

Detection of communities in Ad hoc Networks and Robust Spanning Trees

Kirsch Laurent
Université du Luxembourg
FSTC - MICS
Luxembourg
laurent.kirsch.001@student.uni.lu

Kirtz Sandy
Université du Luxembourg
FSTC - MICS
Luxembourg
sandy.kirtz.001@student.uni.lu

Abstract—In this paper we present DA-GRS-C, an algorithm that can be used to detect communities in efficient manner. It drastically improves communication paths by building a spanning tree that reflects the community structure of the network. It is based on DA-GRS and therefore benefits from most of the advantages of DA-GRS, most notably efficiency and a high abstraction level.

I. INTRODUCTION

Pervasive systems and mobile computing have raised many new possibilities. The ultimate aim is to be able to access given services anywhere and with zero configuration effort. While major improvements have been made in technology the main challenge is to design protocols that are adapted to this new context.

These new ad hoc environments require several features that must be provided. First of all a device must be able to react to the appearance and disappearance of other devices in its communication area. Another problem is the decentralized nature of such environments. Different mechanisms must be provided to enable service discovery and localization in such an environment. Additionally most routing protocols used in infrastructured networks are not suitable for these ad hoc networks. The devices have to self-organize and set up local group communication systems, from simple one-hop connections to overlay structures such as spanning trees.

A very important characteristic of ad hoc networks is that they usually show the presence of community structures. Such communities may group together devices that are cooperating on some task, such as providing a service or sharing some resources. To be efficient in solving their common tasks these devices are usually densely interconnected whereas the connections with the outer world as for example other communities are usually sparse.

Communication between devices is either based on one-hop communication or if the communication partner is not a direct neighbor it may be based on spanning trees. In order to guarantee efficient communication between devices of the same community we would like to have the devices that form a community to be located in the same subtree of the spanning tree. However, most existing algorithms such as DA-GRS (II) building spanning trees are not designed to detect community

structures, which usually results in inefficient communication paths.

While there exist some methods to detect communities in ad hoc networks, most of them have only theoretical value, but are not relevant in practice as they cannot be implemented efficiently. The approaches presented in the next sections are good examples in this respect. Either they require some knowledge that we do not have (cf. Computer science approaches in II-A) or they are very slow (cf. recent approaches in II-A). In this paper we describe a new method, DA-GRS-C, which can be implemented efficiently in practice and deployed on devices with low computing power. The ultimate aim of community detection is that communities may manage themselves independently and only add new devices to the community or create connections to other communities if it makes really sense. We will concentrate in this paper on the community detection part, not discussing the part on how to determine whether it makes sense to create new links. In this paper we will use the graph representation of networks to describe and illustrate the different methods and concepts.

We rely on tree structures to determine the community structure as it is an efficient and robust structure for routing. We strongly believe that a community detection algorithm that relies on tree structures could not only allow us to manage networks more efficiently by managing communities independently, but also improve communication paths between the devices that are most likely to often communicate. Another reason is that most existing community detection algorithms are recursive processes and usually the different steps of these algorithms are represented using trees. Based on some heuristic one can then determine which step reflects the community structure best.

The rest of this paper is organized as follows. First we will present some of the existing community detection methods as well as the DA-GRS algorithm, which our proposed community detection algorithm is based on. In the subsequent sections we present our proposed method DA-GRS-C (III-B) and give an overview of our test results (IV). We discuss several aspects of DA-GRS-C such as its benefits, but also its limitations. In section (V) we present a method to detect inter community edges. In the future work section (VI) we present some ideas on how to extend DA-GRS-C to make it

suitable for a wider range of networks. Finally, in the last section (VII), we conclude on the work done so far.

II. RELATED WORK

In this section the main topics related to our work are investigated. Concerning detection of communities in a network we first mention the existing community detection algorithms in II-A. In II-B we explain our idea and introduce the DA-GRS algorithm and spanning trees.

A. Community detection algorithms

This subsection presents some existing approaches (cf. [3] [4]) for detecting communities. First of all we define what a community is. In general every network can be represented by a set of vertices representing the devices and a set of edges representing the radio links between the devices. A very common characteristic of networks are communities. A community is a group of nodes that are very densely connected. Connections between different communities are much sparser. We say that a network has a clear cut community structure, if it divides naturally into groups of nodes with dense connections within the different groups and sparser connections between the different groups. Fig. 1 shows a network which has this property. The main question is how we can detect these communities, if we have a network that has a clear cut community structure. During the last years researchers found some approaches to detect communities. These approaches can be divided into two parts:

- Traditional approaches:
 - *Computer Science approach*: This approach is closely related to the graph partitioning problem of a given network. This idea consists of dividing a network of n nodes into g groups with roughly equal size, while minimizing the number of external links that are between nodes in different groups. In practice, this approach is realized by iterative bisection: Find the best division into two groups of the complete graph, and then further subdivide those two groups until the required number of groups is achieved. The most common algorithms for this approach are the spectral bisection method and the Kernighan Lin algorithm. The disadvantage of both algorithms is that the number of communities g has to be known in advance.
 - *Sociological approach*: Contrary to the previous approach where we have to know the number of communities in advance, we do not require this knowledge here. The most commonly used technique is hierarchical clustering. The idea is to compute a similarity measure x_{ij} for all pairs of nodes (i, j) that are part of the given network. Afterwards we enter a recursive process. We start with an empty network, i.e. we add all nodes that are part of the given network but there are no edges. In order to increase similarity we start with the pair of nodes

with the strongest similarity and add this edge to the empty network. The disadvantage of this approach is that it is good at finding nodes that are very similar with respect to the chosen similarity measure, but it has problems to attribute peripheral nodes, i.e. nodes for which it is not clear to which major community they belong to.

- *Recent approaches*: Contrary to the hierarchical clustering method these methods start with a completely connected network and remove edges based on a betweenness measure. The idea of these methods is to consider the traffic on different edges, edges with high traffic are considered to be inter-community edges. There are three different ways to compute the betweenness measure:

- 1) *Shortest path measure*: Find the shortest path between all pairs of vertices
- 2) *Random Walk measure*: Calculate the expected number of times that a random walk between two vertices will pass through a particular edge and sum up for all pairs of vertices
- 3) *Current / Flow measure*: Place a unit resistance on every edge and a unit source s and a unit sink t on a vertex pair (s, t) . The current in the network will flow from source to sink by passing a multitude of paths, those with least resistance carrying the greatest fraction of the current. The current-flow betweenness for an edge is defined to be the absolute value of the current along the edge summed over all source/sink pairs. It can be calculated by using Kirchhoff's law.

One of the common algorithms is the Girvan and Newman algorithm: The idea of this algorithm is to calculate the betweenness measures for all paths starting at a single vertex i . By repeating this for all nodes we then sum up the results to obtain the betweenness scores for all edges. One disadvantage of this algorithm is that m edges have to be removed. We remove one edge per iteration. However, each iteration has a complexity between $O(mn)$ and $O(mn^2)$ and on sparse graphs even $O(n^3)$. Thus this algorithm is too slow.

B. DA-GRS

Before we describe DA-GRS, we give the definition of a spanning tree. Consider a completely connected graph G with a set of vertices V and a set of edges E . A spanning tree over graph G is a graph G' with the same set of vertices V and with a set E' of edges comprised in the set E , such that there are no cycles in G' and such graph G' is completely connected.

DA-GRS (cf. [1] [2]) (Dynamicity Aware Graph Relabeling System) builds a spanning tree over some graph by taking into account locality of communications, i.e. it uses only one-hop knowledge. It also takes into account the dynamicity of the network. DA-GRS provides a generic way to design higher level algorithms that must react to events, such as the appearance and disappearance of nodes in the neighborhood, by

providing an interface that can be implemented at application level. DA-GRS uses tokens to build the spanning tree. There is one token per subtree. At the beginning all the nodes in the graph have a token and thus each node can be considered to be a subtree on its own. The merging process of two subtrees can only take place between two nodes that both have a token. By allowing only nodes that both have a token to merge their respective subtrees, DA-GRS ensures that a tree will be built. After the merging process is complete one of the tokens will be deleted. The tokens circulate through their respective subtree either randomly or based on some heuristic. DA-GRS converges when all the subtrees of a connected component have merged to form one big tree. This final tree is a spanning tree.

The building of the spanning tree by DA-GRS is based on four rules listed below:

- *Token regeneration*: If a subtree doesn't have a token a new token will be regenerated
- *Breaking of a connection*: If a tree link between two nodes breaks, which means an edge is deleted, the node that is on the side of the link where the token currently is has nothing to do in order to maintain a token in his subtree, but the other node of the deleted edge has now lost the route to the token, so this node has to regenerate a token.
- *Merging of subtrees*: During the merging process of two subtrees, the two nodes of the two subtrees will merge and one of the token of the two nodes will be deleted.
- *Circulation of the token*: During the building process of the spanning tree each token in the subtree will circulate to its subtree randomly or based on some heuristics

III. THE preferred node metric AND DA-GRS-C

Our contribution consists of the preferred node metric that allows us to evaluate in an efficient way which are the preferred neighbours of a given node n . Preference of a neighbour is in this context a measure for the likelihood that the neighbour is in the same community as the node n that we are considering. We used this metric to design DA-GRS-C, which is a modified version of DA-GRS that reflects the community structure in the spanning tree that is built by putting those nodes that belong to a same community in the same subtree.

A. Preferred Node Metric

1) *General Idea*: The general idea of the *preferred node metric* is based on the definition of a community, which says that a community is characterized by the fact that between the nodes inside the same community links are denser than between nodes of different communities. Thus it holds that for a node n inside a community c , most of its neighbors will also be inside the same community c . This means that 2 nodes inside the same community have a lot of common neighbors. We use both the definition and this fact to claim the following: *If a neighbor a of a node n is inside the same community as n , then the probability that they will have many*

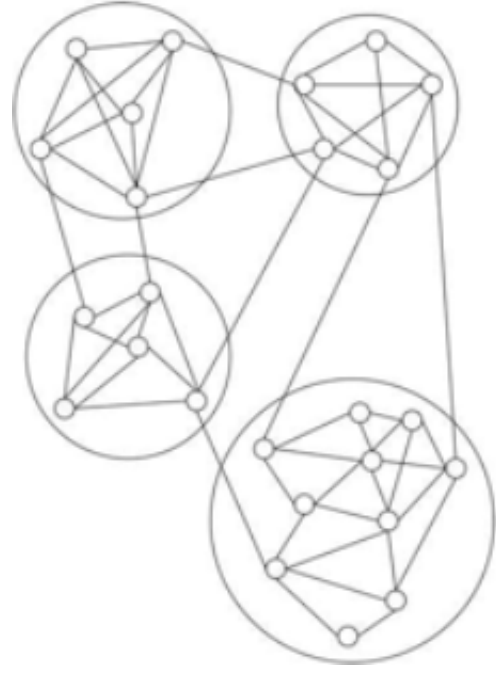


Fig. 1. This figure shows a network with community structure. In this particular example we have four communities with a different number of densely inter-connected nodes. The communities in this network are represented by the circles. As can be seen in this figure, the communities have a high density of internal links but a much lower density of external links, the links between the communities.

common neighbors is much higher than for nodes of different communities.

2) *Formal Description*: This claim is used to build the *preferred node metric*. To calculate the preference score of a node a with respect to another node n we simply consider the size of the set that results from the intersection from both the neighbor set of a and the neighbor set of n and add 1 to it. We add 1 to avoid preferences of 0. This does not affect results, as the difference between the different scores will remain the same. The *preferred node metric* can be formalized as follows:

$$P_n(a) = |N(n) \cap N(a)| + 1$$

where $P_n(a)$ denotes the preference of node a with respect to n and $N(x)$ is the set of neighbours of x . Note that the *preferred node metric* is symmetric, i.e. $P_n(a) = P_a(n)$. In fig. 2 we give an example on how the *preferred node metric* is calculated in practice.

B. Description of the DA-GRS-C algorithm

Consider two nodes a and b with both of them having a token (thus they are in different subtrees) and with a radio link. The main difference between DA-GRS and DA-GRS-C is that DA-GRS merges trees as soon as two tokens meet. Thus DA-GRS would merge the subtrees of a and b whereas DA-GRS-C would only merge the trees if for node a node b is a node with high preference and for node b node a is a node with high preference. However we need a high preference threshold to define when a node is of high preference.

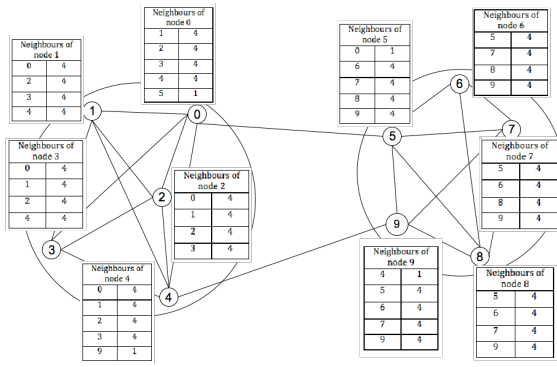


Fig. 2. Example for the Preferred Neighbor Metric: To calculate the preference of a node, for example node 1, with respect to another node, as for example node 2, we consider the size of the intersection of the neighbor set of 1, which is $\{0, 2, 3, 4\}$ and of the neighbor set of 2 which is $\{0, 1, 3, 4\}$ which gives us the set $\{0, 3, 4\}$. The size of this set is 3. We add 1 and obtain a preference score of 4.

A node a is considered to be of high preference with respect to a node n if $P_n(a)$ is above P_n where P_n is the average of all $P_n(a_i)$ where $a_i \in N(n)$.

The problem with the *preferred node metric* is, that a node might wait forever for a preferred node to obtain a token, because both nodes are already in the subtree. We refer to this as the *growing subtree problem*. As each node has only one hop knowledge a node has no easy way to determine whether another node is already part of the same subtree (fig. 3). This problem can, however, be easily solved. A token may contain some information. Thus, we just need to keep track of the nodes that are part of each subtree, by adding this information to the token. In that case a node that gets the token can easily determine whether a preferred node is already part of the subtree and whether it makes sense to wait for an occasion to create a link with this node.

This information can be added easily to the token. At the beginning of the DA-GRS-C algorithm each node is a subtree on its own, just as for the normal DA-GRS algorithm. Thus a node just has to add its own *id* information to the token. In subsequent merging processes, before deleting a token, it is sufficient to add its information to the remaining token and we will have information about the complete subtree in the token. Note, however, that this algorithm works only efficiently for static networks, because after a breaking of a link, the information in the token will be outdated. In the section about future work we describe a possible method to modify this algorithm in such a way that it will work with dynamic networks.

C. Description of the merging process

If a node n has the possibility to merge trees by creating a tree link with another node a it does the following:

- determine the set S of one hop neighbors
- add these to a preference table and give them an initial preference value of 1
- for each one hop neighbor b (a included)

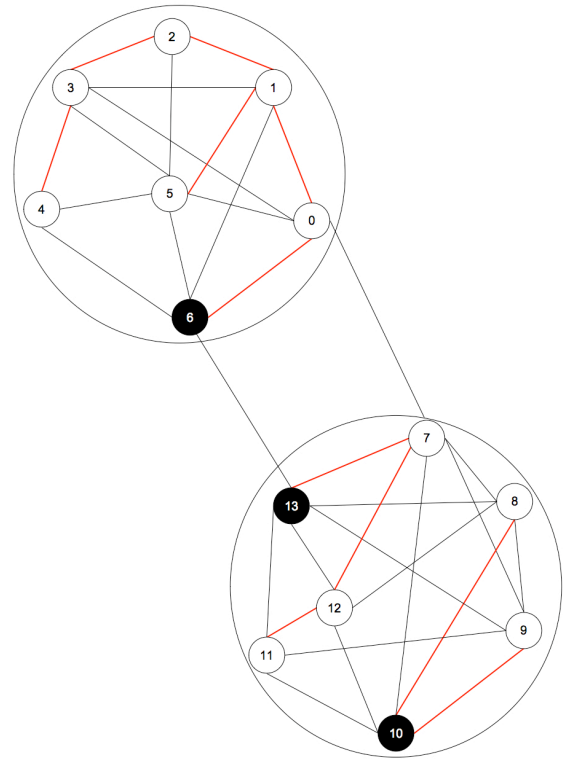


Fig. 3. *Growing Subtree Problem*: Nodes 6 and 13 have the possibility to merge their subtrees. However, when they calculate their respective preference scores they will get a preference score of 1 for each other, while the other neighbors will all get a preference score of 4. However, in this case all the nodes with a high preference score are already in the same subtree, i.e. for node 13 nodes 7, 11 and 12 are already in its subtree and for node 6 nodes 0, 5 and 4 are also in its subtree. However there is no easy way for nodes 6 and 13 to find out whether preferred nodes are already in the same subtree. For instance node 6 cannot know that he is linked to node 5 via nodes 0 and 1. This is what we call the *growing subtree problem*.

- get the set B of one hop neighbors of b
- increment the preference value of all nodes in S that are also part of B
- remove all of the nodes that are already part of the same subtree from the preference table
- calculate the average preference score of the remaining nodes
- if the preference score of node a is above average, be ready to create a new link (it will only be created if node a is also ready to create a link with n)

This method is very efficient, as we are determining the preference of all neighbor nodes in parallel. Note that the evaluation of the *preferred node metric* during the merging process implemented a bit differently than described by fig 2. The preference score will, however, be exactly the same.

D. Real world realization

Real world ad hoc networks are decentralized. This is not a problem. Our DA-GRS-C algorithm is based on DA-GRS which has been designed in such a way that it uses only

one hop knowledge and thus fits perfectly in a decentralized environment. Our modifications in DA-GRS-C do not affect the decentralized design of the original DA-GRS algorithm. In order to be able to calculate a preference table a node needs to know the neighbor set of each neighbor. In a real world scenario nodes can advertise their neighbor sets by using periodic beaconing.

IV. EXPERIMENTATION, METRICS, TEST RESULTS AND OBSERVATIONS

A. Experimentation setup

As mentioned in the section about DA-GRS-C this algorithm has been designed for static networks in a first iteration. Therefore we only tested with static networks. To do those tests, we used a random network generator (RNG). This RNG allows us to specify the number of nodes, the average degree of each node and the number of communities that we want. Additionally we can specify a very important parameter $z\text{-out}$, which is the average number of links that each node has with nodes in other communities. Thus this parameter allows us to specify the density of links between different communities. Another important point is that this random network generator assigns predefined communities to the nodes. Our tests were done using a simulator, based on graphstream, which is a framework of Java APIs for simulations in the context of graphs.

B. Evaluation metrics

To be able to evaluate the results we have passed the predefined communities to the simulator, but the simulator used those only after DA-GRS-C had finished to build the tree to evaluate results. In the next section we present to metrics that we used to evaluate the results. It is due to these metrics that we need a priori knowledge about the correct community assignments.

1) *The inter-community-tree-edge metric (ICTE)*: The idea of this metric is that if we build a tree of n nodes, then we need $n - 1$ edges to link the different nodes. In the case of an optimal spanning tree, all of the nodes of one community will be in the same subtree. An optimal spanning tree can thus be broken into c subtrees (if c is the number of communities), where each subtree will group all of the nodes that belong to one community. If we want to reconnect the different spanning trees later on it becomes clear that we need $c - 1$ tree links. Thus if a spanning tree is optimal, then there are only $c - 1$ inter-community-tree-links. Based on this idea we suggest the following metric to evaluate spanning trees:

$$Q = \frac{c-1}{n}$$

where Q is the quality of the results given as a value between 1 and 0 (1 being perfect and 0 disastrous), c is the number of communities and n is the number of inter-community-tree-links. The advantage of this metric is that it is very efficient to compute. It has however a serious disadvantage in the sense that it is very aggressive especially for a very low number of

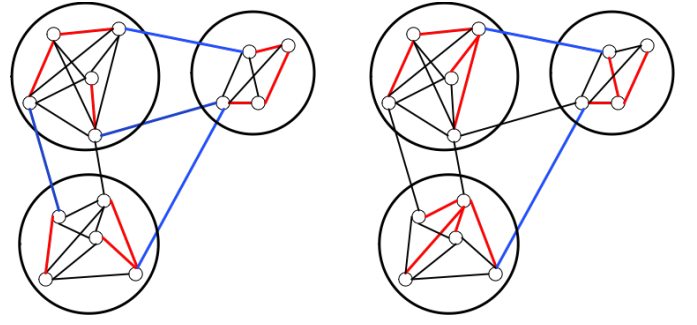


Fig. 4. Examples of suboptimal (left) and optimal (right) spanning trees

communities. Another serious disadvantage is that it does not take into account the exact number of misplaced nodes.

2) *The misplaced nodes metric (MN)*: The idea of this metric is to consider the number of subtrees inside each community. In the case of an optimal spanning tree there is only one. If the spanning tree is not optimal we take into account the size of the different subtrees, the size of a subtree being given by the number of nodes in the subtree. The nodes that are part of the subtree with the biggest size are considered to be placed correctly, while all the other nodes are considered to be misplaced. This gives us the following metric:

$$Q = \frac{n - \sum_{i=1}^c m_i}{n}$$

where Q is the quality of the results, c is the number of communities, m_i is the number of misplaced nodes in the i -th community. The advantage of this metric is that it is much more accurate than ICTE as we here we count the exact number of misplaced nodes. On the other hand this metric is not so efficient in terms of computation time.

3) *Evaluation example for ICTE and MN*: Figure 4 shows two identical networks. The circles represent the communities. The bold (red and blue) edges in those networks represent the spanning trees. Blue links are tree links between different communities, red links are links between nodes inside the same community. We observe that there is a difference between the spanning trees. The spanning tree of the network represented on the right is optimal with respect to the community structure of the network as all the nodes inside a community are also inside the same subtree. It is obvious that both metrics, ICTE and MN, will evaluate the quality of the subtree as being 1. On the other hand the spanning tree of the network at the left side is not optimal as the nodes of the same community are not always grouped together in the same subtree. When applying ICTE to this network we will obtain a quality measure of $\frac{3-1}{4} = 0.50$. With MN we obtain a quality measure of $\frac{14-(2+2+0)}{14} \simeq 0.71$. Note that this is only a theoretical example to illustrate ICTE and MN and is not a spanning tree obtained by some real run of DA-GRS or DA-GRS-C.

Nr. of communities	4	8	12	16
ICTE	0.24	0.62	0.73	0.76
MN	0.88	0.94	0.94	0.92

TABLE I

RESULTS FOR DA-GRS WITH 90 NODES, AN AVERAGE DEGREE OF 16 PER NODE AND A $z\text{-out}$ OF 2. VALUES ARE AVERAGES OF 100 RUNS.

Nr. of communities	4	8	12	16
ICTE	0.07	0.18	0.28	0.31
MN	0.49	0.60	0.68	0.64

TABLE II

RESULTS FOR DA-GRS WITH 90 NODES, AN AVERAGE DEGREE OF 16 PER NODE AND A $z\text{-out}$ OF 8. VALUES ARE AVERAGES OF 100 RUNS.

C. Test Results and Observations

1) *Original DA-GRS*: Before running some tests for DA-GRS-C, we ran some tests on DA-GRS to see how good it performs with respect to community detection. Table I contains some of the results for DA-GRS. As expected ICTE is much more aggressive, especially for a small number of communities. Our result files showed that for MN the size of the misplaced subtrees is usually 1, which is another explanation why the quality of results when evaluating with ICTE is so much worse than when evaluating with MN. A size of 1 means that the number of excessive inter-community-tree edges is approximately equal to the number of misplaced nodes.

$$n_{exc} \simeq n_{mispl} = n$$

However, in the case of ICTE we compare this number n to the number of communities which is a very small number while in the case of MN we compare it to a much bigger number, namely the number of communities. As this size of misplaced subtrees is usually 1, MN seems to be much more precise than ICTE.

It seems to be very surprising that DA-GRS is performing so well with respect to the misplaced nodes metric, because DA-GRS was not designed to detect communities. However, as at the start of the algorithm each node is a subtree on its own and as the density of links inside the communities is much higher than between different communities, the probability that a node will merge subtrees with a node from the same community is very high, especially during the first phase of the tree building process. Thus DA-GRS detects communities quite well when the community structure is clear, i.e. inside the communities the density of links is much higher than between the different communities. We obtained similar numbers for different numbers of nodes. For a higher $z\text{-out}$, however, results are disastrous even for the second metric as is shown by table II, that contains some results for a $z\text{-out}$ of 8.

2) *DA-GRS-C*: For DA-GRS we ran similar tests. Table III contains the results that we obtained for DA-GRS-C when using the MN metric. We can see that for networks with a clear cut community structure (as for $z\text{-out} = 2$) results are perfect with respect to community detection. The spanning tree that is

Nr. of communities	4	8	12	16
MN for $z\text{-out} = 2$	1	1	1	1
MN for $z\text{-out} = 8$	0.51	0.90	0.91	0.84

TABLE III

RESULTS FOR DA-GRS-C WITH 90 NODES AND AN AVERAGE DEGREE OF 16 PER NODE. ALL VALUES ARE AVERAGES FOR 100 RUNS. THEY REPRESENT THE QUALITY MEASURE OBTAINED WITH THE MN METRIC.

Nr. of communities	4	4	8	8
DA-GRS version	orig.	-C	orig.	-C
DFTM	49.90	54.06	65.33	68.52
RTM	58.66	78.19	73.10	93.32

TABLE IV

CONVERGENCE TIME OF DA-GRS AND DA-GRS-C. ALL OF THE VALUES ARE AVERAGE VALUES FOR 100 RUNS. THE NETWORKS ON WHICH WE DID THE TESTS, HAVE 90 NODES, AN AVERAGE DEGREE OF 16 PER NODE AND AN AVERAGE $z\text{-out}$ OF 2. CONVERGENCE TIME IS MEASURED BY COUNTING THE NUMBER OF TOKEN MOVEMENTS OF THE REMAINING TOKEN. (CF. IV-C3)

built by DA-GRS-C is always optimal in that case. However, when the density of links between different communities gets close to the density of links inside communities, as is the case for a $z\text{-out}$ equal to 8, when the average degree of the nodes is 16, results get much worse. However, in that case one might argue whether it really makes sense to speak of different communities, because communities are defined as having a high density of links inside and a low density of links between them which does not hold true in that particular configuration.

3) *Convergence Time of DA-GRS and DA-GRS-C*: Intuitively one would say that DA-GRS-C will take a lot more time to converge than DA-GRS, because a node only merges subtrees when the other node is a preferred node, whereas DA-GRS merges with every node. We did some tests to find out whether our intuition is right. The same tests (i.e. on the same networks) were done for both DA-GRS and for DA-GRS-C. During these tests we used two token movement strategies. The first strategy is random token movement (RTM). The second is depth first token movement (DFTM), which works in the same way than a typical depth-first-search. We measured the convergence time, by counting how many token movements are necessary to converge. Thereby we do not count all of the token movements, because most token movements are executed in parallel. We count how many times the final token (the token that will remain when the spanning tree is complete) has been moved around during the building process. Table IV lists our results. From these results we can observe that DFTM is a much better token movement strategy than RTM for both DA-GRS and DA-GRS-C. The gap between DFTM and RTM is quite significant. Another observation that can be made is, that while there is a difference in convergence speed between DA-GRS and DA-GRS-C, it is not of big significance. Thus our intuition that DA-GRS-C is converging much slower than DA-GRS is wrong. This can again be explained by the fact that, as at the start of the algorithm each node is a subtree on its own and as the density of links inside the communities is much higher than between different communities, the probability

that a node will immediately have the possibility to merge subtrees with another node that has a high preference score is high, especially during the first phase of the tree building process. The marginal difference results from the last phase of the tree building process, because then it will take a bit more time before two nodes with high preference can merge their subtrees. Nevertheless the impact of this is very small as can be seen from the results.

V. DETECTING INTER-COMMUNITY TREE EDGES

As shown in the previous section, the results obtained by DA-GRS-C are optimal when the community structure is clear cut. However our ultimate goal of detecting communities is not only to build optimal spanning trees, but to be able to manage communities on itself and DA-GRS-C does not tell us where the different communities are situated in the subtree. Thus we need a metric to be able to detect inter-community tree edges. This metric is again the *preferred node metric*.

A. Detecting inter-community (tree) edges with the preferred node metric

In section III-A we introduced the *preferred node metric* that is used to improve the results for DA-GRS-C. We may also use it to detect inter-community tree edges. Each node n calculates its preference table in the same way as for the merging process of DA-GRS-C. The neighbors, that have a preference score that is lower than the average of all preference scores minus the standard deviation of all preference scores divided by two, are considered to be part of another community. Therefore the edges between n and those neighbors are inter-community tree edges. This can be formalized as follows:

If $P_n(a) < (P_n - \frac{stdDev}{2})$ then the edge formed by a and n is an inter-community tree edge. An example of this idea is shown in Fig. 5

B. Measurements and results

To evaluate the results obtained in this way, we again made use of our a priori known communities, by passing them to the simulator. We tested it on some random generated networks. We used the same networks as for DA-GRS-C. To evaluate results we introduce the notion of *positive IC edges*, *false positive IC edges* and *false negative IC edges*:

- 1) *positive IC edge*: an edge is considered to be a positive IC edge if our metric indicates that it is (or is not) an inter-community edge and it really is (or is not) an inter-community edge.
- 2) *false negative IC edge*: an edge is considered to be a false negative IC edge, if our metric indicates that an edge is not an inter-community edge, but it actually is an inter-community edge.
- 3) *false positive IC edge*: an edge is considered to be a false positive IC edge, if the metric indicates that an edge is an inter-community edge, but it actually is not an inter-community edge.

Ratio of...	positive	false positive	false negative
$z-out = 2$	1	0	0
$z-out = 8$	0.60	0.05	0.35

TABLE V

RESULTS FOR THE INTER-COMMUNITY EDGE DETECTION METRIC FOR NETWORKS WITH 90 NODES, AN AVERAGE DEGREE OF 16 PER NODE AND 4 COMMUNITIES. VALUES ARE AVERAGE RATIOS FOR 100 RUNS.

We use these notions after each merging process of DA-GRS-C by comparing the results of the metric, with the actual communities and thereby counting the number of positive IC edges the number of false positive IC edges and false negative IC edges. Thus we apply this metric only to tree edges. After DA-GRS-C converges we calculate ratios by dividing the number of positive IC edges, respectively the number of false positive IC edges or the number of false negative IC edges by the total number of tree edges in order to get more meaningful results. Table V shows the results obtained for the different random networks.

The results show that for a clear community structure this method is appropriate. For clear community structures it gives us optimal results. Thus in a step we can use the results of this metric to determine where in the spanning tree of DA-GRS-C the subtrees representing the different communities are located. It is particular easy as DA-GRS-C is also always optimal for clear community structures. Thus, in that case there is always only one subtree per community, which makes it very easy to detect the different communities with the help of this inter-community edge detection metric.

For a less clear cut community structure, it seems that this metric has problems to detect all of the inter-community tree edges, but as for DA-GRS-C it is not clear whether it really makes sense to speak about different communities when we have a $z-out$ of 8 (cf. IV-C2).

VI. FUTURE WORK

The main drawback of DA-GRS-C is that it makes only sense if we have static networks. In a real-world scenario, however, ad hoc networks are usually dynamic. In the subsequent paragraphs we describe an idea on how to extend DA-GRS-C in such a way that it will also work in the context of dynamic networks. We also present an idea on how to detect communities without making use of DA-GRS at all.

A. DA-GRS-C for dynamic networks

The main problem with DA-GRS-C as presented above is that the information about subtrees contained in the token will be outdated. To fix this we need two extensions to the original DA-GRS-C algorithm.

1) *Description of the extensions*: The first extension is to also include information about the edges that build the subtree inside the token. The size of the token will not be a real problem. If we include the edges, we do not need a list of nodes any longer. An edge is represented by a pair of nodes. Thus the node list can be determined, if we have the edge list. The second extension is to store the information stored

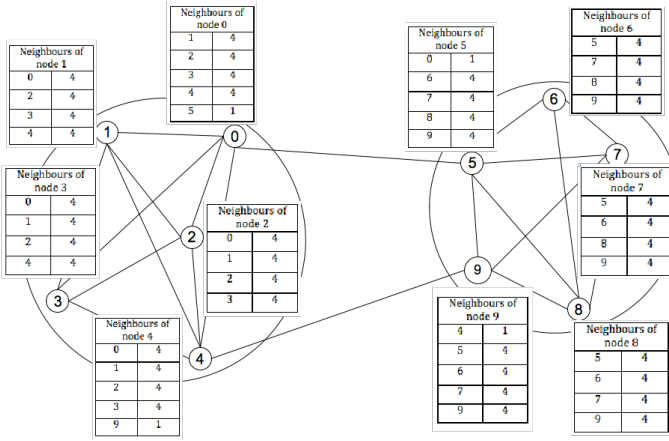


Fig. 5. This figure give an example on how to detect community boundaries by using the *preferred node metric*. The preference table of each node indicates which neighbors are in the same community. In this figure we can easily deduce that for the node 0 the node 1, node 2, node 3, node 4 are in the same community according to the high preference score of 4. In contrast neighbor node 5 seems to be an different community according to the low preference score of 1. The same can be deduced for node 5 whose neighbors: node 6, node 7, node 8 and node 9 are in the same community according to the preference score of 4.

in the token locally in each node, when the token passes. By doing so each node has some knowledge about the subtree. Even if this knowledge will not be up to date all the time, it is nevertheless sufficient to update the information in the token when a link breaks.

2) *Using these extensions to deal with broken links:* If a tree link (formed by nodes n and m) breaks, one of the nodes n that was part of that link has to regenerate a token. The information that n has stored locally will be sufficient to regenerate a token containing correct information, because the part of the subtree in which topological changes might have occurred in the meanwhile is on the other side of the broken link and thus the information corresponding to this subtree will anyway not be included in the new token. On the other side of the broken link matters are not so easy. One of our ideas is to simply wait until node m gets the token. It can then update the information in the token in the same way as n did previously when it regenerated a token. Another idea is to send a multicast message containing information about the subtree that has gone, along all of the remaining tree links and keep forwarding this message in each node along all tree links until the token is met in some node or until we reach a leaf node. With the help of the information in this multicast message the token information can then be updated as previously. However, for both approaches, we will have temporarily outdated information in the token. During this period the token can nevertheless continue to merge subtrees, if it meets other tokens. A possibility to deal with this is to check whether the tokens which will trigger the merging process do not have common edges and if they do, then simply not merge the subtrees.

3) *Open Questions:* There are many open questions left. One of them is whether it is really necessary to perform the check with the common edges during the merging process. Another open question is convergence time and efficiency in general.

B. Another approach for DA-GRS-C

In our DA-GRS-C approach we include information about the nodes of the subtree in the token to solve the *growing subtree problem*. A different approach could be to use a probabilistic approach. Instead of creating links only between preferred nodes, we could introduce probabilities. A node could then create a link with another node of low preference (below average) with a small probability while creating new links with a node of high preference with a high probability. The advantage of this approach is that it works for both static and dynamic networks without modifications in the latter case. Another advantage is that we do not need to include information about the subtrees in the token and thus we do not have to care about outdated information in the token. The big disadvantage is however that we will lose a lot of time at the end of the algorithm when there are only nodes left that have a low preference, because even in that case links to these nodes will only be built with a very small probability.

C. A different approach for detecting communities

A totally different method for detecting communities is based on the inter-community-link-detection algorithm described in section V. We can use this algorithm to detect the inter-community-edges. We can then determine which nodes should be grouped together into a community, by looking at how the inter-community-links fit into the topology of the network.

VII. CONCLUSION

We showed that DA-GRS-C provides excellent results with respect to community detection and in the context of static networks, as long as the community structure is clear cut. Nevertheless we also noticed that results get worse if the community structure blurs, i.e. if there is no longer a big difference between the density of links inside the communities and the density of links between the communities. Another observation was that the difference in convergence time between DA-GRS and DA-GRS-C is not of big importance. The best convergence time can be achieved by using Depth-First Token Movement strategy. DA-GRS-C can also be implemented efficiently in practice, in such a way that it is also possible to use it on mobile devices, which usually have very limited computational power. The main disadvantage of DA-GRS-C is that it is suitable only for static networks so far. However, there are many ideas on how to extend DA-GRS-C for dynamic networks, but at present there are no concrete implementations nor test results for the methods suggested in VI-A yet. As for DA-GRS-C the inter-community-edge detection algorithm provides excellent results as long as the community structure is clear cut. As it is based on the *preferred node*

metric it is also very efficient. This algorithm can be used in combination with DA-GRS-C to detect the community boundaries once the spanning tree is built. It can also be used on its own to detect communities. So far we have tested this algorithm only in combination with DA-GRS-C, but nevertheless it seems interesting to use the intercommunity-edge-detection-algorithm on its own in scenarios that do not require a spanning tree.

REFERENCES

- [1] A. Casteigts, "Model driven capabilities of the da-grs model," in *ICAS*, 2006, p. 24.
- [2] A. Casteigts and S. Chaumette, "Dynamicity aware graph relabeling systems (da-grs), a local computation based model to describe manet algorithms," in *IASTED PDCS*, 2005, pp. 231–236.
- [3] M. E. J. Newman, "Detecting community structure in networks," *The European Physical Journal B - Condensed Matter*, vol. 38, no. 2, p. 321, 2004. [Online]. Available: <http://dx.doi.org/10.1140/epjb/e2004-00124-y>
- [4] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Phys. Rev. E*, vol. 69, no. 2, p. 026113, Feb 2004.