WIRELESS COMMUNICATIONS AND MOBILE COMPUTING
Wirel. Commun. Mob. Comput. 2016; **16**:1694–1712
Published online 3 November 2015 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/wcm.2646

RESEARCH ARTICLE

Tree-based channel assignment schemes for multi-channel wireless sensor networks

Caglar Terzi* and Ibrahim Korpeoglu

Department of Computer Engineering, Bilkent University, Ankara, 06800, Turkey

ABSTRACT

Many sensor node platforms used for establishing wireless sensor networks (WSNs) can support multiple radio channels for wireless communication. Therefore, rather than using a single radio channel for whole network, multiple channels can be utilized in a sensor network simultaneously to decrease overall network interference, which may help increase the aggregate network throughput and decrease packet collisions and delays. This method, however, requires appropriate schemes to be used for assigning channels to nodes for multi-channel communication in the network. Because data generated by sensor nodes are usually delivered to the sink node using routing trees, a tree-based channel assignment scheme is a natural approach for assigning channels in a WSN. We present two fast tree-based channel assignment schemes (called bottom up channel assignment and neighbor count-based channel assignment) for multi-channel WSNs. We also propose a new interference metric that is used by our algorithms in making decisions. We validated and evaluated our proposed schemes via extensive simulation experiments. Our simulation results show that our algorithms can decrease interference in a network, thereby increasing performance, and that our algorithms are good alternatives for static channel assignment in WSNs. Copyright © 2015 John Wiley & Sons, Ltd.

KEYWORDS
wireless sensor networks; multi-channel; tree-based channel assignment; static channel assignment

***Correspondence**
Caglar Terzi, Department of Computer Engineering, Bilkent University, Ankara, 06800, Turkey.
E-mail: terzi@cs.bilkent.edu.tr

1. INTRODUCTION

Wireless sensor networks (WSNs) have applications in a variety of sectors, including industrial automation, the environment, health care, and the military [1,2]. A WSN is composed of a large number of sensor nodes capable of sensing physical or environmental conditions such as temperature, humidity, sound, light, pressure, and so forth. Sensor nodes in a WSN communicate with each other over a radio channel occupying a portion of the spectrum allocated for the wireless technology used by the nodes. Depending on that technology, there may be one or more channels available for a pair of sensor nodes to communicate over. Sensor nodes that use the 802.15.4/ZigBee wireless communication standard, for example, can use one channel if radios operate at 868 MHz, one of 10 channels if radios operate at 915 MHz and one of 16 channels if radios operate at 2.4 GHz [3,4].

When a single channel is used in the network, all nodes need to share the same channel, which may cause interference and packet collisions and decrease aggregate throughput. To decrease interference and support concurrent communication, multiple channels can be utilized if the node platform supports them [5]. Many sensor node platforms available today (i.e., Chipcon CC2420 [6], Nordic nRF905 [7], and Crossbow TelosB [8]) support multiple channels using 802.15.4/ZigBee technology, which enables different channels to be used for different pairs of nodes.

At 2.4 GHz, ZigBee provides 16 channels that can be utilized concurrently, channels 11 to 26, as shown in Figure 1 [9]. The channel width is 2 MHz, and channel separation is 5 MHz; hence, adjacent channels do not overlap (and hence, all channels are orthogonal) and are expected not to interfere with each other. In practice, however, there can be some interference between two adjacent channels, as observed by [10]. Hence, instead of 16, one can consider eight orthogonal ZigBee channels to be utilized in practice.

When multiple channels are available in a WSN, node pairs can use different channels to reduce interference in the vicinity. A single node, however, can still operate on only one channel at a time, unless it has multiple radio transceivers. Reducing interference in the network by using multiple channels can reduce packet corruptions and collisions. Another benefit is concurrent communication.

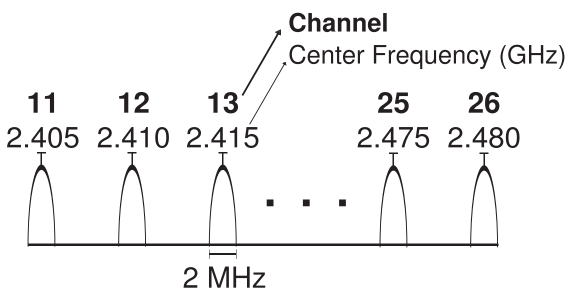


Figure 1. ZigBee channels at 2.4 GHz.

Close node pairs can communicate simultaneously, if they operate on non-overlapping channels.

Multi-channel operation in a WSN can be achieved by dynamic (short term) or static (long term) channel assignment [11]. Short-term channel assignment involves a different channel to be assigned to a communicating pair with every packet or so. Some Medium Access Control (MAC) protocols designed for multi-channel WSNs follow this approach [12,13]. They use time synchronization at the nodes and perform fast channel switching. This method, however, causes a high overhead for resource-constrained sensor nodes, as noted by [10]. Therefore, long-term channel assignments may be more preferable for WSNs, which is the focus of our paper. For such channel assignment in WSNs, because the data are usually gathered from the nodes to the sink using routing trees, tree-based approaches are quite natural. In [10], for example, the authors propose a tree-based channel assignment protocol for WSNs that does not switch channels and does not use time synchronization among nodes.

In this paper, we propose two efficient tree-based channel assignment protocols with the aim of decreasing the network interference. We also propose a new interference metric that is used by our algorithms in making channel assignment decisions. We validate and evaluate the performance of our algorithms via extensive simulation experiments and compare them with other work from the literature. Our results show that our algorithms are effective in decreasing network interference and are feasible to use for static channel assignment in multi-channel wireless sensor networks. We also identify under which conditions network performance can be improved.

As a summary, contributions of the paper can be listed as follows:

- Proposed algorithms are long-term channel assignment algorithms; therefore, they require less frequent channel switching, which decreases power consumption and complexity in sensor nodes.
- Multi-channel networks are considered in the paper. In the simulations, it is shown that proposed algorithms are efficient for different number of available channels (two to eight channels are tested).
- Also, simple performance metrics considering interference are defined to evaluate algorithms.
- Proposed channel assignment algorithms are independent from the physical layer and link layer technologies and can work on a variety of link layers supporting multiple channels.

The rest of the paper is organized as follows: In Section 2, we discuss some related work from the literature. In Section 3, we explore a tree-based channel assignment. In Section 4, we formally present the problem and describe our proposed solution in detail. We describe our simulation environment and present our simulation results in Section 5, and we conclude the paper in Section 6.

2. RELATED WORK

A controlled channel sharing strategy for cognitive radio networks is presented in [14]. In this work, channel assignment for wireless access points is considered. We focus, however, on channel assignment for WSNs.

In [12], Zhou *et al.* propose a channel assignment scheme for multi-channel WSNs and divide their scheme into two parts: frequency assignment and media access. For frequency assignment, in order to reduce interference and hidden terminal problems, nodes within two communication hops are assigned different frequencies, if possible. In media access, neighbor nodes can compete for the medium by using a slotted carrier sense multiple access protocol. The authors also state that packet sizes in WSNs are relatively small; therefore, Request to Send/Clear to Send (RTS/CTS) packets used in wireless ad hoc networks are an overhead and not suitable for WSNs. They focus more on media access aspects, and therefore, their work is different than ours here.

In [13], Zhang *et al.* propose a multi-channel MAC protocol for multi-hop ad hoc networks, which is based on time division multiple access (TDMA). The proposed algorithm requires time synchronization between nodes to switch the channels to a common frequency for communicating pairs. The protocol supports both unicast and broadcast communication. Pal and Nasipuri [15] also proposes a dynamic channel selection scheme, which tries to control overhearing in the network, to decrease the wasted energy in a node that occurs when receiving packets intended for other nodes. In this proposal, nodes temporarily switch their channel to the channel of the receiver in order to send data. In [16], the authors propose a channel assignment scheme that is a hybrid approach of dynamic, static, centralized, and distributed concepts. References [13,15,16] all focus on dynamic channel assignment, whereas we focus on static (long term) channel assignment. Long-term channel assignment schemes avoid the communication and configuration overhead of frequent channel switching and strict time synchronization. Because wireless sensor nodes are resource constrained, we did not prefer the use of a channel assignment scheme that performs fast channel switching, at every time slot, for example.

So and Vaidya [17] proposes a MAC protocol for ad hoc wireless networks that utilizes multiple channels dynamically with the aim of increasing performance. The IEEE 802.11 standard is assumed as the link layer; however, a new MAC protocol is designed rather than using the predefined 802.11 MAC protocol. The latter was originally designed for single-channel operation and does not perform well in a multi-channel scenario because of the multi-channel hidden terminal problem. The work tackles the problem at the data plane, at the MAC layer, whereas we tackle it at the control plane, before the network starts operating.

Jeong *et al.* [18] propose a WSN architecture where the network is divided into clusters and the cluster heads have dual heterogeneous radios. All sensor nodes except the cluster heads have only IEEE 802.15.4 radios. One of the radios in a cluster head is used to communicate with the sensor nodes, and the other radio (802.11 radio) is used to communicate with the other cluster heads. Cluster heads have more processing power and a more powerful battery so as to increase network lifetime. Although the channels used by the 802.11 and 802.15.4 radios are chosen as orthogonal, 802.11 traffic especially causes severe interference with 802.15.4 traffic. The authors state that even a 30-cm distance between radios does not cancel out the interference. They propose an adaptive aggregation and scheduling algorithm for the cluster heads to decrease interference. Our work here does not assume a clustered architecture.

The channel assignment proposed in [19] tries to utilize multiple channels in a WSN to increase throughput in a fair manner. It forms sub-trees, and for each sub-tree, it uses a distinct set of channels. But it uses a separate channel in each level of a sub-tree, and this causes frequent channel switching in a node while receiving packets from children and sending them to the parent. Our scheme, however, assigns channels for longer terms and therefore does not require frequent channel switching. Additionally, [19] requires a slotted operation and therefore is more suitable to run on a TDMA-based MAC protocol, where our scheme is not slotted and can therefore run equally well on both TDMA-based and code division multiple access-based MAC protocols.

Yu *et al.* [20] propose a distributed game-based channel assignment algorithm for WSNs. Game-based channel assignment algorithm takes both network topology and routing information into account. Because it is a distributed algorithm, all sensor nodes are involved in the channel assignment, which may bring more overhead and complexity to sensor nodes, and therefore can cause more power consumption. In our solution, channel assignment algorithm runs in the base station (BS), as a centralized algorithm, to avoid this situation. Channel assignment problem in WSNs is also considered in [21,22]. The difference between these solutions and ours is that our solution considers channel assignment and routing jointly. That is, routing tree of the network is also formed during channel assignment phase. Therefore, a separate routing algorithm is not needed. At the end of our channel assignment, each node in the network knows its parent and can send the packets it receives from its children to this parent. In this way, data from all sensor nodes reach the BS.

In [10], Wu *et al.* propose a tree-based multi-channel protocol for WSNs called GreedyPMIT to decrease network interference. They divide the problem into two main parts: channel assignment and routing. Their proposed solution forms a shortest-path tree, which eliminates the complexity of the routing algorithm, which now simply forwards packets to the parent of each node. They propose a channel assignment algorithm that partitions the network into k trees rooted at the BS where k is the number of available non-overlapping channels. The authors state and show that time synchronization between each node and channel switching bring an overhead to WSNs, because of their limited power processors and clock drifts. Therefore, they avoid using these methods in their algorithm and focus on partitioning the network into trees. The work in [10] uses a tree-based channel assignment and was an inspiration for our work in this paper. Therefore, we provide a detailed explanation of the algorithm in [10] in the next section. Our algorithms, however, apply different techniques in a tree-based channel assignment and therefore are different than that in [10].

3. TREE-BASED CHANNEL ASSIGNMENT

Because channel switching and time synchronization in dynamic channel assignment schemes are significant overheads in WSNs, as described in [10], static channel assignments may be more suitable. Additionally, because tree-based routing is common for WSNs because of the existence of well-defined sink points, a tree-based channel assignment scheme, where the routing tree is divided into sub-trees with different channels, is a natural approach to consider.

In [10], the scheme divides the network into k vertex-disjoint trees rooted at the sink node, that is, the BS, where k is the number of available channels. The value of k depends on the wireless link technology and can be, for example, 8 or 16 for 802.15.4 links. The authors of [10] and our algorithms assume that the BS has k radios to communicate with each tree. Each tree operates on a non-overlapping channel, so there will be no interference between any two trees (zero inter-tree interference). The only interference, then, will be in between the same tree nodes (intra-tree interference), and the aim of the proposed scheme is to minimize that interference. Because our algorithms in this paper have some commonalities with the algorithm in [10], we discuss the algorithm in more detail next.

To minimize intra-tree interference, [10] first defines an interference metric. The authors consider the interference of a node as the interference that can potentially be received by the node from the other same tree nodes in its interference disk. After calculating the interference values of all nodes, the interference value of a tree T is defined as the interference of the non-leaf node with the maximum interference value. The interference values of the leaf nodes are not taken into account because leaf nodes in a WSN do not receive data.

The aim is to partition the network so that the maximum intra-tree interference of all trees will be minimized. Wu *et al*. [10] state this problem as the PMIT problem (*partition* a sensor network into k vertex-disjoint trees with *minimizing* the maximum intra-tree *interference* value of all *trees*) and prove that it is NP-complete. Consequently, they propose a greedy algorithm, named GreedyPMIT, that tries to reduce the maximum intra-tree interference value of all trees.

Algorithm 1 is the pseudocode of the GreedyPMIT algorithm proposed in [10]. It first applies a breadth-first search (BFS) to form a fat tree rooted at the BS (Algorithm 2). That fat tree is the shortest-path tree. Nodes keep their heights in the tree, and they may have multiple parents. From the first level of the fat tree (children of the BS) to the last level, Algorithm 1 assigns channels to the nodes one by one for each level. Then, the nodes on each level are sorted in ascending order by the number of parents in the fat tree. In that sorted order, the algorithm assigns a channel, that is, tree, to that node, which has the least interference after being added. The authors also prove that the complexity of their algorithm is $O(dkn^2)$ in the worst case, where d is the diameter of the graph $G = (V, E)$ (where V is the set of nodes and E is the set of edges indicating direct reachability), n is the number of nodes, and k is the number of channels.

Algorithm 1 GreedyPMIT in [10]

```
Input: $k$ channels, a graph $G = (V, E)$, a root $r$, and the
    interference set of every node.
Output: For each node $u$, $channel_u$ and $parent_u$.
 1: use BFSFatTree algorithm to construct a fat tree rooted
    at $r$.
 2: for each channel $i$ do
 3:    $T_i = r$;
 4: end for
 5: for each node $u$ do
 6:    $channel_u = 0$;
 7:    $parent_u = null$;
 8: end for
 9: $level = 1$;
10: repeat
11:    $node\_list = \{u | height(u) == level; channel_u == 0\}$.
12:    sort node_list in ascending order by the number of
       node's parents.
13:    for each node $u$ in node_list do
14:       find $T_i$, which keeps connected and has the least
          interference after adding $u$.
15:       $T_i = T_i \cup \{u\}$;
16:       $channel_u = i$;
17:       $parent_u = v$, which connects $u$ and has the least
          interference among all nodes in $T_i$.
18:       update the interference value of $T_i$.
19:    end for
20:    $level{+}{+}$;
21: until $level >$ the maximum height of the fat tree
```

4. OUR PROPOSED CHANNEL ASSIGNMENT SCHEMES

To decrease network interference, we propose new greedy tree-based channel assignment algorithms for single-sink WSNs. Our algorithms are long-term channel assignment schemes, and they are centralized. Because the node channels are not changed dynamically, we prefer to use centralized algorithms.

Each sensor network has a sink node (BS) that has higher processing capability. Such a BS with higher processing power is a good location to implement the algorithm, because most of the time, the BS may need to know about the network topology for other purposes anyway, and this information can be used for our channel assignment scheme as well.

Even though topology information is not available at the BS, it can be obtained by broadcasting a topology inquiry message from the BS and collecting the related replies from the sensor nodes using a default channel where all nodes start operating at. Moreover, sensor node locations can be obtained via a distributed localization algorithm. For some applications, nodes can be placed manually, and therefore, node locations can be known at network setup time. In this case, there is no overhead for the BS to learn the network topology.

While collecting topology information, the nodes can operate on a single well-defined default bootstrap channel. After learning the topology of the network, our algorithm can start running in the BS to compute the channels to be assigned to each sensor node. Afterwards, the configuration information for each node can be pushed to the nodes over the default channel, and each node can learn about its configuration (its assigned channel and its parent and children in its routing sub-tree).

Instead of a centralized algorithm, a distributed algorithm could be used. However, a distributed algorithm requires sensor nodes to do extra computation and communication as well, besides the BS, for assigning channels. Consequently, distributed solution would bring more overhead and complexity to sensor nodes and therefore can cause more power consumption. This is not desirable for resource-constrained sensor nodes. Even when we leave computation overhead and power restrictions aside, a distributed algorithm may be less effective, because sensor nodes will be having partial information about the network topology and channel assignment based on this partial information will be causing more interference and collisions. For these reasons, it may be harder, less energy efficient, and less effective to implement channel assignment in a distributed manner in a sensor network. Therefore, we preferred a centralized solution.

We propose a centralized heuristic algorithms because the problem is NP complete, as shown in [10]. The main difference between our algorithms and the algorithm in [10] is the tree union operation, which will be explained

Algorithm 2 BFSFatTree in [10]

```
Input: a graph $G = (V, E)$ and a root $r$.
Output: For every node $u$, its parent set parentSet(u) and
    its height in the tree height(u).
 1: for each node $u$ in $G$ do
 2:     height(u) = MAXIMUM INTEGER;
 3:     parentSet(u) = null;
 4: end for
 5: $S = r$;
 6: height(r) = 0;
 7: for each node $u$ in $S$ do
 8:     for each node $u$'s neighbor $v$ do
 9:         if height(v) > height(u) then
10:             height(v) = height(u) + 1;
11:             parentSet(v) = parentSet(v) $\cup \{u\}$.
12:             $S = S \cup \{v\}$.
13:         end if
14:     end for
15: end for
```

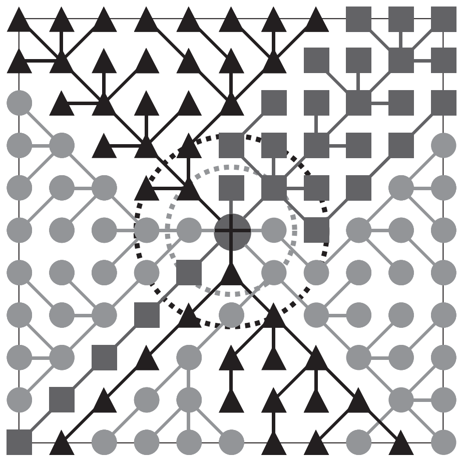


Figure 2. A tree formation example, where lighter and darker dotted circles denote communication range and interference range of base station, respectively.

later in this section. We also define a new interference metric. Our metric takes the physical distance between the same tree nodes within the interference disk into account, rather than just the number of nodes as performed in [10]. Accordingly, the interference value of a node is the sum of interferences received from the same tree nodes falling into its interference disk. More formally, the interference value of a node u is calculated as $\sum_{i=1}^{|N|} \frac{1}{d(u,\upsilon_i)^2}$, where N is the set of the same tree nodes in the interference range of node u; $\upsilon_i \in N$ and $d(u, \upsilon_i)$ is the Euclidean distance between nodes u and υ_i. After calculating the interference values of all nodes, the interference value of a tree T is defined as the interference value of the non-leaf node with a maximum interference value. That is, $int(T) = max\{int(u)$: u is a non-leaf node of $T\}$. We define this additional metric, because according to the signal-to-interference-plus-noise ratio (SINR) physical interference metric [23], the number of nodes in the interference disk and the distance between the nodes are important for the total interference received at a node.

Assigning channels in a greedy manner has some drawbacks. Because the trees are formed greedily, nodes in the same tree can unintentionally be clustered nearby, which causes high interference. An example of this situation is given in Figure 2, which is drawn by our Java simulator. The figure shows a network with 121 nodes, where the communication range and the interference range are set as 1.5 and 2.25 units, respectively. The number of channels (k) is set to three in this example. The channel a node is using is indicated by the shape of the node (square, circle, or triangle). As we can see, there are clusters (blocks) of nodes that use the same channel: At the top right of the figure, there is a block of square-shaped trees; at the top left, there is a block of triangle-shaped trees; and at the sides, there are blocks of circle-shaped trees. These blocks increase the interference between the trees, and hence in the network, because all or most of the nodes in a node's interference range become part of the same tree, which increases the interference value of the tree.

Our main aim is to decrease the number and size of such blocks, which will also help decrease network interference. To overcome this problem, we define two new algorithms, called bottom-up channel assignment (BUCA) and neighbor count-based channel assignment (NCCA). Each of these algorithms has two versions. The first versions (called BUCA-N and NCCA-N) use a default (node count based) interference metric as used in [10], that is, the number of nodes in the interference disk, to make decisions. The second versions (called BUCA-D and NCCA-D) use the distance-based interference metric, explained before, to make decisions. Similarly, we refer to the original GreedyPMIT algorithm with a node-count-based interference metric as GreedyPMIT-N and to the GreedyPMIT with the new interference metric, that is, a distance-based interference metric, as GreedyPMIT-D.

To decrease the number and size of the blocks (clusters), we want to initially form more than k trees and then carefully unite those trees into k trees to decrease the negative effects of the greedy approach. The number of initial trees can be as much as the number of the BS's one-hop neighbors. Therefore, we initially form c trees, where c is the number of children (one-hop neighbors) BS has.

Our algorithms are run by the BS, which is more powerful than the other nodes. Because the BS knows the network topology, other nodes do not consume energy when the algorithm is run and the final tree is formed. After forming the final tree, the BS sends the configuration information to the network. This action begins at the root of each sub-tree and then propagates down to the descendants.

As described in Algorithms 3 and 4, BUCA-N and BUCA-D form the trees from bottom to top. Level 1 is the top level, that is, the closest level to BS. In Algorithm 3, like GreedyPMIT, first, a BFS fat tree is formed using Algorithm 2. However, unlike GreedyPMIT, we start from the last level of the fat tree to assign channels to the nodes. For each level from bottom to top, the nodes are sorted in ascending order by the number of parents in the fat tree. Then, in that sorted order, the algorithm assigns a channel, that is, tree, to that node to reduce overall interference. We also consider the distance between a node and its parent, as well as the number of children a parent has.

Like GreedyPMIT, NCCA-N and NCCA-D form trees from top to bottom. Different from GreedyPMIT, however, NCCA considers c the number of initial trees, instead of k (channel count) the number of initial trees. All of our four algorithms (NCCA-N, NCAA-D, BUCA-N, and BUCA-D) form c (child count of BS) the number of initial trees. For example, c is equal to 8 in Figure 2. Then, they decrease the number of trees to k by uniting them. Therefore, for NCCA, we give c as the channel number input to GreedyPMIT (Algorithm 1). The rest is similar to GreedyPMIT, except the last part. After the last line of GreedyPMIT, line 21, we calculate union interference (*unionInterf*) matrix, which stores the interference values of the trees, which may unite with each other, that is, possible pairs. Then we call the function, which unites trees, that is, *UniteTrees*. If the number of available channels (k) is greater than or equal to the number of initially formed trees (c), there will be no union operation. In this case, $(k - c)$ channels cannot be used because the BS can only communicate with c nodes. The union operation tries to minimize the maximum interference, as all our algorithms do; the difference between BUCA and NCCA is the forming procedure of the initial c number of trees.

We assign channels to nodes after tree construction part. The reason for this is to avoid assigning the same channels to sub-trees that are close to each other. Because we do not know the future growing positions of sub-trees during the tree construction, it is possible to assign the same channels to sub-trees that will come close to each other in the subsequent steps of tree construction.To control this in a better manner, we first construct the trees and then unite them considering their layout.

Algorithm 5 explains the union operation. At first, we have c trees, and we try to decrease them to k trees. To do this, we propose an iterative approach, which decreases the total number of trees step by step by pairing, that is, uniting, the trees in each iteration. Before pairing, we calculate the number of pairs needed for each iteration, as explained in Algorithm 6. We want to unite the initial c number of unpaired trees as $N_1, N_2, \ldots, N_k$, assuming $|N_i - N_j| \leq 1$ for any i, j to form a balanced network, where N_i denotes the number of initial trees paired (united) to form the final tree T_i, and $\sum_{i=1}^{k} N_i = c$. We calculate the number of pairs according to the assumption in the *CalculatePairs* method because this assumption is needed to do so and the final tree may not ensure this assumption when uniting the residual trees in Algorithm 5. Detailed explanations of Algorithms 5 and 6 are given in Section 4.1.

We pair the trees as described in Algorithm 7 so that interference is reduced, and move to the next iteration with fewer but bigger trees. This algorithm takes as input

Algorithm 3 BUCA

```
Input: k channels, a root r, a graph $G = (V, E)$, and the interference set of every node.
Output: set of c initial trees T, where c is the number of neighbors of root r, $channel_u$, and $parent_u$ for every node u.
1: use BFSFatTree algorithm described in [10] to construct a fat tree rooted at r.
2: for each node u do
3:     $channel_u = 0$; $parent_u = null$; $addedChildrenSize(u) = 0$;
4:     childrenSize(u) = # of children count in BFS fat tree;
5: end for
6: $c = 0$;
7: for each neighbor n of r do
8:     $T_c = \{r, n\}$; $channel_n = c$; $parent_n = r$; c++;
9: end for
10: level = maximum level of BFS fat tree;
11: repeat
12:     $node\_list = \{u | height(u) == level; channel_u == 0\}$.
13:     sort node_list in ascending order by the number of node's parents.
14:     for each node u in node_list do
15:         $parents = \{p | p \in parentSet(u)\}$.
16:         $onlyChild = \{p | p \in parents; childrenSize(p) == 1\}$.
17:         if $onlyChild \neq \emptyset$ then
18:             $farthestParent = \{p | p \in onlyChild; distance(u, p) == max\}$.
19:             AddChild(u, farthestParent[random available index]).
20:         else
21:             $noAddedChildren = \{p | p \in parents; addedChildrenSize(p) == 0\}$.
22:             if $noAddedChildren \neq \emptyset$ then
23:                 $minChild = \{p | p \in noAddedChildren; childrenSize(p) == min\}$.
24:             else
25:                 $minInterf = \{p | p \in parents; interf(p,u) == min\}$.
26:                 $minChild = \{p | p \in \text{minInterf}; \text{childrenSize(p)} == \min\}$.
27:             end if
28:             $farthestParent = \{p | p \in minChild; distance(u,p) == max\}$.
29:             AddChild(u, farthestParent[random available index]).
30:         end if
31:     end for
32:     level−−;
33: until $level > 1$
34: calculate unionInterf matrix, where unionInterf(i,j) denotes the interference value of tree i when it is united with tree j.
35: UniteTrees(unionInterf, k, c, T).
```

Algorithm 4 AddChild

```
Input: child $u$ and parent $p$.
Output: $T_i$, $channel_u$, $parent_u$, $addedChildrenSize(p)$.
 1: if $DummyT_u == \emptyset$ then
 2:     $DummyT_u = \{u\}$;
 3: end if
 4: if $DummyT_p == \emptyset$ then
 5:     $DummyT_p = \{p\}$;
 6: end if
 7: if $p$ is not a level 1 node then
 8:     $DummyT_p = DummyT_p \cup DummyT_u$;
 9: else
10:     find $T_i$ where $p \in T_i$.
11:     $T_i = T_i \cup DummyT_u$;
12:     $channel_u = i$;
13: end if
14: $parent_u = p$;
15: $addedChildrenSize(p){+}{+}$.
```

Algorithm 5 UniteTrees

```
Input: $unionInterf$ matrix, where $unionInterf(i,j)$ denotes
    the interference value of tree $i$ when it is united with
    tree $j$, $k$ is the number of available channels, $c$ is the
    number of BS's neighbors, and $T$ is the set of trees.
Output: final tree $T$.
 1: if $k < c$ then
 2:     $neededPairs = CalculatePairs(c, k)$;
 3:     $level = 0$;
 4:     while $neededPairs > 0$ do
 5:         $T = MarkAndPair(T, unionInterf, neededPairs, level)$;
 6:         $neededPairs = CalculatePairs(neededPairs, k)$;
 7:         $level{+}{+}$;
 8:     end while
 9:     if $\lfloor \frac{c}{k} \rfloor \geq 2$ then
10:         for $i = level \rightarrow 0$ do
11:             for each tree $T_i$ in $residuals[i]$ do
12:                 unite $T_i$ with a tree in $T$ that has the least
                    interference after adding $T_i$.
13:             end for
14:         end for
15:     end if
16: end if
```

the *unionInterf* matrix, where $unionInterf(i,j)$ denotes the interference value of tree i when it is united with tree j. We mark the elements in this matrix increasingly, until we can pair p pairs, where p denotes the number of pairs needed for each iteration. We repeat this procedure in Algorithm 5, until we reach k trees, that is, when *neededPairs* is equal to 0. For some iterations, we have residual trees that are not paired. At the end, in reverse order, that is, starting with the residual tree from the last iteration, we add these residual trees to one of the k trees formed previously, as explained in lines 9 to 15 in Algorithm 5, to minimize the global maximum interference. Finally, we will have k trees, with hopefully less interference. A detailed explanation of Algorithm 7 is presented in Section 4.2.

Algorithm 6 CalculatePairs

```
Input: number of unpaired trees $upt$, number of available
    channels $k$.
Output: $neededPairs$.
 1: $minBranchedTree = \lfloor \frac{upt}{k} \rfloor$.
 2: $remainedBranches = upt \% k$;
 3: if $minBranchedTree \% 2 == 1$ then
 4:     $maximumEvenNo = (minBranchedTree - 1) \times k + remainedBranches \times 2$;
 5: else
 6:     $maximumEvenNo = minBranchedTree \times k$;
 7: end if
 8: $neededPairs = maximumEvenNo/2$.
```

In Figure 2, we showed an inefficient channel assignment example, which produces blocks of the same tree nodes. We presented our algorithms, which try to overcome this problem and produce better results. In Figure 3, we see the same network when our BUCA-N is used, and we observe that the trees are distributed in the network more uniformly than they were in Figure 2.

Proposed channel assignment schemes, NCCA and BUCA, are independent of the physical and link layer technologies and protocols used in the sensor networks. Therefore, they can work properly with a variety of MAC protocols. Hence, we are not specifying and fine-tuning the schemes for a particular MAC protocol. This is one of the advantages of our approach that is using long-term scheduling: Channel assignment becomes less coupled with the MAC protocol used. Table I summarizes the differences between our four algorithms and GreedyPMIT.

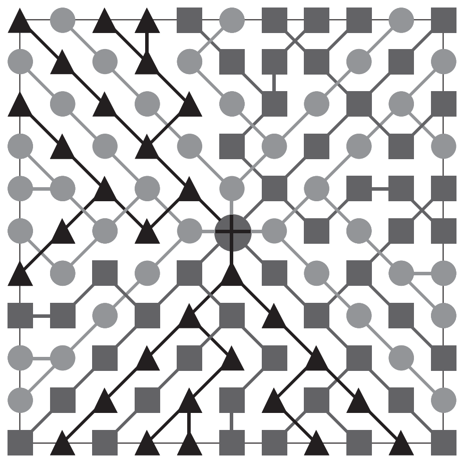


Figure 3. Example tree formation when bottom-up channel assignment-N is used.

Table I. Comparison of our proposed algorithms and GreedyPMIT.

Algorithms	Initial tree formation	No. of trees formed initially	Unites initial trees?	Interference m
GreedyPMIT-N	Top-down	k	No	No. of Nodes
GreedyPMIT-D	Top-down	k	No	Distance between nodes
NCCA-N	Top-down	c	If ($k < c$)	No. of nodes
NCCA-D	Top-down	c	If ($k < c$)	Distance between nodes
BUCA-N	Bottom-up	c	If ($k < c$)	No. of nodes
BUCA-D	Bottom-up	c	If ($k < c$)	Distance between nodes

NCCA, neighbor count-based channel assignment; BUCA, bottom-up channel assignment.

Algorithm 7 MarkAndPair

```
Input: set of trees $T$, $unionInterf$ matrix (where $unionInterf(i,j)$ denotes the interference value of tree $i$ when it is united with tree $j$ and $p$ denotes the number of pairs needed), and level $l$.
Output: paired trees $T$, residual trees $residuals$.
1: $maxPairs = 0$;
2: $max = 0$;
3: $count = 0$;
4: while $maxPairs < p$ do
5:     set every element of $mins$ array with $MAXIMUM\ INTEGER$.
6:     for $i = 0 \rightarrow T.length$ do
7:         for $j = 0 \rightarrow T.length$ do
8:             if $i \neq j$ and $unionInterf[i][j] > max$ and $unionInterf[i][j] < mins[i]$ then
9:                 $mins[i] = unionInterf[i][j]$;
10:            end if
11:        end for
12:    end for
13:    sort the $mins$ array in ascending order.
14:    if $count == 0$ then
15:        $max = mins[2 \times p - 1]$;
16:    else
17:        $max = mins[0]$;
18:    end if
19:    for $i = 0 \rightarrow T.length$ do
20:        for $j = 0 \rightarrow T.length$ do
21:            if $i \neq j$ and $unionInterf[i][j] \leq max$ then
22:                $marked[i][j] = true$;
23:            end if
24:        end for
25:    end for
26:    $maxPairs$ = number of trees in maximum matching calculated by using marked elements.
27:    $count{+}{+}$;
28: end while
29: unite the trees according to maximum matching.
30: $residuals[l]$ = set of unmatched trees in this level $l$ (if any).
```

4.1. Uniting trees

Figure 4 shows an example of the union operation, where the number of children of BS (c) $= 16$ and the number of available channels (k) $= 3$. As performed in the first step of *CalculatePairs* (Algorithm 6), we calculate $\lfloor \frac{c}{k} \rfloor$ as $\lfloor \frac{16}{3} \rfloor = 5$ and try to divide the three trees as a union of five initial (Step 1) trees, that is, 5-5-5. Then we must add the number of remaining trees, which is $16\%3 = 1$ (only one tree); consequently, our three trees will be united as a 6-5-5 number of initial trees. This is the expected final version of the trees. We unite the trees so that the difference between the number of united trees for each final tree is minimal. That is, in this example, it is $6 - 5 = 1$. However, these numbers are used only for calculating the number of pairs for each step, and at the end, the trees can be formed as 7-5-4 or another combination if that results in the minimum interference. In this figure, we assume that the final tree is a 6-5-5 tree. The next step in this iteration is to calculate the number of pairs needed, which is the greatest even number for each tree, which is less than or equal to the united trees' count. Therefore, we take 4 for the trees with united tree counts equal to 5 and take 6 for the other one. Then we add those even numbers and divide by 2 to find the number of pairs. Calculating $\frac{6+4+4}{2}$, we find that we need seven pairs for this iteration, as described in Algorithm 6. Finally, we need to pair these seven trees so that the interference will be minimal. The pairing operation is explained in *MarkAndPair* (Algorithm 7). We can assume that the pairing operations in Figure 4 are carried out according to that algorithm, which will be explained in the next section.

In the second step, the trees in the first step are paired according to Algorithm 7, and we obtain seven paired trees and two residual trees (Trees 11 and 14). The residual trees are trees that have no pair in this level. These two trees come from rounding the two 5s to 4 in the previous step, and they were left unpaired to be used in the following steps. Similar to the previous step, we calculate the number of pairs with a result of three.

In the third step, we obtain $k = 3$ trees, where each is a union of four trees. We also obtain a residual tree in this level and save it, as in the previous step. In this step, we reach $k = 3$ trees; therefore, we do not need to calculate the pairs in the following step (which can be calculated as 0 from Algorithm 6). We now need to unite the residual trees to form the final tree. When doing so, we unite them in reverse order. We unite bigger residual trees (the trees from the previous steps) first, because they are more prone to increasing interference. As seen in Figure 4, we unite the residual tree of step (c) in the fourth step and the residual trees of step (b) in the last step to minimize the final interference. Finally, we have three trees that are a union of 16 trees, and a 6-5-5 formation can be seen from the united trees, where two trees are a union of five step 1 (initial) trees and the other tree is the union of six step 1 trees.

4.2. Marking and pairing trees

To determine which tree will be paired with which, we must first calculate the interference values when tree i is paired (united) with tree j, for all (i,j) pairs. Then, we form a matrix M, where the value in $M(i,j)$ is equal to $M(j,i)$, which is the interference value when tree i is united with tree j, or vice versa. We also label self-interference, $M(i,i)$ as X, which will not be processed in any step in the algorithm. The procedure of choosing pairs is described in Tables II–IV with an example where we have seven trees to be united, and the number of pairs needed to be formed in this iteration is three. Therefore, at the end, six trees will have unique pairs, and the other tree will be a residual tree.

To pair the trees with minimal interference, we develop a straightforward method. First, we find the minimum interference values in each row and sort them in ascending order. In the example, the minimum numbers are 13, 13, 14, 16, 15, 17, and 14 in the row order. The ascending order sorted version of them is 13, 13, 14, 14, 15, 16, and 17. Because we need three pairs, we first select the first $3 \times 2 = 6$th minimum number and check if we can pair three trees by using it; therefore, we remove 17 from the list. To do this, we set the maximum interference max as the sixth minimum number, that is, 16. The second step is to mark the elements of the matrix that are less than or equal to $max = 16$, as shown by the bold elements in Table II. Later, we check whether we can form three pairs with all unique elements with marked elements. By "all unique," we mean that if a tree i is paired with a tree j, then it cannot be paired with any tree again; therefore, the remaining pairs cannot contain tree i or j.

We check how many trees can be paired in our simulations with a brute-force approach; hence, the approach has $O((r-1)!)$ complexity in the worst case, where r is the number of rows in the matrix. This pairing problem can be defined as finding the maximum matching in a non-bipartite graph. We can consider the interference matrix as a graph $G = (V, E)$, where V is the set of trees and E is the set of edges between them. There is an edge between two trees i and j if $M(i,j)$ is marked (bold). Then the matching is the set of edges in the graph with unique vertices. Maximum matching is a matching that has the maximum

Table II. Demonstration of marking, step 1.

	1	2	3	4	5	6	7
1	X	**13**	**14**	19	**15**	21	**14**
2	**13**	X	26	**16**	18	17	19
3	**14**	26	X	18	22	32	25
4	19	**16**	18	X	23	19	20
5	**15**	18	22	23	X	40	33
6	21	17	32	19	40	X	25
7	**14**	19	25	20	33	25	X

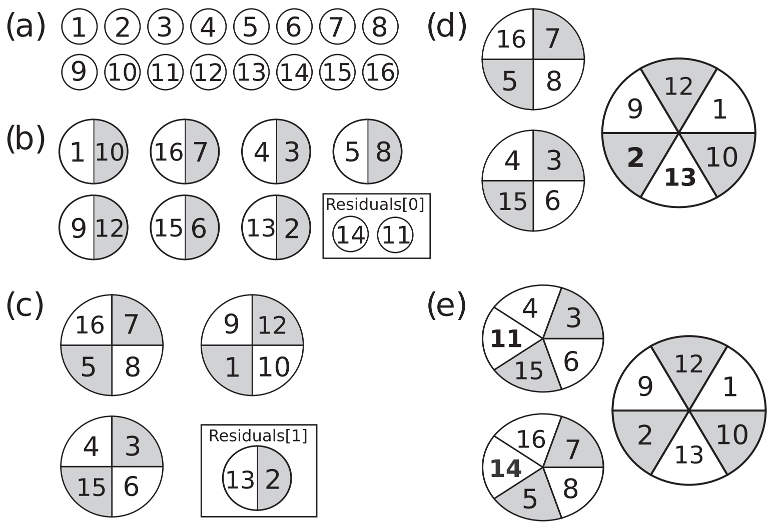


Figure 4. Demonstration of uniting trees. (a) $c = 16, k = 3, \lfloor c/k \rfloor = 5, c\%k = 1$, divided as 6-5-5, $(6+4+4)/2 = 7$ pairs and (b) $c = 7, k = 3, \lfloor c/k \rfloor = 2, c\%k = 1$, divided as 3-2-2, $(2+2+2)/2 = 3$ pairs. (c) Now, we have three trees; then we will unite residual trees to those trees in reverse order in steps (d) and (e): (d) Residual tree in step (c) is added (denoted with bold numbers), and (e) residual trees in step (b) are added (denoted with bold numbers).

number of edges possible. This definition is the same as our maximum-pair definition, where we calculate the maximum pairs in the given marked trees and compare the result with the number of pairs needed. If we reach the number of pairs needed, we pair the trees according to the maximum matching. Although our solution in our simulations has a $O((r-1)!)$ running time, this time can be decreased to $O(|V|^4)$ by using Edmond's maximum matching algorithm [24] or to $O(|E|\sqrt{|V|})$ by using Micali and Vazirani's maximum matching algorithm [25].

By using marked trees, we can calculate the maximum number of pairs using one of the matching algorithms. For example, the maximum number of pairs that can be formed according to Table II is two. One example of these two pairs is {1,3} and {2,4}. As evident in the table, if {1,2} were chosen as a pair, we would not be able to find second pair; therefore, either 3, 5, or 7 should be chosen as the pair of 1 to find a maximum matching in this step.

The iteration described will be repeated until the number of maximum matchings is greater than or equal to the number of pairs needed. In the next iteration, we again find the minimum interference values in each row; however, we skip the marked interference values and do not take them into account while finding this minimum. Then, in these *r* minimum interference values, we find their minimum and set them as *max*. Then we mark all the elements in the matrix that are equal to *max*. In the example, *max* is equal to 17, which is the minimum number in all unmarked elements. Therefore, we mark all the elements in the matrix that are equal to 17, as seen in Table III. Next, we calculate the maximum matchings with the given marked trees. Only the pair {2,6} is added to the previous pair list, which does not help form the third pair because as we mentioned before, Tree 2 should be paired with Tree 4 in order to form two pairs; therefore, we cannot use the newly added pair {2,6} and cannot form three pairs in this iteration.

Now, we need to find the next minimum unmarked interference value and calculate the maximum matching. The next value is 18; therefore, we set *max* as 18 and mark the values that are equal to 18. The new matrix in this iteration can be seen in Table IV. And now, we can have three pairs with these marked trees; one example of the maximum matchings is {1,7}, {2,5}, and {3,4}, with a residual Tree 6. As evident in the table, another maximum matching can be formed as {1,5}, {2,6}, and {3,4}, with a residual Tree 7. Our algorithms choose one of the maximum matchings according to the algorithm used for finding them.

Table III. Demonstration of marking, step 2.

	1	2	3	4	5	6	7
1	X	**13**	**14**	19	**15**	21	**14**
2	**13**	X	26	**16**	18	**17**	19
3	**14**	26	X	18	22	32	25
4	19	**16**	18	X	23	19	20
5	**15**	18	22	23	X	40	33
6	21	**17**	32	19	40	X	25
7	**14**	19	25	20	33	25	X

Table IV. Demonstration of marking, step 3.

	1	2	3	4	5	6	7
1	X	**13**	**14**	19	**15**	21	**14**
2	**13**	X	26	**16**	**18**	**17**	19
3	**14**	26	X	**18**	22	32	25
4	19	**16**	**18**	X	23	19	20
5	**15**	**18**	22	23	X	40	33
6	21	**17**	32	19	40	X	25
7	**14**	19	25	20	33	25	X

4.3. Complexity of our proposed algorithms

Because the PMIT problem is NP complete, the proposed algorithm in [10] is greedy and has a time complexity of $O(dkn^2)$ in the worst case, where d is the diameter of the graph $G = (V, E)$, where V is the set of nodes and E is the set of edges indicating direct reachability; n is the number of nodes; and k is the number of channels, as explained before. Our algorithms also have a polynomial time complexity. The time complexity of GreedyPMIT stems from the tree formation part; however, in our case, the complexity of the tree unions should also be considered. We have a similar time complexity to GreedyPMIT in the initial tree formation part; however, rather than forming k trees, we initially form c trees.

As shown in Algorithm 5, we repeat calling *MarkAndPair* and *CalculatePairs* methods while *neededPairs* are greater than 0. In each iteration, *neededPairs* decreases by pairing two trees, and we put the residual trees (if any) into a *residuals* array. Because we are dealing only with the paired trees and not the residuals in the while loop, the number of iterations in that while loop depends on only the paired trees, and *neededPairs* is calculated according to the paired trees. In the worst case, *neededPairs* can be decreased into $\lfloor \frac{neededPairs}{2} \rfloor$ in each iteration, because we can have at most that number of pairs. Therefore, the while loop executes at most $O(\log c)$ times, where c is the number of neighbors in the BS's communication range because it is the first and greatest parameter that *CalculatePairs* takes. Because *CalculatePairs* runs in constant time, we should examine the running time of the *MarkAndPair* method.

In Algorithm 7 (*MarkAndPair*), the operations outside the while loop run in constant time; therefore, we should calculate the running time of the while loop. The first loop in lines 6 to 12 runs in at most $O(c^2)$ time, because *T.length*, that is, the number of trees in the network, can be at most c. The for loop in lines 19 to 25 also runs in $O(c^2)$. The critical part of this algorithm is finding the maximum pairs from the marked elements, performed in line 26. This pairing problem is defined as finding the maximum matching in a non-bipartite graph, and the running time of finding the maximum pairs in line 26 can be decreased to $O(|V|^4)$ by using Edmond's maximum matching algorithm [24] or to $O(|E|\sqrt{|V|})$ by using Micali and Vazirani's maximum matching algorithm [25], as described before. The maximum matching operation is repeated until the number of maximum matchings is greater than or equal to the number of pairs needed p. In every iteration, in the worst case, we can mark only two entries *marked*[i][j] and *marked*[j][i] at line 22 and calculate the maximum matching again. Therefore, in the worst case, $O(c^2)$ iterations are needed to reach to p maximum matchings. Consequently, running time of *MarkAndPair* algorithm can be calculated as $O(c^2 \times c^4) = O(c^6)$, when Edmond's algorithm [24] is used for finding the maximum matching, because the complexity of algorithm is $O(|V|^4)$, and $|E|$ can be equal to c in maximum. If we use Micali and Vazirani's maximum matching algorithm [25] with $O(|E|\sqrt{|V|})$ complexity, running time of *MarkAndPair* algorithm will be calculated as $O(c^2 \times c^2 \times \sqrt{c}) = O(c^{4.5})$, because $|V|$ can be c and $|E|$ can be $\frac{c^2}{2}$ in maximum. Because Micali and Vazirani's maximum matching algorithm [25] gives a faster running time, we will prefer their solution for calculating the complexity.

Because the *MarkAndPair* algorithm is called $O(\log c)$ times in the worst case in Algorithm 5, the overall worst-case running time of the *UniteTrees* algorithm will be $O(c^{4.5} \log c)$ when Micali and Vazirani's algorithm [25] is used. Because we form c trees initially and use the same algorithm as GreedyPMIT in NCCA-N and NCCA-D and a similar one in BUCA-N and BUCA-D, the time complexity of the tree formation is $O(dcn^2)$. Consequently, our final complexity will be $O(c^{4.5} \log c + dcn^2)$.

5. PERFORMANCE EVALUATION

In this section, we present the simulation environment and the simulation experiments we performed to evaluate our algorithms (BUCA and NCCA). We compare our algorithms with GreedyPMIT [10] and show the improvements. At the end of this section, we also analyze the complexity of our algorithms.

5.1. Simulation environment and scenarios

We coded a custom simulator in Java and ran our simulations in a Linux machine with an eight-core 64-bit processor and 4-GB memory. In the simulations, the sink node (BS) is placed in the middle of the sensing field, and sensor nodes are placed around it. The nodes are positioned so that every node has four nodes around it (except the nodes on the edges and corners), the left, right, top, and bottom. The distance between these four nodes and the center node is defined as one unit of distance. The network topology is a grid containing nodes at the intersection points, as shown in Figure 3. We assume 2.4-GHz ZigBee channels are used. Hence, the channel model we use is the same with ZigBee channel model. Low power transmitter, low bit rate and short range are typical properties of ZigBee channels. We assume power attenuates in proportion with the square of the separation between transmitter and receiver. We repeat each simulation experiment 100 times and take the average of these runs.

In the experiments, we vary the number of nodes, the communication range, the interference range, and the number of channels available to evaluate the performance of our algorithms. Although we set the communication range as 1-unit, 1.5-unit, and 2-unit distances, we only demonstrate the results for 1.5 and 2 units because the results of our algorithms are mostly the same as GreedyPMIT's when we set the communication range to one unit. For one unit case, because the communication range is low, the nodes do not have many options when selecting their parents, that is, tree, and therefore, the formed trees in both algorithms are about the same, which results in a minor performance difference.

Another important simulation parameter is the interference range, which is defined as 1.5 times of the communication range in GreedyPMIT [10]. For a fair comparison, we also set our interference range at 1.5 times the communication range. The other important variable is the number of nodes in the network. We choose an $x \times x$ network and choose x as an odd number, which locate the BS exactly in the middle of the network. The variable x is set to odd numbers between 11 and 33, inclusively; therefore, the number of nodes is set to 121, 169, 225, 289, 361, 441, 529, 625, 729, 841, 961, and 1089.

The final variable is the available number of channels, which is also equal to the number of trees in the network. Although there are 16 channels in 2.4 GHz in ZigBee, [10] demonstrates that adjacent channels cause interference and decrease overall throughput. For that reason, it is better to use non-adjacent channels. That is, a maximum of eight channels can be used. Although adjacent channels can be used if the nodes are separated by a sufficient distance, by using only non-adjacent channels, we guarantee that any two different-channel trees will have no inter-tree interference even if they are close to each other. Consequently, we choose the number of channels to be between two and eight inclusively. As expected, network interference increases with fewer channels in our algorithms and GreedyPMIT. In the performance evaluation, we present the actual interference values and the performance improvement compared with GreedyPMIT. The performance improvement is shown with the interference decrease in percentages in Figures 5 to 9, and the actual interference values can be seen in Figures 10 to 13. We define interference decrease as the difference between interference incurred when GreedyPMIT is used and interference incurred when our algorithm is used. Therefore, there can be negative interference decrease in case GreedyPMIT causes lower total interference than our algorithm. However, this case is rare in our simulations. In general, our algorithms perform better than GreedyPMIT, and we have positive interference decrease in most cases.

As a reminder, the interference value of a tree is defined as the interference value of the non-leaf node with the maximum interference value.

5.2. Simulation results

We perform two sets of simulations to compare our algorithms with GreedyPMIT. In the first set of simulations, we compare maximum interference value of the network,

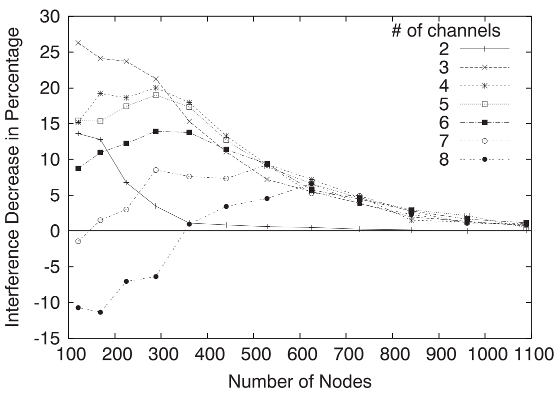


Figure 5. Comparison of GreedyPMIT-N and bottom-up channel assignment-N when communication range is equal to 1.5 units. The *y*-axis is the interference decrease, that is, performance improvement, of our algorithms against GreedyPMIT.

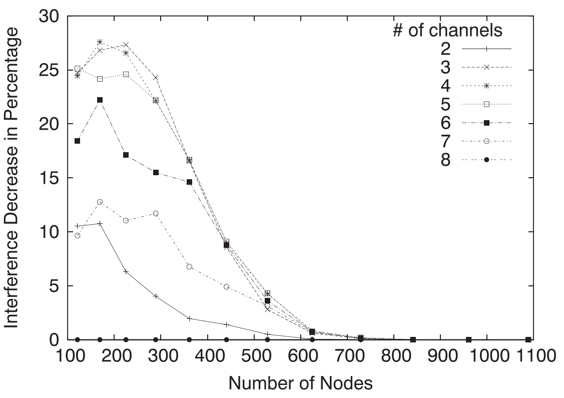


Figure 6. Comparison of GreedyPMIT-N and neighbor count-based channel assignment-N when communication range is equal to 1.5 units.

which is described before. Additionally, packet drop rates using SINR metric are compared in the second set of simulations. Because our algorithms are independent of the MAC protocol, we thought that a custom simulator is sufficient to simulate the network for finding the maximum interference values of the nodes according to the neighbor counts and distances and calculating packet drop rates considering SINR threshold. In this way, we focused better on topology and distance effects in the overall sensor network.

5.2.1. Simulations with maximum interference value.

In this section, we present and discuss our simulation results. We interpret our results based on the value of the communication range variable and also based on the value of the interference range indirectly. We also compare our six algorithms with respect to the distance-based interference metric, which we think it is more accurate than the node-count metric.

The first part of the simulation results (Figures 5 and 6) is for the case where the communication range is set to be 1.5 units. As explained earlier, the interference range is fixed and set as 1.5 times of communication range. Hence, it is set to 2.25 units. When we set the communication range as 1.5 units, a node can have at most eight nodes in its one-hop communication range and 20 nodes in its interference range. This situation can be observed in Figure 2, where the inner dashed circle denotes the node's communication disk and the outer dashed circle denotes its interference disk. With a 1.5-unit communication range, the BS's number of neighbors, that is, the value of c, will also be at most eight. As explained in the previous section, we examine our four algorithms against GreedyPMIT-N and GreedyPMIT-D while changing the number of available channels to between two and eight inclusively.

Figure 5 demonstrates the comparison between GreedyPMIT-N and our algorithm BUCA-N when the communication range is set to 1.5 units. The performance increase in the figure can be observed from the *y*-axis, which is the interference decrease in percentage when compared with GreedyPMIT-N. This figure shows that (except for the case where the channel count is two) when the number of channels increases, the performance for smaller networks decreases because the number of union operations decreases. For example, the number of union operations will be five for the three-channel case and only one for the seven-channel case. Further, there will be no union operation when the number of channels is eight when communication range is equal to 1.5 units. These results show that the union operation plays a big role in performance of our algorithms. When the number of nodes increases, the performance depending on channel count will be nearly equal and will diverge to 0, that is, a similar performance to GreedyPMIT. The reason for the performance decrease when channel count is two is likely due to the high interference. With two available channels, we have only two trees; therefore, interference may not improve much.

In Figure 6, we present the performance improvement of Algorithm NCCA-N against GreedyPMIT-N when the communication range is 1.5 units. First, we notice that the eight-channel case shows no improvement compared with GreedyPMIT-N because for the 1.5-unit communication range, the BS will have eight neighbors, that is, $c = 8$, and our algorithms form c trees in the same way as GreedyPMIT-N. Then our algorithms unite the trees to decrease the number of trees to k. In the eight-channel scenario, both c and k are equal to eight; consequently, there will be no union operation, and the interference value of the network will be the same as GreedyPMIT.

As stated before, using two channels in BUCA-N (Figure 5) does not show much improvement, although it has more union operations than the others, that is, six operations. This property is also seen in Figure 6.

The other similarity between BUCA-N and NCCA-N is that networks with fewer channels (three, four, and five) perform better when we compare them with six-channel,

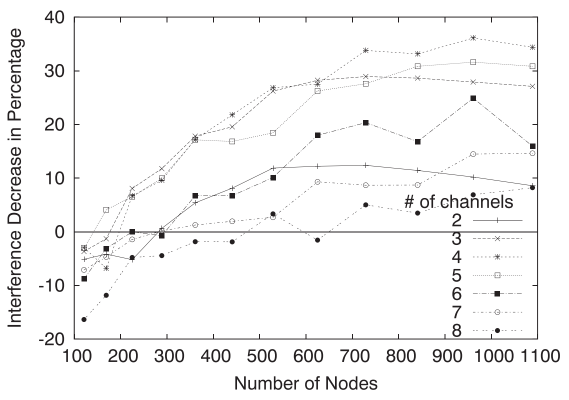


Figure 7. Comparison of GreedyPMIT-N and bottom-up channel assignment-N when communication range is equal to two units.

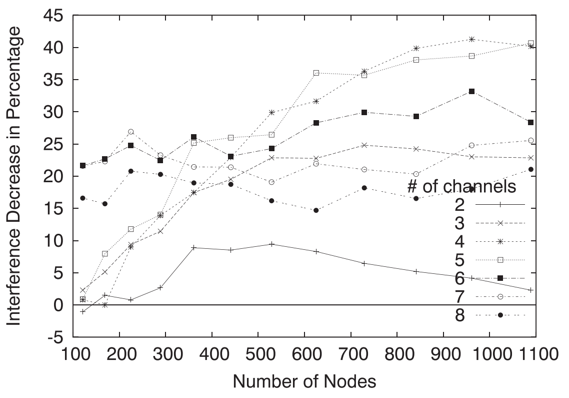


Figure 8. Comparison of GreedyPMIT-N and neighbor count-based channel assignment-N when communication range is equal to two units.

seven-channel, and eight-channel cases. The latter cases have two, one, and zero union operations, and the former cases have five, four, and three union operations, respectively. A difference between BUCA-N and NCCA-N is that, except for the two-channel case, the performance increase in NCCA-N is higher in smaller sized networks; however, performance decreases faster when network size increases, and reaches 0 when node size is equal to 729. In these simulations, the performance improvement can be as much as 30% compared with GreedyPMIT.

We also did simulation experiments with a communication range of two units (Figures 7 to 9). In these experiments, there were 12 other nodes in a node's one-hop vicinity; therefore, the BS has 12 neighbors. Further, the number of nodes in a node's interference range is 28, with an interference range equal to $2 \times 1.5 = 3$ units. On the other hand, the number of nodes in the communication range and the interference range is 8 and 20, respectively, for a communication range of 1.5 units. Consequently, a higher communication range causes an increase in interference because of the increase in the number of nodes in the interference range. In the following set of simulation results, we observe the performance of our algorithms when the communication range is two units.

Figure 7 compares the performance between GreedyPMIT-N and BUCA-N when the communication range is two units. This figure shows a very different result than Figure 5, which represents the comparison between the same algorithms when the communication range is 1.5 units. In Figure 5, performance decreases when the number of nodes increases; however, in Figure 7, where the communication range is two units, we observe the opposite result, where an increase in the number of nodes results in an increase in performance. The number of nodes in one node's interference range is more than in the previous scenario, and this number plays an important role in the interference, because it seems that it is an overhead for smaller networks. However, when the number of nodes increases, the effect of that overhead also decreases, and BUCA-N starts to perform better than GreedyPMIT-N.

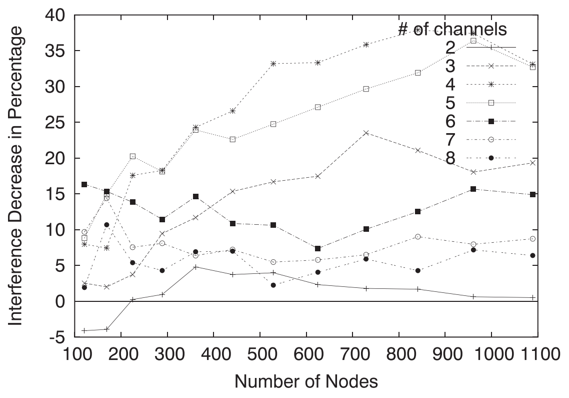


Figure 9. Comparison of GreedyPMIT-D and neighbor count-based channel assignment-D when communication range is equal to two units.

As evident in the figure, except for the eight-channel case, our algorithm performs better than GreedyPMIT when the number of nodes is greater than or equal to 361; eight-channel case starts to perform better in larger networks. If we omit the two-channel case again, the higher performance with fewer channels (three, four, and five) is evident. Although larger networks show no improvement in performance for the 1.5-unit communication range, our algorithms achieve remarkable improvements with a two-unit communication range: an up to 36% performance.

Figure 8 compares the performance between NCCA-N and GreedyPMIT-N when the communication range is two units. Generally, Figure 8 shows better results than Figure 7 for smaller and larger networks alike. First, there is no negative result, except for in a two-channel case with 121 nodes. In other cases, our algorithm performs better than GreedyPMIT and achieves as much as a 40% performance increase.

Figure 9 compares the performance of GreedyPMIT-D with NCCA-D when the communication range is two units, which is similar to Figure 8. We can state that NCCA-D performs generally better than GreedyPMIT by achieving a positive interference decrease. Similar to the comparison of GreedyPMIT-N with NCCA-N with a communication range of two units, the two-channel and six-channel to eight-channel cases show more stable results when compared with the three-channel to five-channel cases, where performance increases when network size increases. And again, the two-channel case does not show much performance improvement because of the high interference caused by two physically close trees. The simulations with a communication range of two units (Figures 7 to 9) generally perform better than GreedyPMIT. We think the fluctuations in the results are due to the increased number of the same tree nodes in a node's vicinity, which is caused by increased communication range.

Next, we present simulation experiments when a distance-based interference metric is used for calculating the final interference (Figures 10 to 13). Because we take GreedyPMIT as our base algorithm, algorithms with a suffix -N use the number of nodes in the interference disk, which is the default interference metric of GreedyPMIT. Although using this metric is plausible and easy for implementation, the distance between the nodes in the interference disk is also important in the real world, which is why we propose the distance-based interference metric. Therefore, we compare the interferences of the final trees formed from all six algorithms, where four are new algorithms proposed by us, using the distance-based interference metric, regardless of their tree formation metrics. Although the algorithms with suffix -N use the number of nodes as their interference metric, their final performance will be compared by using the distance-based interference metric, to simulate the real world, where not only the number of nodes but also the distance between them are significant. Unlike the previous figures in this section, now, we show the actual interference values in the network instead of the performance increase. For the actual interference

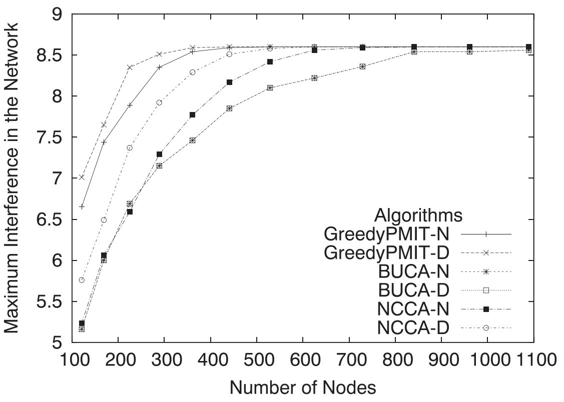


Figure 10. Comparison of all algorithms, when the final interference is calculated using the distance-based interference metric, where $k = 4$ and communication range is equal to 1.5 units. BUCA, bottom-up channel assignment; NCCA, neighbor count-based channel assignment.

values, the performance is higher if the interference value is lower.

Figure 10 shows the final maximum interferences of the networks with different numbers of nodes when the final interference is calculated using the distance-based interference metric, regardless of the metric used in the tree formations. In this figure, the communication range is 1.5 units, and the available number of channels, that is, k, is 4. The results are very similar for channel counts between two and eight, inclusively, when the communication range is 1.5 units. Therefore, we show only the results of the four-channel case, where the difference between each algorithm is more evident. Because we show that our proposed algorithms (BUCA and NCCA) generally perform better than GreedyPMIT, the interference values of GreedyPMIT-N are higher than BUCA-N and NCCA-N, and the interference values of GreedyPMIT-D are higher than BUCA-D and NCCA-D, as shown in Figure 10, where higher interference indicates lower performance. As shown in Figures 5 and 6, there is no performance improvement for larger networks when the communication range is 1.5 units; therefore, the interference values for larger networks are similar to each other in this figure.

In Figure 10, for the algorithms other than BUCA-N and BUCA-D, networks with more than or exactly 841 nodes reach an 8.6 interference value, which is the maximum interference that can be achieved when the communication range is set as 1.5 units and when all the nodes in the interference disk are the same tree nodes for at least one node in the network. This is the worst-case tree formation, and it apparently cannot be avoided for larger networks.

An interesting result for Figure 10 shows that although NCCA-N and GreedyPMIT-N do not form trees according to a distance-based interference metric, their interference values are lower than their distance-based interference metric versions NCCA-D and GreedyPMIT-D, respectively, in smaller networks. Although this result was not expected, it can be explained with the greedy property of the algorithms, where the best parent of a node for a level is chosen when the trees are formed, and cannot be changed in subsequent levels.

In three parts, we demonstrate the results of the simulation experiments, where the communication range is equal to two units, because the results in that case are not similar at different k values, unlike in the previous case (communication range = 1.5 units). This case is examined in three channel cases, where $k = 3, 5,$ and 7, and in increasing order so as to better observe the effect of the number of channels on the algorithms' performances. We start with the results of the three-channel case.

Figure 11 shows the final maximum interferences of the networks when the final interference is calculated using the distance-based interference metric when $k = 3$ and the communication range is equal to two units. GreedyPMIT-N and GreedyPMIT-D again have the highest interference values. Contrary to Figure 10, Figure 11 shows that the algorithms with the distance-based interference metric, suffix -D, exhibit better or equal performances when

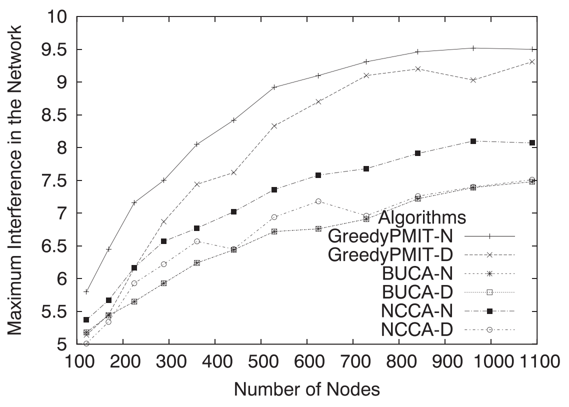


Figure 11. Comparison of all algorithms, when the final interference is calculated using distance-based interference metric, where $k = 3$ and communication range is equal to two units. BUCA, bottom-up channel assignment; NCCA, neighbor count-based channel assignment.

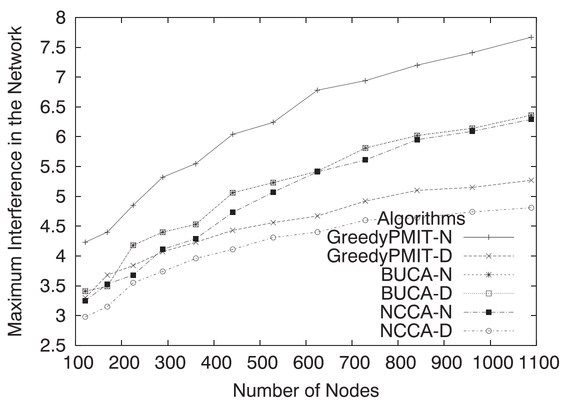


Figure 13. Comparison of all algorithms, when the final interference is calculated using the distance-based interference metric, where $k = 7$ and communication range is equal to two units. BUCA, bottom-up channel assignment; NCCA, neighbor count-based channel assignment.

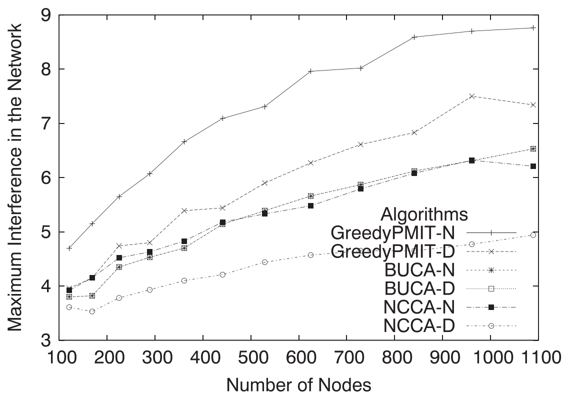


Figure 12. Comparison of all algorithms, when the final interference is calculated using the distance-based interference metric, where $k = 5$ and communication range is equal to two units. BUCA, bottom-up channel assignment; NCCA, neighbor count-based channel assignment.

compared with the same-named algorithms with suffix -N, as expected. Also, only GreedyPMIT algorithms can reach interference values close to the maximum interference, which is approximately 9.55, when the communication range is two units. The performance increase can be better observed in this figure because the interference differences between GreedyPMIT algorithms and our algorithms are greater.

Figure 12 shows the final maximum interferences of the networks when the final interference is calculated using the distance-based interference metric, when $k = 5$ and the communication range is equal to two units. We can observe the actual interference differences between the three-channel and five-channel cases by examining Figures 11 and 12, respectively. As expected, the interference values in a five-channel case are lower than those in the three-channel case because the network is divided into more trees in the former case, and consequently, the number of nodes per tree decreases, which lower interference values than in the three-channel case. Because we have more trees in this figure, the maximum interference values in the network do not reach the exact maximum interference value. We also observe that the performance of the BUCA algorithms is lower than that of the NCCA-D algorithm when the number of available channels increases.

Figure 13 shows the final maximum interferences of the networks when the final interference is calculated using the distance-based interference metric, when $k = 7$ and the communication range is equal to two units. The decrease in interference values according to the increase in the available number of channels can be clearly observed in Figures 11 to 13. Also notable are the performance decrease of the BUCA algorithms and the performance increase of GreedyPMIT-D when the number of available channels increases.

Our algorithms NCCA and BUCA perform better than GreedyPMIT and achieve performance increase up to 40%. However, which one is better depends on other parameters and conditions. For communication range of two units (Figures 7 to 9), both NCCA and BUCA perform similar results. For 1.5-unit communication range (Figures 5 and 6), NCCA has better results for smaller networks; however, BUCA performs better when network size increases. As stated before, BUCA has some improvements to the GreedyPMIT and directly to the NCCA. Besides the tree formation order (bottom-up versus top-down), BUCA also considers the children count of possible parent nodes and distance between of a node and its possible parents. We can observe the positive effect of these improvements for larger networks. When the number of nodes increases, the importance of the child–parent distance and the nodes' children count emerges.

5.2.2. Simulations with signal-to-interference-plus-noise ratio metric.

In addition, we propose a SINR metric that is similar to the distance metric (algorithms with suffix -D). The difference between the distance and SINR metrics is that the latter one considers SINR values of all same tree nodes in the network, not only the nodes within an interference disk as considered in the former one. We simulate this case, and the results are presented in Figures 14 and 15. It is evident in these figures that our algorithms perform better than GreedyPMIT in most cases. Our algorithms provide performance increase up to 22% when compared with GreedyPMIT.

To analyze our algorithms better and compare it with GreedyPMIT, we compare the packet drop rates of these algorithms (Figures 16 and 17) as well. For these simulations, we set the number of nodes as 121 and communication range as 1.5 units. For analyzing drop rates, we consider SINR, which affects the success probabilities of packets. We consider packets received with SINR value lower than a threshold as dropped.

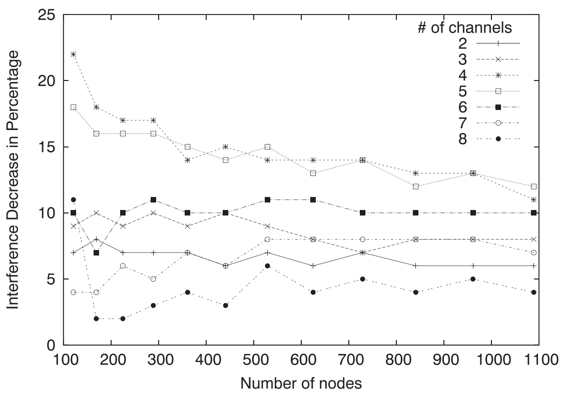


Figure 14. Comparison of GreedyPMIT with bottom-up channel assignment with signal-to-interference-plus-noise ratio metric when communication range is equal to 1.5 units.

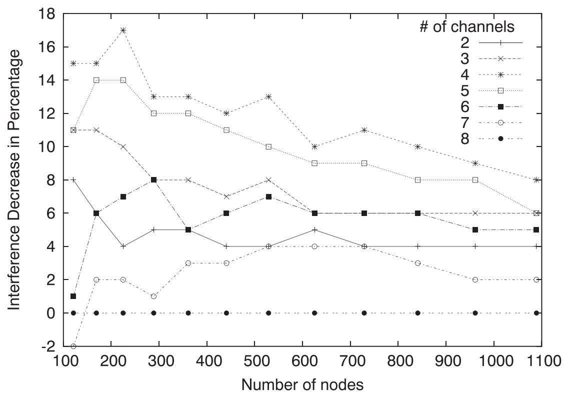


Figure 15. Comparison of GreedyPMIT and neighbor-count-based channel assignment with signal-to-interference-plus-noise ratio metric when communication range is equal to 1.5 units.

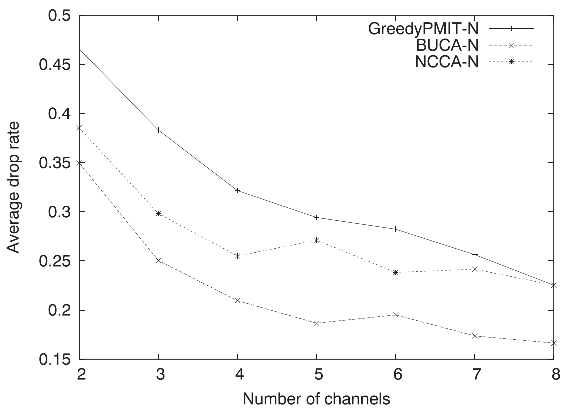


Figure 16. Average drop rates of GreedyPMIT-N, bottom-up channel assignment (BUCA)-N, and neighbor count-based channel assignment (NCCA)-N when the number of nodes is equal to 121 and communication and interference ranges are equal to 1.5 units.

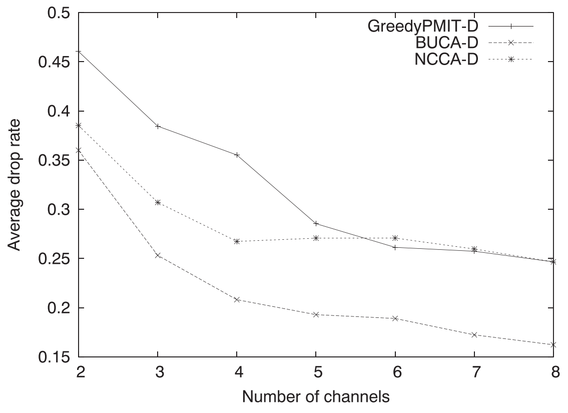


Figure 17. Average drop rates of GreedyPMIT-D, bottom-up channel assignment (BUCA)-D, and neighbor count-based channel assignment (NCCA)-D when the number of nodes is equal to 121 and communication and interference ranges are equal to 1.5 units.

Figures 16 and 17 show that our algorithms are generally better than GreedyPMIT when drop rates are compared. In these figures, the decrease in the drop rates is evident when the number of channels increases. Because the channels are orthogonal, this is an expected result. Another result is the lower drop rates of BUCA when compared with NCCA and GreedyPMIT.

5.3. Complexity analysis

Also, we compare the time complexities of our algorithms and GreedyPMIT. For comparison, we calculate the results of the functions inside Big-O notations, that is, dkn^2 for GreedyPMIT and $max(c^{4.5} \log c, dcn^2)$ for our algorithms.

Although $c^{4.5}\log c$ complexity seems to be high, dcn^2 had a higher result for each c and n values in our simulations. Because c only depends on the communication range, our algorithms have the same complexity result for a communication range value, independent from the number of available channels, k. Then, the complexity comparison between our algorithms and GreedyPMIT becomes $O(dcn^2)$ versus $O(dkn^2)$, respectively, according to our simulation parameters. Figures 18–21 show the results of $O(f(x))$ values for complexities of our algorithms and GreedyPMIT. As described before, c can be 12 in maximum in our simulations, and k can be 2 in minimum; therefore, GreedyPMIT runs six times faster than our algorithms in worst case, when comparing the results of the functions inside Big-O notations. In best case, our algorithms and GreedyPMIT have the same running time, as

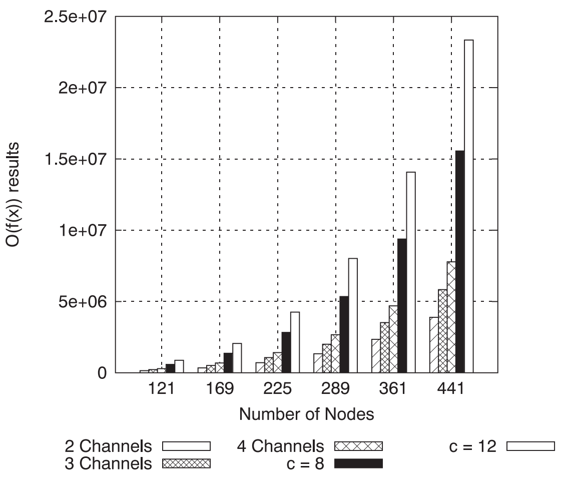


Figure 18. Complexity comparison of our algorithms versus GreedyPMIT. 2-4 Channels denote GreedyPMIT with respective available channels, c = 8 and c = 12 denote our algorithms with communication range is equal to 1.5 and 2 units, respectively.

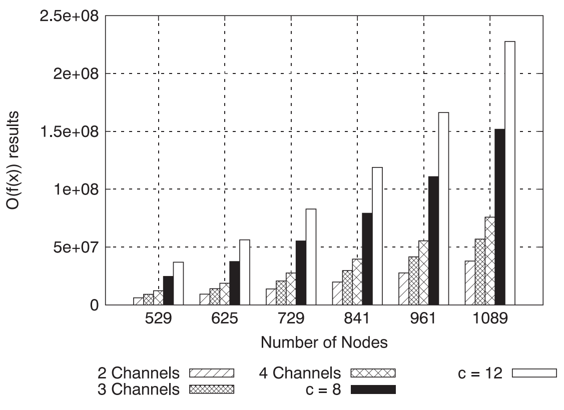


Figure 19. Complexity comparison of our algorithms versus GreedyPMIT. 2-4 Channels denote GreedyPMIT with respective available channels, c = 8 and c = 12 denote our algorithms with communication range is equal to 1.5 and 2 units, respectively.

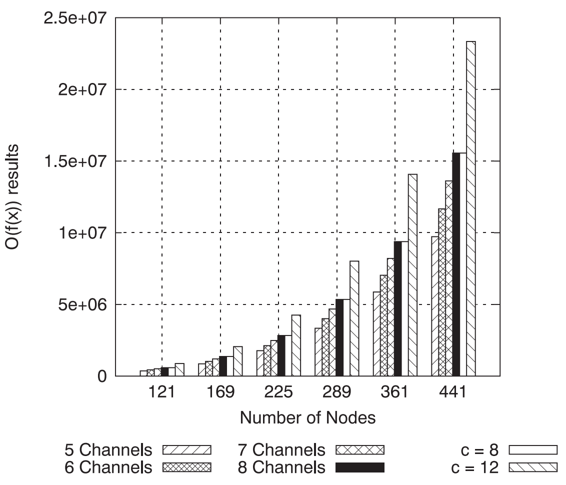


Figure 20. Complexity comparison of our algorithms versus GreedyPMIT. 5-8 Channels denote GreedyPMIT with respective available channels, c = 8 and c = 12 denote our algorithms with communication range is equal to 1.5 and 2 units, respectively.

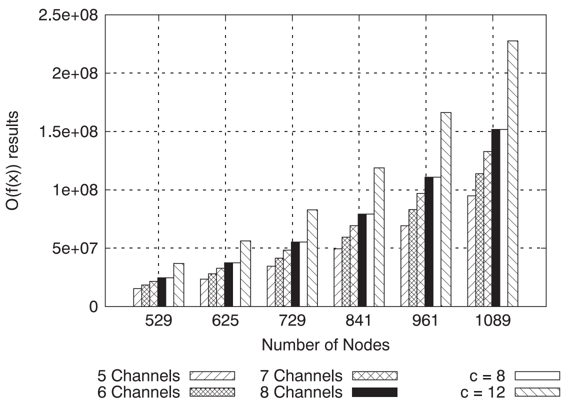


Figure 21. Complexity comparison of our algorithms versus GreedyPMIT. 5-8 Channels denote GreedyPMIT with respective available channels, c = 8 and c = 12 denote our algorithms with communication range is equal to 1.5 and 2 units, respectively.

seen from Figures 20 and 21 , when $k = 8$ and communication range is equal to 1.5, that is, $c = 8$. Tree formation is carried out once at the beginning, and the same tree formation will be used for a long time. Also, our algorithms provide performance increase up to 40% when compared with GreedyPMIT; therefore, our algorithms can be preferred for assigning channels, although GreedyPMIT runs faster.

6. CONCLUSIONS

In this paper, we propose static channel assignment algorithms that can be used to decrease interference in multi-channel wireless sensor networks. In this way, we aim to increase the network throughput and reduce packet collisions and packet corruptions. We propose two greedy tree-based channel assignment algorithms (BUCA and NCCA) with two versions each. We use a tree-based approach as in [10] and partition the network into trees, where each node is connected to a tree and a channel is assigned to each tree. By assigning each tree the same channel, the only interference in the network can be intra-tree interference, and we decrease this by carefully uniting the initially formed trees. We also propose a distance-based interference metric to use in making channel assignment decisions.

To evaluate the performance of our algorithms, we performed extensive simulation experiments for various values of the number of channels, transmission range, and the number of nodes in the network. We also compare packet drop rates and complexities of our algorithms with GreedyPMIT. The results show that our algorithms can decrease interference and packet drop rates and, also, they can provide better network performance. Hence, they are good candidates for static channel assignments for wireless sensor networks capable of using multiple channels. We also analyzed under which condition performance can be improved.

ACKNOWLEDGEMENT

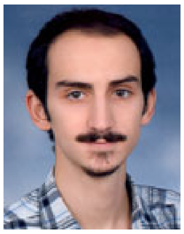
This work is supported in part by The Scientific and Technological Research Council of Turkey (TUBITAK) with project 113E274.

REFERENCES

1. Akyildiz I, Su W, Sankarasubramaniam Y, Cayirci E. Wireless sensor networks: a survey. *Computer Networks* 2002; **38**: 393–422.
2. Yick J, Mukherjee B, Ghosal D. Wireless sensor network survey. *Computer Networks* 2008; **52**: 2292–2330.
3. ZigBee Specification, 2008. http://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2011/kjb79_ajm232/pmeter/ZigBee [accessed on March 18 2014].
4. IEEE 802.15.4-2003 Standard, 2003. http://standards.ieee.org/getieee802/download/802.15.4-2003.pdf [accessed on March 18 2014].
5. Tragos EZ, Zeadally S, Fragkiadakis AG, Siris VA. Spectrum assignment in cognitive radio networks: a comprehensive survey. *Communications Surveys & Tutorials* 2013; **15**: 1108–1135.
6. CC2420 2.4 GHz IEEE 802.15.4 ZigBee-ready RF Transceiver, 2007. http://www.ti.com/lit/ds/symlink/cc2420.pdf [accessed on March 18 2014].
7. nRF905 Single Chip 433/868/915 MHz Transceiver, 2008. http://www.nordicsemi.com/eng/content/download/2452/29528/file/Product_Specification_nRF905_v1.5.pdf [accessed on March 18 2014].
8. TelosB Datasheet. http://bullseye.xbow.com:81/Products/Product_pdf_files/Wireless_pdf/TelosB_Datasheet.pdf [accessed on August 15 2013].
9. Graphical Representation of Wi-Fi Channels in the 2.4 GHz Band. http://en.wikipedia.org/wiki/File:2.4_GHz_Wi-Fi_channels_(802.11b,g_WLAN).svg. [accessed on March 18 2014].
10. Wu Y, Stankovic JA, He T, Lin S. Realistic and efficient multi-channel communications in wireless sensor networks. In *INFOCOM The 27th Conference on Computer Communications*, Phoenix, 2008; 1193–1201.
11. Incel OD. A survey on multi-channel communication in wireless sensor networks. *Computer Networks* 2011; **55**: 3081–3099.
12. Zhou G, Huang C, Yan T, He T, Stankovic JA, Abdelzaher TF. MMSN: multi-frequency media access control for wireless sensor networks. In *INFOCOM 2006 Proceedings of the 25th IEEE International Conference on Computer Communications*, Barcelona, 2006; 1–13.
13. Zhang J, Zhou G, Huang C, Son SH, Stankovic JA. TMMAC: an energy efficient multi-channel MAC protocol for ad hoc networks. In *IEEE International Conference on Communications (ICC '07)*, Glasgow, 2007; 3554–3561.
14. Bansal T, Li D, Sinha P. Opportunistic channel sharing in cognitive radio networks. *IEEE Transactions on Mobile Computing* 2014; **13**: 852–865.
15. Pal A, Nasipuri A. A distributed channel selection scheme for multi-channel wireless sensor networks. In *Proceedings of the 13th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '12)*, Hilton Head Islands, 2012; 263–264.
16. Prajapati A, Ganesan S. S-CAS: smart channel assignment scheme for wireless sensor networks. In *IEEE International Conference on Electro/Information Technology (EIT)*, Normal, 2010; 1–6.
17. So J, Vaidya NH. Multi-channel MAC for ad hoc networks: handling multi-channel hidden terminals using a single transceiver. In *Proceedings of the 5th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, Tokyo, 2004; 222–233.
18. Jeong Y, Kim J, Han SJ. Interference mitigation in wireless sensor networks using dual heterogeneous radios. *Wireless Networks* 2011; **17**: 1699–1713.
19. Yang Y, Liu Y, Ni LM. Level the buffer wall: fair channel assignment in wireless sensor networks. In *IEEE 6th International Conference on Mobile Adhoc and Sensor Systems (MASS '09)*, Macau, 2009; 208–216.
20. Yu Q, Chen J, Fan Y, Shen X, Sun Y. Multi-channel assignment in wireless sensor networks: a game theoretic approach. In *INFOCOM 2010 Proceedings IEEE*, San Diego, 2010; 1–9.

21. Yen HH, Lin CL. Integrated channel assignment and data aggregation routing problem in wireless sensor networks. *Communications, IET* 2009; **3**: 784–793.
22. Chen J, Yu Q, Chai B, Sun Y, Fan Y, Shen X. Dynamic channel assignment for wireless sensor networks: a regret matching based approach. *IEEE Transactions on Parallel and Distributed Systems* 2015; **26**: 95–106.
23. Jain K, Padhye J, Padmanabhan VN, Qiu L. Impact of interference on multi-hop wireless network performance. In *Proceedings of the 9th Annual International Conference on Mobile Computing and Networking*, San Diego, 2003; 66–80.
24. Edmonds J. Paths, trees, and flowers. In *Classic Papers in Combinatorics*, Gessel I, Rota GC (eds). Birkhauser Boston: New Jersey, 1987; 361–379.
25. Micali S, VazIrani VV. An o($\sqrt{|v||e|}$) algorithm for finding maximum matching in general graphs. In *21st Annual Symposium on Foundations of Computer Science*, 1980; 17–27.

AUTHORS' BIOGRAPHIES



Caglar Terzi is currently a PhD candidate in the Computer Engineering Department at Bilkent University, Turkey. He received his MS and BS degrees from Bilkent University, both in Computer Engineering, in 2012 and 2010, respectively. His research interests are algorithm and protocol design for wireless networks, especially for wireless sensor networks. He is a student member of ACM and IEEE.

Ibrahim Korpeoglu received his PhD and MS degrees from University of Maryland at College Park, both in Computer Science, in 2000 and 1996, respectively. He received his BS degree (summa cum laude) in Computer Engineering from Bilkent University in 1994. Since 2002, he has been a faculty member in the Department of Computer Engineering at Bilkent University. He is currently an associate professor. Before joining Bilkent, Dr. Korpeoglu worked in several research and development companies in the USA, including Ericsson, IBM T.J. Watson Research Center, Bell Laboratories, and Bell Communications Research (Bellcore). He received the Bilkent University Distinguished Teaching Award in 2006 and the IBM Faculty Award in 2009. He is a member of ACM and a senior member of IEEE.