Macrothink Institute™

Network Protocols and Algorithms
ISSN 1943-3581
2015, Vol. 7, No. 3

A Generic Algorithm to Determine Maximum Bottleneck Node Weight-based Data Gathering Trees for Wireless Sensor Networks

Natarajan Meghanathan

Department of Computer Science, Jackson State University

1400 John R. Lynch Street, Jackson, MS 39217, USA

Tel: 1-601-979-3661 E-mail: natarajan.meghanathan@jsums.edu

Received: July 4, 2015 Accepted: November 1, 2015 Published: November 28, 2015

DOI: 10.5296/npa.v7i3.7961 URL: http://dx.doi.org/10.5296/npa.v7i3.7961

Abstract

We propose a generic algorithm to determine maximum bottleneck node weight-based data gathering (MaxBNW-DG) trees for wireless sensor networks (WSNs) and compare the performance of the MaxBNW-DG trees with those of maximum and minimum link weight-based data gathering trees (MaxLW-DG and MinLW-DG trees). Assuming each node in a WSN graph has a weight, the bottleneck weight for the path from a node *u* to the root node of the DG tree is the minimum of the node weights on the path (inclusive of the weights of the end nodes). The MaxBNW-DG tree algorithm determines a DG tree such that each node has a path of the largest bottleneck weight to the root node. We observe the MaxBNW-DG trees to incur lower height, larger percentage of nodes as leaf nodes and a larger weight per intermediate node compared to the leaf node; the tradeoff being a larger a network-wide data aggregation delay due to larger number of child nodes per intermediate node. The MaxBNW-DG algorithm could be used to determine DG trees with larger trust score, larger energy (and other such criterion for node weight) per intermediate node compared to the leaf node.

Keywords: Data Gathering Trees, Wireless Sensor Networks, Bottleneck Node Weight

1. Introduction

A Wireless Sensor Network (WSN) is a network of autonomous sensor nodes that can

typically sense/measure one or more environmental parameters of interest such as pressure, temperature, humidity, etc and forward the sensed data to a control center called the sink. The sensor nodes are battery charged and operate at a limited transmission range to conserve energy as well as to reduce interference occurring due to simultaneous transmissions across the shared wireless medium. It would consume a significant amount of energy in the network if each of the sensor nodes attempt to directly transmit their data to the sink, leading to premature node failures. Hence, sensor nodes typically forward their data along a specialized communication topology (like cluster [1], grid [2-3], chain [4], tree [5], connected dominating set [6], etc) and data (from the various nodes) gets aggregated as it propagates through this topology. Several articles have been published (e.g., [5]) comparing the performance of these different communication topologies for data gathering. Most of the work available in the literature focus on determining communication topologies that minimize the energy consumption at the sensor nodes and prolong the network lifetime (typically, defined as the time the network gets disconnected due to the failure of one or more nodes that run out of energy). In [5] and other related works (e.g., [7-8]), data gathering trees (DG trees) have been observed to be the most energy-efficient among these topologies, as the links of the DG trees typically span over a shorter distance and the number of links involved in the transmission and reception of data are to the bare minimum, but still span across all the nodes.

The algorithms proposed until now for DG trees focus on using a specific attribute such as the residual (available) energy of a node [9], link distance [10], link expiration time [10], node degree [11], etc as the underlying criterion to determine the trees and thereby evaluate the effectiveness of these criterion to minimize energy consumption and maximize network lifetime. Though a slew of such algorithms (based on a specific node or link selection criterion) are available in the literature, we could not come across a generic algorithm that could determine a maximum bottleneck node weight-based DG tree (MaxBNW-DG tree). Assuming the WSN is modeled as a weighted graph (each node has a weight), we define the bottleneck weight for a path from any node to the root node in a DG tree as the minimum of the node weights on the path (including the weights of the end nodes of the path). The MaxBNW-DG tree is a DG tree in which every node is located on a path of the largest bottleneck weight to the root node.

To the best of our knowledge, we have not come across in the literature a generic algorithm that can determine a maximum bottleneck node weight-based DG tree for a WSN graph with node weights. Given that the root node of a MaxBNW-DG tree is the node with the largest weight, we hypothesize a MaxBNW-DG tree to connect every other node on the shortest path to the root node (due to the need to be connected to the root node on a path of larger bottleneck node weight, the fewer the intermediate nodes on the path, the larger the chances of finding a path with a larger bottleneck node weight value) as well as distribute the nodes in such a way that the intermediate nodes are more likely to be of a larger weight compared to that of the leaf nodes, resulting in the average weight per intermediate node to be greater than the average weight per leaf node. Also, we hypothesize a MaxBNW-DG tree to have fewer intermediate nodes (due to the need for connecting all the nodes on paths of

largest bottleneck weight to the root node) and incur a significantly larger number of leaf nodes (and thereby contribute towards reduced overall energy consumption).

A generic algorithm to determine MaxBNW-DG trees could have several applications. For example, as shown in this paper, if each node in a WSN has a trust score (such that larger the trust score for a node, the more reliable is the node), the MaxBNW-DG tree algorithm could be used to construct a DG tree such that the intermediate nodes of the tree are more likely to have a larger trust score than the leaf nodes of the tree. This way, an intermediate node could simply discard data emanating from leaf nodes that have a lower trust score and there will not be any appreciable loss of information as part of network-wide data aggregation (on the other hand, it would lead to appreciable loss of information if aggregated data originating from an intermediate node with a lower trust score is discarded by a node higher up in the DG tree). The only way a node u with a lower trust score could become an intermediate node is when none of its neighbor nodes are in the neighborhood of nodes that have a path of relatively larger bottleneck weight to the root node compared to the bottleneck weight of the path from node u to the root. Likewise, the MaxBNW-DG tree algorithm could be used to determine energy-aware data gathering trees such that the average residual energy per intermediate node is larger than the average residual energy per leaf node.

Our contributions in this paper are as follows: We propose a generic algorithm to determine maximum bottleneck node weight-based data gathering trees in which a node u serves as an intermediate node only if none of its adjacent nodes have neighbor nodes of relatively larger bottleneck weight path to the root node compared to the bottleneck weight of the path from node u to the root. As case studies, we show that the proposed algorithm could be applied to determine MaxBNW-DG trees for network graphs in which the node weights are related to the link weights and the degree of the nodes as well as when the node weights are not related to the link weights and node degree. The results of our simulation studies show that in the case of MaxBNW-DG trees, the average weight per intermediate node is always greater than the average weight per leaf node, irrespective of the number of intermediate nodes and leaf nodes in the trees. We observe that the average height of a MaxBNW-DG tree is far lower than that of a MaxLW-DG and MinLW-DG trees, whereas the delay for data aggregation incurred with a MaxBNW-DG tree is much larger than that of the link weight-based DG trees. Due to lower height, we also observe the MaxBNW-DG trees to involve the majority of the nodes as leaf nodes such that the percentage of nodes as leaf nodes in a MaxBNW-DG tree could be as large as twice the percentage of nodes as leaf nodes in a MaxLW-DG tree or a MinLW-DG tree.

The rest of the paper is organized as follows: Section 2 presents the maximum bottleneck node weight-based data gathering tree algorithm, illustrates its execution with an example, evaluates its run-time complexity as well as demonstrates its proof of correctness. In Section 3, we present algorithms to determine the maximum and minimum link weight-based data gathering trees and illustrate their execution with examples. Section 4 presents an algorithm used to compute the height and aggregation delay of a DG tree. Section 5 presents the results from two sets of simulation studies: the first set of studies compare the performance of the MaxBNW-DG trees and MaxLW-DG trees wherein the node weight is simply the sum of the

weights of the links, assigned randomly to each link; the second set of studies compare the performance of the MaxBNW-DG and MinLW-DG trees wherein the node weights and link weights are not related (node weight is the energy of a node, a randomly assigned value, and the link weight is the square of the Euclidean distance between the end nodes of the link). Section 6 discusses about modifying the proposed generic algorithm to determine minimum bottleneck node-weight based data gathering trees (MinBNW-DG trees) wherein the bottleneck weight of a path is defined as the maximum of the node weights and the focus would be to determine a DG tree for which the path from any node to the root node has the minimum bottleneck weight. Section 7 presents related work and discusses how the proposed generic algorithms for maximum and minimum bottleneck node weight-based data gathering trees could be applied to determine data gathering trees that have a larger node weight for the intermediate nodes (maximum bottleneck node weight-based DG trees) or lower node weight for the intermediate nodes (minimum bottleneck node weight-based DG trees) based on various criteria for node weights considered in the literature. Section 8 concludes the paper. Throughout the paper, we use the terms 'data gathering' and 'data aggregation' as well as the terms 'edge' and 'link, 'node' and 'vertex' interchangeably. They mean the same.

2. Generic Algorithm to Determine Maximum Bottleneck Node Weight-based Data Gathering Trees

2.1 System Model

The wireless sensor network (WSN) is assumed to be modeled as a weighted graph $G = (V, E, W_V, W_E)$, where V is the set of vertices that are connected by a set E of edges and W_V and W_E represent the set of weights (positive real numbers) for the vertices and edges respectively (one weight per vertex and one weight per edge). The node weights may or may not be related to the link weights. If two or more nodes (likewise, two or more links) have the same weight, we arbitrarily break the ties. Each node in the WSN is represented by a vertex in the graph and the transmission range of all nodes is identical to each other. We assume that two vertices are connected with an edge if the Euclidean distance between the corresponding nodes in the network is less than or equal to the transmission range of the nodes. A data gathering (DG) tree is a rooted spanning tree (the root node is called the leader node) and data aggregation proceeds from the leaf node through the intermediate nodes all the way to the leader node. An intermediate node in the DG tree has to wait to receive aggregated data (or individual data if the child node is a leaf node) from each of its immediate child nodes, aggregates all the data with its own data and forwards the final aggregated data to its own upstream node in the DG tree. We assume the sensor nodes to operate with sufficient energy throughout the data aggregation session so that there are no node failures due to energy exhaustion and the best possible performance could be extracted from the benchmarking algorithms with respect to the metrics that they are designed to optimize. With regards to channel access, we assume the sensor nodes to be both CDMA (Code Division Multiple Access) and TDMA (Time Division Multiple Access)-enabled [12-13]. An intermediate node assigns unique TDMA time slots to each of its immediate child nodes so that it receives

aggregated data from at most one child node during a time slot. There could be simultaneous data aggregation using different CDMA codes at two intermediate nodes that are at the same level or different levels of the DG tree. Two intermediate nodes at the same level or different levels of the DG tree could simultaneously aggregate data from their respective child nodes using different CDMA codes.

2.2 Description of the MaxBNW-DG Algorithm

We define the bottleneck weight of a path as the minimum of the node weights along the path (including the weights of the end nodes). The MaxBNW-DG algorithm (pseudo code in Figure 1) is based on the hypothesis that by accommodating nodes with larger weights as intermediate nodes, one could determine a DG tree wherein each node could join the tree through an upstream node that has the largest bottleneck weight path to the root node. For each node, we refer to the bottleneck weight of the path connecting the node to the root node of the DG tree as the *Tree-join-weight* of the node. The algorithm maintains three lists as part of its execution: *Candidate-Nodes-List*, *Optimized-Nodes-List* and *Parent-Node-List*. The *Candidate-Nodes-List* is maintained as a maximum heap (priority queue) [14] and it contains the *Estimated-join-weights* of the nodes that are yet to be included in the DG tree. The *Estimated-join-weight* of a node is an estimate of the largest bottleneck weight path of the node to the root node. The *Parent-Node-List* maintains the predecessor node (immediate upstream node) for each node in the DG tree and is updated whenever a node has found an upstream node through which it can join the tree on a path whose bottleneck weight is larger than the currently know bottleneck weight of the path to the root node.

We designate the node with the largest individual weight as the leader node and include it in the *Candidate-Nodes-List* and set its *Estimated-join-weight* to the weight of the node itself. The *Estimated-join-weight* of every other node is initialized to -1 and the algorithm seeks to improve it in each iteration as much as possible. In each iteration, the algorithm removes the node with the largest *Estimated-join-weight* from the *Candidate-Nodes-List* and includes it to the *Optimized-Nodes-List* (i.e., one node is optimized per iteration); the *Tree-join-weight* of the node is set to the value of the *Estimated-join-weight* at the time of its removal from the *Candidate-Nodes-List* and inclusion to the *Optimized-Nodes-List*. Further, we explore the neighborhood of the optimized node (say, node u) and see if it could become the predecessor node for any of its neighbors (say, node v) that are yet to be optimized. If the *Estimated-join-weight* of a neighbor node *v* could be improved, we set it to the minimum of the *Estimated-join-weight* of node *u* and the individual weight of node *v*, set node *u* as the predecessor node for node *v* as well as include node *v* in the *Candidate-Nodes-List* (if node *v* is not yet in the *Candidate-Nodes-List*). We run the iterations until the *Candidate-Nodes-List* gets empty. If all the nodes are optimized by then, it means the algorithm has determined a DG tree spanning all the nodes and returns the set of edges connecting a node with its predecessor node. If there are one or more nodes that are yet to be optimized and the *Candidate-Nodes-List* has already become empty, it means the underlying network graph is not connected and the algorithm returns NULL.

Input: Graph $G = (V, E, W_V, W_E)$

Output: $T_{Max}^{BNW}(V, E_T)$ where E_T is the set of edges in the MaxBNW-DG tree

Auxiliary Variables: *Optimized-Nodes-List*, *Candidate-Nodes-List*, *Parent-Node-List*
Estimated-join-weight, *Tree-join-weight*, *Leader Node*

Initialization: *Optimized-Nodes-List* = ϕ; *Candidate-Nodes-List* = ϕ, *Parent-Node-List* = ϕ, E_T = ϕ
$\forall u \in V$: *Estimated-join-weight*(*u*) = -1, *Tree-join-weight*(*u*) = -1

Begin *MaxBNW-DG Algorithm*

1 *Leader Node s* = {*u* | $Maximum_{u \in V}(W_V(u))$ }
2 *Parent-Node-List*(*s*) = NULL
3 *Candidate-Nodes-List* = *Candidate-Nodes-List* ∪ {*s*}
4 *Estimated-join-weight*(*s*) = W_V(*s*)
5 ***while*** (*Candidate-Nodes-List* ≠ ϕ) ***do***
6 Node *u* = {*i* | $Maximum_{i \in Candidate-Nodes-List}$ (*Estimated-join-weights*(*i*)}
7 *Candidate-Nodes-List* = *Candidate-Nodes-List* - {*u*}
8 *Optimized-Nodes-List* = *Optimized-Nodes-List* ∪ {*u*}
9 *Tree-join-weight*(*u*) = *Estimated-join-weight*(*u*)
10 ***for*** (Node *v* ∈ *Neighbors*(*u*) AND *v* ∉ *Optimized-Nodes-List*) ***do***
11 ***if*** (*Estimated-join-weight*(*v*) < ***Minimum***(W_V(*v*), *Tree-join-weight*(*u*))) ***then***
12 *Estimated-join-weight*(*v*) = ***Minimum***(W_V(*v*), *Tree-join-weight*(*u*))
13 *Candidate-Nodes-List* = *Candidate-Nodes-List* ∪ {*v*}
14 *Parent-Node-List*(*v*) = *u*
15 ***end if***
16 ***end for***
17 ***end while***
18 ***if*** (*Candidate-Nodes-List* = ϕ AND | *Optimized-Nodes-List* | < | *V* |) ***then***
19 ***return*** NULL; // the graph is not connected
20 ***end if***
21 ***if*** (*Candidate-Nodes-List* = ϕ AND | *Optimized-Nodes-List* | = | *V* |) ***then***
22 ***for*** (Node *u* ∈ *V* AND *u* ≠ *Leader Node*) ***do***
23 $E_T = E_T \cup \{(u, Parent\text{-}Node\text{-}List(u))\}$
24 ***end for***
25 ***end if***
26 ***return*** $T_{Max}^{BNW}(V, E_T)$

End *MaxBNW-DG Algorithm*

Figure 1: Pseudo Code for the Algorithm to Determine Maximum Bottleneck Node Weight-DG Trees

2.3 Time-Complexity of the MaxBNW-DG Algorithm

It takes $\Theta(|V|)$ time to determine the leader node among the nodes in the network (on the basis of the individual node weights), where V is the set of all the vertices in the WSN graph and the initialization to the auxiliary variables can be done in constant time. Hence, lines 1-4 take $\Theta(|V|)$ time. Since we implement the *Candidate-Nodes-List* as a maximum heap [14], it takes $\Theta(\log|V|)$ time to remove the root of the heap (i.e., the root is the node with the largest *Estimated-join-weight* among the nodes in the *Candidate-Nodes-List*). Lines 10-15 explore the neighborhood of a particular vertex in each iteration; overall, this amounts to visiting the edges of the graph twice - once for each of the end vertices of an edge to examine whether their *Estimated-join-weight* could be increased; if the *Estimated-join-weight* for a node gets changed, then the *Candidate-Nodes-List* (maximum heap) has to be re-heapified to restore the heap property. It takes $\Theta(\log|V|)$ to re-heapify a heap once [14]; lines 10-15 are executed a total of $2*|E|$ times; hence, the time-complexity for lines 10-15 is $\Theta(|E|*\log|V|)$. It takes $\Theta(|V|)$ time (lines 21-23) to go through the *Parent-Node-List* for all the nodes and come up with the set of edges (E_T) for the MaxBNW-DG tree. Considering the above-said time complexities involved in different stages of the algorithm, we could say the overall time complexity of the MaxBNW-DG algorithm is: $\Theta(|V|)$ for lines 1-4 + $\Theta(|E|*\log|V|)$ for lines 10-15 + $\Theta(|V|)$ for lines 21-23 = $\Theta(|V| + |E|*\log|V|)$.

2.4 Proof of Correctness

Lemma 1: The *Tree-join-weight* of any vertex v cannot be more than its individual weight, $W_V(v)$.

Proof: Consider a vertex v that is not yet included in the *Optimized-Nodes-List*. During each of the iterations in which vertex v is a neighbor node of a vertex u being optimized (in lines 11-12 of the MaxBNW-DG algorithm), we explore whether the *Estimated-join-weight* of the vertex v can be improved by comparing it with the ***minimum*** of the *Tree-join-weight* of the neighbor node u being optimized for that iteration and the individual weight of node v, $W_V(v)$. In each such iterations, $W_V(v)$ is one of the two arguments used to evaluate the minimum. Hence, the *Estimated-join-weight* of vertex v cannot be above its individual weight $W_V(v)$. With that said, at the time of its removal from the *Candidate-Nodes-List* and inclusion to the *Optimized-Nodes-List*, the *Tree-join-weight* of vertex v cannot be more than its individual weight, $W_V(v)$.

Lemma 2: The *Tree-join-weight* of the vertices added to the *Optimized-Nodes-List* are in the non-increasing order.

Proof: Consider two vertices u and u' such that u gets added to the *Optimized-Nodes-List* during some iteration(s) before the inclusion of node u' to the *Optimized-Nodes-List*. As per this lemma, *Tree-join*-weight(u') $\leq$ *Tree-join-weight*(u). We prove this by contradiction. Let *Tree-join*-weight(u')>*Tree-join-weight*(u), but still vertex u gets added to the *Optimized-Nodes-List* ahead of vertex u'. In other words, at the time when node u was optimized, its *Tree-join-weight* should have been greater than that of node u'; but, after the optimization of node u, the *Estimated-join-weight* of node u' should have increased beyond

the *Tree-join-weight* of node *u*. This would be possible only if one or more descendants of node *u* or nodes optimized after node *u* had a *Tree-join-weight* larger than that of node *u*. Since we only take the minimum of the *Tree-join-weight* of a parent node and the individual weight of the child node while trying to update the *Estimated-join-weight* of a child node (lines 11-12 in Figure 1) , the *Tree-join-weight* of a child node can be only at most the *Tree-join-weight* of the parent node. Hence, it would not be possible for the descendants of node *u* to cause the *Estimated-join-weight* of node *u*' to increase beyond the *Tree-join-weight* of node *u*. The other possibility is that there existed a node (say, node *x*) that was optimized after node *u* and the *Tree-join-weight* of node *x* contributed to the increase in the *Estimated-node-weight* of node *u*'. Nevertheless, given that node *x* was optimized after node *u*, the *Tree-join-weight* of node *x* can be only at most the *Tree-join-weight* of node *u* (otherwise, node *x* would have been optimized before node *u*) and also the *Tree-join-weight*s of the descendants of node *x* on a path leading to node *u*' can be also only at most the *Tree-join-weight* of node *x*, which could in turn be at most the *Tree-join-weight* of node *u*. Thus, if node *u*' were to be optimized some iteration(s) after node *u*, the *Tree-join-weight*(*u*') ≤ *Tree-join-weight*(*u*) and not the other way around.

Theorem: For each node, there exists a path of the largest bottleneck weight to the root node in the MaxBNW-DG tree.

Proof: We prove by contradiction. Assume that there exists a node *v* whose bottleneck weight for the path P to the root node (determined by the MaxBNW-DG algorithm) is less than the bottleneck weight for a different path P' (from node *v* to the root node) that exists in the graph, but not found by the MaxBNW-DG algorithm. We will now explore whether such a hypothetical path P' can exist. There should be at least one intermediate node (say, node *z*) on the path P' that was not yet optimized by the MaxBNW-DG algorithm at the time of optimizing the parent node *u* of *v* on the path P; because, if all the intermediate nodes on the path P' were optimized, then we would have identified the path P' as part of the neighborhood exploration steps of the MaxBNW-DG algorithm and chose P' over P (if P' had a larger bottleneck weight than P). From Lemma 2, the prospective *Tree-join-weights* of one or more intermediate nodes (like node *z*) that are not yet optimized (at the time of optimizing node *u*) and located in path P' had to be less than or equal to the *Tree-join-weight* of node *u*. Hence, even if the intermediate node *z* of lower *Estimated-join-weight* was optimized after the optimization of node *u*, the *Tree-join-weight* of node *z* would be less than or equal to the *Tree-join-weight* of node *u* and consequently, the *Tree-join-weights* of the downstream nodes of node *z* leading to node *v* on the path P' could only get even smaller than the *Tree-join-weight* of node *u*, resulting in the bottleneck weight of path P' to be less than or equal to the bottleneck weight of path P. This is a contradiction to our assumption. Thus, the paths found by the MaxBNW-DG algorithm from each of the vertices to the root node are the ones with the largest bottleneck weights.

2.5 Example

Figure 2 presents an example illustrating the execution of the MaxBNW-DG algorithm. The link weights are randomly generated; the individual weight of a node is the sum of the

weights of its adjacent links and is shown inside the node circles that also have the node's ID. The *Estimated-join-weights* of the nodes are shown outside the node circles. As node 2 has the largest individual node weight, it is the root node (leader node) of the MaxBNW-DG tree and its *Tree-join-weight* is set to its node weight itself. The *Estimated-join-weights* of the other vertices are set to -1. We show the sequence of iterations (a total of 8 iterations on a graph of 8 vertices), with a node optimized per iteration, and the resulting changes in the *Estimated-join-weights* and *Tree-join-weights* of the vertices. In each iteration, we optimize the vertex with the largest *Estimated-join-weight* among the vertices that are yet to be optimized and set its *Tree-join-weight* to the *Estimated-join-weight* as well as explore whether the *Estimated-join-weights* of its neighbors (that are not yet optimized) could be improved.

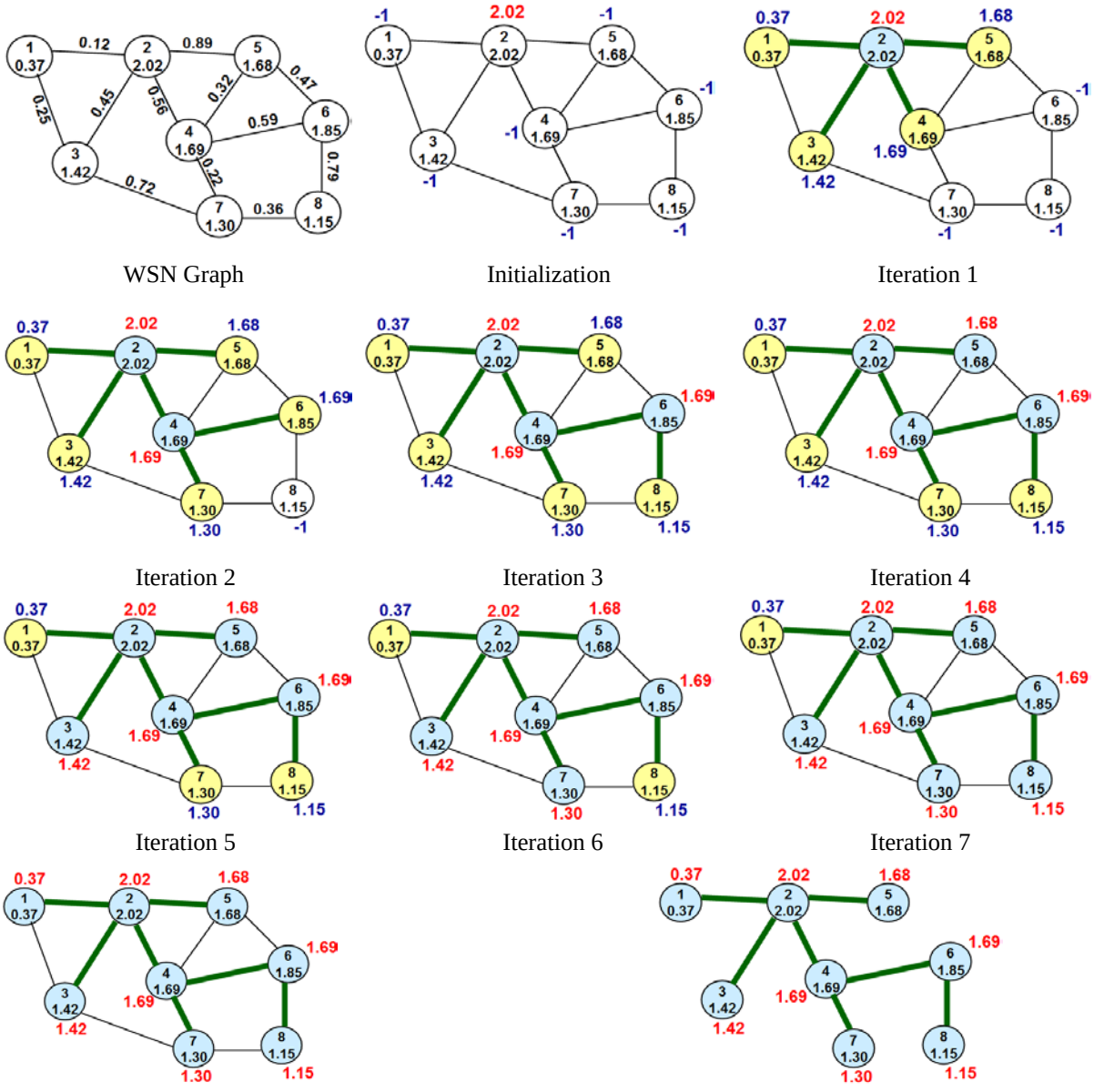


Figure 2: Example to Illustrate the Execution of the Algorithm to Determine Maximum Bottleneck Node Weight-based Data Gathering Tree

We can see that lemmas 1 and 2 as well as the theorem proving the correctness of the MaxBNW-DG algorithm (all explained in Section 2.4) stand vindicated in the example. The *Tree-join-weight* of any node is at most the individual weight of the node (Lemma 1); the *Tree-join-weights* of the sequence of vertices being optimized are in the non-decreasing order (2.02, 1.69, 1.69, 1.68, ..., 0.37; Lemma 2). Comparing the original graph with the final MaxBNW-DG tree, we could also confirm that each node is connected to the root/leader node (node 2) on a path of the largest bottleneck weight (including the weights of the end nodes of the path); there exists no other paths (from the individual nodes to the root node) of bottleneck weight larger than those determined by the algorithm (Theorem).

3. Algorithms to Determine Maximum and Minimum Link Weight-based Data Gathering Trees

In this section, we present the algorithms that could be used to determine maximum and minimum link-weight-based data gathering trees. A maximum link weight-based DG tree (MaxLW-DG tree) is the one in which the sum of the weights of the links is the maximum. In pursuit of an optimal solution, the MaxLW-DG tree algorithm prefers to include links of relatively larger weight as part of the DG tree. Hence, the MaxLW-DG tree algorithm could be used for applications in which links of larger weight are preferred for inclusion in a DG tree (for example, the algorithm could be used to construct DG trees whose constituent links have a larger trust score). A minimum link-weight based DG tree (MinLW-DG tree) is the one in which the sum of the link weights is the minimum. MinLW-DG trees tend to include links of relatively lower weight; applications of the MinLW-DG trees could be to determine DG trees whose constituent links undergo reduced energy loss during transmission such that the overall energy loss due to transmission across all the links is minimum.

Figure 3 presents the pseudo code of the MaxLW-DG tree algorithm. We maintain the same auxiliary structures as those used by the MaxBNW-DG algorithm with their definitions slightly changed: The *Tree-join-weight* for a node is the maximum of the weights of the incident links through which a node can join the DG tree. The *Estimated-join-weight* for a node is the estimate of the optimal (maximum) weight of the link through which the node can join the DG tree. The usage of the rest of the auxiliary variables (like the *Candidate-Nodes-List*, *Optimized-Nodes-List*, *Parent-Node-List*) are the same as that of the MaxBNW-DG tree algorithm. The *Candidate-Nodes-List* is maintained as a maximum heap, based on the *Estimated-join-weights* of the vertices. In each iteration, we remove the vertex with the largest *Estimated-join-weight* among the vertices in the *Candidate-Nodes-List* and add to the *Optimized-Nodes-List* as well as explore its neighbors. For a vertex v that is a neighbor node of a vertex u being optimized in a particular iteration, we check if the current *Estimated-join-weight* of node v is less than the weight of the link (u, v); if so, we update *Estimated-join-weight* of node v to the weight of the link (u, v) and set node u as the parent for node v. We start with an arbitrarily selected node as the root node and set its *Estimated-join-weight* (that is also its *Tree-join-weight*) to ∞, while the *Estimated-join-weight* of the other vertices is set to -1; in each iteration, we attempt to increase the *Estimated-join-weights* of the vertices (that are not yet in the *Optimized-Node-List*) as much as possible when the neighborhood of the vertices that are being optimized is explored.

Input: Graph $G = (V, E, W_V, W_E)$

Output: $T_{Max}^{LW}(V, E_T)$ where E_T is the set of edges in the MaxLW-DG tree

Auxiliary Variables: *Optimized-Nodes-List*, *Candidate-Nodes-List*, *Parent-Node-List*
Estimated-join-weight, *Tree-join-weight*, *Leader Node*

Initialization: *Optimized-Nodes-List* = ϕ; *Candidate-Nodes-List* = ϕ, *Parent-Node-List* = ϕ, E_T = ϕ
$\forall u \in V$: *Estimated-join-weight*(*u*) = -1, *Tree-join-weight*(*u*) = -1

Begin *MaxLW-DG Algorithm*

1 *Leader Node s* is arbitrarily chosen among the vertices in *V*
2 *Parent-Node-List*(*s*) = NULL
3 *Candidate-Nodes-List* = *Candidate-Nodes-List* ∪ {*s*}
4 *Estimated-join-weight*(*s*) = ∞
5 ***while*** (*Candidate-Nodes-List* ≠ ϕ) ***do***
6 Node *u* = {*i* | $\underset{i \in Candidate-Nodes-List}{Maximum}$ (*Estimated-join-weights*(*i*)}
7 *Candidate-Nodes-List* = *Candidate-Nodes-List* - {*u*}
8 *Optimized-Nodes-List* = *Optimized-Nodes-List* ∪ {*u*}
9 *Tree-join-weight*(*u*) = *Estimated-join-weight*(*u*)
10 ***for*** (Node *v* ∈ *Neighbors*(*u*) AND *v* ∉ *Optimized-Nodes-List*) ***do***
11 ***if*** (*Estimated-join-weight*(*v*) < W_E(*u*-*v*)) ***then***
12 *Estimated-join-weight*(*v*) = W_E(*u*-*v*)
13 *Parent-Node-List*(*v*) = *u*
14 ***end if***
15 ***end for***
16 ***end while***
17 ***if*** (*Candidate-Nodes-List* = ϕ AND | *Optimized-Nodes-List* | < | *V* |) ***then***
18 ***return*** NULL; // the graph is not connected
19 ***end if***
20 ***if*** (*Candidate-Nodes-List* = ϕ AND | *Optimized-Nodes-List* | = | *V* |) ***then***
21 ***for*** (Node *u* ∈ *V* AND *u* ≠ *Leader Node*) ***do***
22 $E_T = E_T \cup$ {(*u*, *Parent-Node-List*(*u*))}
23 ***end for***
24 ***end if***
25 ***return*** $T_{Max}^{LW}(V, E_T)$

End *MaxLW-DG Algorithm*

Figure 3: Pseudo Code for the Algorithm to Determine Maximum Link Weight-based DG Trees

Figure 4 presents an example (same graph as in Figure 2) to illustrate the construction of MaxLW-DG trees, starting from node 2 as the root node (to be consistent with the example

illustration for MaxBNW-DG trees). The *Estimated-join-weight* of the root node is set to a very large value of 1000 (equivalent to setting the value to ∞) and *Estimated-join-weight* of the other vertices is set to -1. In each iteration, we optimize the node with the largest *Estimated-join-weight* value (also set as the node's *Tree-join-weight* value) and explore its neighbors; we increase the *Estimated-join-weight* of a neighbor node if it is currently less than the weight of the edge connecting the neighbor node to the node being optimized during that iteration. Unlike the MaxBNW-DG algorithm, we do not observe a monotonically decreasing sequence of the values of the *Tree-join-weights* of the nodes optimized. Also, we can notice that the *Tree-join-weight* of any node need not be the maximum of the weights of the links incident on the node.

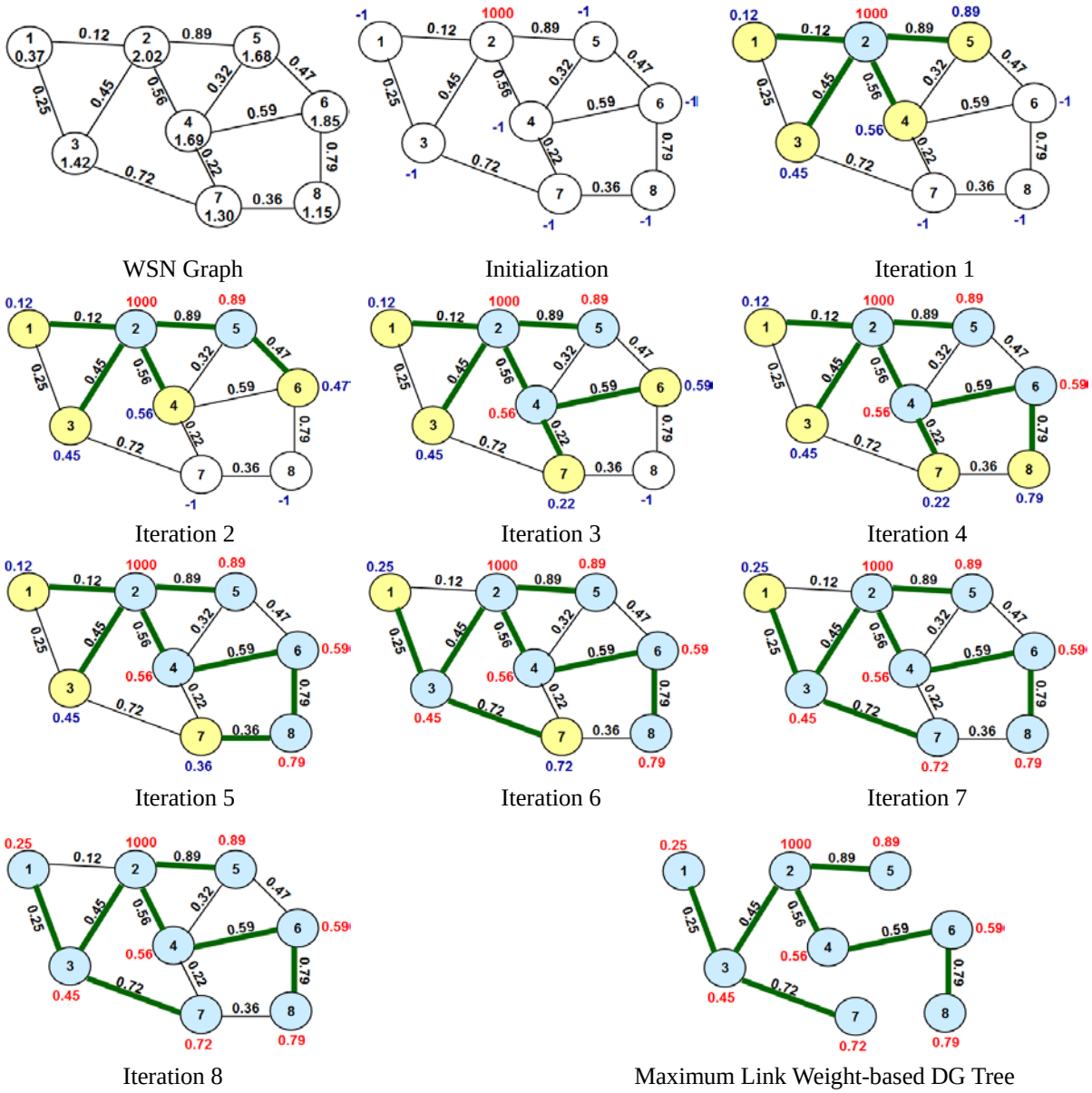


Figure 4: Example to Illustrate the Execution of the Algorithm to Determine Maximum Link Weight-based Data Gathering Tree

Figure 5 presents the pseudo code for the MinLW-DG tree algorithm. Here, the

Tree-join-weight for a node is the minimum of the weights of the incident links through which a node can join the tree; the *Estimated-join-weight* for a node is the estimate of the optimal (minimum) weight of the link through which a node can join the DG tree. The *Candidate-Nodes-List* is maintained as a minimum heap, based on the *Estimated-join-weight* of the vertices. We start with an arbitrarily chosen node as the root node whose *Estimated-join-weight* is set to -∞ and the *Estimated-join-weight* of the other vertices is set to ∞. In each iteration of the algorithm, we attempt to reduce the *Estimated-join-weight* of the vertices (other than the root node) as much as possible. In each iteration, we remove the vertex with the smallest *Estimated-join-weight* among the vertices in the *Candidate-Nodes-List* and add to the *Optimized-Nodes-List* as well as explore its neighbors. For a vertex *v* that is a neighbor node of a vertex *u* that is currently being optimized during an iteration, we reduce *Estimated-join-weight* of node *v* if it is larger than the weight of the link (*u*, *v*) and set the *Estimated-join-weight* of node *v* to the weight of the link (*u*, *v*).

Input: Graph $G = (V, E, W_V, W_E)$

Output: $T_{Min}^{LW}(V, E_T)$ where E_T is the set of edges in the MinLW-DG tree

Auxiliary Variables: *Optimized-Nodes-List, Candidate-Nodes-List, Parent-Node-List*
Estimated-join-weight, Tree-join-weight, Leader Node

Initialization: *Optimized-Nodes-List* = ϕ; *Candidate-Nodes-List* = ϕ, *Parent-Node-List* = ϕ, E_T = ϕ
$\forall u \in V$: *Estimated-join-weight*(*u*) = ∞, *Tree-join-weight*(*u*) = ∞

Begin *MinLW-DG Algorithm*

1 *Leader Node s* is arbitrarily chosen among the vertices in *V*
2 *Parent-Node-List*(*s*) = NULL
3 *Candidate-Nodes-List* = *Candidate-Nodes-List* ∪ {*s*}
4 *Estimated-join-weight*(*s*) = -∞
5 ***while*** (*Candidate-Nodes-List* ≠ ϕ) ***do***
6 Node *u* = {*i* | $\underset{i \in Candidate-Nodes-List}{Minimum}$ (*Estimated-join-weights*(*i*)}
7 *Candidate-Nodes-List* = *Candidate-Nodes-List* - {*u*}
8 *Optimized-Nodes-List* = *Optimized-Nodes-List* ∪ {*u*}
9 *Tree-join-weight*(*u*) = *Estimated-join-weight*(*u*)
10 ***for*** (Node *v* ∈ *Neighbors*(*u*) AND *v* ∉ *Optimized-Nodes-List*) ***do***
11 ***if*** (*Estimated-join-weight*(*v*) > W_E(*u*-*v*)) ***then***
12 *Estimated-join-weight*(*v*) = W_E(*u*-*v*)
13 *Parent-Node-List*(*v*) = *u*
14 ***end if***
15 ***end for***
16 ***end while***
17 ***if*** (*Candidate-Nodes-List* = ϕ AND | *Optimized-Nodes-List* | < | *V* |) ***then***
18 ***return*** NULL; // the graph is not connected
19 ***end if***
20 ***if*** (*Candidate-Nodes-List* = ϕ AND | *Optimized-Nodes-List* | = | *V* |) ***then***

21 ***for*** (Node $u \in V$ AND $u \neq$ *Leader Node*) ***do***

22 $E_T = E_T \cup \{(u, Parent\text{-}Node\text{-}List(u))\}$

23 ***end for***

24 ***end if***

25 ***return*** $T_{Min}^{LW}(V, E_T)$

End *MinLW-DG Algorithm*

Figure 5: Pseudo Code for the Algorithm to Determine Minimum Link Weight-based DG Trees

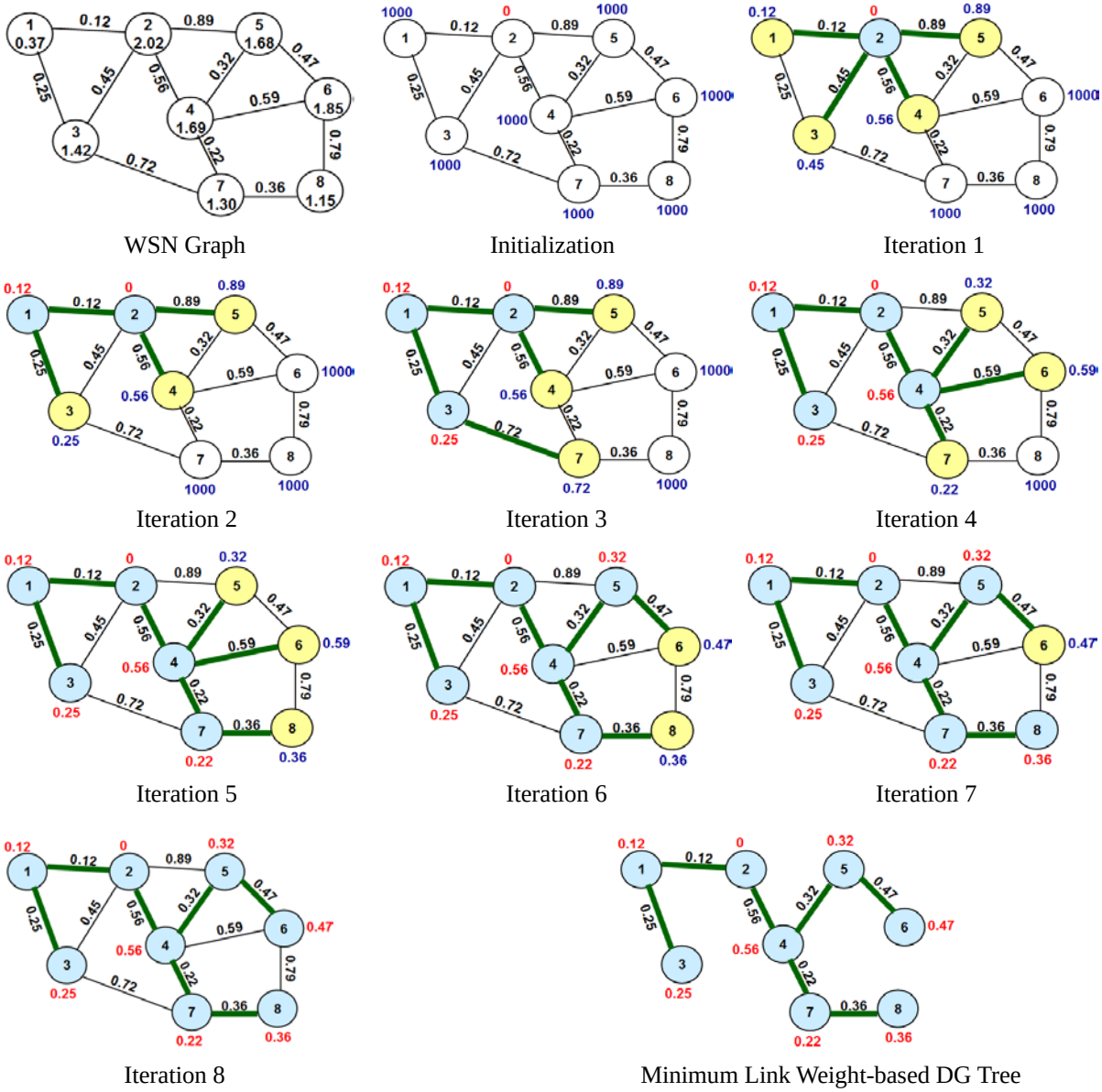


Figure 6: Example to Illustrate the Execution of the Algorithm to Determine Minimum Link Weight-based Data Gathering Tree

Figure 6 presents an example to illustrate the execution of the MinLW-DG tree algorithm on the same graph that was used to illustrate the execution of the MaxBNW-DG and MaxLW-DG algorithms. We use the same root node (node 2) with its initial weight set to 0

(given that the link weights > 0, no other node could ever have a *Tree-join-weight* value of 0) and set the *Estimated-join-weight* for the other nodes to be 1000 (in lieu of ∞).

In each iteration, we seek to decrease the *Estimated-join-weight* of the nodes as much as possible; we consider the vertex with the lowest *Estimated-join-weight* (among the vertices that are yet to be optimized) to have been optimized and set its *Tree-join-weight* to the *Estimated-join-weight* as well as attempt to decrease the *Estimated-join-weight* of each of its neighbor nodes (that are yet to be optimized) if the edge connecting the node with the neighbor node is of a weight lower than the currently known *Estimated-join-weight* of the neighbor node. Similar to the example in Figure 4, the *Tree-join-weights* of the vertices optimized over the sequence of iterations do not monotonically increase or decrease. Also, the *Tree-join-weight* of a node need not be the minimum of the weights of the links incident on the node. We observe the MinLW-DG tree obtained in Figure 6 to be different from the MaxLW-DG tree obtained in Figure 4 and the MaxBNW-DG tree in Figure 2 even though all the three algorithms were run on the same graph.

4. Algorithm to Determine the Height of the DG Tree and Delay for Data Aggregation

In this section, we describe a benchmarking algorithm [15] to determine the delay for data aggregation spanning over all the nodes of the data gathering (DG) tree. We first identify the level of each node in the DG tree, with the root node (leader node) set to be at level 0, its immediate child nodes at level 1 and so on. Two different intermediate nodes at a particular level can simultaneously communicate with their respective child nodes at the same time using different CDMA (Code Division Multiple Access) codes [12] to avoid any interference; an intermediate node could only communicate sequentially with its immediate downstream nodes using TDMA (Time Division Multiple Access) time slots (one data aggregation per time unit).

Figure 7 illustrates the pseudo code for an algorithm to compute the height of the data gathering tree, using the MaxBNW-DG tree as reference. The algorithm can be used to determine the height of any data gathering tree rooted at a particular node. We perform Breadth First Search (BFS) [14] on the edges of the DG tree to determine the height of the tree as well as the level (distance) of the vertices from the root node of the tree. As part of this procedure, we use the auxiliary variables, *Neighbor-List* storing the neighbors of each of the vertices in the DG tree; *Nodes-Explored* storing the list of nodes whose neighborhood is already explored as part of BFS; *Candidate-Nodes-List* containing the list of nodes (maintained in a First-in First-out basis; the front vertex in the list is extracted using a dequeue operation) that are visited while exploring the neighborhood of some other vertices, but their own neighborhood is not yet explored; and *Node-Level* that stores the level (distance) of a node from the root node of the DG tree. In addition to the *Height* of the DG tree, the algorithm outputs the *Child-Nodes-List* that contains the list of child nodes (if any) for each node in the DG tree and a two-dimensional data structure referred to as *Nodes-At-Levels* that stores a list of nodes at each level of the DG tree. The *Node-Level* for the leader node is 0. In each iteration of the BFS algorithm, we remove (dequeue) the vertex in the front of the *Candidate-Nodes-List* and add it to the *Nodes-Explored* list as well as explore its neighbors.

If a neighbor node *v* that has not been visited (explored) so far is visited for the first time as part of the exploration of the neighborhood of a node *u*, we add node *v* to the *Nodes-Explored* list as well as set *Node-Level*[*v*] = *Node-Level*[*u*] + 1 and set node *u* to be the parent node for node *v* in the DG tree. We also add node *v* to the list of nodes at level *Node-Level*[*v*]. If the value for *Node-Level*[*v*] is greater than the currently known height of the DG tree, we set the height of the tree to *Node-Level*[*v*], implying we have found a node that is farther away from the leader node of the DG tree. The complexity of the algorithm to determine the height of a DG tree is the complexity incurred with running the BFS algorithm on the *V*–1 edges of the DG tree spanning over all the |*V*| vertices of the graph. Hence, the complexity of the algorithm to determine the height of a DG tree is Θ(|*V*|).

Input: DG Tree $T_{Max}^{BNW}(V, E_T)$, Leader Node *s*

Output: *Height*, *Nodes-At-Levels*, *Child-Nodes-List*

Auxiliary Variables: *Nodes-Explored*; *Candidate-Nodes-List*, *Child-Nodes-List*; *Neighbor-List*; *Node-Level*

Initialization: $\forall i \in V$: *Neighbor-List*[*i*] = ϕ, *Node-Level*[*i*] = 0; *Child-Nodes-List*[*i*] = ϕ *Nodes-At-Levels* = ϕ; *Height* = 0; *Nodes-Explored* = ϕ; *Candidate-Nodes-List* = ϕ

Begin *Algorithm Height-DG Tree*

1 ***for*** (edge (*u*, *v*) $\in E_T$) ***do***
2 *Neighbor-List*[*u*] = *Neighbor-List*[*u*] $\cup$ {*v*}
3 *Neighbor-List*[*v*] = *Neighbor-List*[*v*] $\cup$ {*u*}
4 ***end for***
5 *Candidate-Nodes-List* = {*s*}
6 *Node-Level*[*s*] = 0
7 *Nodes-At-Levels*[0].add({*s*}) // adds the leader node *s* to the list of nodes at level 0
8 ***while*** (*Candidate-Nodes-List* ≠ ϕ) ***do***
9 Node *u* = Dequeue(*Candidate-Nodes-List*) // removes *u* from *Candidate-Nodes-List*
10 *Nodes-Explored* = *Nodes-Explored* $\cup$ {*u*}
11 ***for*** (Node *v* $\in$ *Neighbor-List*(*u*) AND *v* $\notin$ *Nodes-Explored*) ***do***
12 *Node-Level*[*v*] = *Node-Level*[*u*] + 1
13 *Nodes-At-Levels*[*Node-Level*[*v*]].add({*v*})
14 *Child-Nodes-List*(*u*) = *Child-Nodes-List*(*u*) $\cup$ {*v*}
15 **if** (*Height* < *Node-Level*[*v*]) **then**
16 *Height* = *Node-Level*[*v*]
17 **end if**
19 ***end for***
20 ***end while***
21 ***return*** *Height*, *Nodes-At-Levels*, *Child-Nodes-List*

End *Algorithm Height-DG Tree*

Figure 7: Pseudo Code for an Algorithm to Determine the Height of a Data Gathering Tree

Figure 8 presents the pseudo code that makes use of the height of the DG tree and the list of nodes at various levels of the tree to determine the delay for data aggregation. An intermediate node cannot forward an aggregated data to its upstream node unless it gets the aggregated data from all its immediate downstream child nodes. We assume it takes 1 time unit for a child node to send aggregated data to its parent node. We assume the leaf nodes of the DG tree (at all the levels) to have data readily available at time unit 0 (hence, the delay at any leaf node is 0). The delay at an intermediate node u, *Delay*[u], is iteratively computed over all its child nodes as shown in lines 6-8 of Figure 8 and explained here: We first sort the delays of the child nodes of u and maintain a *Sorted-Delay-Child-Nodes*[u]. We maintain a temporary estimate of the delay (initialized to zero) at the intermediate node u using a running variable *Temp-Delay*[u] that is incremented as follows for each child node of u considered in the increasing order of their delays: For a child node $v \in$ *Child-Nodes-List*[u], *Temp-Delay*[u] = Maximum(*Temp-Delay*[u] + 1, *Delay*[v] + 1), as we assume it takes one time unit for data to reach from a child node to the immediate parent node. The increment of the *Temp-Delay*[u] by 1 within the Maximum function takes into consideration scenarios when the delay at the child node v would be smaller than the *Temp-Delay* already computed at the parent node u due to data aggregation delays involving the siblings of node v (i.e., the other child nodes of node u). The delay associated with an intermediate node u, *Delay*[u], is the final value of *Temp-Delay*[u] after going through the delay-based sorted list of the child nodes of u.

Input: *Height*, *Nodes-At-Levels*, *Child-Nodes-List*, Leader Node *s*

Output: *Delay*[*s*]

Auxiliary Variables: *Temp-Delay*; *Sorted-Delay-Child-Nodes*

Initialization: $\forall i \in V$: *Delay*[*i*] = 0, *Temp-Delay*[*i*] = 0, *Sorted-Delay-Child-Nodes*[*i*] = ϕ

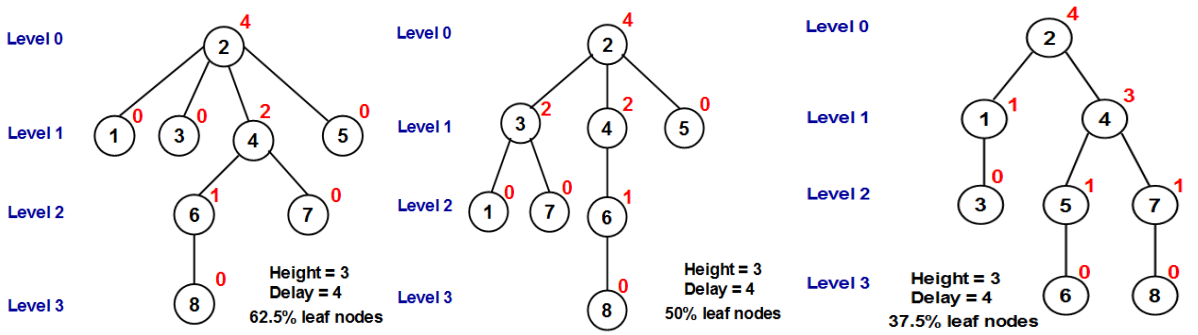
Begin *Algorithm Aggregation-Delay-DG Tree*

```
1    for (Node-level = Height to 0) do
2        for (Node u ∈ Nodes-At-Levels.get(Node-level) ) do
3            for (Node v ∈ Child-Nodes-List[u]) do
4                Insert (v, Delay[v]) at appropriate entry in Sorted-Delay-Child-Nodes[u]
5            end for
6            for (tuple (v, Delay[v]) in Sorted-Delay-Child-Nodes[u]) do
7                Temp-Delay[u] = Maximum(Temp-Delay[u] + 1, Delay[v] + 1)
8            end for
9            Delay[u] = Temp-Delay[u]
10       end for
11   end for
12   return Delay[s]
```

End *Algorithm Aggregation-Delay-DG Tree*

Figure 8: Pseudo Code for an Algorithm to Determine the Aggregation Delay of a Data Gathering Tree

The above procedure is repeated at all the intermediate nodes, including the leader node. The aggregation delay at the leader node is considered to be the aggregation delay of the entire DG tree. The overall complexity of the algorithm to compute the aggregation delay of a DG tree is dominated by the time incurred to sort the delays of the child nodes of the intermediate nodes at the different levels. At the worst case, an intermediate node could have the rest of the nodes as its child nodes and it would take $\Theta(|V|*\log|V|)$ time to sort the delays of the $|V|$-1 child nodes. To calculate the *Temp-Delay* values and the delay at each of the nodes, we spend one time unit for each edge at its upstream node. Hence, the overall complexity of the algorithm to compute the aggregation delay of the DG tree is $\Theta(|V|*\log|V|)$.



(a) MaxBNW-DG Tree (b) MaxLW-DG Tree (c) MinLW-DG Tree

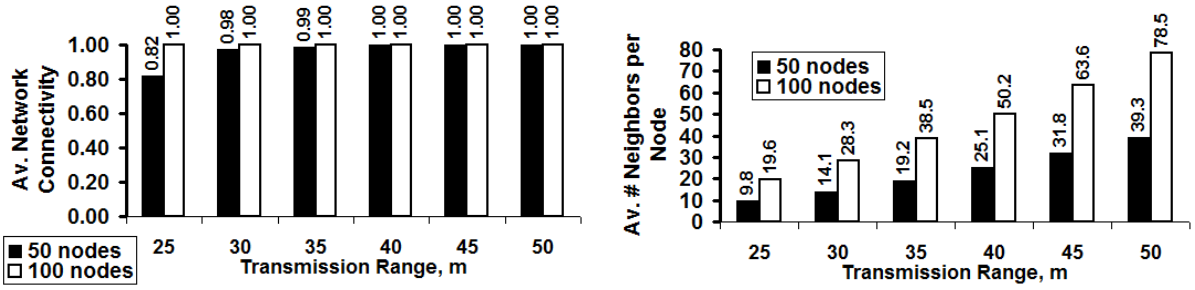
Figure 9: Height, Aggregation Delay and Percentage of Nodes as Leaf Nodes for the DG Trees

Figure 9 illustrates the execution of the algorithms to determine the height and aggregation delay for the (a) MaxBNW-DG, (b) MaxLW-DG and (c) MinLW-DG trees determined in the examples illustrated in Figures 2, 4 and 6. The numbers inside the circles indicate the vertex ID, the numbers outside the circles indicate the aggregation delay at the nodes. We can observe the MaxBNW-DG tree to incur a combination of lower height and a larger percentage of nodes as leaf nodes compared to those incurred with the link weight-based DG trees. Also, in spite of the structural differences, the aggregation delay incurred for all the three DG trees in the example for Figure 9 is equal, which need not be the case all the time, as seen in the simulations.

5. Simulations

We conducted the simulations by implementing a discrete-event simulator in Java for wireless sensor networks. The network dimension is 100m x 100m and the number of nodes is 100. The transmission range of a sensor node is varied from 25m to 50m, in increments of 5m. The number of nodes in the network is set to be 50 (moderate-high node density) and 100 (high-very high node density) and are randomly distributed in the network. We implemented the algorithms for determining Maximum Bottleneck Node Weight-based data gathering, Maximum and Minimum link-weight based data gathering trees in a weighted WSN graph. We conduct two sets of simulations: The first set of simulations compare the performance of the MaxBNW-DG trees vis-a-vis the MaxLW-DG trees wherein the node weights and link

weights are considered to be related. The second set of simulations compare the performance of the MaxBNW-DG trees vis-a-vis the MinLW-DG trees wherein the node weights and link weights are considered to be not related. The performance metrics measured in each set of simulations include (for both sets of simulations): Height (Figures 11-a, 12-a, 14-a, 15-a, 18-a, 19-a), Aggregation Delay (Figures 11-b, 12-b, 14-b, 15-b, 18-b, 19-b), Number of Child Nodes per Intermediate Node (Figures 11-c, 12-c, 14-c, 15-c, 18-c, 19-c) and Percentage of Nodes as Leaf Nodes (Figures 11-d, 12-d, 14-d, 15-d, 18-d, 19-d), Average Weight of the Internal Nodes and Average Weight of the Leaf Nodes for MaxBNW-DG vs. MaxLW-DG trees (Figures 13 and 20) and Average Energy of the Internal Nodes and Average Energy of the Leaf Nodes for MaxBNW-DG vs. MinLW-DG trees (Figure 16). Figure 10 displays (a) average connectivity (averaged over all the network graphs generated for both the sets of simulations) and (b) average number of neighbors per node (given by: $\pi R^2 N/A$, where R is the transmission range of the nodes, N is the number of nodes in the network and A is the network area) in networks of 50 and 100 nodes as a function of the different transmission ranges.



(a) Network Connectivity

(b) Number of Neighbors per Node

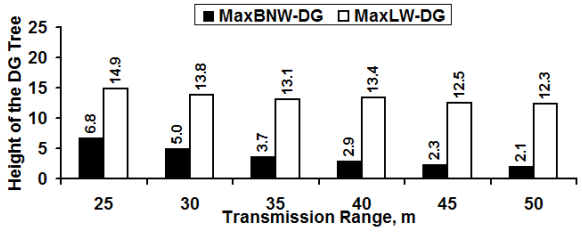
Figure 10: Average Network Connectivity and Average Number of Neighbors per Node

5.1 Maximum Bottleneck Node Weight vs. Maximum Link Weight-based Data Gathering Trees

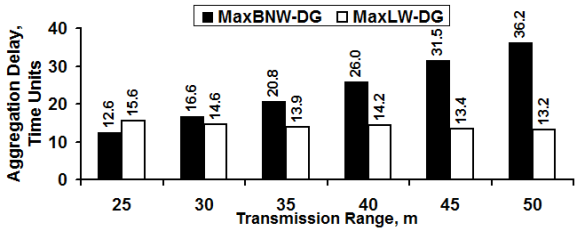
For the first set of simulations, we set the link weights to be randomly distributed in the range [0...1]. The weight of a node is the sum of the weights of the links incident on it. Thus, the node weight is directly related to the link weights as well as indirectly related to the degree of the nodes. A node with larger weights for the incident links and/or a larger number of incident links (i.e., larger degree) is likely to have a larger node weight. For each value of the number of nodes and the transmission range stated above, we generated 200 network graphs (with link weight assignment as mentioned) and averaged the results for the performance metrics (shown in Figures 11, 12 and 13). The environment considered in this set of simulations could mimic scenarios wherein each link in the network has a trust score and the trust score for a node is the sum of the trust scores of its adjacent links. One would then prefer to construct DG trees that involve nodes of larger trust scores as intermediate nodes (in the case of MaxBNW-DG trees) or include links of larger trust scores (in the case of MaxLW-DG trees).

We observe the average height (Figures 11-a, 12-a) for the MaxBNW-DG trees to be significantly lower than that of the MaxLW-DG trees (by factors of 3-9 in networks of

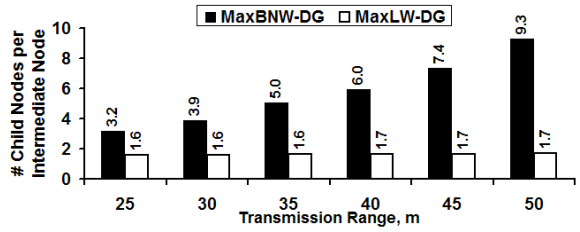
high-very high node density and factors of 2-6 in networks of moderate-high node density), with the difference in height between the two DG trees increasing with increase in the transmission range per node (for a fixed number of nodes). The characteristic of shorter distance from any node to the leader node of a MaxBNW-DG tree is very useful for real-time reporting of the sensed data by an individual node to the leader node and also facilitates one-to-one communication between any two nodes in the network through the leader node.



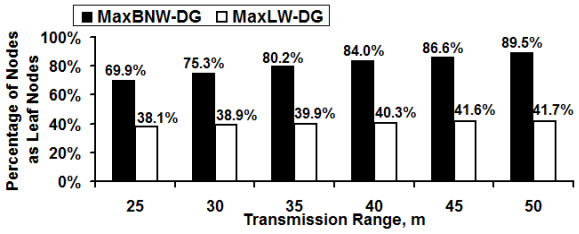
(a) Height of the DG Tree



(b) Aggregation Delay

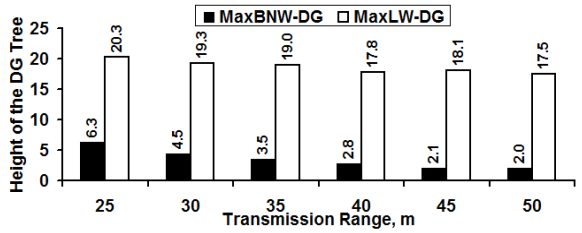


(c) Child Nodes per Intermediate Node

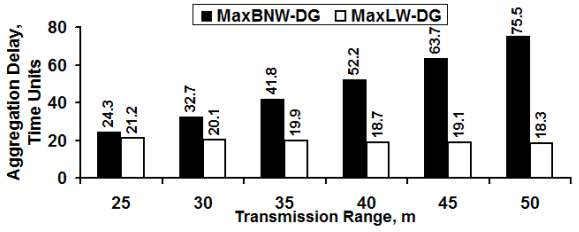


(d) Percentage of Nodes as Leaf Nodes

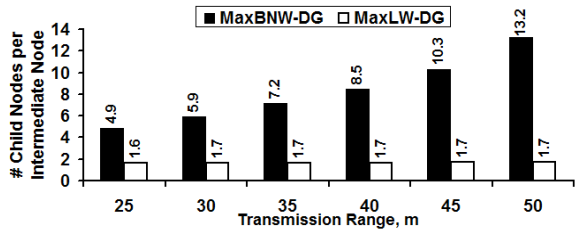
Figure 11: Height, Aggregation Delay, Avg. # Child Nodes per Intermediate Node and Percentage of Nodes as Leaf Nodes [MaxBNW-DG Trees vs. MaxLW-DG Trees: Moderate-High Node Density]



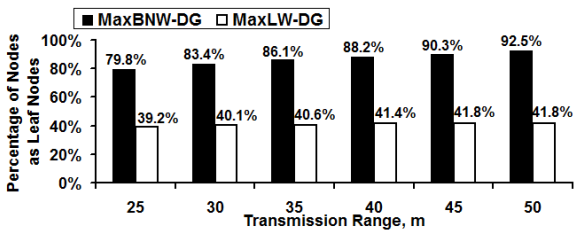
(a) Height of the DG Tree



(b) Aggregation Delay



(c) Child Nodes per Intermediate Node



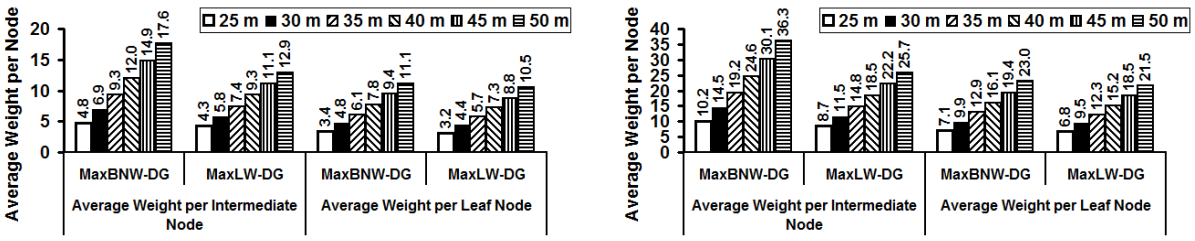
(d) Percentage of Nodes as Leaf Nodes

Figure 12: Height, Aggregation Delay, Avg. # Child Nodes per Intermediate Node and Percentage of Nodes as Leaf Nodes [MaxBNW-DG Trees vs. MaxLW-DG Trees: High-Very High Node Density]

The fewer the number of hops between any two nodes in a DG tree, the lower the energy consumption in the network. While the height of the MaxBNW-DG trees decreases significantly with increase in node density (the tree becomes more shallow by accommodating a significantly larger number of child nodes per intermediate node), the height of the MaxLW-DG tree only marginally decreases with increase in node density (as we observe the number of child nodes per intermediate node to remain about the same with increase in node density).

The MaxBNW-DG trees incur a significantly larger percentage of nodes as leaf nodes (Figures 11-d, 12-d) in the network (almost double the percentage of nodes as leaf nodes incurred by the MaxLW-DG trees) and the difference increases with increase in node density. While the MaxBNW-DG trees exhibit a significant increase in the percentage of nodes as leaf nodes increases with increase in node density, the MaxLW-DG trees exhibit minimal impact of the increase in node density on the percentage of nodes as leaf nodes. A larger percentage of nodes as leaf nodes in a DG tree is a favorable characteristic for lowering the overall energy consumption and increasing the network lifetime. Note that the leaf nodes spend energy only to transmit the data packets; whereas, the intermediate nodes spend energy to receive the data packets from the child nodes, aggregate them as well as forward to their upstream node in the DG tree. Hence, the network lifetime is more likely to be longer if the number of intermediate nodes is lower.

We also observe the average weight per intermediate node (Figure 13) in a MaxBNW-DG tree to be appreciably greater (20-40% greater) than the average weight per intermediate node in a MaxLW-DG tree. In networks where link weights represent trust scores for a link and node weights (sum of the incident link weights) are modeled as the trust scores for the nodes, we thus see that a MaxBNW-DG tree is more likely to have a collection of intermediate nodes that are relatively more trustworthy than that of a MaxLW-DG tree constructed on the same network graph. Though the average weight per leaf node (Figure 13) is not of any practical significance, we do observe the average weight per leaf node in a MaxBNW-DG tree to be about 5-8% greater than the average weight per leaf node in a MaxLW-DG tree.



(a) Moderate-High Node Density [50 Nodes] (b) High-Very High Node Density [100 Nodes]

Figure 13: Average Weight per Node [MaxBNW-DG Trees vs. MaxLW-DG Trees]

As the node density increases, MaxBNW-DG trees incur a significantly larger aggregation delay (Figures 11-b, 12-b) if data has to be aggregated across all the nodes (network-wide). It could be attributed to the DG tree being more shallow and each

intermediate node having a larger number of child nodes (Figures 11-c, 12-c), especially as the node density gets higher. We observe the average number of child nodes per intermediate node incurred with the MaxBNW-DG trees to be about 2.5 - 7.5 times larger than that incurred with the MaxLW-DG trees. The aggregation delay incurred with the MaxBNW-DG trees could be as large as four times the aggregation delay incurred with the MaxLW-DG trees.

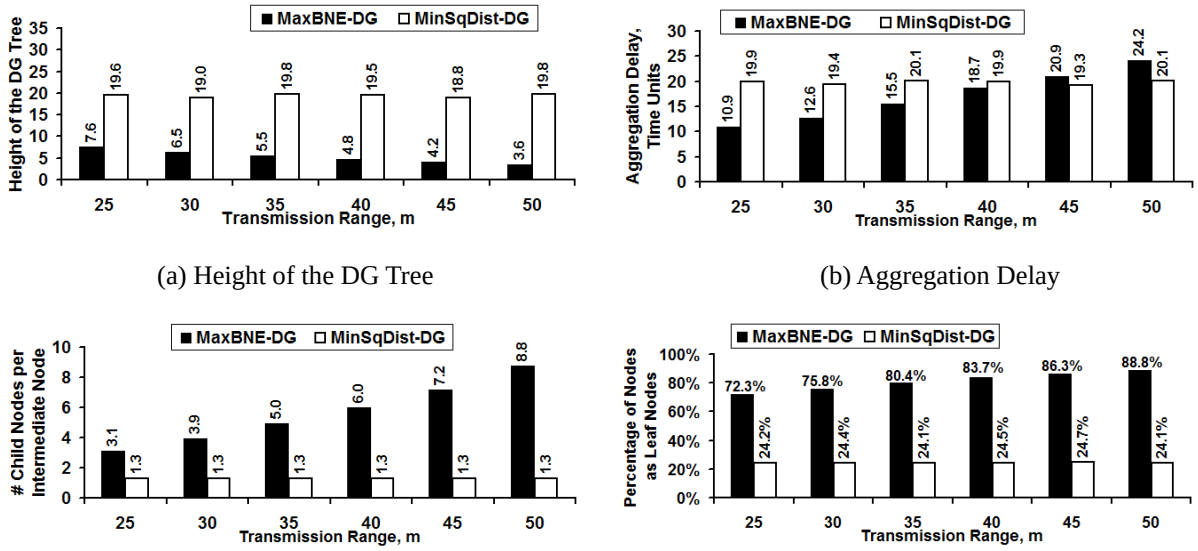
5.2 Maximum Bottleneck Node Weight vs. Minimum Link Weight-based Data Gathering Trees

For the second set of simulations, we set the node weights to be independent of the link weights. For each transmission range value, we generate 200 network graphs in each of which the nodes are randomly distributed in the network area. In each such graphs, the weights for the nodes are randomly assigned in the range [0...1] to represent the residual energy of the nodes; the weight for a link (u, v) is the square of the Euclidean distance between the two end nodes u and v. We model the link weight as per the commonly used energy consumption model [13] for sensor networks, according to which the energy lost to transmit a k–bit message over a distance d is given by $E_{elec}*k + \in_{amp} *k*d^2$, wherein E_{elec} and $\in_{amp}$ are respectively the energy expended by a radio to run the transmitter or receiver circuitry and the transmitter amplifier. Energy lost during transmission over a link is typically modeled as the square of the distance between the end nodes of the link [1-11].

We run the MaxBNW-DG algorithm to determine a maximum bottleneck node energy-based data gathering tree for which the bottleneck node energy (minimum energy) of the constituent nodes on a path (including the end nodes of the path) from any node to the leader node of the MaxBNW-DG tree is the maximum as well as the average energy per intermediate node is significantly larger than the average energy per leaf node. The MinLW-DG tree algorithm determines a DG tree for which the sum of the squares of the distances between the end nodes of all the links in the DG tree is the minimum; this way, the transmission energy lost per link is minimized. As can be noticed, the node weights (node energy) and link weights (square of the distance between the end nodes of the link) are not related to each other. Since we work on static snapshots of the network graphs and do not simulate any packet transfer across the links, the energy lost at a node due to packet transmission or reception does not need to be taken into account. We refer to the MaxBNW-DG tree as the MaxBNE-DG tree and the MinLW-DG tree as the MinSqDist-DG tree.

We observe the MaxBNE-DG trees to exhibit a similar trend of performance (as the MaxBNW-DG trees in Section 5.1) with respect to all the metrics. Though the height of the MaxBNE-DG trees (Figures 14-a, 15-a) tend to be about 50-100% larger than that of the MaxBNW-DG trees (seen in Section 5.1), we observe a similar trend of decrease in the height of the DG tree with increase in node density. The distance (number of hops) between any node to the leader node of the MaxBNE-DG tree does not increase significantly and remains comparable to that of the MaxBNW-DG trees. The height for the MinSqDist-DG trees is

about 50% larger than that of the MaxLW-DG trees (seen in Section 5.1); however, similar to that of the MaxLW-DG trees, the height of the MinSqDist-DG trees remains about the same or even slightly increases with increase in the transmission range of the nodes (for a given number of nodes). The height of the MinSqDist-DG trees is about a factor of 2.5-5.5 and 3.5-8.5 larger than that of the MaxBNE-DG trees for networks of moderate-high density and high-very high density respectively.



(a) Height of the DG Tree

(b) Aggregation Delay

(c) Child Nodes per Intermediate Node

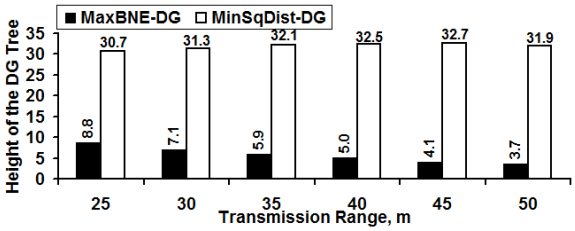
(d) Percentage of Nodes as Leaf Nodes

Figure 14: Height, Aggregation Delay, Avg. # Child Nodes per Intermediate Node and Percentage of Nodes as Leaf Nodes [MaxBNE-DG Trees vs. MinSqDist-DG Trees: Moderate-High Node Density]

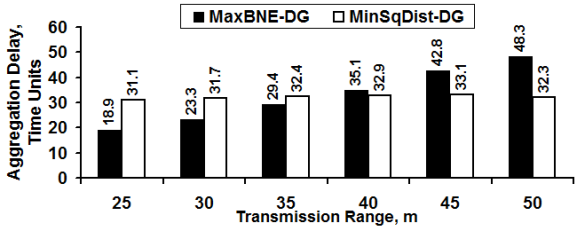
As is the case of the MaxBNW-DG trees, the number of child nodes per intermediate node for the MaxBNE-DG trees (Figures 14-c, 15-c) is much higher than that incurred with the MinSqDist-DG trees whose average number of child nodes per intermediate node is even lower than that of the MaxLW-DG trees. With about 1.3 child nodes per intermediate node, the network-wide aggregation delay for the MinSqDist-DG tree is almost the same as the height of the tree. The network-wide aggregation delay (Figures 14-b, 15-b) incurred with the MaxBNE-DG trees is lower than that of the MinSqDist-DG trees when the height of the MaxBNE-DG trees is moderately larger and the child nodes are evenly spread across all the intermediate nodes (noticeable when the node density is moderate-high and transmission ranges are in the lower end of the values used); the delay incurred with the MaxBNE-DG trees increases to become more than that incurred with the MinSqDist-DG trees when the transmission range gets very high and the height of the MaxBNE-DG trees decreases significantly (leading to a very large number of child nodes per intermediate node).

For a given node density, the aggregation delay incurred with the MaxBNE-DG trees is significantly smaller than that of the MaxBNW-DG trees (by a factor of 50-100%); Thus, though (in general), the maximum node weight-based DG trees appear to incur a larger aggregation delay compared to the link-weight based data gathering trees, the actual magnitude of the aggregation delay depends on the criterion used for node weight and the impact of it on the height of the DG tree and the number of child nodes per intermediate node.

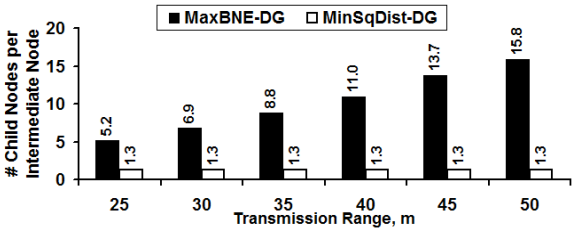
If the criterion for node weight is related to the number of incident links on the node (like the criterion used in the first set of simulations), nodes with a larger degree are more likely to be favored as intermediate nodes and the height of the DG tree decreases significantly with increase in node density, leading to a significant increase in the aggregation delay. On the other hand, if the criterion for node weight is independent of the degree of the node (like the node energy criterion used in the second set of simulations), the height of the DG tree remains at a moderate value and the aggregation delay is only moderately high compared to that incurred with the link weight-based data gathering trees.



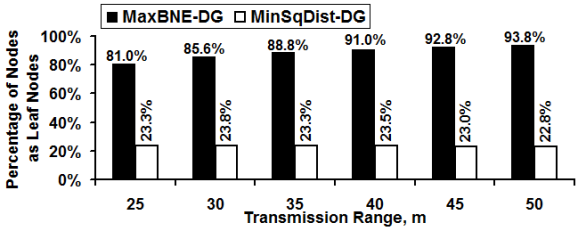
(a) Height of the DG Tree



(b) Aggregation Delay

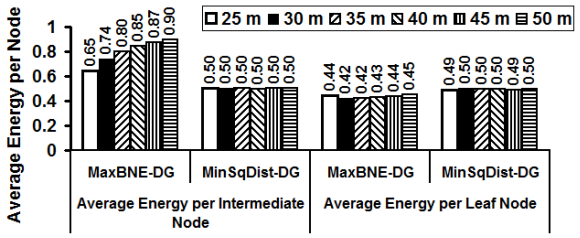


(c) Child Nodes per Intermediate Node

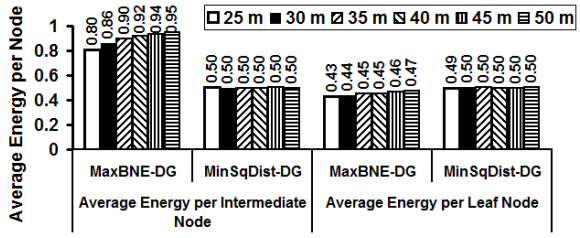


(d) Percentage of Nodes as Leaf Nodes

Figure 15: Height, Aggregation Delay, Avg. # Child Nodes per Intermediate Node and Percentage of Nodes as Leaf Nodes [MaxBNE-DG Trees vs. MinSqDist-DG Trees: High-Very High Node Density]



(a) Moderate-High Node Density [50 Nodes]



(b) High-Very High Node Density [100 Nodes]

Figure 16: Average Energy per Node [MaxBNE-DG Trees vs. MinSqDist-DG Trees]

With regards to the percentage of nodes as leaf nodes (Figures 14-d, 15-d), we observe both the MaxBNW-DG and MaxBNE-DG trees to incur almost about the same value, with the latter having a slightly larger number of leaf nodes (with values above 90% in networks of high-very high node density and high transmission range). On the other hand, the percentage of nodes observed as leaf nodes in the MinSqDist-DG trees is below 25% (and also about 15-18% lower than that incurred with the MaxLW-DG trees). This could be attributed to the significantly larger height of the MinSqDist-DG trees.

With regards to the average energy per node (Figure 16), we observe the average energy per intermediate node for the MaxBNE-DG trees to be almost twice the average energy per

node for the MinSqDist-DG trees. The ratio of the average energy per intermediate node for the MaxBNE-DG trees to that of the MinSqDist-DG trees is even larger than the ratio of the average weight per intermediate node for the MaxBNW-DG trees to that of the MaxLW-DG trees (despite there being an upper limit on the energy per node and no such direct constraint on the weight per node in the simulations of section 5.1). On the other hand, the MinSqDist-DG trees appear to sustain an even distribution of the energy levels (the average energy per intermediate node is about the same as the average energy per leaf node, despite the percentage of nodes as leaf nodes being below 25%).

6. Minimum Bottleneck Node Weight-based Data Gathering Trees

In this section, we show how the proposed generic algorithm for maximum bottleneck node weight-based data gathering trees (MaxBNW-DG trees) could be easily modified to determine minimum bottleneck node weight-based data gathering trees (MinBNW-DG trees) for which the bottleneck weight of a path from a node to the root node of the DG tree is defined as the largest of the node weights on the path (including the weights for the end nodes) and the focus would be to determine a DG tree for which the path from any node to the root node has the minimum bottleneck weight. To determine such MinBNW-DG trees, one would have to set the root node to be the node with the smallest node weight, set the initial values for the *Estimated-join-weight* of the other nodes to be ∞ and in each iteration attempt to reduce the *Estimated-join-weight* of the nodes as much as possible (by comparing with the maximum of the *Tree-join-weight* of the node being optimized and the individual weight of the node and updating the *Estimated-join-weight* if it is still larger than the maximum of the above two values). Figure 17 presents the pseudo code for the algorithm to determine MinBNW-DG trees. An example scenario for employing the minimum bottleneck node weight-based DG trees would be when node weights are modeled as node velocities in a mobile sensor network and we target to determine stable data gathering trees such that the maximum velocity for any node on the path is the minimum.

Figures 18-20 present simulation results for MinBNW-DG trees wherein the node weights are modeled as the node velocities assigned randomly from the range [0...1]. We refer to the MinBNW-DG trees as the MinBNV-DG trees as they represent the node velocity in this simulation study. The values for the other operating parameters are the same as those used in Sections 5.1 and 5.2. We compare the values of the performance metrics for the MinBNV-DG trees with that of the MaxBNE-DG trees. Both the trees exhibit almost a similar performance with respect to the height (Figures 18-a, 19-a), aggregation delay (Figures 18-b, 19-b), number of child nodes per intermediate node (Figures 18-c, 19-c) and the percentage of nodes as leaf nodes (Figures 18-d, 19-d).

Input: Graph $G = (V, E, W_V, W_E)$

Output: $T_{Min}^{BNW}(V, E_T)$ where E_T is the set of edges in the MinBNW-DG tree

Auxiliary Variables: *Optimized-Nodes-List, Candidate-Nodes-List, Parent-Node-List*
Estimated-join-weight, Tree-join-weight, Leader Node

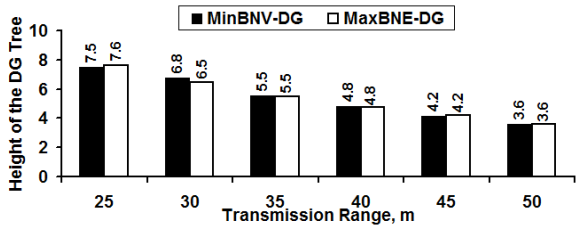
Initialization: *Optimized-Nodes-List* = ϕ; *Candidate-Nodes-List* = ϕ, *Parent-Node-List* = ϕ, E_T = ϕ
$\forall u \in V$: *Estimated-join-weight*(*u*) = ∞, *Tree-join-weight*(*u*) = ∞

Begin *MinBNW-DG Algorithm*

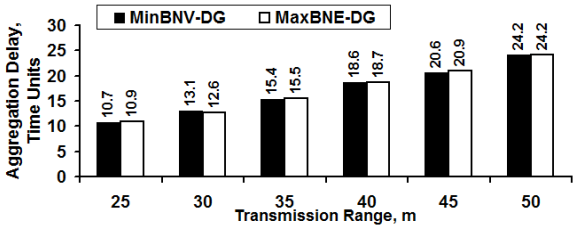
1 *Leader Node s* = {*u* | $Minimum(W_V(u))$ }
2 *Parent-Node-List*(*s*) = NULL
3 *Candidate-Nodes-List* = *Candidate-Nodes-List* ∪ {*s*}
4 *Estimated-join-weight*(*s*) = W_V(*s*)
5 ***while*** (*Candidate-Nodes-List* ≠ ϕ) ***do***
6 Node *u* = {*i* | $\underset{i \in Candidate-Nodes-List}{Minimum}$ (*Estimated-join-weights*(*i*)}
7 *Candidate-Nodes-List* = *Candidate-Nodes-List* - {*u*}
8 *Optimized-Nodes-List* = *Optimized-Nodes-List* ∪ {*u*}
9 *Tree-join-weight*(*u*) = *Estimated-join-weight*(*u*)
10 ***for*** (Node *v* ∈ *Neighbors*(*u*) AND *v* ∉ *Optimized-Nodes-List*) ***do***
11 ***if*** (*Estimated-join-weight*(*v*) > ***Maximum***(W_V(*v*), *Tree-join-weight*(*u*))) ***then***
12 *Estimated-join-weight*(*v*) = ***Maximum***(W_V(*v*), *Tree-join-weight*(*u*))
13 *Parent-Node-List*(*v*) = *u*
14 ***end if***
15 ***end for***
16 ***end while***
17 ***if*** (*Candidate-Nodes-List* = ϕ AND | *Optimized-Nodes-List* | < | *V* |) ***then***
18 ***return*** NULL; // the graph is not connected
19 ***end if***
20 ***if*** (*Candidate-Nodes-List* = ϕ AND | *Optimized-Nodes-List* | = | *V* |) ***then***
21 ***for*** (Node *u* ∈ *V* AND *u* ≠ *Leader Node*) ***do***
22 $E_T = E_T \cup \{(u, Parent\text{-}Node\text{-}List(u))\}$
23 ***end for***
24 ***end if***
25 ***return*** $T_{Min}^{BNW}(V, E_T)$

End *MinBNW-DG Algorithm*

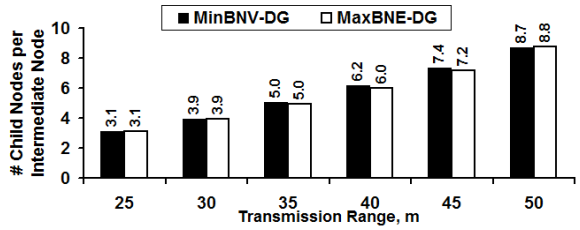
Figure 17: Pseudo Code for the Algorithm to Determine Minimum Bottleneck Node Weight-DG Trees



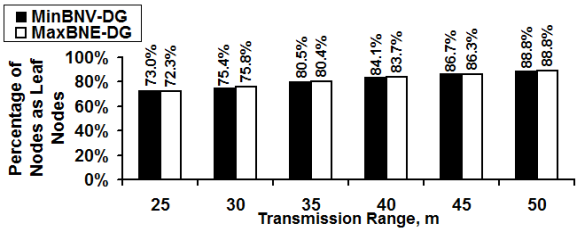
(a) Height of the DG Tree



(b) Aggregation Delay

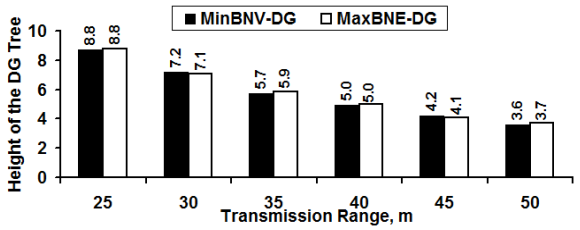


(c) Child Nodes per Intermediate Node

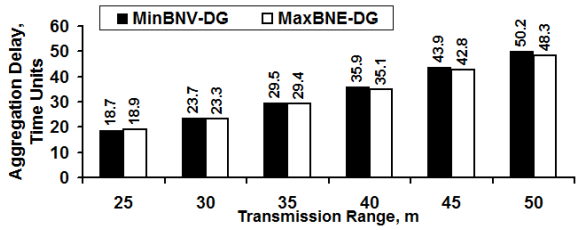


(d) Percentage of Nodes as Leaf Nodes

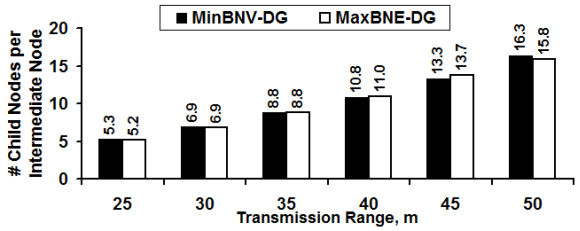
Figure 18: Height, Aggregation Delay, Avg. # Child Nodes per Intermediate Node and Percentage of Nodes as Leaf Nodes [MinBNV-DG Trees vs. MaxBNE-DG Trees: Moderate-High Node Density]



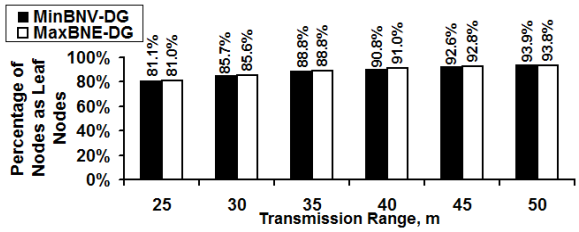
(a) Height of the DG Tree



(b) Aggregation Delay



(c) Child Nodes per Intermediate Node

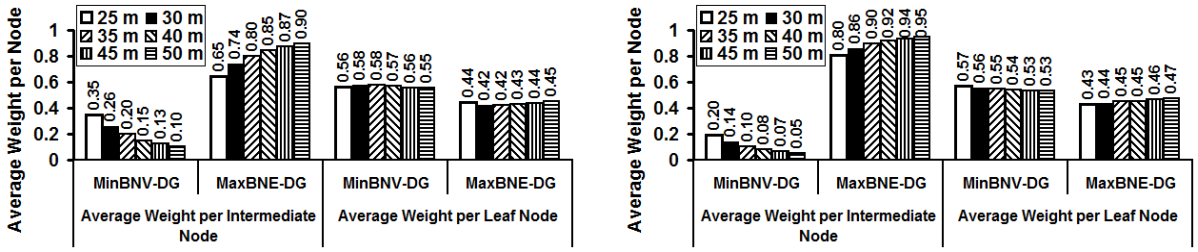


(d) Percentage of Nodes as Leaf Nodes

Figure 19: Height, Aggregation Delay, Avg. # Child Nodes per Intermediate Node and Percentage of Nodes as Leaf Nodes [MinBNV-DG Trees vs. MaxBNE-DG Trees: High-Very High Node Density]

As expected for MinBNW-DG trees, the average weight per intermediate node (Figure 20) in the MinBNV-DG trees is significantly lower than the average weight per leaf node. While the average weight per intermediate node for the MinBNV-DG trees decreases with increase in node density, the average weight per intermediate node for the MaxBNE-DG trees

increases with increase in node density (both the algorithms make use of the increase in the number of nodes to accomplish the expected performance for their respective data gathering trees as much as possible, with respect to the average weight of the intermediate nodes). We thus see that both the MaxBNW-DG and MinBNW-DG trees exhibit an identical performance (with respect to height, aggregation delay, number of child nodes per intermediate node, percentage of nodes as leaf nodes) and optimize (maximize or minimize) the bottleneck weight of the paths (as appropriately defined) for their respective data gathering trees.



(a) Moderate-High Node Density [50 Nodes]

(b) High-Very High Node Density [100 Nodes]

Figure 20: Average Weight per Node [MinBNV-DG Trees vs. MaxBNE-DG Trees]

7. Related Work

In this section, we discuss sample works that have been proposed in the literature for node-weight based data gathering trees and we show how the proposed generic MaxBNW-DG algorithm could be used to study/simulate each of them. We also highlight the strengths of the MaxBNW-DG algorithm over some of the related works proposed in the literature. The performance of the energy-aware maximal leaf nodes-based data gathering tree (EMLN-DG tree) [16] also vindicates the observations made in this paper. EMLN-DG trees were constructed with nodes having a larger value for the product of the residual energy available at the nodes and the number of uncovered neighbors during each iteration of the algorithm. EMLN-DG trees were observed to incur a relatively lower energy loss per round, higher value for the round of first node failure (node lifetime) as well as a lower aggregation delay per round compared to that of the well-known LEACH [1] and PEGASIS [4] algorithms for data gathering. The proposed generic algorithm for MaxBNW-DG trees could be used to determine EMLN-DG trees using the product of the available energy at the nodes and the number of uncovered neighbors (or the degree of the nodes) as the node selection criterion.

In [9], a heuristic for maximizing the lifetime (MLT) of the data gathering trees was proposed by suggesting to include nodes in the tree (one by one, starting from the sink node that is assumed to have the largest energy among the nodes - like the leader node of our MaxBNW-DG tree) such that the network lifetime resulting with the inclusion of a new node v to the tree is at least the network lifetime incurred without the inclusion of node v to the tree or (if the reduction is unavoidable) the reduction in the network lifetime due to the inclusion of a node is minimized; the node that best meets the above criterion is preferred for inclusion

in the DG tree in each iteration of the algorithm. Data gathering trees determined with the above MLT heuristic could be very well determined using our proposed MaxBNW-DG algorithm by modeling the node selection criterion as the residual energy of the nodes. Recall that in each iteration of the MaxBNW-DG algorithm, we explore the neighborhood of the vertex being optimized by taking the minimum of the *Tree-join-weight* of the vertex and the weight of the neighbor node, and increase the currently *Estimated-join-weight* of the neighbor node if it is less than the minimum of the above two variables. This way, if a neighbor node has a larger weight than the *Tree-join-weight*, the currently *Estimated-join-weight* is the *Tree-join-weight*; otherwise, the currently *Estimated-join-weight* is the weight of the neighbor node itself. And in the beginning of each iteration of the MaxBNW-DG algorithm, we pick the vertex that has the largest *Estimated-join-weight* - this way, the *Tree-join-weight* of a node optimized in iteration i+1 is either the same as the *Tree-join-weight* of the node optimized in iteration i or as close as possible (i.e., with minimum difference) if less than the *Tree-join-weight* of the node optimized in iteration i (this is equivalent to the MLT heuristic's idea of either maintaining the same network lifetime without the inclusion of a new node or minimizing the reduction in the network lifetime with the inclusion of a new node, if the reduction is unavoidable).

In [17], a minimum transmission data gathering tree algorithm was proposed for compressive sensing to construct DG trees in which the nodes that report data at a high frequency are preferred to be close to the root of the tree and nodes that report data less frequently are preferred to be away from the root. We can transform this problem to the problem of constructing a maximum bottleneck weight-based data gathering tree with the node selection criterion as the frequency of data reporting; nodes that occasionally report data are more likely to be the leaf nodes and nodes that often send updates to the root/leader node are more likely to be the intermediate nodes that are relatively closer to the leader node. Since the MaxBNW-DG trees are characteristic to be of reduced height, the transmissions will be only on fewer links of the DG tree (thereby reducing the overall energy consumption and prolonging the network lifetime).

If the sensor nodes in the network differ in the wake-up frequencies (proportional to their available energy) [18], the MaxBNW-DG tree algorithm could also be used to determine DG trees for which the path from a node to the leader node is guaranteed to involve nodes whose wake-frequency is at least as large as the wake-frequency of the end nodes of the path; this way, data originating from any node could reach the leader node without any in ordinate delay due to the intermediate nodes on the tree. In a similar context, if each sensor node in the network has an admissible flow [19], the MaxBNW-DG tree algorithm could also be used to determine DG trees such that the path from any node to the leader node is guaranteed to honor the admissible flow of the end nodes of the path (i.e., every intermediate node as well as the node at one end of the path are guaranteed to have an admissible flow that is at least the admissible flow of the node at the other end of the path). In [20], the authors have proposed a distributed bottleneck flow control technique for mobile ad hoc networks (MANETs) wherein each node monitors its channel to determine the residual capacity and the source manages to determine a path of the largest bottleneck weight. The MaxLW-DG

algorithm discussed in this paper can serve as a benchmarking algorithm to evaluate the relative performance of the distributed protocol. The MaxBNW-DG and MaxLW-DG algorithms can also be used in conjunction with some of the recently proposed congestion control algorithms (e.g., [21]).

In [22], an iterative algorithm was proposed to transform an arbitrary spanning tree of the nodes to a data gathering tree that is more likely to involve edges between nodes having larger energy (i.e., the degree of the nodes in the data gathering tree is proportional to the energy of the nodes); the algorithm defines a parameter called the inverse lifetime of a node as the ratio of the degree of the node in the tree divided by the available energy at the node, and groups the nodes in the network into three categories: (i) bottleneck nodes whose inverse lifetime is greater than a threshold; (ii) vulnerable nodes whose inverse lifetime is within a range of the threshold and (iii) safe nodes are the nodes that are neither bottleneck nor vulnerable. For any two safe nodes u and v in an edge (u, v) that is not part of the initial spanning tree T, the algorithm explores the possibility of using edge (u, v) to replace an edge between two bottleneck nodes or between a bottleneck node or a vulnerable node in T. The weakness of this algorithm is that the chances of finding an edge (u, v) involving two safe nodes u and v to replace an edge in the original spanning tree reduces significantly even with a moderate reduction in the number of safe nodes; as a result, as the number of safe nodes reduces, most of the edges in the final data gathering tree could end up being the edges in the initial arbitrarily determined spanning tree (determined without any consideration for the energy level of the nodes or their degree). On the other hand, our proposed MaxBNW-DG algorithm is aware of the node weights throughout the process of constructing the data gathering trees and hence is inherently capable of determining DG trees (of relatively larger lifetime) for which the average energy per intermediate node is guaranteed to be greater than the average energy per leaf node, for all scenarios. Also, the complexity of the algorithm proposed in [22] has been shown proportional to the product of $|V|$, $|E|$ and $\log|V|$; whereas, the complexity of our MaxBNW-DG algorithm is $\Theta(|V| + |E|*\log|V|)$, where $|V|$ and $|E|$ are respectively the number of vertices and edges in the graph.

In [11], it was observed that a connected dominating set [14] of the vertices that have a larger value for the product of the available energy and degree of the node could be used as the basis to construct a data gathering tree that significantly prolongs the network lifetime. A connected dominating set (CDS) is a subset of the vertices in a graph such that for every vertex u in the graph, u is either in the CDS or is a neighbor of a node in the CDS [14]. CDSs are preferred for minimizing the broadcast overhead in ad hoc and sensor networks [23], as a message broadcast by every CDS node in their respective neighborhoods is guaranteed to reach the rest of the nodes in the network. Since the entire graph (if connected) is a candidate for being a CDS, we typically desire to minimize the size of the CDS (number of nodes being part of the CDS) as much as possible to reduce the broadcast overhead. The proposed generic algorithms for MaxBNW-DG or MinBNW-DG trees could be directly used to determine such connected dominating sets (based on different node selection criterion): as the MaxBNW-DG and MinBNW-DG trees are characteristic of incurring a larger percentage of nodes as leaf nodes (at least 80% of the nodes are leaf nodes), a CDS of the intermediate nodes (at most

20% of the nodes) of the MaxBNW-DG or MinBNW-DG trees would be of lower size and could be constructed by simply adding edges between any two intermediate nodes as long as they are within the neighborhood of each other.

8. Conclusions and Future Work

The performance of the maximum bottleneck node weight-based data gathering trees in both the simulation studies (Sections 5.1 and 5.2) and comparison with related work (Section 7) truly vindicate the generic nature of the proposed algorithm to work for any suitable criterion of node weights (that are dependent or not dependent on link weights) and stay true to its original hypothesis - the algorithm is likely to determine data gathering trees with a lower height, larger weight per intermediate node on average (compared to the leaf node) as well as a significantly larger percentage of nodes as leaf nodes (contributing towards lower energy consumption). Thus, given a choice between constructing DG trees of larger bottleneck node weights or larger (smaller) link weights, one could choose the maximum bottleneck-weight based DG trees for reduced hop count between any node to the leader node of the tree (facilitates real-time reporting of events), larger average weight per intermediate node (e.g., for trust-based data gathering) and significantly larger percentage of nodes as leaf nodes (as large as 90%) that could contribute towards lower energy consumption.

The tradeoff is a larger aggregation delay for network-wide data gathering incurred with the maximum bottleneck weight-based DG trees, especially those for which the criterion for node weight is related to the node degree (like in the simulations of Section 5.1 wherein the node weight is the sum of the link weights), attributed to the significantly lower height, a larger number of child nodes per intermediate node and a larger percentage of nodes as leaf nodes. We observe the MaxLW-DG trees (the link weight is a random number from [0...1]) to incur the lowest aggregation delay; the MinLW-DG trees (the link weight is the square of the Euclidean distance between the end nodes of the link) incur a significantly larger height that contributes to a much higher aggregation delay (that is even larger than the aggregation delay incurred with the MaxBNE-DG trees in networks with moderate node density).

Overall, observing the results obtained for the simulation studies of Section 5.2 and comparing them with the results obtained in Section 5.1, we opine that if the criterion used for node weight is independent of the link weight, the maximum bottleneck node weight-based data gathering trees could incur a network-wide aggregation delay that would be comparable (or only moderately higher) to that incurred with the link-weight based data gathering trees and at the same time retain all the other desirable performance measures (for longer network lifetime with reduced energy consumption) such as lower height, larger percentage of nodes as leaf nodes and a larger weight per intermediate node (also suitable for trust-based data gathering trees).

As part of future work, we plan to work on developing distributed versions of the MaxBNW-DG and MinBNW-DG algorithms for optimizing energy consumption and stability respectively, and compare their performance with that of the benchmarking

algorithms proposed in this paper. We also intend to use centrality metrics (like betweenness centrality) [24] to arrive values for node weights and use these to determine MaxBNW-DG and MinBNW-DG trees and evaluate the relative performance of these trees.

Acknowledgment

The research is financed by the NASA EPSCoR sub award (#: NNX14AN38A) from University of Mississippi.

References

[1] R. K. Yadav, V. Kumar and R. Kumar, "Energy Efficient Hybrid Clustering Protocol for Multilevel Heterogeneous Wireless Sensor Network," *Proceedings of the International Conference on Computer and Communications Technologies*, pp. 1-5, Hyderabad, India, December 2014. http://dx.doi.org/10.1109/ICCCT2.2014.7066736.

[2] N. Meghanathan, "Grid Block Energy based Data Gathering Algorithms for Wireless Sensor Networks," *International Journal of Communication Networks and Information Security*, vol. 2, no. 3, pp. 151-161, December 2010.

[3] C-J. Lin, P-L. Chou and C-F. Chou, "HCDD: Hierarchical Cluster-based Data Dissemination in Wireless Sensor Networks with Mobile Sink," *Proceedings of the International Conference on Wireless Communications and Mobile Computing*, pp. 1189-1194, Vancouver, Canada, July 2006. http://dx.doi.org/10.1145/1143549.1143787.

[4] P. Jesus, C. Baquero and P. S. Almeida, "A Survey of Distributed Data Aggregation Algorithms," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 381-404, September 2014. http://dx.doi.org/10.1109/COMST.2014.2354398.

[5] N. Meghanathan, "A Comprehensive Review and Performance Analysis of Data Gathering Algorithms for Wireless Sensor Networks," *International Journal of Interdisciplinary Telecommunications and Networking*, vol. 4, no. 2, pp. 1-29, April-June 2012. http://dx.doi.org/ 10.4018/jitn.2012040101.

[6] N. Meghanathan, "A Data Gathering Algorithm based on Energy-aware Connected Dominating Sets to Minimize Energy Consumption and Maximize Node Lifetime in Wireless Sensor Networks," *International Journal of Interdisciplinary Telecommunications and Networking*, vol. 2, no. 3, pp. 1-17, July 2010. http://dx.doi.org/10.4018/jitn.2010070101.

[7] D. Tao, S. Tang, X. Li and H. Ma, "Energy Efficient Data Gathering with Mobile Sinks in Hybrid Sensor Networks," *Ad hoc and Sensor Wireless Networks*, vol. 27, no. 1-2, pp. 1-25, 2015.

[8] I. Dietrich and F. Dressler, “On the lifetime of Wireless Sensor Networks,” *ACM Transactions on Sensor Networks*, vol. 5, no. 1, Article # 5, February 2009. http://dx.doi.org/10.1145/1464420.1464425.

[9] J. Liang, J. Wang, J. Cao, J. Chen and M. Lu, “An Efficient Algorithm for Constructing Maximum Lifetime Tree for Data Gathering without Aggregation in Wireless Sensor Networks,” *Proceedings of the 29th IEEE Conference on Computer Communications*, pp. 1-5, San Diego, CA, USA, March 2010. http://dx.doi.org/10.1109/INFCOM.2010.5462181.

[10] N. Meghanathan, "Link Expiration Time and Minimum Distance Spanning Trees based Distributed Data Gathering Algorithms for Wireless Mobile Sensor Networks," *International Journal of Communication Networks and Information Security* (*IJCNIS*), vol. 4, no. 3, pp. 196-206, December 2012.
[11] L. King and N. Meghanathan, "A Weighted-Density Connected Dominating Set Data Gathering Algorithm for Wireless Sensor Networks," *Journal of Computer and Information Science*, vol. 2, no. 4, pp. 3-13, November 2009. http://dx.doi.org/10.5539/cis.v2n4p3.
[12] A. J. Viterbi and J. K. Omura, *Principles of Digital Communication and Coding*, 1st edition, Dover Publications, January 2009.
[13] A. F. Molisch, *Wireless Communications*, 2nd edition, Wiley, December 2010.
[14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, “Introduction to Algorithms,” 3rd edition, MIT Press, 2009.
[15] N. Meghanathan, "A Benchmarking Algorithm to Determine Minimum Aggregation Delay for Data Gathering Trees and an Analysis of the Diameter-Aggregation Delay Tradeoff," *Algorithms*, vol. 8, no. 3, pp. 435-458, July 2015. http://dx.doi.org/10.3390/a8030435.
[16] N. Meghanathan, "An Algorithm to Determine Energy-aware Maximal Leaf Nodes Data Gathering Tree for Wireless Sensor Networks," *Journal of Theoretical and Applied Information Technology*, vol. 15, no. 2, pp. 96-107, May 2010.
[17] R. Xie and X. Jia, "Minimum Transmission Data Gathering Trees for Compressive sensing in Wireless Sensor Networks," *Proceedings of the IEEE Global Telecommunications Conference*, pp. 1-5, Houston, TX, USA, December 2011. http://dx.doi.org/10.1109/GLOCOM.2011.6134304.
[18] U. Jang, S. Lee and S. Yoo, "Optimal Wake-up Scheduling of Data Gathering Trees for Wireless Sensor Networks," *Journal of Parallel and Distributed Computing*, vol. 72, no. 4, pp. 536-546, April 2012. http://dx.doi.org/10.1016/j.jpdc.2012.01.008.
[19] J. Stanford and S. Tongngam, "Approximation Algorithm for Maximum Lifetime in Wireless Sensor Networks with Data Aggregation," *Proceedings of the Seventh ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, pp. 273-277, Las Vegas, NV, June 2006. http://dx.doi.org/10.1109/SNPD-SAWN.2006.22.
[20] C. Zhou and N. F. Maxemchuk, "Distributed Bottleneck Flow Control in Mobile Ad hoc Networks," *Network Protocols and Algorithms*, vol. 3, no. 1, pp. 22-45, 2011. DOI: 10.5296/npa.v3i1.576.
[21] I. A. Najm, M. Ismail, J. Lloret, K. Z. Ghafoor, B. B. Zaidan and A. A. Tareq Rahem, "Improvement of SCTP Congestion Control in the LTE-A Network," *Journal of Network and Computer Applications*, 2015. DOI: http://dx.doi.org/10.1016/j.jnca.2015.09.003.
[22] Y. Wu, S. Fahmy and N. B. Shroff, "On the Construction of a Maximum Lifetime Data Gathering Tree in Sensor Networks: NP-Completeness and Approximation Algorithm," *Proceedings of the 27th IEEE Conference on Computer Communications*, pp. 1013-1021, Phoenix, AZ, USA, April 2008. http://dx.doi.org/10.1109/INFOCOM.2008.80.
[23] N. Meghanathan and I. Dasari, "Performance Comparison Study of Connected Dominating Set Algorithms for Mobile Ad hoc Networks under Different Mobility Models,"

International Journal of Combinatorial Optimization Problems and Informatics, vol. 4, no. 2, pp. 12-30, May-August 2013.
[24] M. E. J. Newman, *Networks*: *An Introduction*, 1st edition, Oxford University Press, Oxford, UK, May 2010.

Copyright Disclaimer