Probabilistic Connectivity of Underwater Sensor Networks

Md Asadul Islam

University of Alberta

mdasadul@ualberta.ca

August 21, 2014

Overview

1. Problem Formulation and Thesis Contributions
2. Overview on k-Trees and Partial k-Trees
3. Probabilistic Connectivity of Tree Network
4. A-CONN problem
5. AR-CONN and SR-CONN problems
6. Future Research Directions

Why UWSNs?

UWSNs fuelled by many important underwater sensing applications and services such as

- Scientific applications
- Industrial applications
- Military and homeland security applications
- Humanitarian applications

Challenges of the underwater communication channel

Radio Communication

- suffer strong attenuation in salt water
- short distances (6-20 m) and low data rates (1 Kbps)
- require large antennas and high transmission power

Optical Communication

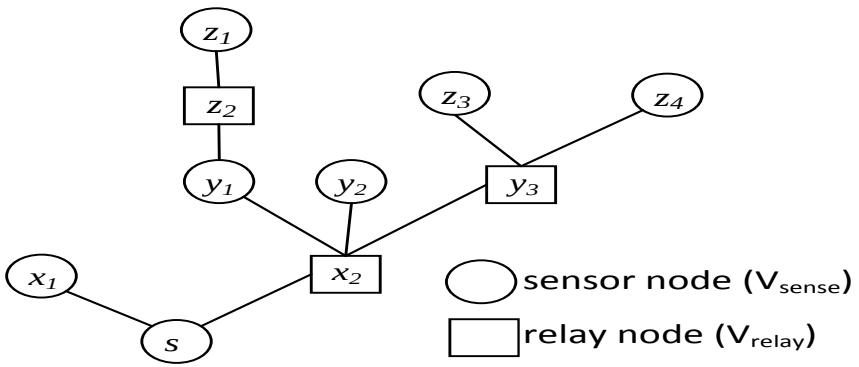
- strongly scattered and absorbed underwater
- limited to short distances (40 m)

Acoustic Communication

- suffers from attenuation, spreading, and noise
- very long delay because of low propagation speed.
- it is most practical method upto now

Node Locality Sets

- $V = V_{sense} \cup V_{relay}$ the set of nodes in a given UWSN
- The geographic area considered rectangles of a superimposed grid layout.
- At time T, each node x can be in any one of a possible set of grid rectangles denoted $\mathrm{Loc}(x) = \{x[1], x[2], \ldots\}$.
- Node x can be grid rectangle $x[i]$ with a certain probability $p_x(i)$.
- Truncate some locality sets of low probability for convenience thus, $\sum_{x[i] \in \mathrm{Loc}(x)} p_x(i) \leq 1$, if $\mathrm{Loc}(x)$ is truncated.



x_1	0.2	0.5	0.1	z_1	0.6	0.2	
	0.07	0.05	0.08		0.1	0.1	
x_2	0.25	0.3		z_2	0.5	0.25	0.2
	0.16	0.29					
y_1	0.2	0.3	0.1	z_3	0.25	0.36	
	0.1	0.15	0.15		0.14	0.25	
y_2	0.25	0.35		z_4	0.7	0.15	0.15
	0.25	0.15					
y_3	0.16	0.18	0.14	s	0.18	0.1	0.03
	0.3	0.22			0.34	0.1	0.14

Figure: Network with Probabilistic locality set

Node Reachability

- node x can reach node y if the acoustic signal strength from x to y (and vice versa) exceeds a certain threshold value.
- we set $E_G(x[i], y[j]) = 1$ iff the two nodes x and y can reach each other if they are located anywhere in their respective rectangles $x[i]$ and $y[j]$.
- connectivity between x and y is ignored if they can reach each other at some (but not all) pairs of points in their respective rectangles.
- ignoring connectivity in such cases results in computing lower bounds on the network connectivity, as required.

Kinematic Model

We note that this area is new to networking researchers where the obtained analytical results are rooted in the mathematically deep field of fluid dynamics.

- A particle pathline is a path followed by an individual particle in a flow
- A *stream* function denoted by ψ measures the volume flow rate per unit depth.
- Curves where ψ is constant are called *streamlines*

The stream function can be presented

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

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

$$\dot{x} = -\frac{\partial\psi}{\partial y}; \dot{y} = \frac{\partial\psi}{\partial x} \tag{2}$$

Kinematic Model (cont.)

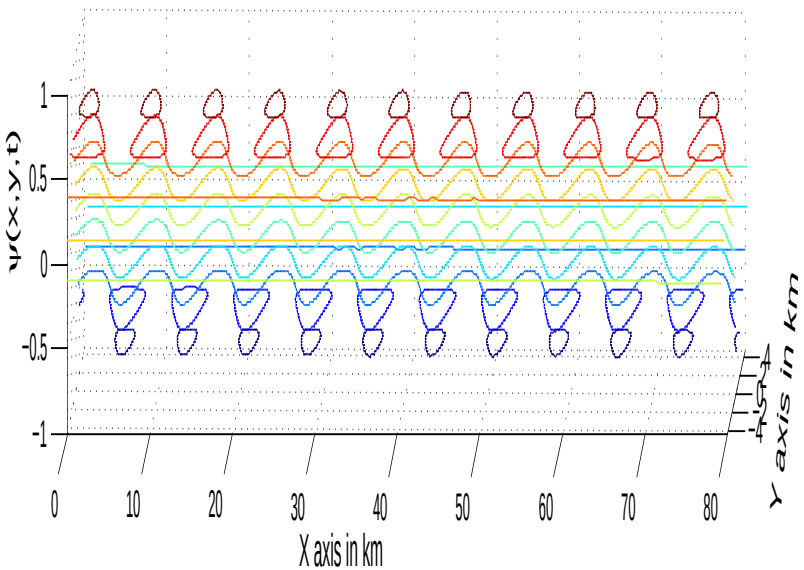


Figure: A 3D plot of 1

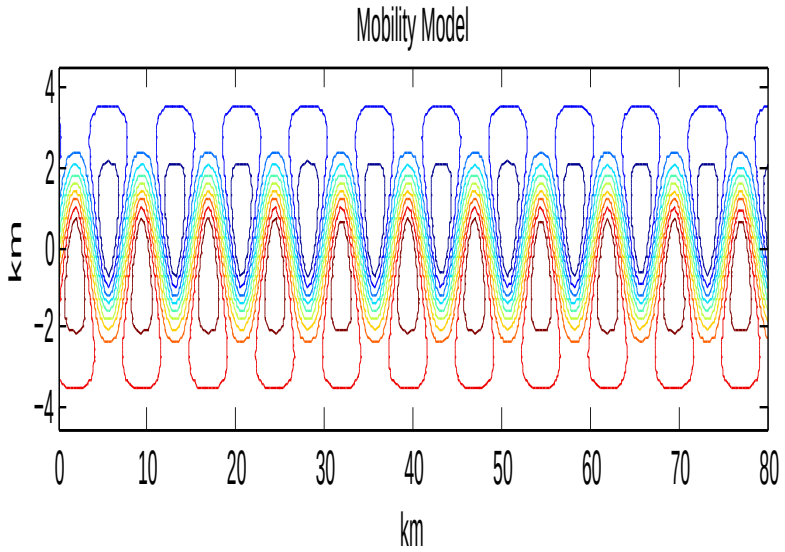


Figure: A plot of 1 at $t = 0$.

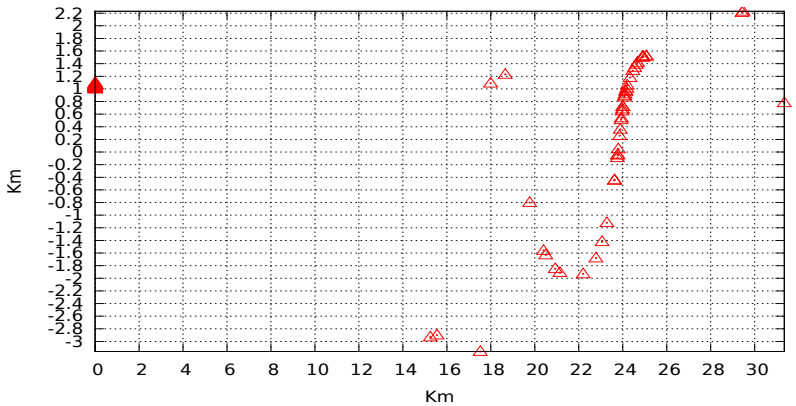


Figure: Start and end points of 50 nodes

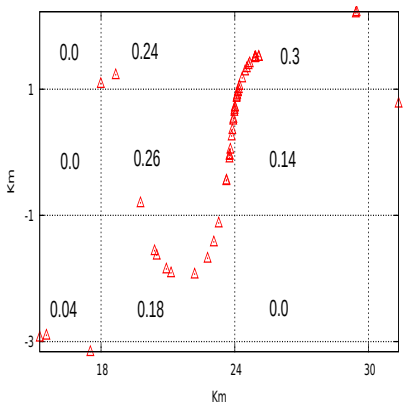


Figure: probabilistic distribution

Problem Definition

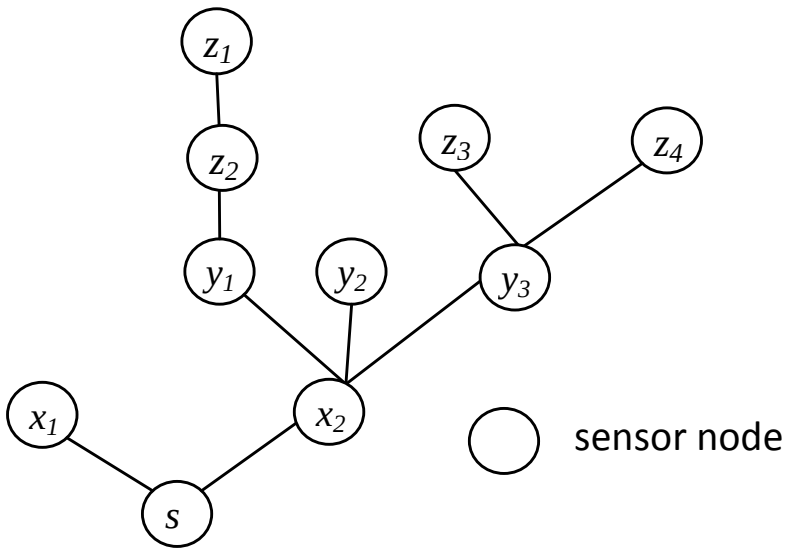
We define four probabilistic connectivity problem. They are

- A-CONN problem
- AR-CONN problem
- S-CONN problem
- SR-CONN problem

the A-CONN problem

Definition (**the *A-CONN* problem**)

Given a probabilistic network G with no relay nodes, compute the probability $Conn(G)$ that the network is in a state where the sink node s can reach all sensor nodes.



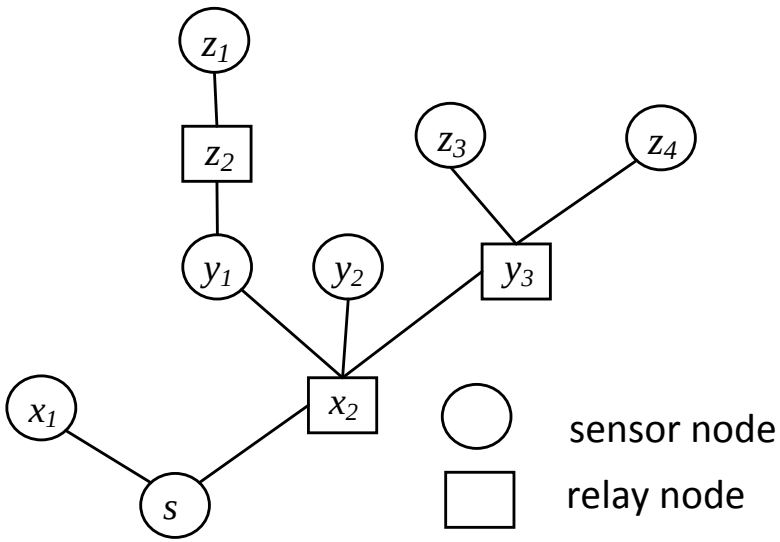
x_1	0.2	0.5	0.1	z_1	0.6 0.2 / 0.1 0.1
	0.07	0.05	0.08		
x_2	0.25	0.3		z_2	0.5 0.25 0.2
	0.16	0.29			
y_1	0.2	0.3	0.1	z_3	0.25 0.36 / 0.14 0.25
	0.1	0.15	0.15		
y_2	0.25	0.35		z_4	0.7 0.15 0.15
	0.25	0.15			
y_3	0.16	0.18	0.14	s	0.18 0.1 0.03 / 0.34 0.1 0.14
	0.3	0.22			

Figure: Network with Probabilistic locality set

the AR-CONN problem

Definition (**the** *AR-CONN* **problem**)

Given a probabilistic network G where V_{relay} is possibly non-empty, compute the probability $Conn(G)$ that the network is in a state where the sink node s can reach all sensor nodes.



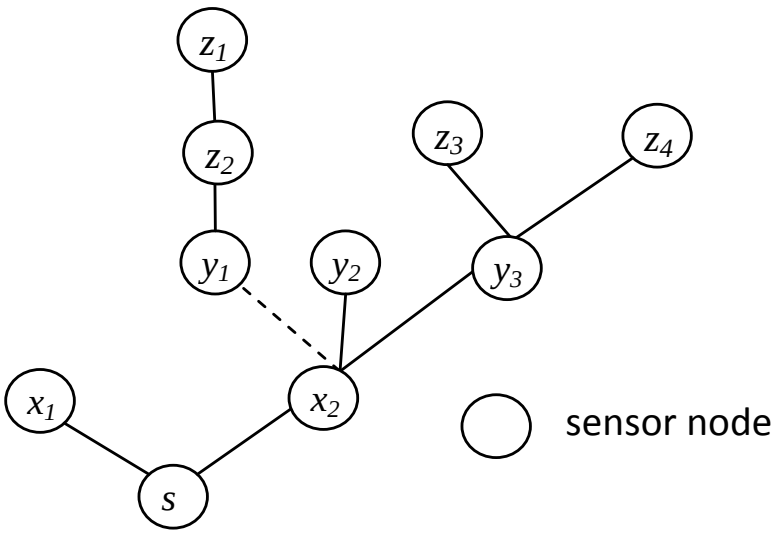
x_1	0.2	0.5	0.1	z_1	0.6 0.2 / 0.1 0.1
	0.07	0.05	0.08		
x_2	0.25	0.3		z_2	0.5 0.25 0.2
	0.16	0.29			
y_1	0.2	0.3	0.1	z_3	0.25 0.36 / 0.14 0.25
	0.1	0.15	0.15		
y_2	0.25	0.35		z_4	0.7 0.15 0.15
	0.25	0.15			
y_3	0.16	0.18	0.14	s	0.18 0.1 0.03 / 0.34 0.1 0.14
	0.3	0.22			

Figure: Network with Probabilistic locality set with relay nodes

the S-CONN problem

Definition (**the *S-CONN* problem**)

Given a probabilistic network G with no relay nodes, and a required number of sensor nodes $n_{req} \leq |V_{sense}|$, compute the probability $Conn(G, n_{req})$ that the network is in a state where the sink node s can reach a subset of sensor nodes having at least n_{req} sensor nodes.



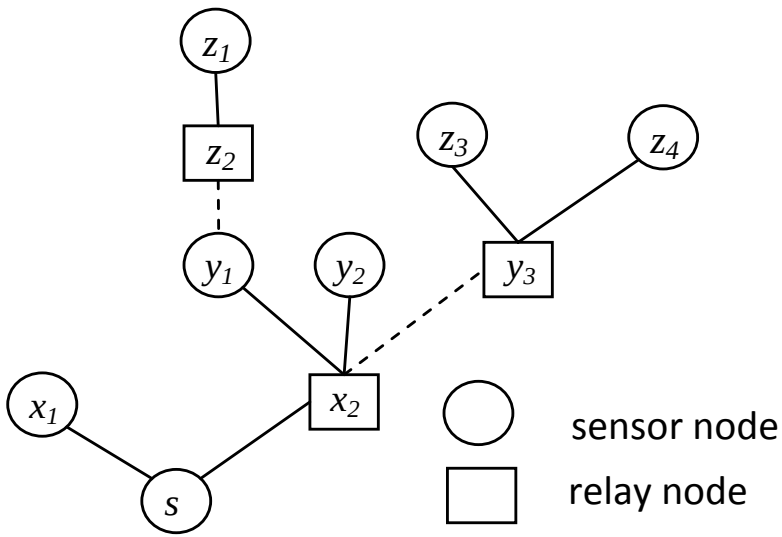
x_1	0.2	0.5	0.1	z_1	0.6 0.2
	0.07	0.05	0.08		0.1 0.1
x_2	0.25	0.3		z_2	0.5 0.25 0.2
	0.16	0.29			
y_1	0.2	0.3	0.1	z_3	0.25 0.36
	0.1	0.15	0.15		0.14 0.25
y_2	0.25	0.35		z_4	0.7 0.15 0.15
	0.25	0.15			
y_3	0.16	0.18	0.14	s	0.18 0.1 0.03
	0.3	0.22			0.34 0.1 0.14

Figure: Network with Probabilistic locality set and $n_{req} = 7$

the SR-CONN Problem

Definition (**the *SR-CONN* problem**)

Given a probabilistic network G where V_{relay} is possibly non-empty, and a required number of sensor nodes $n_{req} \leq |V_{sense}|$, compute the probability $Conn(G, n_{req})$ that the network is in a state where the sink node s can reach a subset of sensor nodes having at least n_{req} sensor nodes.



x_1	0.2	0.5	0.1	z_1	0.6 0.2 / 0.1 0.1
	0.07	0.05	0.08		
x_2	0.25	0.3		z_2	0.5 0.25 0.2
	0.16	0.29			
y_1	0.2	0.3	0.1	z_3	0.25 0.36 / 0.14 0.25
	0.1	0.15	0.15		
y_2	0.25	0.35		z_4	0.7 0.15 0.15
	0.25	0.15			
y_3	0.16	0.18	0.14	s	0.18 0.1 0.03 / 0.34 0.1 0.14
	0.3	0.22			

Figure: Network with Probabilistic locality set and relay nodes where $n_{req} = 4$

Network State

- A probabilistic graphs arises when network is in some particular network states.
- A state S of V can be specified by $\{v_1[i_1], v_2[i_2], ..., v_n[i_n]\}$
- If locations are independent, we have $Pr(S) = \prod_{v_\alpha \in V} p_{v_\alpha}[i_\alpha]$.
- In the *A-CONN* and *AR-CONN* problem, a state is **operating** if the sink s can reach all sensor nodes in V_{sense} .
- Similarly, in the *S-CONN* and *SR-CONN* problem, a state S is **operating** if the sink node s can reach a n_{req} sensor nodes.

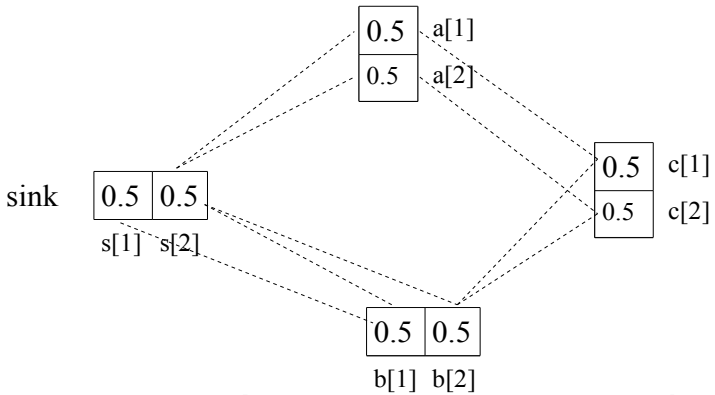


Figure: An example network

Thesis Contribution

An efficient dynamic programming algorithms

1. for the $\mathrm{SR\text{-}CONN}$ problem on probabilistic networks whose underlying graphs are trees.
2. for the $\mathrm{A\text{-}CONN}$ problem on partial k-trees.
3. for the $\mathrm{AR\text{-}CONN}$ and $\mathrm{SR\text{-}CONN}$ problems on partial k-trees.

All cases the algorithm runs in polynomial time for any fixed k.

k-Trees and Partial k-Trees

Definition

For a given integer $k \geq 1$, the class of k-trees is defined as follows

1. A k-clique is a k-tree.
2. If G_n is a k-tree on n nodes then the graph G_{n+1} obtained by adding a new node adjacent to every node in k-clique of G_n.

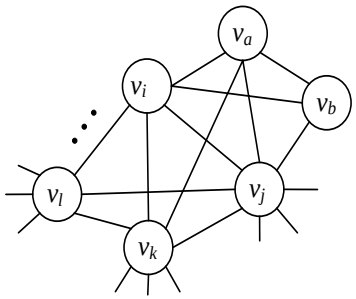


Figure: a fragment of a 3-tree

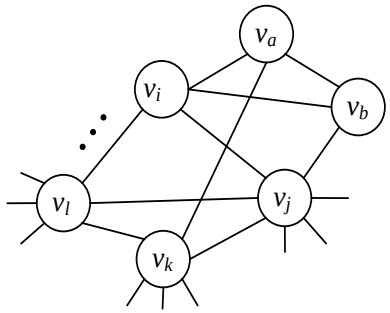


Figure: a fragment of partial 3-tree

A partial k-tree is a k-tree possibly missing some edges and a *k-perfect elimination sequence* (k-*PES*) of G is an ordering $(v_1, v_2, \ldots, v_r)$ of $v(G)$

Example

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

Dynamic Programing on Partial k-Trees

We inspired by Wald and Colbourn 1983 dynamic programming algorithm for solving the Steiner tree problem on partial 2-trees where each edge $\alpha = (x, y)$ of G associates six cost measures, which summarize the cost incurred so far of the subgraph S which has been reduced onto the edge (x, y)

1. $st(\alpha)$ is the minimum cost of a Steiner tree for S, in which x and y appear in the same tree.
2. $dt(\alpha)$ is the minimum cost of two disjoint trees for S including all targets, one tree involving x and the other y.
3. $yn(\alpha)$ is the minimum cost of a Steiner tree for S, which includes x but not y.
4. $ny(\alpha)$ is the minimum cost of a Steiner tree for S, which includes y but not x.
5. $nn(\alpha)$ is the minimum cost of a Steiner tree for S, which includes neither x nor y.
6. $none(\alpha)$ is the cost of omitting all vertices of S from a Steiner tree.

Our Approach

- Store the six measures associated with edge $\alpha = (x, y)$ in a table, denoted $T_{x,y}$.
- The table provides key-value mappings.
- The keys replace the use of some of the names st, dt, yn, etc. with set notation.
- Not all names correspond to keys in this formulation.

Examples of names that correspond to keys are:
$st = \{x, y\}_1, dt = \{x\}_1\{y\}_1, yn = \{x\}_1\{y\}_0,$ and $ny = \{x\}_0\{y\}_1$.

Set Partitions

- Given a set X, a partition of X is a set $\{X_1, X_2, \ldots, X_r\}$, $1 \leq r \leq |X|$ such that
  1. $X = \bigcup_{i=1,2,\ldots,r} X_i$.
  2. The X_is are pairwise disjoint.
- In our algorithm, X is a set of nodes in a k-clique, and the partition of X are used as part of states of dynamic programs.
- The number of all possible partition of a set X on n elements are known as Bell numbers, denoted B_n.
- The first few Bell numbers are

$$B_0 = B_1 = 1, B_2 = 2, B_3 = 5, B_4 = 15,$$
$$B_5 = 52, B_6 = 203, B_7 = 877, B_8 = 4140, \ldots$$

- Bell numbers satisfy the following recurrence equation:

$$B_0 = 1, B_1 = 1 \text{ and } B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k.$$

The Random and Greedy Method

The Random Method.

- **Case** $k = 1$**.** Choose any spanning tree.
- **Case** $k = 2$**.**
  - Choose a spanning tree $G' \subseteq G$.
  - Fix an ordering of the remaining edges not in G'.
  - For each edge e in the fixed order, test whether $G' + e$ is a partial 2-tree.
  - If yes, add e to G', and proceed to the next edge in the fixed order.
- **Case** $k = 3$**.**
  - Build a partial 2-tree $G' \subseteq G$.
  - Fix an ordering of the edges not in G'.
  - For each edge e, test whether $G' + e$ is a partial 3-tree.
  - If yes, add e to G', and proceed to the next edge in the fixed order.

The Greedy Method.

- **Case** $k = 1$**:** Choose a minimum spanning tree.
- **Case** $k = 2$ **and** 3**:** As in cases $k = 2$ and 3, respectively, of the random method, where we sort the remaining edges in a non-decreasing order of their costs to obtain the fixed order

Probabilistic Connectivity of Tree Network

- Notation and Definition
- Pseudo-code for $\mathrm{SR\text{-}CONN}$ problem on tree network
- Running time Analysis

Notation and definition

- $type(x)$: $type(x) = 0$ and 1 if x is a relay node and a sensor node respectively.
- $n_{sense}(X)$: # of sensor nodes in a given subset of nodes $X \subseteq V$.
- $n_{relay}(X)$: # of relay nodes in a given subset of nodes $X \subseteq V$.
- $n(X) = n_{sense} + n_{relay}$.
- $n_{sense,min}(X)$: The minimum number of sensor nodes in a given subset $X \subseteq V$. So, $n_{sense,min}(X) = max(0, n_{req} - n_{sense}(\overline{X}))$

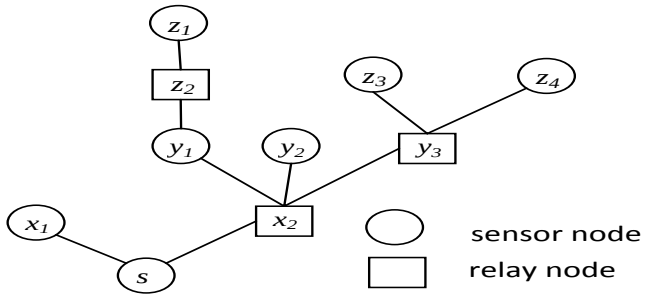


Figure: A tree network

Example

In figure , consider node x_2. $n(V_{x_2}) = 8$ where $n_{sense}(V_{x_2}) = 5$ and $n_{relay}(V_{x_2}) = 3$. Assuming $n_{req} = 5$ in an instance of the SR-CONN problem then $n_{sense,min}(V_{x_2}) = 3 = n_{req} - n_{sense}(\{s, x_1\}) = 5 - 2$.

SR-CONN problem

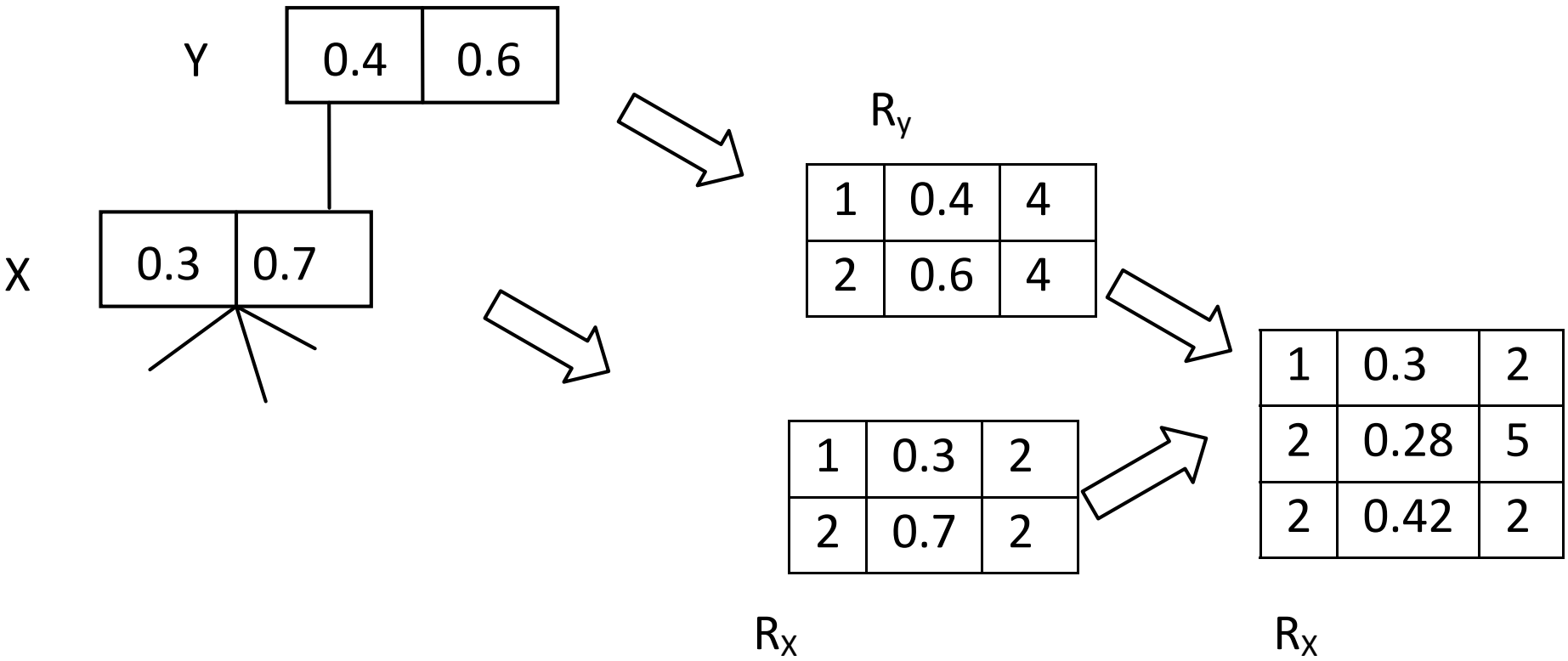


Figure: Table merge in Tree networks and $n_{req} = 5$

Pseudo-code for function Conn

Function $\mathrm{Conn}(G, T, n_{req})$
Input: the *SR-CONN* problem where G has a tree topology T with no relay leaves
Output: $Conn(G, n_{req})$

1. **foreach** (node x and a valid location index i)
   set $R_x(i, type(x)) = 1$
2. **while** (T has at least 2 nodes)
   {
3. Let y be a non-sink leaf of T, and $x = parent(y)$
4. **foreach** (key $(i, count) \in R_y$) $R_y(i, count) *= p_y(i)$
5. set $R'_x = \phi$
6. **foreach** (pair of keys $(i_x, count_x) \in R_x$ and $(i_y, count_y) \in R_y$)
   {
7. $count = \min(n_{req}, count_x + count_y)$
8. **if** $(count < n_{sense,min}(\{x\} \cup V_y \cup_{z \in DCH(x)} V_z))$ **continue**
9. $R'_x(i_x, count) += R_y(i_y, count_y) \times R_x(i_x, count_x) \times E_G(x[i_x], y[i_y])$
   }
10. set $R_x = R'_x$; remove y from T
    }
11. return $\sum_{s[i] \in \mathrm{Loc}(s)} R_s(i, n_{req}) * p_s(i)$

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.

Theorem

Function Conn solves the $\mathrm{SR\text{-}CONN}$ problem in $O(n \cdot n_{req}^2 \cdot \ell_{max}^2)$ time

Proof. We note the following.

- Step 1: storing the tree T require $O(n)$ time.
- Step 2: the main loop performs $n-1$ iterations. Each of Steps 3, 5, and 10 can be done in constant time.
- Step 4: this loop requires $O(n_{req} \cdot \ell_{max})$ time.
- Step 6: this loop requires $O(n_{req}^2 \cdot \ell_{max}^2)$ iterations. Steps 7, 8, and 9 can be done in constant time.

Thus, the overall running time is $O(n \cdot n_{req}^2 \cdot \ell_{max}^2)$ time.

Theorem

Function Conn solves the $\mathrm{AR\text{-}CONN}$ problem in $O(n \cdot \ell_{max}^2)$ time

A-CONN problem

- Cliques for A-CONN
- State types for A-CONN problem
- Algorithms for A-CONN problem
- simulation results for A-CONN problem

Cliques for A-CONN problem

- $K_{v_i,base}$: the k-clique to which node v_i is attached. For the 3-tree in the figure, $K_{v_i,base}$ is the triangle (3-clique) on nodes $\{v_j, v_k, v_l\}$.
- $K_{v_i,1}, K_{v_i,2}, \ldots, K_{v_i,k}$: all possible k-cliques involving node v_i when this node becomes a k-leaf. For the 3-tree in the figure, we may set $K_{v_i,1}$ = the triangle (v_i, v_j, v_k), $K_{v_i,2}$ = the triangle (v_i, v_j, v_l), $K_{v_i,3}$ = the triangle (v_i, v_k, v_l).

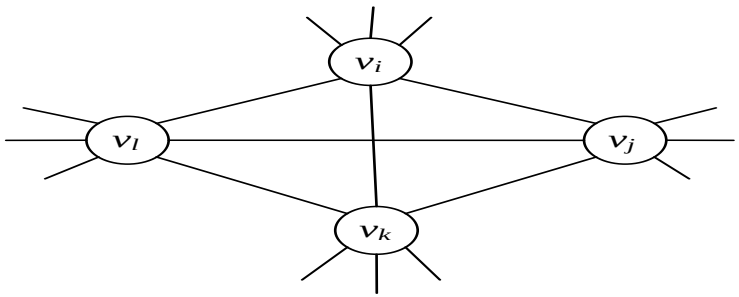


Figure: A fragment of a 3-tree

State types for A-CONN problem

$$A\text{-}type(S) = (V_1^{\mathrm{Loc}(V_1)}, V_2^{\mathrm{Loc}(V_2)}, \ldots, V_r^{\mathrm{Loc}(V_r)})$$

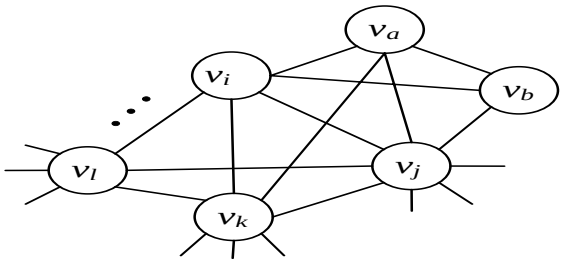


Figure: A 3-tree fragment

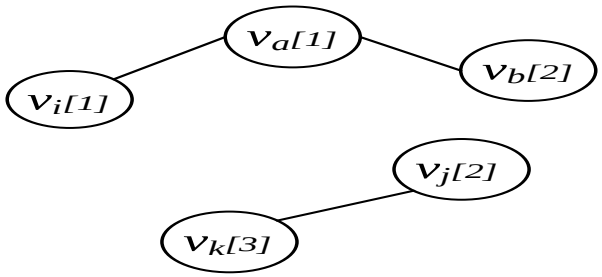


Figure: A state of A-CONN problem

Example

- $G_{v_i,1}$ induced on $\{v_a, v_b, v_i, v_j, v_k\}$.
- Assuming $S = \{v_a[1], v_b[2], v_i[1], v_j[2], v_k[3]\}$ is a possible network state of $G_{v_i,1}$
- Suppose that S has 2 connected components : $\{v_a, v_b, v_i\}$ and $\{v_j, v_k\}$
- Thus, $A\text{-}type(S) = (\{v_a, v_b, v_i\}^{(1,2,1)}, \{v_j, v_k\}^{(2,3)})$.

A sample Table

A table $T_{v_i,\alpha}$, where $\alpha = base, 1, 2, \ldots, k$ is key-value mapping where

- a key is a network state type of the graph $G_{v_i,\alpha}$
- each value is the probability.

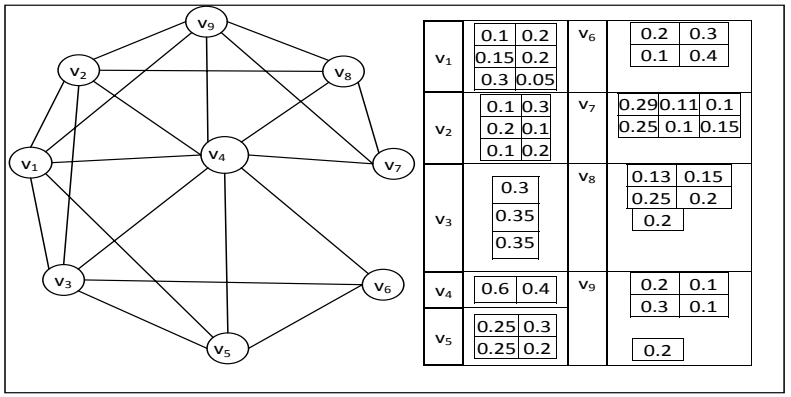


Figure: A 3-tree

Example

- The algorithm associates a table with each triangle.
- Each triangle can be partitioned in 5 different ways
- The number of possible keys that appear in the table of triangle (v_1, v_2, v_3) is $5 \times 6 \times 6 \times 3 = 648$ Why?

An Example Table

.	.
.	.
$\{v_1, v_2, v_3\}^{(1,1,1)}$	0.0008
$\{v_1, v_2\}^{(1,1)}\{v_3\}^{(1)}$	0.007
$\{v_1, v_3\}^{(1,1)}\{v_2\}^{(1)}$	0.006
$\{v_2, v_3\}^{(1,1)}\{v_1\}^{(1)}$	0.0001
$\{v_1\}^{(1)}\{v_2\}^{(1)}\{v_3\}^{(1)}$	0.0002
.	.
.	.
.	.

Main Idea

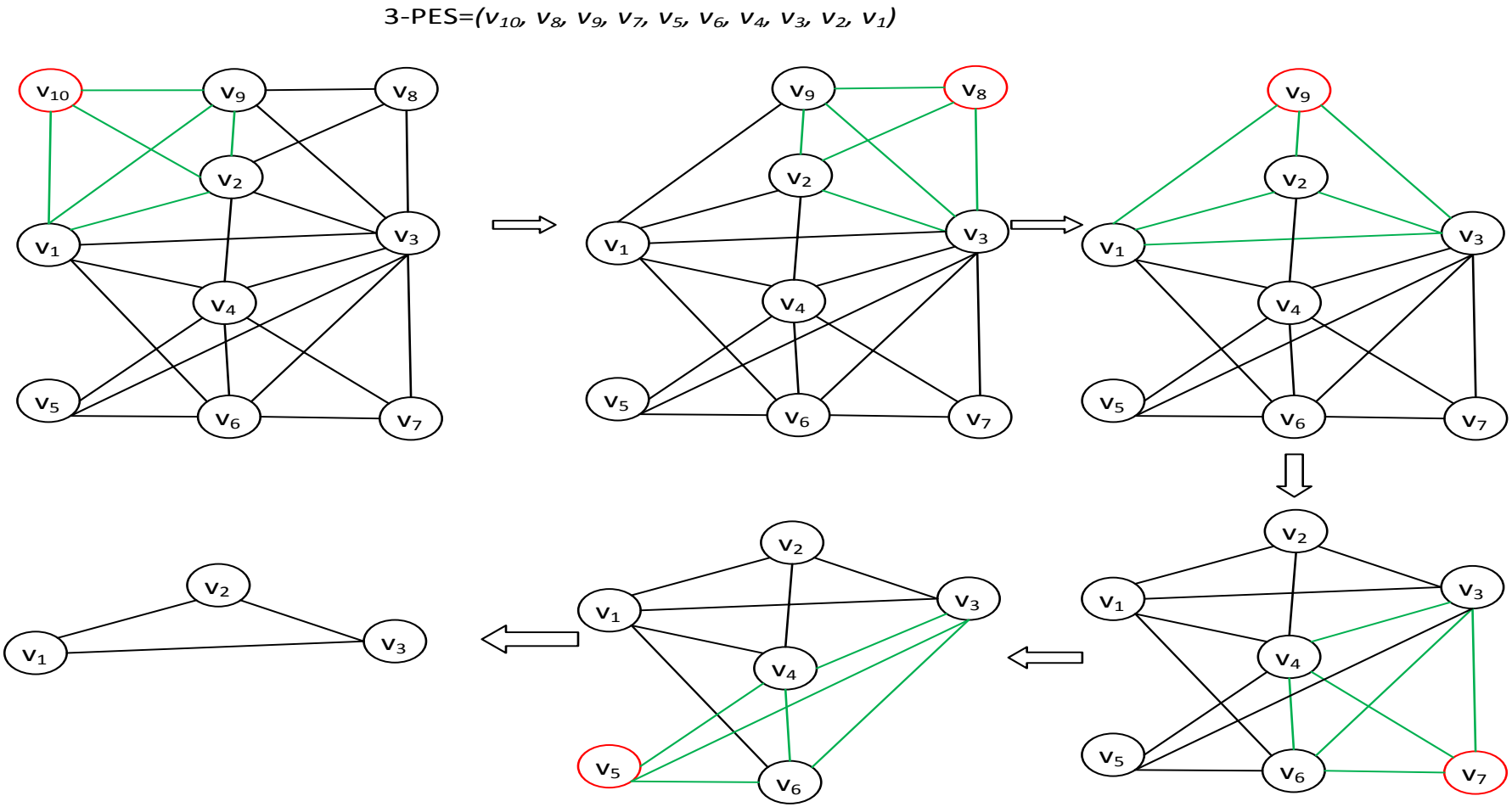


Figure: Mergeing Tables and Deleting nodes

Algorithm Organization

The overall algorithm is organized around 3 functions :

- a top level main function
  - process each node v_i according to PES
  - finds all the tables associates with v_i and merge them with the help of *t_merge*
  - update the base table $T_{v_i,base}$ and removes node v_i
- a middle level table merge function (*t_merge*), and
  - merge tables with the help of *p_merge*
  - updates the locality sets and probability
- a low level partition merge function (*p_merge*)

Table Merge and Partition merge

T_1 on (v_1, v_2, v_3)

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

$\times$

T_2 on (v_1, v_3, v_4)

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

$\Rightarrow$

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

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

Figure: Operation in Table Merge

$$\boxed{\{v_1, v_2\}\{v_3\}} \times \boxed{\{v_1\}\{v_3, v_4\}} \Rightarrow \boxed{\{v_1, v_2\}\{v_3, v_4\}}$$

Figure: Operation in Partition Merge

$T_{v_1,base}$ on (v_1, v_2, v_3, v_4)

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

$\Rightarrow$

$T_{v_1,base}$ on (v_2, v_3, v_4)

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

Figure: Operation in Main

Function Main

Algorithm 1: Function Main(G, PES)

Input: An instance of the $A\text{-}CONN$ problem where G has a partial k-tree topology with a given PES=$(v_1, v_2, \ldots, v_n)$

Output: $Conn(G)$

Notation: $Temp$ is a temporary table.

1 **Initialization:** initialize a table T_H for each k-clique H of 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 $Temp = t\_merge(Temp, T_{v_i,j})$
end
6 $T_{v_i,base} = t\_merge(Temp, T_{v_i,base})$
7 **foreach** $(key \in T_{v_i,base})$ **do**
8 **if** (v_i *is a singleton part of* key) **then**
9 delete key from $T_{v_i,base}$
end
else
10 delete v_i and its associated position from key
end
end
end
11 **return** $Conn(G) = \sum$ values in table $T_{v_{n-k},base}$ corresponding to state types that have exactly one connected component

Running Time

Theorem

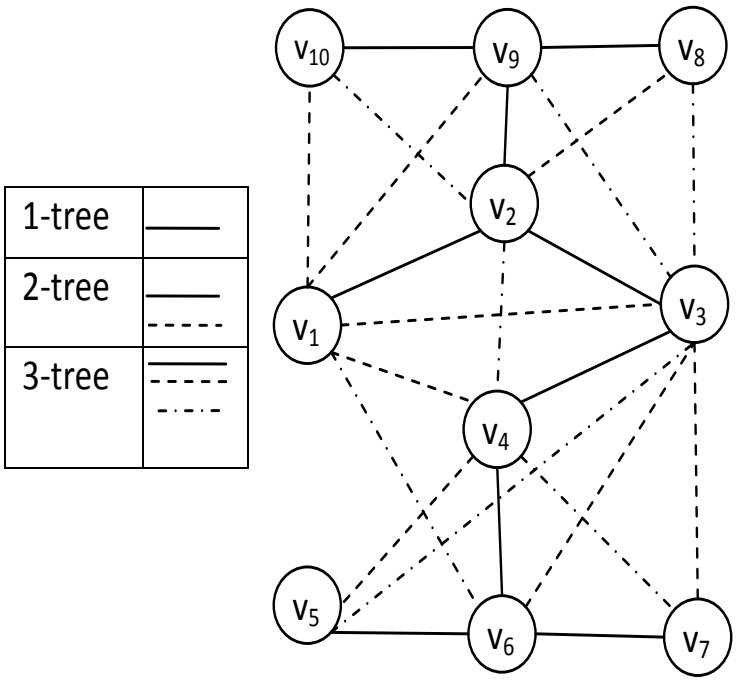
In the A-CONN algorithm we have

1. The maximum length of any table is $\in O(B_k l_{max}^k)$.
2. The worst case running time is $O(n(B_k l_{max}^k)^2)$.

Simulation Results for A-CONN problem

- Test Networks
- Running Time
- Connectivity for different partial k-trees
- Effect of subgraph selection method

Test Networks



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

Figure: G_{10}

Test Networks(Cont.)

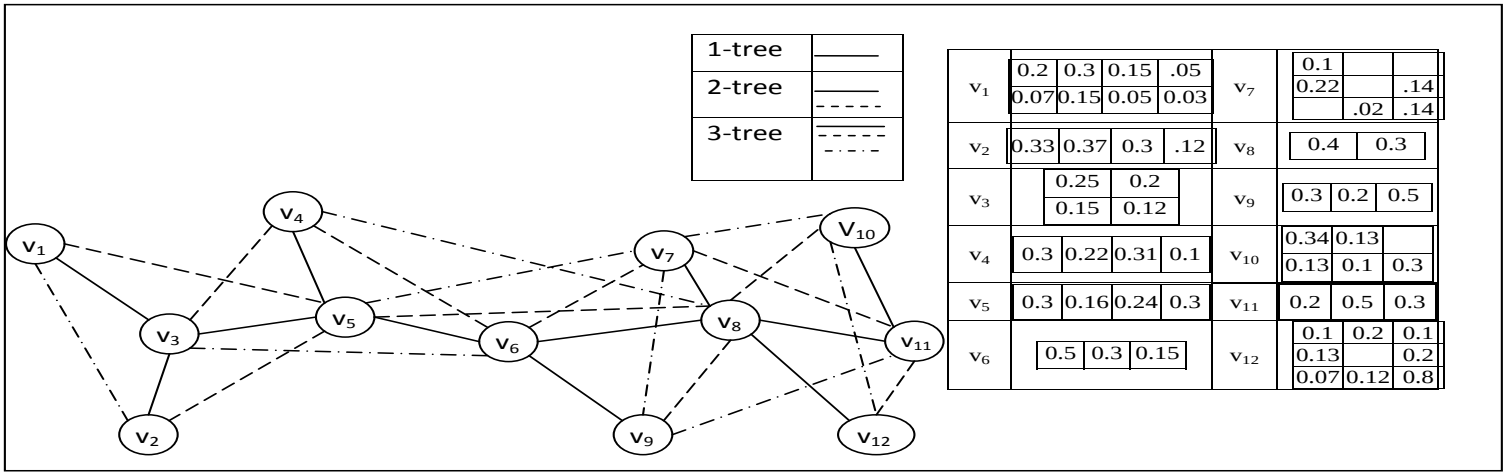


Figure: G_{12}

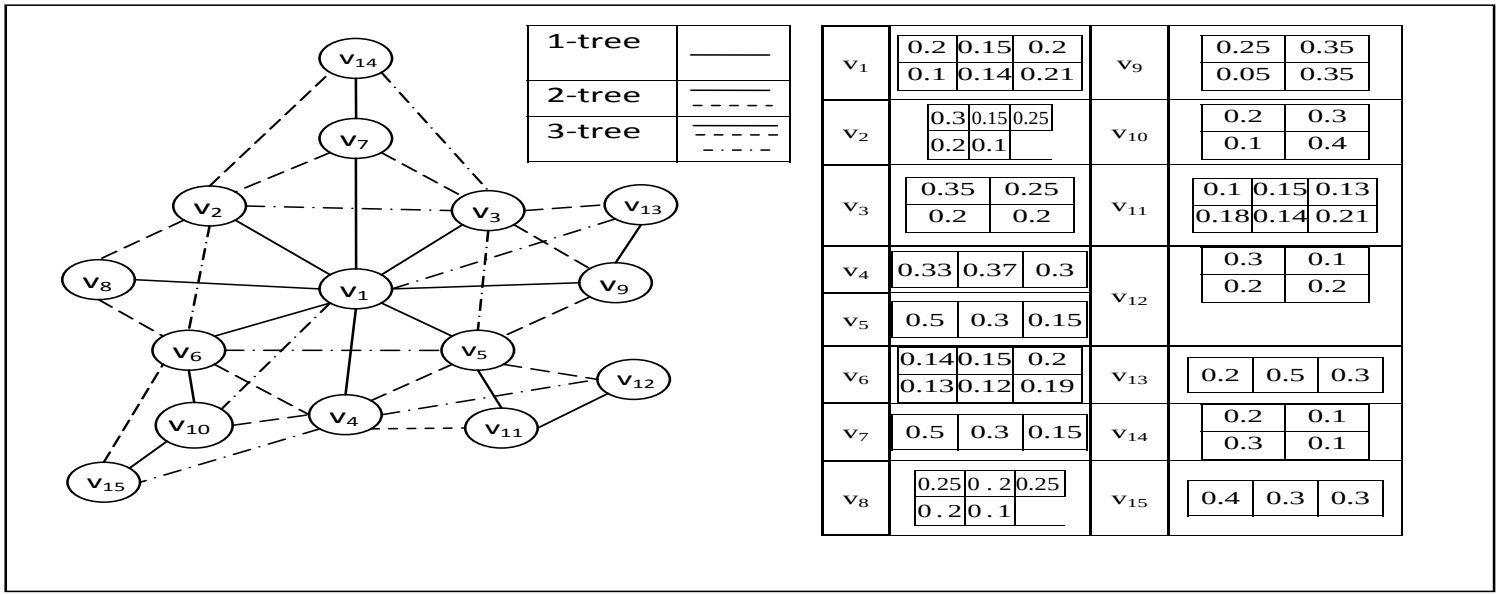


Figure: G_{15}

Running Time

k	Network G_{10}	Network G_{12}	Network G_{15}
1	90	130	200
2	1000	1380	1480
3	60000	875000	940000

Table: Running time in milliseconds

Connectivity for different partial k-trees

k	Network G_{10}	Network G_{12}	Network G_{15}
1	0.62	0.336	0.82
2	0.71	0.36	0.99
3	0.75	0.37	1

Table: Connectivity lower bounds using different partial k-trees

Effect of subgraph selection method

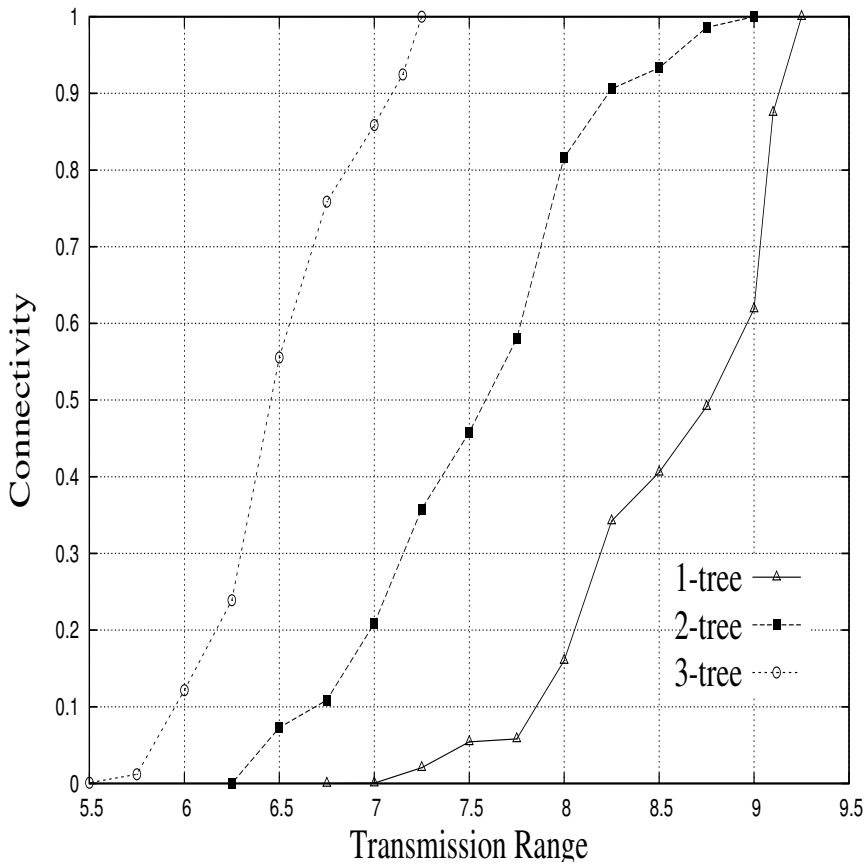


Figure: Connectivity versus transmission range with random subgraph selection

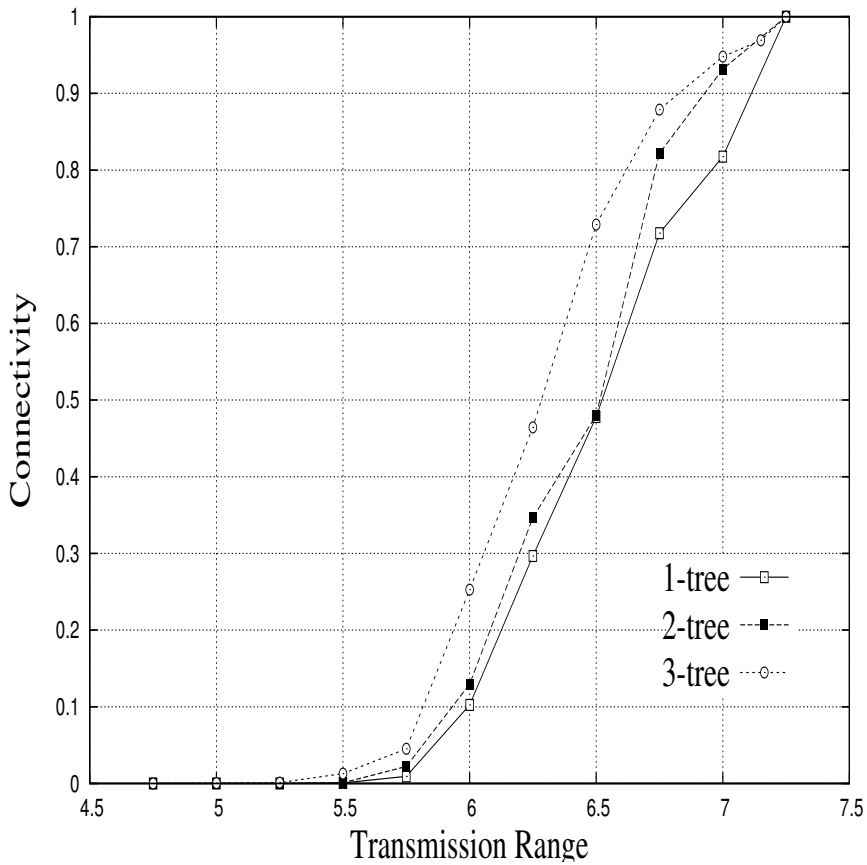


Figure: Connectivity versus transmission range with greedy subgraph selection

AR-CONN and SR-CONN problems

- AR-CONN State types
- merging AR-CONN state types
- removing bad states from AR-CONN state types
- Simulation Results for AR-CONN problem.
- SR-CONN State types
- merging SR-CONN state types
- removing bad states from SR-CONN state types
- Simulation Results for SR-CONN problem.

AR-CONN state types

$$AR\text{-}type(S) = \{V_{1,b(V_1)}^{Loc(V_1)}, V_{2,b(V_2)}^{Loc(V_2)}, \ldots, V_{r,b(V_r)}^{Loc(V_r)}\}$$

Example

In the figure, denote by $G_{v_i,1}$, the graph induced on nodes $\{v_a, v_b, v_i, v_j, v_k\}$. Assume that $\{v_b\} \in V_{sense}$ and $\{v_a, v_i, v_j, v_k\} \in V_{relay}$. Suppose that state S of the figure is a possible network state of $G_{v_i,1}$. Then $AR\text{-}type(S) = \{\{v_i\}_1^{(1)}, \{v_j, v_k\}_0^{(2,3)}\}$. Here, the indicator $b(\{v_i\}) = 1$ since v_b is a sensor node connected to v_i in S.

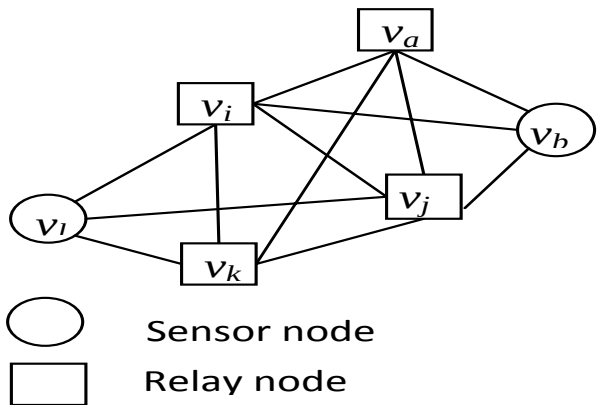


Figure: A fragment of a 3-tree

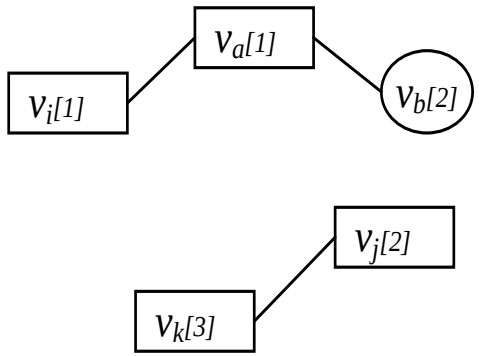


Figure: A state S on $\{v_a, v_b, v_i, v_j, v_k\}$

Merging State types

Table on (v_1, v_2, v_4, v_5)

.	.
.	.
$\{v_1, v_2\}_1^{(1,1)}\{v_4, v_5\}_0^{(2,4)}$	0.0008
.	.
.	.
.	.

$\times$

Table on (v_1, v_2, v_5, v_6)

.	.
.	.
$\{v_2, v_6\}_1^{(1,1)}\{v_1, v_5\}_1^{(1,4)}$	0.0006
.	.
.	.
.	.

$\Rightarrow$

Table on $(v_1, v_2, v_4, v_5, v_6)$

.	.
.	.
$\{v_1, v_2, v_4, v_5, v_6\}_1^{(1,1,2,4,1)}$	0.00062
.	.
.	.
.	.

Figure: Merging two tables

Removing Bad State Types

- For the AR-CONN problem, a state S of the subgraph $G_{v_i,base}$ reduced onto the k-clique $K_{v_i,base}$.
- S is considered bad if node v_i appears as a singleton part and the indicator $b(v_i) = 1$.
- The algorithm removes all such bad state types from $T_{v_i,base}$.

Running Time of AR-CONN

Theorem

In the AR-CONN algorithm we have

1. The maximum length of any table is $O(B_k l_{max}^k 2^k)$
2. The worst case running time is $O(n(B_k l_{max}^k 2^k)^2)$

Simulation Results

- Test Networks
- Running time
- Effects of Adding Relay nodes

Test Networks

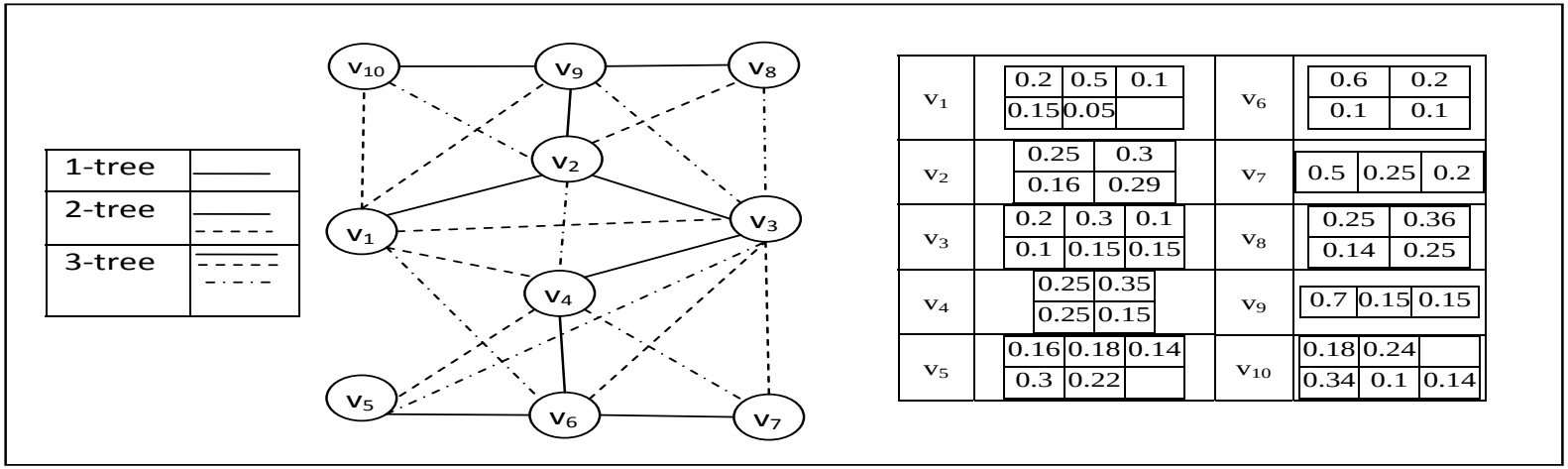


Figure: Network G_{10}

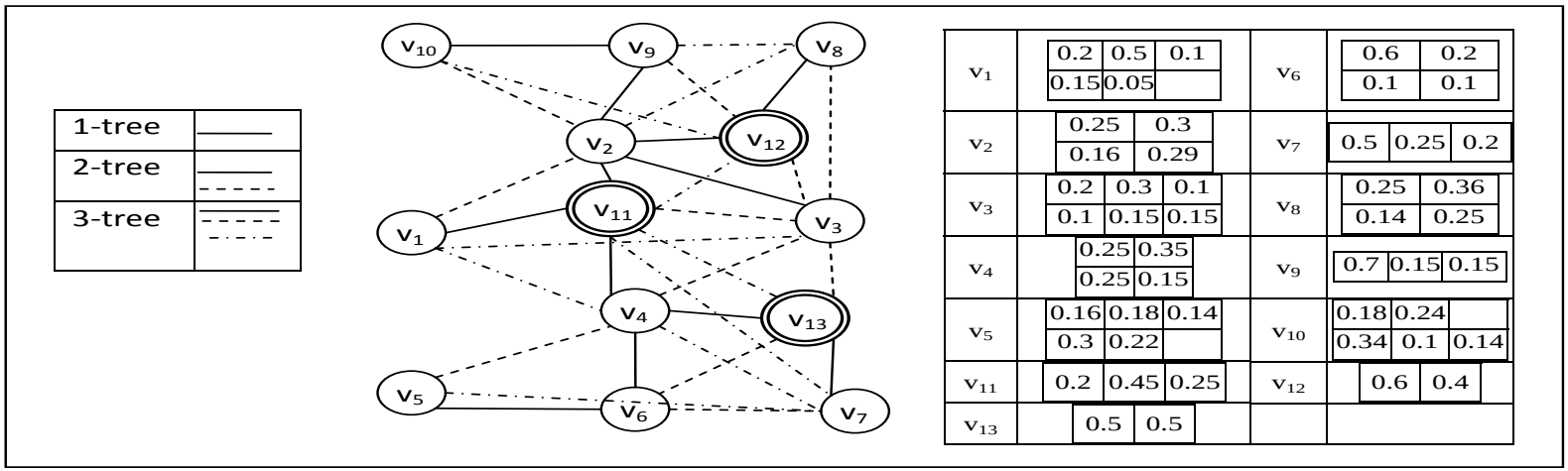


Figure: Network $G_{10,3}$

Running Time

k	Network G_{10}	Network $G_{10,3}$
1	90	110
2	1000	6000
3	6000	8000

Table: Running time in milliseconds

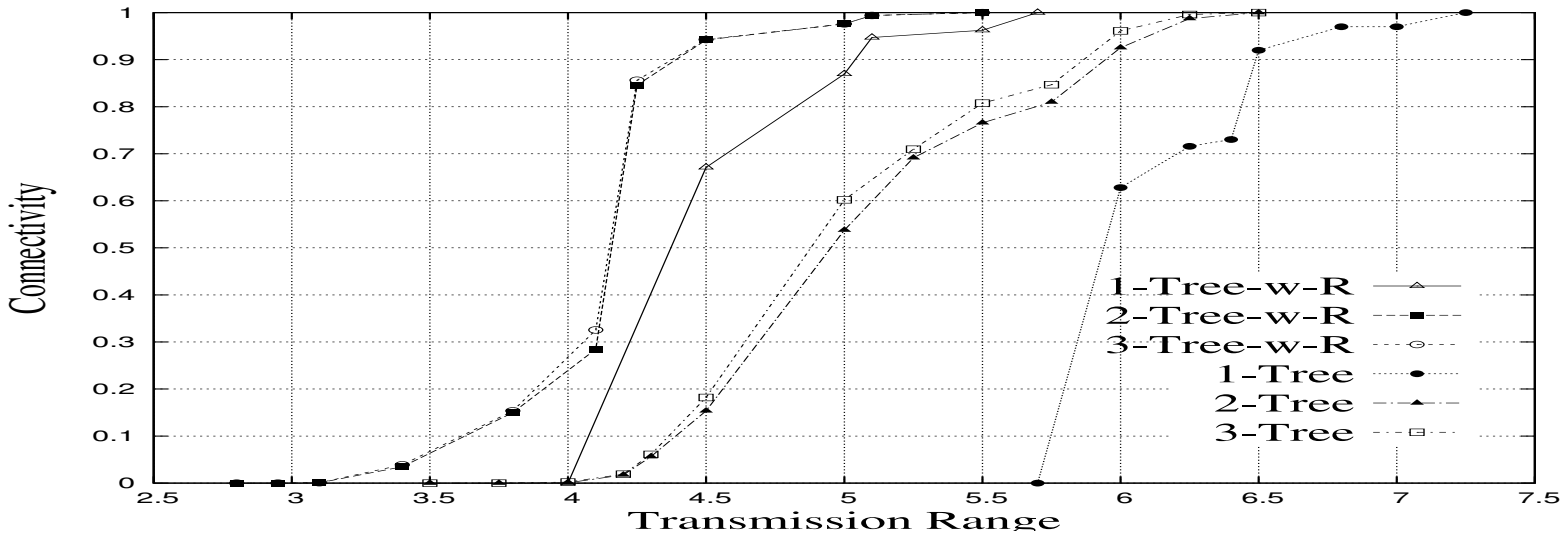
Effect of adding relay nodes

k	Network G_{10}	Network $G_{10,3}$
1	0.30	0.86
2	0.54	0.96
3	0.60	0.98

Table: Connectivity with respect to k

Effect of adding relay nodes for various node R_{tr}:

- Each obtained curve exhibits a notable monotonic increasing behaviour as R_{tr} increases.
- This behaviour is due to the appearance of more edges, and the potential increase in the probability of each edge as R_{tr} increases.
- Increasing R_{tr}, however, requires increasing node energy consumption.
- To achieve a desired $Conn(G)$ value, a designer may utilize the obtained curves to assess the merit of increasing R_{tr} versus deploying more relay nodes.



The SR-CONN State Types

$$SR\text{-}type(S) = \{V_{1,c(V_1)}^{Loc(V_1)}, V_{2,c(V_2}^{Loc(V_2)}, \ldots, V_{r,c(V_r}^{Loc(V_r)}\}$$

Example

In the figure, denote by $G_{v_i,1}$, the graph induced on nodes $\{v_a, v_b, v_i, v_j, v_k\}$. Assume that $\{v_b, v_i\} \in V_{sense}$ and $\{v_a, v_j, v_k\} \in V_{relay}$. Suppose that state S of the figure is a possible network state of $G_{v_i,1}$. Then $SR\text{-}type(S) = \{\{v_i\}_2^{(1)}, \{v_j, v_k\}_0^{(2,3)}\}$. Here, the count $c(\{v_i\}) = 2$ since both of v_i and v_b are sensor nodes.

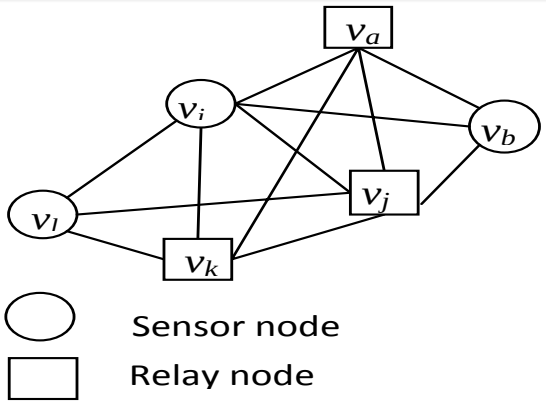


Figure: A 3-tree fragment

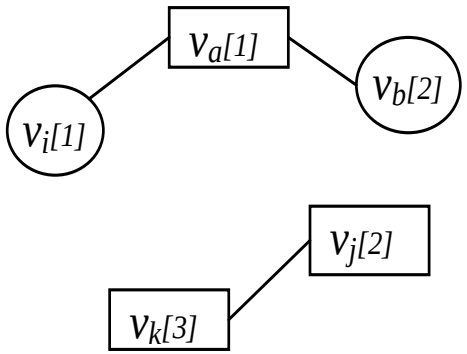


Figure: A state S on $\{v_a, v_b, v_i, v_j, v_k\}$

Merging State types

Table on (v_1, v_2, v_4, v_5)

.	.
.	.
$\{v_1, v_2\}_3^{(1,1)}\{v_4, v_5\}_0^{(2,4)}$	0.0008
.	.
.	.
.	.

$\times$

Table on (v_1, v_2, v_5, v_6)

.	.
.	.
$\{v_2, v_6\}_3^{(1,1)}\{v_1, v_5\}_2^{(1,4)}$	0.0006
.	.
.	.
.	.

$\Rightarrow$

Table on $(v_1, v_2, v_4, v_5, v_6)$

.	.
.	.
$\{v_1, v_2, v_4, v_5, v_6\}_6^{(1,1,2,4,1)}$	0.00062
.	.
.	.
.	.

Figure: Merging two tables

Bad State Types

- For the SR-CONN problem, a state S of the subgraph $G_{v_i,base}$ reduced onto the k-clique $K_{v_i,base}$.
- S is considered bad if node v_i appears as a singleton part and the indicator $c(v_i) \geq 1$.
- The algorithm removes all such bad state types from $T_{v_i,base}$.

Running Time SR-CONN

Theorem

In the SR-CONN algorithm we have

1. The maximum length of any table is $O(B_k l_{max}^k n_{req}^k)$
2. The worst case running time is $O(n(B_k l_{max}^k n_{req}^k)^2)$

SR-CONN Simulation Results

A designer has at least 3 options to achieve a minimum required $Conn(G, n_{req})$ value:

- tuning the n_{req} parameter,
- tuning node transmission range R_{tr}, and
- tuning the number of deployed relay nodes.

Effect of varying n_{req}

- Figure illustrates the achieved $Conn(G, n_{req})$ as n_{req} varies in the range $[1, 10]$.
- Figure illustrates the achieved $Conn(G, n_{req})$ as n_{req} varies in the range $[1, 10]$, and R_{tr} varies in the range $[2.5, 5.5]$.

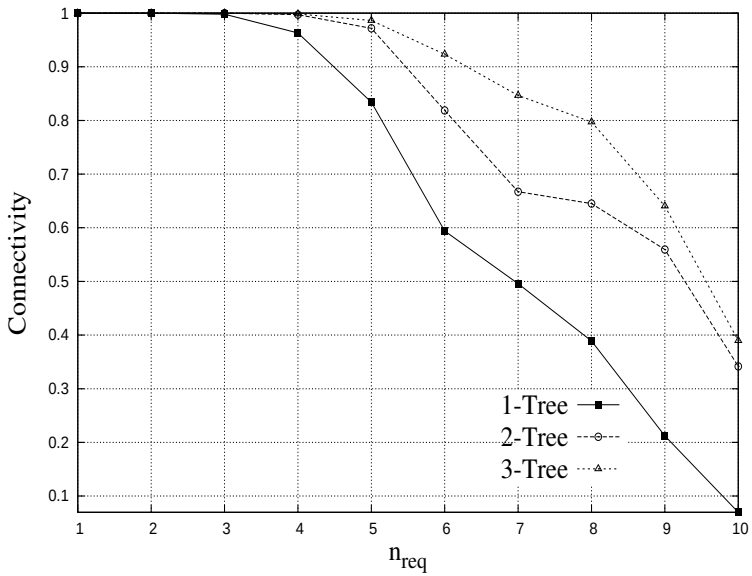


Figure: Connectivity versus n_{req}

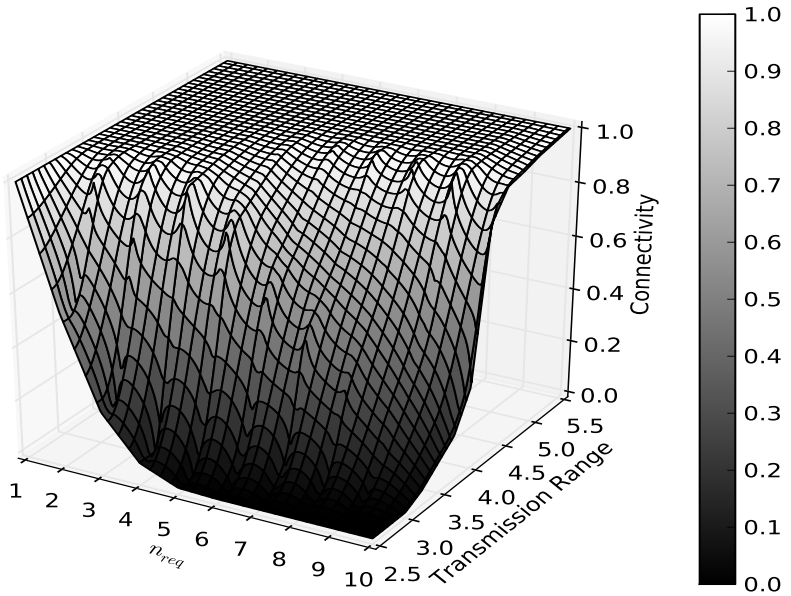


Figure: Connectivity versus n_{req} and transmission range

Concluding Remarks

- This thesis is motivated by recent interest in UWSNs as a platform for preforming many useful tasks.
- A challenge arises since sensor nodes incur small scale and large scale movements that can disrupt network connectivity.
- Thus, tools for quantifying the likelihood that a network remains completely or partially connected become of interest.
- the thesis has formalized 4 probabilistic connectivity problems, denoted $\mathrm{A\text{-}CONN}$, $\mathrm{S\text{-}CONN}$, $\mathrm{AR\text{-}CONN}$, and $\mathrm{SR\text{-}CONN}$.
- The obtained results show that all of the 4 problems admit polynomial time algorithms on k-trees (and their subgraph), for any fixed k.

Future Research Directions

- Investigating the applicability of our algorithm to some other classes of graph to be a worthwhile direction.
- It is interesting to analyze the delays incurred in typical data collection rounds.
- It is worthwhile to investigate area coverage assuming a probabilistic locality model of the nodes.

Thanks!