Network Communities and Robust Spanning trees in MANETs

Laurent Kirsch
Sandy Kirtz

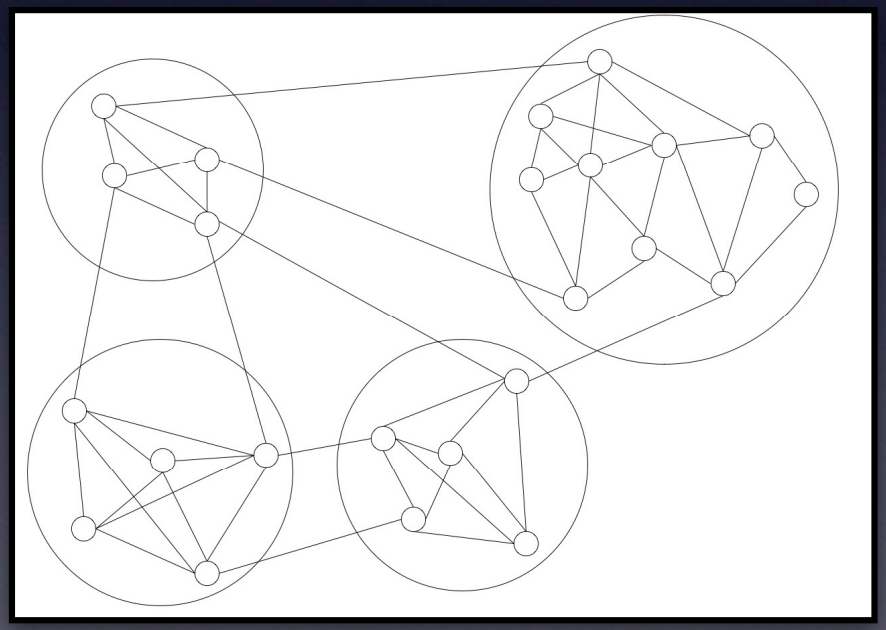
Part 1:
Introduction & Review

Aim of the Project

- *Aim:* Detection of communities within a MANET using tree structures
- *Why tree structures:*
  - most community detection algorithms are a recursive process
  - a spanning tree could reflect this community structure
- *How:* modify DA-GRS in such a way that it detects communities and provides better results with respect to community detection

Communities

- A community is a set of densely interconnected devices
- There are many links between devices inside a community
- There are only a few links between devices of different communities



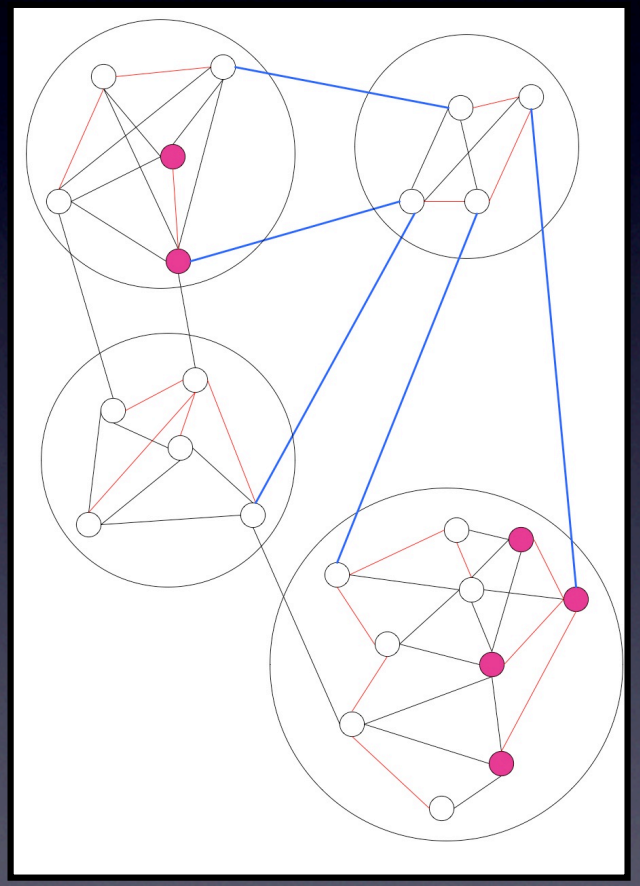
DA-GRS algorithm

- Builds a spanning tree over a graph
- Takes into account:
  - Locality of communications (only one-hop knowledge)
  - Dynamicity of the network
- Uses tokens (one per subtree) to make sure a tree will be built
- Circulation of the token is random (or based on some heuristic)

How to assess DA-GRS matching with Community structure?

- Metric 1: quality = (N-1)/I
  N: Number of communities
  I: number of inter-community-tree-edges

- Metric 2: quality = (n-m)/n
  n: total number of nodes
  m: total number of misplaced nodes



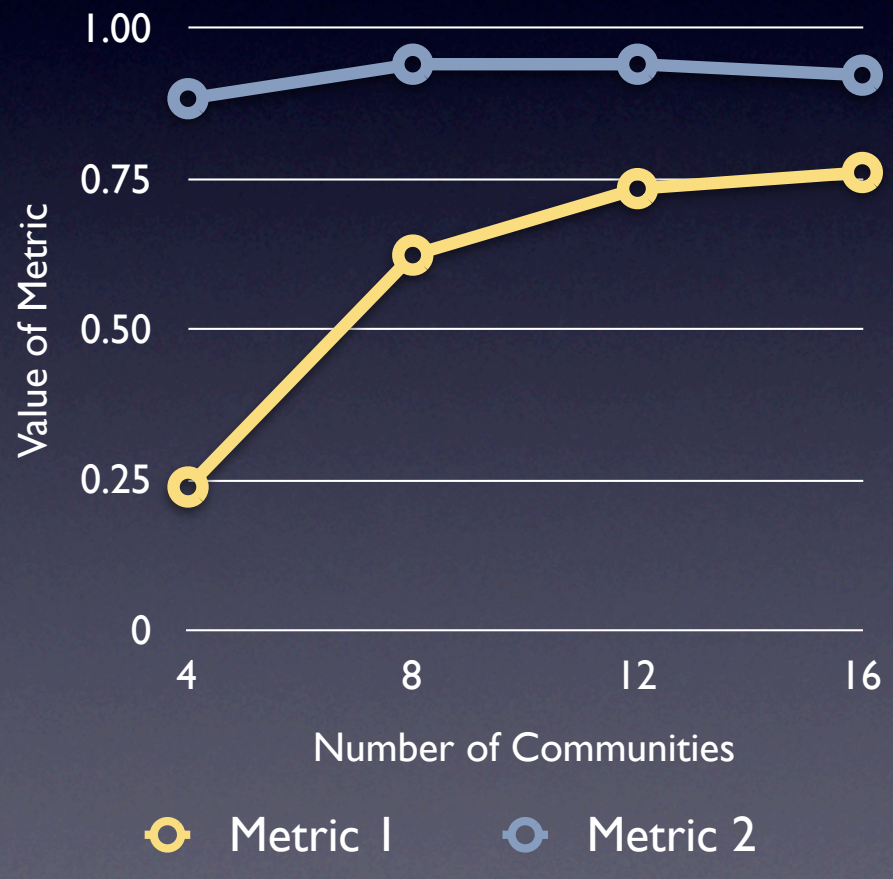
Details on testing

- Randomly generated static networks with pre-defined community attribution
- Network generation ensures some graph properties based on community attribution:
  - Expected degree of nodes (d)
  - Expected internal (inside community, z-in) degree and external (inter community, z-out) degree
- Assess how DA-GRS can detect this community structure
- Restriction to static networks in a first iteration

Original DA-GRS

Results for 90 nodes (d = 16, z-out = 2)

Nr. of communities	4	8	12	16
Metric 1	0.24	0.62	0.73	0.76
Metric 2	0.88	0.94	0.94	0.92



Observations for Original DA-GRS

- Results are much better than expected
- Metric 1 is much more aggressive than metric 2
- Misplaced subtrees are usually of size 1

Part 2:
Optimizing DA-GRS

How to optimize DA-GRS ?

- Optimize Merging Process
- For Original DA-GRS when two nodes can merge they will merge
- *Idea:*
  - Favor building of tree links inside the communities
  - Connect the different communities in a second step

Detecting pairs of nodes inside the same community

- *Reminder:* Nodes inside a community are densely linked
- *Idea:* A pair of nodes inside a same community has a bigger set of common neighbors than two nodes from different communities
- In practice a node can use periodic beaconing to advertise its neighbor set

Preference Table

- Build a preference table (contains information about all one-hop neighbors) inside each node having a token
- The preference of a node b with respect to node n, denoted by Pn (b) is defined to be the size (+1) of the set that results from the intersection of the neighbor set of node n with the neighbor set of node b, denoted by |N(n) ^ N(b)| + 1 where:

  N(x) is the set of neighbors of node x
  |S| is the cardinality of the set S
  ^ is the intersection operation

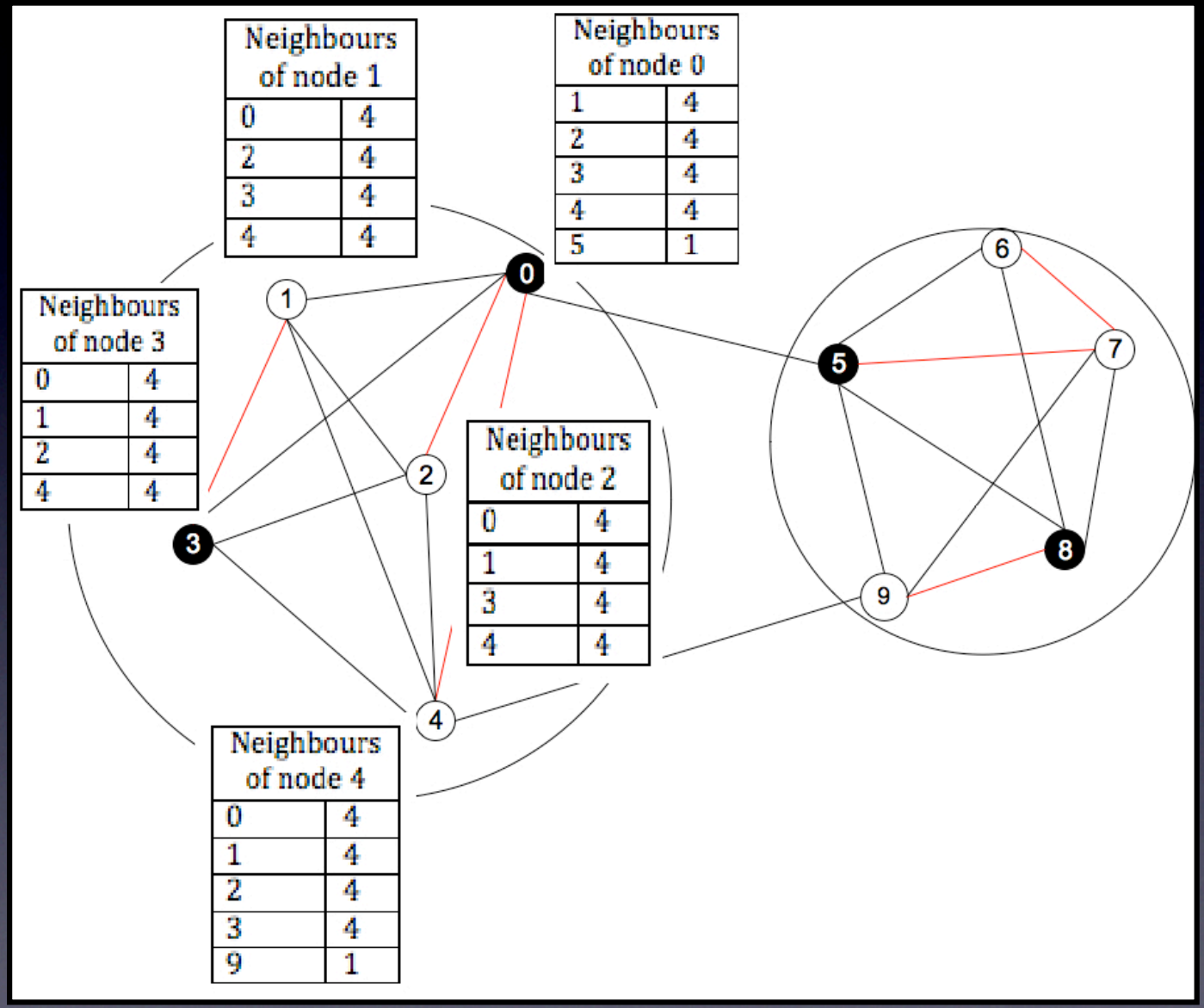
- Nodes with the higher preference will be considered to be in the same community than n

Using this during the merging process

If 2 nodes (n1, n2) can merge their respective subtrees (both nodes have a token) they only create a tree link if Pn1(n2) is above Pn1 and Pn2(n1) is above Pn2.

(Pn is the average of Pn(i) on all i of N(i))

Example on how to calculate preference tables

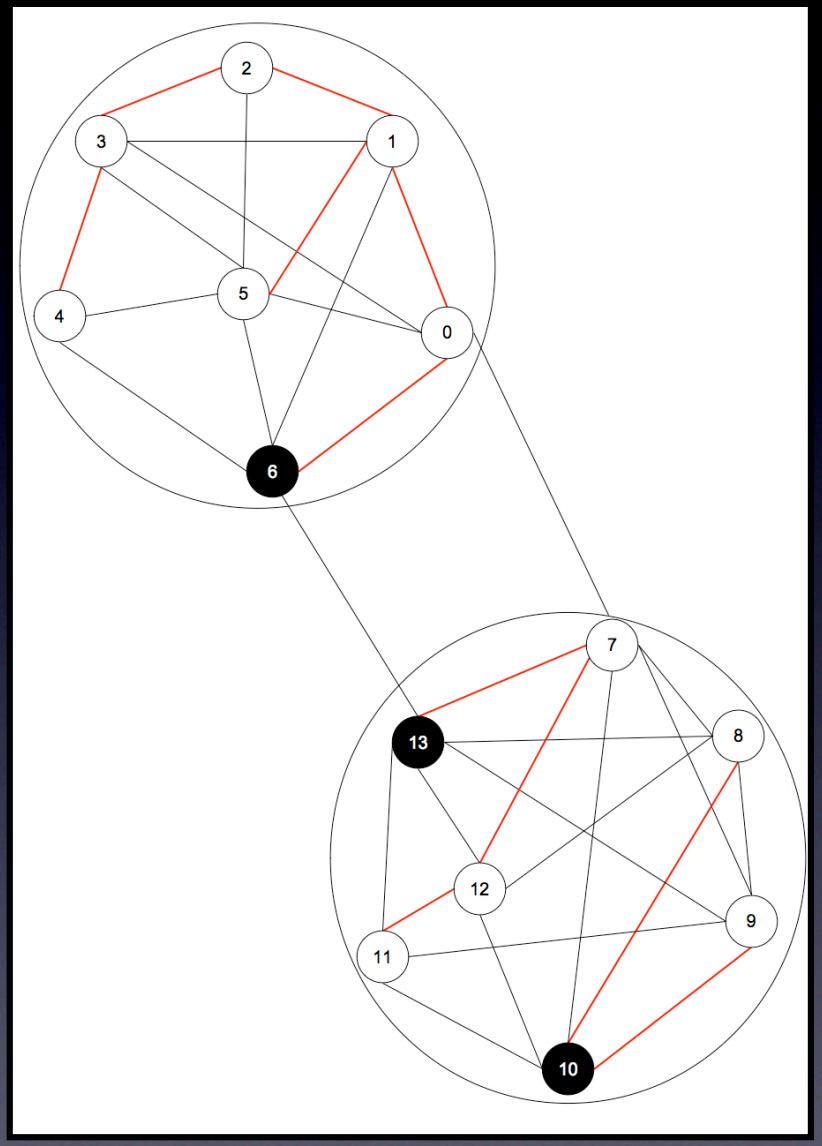


Preferred nodes and growing subtree

Preferred nodes already in the same subtree as the current node shall be ignored:

- avoid waiting for an impossible merge
- allow less preferred nodes to create tree links (and link different communities)

Growing Subtree illustration



Solution

- *Idea:* Include information about the subtree inside the token
- *How:*
  - *Reminder:* At the beginning of the algorithm each node is a subtree on its own and has its own token
  - Include own id in the token (can be easily done at the beginning)
  - When merging two trees, include all of the information of the token that disappears in the token that remains (token will contain all ids of the nodes in the subtree at each step)

Complete Merging Process

If 2 nodes (n1, n2) can merge their subtrees:

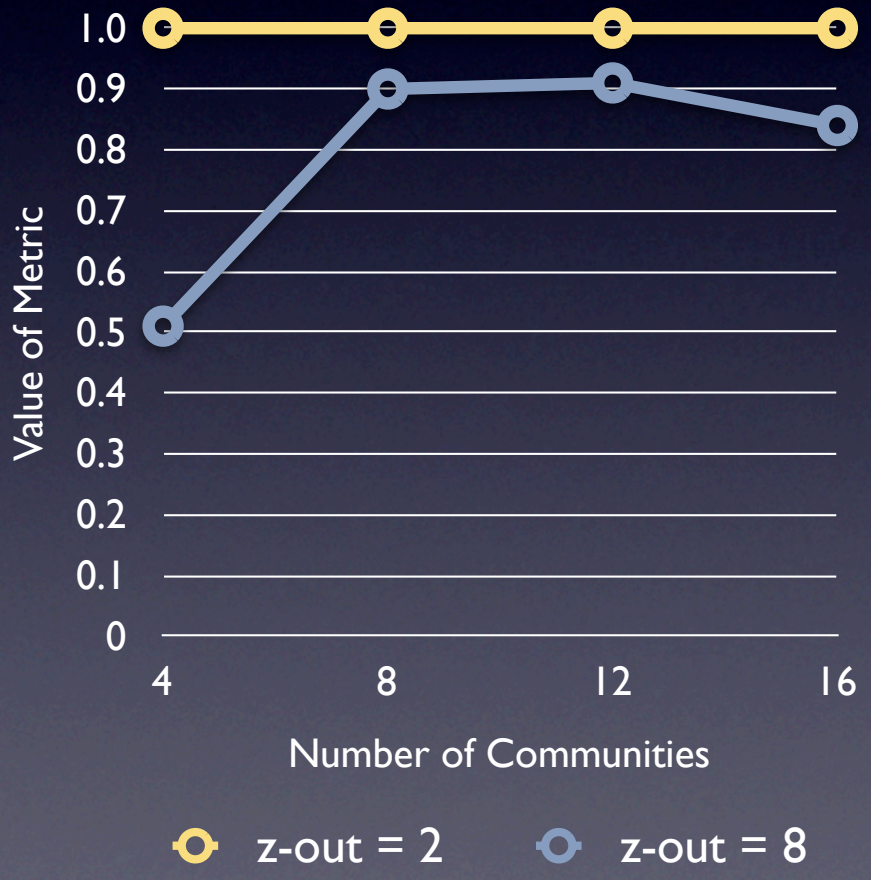
1. Build a preference table for both n1 and n2
2. Remove the nodes that are already in the same subtree
3. Only create a tree link if Pn1(n2) is above Pn1 and Pn2(n1) is above Pn2.

Optimized DA-GRS

Results for 90 nodes (d = 16)

Nr. of communities	4	8	12	16
z-out 2	1	1	1	1
z-out 8	0.51	0.90	0.91	0.84

2nd quality measure only



Observations for Optimized DA-GRS

- The spanning tree that we get with the optimized version of the DA-GRS algorithm is optimal as long as the community structure is clear cut (many links inside communities, few links between different communities)

- But what about convergence time ?

Impact on Convergence Time

- Convergence time may have increased as the optimized version of DA-GRS does not build a tree link each time it could build a tree link as does the original DA-GRS
- How to optimize convergence time ?
- Explore different token movement strategies

  2 main candidates:

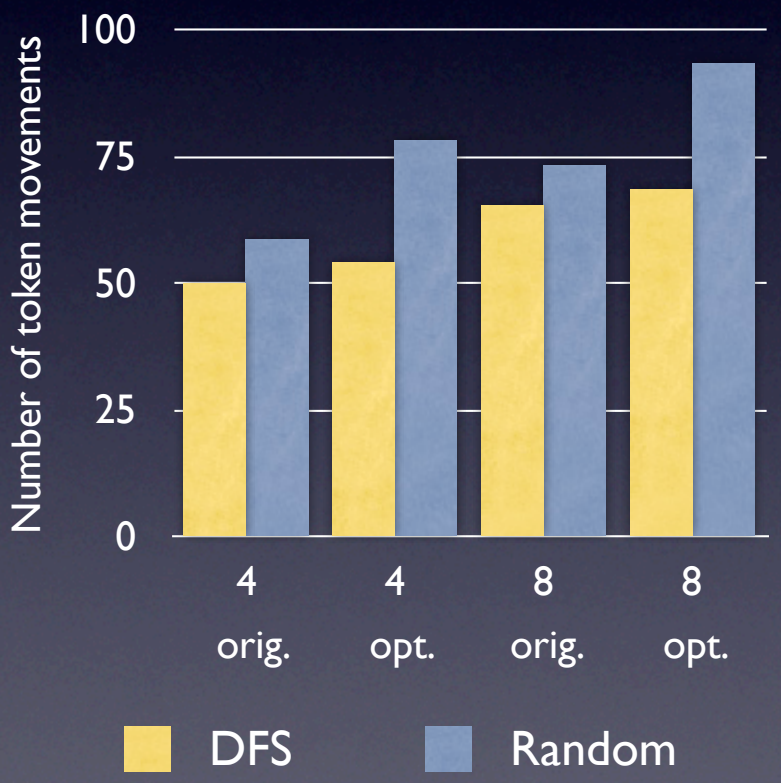
  1. random token movement
  2. DF (depth first) token movement

Convergence time

Results for 90 nodes (d = 16, z-out = 2)

Nr. of communities	4		8	
orig./opt.	orig.	opt.	orig.	opt.
DFS	49.9	54.06	65.33	68.52
Random	58.66	78.19	73.1	93.32

orig. = original version
opt. = optimized version



Observations

- DFS token movement is much more efficient than random movement

- The convergence time of the optimized version of DA-GRS is only marginally higher than the convergence time of the original version of DA-GRS

Detecting inter-community-edges

- Idea is the same as for the merging process
- for each node:
  - determine neighbor preference table
  - neighbors with a preference score lower than

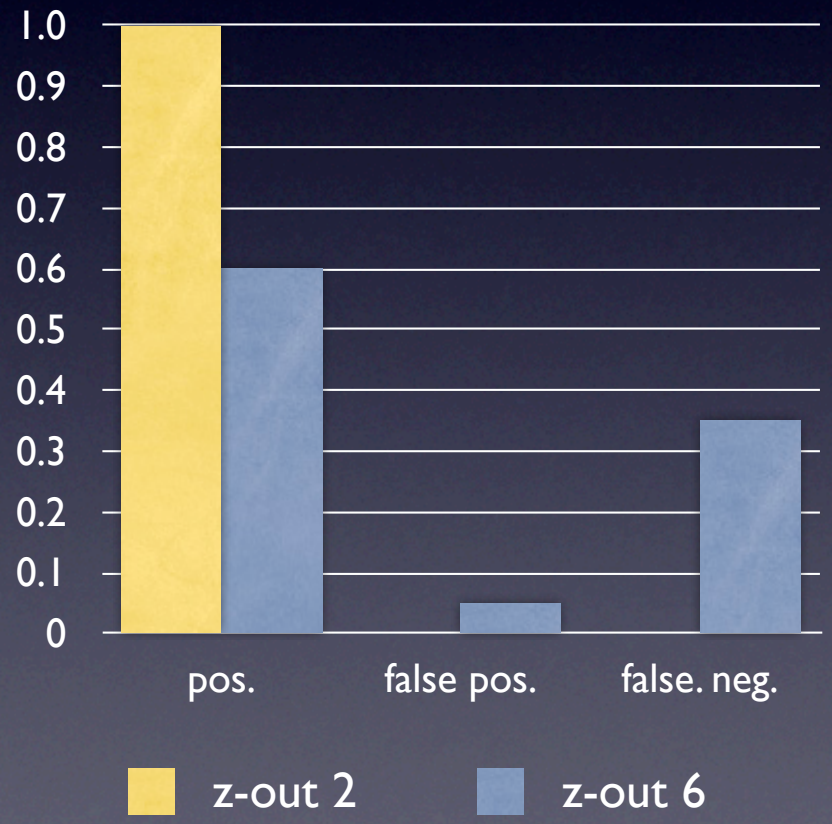
    (average - stdDev/2)

    are considered to be inter-community edges
- Knowing the inter-community-edges it is possible to determine the different communities

Some results

Results for 90 nodes (d = 16, c = 4)

Rate of ...	positive	false positive	false negative
z-out 2	1	0	0
z-out 8	0.6	0.05	0.35



Observations

- Results with this method are very good as long as the network communities are clear cut (i.e. many links inside communities, few links between different communities)
- If the community structure is not so clear cut, results are worse

Part 3: Future Work

Suggested method for dynamic networks

Idea

- include information about subtree edges inside the token
- extract information contained in the token and save it locally in the nodes
- this local information can be used to regenerate a token with correct information after a tree link breaks

Open Questions

- There are however some open questions left concerning convergence, freshness of the information in the tokens and nodes, ...
- This method has not yet been implemented, nor tested in practice

Contribution

- Efficient method to build an optimal spanning tree with respect to community detection (for static networks only so far)
- Efficient method to detect communities

References

- Model Driven capabilities of the DA-GRS model, Arnaud Casteigts
- Dynamicity Aware Graph Relabeling Systems (DA-GRS), a local computation based algorithm to describe MANET algorithms, Arnaud Casteigts and Serge Chaumette
- Finding and evaluating community structure in networks, M. E. J. Newman and M. Girvan
- Detecting community structure in networks, M. E. J. Newman
- Robustness of community structure in networks, Brian Karrer, Elizaveta Levina, and M. E. J. Newman

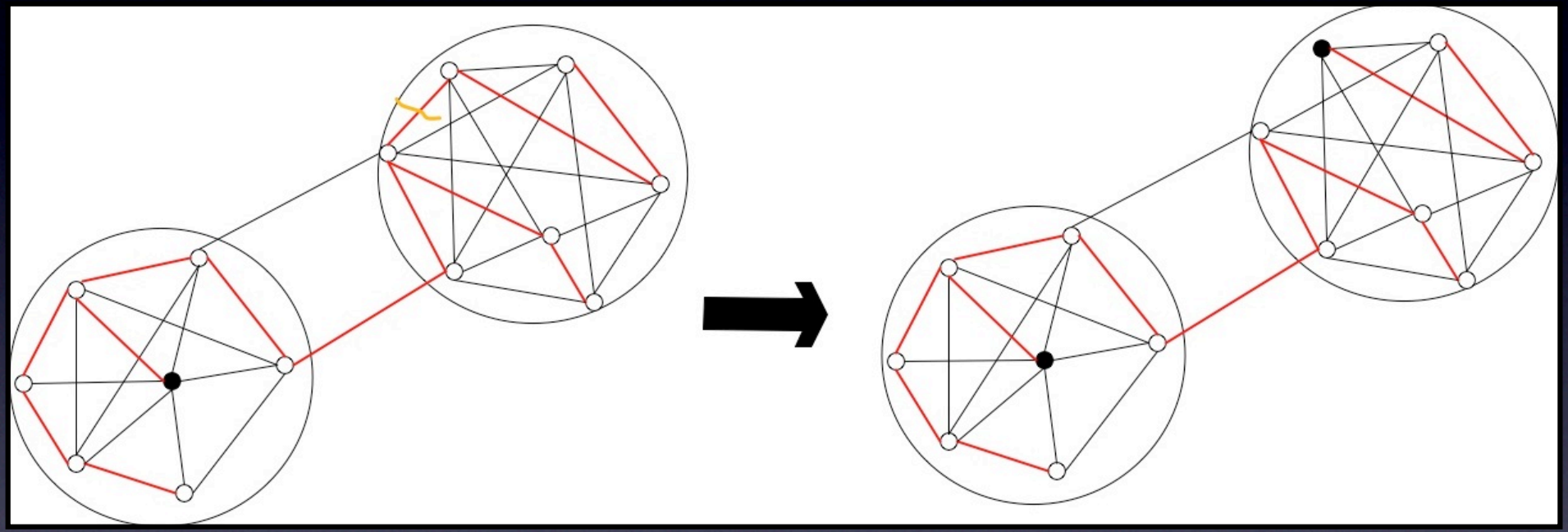
Thanks for listening

Dynamic Networks

- If a link breaks we get 2 subtrees
- The information about the subtree that is on the side of the node that has to regenerate a new token will be up to date and thus this node may regenerate a token with correct information
- The information in the token in the other subtree, will be updated as soon as the node that is on that side of the broken link gets the token, it will remove the information in the token about the subtree that has gone due to the broken link

Additional step for merging process

- Due to the fact that information in one of the tokens will not be immediately up to date
- Before merging two links, check that the two tokens have no common nodes
  - if this holds, merging is safe
  - if not, do not merge



Preference Table

- Build a preference table inside each node
- This table collects the preferable level for each of the neighbors of node n
- *How:*

  for the node n that we are considering:

  1. Add all of the one-hop neighbors of node n to a table with a count of 1.
  2. Determine all of the neighbors of one of our neighbors
  3. If some of these neighbors are in common with the neighbors of node n, increment the count for those.
  4. Repeat 2 and 3 for all neighbors

- Nodes with the higher count will be considered to be in the same community than n

DA-GRS rules

- 4 rules handling 4 events:
  1. Token regeneration
  2. Breaking of a link
  3. Merging of subtrees
  4. Circulation of the token
- Events can be passed and used at application layer