On Probabilistic Connected Components in Underwater Sensor Networks

Salwa Abougamila
Department of Computing Science
University of Alberta
Edmonton, T6G 2E8, Canada
E-Mail: abougami@ualberta.ca

Mohammed Elmorsy
Department of Computing Science
University of Alberta
Edmonton, T6G 2E8, Canada
E-Mail: elmorsy@ualberta.ca

Ehab S. Elmallah
Department of Computing Science
University of Alberta
Edmonton, T6G 2E8, Canada
E-Mail: elmallah@ualberta.ca

***Abstract*—In this paper, we consider Underwater Sensor Networks (UWSNs) where nodes can move freely with underwater currents. In such networks, it is of interest to estimate the likelihood that a network has a connected component of (at least) a given size during some interval of time of interest after deployment. We formalize the problem using a probabilistic graph model, and develop a dynamic programming algorithm to solve the problem exactly when the graph has an interval representation. The interval representation model is motivated by scenarios where nodes move along a path in a relatively long but thin geographical area. We present numerical results on the performance of the algorithm under varying conditions of the required component size, and the size and structure of the set of intervals representing the probabilistic graph.**

I. Introduction

In this paper, we are interested in analyzing a certain connectivity property in mobile networks, with a particular emphasis on the class of Underwater Sensor Networks (UWSNs). Research on UWSNs continues to attract interest and gain momentum worldwide for their potential use in diverse important areas of applications. In particular, current surveys on UWSNs (e.g., [1]–[4]) discuss their application in many scientific, humanitarian, environmental, military, and industrial areas.

The design of UWSNs is known to face a number of challenges that do not exist in conventional terrestrial radio-based sensor networks. Prominent among these challenges is the difficulty of obtaining adequate communication bandwidth and low delay in the underwater communication channel. Currently, acoustic communication devices that provide connectivity over distances of 1 to 10 KMs, and a bandwidth of 10 KHz offer an attractive compromise over other devices that operate on very short, or very long ranges (see, e.g., [1]).

Another important design challenge comes from the strong influence of water currents, waves, and tides on the movement of the deployed nodes. Accordingly, research on UWSNs distinguishes between static, semi-mobile, and mobile deployments (see, e.g., [4]). In this respect, our interest in this paper is on fully mobile networks composed of nodes that move freely according to water currents with no, or little, self mobility capability (called RAFOS floats, and also drifters).

Since maintaining connectivity is crucial for tasks that require node collaboration, analyzing connectivity poses important, yet challenging, problems. We remark that in [5], the authors have presented an underwater mobility model, called the *meandering current mobility* (MCM) model, and used this model to analyze UWSNs with free-floating sensors. The work in [5] uses simulation to analyze various network coverage, connectivity, and localization aspects.

Our work here follows a parallel line of investigation, but focuses on the ability of the network to form a connected component of a minimum required size. To this end, we formalize a problem, called the *Probabilistic Component* (P-COMP) problem that considers the above performance measure. Our main contribution is the development of a solution methodology that is most useful when the nodes move along a relatively narrow, albeit long, pathway.

For an overview of some related results in the literature, we note that previous work on the design of UWSNs is extensive, and spans all layers of the networking protocol stack. For example, in the MAC layer, the authors of [6] investigate the effect of collision-tolerant MAC protocols on the exchange of localization messages in single-hop acoustic UWSNs. Surveys on node localization in UWSNs appear in [7]–[9]. In [10], the authors consider improving the accuracy of time synchronization and ranging between UWSN nodes; their devised method compensates for the stratification effect when performing ranging operations. In [11] and [12], the authors aim at improving localization accuracy. The work in [13] develops signal processing techniques for high frequency acoustic signals to infer relative positions of several sources.

In the UWSN literature, results involving large scale mobility (where nodes move long distances) are scarce compared to results involving small scale mobility. Our work here is related to large scale mobility. Previous results in this direction, such as the work in [10], [14], and [15], rely on the use of a method to predict node mobility. For example, in [10], an adaptive estimation approach, called *interactive multiple mode* (IMM) filter is used to devise a node localization scheme. And, in [14], the authors use a linear prediction scheme in their localization method to deal with nodes deployed near the seashore.

Aside from the simulation analysis done in [5] on quantifying the size of the largest connected component in UWSNs under large scale mobility, we are not aware of further results in this direction. Thus, our results reported here on the

978-1-5386-4413-3/18/$31.00 ©2018 IEEE

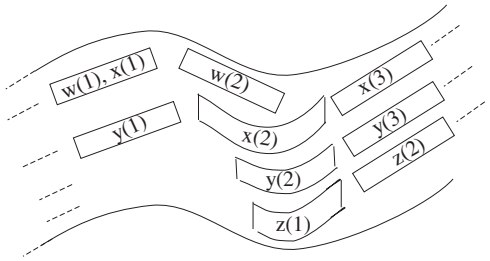

Fig. 1: A schematic of four nodes (w through z) moving along a pathway of a small cross-section

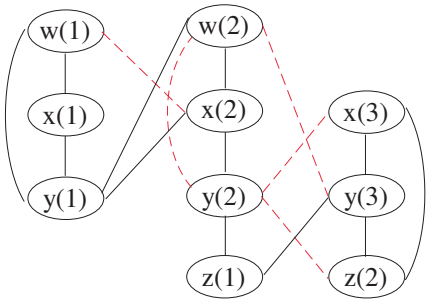

Fig. 2: Solid lines define an interval graph corresponding to the interval diagram in Fig. 3

P-COMP problem complements the above research work. Our approach relies on the use of a probabilistic locality model (reviewed below). Our previous work that uses this model includes the results in [16] on analyzing reachability to a sink node in tree-like probabilistic graphs, and [17], [18] on a probabilistic localization problem.

In addition, to adopting a probabilistic locality model, we also specialize our analysis to environments where nodes move along a narrow (e.g., a few multiples of a node transmission radius) but long pathways. Some of the obtained results in [5], e.g., consider a similar environment where node displacements away from a water jet axis are on the order of ± 4 KMs. Such range is considered small when the transmission range of a typical node is in the range of 3 to 10 KMs.

The main contributions of the paper can be summarized as follows. First, we present in Section III the concept of probabilistic interval graphs that is used to model movements of free-floating UWSN nodes along a relatively narrow but long pathways. Next, we present an exact polynomial time algorithm for solving the P-COMP problem on this class of probabilistic graphs.

The remaining part of the paper is organized as follows. In Section II we present basic assumptions and definitions that enable us to define the P-COMP problem. Section III discusses the class of interval graphs and introduces some needed notation. In Section IV we present the main algorithm, and in Section V, we present the obtained numerical results and draw some observations.

II. Network Model and Problem Formulation

In this section, we first review the probabilistic locality model used to define the P-COMP problem, and then define the problem. In essence, the probabilistic locality model provides a simple means of capturing the randomness in describing node locations while using as few parameters as possible. Our presentation of the model follows closely the definitions introduced in our previous work in [16]–[18].

A. Probabilistic Locality

We consider an UWSN composed of a set V of nodes. During an interval of time of interest after deployment in water, each node $x \in V$ can be in any one of a possible finite set of regions, denoted $\mathrm{Loc}(x) = \{x(1), x(2), x(3), \cdots\}$. We refer to $\mathrm{Loc}(x)$ as the *locality* set of x. In addition, during the interval of interest, each node x visits region $x(i) \in \mathrm{Loc}(x)$ with a known probability, denoted $p_x(i)$. If node x, when located anywhere in region $x(i)$, can reach node y, when located anywhere in region $y(j)$, we set the link indicator $E_G(x(i), y(j)) = 1$.

The above model consolidates relevant location information using a probabilistic graph abstraction. We denote such graph by $G = (V, E_G, \mathrm{Loc}, p)$.

To construct the above locality model, one needs to identify the locality sets over which the probability distribution is computed. Obtaining such information is possible by utilizing a physical node mobility model, such as the kinematic model used in the MCM of [5]. Optimizing the construction of the locality sets so that the resulting probabilistic graph yields accurate solutions to a given problem, however, is a separate problem that lies outside the scope of the paper.

We note, however, that the simplicity of the above model lends itself to other applications such as modelling movement of people or vehicles among different areas of interest.

Example. Fig. 1 sketches a possible distribution of locality sets of four nodes (w through z) as they move along an assumed relatively narrow pathway. Fig. 2 illustrates the nodes, and possible links (solid and dashed lines) that may exist in the probabilistic graph G of the network. ■

B. Problem Definition

Given a probabilistic graph G, a network state T of G is obtained by assigning a region $x(i) \in \mathrm{Loc}(x)$ to each node x. For example, $T = \{w(1), x(1), y(2), z(2)\}$ is a possible state of G in Fig. 2. In the literature, nodes of UWSNs are commonly assumed to be moving independently of each other [5]. The probability that network state T arises is then $\Pr(T) = \prod_{x(i) \in T} p_x(i)$. We now have:

Definition (the P-COMP problem). Given a probabilistic graph $G = (V, E_G, \mathrm{Loc}, p)$, and an integer k, find the probability $\mathrm{PComp}(G, k)$ that G is in a network state where one of the connected components has $\geq k$ nodes. ■

The P-COMP problem can be solved exactly by an algorithm that exhaustively enumerates all possible network states that satisfy the minimum required component size. More specifically, if we denote by $\mathrm{KCC}(G, k)$ the set of states

of G that satisfy the requirement, then $\mathrm{PComp}(G,k) = \sum_{T \in \mathrm{KCC}(G,k)} \Pr(T)$.

The time complexity of the above algorithm, however, grows exponentially with the size of the input problem instance. Hence, the need to find effective algorithms to compute lower (and upper) bounds on the solution. In this paper, we use the approximation to a probabilistic interval graph as a means of computing such bounds, as described next.

III. Interval Graph Approximation

The decision version of the P-COMP problem on arbitrary probabilistic graphs is expected to be NP-hard. Therefore, our work in this paper considers opportunities for approximating a given arbitrary UWSN probabilistic graph by a probabilistic *interval* graph. As mentioned below, this opens the door of obtaining efficient approximate solutions to the problem. We argue that good approximations exist when the nodes of the UWSN follow a pathway of narrow width. To this end, we review in this section standard graph theoretic information about the class of interval graphs (see, e.g., [19], [20]). Next, we introduce more notation needed to present the main algorithm.

A. Background on Interval Graphs

To get started, we review the following graph theoretic concepts [19], [20]. A finite collection of intervals $(I_1, I_2, \cdots, I_n)$ of the real line form an interval diagram. See, e.g., Fig. 3 where (for clarity) intervals are drawn parallel to the real line. The interval graph G associated with the collection of intervals has a node corresponding to each interval, and a link between two nodes if the intersection of the two intervals is nonempty. (Two intervals touching each other have a nonempty intersection.) As mention in [19], [20], interval graphs have applications in scheduling, behavioural psychology, temporal reasoning in artificial intelligence, and other fields.

Example. Fig. 3 illustrates an interval diagram on a collection of ten intervals. The solid lines in Fig. 2 show the corresponding interval graph. ■

Interval graphs enjoy a number of efficient polynomial time algorithms for solving problems that are NP-complete on arbitrary graphs. Given an arbitrary graph G, one can approximate its structure by an interval graph G^I that can either be a subgraph or a supergraph of G. The above facts enable the derivation of lower and upper bounds on solving a given NP-complete problem on G in polynomial time. Our work here makes use of this approach.

In the context of modelling node movements along a narrow pathway, we are encouraged by the possibility of finding a probabilistic interval graph G^I that serves as a good subgraph of the graph G of the given problem instance. In Fig. 3, e.g., we have laid out the intervals so as to stay close to the layout of regions in Fig 1. The corresponding interval graph is shown in Fig. 2 with solid links. Here, only 5 links (the dashed links) are lost in the approximation. The above argument provides a motivation to examine in more depth optimized ways to construct such interval subgraphs. We leave this problem for future research.

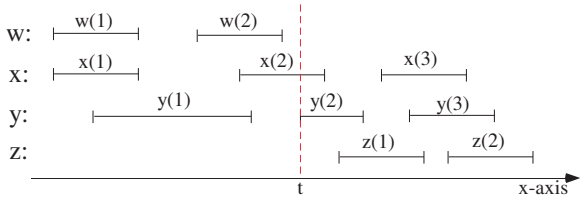

Fig. 3: An interval diagram corresponding to a probabilistic locality model where each of the four nodes (w through z) can be in any one of a number of possible regions

We henceforth assume that the graph G in a given instance of the P-COMP problem is an interval graph represented by a given interval diagram $D(G)$.

B. More Notation

For an interval I in $D(G)$, we denote by $left(I)$ and $right(I)$ the left and right points, respectively, of the interval I. For a given point t on the real line, we refer to the following sets of nodes:

- $V_{left}(t)$: node $x \in V_{left}(t)$ if all x's intervals end on, or to the left of, point t (i.e., each interval of x ends at a point $\leq t$)
- $V_{right}(t)$: node $x \in V_{right}(t)$ if all x's intervals start on, or to the right of, point t (i.e., each interval of x starts at a point $\geq t$)
- $V_{LR}(t)$: node $x \in V_{LR}(t)$ if x has at least one interval that ends on, or to the left of, point t, and at least one interval that starts on, or to the right of, point t
- $V_{include}(t)$: node $x \in V_{include}(t)$ if x has an interval that includes t. So, the interval starts (ends) at a point $\leq t$ (respectively, $\geq t$).

We remark that for any point t, $V_{left}(t)$, $V_{LR}(t)$, and $V_{right}(t)$ form a partition of the set V of nodes.

Example. For the diagram of Fig. 3, let $t = left(I_y(2))$ then $V_{left}(t) = \{w\}$, $V_{LR}(t) = \{x, y\}$, $V_{right}(t) = \{z\}$, and $V_{include}(t) = \{x, y\}$. ■

IV. Main Algorithm

In this section, we present an algorithm that takes as input an instance $(G, k, D(G))$ of the P-COMP problem on a probabilistic interval graph having an interval diagram $D(G)$. The algorithm computes an exact solution to the given problem instance. We start by presenting some needed ingredients.

A. Intervals Processing

The algorithm iteratively processes the intervals in $D(G)$ according to a sequence, denoted $\mathcal{I}$, that is computed once at the beginning of the algorithm. The sequence I defines a total order of the collection of intervals where an interval $I_x(i)$ appears before another interval $I_y(j)$ only if $left(I_x(i)) \leq left(I_y(j))$. Given such a sequence $\mathcal{I}$, we use $\mathcal{I}_{<y(j)}$ to denote

the collection of intervals that precedes $I_y(j)$ in $\mathcal{I}$. Likewise, we use $\mathcal{I}_{\leq y(j)}$ to denote $\mathcal{I}_{<y(j)} \bigcup \{I_y(j)\}$.

Example. For the diagram in Fig. 3, we may use $\mathcal{I} = (I_w(1), I_x(1), I_y(1), I_w(2), I_x(2), \cdots, I_z(2))$. For interval $I_y(2)$, we have $\mathcal{I}_{<y(2)} = \{I_w(1), I_x(1), I_y(1), I_w(2), I_x(2)\}$. ■

B. Partial Network States

As mentioned above, the algorithm iteratively processes the intervals according to the computed sequence $\mathcal{I}$, Processing an interval $I_y(j)$ entails identifying (and selecting) *useful structures* over the set of intervals $\mathcal{I}_{\leq y(j)}$ seen thus far by the algorithm. Here, selecting a *structure* corresponds to selecting a subset of intervals $\mathcal{I}'_{\leq y(j)} \subseteq \mathcal{I}_{\leq y(j)}$. Such a subset is considered *useful* if it can be extended to a valid network state (by adding more intervals, so that each node of V is associated with one of its intervals).

More specifically, a subset of nodes $\mathcal{I}'_{\leq y(j)}$ is useful if the following conditions hold:

a) If x has an interval in $\mathcal{I}_{\leq y(j)}$ then x has at most one interval in $\mathcal{I}'_{\leq y(j)}$.
b) If x has all of its intervals in $\mathcal{I}_{\leq y(j)}$ then exactly one interval of x appears in $\mathcal{I}'_{\leq y(j)}$.

Thus, for example, node x violates condition (a) if two of its intervals occur in $\mathcal{I}'_{\leq y(j)}$. Likewise, x violates condition (b) if none of its intervals occur in $\mathcal{I}'_{\leq y(j)}$. In either case, $\mathcal{I}'_{\leq y(j)}$ is **not** extensible to a network state of G.

Definition (partial network state). A subset of intervals forms a (valid) partial network state if it satisfies the above two conditions.

Example. In Fig. 3, the interval $I_y(2)$ is associated with the set $\mathcal{I}_{\leq y(2)} = \{I_w(1), I_x(1), I_y(1), I_w(2), I_x(2), I_y(2)\}$. We then have

- The subset $S_1 = \{I_w(1), I_y(1)\}$ corresponds to a valid partial network state since it can be extended to a full network state by adding, e.g., intervals $I_x(3)$ and $I_z(1)$.
- The subset $S_2 = \{I_w(1), I_w(2)\}$ does **not** correspond to a partial network state since the location of node w is ambiguous.
- The subset $S_3 = \{I_x(1), I_y(1)\}$ does **not** correspond to a partial network state since no interval of node w appears in S_3, and no interval of node w occurs outside $\mathcal{I}_{\leq y(2)}$.

■

C. Dynamic Program State Types

The algorithm processes an interval $I_y(j)$ in the computed sequence $\mathcal{I}$ by generating and storing information about groups of (valid) partial network states. The dynamic program attains its efficiency by restricting its attention to processing the identified groups. Each group corresponds to a dynamic program *state type*, as defined below.

Definition (state types). Let $I_y(j)$ be an interval in the computed sequence $\mathcal{I}$ of intervals, and let $S \subseteq \mathcal{I}_{\leq y(j)}$ be a subset of intervals that form a partial network state. The type of S, denoted $type(S)$, is a triple $(\alpha, I_{lead}, n_{cur})$ where

- α is the set of nodes associated with intervals in S that continue to have intervals not yet processed (i.e., intervals not in $\mathcal{I}_{\leq y(j)}$).
- I_{lead} (the leading interval of S) is an interval in S that extends the most to the right
- n_{cur} is the size of the connected component of the subgraph formed by the nodes in S that includes the node associated with I_{lead}; this component is the unique component in S that can be extended by adding more intervals not in $\mathcal{I}_{\leq y(j)}$

Example. In Fig. 3, assume that the algorithm is processing node $I_y(2)$. Thus, it operates on subsets of the set $\mathcal{I}_{\leq y(2)} = \{I_w(1), I_x(1), I_y(1), I_w(2), I_x(2), I_y(2)\}$. We then have the following example cases.

- The subset $S_4 = \{I_w(1), I_x(2), I_y(1)\}$ forms a valid partial network state where $type(S_4) = (\alpha = \{x, y\}, I_{lead} = I_x(2), n_{cur} = 2)$. Here, $\alpha = \{x, y\}$ since intervals $I_x(3)$ and $I_y(3)$ lie outside $\mathcal{I}_{\leq y(2)}$ (and hence nodes x and y should be remembered). Moreover, S_4 has a component $\{x, y\}$ of size 2 that can be further extended.
- The subset $S_5 = \{I_w(2), I_x(1)\}$ forms a valid partial network state where $type(S_5) = (\alpha = \{x\}, I_{lead} = I_w(2), n_{cur} = 1)$. Here, $\alpha = \{x\}$ since interval $I_x(3)$ lies outside $\mathcal{I}_{\leq y(2)}$ (and hence node x should be remembered). Moreover, S_5 has a component $\{w\}$ of size 1 that can be further extended.

■

D. Algorithm Details

The overall algorithm is organized around 3 functions called Main, Update_include, and Update_solution, as explained below. The functions use two tables, denoted P and Q, to store key-value mappings of the form $P(T)$ and $Q(T)$, where the key T is a dynamic program state type, and the value $P(T)$ is intended to be the occurrence probability of state type T. During any iteration, table P is used as a source, and table Q is used as a destination.

Function Main. Step 1 initializes the solution to zero, and table P to hold one state type $T = (\alpha = \emptyset, I_{lead} = null, n_{cur} = 0)$ that corresponds to the type of an empty partial network state. Step 2 is the main loop of the algorithm. Each iteration processes an interval, denoted $I_y(j)$, in the established total order $\mathcal{I}$. Step 3 is an inner loop that processes interval $I_y(j)$ by considering its effect on each state type $T = (\alpha, I_{lead}, n_{cur})$ in table P.

Step 4 considers the effect of adding (if possible) interval $I_y(j)$ to each network state of type is T. Here, adding $I_y(j)$ is possible only if no other interval of node y occurs in any state of type T (as determined by examining the set α).

Step 5 considers the effect of not adding (again, if possible) interval $I_y(j)$ to any partial network state of type T. Here, not

Function Main(G, k)**:**
Input: An instance of the P-COMP problem on probabilistic interval graphs
Output: return the solution PComp(G, k)

1. Initialize:
   a. $sol = 0.0$
   b. $P(\alpha = \emptyset, I_{lead} = null, n_{cur} = 0) = 1.0$
2. **foreach** (interval $I_y(j)$ in the total order $\mathcal{I}$)
   {
3. **foreach** (state type $T = (\alpha, I_{lead}, n_{cur})$ in table P)
   {
4. // Perform an inclusion step, if possible:
   if $(y \notin \alpha)$ Update_include$(P, Q, T, I_y(j))$
5. // Perform an exclusion step, if possible:
   if ($y \in \alpha$ OR ($y \notin \alpha$ AND $I_y(j)$ is not the last interval of y to be processed)) $Q(T)$ +=$P(T)$
   }
6. exchange pointers to tables P and Q; empty table Q
   }
7. return sol

Fig. 4: Pseudo-code for function Main

adding $I_y(j)$ is not possible if $I_y(j)$ is the last interval of node y, and node y does not appear in any partial network state of type T.

Step 6 exchanges the roles of tables P and Q in preparation of the next iteration. Finally, Step 7 returns the computed solution.

Function Update_include. This function takes as input a dynamic program state type $T = (\alpha, I_{lead}, n_{cur})$, and an interval $I_y(j)$ that can be added to any partial network state of type T. Step 3 deals with cases where such an addition operation creates a component of size $\geq k$. In such cases, function Update_solution is called, and no new dynamic program state type is added to table Q. Otherwise, one of steps 4, 5, or 6 is executed; these steps deal with the following disjoint cases:

- Case (Step 4): $I_{lead} \bigcap I_y(j) \neq \emptyset$, and $I_y(j)$ extends at least as far to the right
- Case (Step 5): $I_{lead} \bigcap I_y(j) \neq \emptyset$, and I_{lead} extends at least as far to the right
- Case (Step 6): interval I_{lead} is followed by $I_y(j)$ (no intersection)

In each of the above cases, a new dynamic program state type, denoted T', is created. Finally, Step 7 updates entry $Q(T')$; this step adds a new entry $Q(T')$ if no such entry currently exists in Q.

Function Update_solution. This function is called when adding an interval $I_y(j)$ to any partial network state of type T creates a component of the required minimum size. In such cases, the value of $P(T) \times p_y(j)$ needs an adjustment

Function Update_include$(P, Q, T = (\alpha, I_{lead}, n_{cur}), I_y(j))$**:**
Input: A state type T in table P, and an interval $I_y(j)$
Output: Either update sol, or form a new state type $T' = (\alpha', I'_{lead}, n'_{cur})$ corresponding to adding interval $I_y(j)$ to every partial network state (over the set of intervals processed thus far) of type T, and update table Q, accordingly.

1. **if** (intervals I_{lead} and $I_y(j)$ intersect each other)
   {
2. $n'_{cur} = n_{cur} + 1$
3. if $(n'_{cur} \geq k)$
   a. Update_solution$(P, T, I_y(j))$
   b. return
4. **Case** $(right(I_y(j)) \geq right(I_{lead}))$:
   a. $I'_{lead} = I_y(j)$
   b. Let x be the node associated with interval I_{lead}
   c. Let $t = left(I_y(j))$
   d. **if** (node $x \in V_{LR}(t)$) $\alpha' = \alpha \bigcup \{x\}$
      else $\alpha' = \alpha \setminus \{x\}$
5. **Case** $(right(I_y(j)) < right(I_{lead}))$:
   a. $I'_{lead} = I_{lead}$
   b. Let $t = right(I_y(j))$
   c. **if** (node $y \in V_{LR}(t)$) $\alpha' = \alpha \bigcup \{y\}$
      else $\alpha' = \alpha \setminus \{y\}$
   }
6. **Else** // intervals I_{lead} and $I_y(j)$ do not intersect each other
   a. $n'_{cur} = 1$ //start a new component
   b. $I'_{lead} = I_y(j)$
   c. Let x be the node associated with interval I_{lead}
   d. Let $t = left(I_y(j))$
   e. **if** (node $x \in V_{LR}(t)$) $\alpha' = \alpha \bigcup \{x\}$
      else $\alpha' = \alpha \setminus \{x\}$
7. $Q(T' = (\alpha', I'_{lead}, n'_{cur}))$ +=$P(T) \times p_y(j)$

Fig. 5: Pseudo-code for function Update_include

Function Update_solution$(P, T = (\alpha, I_{lead}, n_{cur}), I_y(j))$**:**
Input: A state type T in table P, and an interval $I_y(j)$. Adding $I_y(j)$ to any state of type T results in a state with a component of size $\geq k$.
Output: Increment sol by the contribution of the state type T, and the interval $I_y(j)$.

1. Let x be the node associated with interval I_{lead}
2. Let $t = left(I_y(j))$
3. Let β be the set of nodes that are not used in $\alpha \bigcup \{x, y\}$ and have intervals that start on, or to the right of, point t (i.e., $\beta = V_{LR}(t) \setminus (\alpha \bigcup \{x, y\})$)
4. For each node $w \in \beta$, let f_w be the sum of probabilities of intervals of w that have not been processed thus far
5. sol +=$P(T) \times p_y(j) \times \prod_{w \in \beta} f_w$

Fig. 6: Pseudo-code for function Update_solution

before being added to the solution. The adjustment is done by multiplying $P(T) \times p_y(j)$ by the contribution, denoted f_w in the function, of each node w, $w \neq y$, that has no interval in any state of type T, and w has one, or more, interval outside the set of intervals $\mathcal{I}_{\leq y(j)}$ processed thus far.

E. Correctness and Running Time

Correctness of the algorithm can be shown by showing that, at the end of each iteration of Step 2 in function Main, the algorithm stores in table P a complete set of all possible dynamic program state types. In addition, for each possible state type T, $P(T)$ is correct. These properties can be shown inductively by considering the exhaustive processing of each interval in the given problem instance.

We now derive the running time of the overall algorithm as a function of the following parameters: n, $n_{interval}$, $nmax_{LR}$, $nmax_i$, and k where

- n is the number of nodes in the network G,
- $n_{interval}$ is the number of intervals in the diagram $D(G)$,
- $nmax_{LR}$ denotes the size of a maximum set $V_{LR}(t)$, where t is the start point of some interval in the diagram $D(G)$
- $nmax_i$ denotes the size of a maximum set $V_{include}(t)$, where t is the start point of some interval in the diagram

Theorem 1: Function Main solves the P-COMP problem in time $O(2^{nmax_{LR}} \cdot nmax_i \cdot k \cdot n_{interval} \cdot \log(n))$.

Proof. Consider a general iteration of function Main in Step 3 where an interval, denoted $I_y(j)$ is processed. Let t be the point $left(I_y(j))$.

First, we observe that after Step 5, the length of table Q (i.e., the number of distinct state types in the table) is $O(2^{nmax_{LR}} \cdot nmax_i \cdot k)$. the observation follows since each dynamic program state type $(\alpha, I_{lead}, n_{cur})$ has the set $\alpha \subseteq V_{LR}(t)$, $I_{lead} \in V_{include}(t)$, and $n_{cur} \leq k-1$.

Second, we observe that each iteration of the loop in Step 3 processes each row of table P in $O(\log(n))$ time (assuming a suitable implementation for storing subsets of nodes), and that the number of iterations in Step 2 is $O(n_{interval})$.

The big-O expression in the theorem is the product of the above mentioned three big-O expressions. ■

V. Numerical Results

The main algorithm is implemented in C++ using the STL (Standard Template Library) to implement key data structures. The numerical experiments presented in this section aim at exploring the performance of the algorithm as we vary the parameters of the probabilistic interval graph G, and the minimum required component size k. We have experimented with running the algorithm on a laptop computer with 4 GByte of main memory. The maximum recorded user time of solving any of the problem instances discussed below is about 62 seconds.

Two types of probabilistic interval graphs are used below. The first type is constructed in a deterministic way so as to allow more analysis, as well as the ability to repeat the experiments exactly. The second type is constructed with more randomness.

In the deterministic construction method, each node $x \in V$ is associated with L intervals in the diagram $D(G)$ (thus, $|\text{Loc}(x)| = L$). All intervals have equal lengths, denoted I_{length}. Every two consecutive intervals of any node are separated by a gap of fixed width, denoted I_{gap}. Another parameter, denoted I_{offset}, introduces a skew in the interval diagram as follows: the first interval of node 1 starts at point 0 on the real line, the first interval of node 2 starts at point I_{offset}, the first interval of node 3 starts at point $2I_{offset}$, and so on. We refer to an interval diagram constructed in this way on n nodes as an $(L, I_{length}, I_{gap}, I_{offset})$ diagram (see, e.g., Fig. 7).

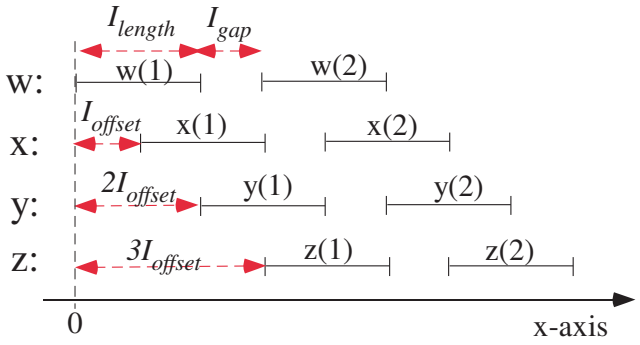

Fig. 7: An interval diagram constructed by the deterministic method where $(L = 2, I_{length} = 2, I_{gap} = 1, I_{offset} = 1)$, and $n = 4$ nodes

In the construction, we set $p_x(i) = \frac{1}{L}$ ($p_x(i)$ is the probability that node x visits region $x(i)$ in its locality set, or equivalently, node x is represented by its ith interval in the constructed diagram $D(G)$).

In the random construction method, the parameter I_{offset} is chosen randomly once for each node, and the parameter I_{gap} is chosen randomly each time a new value is needed. We use $uniform(a, b)$ to refer to a discrete uniform distribution between two real numbers a and b, $a \leq b$.

1. Experiments with the deterministic construction method. Fig. 8 to Fig. 11 present the obtained results as one parameter in the deterministic construction method is varied (on the x-axis), and the remaining parameters assume fixed values.

One striking property of the probabilistic graphs obtained by this method is that for small values of I_{offset} the corresponding interval diagrams have a small degree of skewness and a relatively small number of links. As I_{offset} increases, the skewness level increases causing the graph to have more links. Increasing I_{offset} beyond a certain value (that depends on the other parameters) has the effect of decreasing the number of links.

Thus, for any combination of setting the parameters L, I_{length}, I_{gap}, and n, there is a range of I_{offset} values

that maximizes the number of links, and consequently the computed $\mathrm{PComp}(G,k)$ value.

For example, in Fig. 8 to Fig. 10, setting $I_{offset} = 0.2$ gives uniformly better results, while in Fig. 11, setting $I_{offset} = 0.25$ gives uniformly better results.

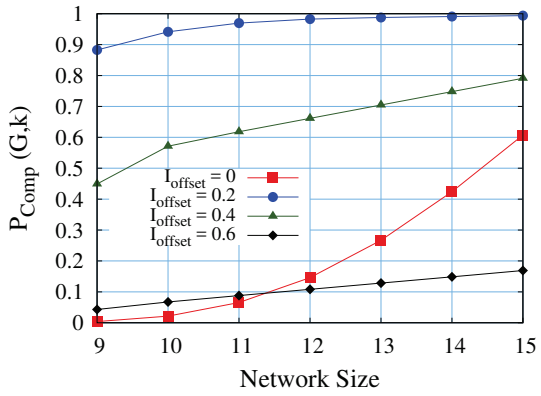

Fig. 8: Varying network size n, where ($L = 2, I_{length} = 1, I_{gap} = 1, I_{offset}$ on curves), $n \in [9, 15]$, and $k = 9$

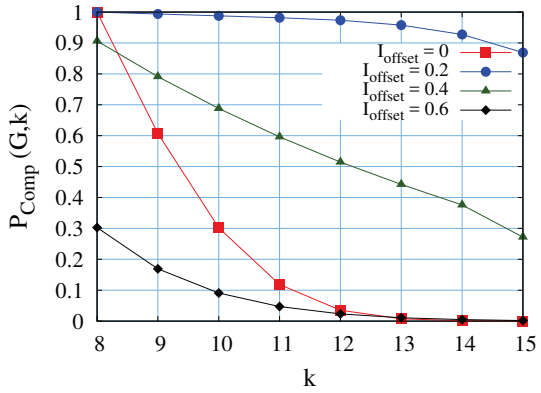

Fig. 9: Varying minimum component size k, where ($L = 2, I_{length} = 1, I_{gap} = 1, I_{offset}$ on curves), $n = 5$, and $k \in [8, 15]$

2. Experiments with the Monte Carlo method. We have also used the Monte Carlo method to verify our software implementation of the algorithm. In the Monte Carlo method, an estimate of each point in each curve in Fig. 8 to Fig. 11 is obtained by averaging 50 values. Each value is an estimate of $\mathrm{PComp}(G,k)$ computed from 500 randomly sampled network states (each random network state sample is drawn according to the node location probabilities of G). A 95% confidence interval is obtained for each point. The obtained Monte Carlo results are very close to the results presented in the above figures with a maximum absolute difference of at most 10^{-2} in the obtained solutions. The agreement of results obtained by the two methods provides an evidence of the correctness of the software implementations.

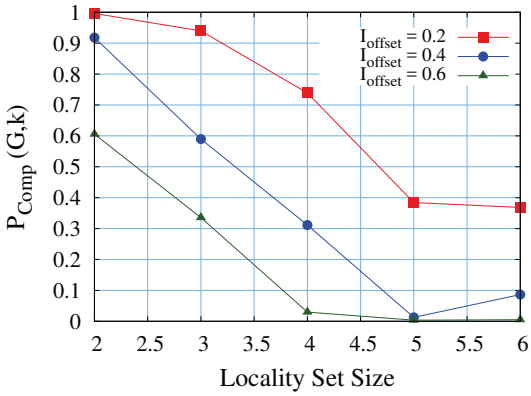

Fig. 10: Varying node locality set size L, where ($L \in [2, 6], I_{length} = \frac{2}{L}, I_{gap} = \frac{2}{L}, I_{offset}$ on curves), $n = 8$, and $k = 5$

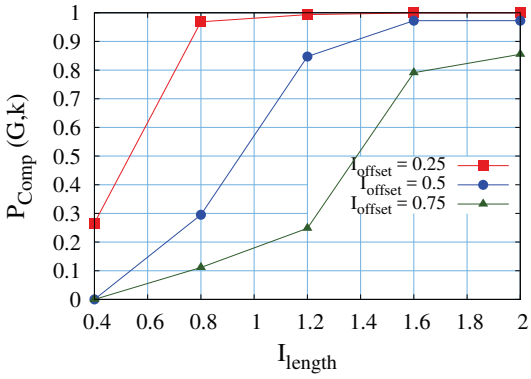

Fig. 11: Varying interval length I_{length}, where ($L = 2, I_{length} \in [0.4, 2.0], I_{gap} = 2, I_{offset}$ (on curves), $n = 15$, and $k = 9$

3. Experiments with the random construction method. Fig. 12 shows the obtained results when I_{gap} and I_{offset} are chosen randomly using a discrete uniform(0,4) distribution. The remaining parameters are ($L = 4, I_{length} = 2$). The above settings generate interval diagrams with significantly less interval intersections than the diagrams in the preceding set of experiments. The obtained curves correspond to setting the minimum component size to $k = \lfloor \frac{n}{2} \rfloor$, $\lfloor \frac{3n}{4} \rfloor$, and n. Given the above circumstances, it is notable that for networks of size $n = 8$ to 10, the probability of obtaining a component of size $\geq \lfloor \frac{3n}{4} \rfloor$ is at least 0.6.

VI. Concluding Remarks

Determining whether an UWSN with uncontrollable node mobility can form a connected component of at least a given size is a basic connectivity problem that arises in many situations. In this paper, we adopt a probabilistic locality model to describe the network during some interval of time of interest. Using the model, we present an algorithm that

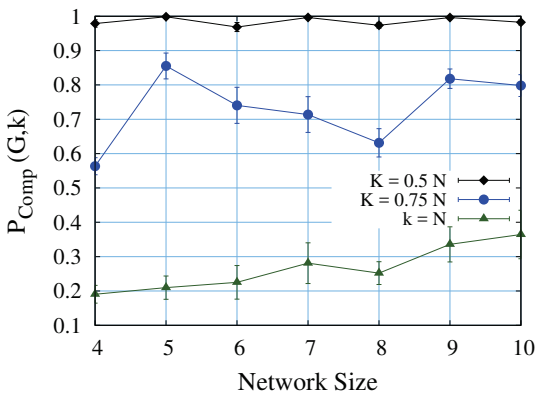

Fig. 12: Varying network size n, where $(L = 4, I_{length} = 2, I_{gap} \in random(0,4), I_{offset} \in random(0,4))$, $n \in [4,10]$, and k (on curves), confidence interval $= 95\%$

computes the probability that a connected component of the desired size can be formed. The presented methodology is most useful when the nodes follow a relatively narrow but long pathway. In such cases, the network topology can be approximated by a probabilistic interval graph. Extensions to the current work include investigating methods for obtaining optimized probabilistic interval graphs of a given arbitrary probabilistic graph that produce accurate results.

REFERENCES

[1] I. F. Akyildiz, D. Pompili, and T. Melodia, "Underwater acoustic sensor networks: research challenges," *Ad hoc networks*, vol. 3, no. 3, pp. 257–279, 2005.

[2] J. Partan, J. Kurose, and B. N. Levine, "A survey of practical issues in underwater networks," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 11, no. 4, pp. 23–33, 2007.

[3] Y. Xiao, *Underwater acoustic sensor networks*. CRC Press, 2010.

[4] 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*, vol. 370, no. 1958, pp. 158–175, 2012.

[5] 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*. IEEE, 2008, pp. 221–225.

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

[7] V. Chandrasekhar, W. Seach, Y. Choo, and H. Ee, "Localization in underwater sensor networks-survey and challanges," in *the 1st ACM International Workshop on Underwater Networks (WUWNet)*, 2006, pp. 33–40.

[8] M. Erol-Kantarci, H. Mouftah, and S. Oktug, "A survey of architectures and localoization techniques for underwater acoustic sensor networks," *IEEE Communications Surveys & Tutorials*, vol. 13, no. 3, pp. 487–502, March 2011.

[9] H.-P. Tan, R. Diamant, W. K. Seah, and M. Waldmeyer, "A survey of techniques and challenges in underwater localization," *Ocean Engineering*, vol. 38, no. 14, pp. 1663 – 1676, 2011.

[10] 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*, vol. 15, no. 3, pp. 530–543, March 2016.

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

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

[13] J. a. Gomes, E. Zamanizadeh, J. M. Bioucas-Dias, J. a. Alves, and T. C. Furfaro, "Building location awareness into acoustic communication links and networks through channel delay estimation," in *the Seventh ACM International Conference on Underwater Networks and Systems (WUWNet)*, 2012, pp. 24:1–24:8.

[14] 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*, vol. 10, no. 3, pp. 335–348, March 2011.

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

[16] M. A. Islam and E. S. Elmallah, "Tree bound on probabilistic connectivity of underwater sensor networks," in *the 13th Annual IEEE Workshop on Wireless Local Networks (WLN)*, Edmonton, Canada, 2014.

[17] S. Abougamila, M. Elmorsy, and E. S. Elmallah, "A factoring algorithm for probabilistic localization in underwater sensor networks," in *the IEEE International Conference on Communications (ICC)*, Paris, France, May 2017.

[18] ——, "A graph theoretic approach to localization under uncertainty," in *the IEEE International Conference on Communications (ICC)*, Kansas, USA, May 2018.

[19] M. C. Golumbic, *Algorithmic graph theory and perfect graphs*. Amsterdam, Netherland: North-Holland Publishing Co., 2004, vol. 57.

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