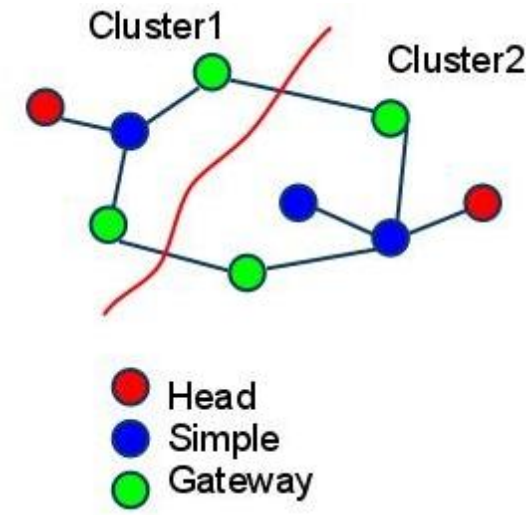
Modular MaxMind Clustering Algorithm on Wiselib

Technical Report

Dimitrios Amaxilatis Computer Engineering & Informatics Department University Of Patras (amaxilatis@ceid.upatras.gr)

I. Clustering

Clustering is the procedure used to group a number of similar things in order to achieve a common goal. This goal may be either a complex computation, which requires more computing power than every single node can offer, or collecting data (sensor data for example) from a wide geographic area. The concept of *similarity* may be specified using many different parameters related to the specific problem we are trying to solve, such as *proximity* to the rest of the nodes, *computing* and *processing ability*, *mobility* inside the area of the network, *energy* requirement for the operation of the node or any *combination* of the above parameters. Also with this grouping of the network we achieve a level of *role assignment* inside the clusters and the network in order to *reduce the volume of traffic* generated, to control and synchronize the networks function.



With clustering each node inside the network assumes a certain role and along with that role some privileges and some obligations in order to achieve the common goal. Those roles can be easily distinguished, and are:

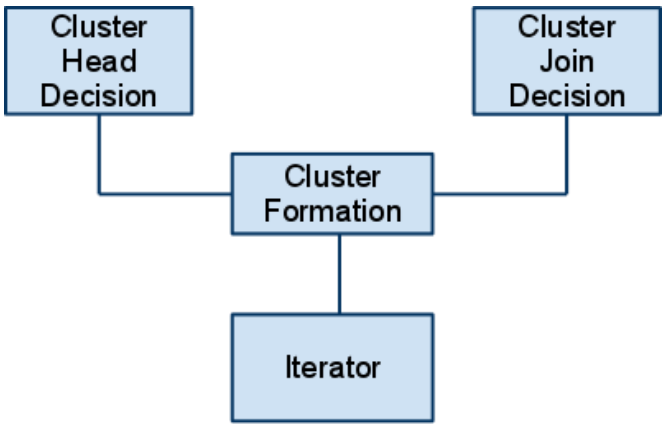
- Leader or cluster_head is the node that is responsible for each one of the clusters.
- Simple nodes are those that have no special role inside the clusters
- Gateway nodes that are located on the outskirts of the clusters and connect them to the rest of the network.

Using this role assignment communication inside the cluster uses the paths between the node in question and the cluster_head, avoiding any extra messages. Also, communication between clusters consists of gateway nodes' message exchange with package routing by the cluster_head that knows the whole cluster's structure and connections.

Although clustering as a procedure has been extremely studied in the past and many algorithms have been proposed for grouping inside computer networks (each of them using selected parameters), there are only a few implementations available for use in real networks .

When studying a clustering algorithm we can distinguish 4 parts that can easily be separated and considered as independent procedures. These procedures can be designated as the selection of cluster_headers (or cluster_head decision), the selection of the cluster the node will choose to join (or cluster_join decision), an iterator that maintains most important information about the network's and cluster's status and composition, as well as a main part that connects the 3 other modules called cluster_formation. Cluster_formation is responsible for synchronizing the other 3 modules and is the one that communicates with the hardware responsible for sending and receiving messages from the network. The structure of a clustering algorithm can therefore be presented as in Figure 1 where every part is a separate module that offers an interface to communicate with the others while it communicates directly only with cluster formation.

Figure 1



II. Wiselib (website)

Wiselib is a software library for wireless sensor networks that intends to implement basic generic algorithms (hardware independent) and can be imported and used on different devices without any modifications. To achieve this Wiselib uses the C++ programming language and the features it offers as template functions (for data type independent programs), inline functions, callbacks and software timers. The main goal is to introduce a level of transparency between the implementation of the algorithm and the real executable that will be used.

At the moment the devices that are fully supported by Wiselib are iSense and the wireless sensor network simulator Shawn that was used during the development of Maxmind algorithm.

III. Shawn (website)

Shawn is a wireless sensor network simulator implemented using C++. Shawn offers the developer a great amount of freedom to modify the network on which the program's behavior is simulated. It offers abilities such as using network topologies imported from files (using xml) as well as getting the results of the simulation and the status of the network after the execution of the algorithm in files(using xml again). Also using Shawn we can test our implementation on a big

variety of simulation topologies and scenarios to check every individual case that may cause potential problems. Finally we can generate images of the network's status (pdf or ps format) any time, in addition to the option for live view of the algorithms execution (using a refreshing image snapshot of the network) during each synchronous round of the simulation.

IV. Modular MaxMind Algorithm (original paper)

The Maxmind clustering algorithm uses each node's unique id to form groups as we want. Formation of the clusters is based on flooding and the resulting clusters have a maximum cluster_head to gateway node distance of d hops.

Execution of the algorithm can be divided in 5 stages:

For starters (stage 1 "floodmax" *rounds 1 to d*) runs for d rounds using flooding during which every node informs the rest of the network for the maximum node id known so far. For each round every node sends a message with his so far maximum id (called the "winner") to every one of his neighbors. When all messages are received for round i, the ids received are processed and nodes decide their new winner value as the maximum of all ids received and the previous winner value.
Then (stage 2 "floodmin" *rounds d+1 to 2*d*) a second flooding stage is executed exactly as before for d rounds. The value used this time for winner is calculated as the minimum of all ids received and the previous winner value. To achieve that nodes use at first the last winner value of floodmax. After all messages are received for round i, ids are processed and the new winner is selected.
Also during the first two stages (rounds 1 to 2*d) of the algorithm every node keeps the id of the node that sent the winner id in a separate array (called "sender").
The following stage is **determining the cluster heads (stage 3 "cluster head election" *round 2*d*)**. During round 2d each node decides who will be his cluster_head and which cluster he will join. To decide the cluster heads nodes uses the values gathered during the first 2d rounds (the winner and sender arrays) and choose according to the following rules:

1. Each nodes searches the values winner[d+1] to winner[2d] for its own id. In case he finds his id then he declares himself a cluster head of his own cluster. If his id was not found the algorithm continues to rule 2
2. Each node searches for node pairs inside the winner array. A node pair is found when a winner value of the first d rounds is found as a winner value of the second d rounds as well. If there are more than one node pairs the pair with the minimum id is selected. If a pair was found then the node joins this cluster else if no pair was found then the algorithm continues to rule 3
3. If there was no cluster decision after rules 1 or 2 then the node chooses the maximum node id from the first d rounds and joins this cluster.

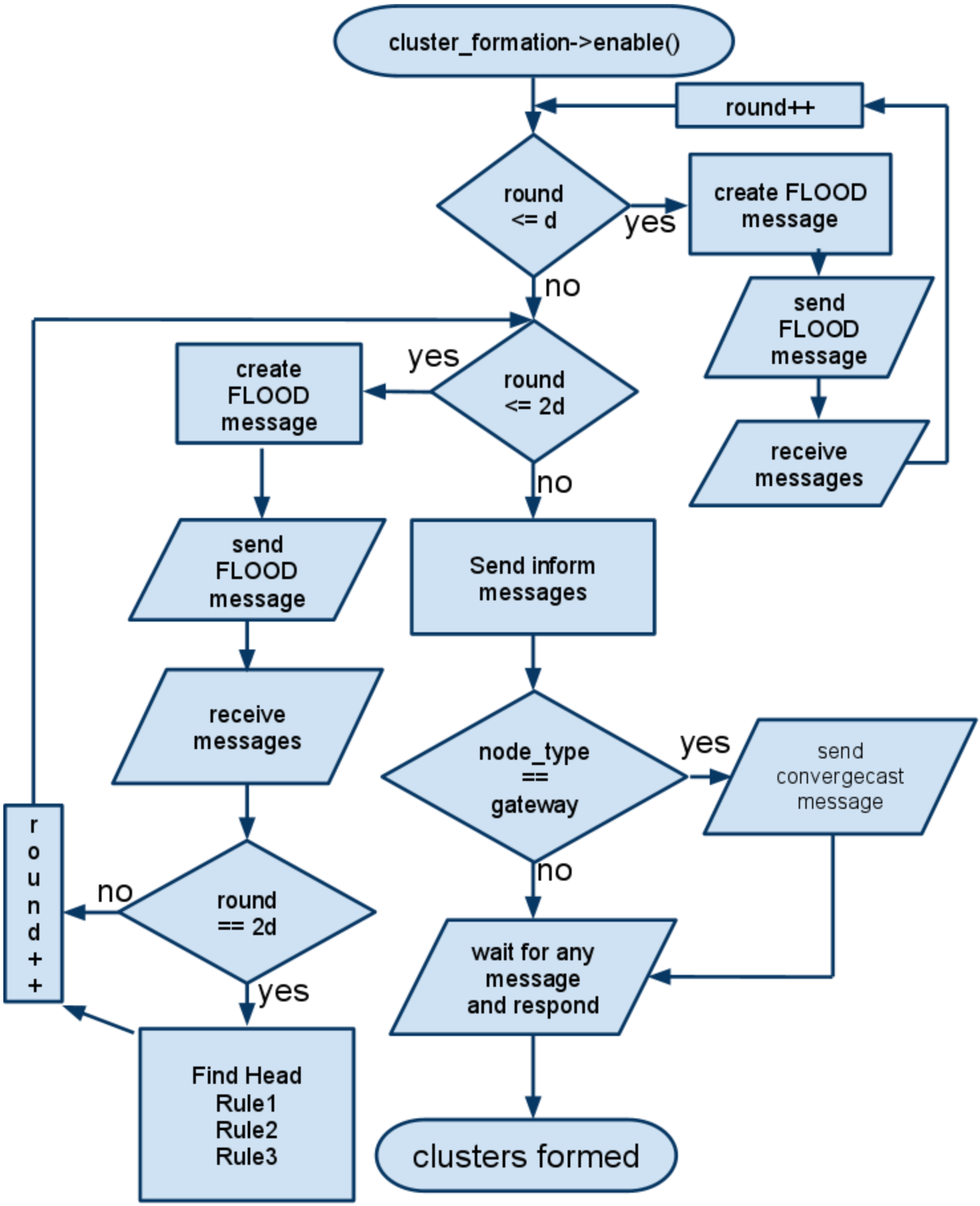
Determining the route form the node to his cluster_head is done using the values of the sender list.
After that the algorithm moves to **(stage 4 "inform" *round 2*d+1*)** the phase during which all nodes inform their neighbors about their choice

in the previous stage (cluster head election). So all 1hop neighbors now know their neighborhood's selections. As a result now all nodes know whether they are connected to nodes from a different cluster and if they have to declare themselves as gateways or remain simple nodes. This declaration determines the node's future behavior.
Finally to confirm of the clusters' formation and to correct any existing errors **(stage 5 "convergecast" rounds *2*d+2 to 4*d+2*)** all gateway nodes start reporting to their cluster_heads, through their selected parents, the network's structure they know. Each non-cluster_head node that receives that message updates his internal structs (inside the iterator) and forwards the received message updated with the information it already knows. After the convergecast stage all nodes have received some information about the cluster's structure and the leaders know the whole cluster.
Also there is a chance where a node of the cluster has been overtaken by a higher id during the cluster formation but in fact belongs to the cluster he is connected to. To correct that a cluster_head, that receives this information through a convergecast message, has to send a rejoin message in order to force the node correct its error (Acts a 4th rule for cluster head decision).
After this stage the cluster formation is over, and all clusters have the correct properties.

To implement the Modular version of the algorithm all procedures described above are separated as shown below:

- Stage 1 **floodmax** has been implemented in cluster_join_decision module
- Stage 2 **floodmin** has been implemented in cluster_join_decision module
- Stage 3 **head election** has been implemented in cluster_head_decision module. All data needed are retrieved from cluster_join_decision and after the cluster head has been elected the iterator is activated and its structures are initialized.
- Stage 4 **inform** has been implemented in iterator module as this is the module that keeps all the information about the node's cluster.
- Stage 5 **convergecast** has been implemented in cluster_join_decision module. This module collects all data from the other modules and creates the payload to be sent to the cluster_head. Also the rejoin message is generated by the cluster_head's cluster_join_decision and is processed by the node's cluster_join_decision.

All stages of the MaxMinD algorithm can be seen in the following flowchart:


cluster_formation->enable()
round++
round <= d
yes
create FLOOD message
send FLOOD message
receive messages
no
round <= 2d
yes
create FLOOD message
send FLOOD message
receive messages
no
Send inform messages
node_type == gateway
yes
send convergecast message
no
wait for any message and respond
clusters formed
round == 2d
no
yes
r o u n d + +
Find Head
Rule1
Rule2
Rule3

V. Results

The results presented below come from simulations run on Shawn. Time corresponds to synchronous Shawn simulation rounds.

Simulations run on a set of topologies that consist of 10, 100, 1000, 2000 and 5000 nodes , range of 2(variable used by Shawn) and using a reliable communication module without any message loss.

For starters we can count the **total duration of clustering**. Shawn simulates a *synchronous* sensor network and as a result we can count the number of *rounds* needed to form the clusters. MaxMind has a quite set number of rounds needed for clustering as the first 4 stages require 2d+1 rounds and cannot be avoided. The last stage consists of convergecast messages that start from the gateway nodes and are forwarded to the cluster head, so some more rounds are needed (at maximum 2d rounds because cluster size may not exceed d hops). The table below shows that *clustering duration* is bounded by the value of d selected. (See Plot 4)

	Rounds for cluster formation					maximum rounds needed
Nodes in topology	10	100	1000	2000	5000	
d = 3	9	10	10	11	13	13
d = 6	15	21	19	21	23	25
d = 9	21	28	29	35	35	37

Another statistic that we can use is the **number of clusters** created for each simulation. For Maxmind clusters are formed *using the node ids*, and clusters are created around nodes with higher ids depend on the *placement* of ids around the network and the diameter of the network. So for example we can see that on the 10 node network with a diameter of 4 hops we can see only 1 cluster formed. (See Plot 5)

	Clusters formed				
Nodes in topology	10	100	1000	2000	5000
d = 3	1	18	12	94	174
d = 6	1	5	16	33	57
d = 9	1	3	9	12	26

Also an important value we can extract from simulations is the **number of nodes per cluster** formed. This show how big a cluster will be and how much *traffic* will be generated inside the cluster, as well as the *traffic* between the clusters (if the clusters consist of a couple of nodes). We can see that cluster size also depends on the d parameter of the algorithm.(See Plot 3)

	Cluster size (average)				
Nodes in topology	10	100	1000	2000	5000
d = 3	10	8.3	21.3	21.28	28.74
d = 6	10	20	62.5	60.60	87.7
d = 9	10	33.5	111.1	166.6	192.3

Another characteristic of the algorithm we should be concerned of is how many nodes are after the first 3 stages of the algorithm have chosen a wrong cluster to join (have been overtaken by a neighbor node). We can see that the **misplaced nodes** are limited as the d parameter of the algorithm reaches the diameter of the network. (See Plot 1)

Misplaced nodes

Nodes in topology	10	100	1000	2000	5000
d = 3	0	2	1	13	60
d = 6	0	4	3	57	89
d = 9	0	0	8	6	37

At last, the total amount of traffic generated inside the network by the cluster formation procedure plays an important role on the total performance of our implementation. We can see that a specific volume of traffic generated during the first 2 stages (flooding stages) cannot be avoided. Also during the final (convergecast) stage we also have traffic that we cannot avoid, because all nodes report to the cluster heads. So we can see that the total traffic (in shawn messages broadcasted) is bounded by the size of the network (total number of nodes) and the d parameter we choose. This is more obvious when we look at Plot 2 where each line is bounded by the O(3d*nodes) line. (See Plot 2)

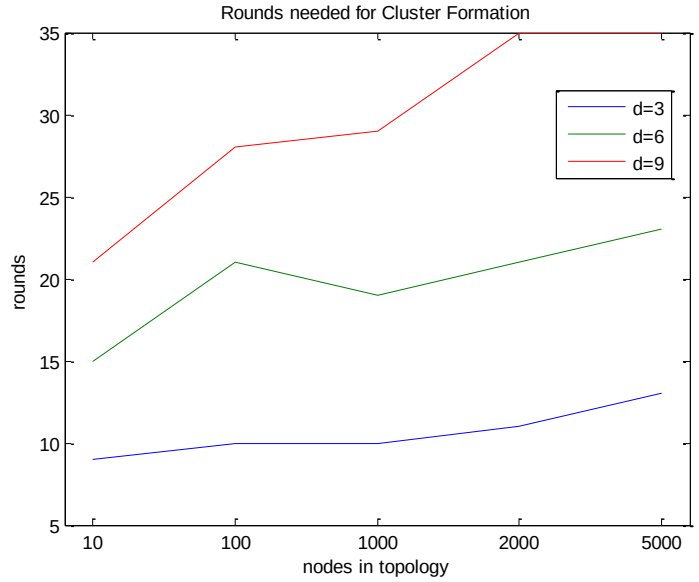
Messages Transmitted

Nodes in topology	10	100	1000	2000	5000
d = 3	84	880	8676	17436	44846
d = 6	144	1610	15843	33764	84605
d = 9	204	2222	23172	47928	122915
d = 9 O(3d*nodes)	270	2700	27000	54000	135000

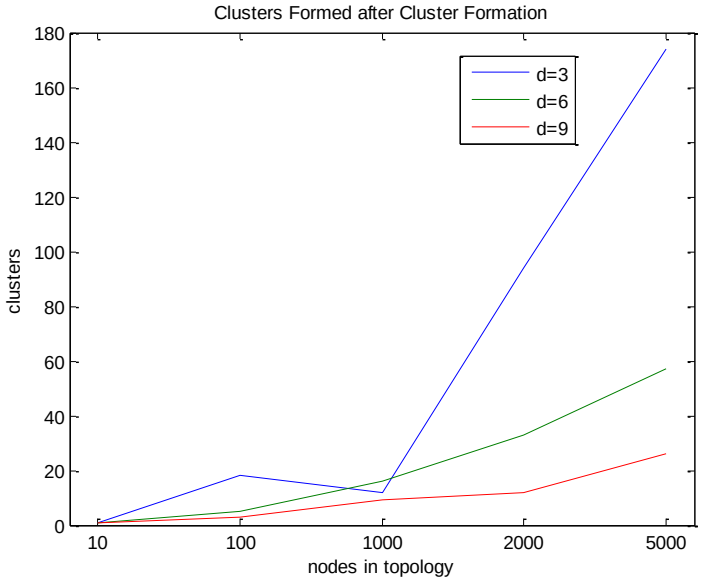
Mixing modules from different clustering algorithms

After implementing and a simple Bfs clustering algorithm using the modular architecture proposed above, we can mix modules from the Bfs algorithm and modules of the MaxMinD algorithm on the same cluster formation module and observe the results for any performance improvement or reusability of the already developed modules in future algorithm development. The new module we are using to replace the Maxmind_Iterator is a simple iterator module that offers limited functionality compared to the Maxmind corresponding module. Actually it is the basic Iterator module used to start developing the Maxmind algorithm with some empty computability functions. By combining the Maxmind modules (Cluster_Head_Decision, Cluster_Join_Decision and Cluster_Formation) with the module of the Bfs algorithm (Simple_Iterator) we see that the performance of clustering shows no real difference. All metrics used above are exactly the same as before with the difference that the new module does not offer node_type designation for each node and some other extra features found in Maxmind, but that are not crucial to the cluster formation procedure. What we want to achieve is being able to combine every module available in order to get clustering algorithms with improved performance using all the good points of each algorithm.

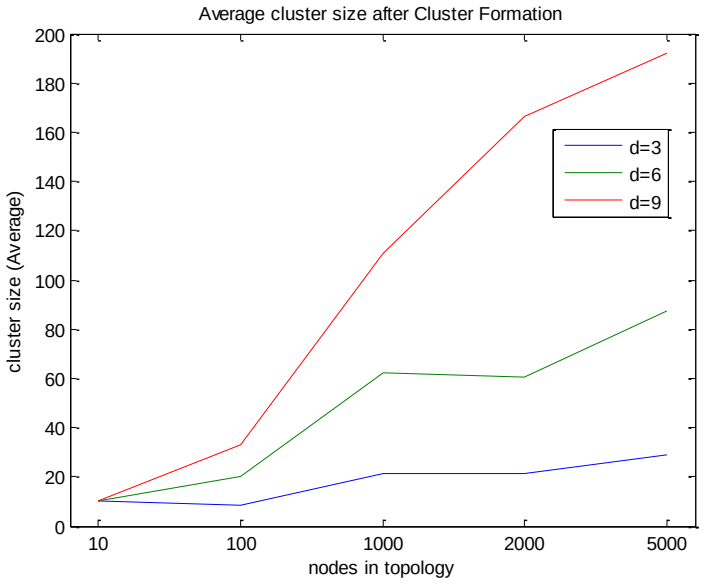
Plot 4



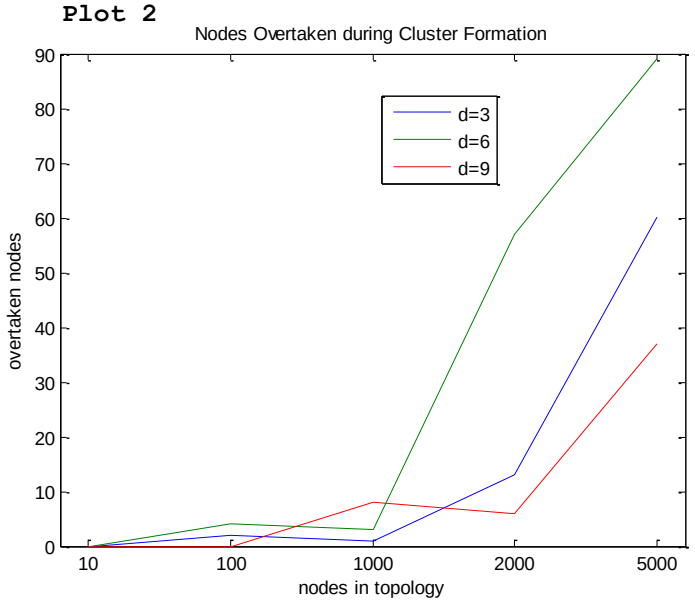
Plot 5



Plot 1



Plot 2



Plot 3