v_i$ is in a partition $P$ of table $T_{(v_i,base)}$ by itself and $tAttach[P]! = 0$ then
        delete the row containing that partition.
    end
    else
        remove node $v_i$ from every row of $T_{(v_i,base)}$
    end
end
return Pr=$\sum$(All probability for single partition in the remaining table)
```

4.3.3 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
set $C$ = the set of common vertices between $T_1$ and $T_2$ , set $Prob\_C = 1$
if $C \neq \emptyset$ then
    foreach row $r$ in $T_1$ do
        foreach row $s$ in $T_2$ do
            $Obj.par$ =pMerge($r.par$,$s.par$)
            foreach vertex $v_i$ in $Obj.par$ where $i = 1, 2, .., k + 1$ do
                $Obj.locmap[v_i] = r.locmap[v_i]||s.locmap[v_i]$
            end
            foreach node $v$ in every partition $P$ in $Obj.par$ do
                foreach node $u$ in every partition $Q$ in $s.par$ do
                    if $v == u$ then
                        $Obj.tAttach[P] = max(Obj.tAttach[P], s.tAttach[Q])$
                    end
                end
                foreach node $w$ in every partition $T$ in $r.par$ do
                    if $v == w$ then
                        $Obj.tAttach[P] = max(Obj.tAttach[P], r.tAttach[T])$
                    end
                end
            end
            foreach vertex $v \in C$ do
                $Prob\_C = Prob\_C * p(r_{(v_i,s.loc[v])})$
            end
            $Obj < Obj.par : Obj.loc >= \frac{Prob[r] \times Prob[s]}{Prob\_C}$
            Insert $Obj$ in $T$ as a row.
        end
    end
end
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.

- 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.

4.4 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 4.1 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.4.2

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 4.1: Accuracy with respect to k

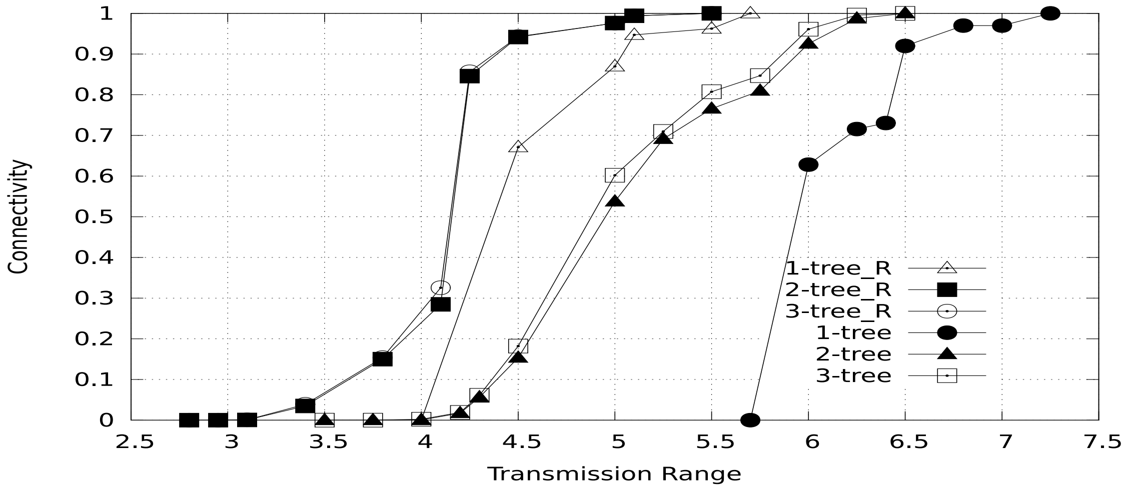


FIGURE 4.2: Connectivity Vs Transmission range for network with relays and network without relays

- 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.4.2 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.

4.5 Summary

Chapter 5

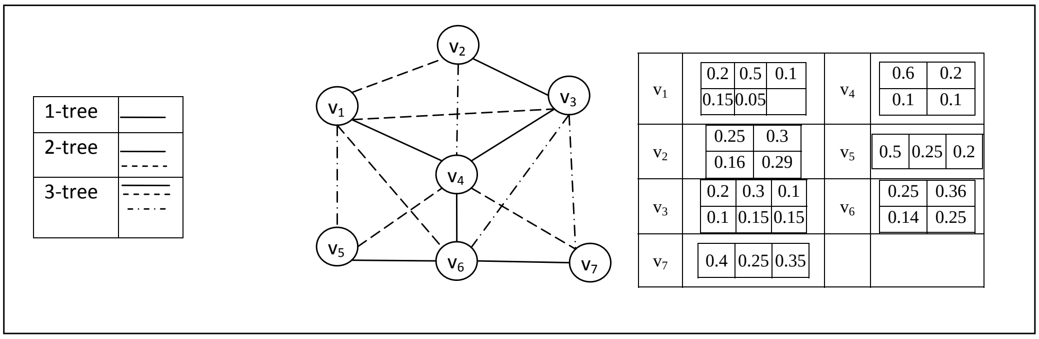
$S - CONN$ Problem

We start this chapter by providing reason why we are interested in a set of node connectivity $S - CONN$ problem. Then we provide an overview of the data structures and variables we used in our exact algorithm. After that we present our exact algorithm which use dynamic programming framework to solve $S - CONN$ problem. Finally we present some simulation results and compare with $A - CONN$ problem.

5.1 Introduction

The underwater environment is very dynamic and it is difficult to measure all nodes connectivity. Also Nodes can be fail for several reasons like water current, water whirls, battery failure etc. In this chaqpter we are interested to measure the connectivity between a set of nodes. Here we present an exact algorithm which is able to provide the connectivty between a predefined number of sensor nodes denoded by n_{req}. By using this $S - CONN$ problem it is possible to verify the all node connectivity $A - CONN$ problem. If $n_{req} = |V|$, all node then both problem are same. In this chapter we compare the results and verify that the algorithms are correct.

Example 5.1. Fig. 5.1 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



v1	0.2 0.5 0.1 0.15 0.05	v4	0.6 0.2 0.1 0.1
v2	0.25 0.3 0.16 0.29	v5	0.5 0.25 0.2
v3	0.2 0.3 0.1 0.1 0.15 0.15	v6	0.25 0.36 0.14 0.25
v7	0.4 0.25 0.35		

Figure 5.1: A partial 3-tree with 7 nodes

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. ■

5.2 Algorithms for $S - CONN$

5.2.1 Data Structures

We know from section 3.2 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 3.2.3. We modified the $merge()$ function which is used for merging two table described in section 3.2.2 and $Main()$ function which is described in section 3.2.1. 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.

5.2.2 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.

5.2.3 Function Merge

In this section steps 6 and 7 are added with the merge function which is mainly described in section 3.2.2

- $Step6$: 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.
- $Step7$: 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 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)
```

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
set $C$ = the set of common vertices between $T_1$ and $T_2$ , set $Prob\_C = 1$
if $C \neq \emptyset$ then
    foreach row $r$ in $T_1$ do
        foreach row $s$ in $T_2$ do
            $Obj.par$ =pMerge($r.par$,$s.par$)
            foreach partition $Par_i$ in $Obj.par$ where $i = 1, 2, 3, ...$ do
                $Obj.Csize[Par_i] = r.Csize[Par1] + s.Csize[Par2]$
                //If there is partition $Par1$ in $r$ and $Par2$ in $s$ where $Par1 \subseteq Par_i$
                and $Par2 \subseteq Par_i$
            end
            foreach vertex $v_i$ in $Obj.par$ where $i = 1, 2, .., k + 1$ do
                $Obj.loc[v_i] = r.loc[v_i]||s.loc[v_i]$
            end
            foreach vertex $v \in C$ do
                $Prob\_C = Prob\_C * s.loc[v]$
            end
            $Obj < Obj.par : Obj.loc >= \frac{Prob[r] \times Prob[s]}{Prob\_C}$
            Insert $Obj$ in $T$ as a row.
        end
    end
end
return Table $T$
```

5.3 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

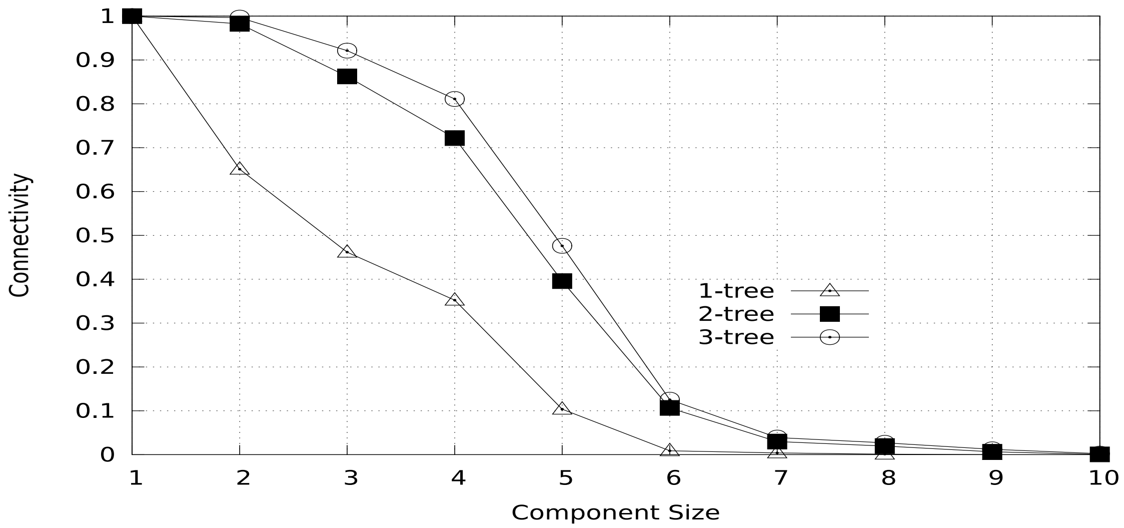


FIGURE 5.2: Connectivity versus component size for Newtork I

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.

5.4 Summary

Chapter 6

$SR - CONN$ Problem

6.1 Introduction

6.2 Problem definition

6.3 Algorithms for $S - CONN$ with Relays

6.3.1 Data Structure

6.3.2 Function Main

```
Algorithm 11: 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
    $Temp = T_{(v_i,1)}$
```

6.3.3 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

```
set $C$ = the set of common vertices between $T_1$ and $T_2$ , set $Prob\_C = 1$
if $C \neq \emptyset$ then
    foreach $row\ r\ in\ T_1$ do
        foreach $row\ s\ in\ T_2$ do
            $Obj.par$ =pMerge($r.par$,$s.par$)
            foreach $partition\ Par_i\ in\ Obj.par\ where\ i = 1, 2, 3, ...$ do
                $Obj.Csize[Par_i] = r.Csize[Par1] + s.Csize[Par2]$
                //If there is partition $Par1$ in $r$ and $Par2$ in $s$ where $Par1 \subseteq Par_i$
                and $Par2 \subseteq Par_i$
            end
            foreach $vertex\ v_i\ in\ Obj.par\ where\ i = 1, 2, .., k + 1$ do
                $Obj.locmap[v_i] = r.locmap[v_i]||s.locmap[v_i]$
            end
            foreach $node\ v\ in\ every\ partition\ P\ in\ Obj.par$ do
                foreach $node\ u\ in\ every\ partition\ Q\ in\ s.par$ do
                    if $v == u$ then
                        $Obj.tAttach[P] = max(Obj.tAttach[P], s.tAttach[Q])$
                    end
                end
                foreach $node\ w\ in\ every\ partition\ T\ in\ r.par$ do
                    if $v == w$ then
                        $Obj.tAttach[P] = max(Obj.tAttach[P], r.tAttach[T])$
                    end
                end
            end
            foreach $vertex\ v \in C$ do
                $Prob\_C = Prob\_C * s.loc[v]$
            end
            $Obj < Obj.par : Obj.loc >= \frac{Prob[r] \times Prob[s]}{Prob\_C}$
            Insert $Obj$ in $T$ as a row.
        end
    end
end
return $Table\ T$
```

6.4 Simulation Results

6.5 Summary

Bibliography

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

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

[3] 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.

[4] 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.

[5] 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.

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

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

[8] 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.

[9] 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.

[10] 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.

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

[12] 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.

[13] 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.

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

[15] 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.

[16] 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.

[17] 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.

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

[19] Sandipa Singh, Sarah E Webster, Lee Freitag, Louis L Whitcomb, Keenan Ball, John Bailey, and Chris Taylor. Acoustic communication performance of the whoi micro-modem 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.

[20] 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.

[21] 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.

[22] 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.

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

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

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

[31] 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.

[32] 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.

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

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

[35] 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.

[36] Amy S Bower. A simple kinematic mechanism for mixing fluid parcels across a meandering jet. *Journal of Physical Oceanography*, 21(1):173–180, 1991.

[37] 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.

[38] 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.

[39] 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.

[40] 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. doi: 10.1109/ICPPW.2001.951990.

[41] Charles J. Colbourn. *The Combinatorics of Network Reliability*. Oxford University Press, Inc., New York, NY, USA, 1987. ISBN 0195049209.

[42] Lowell W Beineke and Raymond E Pippert. The number of labeled ¡i¿ k¡/i¿-dimensional trees. *Journal of Combinatorial Theory*, 6(2):200–205, 1969.

[43] Lowell W Beineke and Raymond E Pippert. The enumeration of labelled 2-trees. *Notices Amer. Math. Soc*, 15:384, 1968.

[44] Stefan Arnborg, Andrzej Proskurowski, and Derek G Corneil. Forbidden minors characterization of partial 3-trees. *Discrete Mathematics*, 80(1):1–19, 1990.

[45] Stefan Arnborg and Andrzej Proskurowski. Linear time algorithms for np-hard problems restricted to partial¡ i¿ k¡/i¿-trees. *Discrete Applied Mathematics*, 23(1):11–24, 1989.

[46] Stefan Arnborg, Derek G Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in ak-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.

[47] 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.