Probabilistic Localization of Underwater Sensor Networks

by

Salwa Abougamila

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

In recent years, Underwater Sensor Networks (UWSNs) have attracted attention for their potential use in many applications. To name a few, UWSNs have been considered in studying marine life, oceanographic data collection, monitoring underwater oil pipelines, and a variety of military and homeland security applications.

UWSN deployments can be either static, semi-mobile, or mobile. For such networks, localization of nodes and observed events arise as a fundamental task where the obtained location information can be used in data tagging, routing, and node tracking. Analyzing the ability of an UWSN to perform a localization task is most challenging for networks with uncontrollable mobile nodes.

In this thesis, we approach the above class of problems by adopting a probabilistic graph model to describe node location uncertainty in semi-mobile and mobile deployments. Using the above model, we formalize a probabilistic node localization performance measure, and a corresponding problem to compute the measure.

Using the theory of partial k-trees, we develop an exact algorithm that runs in polynomial time, for any fixed k. We next consider the structure of network configurations that contribute to solving the formalized problem. We call such structures pathsets. We then devise an iterative algorithm that aims to generating a most probable pathset in each iteration. In addition, we present numerical results that illustrate the quality of the obtained solutions, and the use of the devised algorithms in UWSNs design.

Acknowledgements

First of all, I would like to thank my supervisor, Ehab S. Elmallah, for his guidance throughout my Master degree. I am also very grateful for his sharing of valuable insights that have led to the formulation of the main direction in this thesis, and his valuable in-depth discussions about how to achieve progress in this direction.

I would also like to give special thanks to members of my thesis examination committee: professors Lorna Stewart and Janelle Harms for reviewing my work and providing valuable insights and suggestions. During my work on the thesis, I enjoyed being a member of the Networks research group. I benfited from discussions with M. Shazly, and Md Asadul Islam. Knowledge acquired during my B.Sc. work has been useful in preparing me for the M.Sc. degree. For this, I would like to thank professors M. Abdelkawy, and H. Shehata.

Last but not the least, I would like to thank my loving and supportive husband Mohammed Elmorsy, who encouraged me to follow my dreams, and for his support and patience throughout my graduate studies. During the stressful time of completeing the M.Sc. degree, our wonderful son Yahia has been a constant source of joy and happiness. I am especially grateful to my parents, who supported me emotionally and financially.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Underwater Sensor Networks (UWSNs) have attracted significant attention in recent years as useful systems that serve many applications. In many such networks, nodes can move freely with water currents, and during any interval of time, few nodes can rise to surface to use their GPS devices to determine their geographic location. Thus, other nodes should rely on collaborative work to be able to localize themselves. In this chapter, we consider such networks. We formalize a problem on quantifying the likelihood that a given node in a mobile UWSN can localize itself.

1.1 Introduction

An Underwater Sensor Network (UWSN) is composed of communicating nodes that can be used to perform various actions in underwater environments. The domains of applications of such networks are diverse, and include the following examples (see, e.g., [2] and [25]):

- **Scientific applications:** e.g., observing geological processes
- **Military applications:** e.g., conducting surveillance missions
- **Industrial applications:** e.g., determining routes for underwater cables

The design of UWSNs, however, faces a number of challenges such as

- **Challenges of the underwater communication channel:** several studies have reported that the underwater environment poses problems for radio frequency communication, optical communication, and acoustic communication (see, e.g., [2] and [25]). Currently, acoustic communications offer the most practical solution at the expense of providing low bandwidth and long communication delays.

- **Challenges due to node mobility:** UWSNs deployments are classified as static, semi-mobile, or mobile [20]. Static networks have nodes attached to underwater ground or anchored buoys. Semi-mobile networks may have collection of nodes attached to buoy with limited mobility. Mobile networks may be composed of drifters with no self mobility capability. This type of UWSNs is very useful. However, nodes in such networks are subject to large scale movements, and hence, present a challenge in analyzing the network.

In this thesis, we consider mobile networks where nodes can travel freely. Our interest is in developing methodologies that allow a designer to analyze the likelihood that nodes in a mobile network can localize themselves. To present more details, we review in the next two sections information about modelling nodes that move freely with water currents.

1.2 Node Mobility Model

The work of [10] has adopted a kinematic mobility model of UWSNs called the *meandering current mobility model*. The model is useful for large coastal environments that span several kilometers. In [10], the authors have used a simulation approach that uses the adopted kinematic model to analyze several network connectivity, coverage, and localization measures of UWSNs. The kinematic model is a differential equation whose solution gives a trajectory of a node that moves with water currents. The equation is given below.

$$\psi(x,y,t) = -tanh[\frac{y - B(t)sin(k(x-ct))}{\sqrt{1+k^2B^2(t)cos^2(k(x-ct))}}] + cy \qquad (1.1)$$

where $B(t) = A + \epsilon \cos(\omega t)$ and the x and y velocities are given by

$$\dot{x} = -\frac{\partial \psi}{\partial y}; \dot{y} = \frac{\partial \psi}{\partial x} \tag{1.2}$$

In [10], the authors explored the trajectories generated using the following settings: $A = 1.2$, $c = 0.12$, $k = 2\pi/7.5$, $\omega = 0.4$ and $\epsilon = 0.3$. The trajectories generated by the above equations are very sensitive to the initial deployment point at time $t = 0$, and typically include many vortexes. Our work in this thesis does not use such kinematic model directly. Rather, it uses the concept of probabilistic graphs introduced next. Constructing a probabilistic graph, however, can use the above kinematic model as a means of computing the required parameters.

1.3 Probabilistic Graph Model

Our work in the thesis abstracts away from the use of detailed knowledge of node movement trajectories, as given by the above kinematic model. Rather, we adopt a simple discrete probabilistic model for mobility. Namely, after some time interval T from network deployment time, we model the UWSN using the concept of *probabilistic* graph $G = (V, E_G, \mathrm{Loc}, p)$ where

- V is the set of nodes of the UWSN
- For each node x there is a finite set of regions that form the *locality* set of node x, denoted $\mathrm{Loc}(x) = \{x(1), x(2), \cdots\}$, where each $x(i)$ is a well defined region that node x can visit at the end of the time interval T of interest.
- If node x in region $x(i) \in \mathrm{Loc}(x)$ can communicate with node y in region $y(j) \in \mathrm{Loc}(y)$ then we set the link indicator $E_G(x(i), y(j)) = 1$. Else, we set $E_G(x(i), y(j)) = 0$.
- The probability that node x is in region $x(i)$ (at the end of time T of interest) is $p_x(i)$.

The information required to construct such a probabilistic graph can be obtained by generating many trajectories using the kinematic equation presented in the previous section, and analyzing the location of the nodes after T units of time have elapsed. Different construction methods can lead to different probabilistic graphs on different sets of regions. We leave the problem of constructing an accurate probabilistic graph that models a given UWSN (at some time T) as a topic of future research, and assume that such a graph G is given as input.

1.4 Problem Formulation

In this section, we formulate the main problem dealt with in the thesis. An *anchor* node is a node that can determine its location (e.g., by using a GPS). A non-anchor node relies on contacting at least 3 of its neighbours who know their locations to localize itself. The argument is transitive, that is, each of these neighbours can either be an anchor node, or an ordinary node that has succeeded in localizing itself by contacting at least 3 of its neighbours that have localized themselves.

Definition (the $\mathrm{P\text{-}LOC}$ problem). Given a probabilistic graph $G = (V, E_G, \mathrm{Loc}, p)$, a subset $V_{anchor} \subseteq V$ of anchor nodes, and a target node t, find the probability $\mathrm{PLoc}(G, V_{anchor}, t)$ (or, $\mathrm{PLoc}(G, t)$, for short) that node t can be localized. ■

The formulation of the $\mathrm{P\text{-}LOC}$ problem shares some aspects with the class of network reliability problems, discussed in [13]. For network reliability problems, a node (or link) can either be operating or failed with some known probability. In our problem, a node can be in any one of a possible set of regions (its locality set) with some known probability. The computation of the measure $\mathrm{PLoc}(G, t)$ can proceed as follows. First, we recall that each node $x \in V$ can be in any given region $x(i) \in \mathrm{Loc}(x)$ with probability $p_x(i)$. If such an event occurs, we say that node x is in state $x(i)$. We use the notation $(x, x(i))$ (or, (x, i) for short) to refer to node x being in state $x(i)$. When each node x in the probabilistic graph G is in one of its possible $|\mathrm{Loc}(x)|$ states, we obtain a *network state* T of the probabilistic graph. T is written as $T = \{(x, i) : x \in V$ and $x(i) \in \mathrm{Loc}(x)\}$, or $T = \{(x(i) : x \in V$ and $x(i) \in \mathrm{Loc}(x)\}$ for short.

The probability that state T arises is $\Pr(T) = \prod_{(x,i) \in T} p_x(i)$. A network state T is *operating* if the target node t can be localized in state T. Else (if t can not be localized in T), then state T is *failed*. We further denote by $OP(G, V_{anchor}, t)$ (or,

$OP(G,t)$ for short) the set of all operating network states of G. Since $OP(G,t)$ is a set, any pair of network states contained therein differ in the state of at least one node. So, we can write:

$$\mathrm{PLoc}(G, V_{anchor}, t) = \sum_{T \in OP(G,t)} \Pr(T) \tag{1.3}$$

We also need the following definition. The *underlying* graph of a probabilistic graph is defined as follows:

- V (the set of nodes of the UWSN) is the set of nodes of the underlying graph.
- An edge (x,y) exists in the underlying graph iff $E_G(x(i), y(j)) = 1$ for some indices i and j in the locality sets of nodes x and y, respectively.

Example. Figure 1.1 illustrates a probabilistic graph on 6 nodes. Each node has a locality set of two regions (e.g., $\{t(0), t(1)\}$). Numbers inside rectangles are probabilities in the model. Coloured blue rectangles correspond to anchor nodes. Node t is a target node to be localized. We then note that state $T = \{t(0), v_0(1), v_1(0), v_2(1), v_3(0), v_4(1)\}$ is an operating state where the target t can localize itself. ■

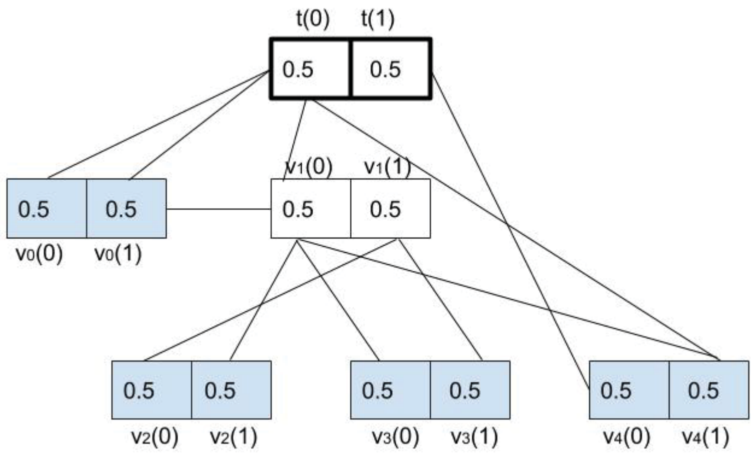

Figure 1.1: Probabilistic graph of an UWSN

1.5 Thesis Organization and Contribution

The main contributions of the thesis are in chapters 3 and 4. In Chapter 3, we present an exact solution to the P-LOC problem when the underlying graph of the probabilistic graph is a partial k-tree, $k \geq 1$. The algorithm runs in polynomial time, for any fixed k. In Chapter 4, we present an iterative algorithm that can compute exact solutions if allowed sufficient number of iterations. Otherwise, the algorithm provides a lower bound (LB) on the solution. Finally, Chapter 5 concludes with some remarks and possible future research directions.

1.6 Literature Review

Currently, there is extensive research on localization of UWSNs (see, e.g., the surveys in [11, 16, 17]). Results on the following specific directions appear in the literature.

- Minimizing localization time by packet scheduling [26, 27]
- Combining localization with time synchronization [22]
- Localization in 3-dimensional spaces [12]
- Improving localization accuracy [7, 8, 29]
- Using prediction in localization [14, 30, 31]
- GPS free localization [24]
- Localization in 2-tier hybrid sensor networks [23]

In the above research work, node mobility is considered a challenging aspect that has been dealt with directly in a relatively few papers. Approaches for handling mobility in such papers can be roughly classified as being suitable for either small scale mobility, or large scale mobility. In a small scale mobility, node movement is analyzed either over a short period of time, or when a node is constrained to travel a relatively short distance. An example of a short period of time is a time period that allows a node to receive multiple consecutive packets. An example of a constrained node movement is when the node is anchored to the underwater ground using a rope. On the other hand, in a large scale mobility, nodes can travel considerable distances over longer periods of time. Examples of work done on localization of nodes subject to large scale mobility include the work of [31] and [22], as highlighted below.

- In [31], the authors consider *seashore environments* where nodes are located near the seashore. The authors observe that some existing results in hydrodynamics demonstrate that the moving speed of such nodes changes continuously and show certain semiperiodic properties. In addition, spatial correlations between node movements exist in such environments. That is, movement of one node is closely related to its nearby nodes.

  Using the above assumptions, the localization scheme devised in [31] adopts a linear prediction method, explained in [28], to predict node mobility. The prediction method is applied over adjacent windows of fixed-length time intervals, called prediction windows. In this work, mobility prediction is used to reduce the number of transmitted messages when the scheme has confidence in the obtained values.

- The work of [22] also deals with large scale mobility by adopting yet another mobility prediction scheme. The scheme utilizes an adaptive estimation approach, called *interactive multiple mode* (IMM), explained in [4]. As mentioned in [22], the IMM filter runs a bank of filters in parallel where each filter incorporates all possible moving patterns of a node. A Markov chain transition matrix is introduced to characterize the transition probability of a node's moving pattern from one mode to another. Using the above method, the authors have considered two moving patterns of a node with added Gaussian noise: a *uniform* motion along a straight line with a constant velocity, and a *maneuver* motion where some nodes make coordinated turns with a constant turn rate using constant speeds. In addition, they obtain simulation results using the random walk pattern, and the meandering current mobility model. In this work, mobility prediction is used to improve localization accuracy.

We note that the two latter reviewed publications utilize methods to predict the location of a mobile node to either reduce the number of messages transmitted by a protocol (and, hence, reducing a node's energy consumption), or improve the localization error. In such approaches, nodes take a corrective action if the prediction method is conceived to produce values different from some measured values. We note, however, that such approaches are not suitable for tackling the P-LOC problem investigated in the thesis. On the other hand, our work in the thesis shows the suitability of using the probabilistic graph model.

1.7 Concluding Remarks

In this chapter we have introduced the concept of probabilistic graphs as a way to model location uncertainty in UWSNs with free nodes moving according to water currents. We have also introduced the probabilistic location problem (P-LOC) that is investigated in the rest of the thesis. In the next chapter, we give background on the classes of partial k-trees used in our first algorithm.

Chapter 2

Background on k-Trees and Partial k-Trees

In this chapter, we give some preliminary information on the classes of k-trees and partial k-trees. The information is required to present our algorithm in the next chapter.

2.1 Graph Notation

We start by reviewing some standard graph theoretic notation. An undirected graph $G = (V, E)$ has a set V of nodes (or vertices), and a set E of edges (or links). The following notation will also be encountered in our presentation.

- $deg_G(v)$ denotes the degree of node v in graph G.
- $N_G(v)$ denotes the set of neighbouring nodes of v in graph G.
- An *induced subgraph* $G' \subseteq G$ on a set $V' \subseteq V$ of nodes contains all edges of G where the two end nodes of each edge lie in V'.
- A clique is a complete graph. A k-clique is a clique on k nodes.
- A separator in a graph G is a subset of nodes whose removal splits G into two, or more, components.

2.2 The k-Trees Family

The Classes of k-Trees. The notion of k-trees, for any fixed $k \geq 1$, is described in the early work of [5, 6]. We use the following definition.

Definition. For a given integer $k \geq 1$, the class of k-trees is defined as follows. A k-clique is a k-tree. Furthermore, if G_n is a k-tree on n nodes then so is the graph G_{n+1} obtained by adding a new node, and making it adjacent to every node in a k-clique of G_n. ■

Thus, conventional trees are 1-trees. A k-leaf of a k-tree G on $k+1$, or more, nodes is a node whose neighbours induce a k-clique. Note that by repeatedly deleting k-leaves from a k-tree G_n, on $n \geq k$ nodes, one reduces G_n to a k-clique. Such a sequence of nodes is called a *k-leaf elimination sequence*.

Example. Figure 2.1 sketches part of a 3-tree G. G has a 3-leaf v_b. ■

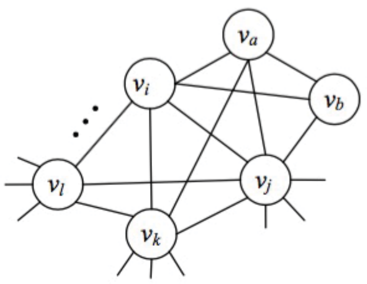

Figure 2.1: A fragment of a 3-tree

The following lists some basic properties of k-trees. References [3, 18] mention other properties.

Lemma 1 *1. Every k-tree that is not a clique has at least two non-adjacent k-leaves.*

2. Given a k-tree G, and a k-clique subgraph $H \subseteq G$, there exists a k-leaf elimination sequence that reduces G to H.

3. Given two non-adjacent nodes u and v of a k-tree G, a subgraph induced on any minimal (u, v)-separator is a k-clique. ■

The Classes of Partial k-Trees. For a given k, a partial k-tree is a subgraph of a k-tree. A *leaf* of a partial k-tree G is a node x that is a k-leaf in some embedding of G in a k-tree $\tilde{G}$. A *perfect elimination* of x from G is the elimination of x and its incident edges and the addition of the necessary edges to complete $N_G(x)$ to a clique. A *k-perfect elimination sequence* (k-PES) of a graph G is an ordering $(v_1, v_2, \cdots, v_n)$, or an ordering of $(v_1, v_2, \cdots, v_{n-k})$ for short, of V_G such that

1. $deg_G(v_1) \leq k$, and
2. for $i = 2, 3, \cdots$, the degree of v_i in the graph obtained by perfectly eliminating $v_1, \cdots, v_{i-1}$ is at most k.

Example. For the graph G in figure 2.2, $(v_a, v_b, v_i, v_j, v_k, v_l)$ is a 3-PES. ■

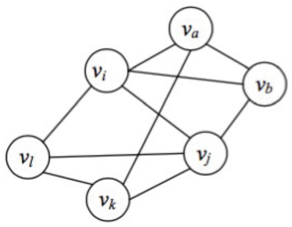

Figure 2.2: A Partial 3-tree

For any fixed k, the class of partial k-trees is related to another well known class, called graphs with tree-width $\leq k$ (see, e.g., [9]).

Relation to Other Graph Classes. Several classes of graphs are known to be partial k-trees for fixed k [9]. For example

- series-parallel graphs, and outerplanar graphs are partial 2-trees.
- Halin graphs and Y-Δ graphs are partial 3-trees.
- An $n \times n$ grid, $n \geq 2$ is a partial n-tree.

2.3 Concluding Remarks

In this chapter we have presented some preliminary information on the classes of k-trees and partial k-trees, and their relations to other classes of graphs. In the next chapter, we present our algorithm to solve the P-LOC problem on partial k-trees, for any given k.

Chapter 3

A Partial k-tree Algorithm

In this chapter, we consider instances of the $\mathrm{P\text{-}LOC}$ problem where the underlying graph G is a partial k-tree, $k \geq 1$, that is a subgraph of a k-tree, denoted $\tilde{G}$. In addition, we assume that G is given as input together with a k-perfect elimination sequence (PES) that reduces $\tilde{G}$ to a k-clique containing the target node t. For such problem instances, we present a dynamic programming algorithm to solve the problem exactly. The exact algorithm runs in polynomial time, for any fixed k.

The chapter starts by reviewing relevant system model information presented in previous chapters. We then present the main ingredients of a dynamic programming algorithm that solves the $\mathrm{P\text{-}LOC}$ problem exactly. The presented algorithm follows a framework in the literature that has enabled the solution of other optimization and enumeration problems on graphs. The algorithm is implemented in the C++ language with the use of the Standard Template Library (STL). The source program is about 1500 lines. To verify the correctness of the implementation, we present some ideas that we have used.

3.1 System Model Summary

We recall that an instance of the $\mathrm{P\text{-}LOC}$ problem is specified by the parameters (G, V_{anchor}, t) where

- G is a probabilistic graph, $G = (V, E_G, \mathrm{Loc}, p)$, on the following parameters

  - V is a set of UWSN nodes
  - $E_G(x(i), y(j)) = 1$ iff the two nodes x and y are assumed to reach each other when located in the regions $x(i)$ and $y(j)$, respectively
  - $\text{Loc}(x)$ is the locality set of node x
  - $p_x(i)$ is the probability that node x is located in region $x(i)$

- V_{anchor} is a subset of anchor nodes of V
- $t \in V$ is a non-anchor node to be cooperatively localized using other nodes.

We use $n = |V|$ to denote the number of nodes in G. To simplify notation (when no confusion can arise), we also use the symbol G to denote the underlying graph of the probabilistic graph G. The underlying graph is defined as follows:

- V (the set of nodes of the UWSN) is the set of nodes of the underlying graph
- Edge (x, y) is in the underlying graph (written as $E(x, y) = 1$) iff $E_G(x(i), y(j)) = 1$ for some indices i and j in the locality sets of nodes x and y, respectively.

Since the same symbol G is used to refer to a probabilistic graph and its underlying graph, sentences involving the relation $E_G(x(i), y(j))$ refer to G as a probabilistic graph. On the other hand, statements such as $E(x, y) = 1$, or references to induced subgraphs of G refer to G as a conventional (deterministic) graph.

In this chapter, we deal with instances (G, V_{anchor}, t) of the P-LOC problem that satisfy the following properties:

1. The underlying graph G in the given instance is a partial k tree, $k \geq 1$, on n nodes, $n \geq k$. We denote by $\tilde{G}$ a k-tree of which G is a subgraph.
2. $\tilde{G}$ has a given k-perfect elimination sequence $PES = (v_1, v_2, \cdots v_{n-k})$. The removal of nodes in the PES from $\tilde{G}$ reduces the graph to a k-clique, say H. We require that the target node t be a node in H. By Lemma 1 in chapter 2, this requirement can always be satisfied.

Note that, the existence of $\tilde{G}$ (specified by property 1) and a PES (specified by property 2) is assured since every underlying graph G (with no loops or parallel edges) is a partial k-tree for $k = n - 1$.

Using the above concepts, we now highlight the main ingredients in the algorithm. Our presentation follows closely the presentation of [21] which in turn builds on the presentations of many previous algorithms in the literature.

3.2 Overview of the Algorithm

The overall structure of the algorithm relies on the following concepts.

C1. The order of processing nodes by the dynamic programming algorithm

C2. The way a subgraph is reduced onto a separator clique of the k-tree $\tilde{G}$

C3. The specification of the dynamic program state types used to make the algorithm efficient

C4. The specification of tables and data structures used to store the states of the dynamic program

For concept C1, the algorithm processes the nodes in the order of the given $PES = (v_1, v_2, \cdots v_{n-k})$. As mentioned earlier, we require that the remaining clique of $\tilde{G}$ on nodes $\{v_{n-k+1}, \cdots, v_n\}$ (obtained after processing and eliminating nodes in the PES) to contain the target node t. To clarify the processing of a typical node v_i in the PES, we use the following notation. The notation refers to the fragment of a 3-tree illustrated in figure 3.1.

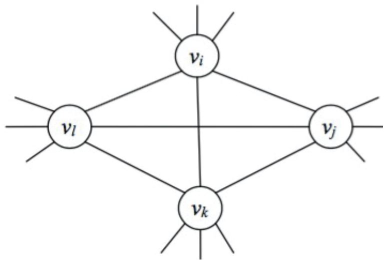

Figure 3.1: A fragment of a 3-tree

- $K_{v_i,base}$: the k-clique to which node v_i is attached in the recursive construction of the graph $\tilde{G}$ according to the reverse of the PES given. In our example of figure 3.1, we assume that $K_{v_i,base}$ is the triangle (v_j, v_k, v_l).

- $K_{v_i,1}, K_{v_i,2}, \cdots, K_{v_i,k}$: the k k-cliques involving node v_i when v_i becomes a k-leaf. Each k-clique is made of node v_i and $k-1$ other nodes in $K_{v_i,base}$. In our example of figure 3.1, one may set, e.g., $K_{v_i,1}$ to the triangle (v_i, v_j, v_k).

After processing and deleting all nodes in the prefix $(v_1, v_2, \cdots, v_{i-1})$ of the PES, node v_i becomes a k-leaf in the current reduced graph of $\tilde{G}$. Information about certain subgraphs on the deleted nodes are maintained in tables associated with the k-cliques $K_{v_i,base}$, $K_{v_i,1}$, $K_{v_i,2}$, $\cdots$, $K_{v_i,k}$. We use $T_{v_i,\alpha}$ to denote the table associated with clique $K_{v_i,\alpha}$, where $\alpha = base, 1, 2, \cdots, k$.

Example. In figure 3.2, we assume that v_b and v_a have been deleted to make v_i a 3-leaf. The information about the induced subgraph on nodes $\{v_a, v_b, v_i, v_j, v_k\}$ is summarized in table $T_{v_i,1}$ associated with the separator clique $K_{v_i,1} = (v_i, v_j, v_k)$. ■

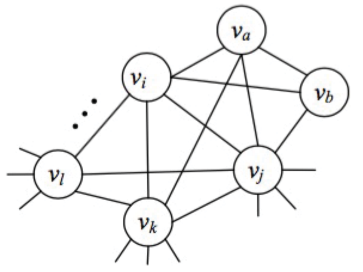

Figure 3.2: A 3-tree fragment

For concept C2, we say in our example above that the induced subgraph on nodes $\{v_a, v_b, v_i, v_j, v_k\}$ is reduced onto the clique (v_i, v_j, v_k). Prior to processing node v_i, the algorithm maintains in each table $T_{v_i,\alpha}$, where $\alpha = base, 1, 2, \cdots, k$, information about a subgraph of G, denoted $G_{v_i,\alpha}$ that has been reduced onto the clique $K_{v_i,\alpha}$. Each such table $T_{v_i,\alpha}$ is initialized to store information about the k-clique $K_{v_i,\alpha}$ itself.

For concept C3, we note that the algorithm needs to store information about useful network states of the graph $G_{v_i,\alpha}$ in table $T_{v_i,\alpha}$. As explained in Chapter 1, a network state of a probabilistic graph assigns to each node x a region in its locality set $\mathrm{Loc}(x)$. The worst case number of possible useful network states of any subgraph processed in the algorithm, say $G_{v_i,\alpha}$, grows exponentially with the number of nodes in the subgraph. The dynamic program strives to achieve efficiency by consolidating information about many network states that are considered of the same *type*. For our P-LOC problem, the needed dynamic program state types are defined next.

Definition (state types of the P-LOC **problem).** Let $G_{v_i,\alpha}$ be a subgraph that has been reduced onto the clique $K_{v_i,\alpha}$. Denote by $V_{v_i,\alpha}$ the set of nodes of $G_{v_i,\alpha}$. Let $S = \{v_a(i_a) : v_a \in V_{v_i,\alpha}, i_a \in \mathrm{Loc}(v_a)\}$ be a network state of $G_{v_i,\alpha}$. Then the type of network state S is the following set defined on the nodes $(x_1, x_2, \cdots x_k)$ of the separator clique $K_{v_i,\alpha}$:

$$type(S) = \{(x_i, loc(x_i), count(x_i), anchors(x_i)) : i = 1, 2, \cdots, k\}$$

where for each node x_i in the clique, we have

- $loc(x_i)$: the location of node x_i in the state S (that is, the region in $\mathrm{Loc}(x_i)$ assumed by node x_i in network state S)

- $count(x_i) \in \{0, 1, 2, 3\}$: For an anchor node x_i, or a node that has been localized thus far, $count(x_i) = 3$. For a non-anchor node x_i that has not been localized thus far, $count(x_i)$ is the number (at most 2) of anchor neighbours, or neighbours that have been localized in the subgraph $G_{v_i,\alpha}$ processed thus far.

- $anchors(x_i)$: For an anchor node x_i, $anchors(x_i) = \emptyset$. For a non-anchor node x_i, $anchors(x_i)$ is the set of anchor neighbours, or neighbours that have been localized in the subgraph $G_{v_i,\alpha}$ processed thus far. If node x_i has more than 3 anchor or localized neighbours, then it suffices to store only 3 such nodes.

■

Example. In figure 3.2, denote by $G_{v_i,1}$ the graph induced by the set of nodes $\{v_a, v_b, v_i, v_j, v_k\}$. $G_{v_i,1}$ is reduced onto the triangle $K_{v_i,1} = (v_i, v_j, v_k)$. Suppose that $S = \{v_a(1), v_b(2), v_i(1), v_j(2), v_k(3)\}$ is a possible network state of $G_{v_i,1}$, where the indices inside brackets are for possible node regions. Suppose also that in state S all edges shown in the figure exist between the five nodes. In addition, assume that v_b and v_j are anchor nodes. Then,

$$\begin{aligned} \mathit{type}(S) = \ & \{(v_i, loc = 1, count = 2, anchors = \{v_b, v_j\}), \\ & (v_j, loc = 2, count = 3, anchors = \emptyset), \\ & (v_k, loc = 3, count = 1, anchors = \{v_j\})\}. \end{aligned}$$

Storing such a state type in table $T_{v_i,1}$, when node v_i becomes a 3-leaf, allows the algorithm to forget about the exact connections in subgraph $G_{v_i,1}$ during subsequent processing steps. ■

For concept C4, each table $T_{v_i,\alpha}$, $\alpha = base, 1, 2, \cdots, k$, provides key-value mapping where

- each key is a network state type of the graph $G_{v_i,\alpha}$ reduced thus far onto the clique $K_{v_i,\alpha}$, and
- each value is the probability of obtaining a network state of $G_{v_i,\alpha}$ having the type specified by the corresponding key.

Example. Consider the previous example on the subgraph $G_{v_i,1}$ induced by the set of nodes $\{v_a, v_b, v_i, v_j, v_k\}$. When node v_i becomes a 3-leaf, table $T_{v_i,1}$ stores many network state types as keys. Let S be the state in the example, and let $key = \mathit{type}(S)$. The value $T_{v_i,1}(key)$ is the probability of obtaining a network state of subgraph $G_{v_i,1}$ of this particular type. Network state S of the previous example is one such network state accounted for by entry $T_{v_i,1}(key)$. ■

3.3 Main Algorithm

The overall algorithm presented in this chapter is organized around the following two functions:

Function Main (Algorithm 1): This function reduces the structure of the probabilistic graph (and the underlying partial k-tree) G by iteratively processing and then deleting nodes according to the given $PES = (v_1, v_2, \cdots, v_{n-k})$. After $n - k$

iterations, the solution is computed from the table associated with the remaining k-clique on nodes $(v_{n-k+1}, \cdots, v_n)$ (which, by design, contains the target node t).

Function Merge (Algorithm 2): The function takes as input two tables, denoted T_1 and T_2, associated with two overlapping cliques. It generates a new table, denoted T_{out}, by processing each pair of records in T_1 and T_2, respectively, to create a new record (to be stored in T_{out}).

```
Algorithm 1: Function Main$(G, V_{anchor}, t, PES)$
Input:  An instance of the $P$-$LOC$ problem where $G$ has a partial $k$-tree
        topology with a given $PES$=$(v_1, v_2, \ldots, v_{n-k})$
Output: PLoc$(G, t)$
Notation: $Temp$ is a temporary table.
1 Initialization: initialize a table $T_H$ for each $k$-clique $H$ of $\tilde{G}$. $T_H$ contains all
  possible state types on nodes of $H$.
2 for $(i = 1, 2, \ldots, |V| - k)$ do
3     $Temp = T_{v_i,1}$
4     for $(j = 2, 3, \ldots, k)$ do
5         $Temp =$ Merge $(Temp, T_{v_i,j})$
      end
6     $T_{v_i,base} =$ Merge $(Temp, T_{v_i,base})$
7     foreach $(key \in T_{v_i,base})$ do
8         delete $v_i$ and its associated position from $key$
      end
  end
9 return $PLoc$ $(G,t)$
  $= \sum$ values in table $T_{v_{n-k},base}$ corresponding to state types
        where target node $t$ can localize itself
```

Algorithm 2: Function $\text{Merge}(T_1, T_2)$

```
Input: Two tables $T_1$ and $T_2$ that may share common nodes
Output: A merged table $T_{out}$
1  Initialization: Clear table $T_{out}$
2  set $C =$ the set of common nodes between $T_1$ and $T_2$
3  foreach ($pair\ of\ state\ types\ key_1 \in T_1\ and\ key_2 \in T_2$) do
4      if ($any\ node\ in\ C\ lies\ in\ two\ different\ positions\ in\ key_1\ and\ key_2$) then
5          continue
       end
6      set $key_{out} =$ the state type obtained from node positions in $key_1$ and $key_2$
7      set $p_{out} = T_1(key_1) \times T_2(key_2)$ adjusted to take the effect of common
       nodes in $C$ into consideration
8      if ($key_{out} \in T_{out}$) then
9          update $T_{out}(k_{out})$ $+=$$p_{out}$
       end
       else
10         set $T_{out}(key_{out}) = p_{out}$
       end
   end
11 return $T_{out}$
```

3.3.1 Function Main

Function Main (Algorithm 1) iteratively reduces the structure of the k-tree $\tilde{G}$ into a k-clique $(v_{n-k+1}, \cdots, v_n)$ containing the target node t. The initialization step computes a table T_H associated with each k-clique H in the k-tree $\tilde{G}$. Initializing table T_H takes into account all possible network states on nodes of H. For each network state S of H, the adjacency relations $E_G(.,.)$ of the probabilistic graph applied to node positions in state S are used to determine the key $type(S)$. The probability $\Pr(S)$ of the network state S then contributes to the probability value stored in $T_H(type(S))$.

The main loop (the loop in Step 2) iteratively processes and deletes nodes in the PES. The main loop has two parts:

- The first part (Steps 3 to 6) process a node v_i by merging a sequence of tables $(T_{v_i,\alpha} : \alpha = base, 1, 2, \cdots, k)$ into one table. Tables are merged in pairs with the use of table $Temp$ to store intermediate results. Table $T_{v_i,base}$ stores the final result.

- The second part (the loop in Step 7) trims table $T_{v_i,base}$ by removing the k-leaf v_i from each key in the table.

After removing the $n-k$ nodes in the PES, table $T_{v_{n-k},base}$ contains state types over the remaining k-clique $(v_{n-k+1}, \cdots, v_n)$ (containing the target node t). The last step computes the answer $\mathrm{PLoc}(G,t)$ form that table.

3.3.2 Function Merge

Function Merge (Algorithm 2) takes as input two tables T_1 and T_2 that typically have a set C of common nodes. For example, for the partial 3-tree fragment in figure 3.2, T_1 (respectively, T_2) may be associated with clique (v_i, v_j, v_k) (respectively, (v_i, v_k, v_l)). Here, the set of common nodes $C = \{v_i, v_k\}$. Each table T_i, $i = 1, 2$, stores information about some subgraph, G_i, that has been reduced onto some k-clique K_i. The function works by processing each pair of state types (table keys) $key_1 \in T_1$ and $key_2 \in T_2$. key_1 and key_2 are *compatible* if each common node in C assumes the same region in both keys. Processing compatible state types key_1 and key_2 results in a new state type, denoted key_{out}, and an associated probability value, denoted p_{out}.

To explain the processing done on compatible keys key_1 and key_2, we consider two network states: S_i, $i = 1, 2$, where S_i is a network state of the subgraph G_i (reduced onto clique K_i), and $key_i = type(S_i)$. Merging network states S_1 and S_2 gives the union state $S_1 \bigcup S_2$. Such union is possible if the two keys key_1 and key_2 are compatible. In such cases, the desired state type $key_{out} = type(S_1 \bigcup S_2)$, and the desired p_{out} is the probability that a network state of type key_{out} arises in the graph $G_1 \bigcup G_2$. The type $type(S_1 \bigcup S_2)$ can be deduced by examining $type(S_1)$, $type(S_2)$, and the connectivity between the nodes in cliques K_1 and K_2 when their nodes are located according to the network state $S_1 \bigcup S_2$. The probability p_{out} is the product $T_1(key_1) \times T_2(key_2)$ divided by a correction term that takes into account the common nodes in the set C. This term equals $\prod p_x(i)$ where $x(i)$ is common between key_1 and key_2

3.4 Correctness

To show correctness, we use an approach similar to the one used in [21]. In the proof approach, a table $T_{v_i,\alpha}$ is considered *complete* with respect to the graph $G_{v_i,\alpha}$ that has been reduced onto the clique $K_{v_i,\alpha}$ if

1. for each $key \in T_{v_i,\alpha}$, the corresponding value $T_{v_i,\alpha}(key)$ is the probability of obtaining a state over the subgraph $G_{v_i,\alpha}$ of type key, and

2. each key not in $G_{v_i,\alpha}$ does not contribute to computing the solution $\mathrm{PLoc}(G, t)$.

One can then show that at the start of each iteration specified by Step 2 of function Main if v_i is a node in the current graph then each table $T_{v_i,\alpha}$, $\alpha = base, 1, 2, \cdots, k$, is complete with respect to the graph $G_{v_i,\alpha}$ reduced thus far onto the clique $K_{v_i,\alpha}$. This loop invariant can be shown by induction on the number of iterations done in the loop.

We remark that the above correctness argument applies to any k-perfect elimination sequence, $k \geq 1$, where the removal of the first $(n-k)$ nodes in the sequence reduces the graph to a k-clique that includes the target node t. As mentioned in the section on software verification below, this aspect is used to verify the correctness of our C++ implementation of the algorithm.

3.5 Running Time

Let n be the number of nodes in G, and ℓ_{max} be the maximum number of locations in the locality set of any node in G. We then have the following result.

Theorem 1 *In the algorithm, we have*

1. *The maximum length of any table is $O((4\ell_{max})^k)$.*

2. *The worst case running time is $O(n \cdot (4\ell_{max})^{2k})$.*

Proof.

- Part (1) follows since each key in any table has k nodes, and each node can have a $count$ value in the set $\{0, 1, 2, 3\}$, and a location in one of at most ℓ_{max} locations. Thus, the number of keys (i.e., the table length) is $O((4\ell_{max})^k)$.
- Part (2) follows since function Merge processes $O((4\ell_{max})^{2k})$ pairs of keys on each call, and the main function calls Merge $O(n)$ times.

■

3.6 Software verification

The devised algorithm is implemented in C++ with the use of the STL (Standard Template Library) container classes. The program length is about 1500 lines (with debugging code). To check correctness of the implementation, we have used the following approaches.

- Testing using probabilistic graphs with a small number of states (e.g., ≤ 16), and verifying the result manually.
- Solving a given instance using two different PESs and verifying that we obtain the same result.
- Testing using probabilistic graphs with simple structure. For example, probabilistic graphs whose underlying graphs are paths. Or, more generally, an underlying graph that has a cutedge (x, y). For such graphs, nodes x and y can assume positions where x can not reach y, or they can assume positions where x can reach y. See, e.g., figure 3.3. The final result can be obtained from the two subgraphs that arise if we remove the edge (x, y).

3.7 Numerical Results

We postpone the presentation and discussion of numerical results until the next chapter where we present another algorithm for solving the $\mathrm{P\text{-}LOC}$ problem.

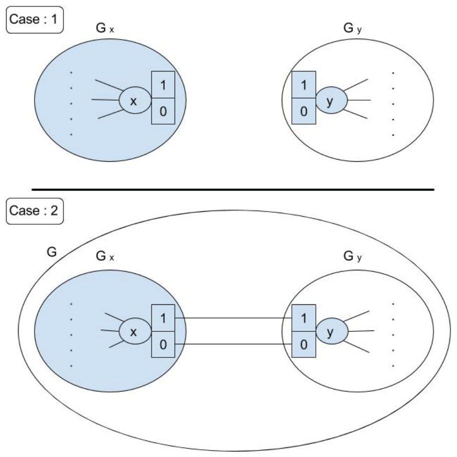

Figure 3.3: A probabilistic graph whose underlying graph contains a cutedge (x, y)

3.8 Concluding Remarks

The classes of partial k-trees, $k \geq 1$, form an important hierarchy of graph classes for algorithm design. In this chapter, we have presented an algorithm that gives exact solutions to the $\mathrm{P\text{-}LOC}$ problem on such classes. The algorithm runs in polynomial time for any fixed k. In the next chapter, we devise another algorithm to solve the $\mathrm{P\text{-}LOC}$ problem. Numerical results on the performance of the two algorithms are presented after introducing the second algorithm.

Chapter 4

A Factoring Algorithm

In the previous chapter, we have presented an exact algorithm to solve the P-LOC problem. The algorithm requires embedding of the underlying graph in a k-tree, $k \geq 1$. The running time of the algorithm grows exponentially with k, and it requires running to completion to get a result.

In this chapter, we devise an algorithm that uses a different approach to tackle the P-LOC problem. Given a probabilistic graph G of an instance of the P-LOC problem, the algorithm works by generating statistically disjoint network configurations of G through an iterative process. The algorithm then computes a lower bound (LB) on $\text{PLoc}(G, V_{anchor}, t)$ from the generated configurations where the target node t can be localized. The algorithm can also generate exact results, if sufficient iterations are performed. In addition to presenting the main algorithm, we present several numerical results to explore its performance and use.

4.1 Definitions and Notations

Throughout this chapter, we deal with a given instance of the P-LOC problem described by the parameters (G, V_{anchor}, t), where $G = (V, E_G, \text{Loc}, p)$ is a probabilistic graph. In this section, we start by introducing the following concepts needed to present our algorithm: *node states*, *network states*, *network configurations*, and *pathsets*.

Node states. We recall that each node $x \in V$ can be in any given region $x(i) \in \mathrm{Loc}(x)$ with probability $p_x(i)$. If such an event occurs, we say that node x is in state $x(i)$. We use the notation $(x, x(i))$ (or, (x, i) for short) to refer to node x being in state $x(i)$.

Network states. When each node x in the probabilistic graph G is in one of its possible $|\mathrm{Loc}(x)|$ states, we obtain a *network state* T of the probabilistic graph. T is written as $T = \{(x, i) : x \in V \text{ and } x(i) \in \mathrm{Loc}(x)\}$, or, $T = \{x(i) : x \in V \text{ and } x(i) \in \mathrm{Loc}(x)\}$ for short, as mentioned in chapter 1.

Example. Suppose that $V = \{v_1, v_2, v_3, v_4\}$, and the locality set of each node $v \in V$ has 2 regions, then $T = \{(v_1, 2), (v_2, 2), (v_3, 2), (v_4, 2)\}$ is a possible network state. ■

The probability that state T arises is $\Pr(T) = \prod_{(x,i) \in T} p_x(i)$. A network state T is *operating* if the target node t can be localized in state T. We write $op(T) = 1$ if T is operating. Else (if t can not be localized in T), then state T is failed, and we write $op(T) = 0$. We further denote by $OP(G, V_{anchor}, t)$ (or, $OP(G, t)$ for short) the set of all operating network states of G. Since $OP(G, t)$ is a set, any pair of network states contained therein differ in the state of at least one node. So, we can write:

$$\mathrm{PLoc}(G, V_{anchor}, t) = \sum_{T \in OP(G,t)} \Pr(T) \tag{4.1}$$

Network Configurations. A network configuration C assigns a state to some (but not necessarily all) nodes of a probabilistic graph G. Nodes not bound to a state in C are called *free* nodes. Thus, e.g., the null configuration where $C = \emptyset$ is a valid configuration where all nodes are free. The probability that a configuration C arises is $\Pr(C) = \prod_{(x,i) \in C} p_x(i)$. We adopt the convention that if $C = \emptyset$ then $\Pr(C) = 1.0$. A configuration C is *operating* (written $op(C) = 1$) if the target node t can be localized using only the assigned nodes in C (i.e., without the help of any free node in C). Else, C is a *failed* configuration, and we write $op(C) = 0$.

Statistically disjoint (s-disjoint) configurations. Two configurations C_1 and C_2 are s-disjoint if there exists at least one node that occurs in both configurations, and this node is assigned to different states in the two configurations. Such a difference between C_1 and C_2 allows us to write $\Pr(C_1 \bigcup C_2) = \Pr(C_1) + \Pr(C_2)$. Here, $C_1 \bigcup C_2$ refers to the event that either exactly C_1 occurs, or exactly C_2 occurs.

Use of network configurations versus use of network states. Let C be a configuration of the given probabilistic graph. Denote its set of free nodes by C_{free}. From the above definitions, it follows that configuration C encodes shared information between possibly many network states. Each such network state T satisfies $T \supseteq C$ (i.e., all assignments in C appear in T). For such cases, we say that network state T is *covered* by configuration C. Thus, if $OP_{conf}(G, V_{anchor}, t)$ (or, $OP_{conf}(G, t)$ for short) is a set of pairwise s-disjoint operating configurations that covers all operating states in $OP(G, t)$ then we can write

$$\mathrm{PLoc}(G, V_{anchor}, t) = \sum_{V \in OP_{conf}(G,t)} \Pr(C) \tag{4.2}$$

The advantage of designing an iterative algorithm that utilizes Equation 4.2 (rather than using Equation 4.1) is that the size of a good covering set $OP_{conf}(G, t)$ of operating configurations is typically much smaller than the size of the set of all operating network states $OP(G, t)$. However, generating such a good covering set of configurations requires carefully designed algorithms. The work of [1, 15] indicate the benefits of using configurations as the basis for designing an algorithm. Our work here extends such an approach to tackle the P-LOC problem.

Network pathsets. A pathset is an operating network configuration. We use this term as it is heavily used in the related literature on network reliability problems (see, e.g., [13]).

4.2 A Factoring Theorem

Our devised algorithm requires a method for generating pairwise s-disjoint pathsets in a systematic way. Our approach to generate such a set relies on the use of an extended form of a *factoring* theorem. The theorem is discussed in [13] and [19] in the context of network reliability problems with 2-state (operate/fail) nodes. A similar, but extended approach is also used in [1] in the context of wireless sensor networks (WSNs) with 2-state nodes, and [15] in the context of WSNs with 3-state nodes. In these latter references, the authors employ mechanisms to extend configurations to pathsets and then use the generated pathsets to guide the factoring process. We use a similar approach in this chapter.

To present the theorem, we introduce the following notations. Let $G = (V, E_G, \mathrm{Loc}, p)$ be a probabilistic graph, and $v \in V$ be one of its nodes with region $v(i) \in \mathrm{Loc}(v)$. Denote by $G \bullet (v, i)$ the constrained probabilistic graph G where node v is assigned region $v(i) \in \mathrm{Loc}(v)$.

Theorem 2

$$\mathrm{PLoc}(G, t) = \sum_{v(i) \in \mathrm{Loc}(v)} p_v(i) \times \mathrm{PLoc}(G \bullet (v, i), t)$$

The Theorem follows since the right hand side (RHS) exhausts all possible states of node v. The above theorem shows factoring on a single node. There are many situations where our algorithm performs factoring on a sequence of nodes to generate multiple configurations. To present more details, we mention that the algorithm applies factoring in the following program context.

1. The algorithm has computed (in a table called R) a set of s-disjoint configurations.

2. The algorithm selects a configuration C that has a high probability $\mathrm{Pr}(C)$ to process further.

3. The algorithm calls a function, called E2P (extend to a pathset), to compute an extension configuration C_{new} such that $C \bigcup C_{new}$ is a pathset

Three cases arise depending on whether function E2P fails or succeeds to extend C as follows:

- **Case 1.** Function E2P fails to extend C, and C has no free non-anchor nodes.
- **Case 2.** Function E2P fails to extend C, and C has at least one free non-anchor node v.
- **Case 3.** Function E2P succeeds in extending C to $C \bigcup C_{new}$.

Our factoring algorithm handles each case as follows.

- **Case 1.** Since C has no free non-anchor node, and C can not be extended so that the target node t can be localized, it follows that C does not cover any pathset. Configuration C is assigned a BAD state and ignored.
- **Case 2.** The algorithm uses the free non-anchor node v for factoring by generating the set of configurations
  - $\mathbf{C} = \{(C, (v, \alpha)) : v(\alpha) \in \text{Loc}(v)\}$ where $|\mathbf{C}| = |\text{Loc}(v)|$

  The notation $(C, (v, \alpha))$ refers to the configuration obtained by adding (v, α) to configuration C. As will be detailed later, this case may arise even though C is in fact extensible; that is, function E2P may give false negatives.
- **Case 3.** The algorithm uses the node-state pairs in C_{new}, say $C_{new} = ((v_1, \alpha_1), (v_2, \alpha_2), \cdots, (v_r, \alpha_r))$ to perform factoring. This is done by generating the following r sets of configurations:
  - $\mathbf{C}_1 = \{(C, (v_1, \overline{\alpha_1})) : \overline{\alpha_1} \neq \alpha_1\}$ where $|\mathbf{C}_1| = |\text{Loc}(v_1)| - 1$
  - $\mathbf{C}_2 = \{(C, (v_1, \alpha_1), (v_2, \overline{\alpha_2})) : \overline{\alpha_2} \neq \alpha_2\}$ where $|\mathbf{C}_2| = |\text{Loc}(v_2)| - 1$
  - $\mathbf{C}_3 = \{(C, (v_1, \alpha_1), (v_2, \alpha_2), (v_3, \overline{\alpha_3})) : \overline{\alpha_3} \neq \alpha_3\}$ where $|\mathbf{C}_3| = |\text{Loc}(v_3)| - 1$
  - $\cdots$
  - $\mathbf{C}_r = \{(C, (v_1, \alpha_1), (v_2, \alpha_2), \cdots, (v_r, \overline{\alpha_r})) : \overline{\alpha_r} \neq \alpha_r\}$ where $|\mathbf{C}_r| = |\text{Loc}(v_r)| - 1$

The following theorem is needed to show correctness.

Theorem 3 *1. In Case 2, the set* $\mathbf{C}$ *is a maximal set of new configurations that can be generated by node* v*, and added to table* R *while preserving the pairwise s-disjointedness property.*

2. In Case 3, the set $\mathbf{C}_{all} = \mathbf{C}_1 \bigcup \mathbf{C}_2 \bigcup \cdots \mathbf{C}_r \bigcup \{(C, C_{new})\}$ *(the last configuration in the union is the pathset* $C \bigcup C_{new}$*) is a maximal set of new configurations that can be generated by nodes of the computed configuration* C_{new}*, and added to table* R *while preserving the pairwise s-disjointedness property.*

Proof. For Case 2, one may verify that the set $\mathbf{C}$ is a maximal set of pairwise s-disjoint configurations that can be generated by using node v. To see that the pairwise s-disjointedness property is maintained after adding the new set to table R, let C', $C' \neq C$, be a configuration in table R that exists before adding $\mathbf{C}$. Let $C'' \in \mathbf{C}$ be one of the added configurations (containing node v). Since configuration C'' contains configuration C, it follows that C' is also s-disjoint from C'', as required.

For Case 3, one may verify that any two configurations in $\mathbf{C}_{all}$ are s-disjoint. Moreover, no new configuration that extends C by using nodes in C_{new} can be added to this set without violating the s-disjointedness property. Hence, $\mathbf{C}_{all}$ is a maximal set of configurations that satisfies the pairwise s-disjointedness property. The proof to show that all configurations in $\mathbf{C}_{all}$ can added to table R while preserving s-disjointedness follows the same argument as in Case 2. ■

4.3 Main Algorithm

The overall algorithm presented in this chapter is organized around the following three functions:

- Function Main (Algorithm 3): This function implements a loop responsible for generating and managing network configurations. The function terminates by computing a lower bound (or an exact solution) on $\mathrm{PLoc}(G, t)$.

- Function Factor (Algorithm 4): This function is called from the main function. It is responsible for generating pairwise s-disjoint configurations, and extending some of these configurations to pathsets.
- Function E2P (Algorithm 5): This function is called from function Factor to extend a given configuration C to a pathset, denoted $C \bigcup C_{new}$.

In this section, we present the details of function Main and Factor. Our design of these two functions follows closely the design presented in [15].

4.3.1 Data Structures

Function Main maintains the generated configurations in a table, denoted R. Table R is managed by adding new configurations to it. $R[i]$ denotes the ith row (record) in the table. Each row in table R is a 4-tuple, denoted $(C, C_{new}, state, p)$, where

- C is a configuration of the probabilistic graph G
- C_{new} is an extension to the configuration C, typically computed by function E2P, so that $C \bigcup C_{new}$ is a pathset. C_{new} is initialized with the empty set.
- $state$: each record in table R is in one of 3 possible states:
  - $ACTIVE$: Each iteration of the main loop of function Main chooses an $ACTIVE$ record $R[i]$ to process by attempting to extend its configuration $R[i].C$ to a pathset by calling function E2P.
  - $GOOD$: a record $R[i]$ that the algorithm succeeds in extending to a pathset and using it to generate new configurations is given the $GOOD$ state after processing.
  - BAD: a record $R[i]$ whose configuration $R[i].C$ has no free nodes and can not be extended to a pathset is given the BAD state.
- p: For a $GOOD$ record $R[i]$, we set $R[i].p = \Pr(C \bigcup C_{new})$. For an $ACTIVE$ record, $R[i].p = \Pr(C)$. Initially, table R stores the null configuration defined by $R[0] = (C = \emptyset, C_{new} = \emptyset, state = ACTIVE, p = 1.0)$.

4.3.2 Function Main

Function Main (Algorithm 3) takes as input an instance of the $\mathrm{P\text{-}LOC}$ problem, and invokes a main loop to generate and store s-disjoint configurations in table R. At the end, the function returns the sum of the probabilities of $GOOD$ configurations in the table. If the algorithm is run to completion (i.e., all $ACTIVE$ nodes are processed), the sum gives the exact solution $\mathrm{PLoc}(G, t)$. Else, the sum gives a LB on the solution. In more detail, Step 1 initializes table R with a single configuration (the null configuration). Steps 2 to 5 form the main loop. New configurations are generated by calling function Factor to process a selected record $R[i]$. Step 6 computes the final result.

```
Algorithm 3: Function Main($G, V_{anchor}, t$):
Input: An instance of the P-LOC problem
Output: The function generates pairwise s-disjoint configurations, and returns
        the sum of the found pathsets
Notation: $R$ is a table that stores the generated configurations
1 Initialization: set
      $R[0] = (C = \emptyset, C_{new} = \emptyset, state = ACTIVE, p = 1.0)$
  (i.e., initially, $R$ has has only one record)
2 for ($iter = 1, 2, \dots, maxIter$) do
3     Let $i$ be the index of an $ACTIVE$ configuration with largest probability
      in table $R$
4     Call function Factor: $result = \mathrm{Factor}(R, i, t)$
5     if ($result < 0$) then
          $R[i].state = BAD$
      end
  end
6 Return $\sum_{i = 0, 1, \dots \text{ where } R[i].state = GOOD} R[i].p$
```

4.3.3 Function Factor

Step 1 of function Factor (Algorithm 4) calls function $\mathrm{E2P}$ (extend to a pathset) to extend the input configuration $R[i].C$ to a pathset. As mentioned in Section 4.2, three cases arise. Steps 2, 3, and 5 handle these three possible cases.

Algorithm 4: Function $\text{Factor}(R, i, t)$**:**

```
Input: Table $R$ storing network configurations, and an index $i$ of a configuration
       $R[i].C$ to use in factoring
Output: If configuration $R[i].C$ can be extended to a pathset a $C \bigcup C_{new}$ by
        function E2P then apply factoring on the sequence of nodes in $C_{new}$ to
        generate new s-disjoint configurations. Else, if function E2P fails,
        choose a free non-anchor node $v$ to be used in factoring. Else, return
        $-1$ (failure)
1 $result = \text{E2P}(R[i].C, R[i].C_{new}, t)$
2 if ($result < 0$ AND there is no free non-anchor node) then
      return $result$
3 else if ($result < 0$ AND there is a free non-anchor node $v$) then
4     1. Generate a set of configurations $\mathbf{C}$ from $C$ and node $v$, as described in case 2
         in the text
      2. foreach (configuration $C' \in \mathbf{C}$) insert record
         $(C', C'_{new} = \emptyset, ACTIVE, p = \Pr(C'))$ in table $R$
      3. Delete record $R[i]$
5 else
      # an extension $C_{new}$ to configuration $C$ is found
6     1. Generate configuration sets $\mathbf{C}_1$, $\mathbf{C}_2$, $\cdots$ from $C$ and $C_{new}$, as described in
         case 3 in the text
      2. foreach (configuration $C' \in \mathbf{C}_1 \bigcup \mathbf{C}_2 \bigcup \mathbf{C}_3 \bigcup \cdots$) insert record
         $(C', C'_{new} = \emptyset, ACTIVE, p = \Pr(C'))$ in table $R$
      3. Set $R[i].status = GOOD$
  end
7 return $+1$
```

4.3.4 Remarks on the Main Algorithm

We conclude the presentation of the main algorithm by mentioning the following points that distinguish the above factoring algorithm framework from the frameworks presented in [1] and [15]:

- Nodes dealt with in this framework are multistate, with a relatively large number of states (as determined by their locality sets). In contrast, the above references deal only with 2-state or 3-state nodes.

- Our framework deals with function $\mathrm{E2P}$ that can return false negatives.
- In our framework, if every call to function $\mathrm{E2P}$ fails then the framework computes a LB by factoring on non-anchor nodes (or, an exact solution if sufficient iterations are allowed).

4.4 Extension to Pathsets

The extension to a pathset (E2P) problem is described as follows: given an instance (G, V_{anchor}, t) of the $\mathrm{P\text{-}LOC}$ problem, and a configuration C of the probabilistic graph G, find an extension C_{new} such that **(a)** $C \bigcup C_{new}$ is a pathset, and **(b)** $\Pr(C_{new})$ is as high as possible.

An effective solution to the E2P problem when integrated within the factoring algorithmic framework, presented in the previous section, allows for good use of each iteration. This follows since each iteration selects one configuration C from a heap of configurations, and then finds an extension to a pathset with as high $\Pr(C_{new})$ as possible. In this section, we devise a heuristic algorithm for the E2P problem. Our devised heuristic algorithm can give false negatives, i.e., it can fail to extend a given configuration C where an extension may actually exist. We first present the key ideas and data structures, and then present more algorithmic details.

4.4.1 Main Ideas

To extend a given input configuration C to a pathset $C \bigcup C_{new}$, the algorithm employs two sets of node-state pairs, denoted Q_{loc} and Q_{unloc}. Initially, set Q_{loc} stores elements (node-state pairs) corresponding to anchor nodes V_{anchor}. Some of these pairs are specified in the input configuration C; the remaining elements are generated from the free anchor nodes. The set Q_{loc} grows in each iteration of the algorithm as new elements can be localized. The growth stops when an element corresponding to the target node t is added to the set. Set Q_{unloc} contains elements (node-state pairs) of nodes that are not yet localized. This set shrinks at each iteration as new elements can be localized. The shrinking stops when an element that corresponds to the target node t is removed from the set.

Thus, the probabilistic graph nodes that appear in the elements stored in $Q_{loc} \bigcup Q_{unloc}$ is the set V of all nodes in G. We remark that the algorithm allows two elements corresponding to the same probabilistic graph node, say (x, i) and (x, j), where $i \neq j$ and $x(i), x(j) \in \text{Loc}(x)$, to appear in Q_{loc}. A difficulty arises in the final stage of the algorithm if the subset of Q_{loc} considered for computing the extension C_{new} contains such pair of elements. The algorithm executes in two phases.

- **Phase 1**, the algorithm iteratively selects an element, say $(x, i) \in Q_{unloc}$ that can be localized using three elements in Q_{loc}. The algorithm stores relevant information about this localization operation in an entry denoted $Q(x, i)$ where Q is a special table intended to store such information. Phase 1 terminates when an element corresponding to the target node t is localized.

- **Phase 2**, views the subset of elements in Q_{loc} that enable the localization of the element corresponding to the target node t as set of vertices V_H of a directed acyclic graph H. H has one source vertex corresponding to the element that includes the target node t. A directed arc from element (x, i) to element (y, j) exists if (y, j) is used to localize (x, i). The sink vertices in H are elements of Q_{loc} corresponding to anchor nodes in the UWSN. Phase 2 identifies the set V_H of vertices H. To compute the desired extension C_{new}, phase 2 performs the following steps:

  – Remove from V_H all elements belonging to the input configuration C.

  – If the remaining set $V_H \setminus C$ contains two elements (x, i), (x, j), $i \neq j$, for the same node x, then the algorithm fails and return -1 to the caller.

  – Else, set $C_{new} = V_H \setminus C$, and return $+1$ to the caller.

4.4.2 Data Structures

As discussed above, the main data structures used are the sets Q_{loc} and Q_{unloc}, and the table $Q(.,.)$. We now present more details about these data structures.

- Q_{loc} is a set whose elements are node-state pairs that have been localized thus far. Initially, Q_{loc} has the following elements:
  - For each anchor node x assigned to a state $x(i)$ in C, include (x, i) in Q_{loc}.
  - For each free anchor node x where $\text{Loc}(x) = \{x(1), x(2), \cdots, x(r)\}$ add elements $\{(x, i) : i = 1, 2 \cdots, r\}$ to Q_{loc}.
- Q_{unloc} is a set whose elements are node-state pairs that have not been localized thus far. Initially, Q_{unloc} has the following elements:
  - For each non-anchor node x assigned to a state $x(i)$ in C, include (x, i) in Q_{unloc}.
  - For each free non-anchor node x where $\text{Loc}(x) = \{x(1), x(2), \cdots, x(r)\}$ add elements $\{(x, i) : i = 1, 2 \cdots, r\}$ to Q_{unloc}.
- $Q(.,.)$ is a table that provides key-value mappings. Keys correspond to elements in $Q_{loc} \bigcup Q_{unloc}$. Two value types are stored for each key as explained below:
  - $Q(x, i).p$: the p field is a probability whose meaning depends on whether element (x, i) appears in Q_{loc} or Q_{unloc}. If $(x, i) \in Q_{loc}$ (respectively, $(x, i) \in Q_{unloc}$) then the value p is final (respectively, temporary). In either case, it is intended to reflect the cost incurred in using the element (x, i) to localize other elements. When $p = 1.0$, there is no (multiplicative) cost incurred in using (x, i). When p is small, there is a high (multiplicative) cost incurred in using (x, i).
  - $Q(x, i).adj$: the adj field is a set of elements used in localizing node (x, i). An anchor node x (positioned in any location) does not require the help of other elements for localization. For such a node x, we set $Q(x, i).adj = \emptyset$.

Initialization. Immediately after initializing sets Q_{loc} and Q_{unloc}, the algorithm initializes table $Q(.,.)$ for each element (x, i) as follows:

- Permanent settings:
  - If $(x, i) \in Q_{loc}$ and $(x, i) \in C$ then $x \in V_{anchor}$, and we set $Q(x, i).p = 1.0$, and $Q(x, i).adj = \emptyset$.
  - If $(x, i) \in Q_{loc}$ and $(x, i) \notin C$ then $x \in V_{anchor}$, and we set $Q(x, i).p = p_x(i)$, and $Q(x, i).adj = \emptyset$
- Temporary settings:
  - If $(x, i) \in Q_{unloc}$ and $(x, i) \in C$, we set $Q(x, i).p = 1.0$, and $Q(x, i).adj = \emptyset$.
  - If $(x, i) \in Q_{unloc}$ and $(x, i) \notin C$, we set $Q(x, i).p = p_x(i)$, and $Q(x, i).adj = \emptyset$

Example. Figure 4.1 shows an underlying graph (a 4×4 double-diagonal grid network) of a probabilistic graph. The probabilistic graph has the following structure:

- Each node x (0 to 15) has a locality set of two regions: region 0 (1) is the bottom (respectively, top) rectangle.
- The probability that node x lies in one of its two possible regions appears inside the respective rectangle. Note that node 0 (and also, 2, 3, and 4) has zero probability of visiting its top region.
- Nodes on the perimeter (coloured blue) are anchor nodes.
- Node 9 is the target node t.

Now, suppose that function E2P is invoked with input configuration $C = \{(0, 0), (2, 0), (3, 0), (4, 0)\}$. The function selects the following elements to move from Q_{unloc} to Q_{loc}.

- Node 5 where $Q(5,0).adj = \{(0,0),(4,0),(2,0)\}$ and $Q(5,0).p = 0.9 \times 1.0 \times 1.0 \times 1.0 = 0.9$

- Node $t = 9$ where $Q(9,0).adj = \{(4,0),(5,0),(8,0)\}$ and $Q(9,0).p = 0.9 \times 1.0 \times 0.9 \times 0.6 = .486$

The function then returns $C_{new} = \{(5,0),(8,0)\}$

■

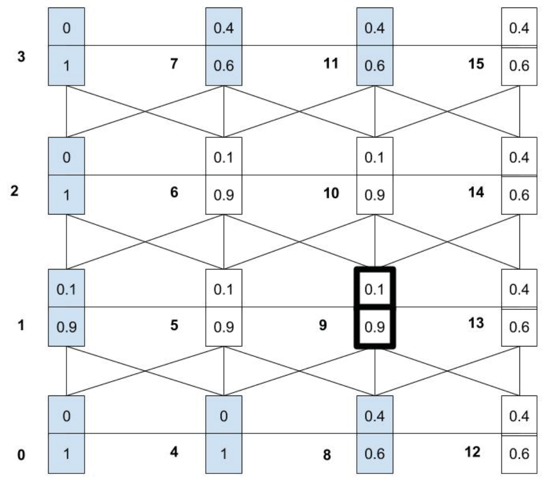

Figure 4.1: Example of function E2P; for each node x, region $x(0)$ (respectivly, $x(1)$) is the bottom (respectivly, top) rectangle

4.4.3 Function E2P

Function E2P (Algorithm 5) highlights the main steps of our devised algorithm. The function takes as input the parameters (C, C_{new}, t) of an instance of the E2P problem, and aims at extending C to a pathset $C \bigcup C_{new}$. The function executes in two phases.

- Steps 2 to 5 form phase 1. This phase utilizes a greedy strategy to build C_{new}. The greedy strategy selects at each step an element (x, α) with a large $Q(x, \alpha).p$ probability that can be moved from Q_{unloc} to Q_{loc}. Phase 1 terminates successfully when an element containing the target node t can be moved to Q_{loc}. The phase terminates with failure (returns -1) if it can not find a way to localize the target.

- Phase 2 executes if phase 1 succeeds. In such cases, phase 2 computes an extension configuration C_{new} from the nodes of the DAG H (assuming V_H does not contain two different elements corresponding to one node).

Algorithm 5: Function E2P(C, C_{new}, t)

```
Input: An instance $(G, V_{anchor}, t, C)$ of the $E2P$ problem
Output: If a solution is found return $+1$, and a solution in $C_{new}$. Else, return $-1$.
1 Initialize $Q_{loc}$, $Q_{unloc}$, and table $Q(.,.)$, as described in the text.
  # Phase 1
2 while (target $t$ is not in $Q_{loc}$) do
3     Choose an element $(x, \alpha) \in Q_{unloc}$ that can be localized by elements in
      $Q_{loc}$ with a high probability. Give preference to selecting the target node
      $t$.
4     if (no such element $(x, \alpha)$ exists) return $-1$
5     Let $\{(y_i, \alpha_i) : i = 1, 2, 3\}$ be the elements in $Q_{loc}$ used to localize $(x, \alpha)$.
          1. Set $Q(x, \alpha).p$ = the product of the $p$ fields in $Q(x, \alpha)$ and $Q(y, \alpha_i)$,
             $i = 1, 2, 3$
          2. Set $Q(x, \alpha).adj = \{(y_i, \alpha_i) : i = 1, 2, 3\}$
          3. Move element $(x, \alpha)$ from $Q_{unloc}$ to $Q_{loc}$
  end
  # Phase 2
6 Let $V_H$ = the elements that make the vertices of the DAG $H$
7 if ($V_H$ contains two elements corresponding to one node in two different
  states) then
      return $-1$
  else
      set $C_{new} = V_H \setminus C$
  end
8 return $+1$
```

4.4.4 Running Time

Let n be the number of nodes in G, and ℓ_{max} be the maximum number of regions in the locality set of any node in G, then we have the following observation.

Theorem 4

Function E2P *can be implemented to run in time* $O((n \cdot \ell_{max})^2 \cdot \log(n \cdot \ell_{max}))$.

Proof. Phase 1 dominates the running time of function E2P. This phase processes elements in $Q_{loc} \bigcup Q_{unloc}$ (at most $n \cdot \ell_{max}$ elements). The operation of selecting an element with the highest probability to move from Q_{loc} to Q_{unloc} can be implemented by keeping nodes in Q_{unloc} in a maximum heap using the $Q(.,.).p$ values as weights. The theorem then follows since phase 1 iterates at most $n \cdot \ell_{max}$ time to determine if the target node t can be localized. Each iteration computes the effect of the most recently added element to Q_{loc} on the remaining elements in Q_{unloc}; this step can be done in $O(n \cdot \ell_{max})$ time, followed by selecting a node from Q_{unloc} at the top of the heap, and refixing the heap.

4.5 Numerical Results

We implemented the factoring algorithm and the E2P function in the C++ language using the STL library of container classes. The overall system is about 800 executable lines. In this section we present numerical results that illustrate some aspects of the algorithm's performance and its potential applications. Our test probabilistic graphs have underlying graphs that form double-diagonal grids (abbreviated x-grids). Figure 4.2 illustrates one rectangle in such a grid. In the figure, each node x has a locality set with 2 regions $\mathrm{Loc}(x) = \{x(0), x(1)\}$. In the figure region $x(1)$ lies on top of region $x(0)$. The four nodes reach each other when they are all either in location 0, or all in location 1 of their respective locality sets. A $L \times W$, $L, W \geq 1$, x-grid has L columns and W rows. We use $G_{L\times W}$ to denote such a graph.

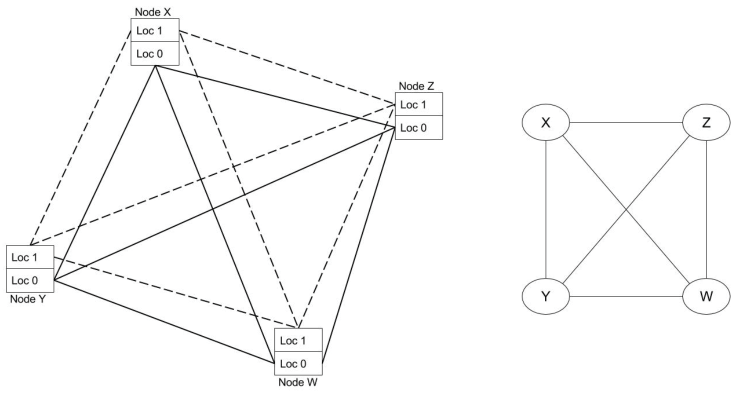

Figure 4.2: Structure of a rectangle in an x-grid

4.5.1 Results on both the k-tree and factoring algorithms

This set of experiments is intended to verify that the exact solution obtained by running the k-tree algorithm coincides with the exact solution obtained when the factoring algorithm is run to completion. Figure 4.4 is an example of underlying graph used in the testing (a subgraph of $G_{3\times4}$). Figure 4.3 is a redrawing of figure 4.4 illustrating that the graph is a partial 3-tree with PES= (8 , 9 , 4 , 0 , 1 , 5 , 2 , 3 , 6).

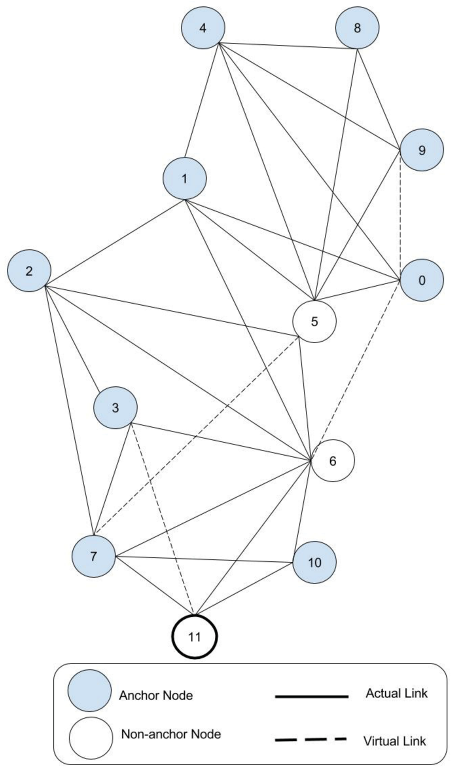

Figure 4.3: A partial 3-tree network

Table 4.1 (and also figure 4.5) summarizes the obtained results when solving the 5 x-grid networks shown in the table. We note that the exact result obtained in each case is the same. This provides a good evidence of the correctness of the implementation of both algorithms. The table also reports the running time required by each algorithm to solve each instance. We remark that in all test cases, the factoring algorithm is substantially faster than the k-tree algorithm (in some cases, by a factor of 10). We henceforth restrict our attention to the performance of the factoring algorithm in the remaining experiments.

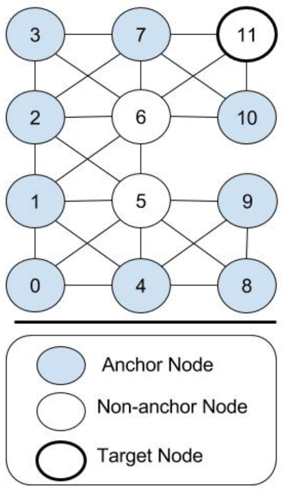

Figure 4.4: Network used in both k-tree and factoring algorithms

Table 4.1: Results on both k-tree and factoring algorithms

Network G	**k-tree algorithm**		**Factoring algorithm**	
	$\mathrm{PLoc}(G)$	Time (ms)	$\mathrm{PLoc}(G)$	Time (ms)
$G_{2\times2}$	0.125	4.241	0.125	0.778
$G_{2\times3}$	0.125	11.022	0.125	0.978
$G_{3\times3}$	0.121094	92.948	0.121094	3.885
$G_{3\times4}$	0.110352	150.077	0.110352	5.248
$G_{4\times4}$	0.0805969	170.141	0.0805969	20.792
$G_{5\times5}$	0.0625	254.784	0.0625	5.62

4.5.2 Effect of increasing network size

In these experiments, we investigate the effect of enlarging the network size on $\mathrm{PLoc}(G, t)$. The results are obtained using at most 1000 factoring iterations. Table 4.2 summarizes the obtained results on x-grids ranging from $G_{2\times2}$ to $G_{6\times6}$. Cases where exact results are obtained are distinguished from cases where LBs are obtained. Figure 4.6 illustrates one of the test networks where the target node (node 15) is at the upper right corner. Anchor nodes in test networks lie on the perimeter. However, a target node that lies on the perimeter is not an anchor node. As can be seen, $\mathrm{PLoc}(G, t)$ decreases as network size increases. This happens since nodes needed to localize t get away from anchor nodes as the network size increases.

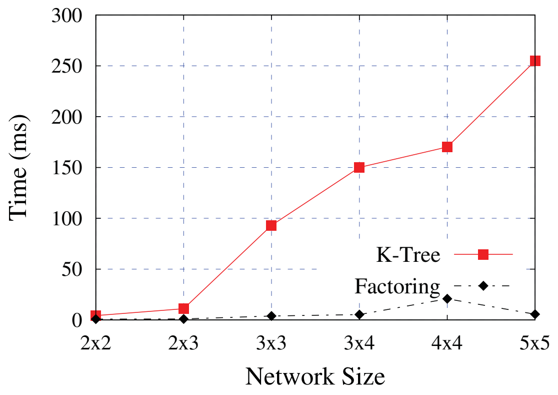

Figure 4.5: Results on both k-tree and factoring algorithms

Table 4.2: Effect of increasing network size (results obtained using the factoring algorithm)

Network G	**Target node** t	$\mathrm{PLoc}(G,t)$	**Running time (ms)**	**Good configurations**	**Bad configurations**	**Status**
$G_{2\times2}$	3	0.125	0.787	2	6	Exact
$G_{3\times2}$	5	0.125	1.018	2	6	Exact
$G_{3\times3}$	8	0.121094	3.59	10	8	Exact
$G_{4\times3}$	11	0.119141	5.884	14	14	Exact
$G_{4\times4}$	15	0.110718	38.45	90	106	Exact
$G_{5\times4}$	19	0.106539	223.918	440	642	Exact
$G_{5\times5}$	24	0.10048	1406	2274	909	LB
$G_{6\times6}$	35	0.0964446	4704.51	3182	128	LB

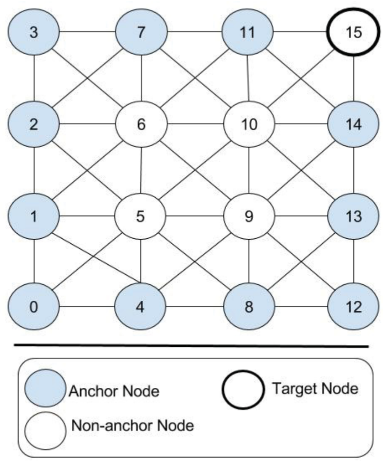

Figure 4.6: Network $G_{4\times4}$ and target node 15

4.5.3 Effect of the target node location

The location of the target node t relative to the anchor nodes V_{anchor}, and the level of connectivity between t and V_{anchor} determines the localization measure $\mathrm{PLoc}(G, V_{anchor}, t)$. To gain more insights into such effects, we have conducted three sets of experiments.

In the first set of experiments, we consider an UWSN whose underlying graph is $G_{7\times7}$, as in figure 4.7. Anchor nodes lie on the perimeter of the grid, except when a perimeter node is the target node t. Table 4.3 shows the results obtained by the factoring algorithm. The results obtained by the factoring algorithm show that $\mathrm{PLoc}(G, t)$ is best when a target node has at least 3 anchor neighbours (e.g., $t = 8$ or $t = 40$). The measure $\mathrm{PLoc}(G, t)$ decreases significantly when t does not have 3 anchor neighbours, since at least one non-anchor node should be localized first (e.g., $t = 0, 48, 16$, or 32). As the distance between t and anchor nodes increases (e.g., node 24) the $\mathrm{PLoc}(G, t)$ decreases to unacceptable levels, e.g., $\mathrm{PLoc}(G, t = 24) = 0.0131541$.

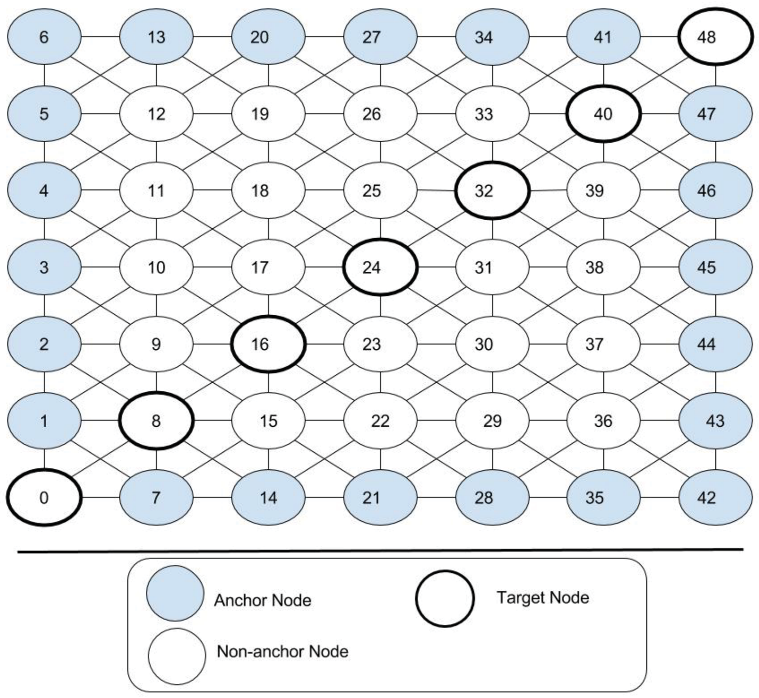

Figure 4.7: A $G_{7\times7}$ network (target node location varies on the diagonal)

In the second set of experiments, we focus on the weakest cases in the above set.

Table 4.3: Effect of varying target location in $G_{7\times7}$ network

Target position	**Target node** t	$\mathrm{PLoc}(G,t)$	**Running time (ms)**	**Good configurations**	**Bad configurations**
$(0,0)$	0	0.0952197	12611.5	3712	38
$(1,1)$	8	0.534746	13421.2	3994	18
$(2,2)$	16	0.136642	12804.9	3886	0
$(3,3)$	24	0.0131541	12956.9	3096	0
$(4,4)$	32	0.132606	8276.2	2810	0
$(5,5)$	40	0.534197	10828.7	3380	8
$(6,6)$	48	0.0952142	10899.3	3402	24

Namely, the cases conisdered here have underlying grid networks $G_{W\times W}$ where the target node is roughly in the middle (e.g., at coordinates $(\lceil W/2\rceil - 1, \lceil W/2\rceil - 1)$), and anchor nodes are on the perimeter. Table 4.4 presents the results obtained using the factoring algorithm. The results show the fast diminshing values of $\mathrm{PLoc}(G,t)$.

Table 4.4: Results on locating a target node in the middle of x-grid

Network G	**Target node** t	$\mathrm{PLoc}(G,t)$	**Running time (ms)**	**Good configurations**	**Bad configurations**	**Status**
$G_{2\times2}$	0	0.125	0.808	2	6	Exact
$G_{3\times3}$	4	0.855469	19.394	112	56	Exact
$G_{4\times4}$	5	0.696259	411.415	1324	1044	LB
$G_{5\times5}$	12	0.215773	1818.91	2762	30	LB
$G_{6\times6}$	14	0.150527	4956.85	3409	0	LB
$G_{7\times7}$	24	0.0131541	12982.2	3096	0	LB
$G_{8\times8}$	27	0.0148608	22769.2	3455	0	LB
$G_{9\times9}$	40	0.000944996	78497.8	4114	0	LB
$G_{10\times10}$	44	0.00123594	121704	4539	0	LB

In the third set of experiments, we check the weakest and strongest nodes in the network distribution. The test network size is 5×5 where the anchor nodes are on the perimeter and the target node position varies over all other positions in the network. If the target node is on the perimeter it becomes a non-anchor node. Figure 4.8 is an example when the target node is node 2. Figure 4.9, shows the results. The figure shows a symmerty between some nodes in the network.

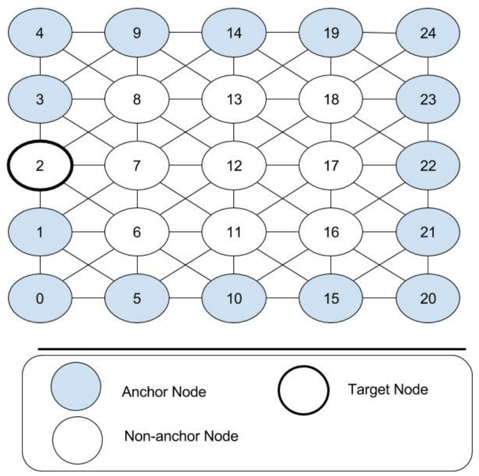

Figure 4.8: A $G_{5\times 5}$ network (target node location varies over all nodes)

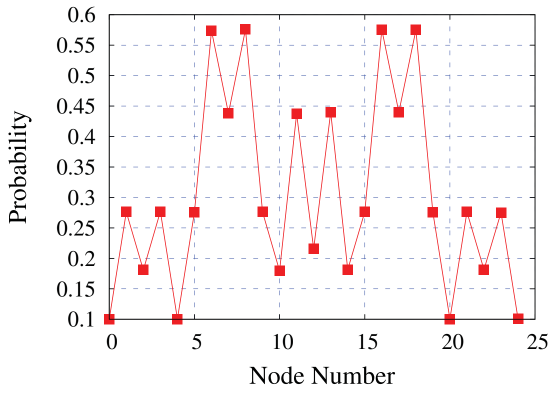

Figure 4.9: Varying target node location in $G_{5\times 5}$

4.5.4 Effect of the number and location distribution of anchor nodes

An UWSN design problem is to decide on an adequate number and location distribution of nodes that can serve as anchor nodes during a given interval of time. The objective may be to guarantee a minimum level of localization to each other node. To serve as anchor nodes, such set of nodes may need to be scheduled to rise to the surface in a synchronized way to activate their GPS devices. The thesis leaves such

Table 4.5: Results (LBs) on varying target node location in $G_{5\times 5}$

Target node t	$\mathrm{PLoc}(G,t)$	**Running time (ms)**	**Good configurations**	**Bad configurations**
0	0.100365	5843	2362	987
1	0.276172	10187	3283	296
2	0.180927	9328	3252	252
3	0.276927	8734	2997	287
4	0.10044	5671	2298	932
5	0.275858	9702	3304	318
6	0.573766	11312	3315	268
7	0.43824	8281	3097	160
8	0.575911	10093	3371	200
9	0.276129	9093	3193	293
10	0.179467	9110	3181	268
11	0.437317	8563	3177	218
12	0.215773	7484	2762	30
13	0.439888	7953	3018	178
14	0.181599	9062	3150	300
15	0.276193	8296	2838	278
16	0.575411	9750	3324	212
17	0.439984	8000	3000	179
18	0.574879	9484	3244	191
19	0.275602	7890	2796	258
20	0.100451	5656	2294	932
21	0.27608	8937	3116	284
22	0.181558	9140	3164	306
23	0.275182	8156	2926	268
24	0.10048	5656	2274	909

interesting network design problem as a topic for future research. However, the algorithms developed in the thesis provide tools to approach such class of UWSN design problems. To illustrate the use of the devised algorithms as tools, suppose the underlying graph of an UWSN is the $G_{5\times 5}$ graph where only 20% (i.e., 5 nodes) can be scheduled to rise to the surface to serve as anchor nodes. We assume the target node is 12 (see, figure 4.10).

Different sets of 5 nodes exist. Table 4.6 lists a collection of 5-node sets that a designer may wish to assess their benefit. Using the factoring algorithm (1000 iterations, and table of at most 10,000 configurations), one obtains the results in

Table 4.6.

Table 4.6: Exact results on locating a set of 5 anchor nodes in $G_{5\times 5}$

Set of 5 **Anchor nodes**	$\mathrm{PLoc}(G,t)$	**Good configurations**	**Bad configurations**	**Running time (ms)**
Case 1: $(0,1,2,3,4)$	0.00390625	2	16	6.603
Case 2: $(4,9,14,19,24)$	0.0078125	2	14	6.404
Case 3: $(3,4,8,9,13)$	0.0	0.0	1	0.407
Case 4: $(22,23,14,2,10)$	0.0078125	2	14	7.522
Case 5: $(6,11,16,17,7)$	0.3125	8	12	7.474
Case 6: $(5,6,7,8,9)$	0.125	2	6	4.639
Case 7: $(0,5,4,18,20)$	0.0	0.0	1	0.413
Case 8: $(0,4,24,20,10)$	0.0	0.0	1	0.287
Case 9: $(7,17,13,18,24)$	0.3125	8	12	8.981
Case 10: $(6,11,5,1,2)$	0.09375	4	8	5.481
Case 11: $(20,21,22,23,24)$	0.00390625	2	16	9.051
Case 12: $(4,8,18,20,17)$	0.125	2	6	2.245
Case 13: $(3,8,13,18,23)$	0.125	2	6	4.596

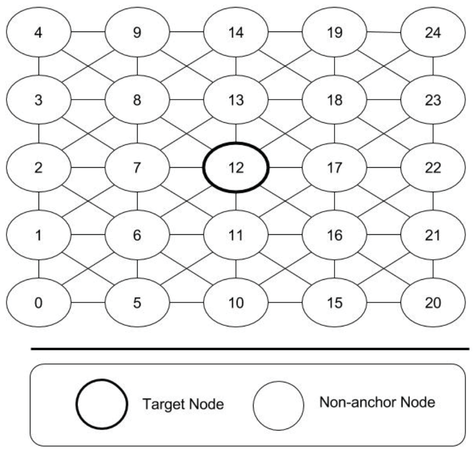

Figure 4.10: A $G_{5\times 5}$ network with a middle target node

4.6 Concluding Remarks

In this chapter, we have presented an iterative algorithm that can produce exact results (if run to completion) for the P-LOC problem. If run for any given user specified number of iterations, the algorithm gives a lower bound on $\mathrm{PLoc}(G, V_{anchor}, t)$.

Each iteration of the algorithm selects a network configuration C that has a high occurrence probability, and attempts to extend it to a pathset $C \bigcup C_{new}$ where $\Pr(C_{new})$ is as high as possible. We have implemented the algorithm, and the obtained results show that for cases that can be solved exactly within a reasonable time, the devised algorithm is remarkably faster than the k-tree algorithm presented in the previous chapter. We have also presented both exact and LB results that are used to develop insight in the dependency of the $\mathrm{PLoc}(G, t)$ measure on several probabilistic graph parameters such as its size, the location of the target node t, and the location distribution of the anchor nodes.

Chapter 5

Concluding Remarks

In this thesis, we have considered UWSNs deployed with mobile nodes. During a given interval of time, not all nodes can obtain their geographic locations on their own. Rather, a small subset of nodes may be able to rise to water surface and use their GPS devices to find their locations. Nodes that can localize themselves without the help of other nodes are called anchor nodes. Non-anchor nodes, during any interval of time, rely on anchor nodes to localize themselves. Each such node requires the help of 3 anchor nodes, or nodes that have succeeded in localizing themselves.

Due to node mobility, nodes can be in different geographic regions with different probabilities. To capture this aspect, the thesis adopts a probabilistic graph model that abstracts away from detailed knowledge of node movement trajectories. Rather, the model focuses on determining important geographic regions that can be visited by the nodes, and probability distributions describing how the nodes visit such regions.

Using the probabilistic graph model, we formalize a probabilistic localization problem called the $\mathrm{P\text{-}LOC}$ problem. The problem asks for computing the probability that a given target node t can localize itself by collaborating with other nodes in the network. Solving the formalized problem faces a challenge since the probabilistic graph has an exponential number of states, where each state specifies a geographic location for each node in the network. We note that although the topic of localization in UWSNs has been intensively examined in the literature (see, e.g., the surveys in [11, 16, 17]), the research direction taken by the $\mathrm{P\text{-}LOC}$ problem has not been adequately considered.

Our main contributions in the thesis include an exact algorithm to solve the problem on the class of partial k-trees, $k \geq 1$. The partial k-tree dealt with here captures links in all possible states of the probabilistic graph. The resulting algorithm has a running time that grows exponentially with k. Our second contribution is an iterative algorithm that works with graph structures, called network configurations. The algorithm can produce exact results if run to completion, or it can produce lower bounds if stopped after a given number of iterations.

For future research, we propose investigating the following related problems.

- In our work, the probabilistic graph $G = (V, E_G, \mathrm{Loc}, p)$ is assumed to be given as part of the input. Such a graph is constructed using some physical mobility model capable of estimating the location of a node under different physical parameters such as the node's initial deployment coordinates, direction and speed of water currents, etc. By varying the set of locations (regions) in such a model, we obtain different probabilistic graphs. These graphs may differ in their *accuracy* in representing the mobile network. More work needs to be done to identify useful measures that capture the accuracy of a probabilistic graph, and methods of constructing reasonably accurate probabilistic graphs.

- The factoring algorithm of Chapter 4 relies on using an efficient and accurate function $\mathrm{E2P}$. Our greedy approach to design the $\mathrm{E2P}$ allows the function to give false negatives. It is of interest to improve the performance of the function so as to minimize false negative whenever possible.

- Our formalized $\mathrm{P\text{-}LOC}$ problem concerns the localization of a single node. Another class of interesting problems is to compute the probability that a monitored event in a certain area can be localized. The consideration of an area (rather than a single node) introduces new challenges. It is hoped, however, that adopting the factoring algorithm to deal with the new class of problems gives effective results. More work is needed to develop such ideas further.

Bibliography

[1] H. M. F. AboElFotoh, E. S. Elmallah, and H. S. Hassanein. A flow-based reliability measure for wireless sensor networks. *International Journal of Sensor Networks*, 2(5/6):311–320, 2007.

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

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

[4] Y. Bar-Shalom, X. R. Li, and T. Kirubarajan. *Estimation with Applications to Tracking and Navigation*. Wiley-Interscience, New York, NY, USA, 2001.

[5] L. W. Beineke and R. E. Pippert. The enumeration of labelled 2-trees. *Notices American Mathematical Society*, 15:384, 1968.

[6] L. W. Beineke and R. E. Pippert. The number of labelled k-dimensional trees. *Journal of Combinatorial Theory*, 6(2):200–205, 1969.

[7] T. Bian, R. Venkatesan, and C. Li. Design and evaluation of a new localization scheme for underwater acoustic sensor networks. In *IEEE Global Telecommunications Conference (GLOBECOM)*, pages 1–5, Honolulu, HI, December 2009.

[8] T. Bian, R. Venkatesan, and C. Li. A refined localization method for underwater tetherless sensor networks. In *IEEE Wireless Communication and Networking Conference (WCNC)*, pages 1–6, Sydney, Australia, April 2010.

[9] A. Brandstädt, V. Bang Le, and J. P. Spinrad. *Graph classes: a survey*, volume 3. SIAM, Philadelphia, PA, USA, 1999.

[10] A. Caruso, F. Paparella, L. F. M. Vieira, M. Erol, and M. 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 771–779, Phoenix, AZ, April 2008.

[11] V. Chandrasekhar, W.K. Seach, Y.S. Choo, and H.V. Ee. Localization in underwater sensor networks-survey and challanges. In *The 1st ACM international workshop on Underwater networks (WUWNet)*, pages 33–40, 2006.

[12] W. Cheng, A.Y. Teymorian, L. Ma, X. Cheng, X. Lu, and Z. Lu. Underwater localization in sparse 3d acoustic sensor networks. In *The 27th IEEE Conference on Computer Communications (INFOCOM)*, pages 798–806, Phoenix, AZ, April 2008.

[13] C. J. Colbourn. *The Combinatorics of Network Reliability*. Oxford University Press, Inc., New York, USA, 1987.

[14] K.T. Dharan, C. Srimathi, and S. Park. A sweeper scheme for localization and mobility prediction in underwater acoustic sensor networks. In *IEEE OCEANS*, pages 1–7, Sydney, NSW, May 2010.

[15] M. Elmorsy, E. S. Elmallah, and H. M. F. AboElfotoh. On path exposure in probabilistic wireless sensor networks. In *IEEE 38th Conference on Local Computer Networks (LCN)*, pages 433–440, Sydney, NSW, October 2013.

[16] M. Erol-Kantarci, H.T. Mouftah, and S. Oktug. Localization techniques for underwater acoustic sensor networks. *IEEE Communications Magazine*, 48(12):152–158, December 2010.

[17] M. Erol-Kantarci, H.T. Mouftah, and S. Oktug. A survey of architectures and localoization techniques for underwater acoustic sensor networks. *IEEE Communications Surveys & Tutorials*, 13(3):487–502, March 2011.

[18] M. C. Golumbic and I. B. Hartman. *Graph theory, combinatorics and algorithms: Interdisciplinary applications*, volume 34. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[19] D. D. Harms, M. Kraetzl, C. J. Colbourn, and J. S. Devitt. *Network Reliability Experiments with a Symbolic Algebra Environment*. CRC Press, 1995.

[20] J. Heidemann, M. Stojanovic, and M. 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.

[21] Md Asadul Islam. Probabilistic connectivity of underwater sensor networks. Master's thesis, Department of Computing Science, University of Alberta, 2014.

[22] J. Liu, Z. Wang, J. Cui, S. Zhou, and B. Yang. A joint time synchronization and localization design for mobile underwater sensor networks. *IEEE Transactions on Mobile Computing*, 15(3):530–543, March 2016.

[23] H. Luo, K. Wu, Y. Gong, and L.M. Ni. Localization for drifting restricted floating ocean sensor networks. *IEEE Transactions on Vehicular Technology*, PP(99):1–15, February 2016.

[24] M.A. Mirza, M.Z. Shakir, and M. Slim-Alouini. A GPS-free passive acoustic localization scheme for underwater wireless sensor networks. In *Eighth IEEE International Conference on Mobile Ad-Hoc and Sensor Systems (MASS)*, pages 879–884, Valencia, April 2011.

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

[26] H. Ramezani, F. Fazel, M. Stojanovic, and G. Leus. Packet scheduling for underwater acoustic sensor network localization. In *IEEE International Conference on Communications Workshops (ICC)*, pages 108–113, Sydney, NSW, June 2014.

[27] H. Ramezani and G. Leus. L-MAC: Localization packet scheduling for an underwater acoustic sensor network. In *IEEE International Conference on Communications (ICC)*, pages 1459–1463, Budapest, June 2013.

[28] M.R. Schroeder. *Linear Prediction Theory: A Mathematical Basis for Adaptive systems*. Springer-Verlag, 1989.

[29] R. Venkatesan T. Bian and C. Li. An improved localization method using error probability distribution for underwater sensor networks. In *IEEE International Conference on Communications (ICC)*, pages 1–6, Cape Town, South Africa, May 2010.

[30] Z. Zhou, J. Cui, and A. Bagtzoglou. Scalable localization with mobility prediction for underwater sensor networks. In *The 27th IEEE Conference on Computer Communications (INFOCOM)*, Phoenix, AZ, April 2008.

[31] Z. Zhou, Z. Peng, J. Cui, Z. Shi, and A. Bagtzoglou. Scalable localization with mobility prediction for underwater sensor networks. *IEEE Transactions on Mobile Computing*, 10(3):335–348, March 2011.