



Doctorat ParisTech

T H È S E

pour obtenir le grade de docteur délivré par
Télécom ParisTech
Spécialité : Électronique et Communications

présentée et soutenue publiquement par

Raphaël MASSIN

Soutenance prévue le 8 novembre 2016

**On the Clustering
of Mobile Ad Hoc Networks**

Directeur de thèse : **Christophe LE MARTRET**

Co-directeur de thèse : **Philippe CIBLAT**

Jury

Jean-Marie GORCE, Professeur des Universités, INSA-Lyon, France

Walid SAAD, Assistant Professor, Virginia Tech, United States of America

Philippe CIBLAT, Professeur, Telecom ParisTech, France

Christophe LE MARTRET, Expert Thales, HDR, Thales Com. & Sec., France

Christian BONNET, Professeur, Eurecom, France

Samson LASAULCE, Directeur de recherche CNRS, Supélec, France

Patrick MAILLÉ, Maître de Conférences, HDR, Telecom Bretagne, France

Jean-François MARCOTORCHINO, Professeur, LIP6, France

Rapporteur

Rapporteur

Directeur de thèse

Directeur de thèse

Examineur

Examineur

Examineur

Examineur

**T
H
È
S
E**

TELECOM ParisTech

école de l'Institut Mines-Télécom - membre de ParisTech

46 rue Barrault 75013 Paris - (+33) 1 45 81 77 77 - www.telecom-paristech.fr

Contents

General Introduction	7
1 Metrics for clustered networks performance evaluation	15
1.1 Introduction	15
1.2 Concept of global network cost function	16
1.2.1 Routing in ad hoc networks	16
1.2.2 Network cost functions to measure clustering algorithm performance	17
1.3 Network model	17
1.4 Case of additive metrics	18
1.4.1 Definitions	18
1.4.2 Case of J_0^L	18
1.4.3 Case of J_0^X	21
1.4.4 Examples for practical J_0^X	24
1.4.5 Application of J_0 to structured networks	25
1.4.6 Generalization to disconnected networks	29
1.5 Numerical analysis	30
1.5.1 Assessment methodology	30
1.5.2 Simulation setup	31
1.5.3 Results	33
1.6 Extending J_0^X to take cluster size into account	35
1.6.1 Definition	36
1.6.2 Example	36
1.6.3 Numerical illustration	37
1.7 Case of concave metrics: the throughput example	37
1.7.1 Definition	38
1.7.2 Shortest widest path	39
1.8 Conclusions	42
2 Distributed clustering algorithm with operational groups	43
2.1 Introduction	43
2.2 Network model	44
2.3 First step from the global network cost function to a distributed clustering solution	44
2.3.1 Definition of simple cost functions	44
2.3.2 Performance assessment	45
2.4 Towards a distributed clustering solution	48

2.4.1	First intermediate solution	48
2.4.2	Second intermediate solution	51
2.4.3	Third intermediate solution	52
2.4.4	Fourth intermediate solution	53
2.4.5	Final solution	53
2.4.6	Simulations	53
2.5	Distributed Clustering with Operational Group	54
2.5.1	Principles of DCOG	54
2.5.2	Functions related to DCOG	55
2.5.3	Detailed DCOG description	56
2.5.4	DCOG convergence	58
2.5.5	DCOG cluster cost function	59
2.5.6	Adaptation to mobility	59
2.6	Numerical results	59
2.6.1	Performance metrics	59
2.6.2	Reference clustering algorithms	61
2.6.3	Simulation setup	63
2.6.4	Asynchronous and synchronous modeling for DCOG	65
2.6.5	Asynchronous DCOG simulation results in static networks	68
2.6.6	Performance in static networks	71
2.6.7	Performance in dynamic networks	80
2.7	Conclusions	84
3	Distributed clustering and coalitional game theory	85
3.1	Introduction	85
3.2	Generic clustering algorithm based on coalition formation theory	87
3.2.1	Useful definitions related to coalition formation game theory	87
3.2.2	Generic coalition formation algorithm for clustering	88
3.2.3	Convergence properties	92
3.2.4	On the cluster constraints and utility function	92
3.3	Clustering algorithm for structured mobile ad hoc networks	93
3.3.1	Utility function	93
3.3.2	Heuristics for node selection	95
3.4	Clustering algorithm for unstructured mobile ad hoc networks	96
3.4.1	Utility function	96
3.4.2	Heuristics for node selection	96
3.5	Numerical results	97
3.5.1	Simulation setup	97
3.5.2	Case of structured ad hoc networks	98
3.5.3	Case of unstructured ad hoc networks	107
3.6	Conclusions	115
	General Conclusions and Perspectives	117
	Appendices	119

A	Brute force graph partitioning	121
A.1	Integer partitioning	121
A.2	Partitions of a graph	123
A.2.1	Trace of the algorithm	124
B	Proofs of Chapter 3	127
B.1	Proof of Result 3.1	127
B.2	Proof of Result 3.3	127
B.3	Proof of Result 3.4	128
B.4	Proof of Result 3.5	128
B.5	Proof of Result 3.6	129
B.6	Proof of Result 3.7	130
	Bibliography	131

General Introduction

Problem statement

The work presented in this PhD thesis has been produced thanks to the collaboration between the department "Communications et Électronique" (COMELEC) of Télécom ParisTech and the "Secteur Temps Réel" (STR) of Thales Communications and Security, in the framework of a "thèse en situation de travail", i.e., a PhD while working. This thesis is devoted to the study of ad hoc network clustering, which consists in building sets of nodes called clusters in order to introduce hierarchy in the network and thus improve its scalability.

An ad hoc network is an infrastructure-less multi-hop wireless network where each node participates in routing by forwarding data for other nodes. Those networks are self-organizing and are used when usual infrastructure based networks are not available or not suitable, e.g., in wireless sensor networks (WSN), vehicular networks (VANET) or military networks. In order to implement practical large ad hoc networks, gathering nodes in clusters was proposed in the early 80s for networking purposes, and then in the 90s to sustain good Quality of Service (QoS). Notably, clustering the network has been proposed in [1] as a way to enable the rapid deployment and dynamic reconfiguration of a scalable wireless network, with support of multimedia applications combining real-time and bursty traffic. In [2], a cluster-based multichannel communications system for VANETs is detailed. In this scheme, the elected cluster-head (CH) operates as the coordinator to collect/deliver the real-time safety messages within its own cluster, and forward the consolidated safety messages to the neighboring CHs. The CH also controls channel assignments for cluster members transmitting/receiving the non-real-time traffics, to improve radio resource allocation (RRA) efficiency. In [3], the authors propose a cluster-based framework to form a wireless mesh network in the context of open spectrum sharing, where the nodes are secondary users of spectrum. When a node not yet member of any cluster finds a channel that is unused by primary users, it becomes CH and invites neighbor nodes sharing the same channel to join the new cluster. The CH node is responsible for intra-cluster channel access control and inter-cluster communications. Then, by negotiating gateway (GW) nodes between clusters, clusters are interconnected into a large network. A clustered network is proposed in [4] to implement large cognitive radio networks, in which the clustering scheme works in conjunction with a network coded cognitive control channel in order to allow cognitive radio devices to opportunistically access the unused spectrum. From a network perspective, cluster-based hierarchical routing has been proved to introduce exponential savings in the amount of information to be stored and exchanged in a large ad hoc network [5], thus allowing the routing to scale. Cellular networks have also used an approach similar to clustering to enable the deployment of

femtocells [6].

Before introducing the different processes needed to build and operate a clustered ad hoc network, let us introduce a key assumption underlying the work presented in this document. Within a cluster, the members delegate intra-cluster RRA to their resource allocator (RA) node which can optimize it locally, similarly to what is done by base stations in cellular systems. A direct consequence of this assumption is that the intra-cluster communications benefit from a more efficient RRA process than the inter-cluster communications, highlighting the importance of building clusters in the right way. As a consequence, we need to impose basic topological properties to the clusters, that we will call *constraints* in the sequel:

- The induced subgraph of the cluster members must be connected, meaning that only intra-cluster links are required to perform communication between two cluster members.
- The cluster size, i.e., the number of cluster members, must be limited to control the signaling overhead.
- The cluster diameter, i.e., the length of the longest shortest path between any pair of the vertices in the induced subgraph of the cluster, must be limited, to prevent the delay and cost of communication between cluster members from increasing too much.

Operating a clustered ad hoc network necessitates running continuously and in parallel several processes. Initially, all nodes are standalone in their own singleton cluster. One of these processes is cluster building. During this process, all nodes detect their radio neighborhood and exchange enough information to form the clusters. Another process consists in allocating one set of radio resources, e.g., one frequency band or one CDMA code, to each cluster. To reduce the interference suffered by a cluster, its neighboring clusters should use sets of radio resources that are orthogonal or quasi-orthogonal to the one used by this cluster. Thanks to this approach, within a cluster the RA dynamically shares these resources with the members of its cluster without worrying of what happens in the neighbor clusters, thus simplifying the RRA. The two last processes are first to perform the RRA within clusters, and second to allocate the radio resources needed to perform inter-cluster communications. One may envision a joint optimization of these four processes. Because the protocols and algorithms used to implement an ad hoc network should be fully distributed, such optimization does not seem feasible to us. Therefore, we have decided to handle separately those different issues, this thesis focusing on the cluster building one. Note that the four processes mentioned above mainly concern the lower layers of the protocol stack, i.e., physical and data link layer, and possibly the network layer depending on how the responsibilities are shared between the data link and network layers.

Usually, in the literature, ad hoc networks are unstructured: there is no special organization of the network and all nodes share the same operational role, i.e., are equal to each other. In this thesis, beyond these usual unstructured networks, we also study the structured networks that have an inherent hierarchical structure associated with their *raison d'être*, and in which single nodes are gathered in *operational groups* (e.g., squad, section). Examples of such networks are public safety and military networks. For the sake of readability, in the sequel *group* refers to operational group. The existence of groups raises two major differences with respect to unstructured networks. Firstly, the traffic is strongly dependent on the network hierarchical

organization, being mostly concentrated within groups. Secondly, the nodes from the same group are very likely to move in the same direction. For these two reasons, we will show that it is beneficial to design clustering solutions that take into account group information, in order to provide more stable networks as well as better end-to-end QoS.

Most existing work about ad hoc network clustering has focused on unstructured networks. For example in [1] the authors propose the LID and HC clustering algorithms where nodes with the lowest identifier, respectively the highest degree, become CH. Non-CH nodes affiliate to their neighbor CH with the lowest identifier, respectively the highest degree. The stability of the clusters formed with LID or HC, which suffer from the *ripple effect*¹, has been improved in [7] thanks to the LCC mechanism that only performs re-clustering when multiple CH nodes become neighbors. The GDMAC algorithm detailed in [8] can be seen as a generalization of LCC, allowing up to K CH nodes to be neighbors. In the VOTE proposal [9], non-CH nodes join a CH only if the number of its cluster members is below a threshold, thus limiting the cluster size. Thanks to the knowledge of node location information, DGMA [10] attempts to estimate nodes relative mobilities and capture group mobility patterns to form stable clusters. A similar approach is followed by LACA [11], which proposes to build clusters using past, current and predicted nodes' positions thanks to the help of a learning automaton, and by GMCA [12] which uses a Gauss-Markov model to calculate the velocity and direction of the nodes. The novelty of SECA [13], lies in the introduction of link qualities (based on received signal strength) combined with the nodes relative mobilities to determine if a node becomes CH. Authors in [14] use centralized genetic algorithms and particle swarm optimization to select stable CH. The above examples constitute a representative extract of a rich literature which shows that a lot of research was done, and is still being done about clustering in unstructured ad hoc networks.

By contrast, only very few papers tackle the problem of clustering in structured networks. The authors in [15] introduce the type-based clustering algorithm (TCA). This clustering scheme associates a stability factor to each node and selects as CH the nodes that have the highest stability factor in a radio neighborhood. The stability factor takes group membership (identified thanks to the IP subnet of each node) into account. A limitation of TCA lies in the fact that two CH nodes cannot be neighbors. A direct consequence in dense networks is the formation of large clusters (with a lot of members). A second example is detailed in [16] which proposes a topology management mechanism for hierarchical group oriented networks, where groups are based on geographical locations.

According to the previous state of the art we have identified the two following interesting research tracks that would need further investigation and that will be addressed in this thesis:

- **Network performance metrics.** Usually the clustering solutions are compared using metrics focused on the technical details of the clusters themselves (e.g., number of clusters, number of cluster modifications, lifetime of CH nodes, amount of signaling, etc.). These metrics provide only indirect information about the QoS provided to the user. To fill this gap, some authors have selected various metrics derived from user throughput. In that case some assumptions are made about the type of *i*) medium access control (MAC), e.g., any version of WiFi, and *ii*) the user traffic profile. As detailed above, four different processes are required

¹The expression ripple effect describes a drawback of some clustering schemes for which a local cluster modification somewhere in the network leads to numerous cluster modifications in the whole network.

to build and operate a clustered network, and from the beginning of the thesis we wanted to be as agnostic as possible about the type of MAC used in the system. We only knew that inter-cluster communications would be implemented in a less efficient way than intra-cluster communications. Consequently, we decided to elaborate a new versatile metric which is able to take into account the cluster and group structures.

- **Distributed clustering algorithms.** In addition to the gap identified for the structured networks, the clustering algorithms proposed for unstructured networks do not fully satisfy our requirements. Indeed, none of these schemes simultaneously handle both cluster size and intra-cluster link quality. In addition, the availability of node locations and velocities is often a key underlying assumption of the most recent proposals, which cannot always be guaranteed. We thus identified the need for new distributed clustering algorithms to cover the case of structured networks, and in unstructured networks, to form clusters satisfying the topology constraints.

Additionally, the existing literature about ad hoc network clustering lacks a theoretical framework. Looking for such a framework, we found out that coalition game theory should be a relevant one. Indeed, it is a branch of game theory used to study the behavior of players when they cooperate among themselves [17]. Considering coalitions as clusters, we thus decided to use coalition game theory for our research of new clustering algorithms.

Coalition game theory has been used for many different applications. For example in [18] an algorithm is proposed for cognitive networks with primary and secondary users (SU) to form disjoint coalitions of SUs, in order to perform collaborative sensing while maximizing the utility in terms of probability of detection and accounting for a false alarm cost. In [19] the players are small cell base stations (SBS) that cooperate to share their radio resources, in order to deal with OFDMA downlink co-tier interference suffered by the small cell user devices from neighboring SBSs. The SBSs form overlapping coalitions and dedicate part of their frequency resources (OFDMA subchannels) to each coalition they belong to, in order to maximize the sum rate. Proposals suited to VANET have also been published. For example, in [20], the roadside units (RSU) receive a payoff based on the amount and class of data sent to vehicles moving in their area. Considering that the moving vehicles can use the underlying vehicle-to-vehicle (V2V) network to exchange data received from the RSUs, the RSUs can form coalitions in order to diversify the classes of data they transmit to the vehicles. This cooperation allows them to increase their revenue. In [21], the vehicles use a coalition formation algorithm to build clusters in order to optimize the V2V exchange of context information about road conditions or driving safety. Within a cluster, the CH gathers information from all its cluster members, performs data fusion and then sends back the result. The authors of [22] propose a distributed algorithm to achieve cooperative communications in ad hoc networks (with multiple sources and one destination, similarly to a WSN), in order to increase the achievable rate. Within a coalition the communications are performed over two phases: the broadcasting and the cooperation phases. During each slot of the broadcasting phase, one coalition member performs broadcast transmission, while the other coalition members are listening. Then, during the cooperation phase, all coalition members transmit a linearly coded signal of all signals received during the previous broadcasting phase, and the destination performs multiuser detection to extract the per node signal. Device-to-device (D2D) communications are the subject of [23] which proposes

a distributed resource management scheme to jointly solve the problem of resource sharing mode selection and spectrum sharing. Here sharing mode can be *i*) cellular mode, *ii*) dedicated D2D mode and *iii*) hybrid mode, where D2D reuse the resources of some cellular links. Single or pairs of cellular users form coalitions to improve the achievable sum rate of the D2D system. These examples confirm that coalitional game theory is a relevant framework to study ad hoc network clustering.

Outline and contributions

In this section, we give the thesis outline, and we mention the most important results.

In Chapter 1 we define network cost functions to benchmark clustering solutions from a system point of view. We want these cost functions to take into account the **costs of communications along end-to-end paths**. To do that, we start by recalling that the quality of network paths depend on the routing capabilities in the network layer, and justify why a network cost function used to assess the system performance should take them into account. We then consider in detail the case of additive metrics (e.g., delay) and define a **novel network cost function J_0 incorporating the traffic structure and the inter-cluster communication cost**. We then apply it to structured networks, and perform a detailed numerical analysis showing that **the clustering solutions providing the best QoS to the end-user depend on the group structure**. Then we extend J_0 to better handle the constraint that cluster size should be limited. Finally we define the network cost function T_0 , derived from J_0 , and suited to the throughput metric.

In Chapter 2 we design a **distributed clustering algorithm suited to structured networks**. First, we identify the key parameters that should be used to build good clusters. Then, we detail the **distributed clustering with operational groups algorithm**, denoted **DCOG**. This algorithm is run at the cluster level and is designed to achieve the following properties: each cluster includes the largest possible number of members of some operational groups, each cluster size is the closest possible to a given maximum, and the diameter of each cluster is limited to a maximum. A key characteristic of DCOG, which differentiates it from the conventional clustering schemes, is that it **does not need to resort to the notion of CH node**. After proving the **theoretical convergence of DCOG**, we compare by simulation its performance against five other clustering algorithms. The first two algorithms are GDMAC and VOTE, chosen from the literature, and the last three are extensions of GDMAC and VOTE that we designed to better handle the network group structure. Our **simulations** show that DCOG leads to better application level performance (measured thanks to J_0) and offers a better stability to mobility.

In Chapter 3, we **revisit DCOG within the coalitional game theory framework**. After specifying how we use the common coalitional game theory notions of **coalition utility, value and cost**, we formally identify coalitions as clusters and players as nodes, and define a **generic coalition formation algorithm for clustering**. This algorithm is run at the node level, and is split in two procedures. The first one performs **switch operations**, i.e., the transfer of some nodes between two clusters, such that the sum of cluster values is always increased, thus

guaranteeing **convergence to a Nash-stable partition** of the network. The second one is used when, due to mobility, a cluster no longer satisfies some topology constraints. In that case a switch operation may have to be performed to restore a correct topology. The important design parameters of this algorithm are *i*) the heuristics used to select the candidate sets of nodes involved in the switch operations, and *ii*) the utility and the associated cost functions used to calculate the switch operation gains. Then we customize this generic algorithm for the two cases of **structured and unstructured network**, leading to the **COG and CLQ algorithms** respectively. In each case, we define specific utility functions and node selection heuristics. Finally, we assess the performance of our algorithms by simulation. We first show that **COG builds stable clusters that include most members of one or several groups**. Then, in unstructured networks, we show that **CLQ performs better than LCC, SECA and VOTE**, three clustering schemes from the literature.

Is clustering really clustering?

Clustering algorithms are also used for several other applications or contexts than the ad hoc networks considered in this thesis. Even if behind all these concepts, the idea is always to gather elements in groups called clusters, it appears that they usually differ along several lines such as the underlying graph topology hypothesis, the theoretical tools used to solve the problem, and the capability to enforce some constraints. We give here some different clustering examples (not meant to be exhaustive) and try to stress the main differences with our problem.

Clustering can be found for instance in *data analysis*. Let us quote A. K. Jain in 1999 [24]: *”Clustering is the unsupervised classification of patterns (observations, data items, or feature vectors) into groups (clusters). The clustering problem has been addressed in many contexts and by researchers in many disciplines; this reflects its broad appeal and usefulness as one of the steps in exploratory data analysis.”* With the rise of *big data*, this quote is even more true than when it was initially written. Within the large scope of data analysis domain, it is worth mentioning the field of *community detection* (CD) [25], where searching for *communities* in a graph is often referred to as clustering. This example of clustering was brought to our attention thanks to Pierre Borgnat (ENS Lyon) and our colleague Jean-François Marcotorchino (Scientific Director at Thales Communications & Security).

Another example of clustering can be found in the signal processing community [26] where the authors use a *diffusion LMS* algorithm drawn from the field of *distributed adaptive estimation*. It is based upon the fact that nodes receiving noisy versions of a given vector of parameters can converge to a consistent estimation of the parameters by exchanging information between nodes. In this paper, they extend this principle to the case where different groups of nodes receive different vectors of parameters. They observe that the algorithm converges towards a state where information exchanged between nodes that are not receiving the same parameters vanishes (whereas within a group, the nodes still continue to estimate correctly the parameters they receive) and thus separates naturally the set of nodes in clusters.

In CD (unweighted) graphs, vertices are connected when they *share a given relation* (e.g., friends on social network). Thus researching a community can be roughly explained as identifying a group of nodes that are *more* in relationship altogether than with the rest of the remaining part of the graph. In other words, there must be more edges inside the community than edges linking

vertices of the community with the rest of the graph. In wireless network (unweighted) graphs, connection between vertices (nodes) follows from the radio coverage and thus depends on the geographical nodes' distribution. In that case, the *relation* is equivalent to *the two nodes are in range*. Moreover, two nodes in range (i.e., two connected vertices) do not necessarily exchange information, which constitutes a difference with the graphs used in community detection. More important, the main difference between the two clustering contexts is that in the case of ad hoc networks, we impose constraints to the resulting cluster topology such as maximal size, maximal diameter, etc., whereas in detection community, the clusters are what they are. To illustrate this difference, let us take the extreme case (but not unrealistic in communications) of a fully connected wireless network graph (leading to a clique). Any community detection algorithm will find obviously only one community, whereas the clustering algorithm will have to find as many clusters as necessary to enforce the constraints (e.g., for a 200 node network, and a maximal cluster size of 10, the minimum number of clusters is equal to 20). Moreover, we also address the problem of clustering taking into account the fact that nodes may follow specific organizations. As a consequence, the clustering algorithms are tailored to include the maximum number of members of the same group inside the same cluster. Up to our knowledge, clustering algorithms from community detection domain do not cover such case. Lastly, it appeared to us that most of the metrics used in CD, and more generally in data analysis, require the knowledge of the whole graph, leading to *centralized* algorithms. In our case, we are looking for *non-centralized* algorithms, i.e., working with local (thus partial) information of the graph. For instance, let us consider the *modularity* measure that is a popular metric used to identify good communities [27]. Using this measure to valued graph in the context of ad hoc networks (e.g. using weights proportional to the link capacity) might work provided constraints can be handled. But computing the modularity needs to know the number of links of the whole graph. Thus, this cannot lead to non-centralized solutions since collecting this information across the network would be prohibitive. We identified recent work in [28–30], where the authors modified the *label propagation* algorithm [31] (issued from the CD domain) to perform distributed clustering in mobile ad hoc networks. However, these proposals do not take into account the cluster size and diameter constraints.

Regarding the clustering example from the signal processing paper, it appears that the context is different from ours and suffers from the main issues listed above for CD and data analysis (except the fact that it is distributed).

To conclude, since the thesis was quickly oriented towards the use of the coalition game theoretical framework, looking thoroughly to clustering algorithms from other domains as the ones mentioned above was clearly out of scope. However, our quick review shows that the other clustering algorithms cannot be applied directly to our problem and suffer from several drawbacks (not distributed, not taking constraints into account, not taking group structure into account, ...). Thus, even though a deep investigation of existing solutions along with compulsory adaptations to our context might lead to interesting algorithms, it is left for further studies (for instance, tweaking the label propagation algorithm to handle the constraints).

Publications

The work presented in this manuscript has led to the following publications:

Journal paper

- R. Massin, C. J. Le Martret, and P. Ciblat, "A Coalition Formation Game for Distributed Node Clustering in Mobile Ad Hoc Networks," under preparation for *IEEE Transactions on Wireless Communications*.

Conference paper

- R. Massin, C. J. Le Martret, and P. Ciblat, "A network cost function for clustered ad hoc networks: application to group based systems," in *25th IEEE International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, Washington D.C., USA, Sept. 2014.
- R. Massin, C. J. Le Martret, and P. Ciblat, "Distributed Clustering Algorithms in Group-Based Ad Hoc Networks," in *23rd European Signal Processing Conference (EUSIPCO)*, Nice, France, Sept. 2015 (invited paper).
- R. Massin, C. J. Le Martret, and P. Ciblat, "Un algorithme de clusterisation distribué pour les réseaux ad hoc structurés", in *Colloque GRETSI*, Lyon (France), Sept. 2015.
- R. Massin, C. J. Le Martret, and P. Ciblat, "Distributed Clustering Algorithm in Dense Group-Based Ad Hoc Networks," in *16th IEEE Mediterranean Ad Hoc Networking Workshop (MedHocNet)*, Vilanova i la Geltru, Spain, June 2016.

International patent

- R. Massin, and C. J. Le Martret, "Procédé pour générer des groupes au sein d'un réseau radio ad hoc," European patent application EP16155087.6, February 2016.

Chapter 1

Metrics for clustered networks performance evaluation

The work presented in this chapter has been partially published in [32].

1.1 Introduction

Ad hoc networks clustering has attracted and is still attracting a lot of interest and a lot of algorithms have been proposed. The performance of these clustering schemes are evaluated using various metrics. A first category of metrics is dedicated to assess the performance in static networks. Examples of metrics of this kind are the number of clusters (similar to the cluster size) [8, 9, 13, 15, 16], the percentage of nodes that are not connected to the network due to clustering [16], the average path size for any pair of source and destination [16]. Another family of metrics is used to measure the performance in dynamic networks. In this category can be found the number of cluster modifications [9, 15], the lifetime of CH nodes [8, 10, 13], the duration a non CH node is affiliated to a given CH [8], the average number of cluster re-affiliations per second [10], the number of times a node becomes CH or loses its CH state, or a non CH node affiliates to another CH [8, 15]. Some metrics fit both categories, such as the one measuring the number of signaling messages sent [8, 10]. These metrics are focused on the technical details of the clustering solutions and do not give any detail about the QoS provided to the user by the network, which is the most interesting for us.

The end-to-end throughput metric from [1] is more attractive. It is defined as follows:

$$\hat{\text{Th}} := \sum_{i=1}^Q f_i \frac{\text{Th}_i}{L_i},$$

with Q the number of connected components in the graph, Th_i the cumulated link throughput of the i th connected component \mathcal{Q}_i . L_i the average hop length in \mathcal{Q}_i and f_i the fraction of node pairs interconnected within \mathcal{Q}_i . In a connected network this metric is written $\hat{\text{Th}} = \text{Th}/L$ with Th the cumulated link throughput in the network and L the average hop length in the network. This expression neither takes into account any traffic profile nor the cluster structure of the network.

Thus the aforementioned metrics do not provide any insight on the effect of the clustering solutions on the user applications. The goal of this chapter is therefore twofold: *i*) specify

a way to benchmark clustering solutions from a system point of view, and *ii*) in the case of structured networks, prove the importance of using group membership information to build clusters. Therefore we define several network cost functions based on additive (e.g., delay) or concave (e.g., link throughput) metrics incorporating the traffic structure and the inter-cluster communication cost. Thanks to these functions we show that in the structured networks, the clustering solutions providing the best QoS to the end-user strongly depend on the group structure. Additionally, we find that to provide the best QoS to the user, routing should take the cluster structure into account.

This chapter is organized as follows. In Section 1.2 we first recall some key features of routing in ad hoc networks, and introduce the concept of network cost functions to measure the performance of clustering algorithms. We introduce the notations and the network model in Section 1.3. In Section 1.4 we define the generic network cost function J_0 dedicated to additive metrics, and apply it to structured networks. In Section 1.5 we provide detailed numerical results showing that in such networks, the clustering solutions providing the best QoS should build clusters close to the groups. Then in Section 1.6 we propose an improved version of J_0 , before deriving in Section 1.7 the cost function T_0 dedicated to the link throughput concave metric. We conclude this chapter in Section 1.8.

1.2 Concept of global network cost function

1.2.1 Routing in ad hoc networks

Within networks, the goal of routing is to allow the end-to-end forwarding of packets from a source to one or more destinations. To find these routes across the network various procedures exist. Let us consider the example of hop by hop routing: for each node along the route from a source to a destination, hop by hop routing finds the next hop to the destination. This next hop can be found using a shortest path algorithm whose goal is to find a route whose cost is the lowest. One well known shortest path algorithm is the Dijkstra's shortest path (DSP) algorithm [33]. The information used by DSP are the nodes and links of the network. Depending on the amount and accuracy of information, the quality of the routes found by DSP is high or low. If only a binary (on/off) information is known about links, then DSP finds shortest routes in term of number of hops. Conversely, if a weight is associated with each link, a route cost can be calculated as the sum of link weights along this route. To illustrate why the latter approach leads to better results, let us take an example. Consider a network with three nodes $\{N_1, N_2, N_3\}$ and links $\{(N_1, N_2), (N_2, N_3), (N_1, N_3)\}$. Looking for a shortest path from node N_1 to node N_3 , the direct one-hop route is shortest compared to the route through N_2 that is two-hop long. Now if the cumulated weight of going through link (N_1, N_3) is larger than the one of going through links (N_1, N_2) and (N_2, N_3) , then from a cost perspective, the two-hop route is better than the direct one. Therefore, knowing the weight of links in addition to their existence allows to find better routes w.r.t. the chosen QoS criterion. Thus, depending on the QoS requirements over a network, different routing protocols should be used, each one using a various amount of input information and thus finding routes with different qualities. An example of protocol whose routes minimize the number of hops is OLSR [34]. An extension of this protocol is QOLSR [35], that finds routes with the lowest end-to-end delay.

1.2.2 Network cost functions to measure clustering algorithm performance

Our goal is to define network cost functions that can be used to assess the performance of the system from application layer to physical layer, in terms of QoS offered to network users. These cost functions are defined in order to take into account the costs of communications from all nodes to all nodes along the end-to-end paths calculated by the routing in the network layer. As discussed previously, the network paths found by a routing protocol depend on the available link information, which can be provided by other protocol layers, i.e., by the data link layer and potentially the physical layer. Thus, depending on the capability of the protocol stack different cost functions should be used to assess the system performance.

In the following, each cost function are identified by a superscript L or X to determine the amount of link information that can be used when calculating the routes. The first cost function defined is J_0 , leading to J_0^L and J_0^X . When there is no superscript, this means that what is being discussed is applicable for both versions of the cost function. The superscript L means that only the link information (binary or link weight) can be used by the routing protocol. If binary information is available, then shortest paths minimize the hop count. This basic information can be acquired by the network layer itself (e.g., using a HELLO protocol such as the one defined in [34]) or can be provided by the data link layer. Richer link weights information can be measured by the network layer itself (e.g., in [35] the routing protocol measures link delays). It can also be provided by the data link layer which uses metrics coming from the physical layer. These metrics, such as received signal strength or signal interference noise ratio (SINR), are measured by the radio receiver during the communications. The superscript X (for cross-layer) means that in addition to link information, the cluster structure is also known, meaning that when calculating the shortest paths, the intra-cluster or inter-cluster quality of each link is taken into account. This is important because in clustered networks the efficiencies of communication over intra-cluster and inter-cluster links are different. This additional information is provided by the data link layer (in charge of building the clusters).

Several types of metrics can be used to calculate a shortest path. The simplest metrics are called additive metrics, for which the cost of a route is the sum of its link weights. An example of this type of metric is the delay. Some other metrics cannot be cumulated. For example the throughput of a route is not the sum but the lowest of its link throughputs. Throughput is a concave metric [36]. There are also some multiplicative metrics, such as for example the packet loss rate. In this chapter we handle in details the case of additive metrics, then propose some initial results concerning concave metrics. The case of multiplicative metrics is left for future work.

1.3 Network model

We consider a graph \mathcal{G} defined by its set of nodes \mathcal{V} and its set of edges \mathcal{E} . The number of nodes of \mathcal{G} is $N := |\mathcal{V}|$, where $:=$ stands for *by definition*. The number of edges of \mathcal{G} is $M := |\mathcal{E}|$. Two nodes i and j are neighbors if $(i, j) \in \mathcal{E}$. Without loss of generality, the concepts developed in this chapter first concern connected¹ undirected graphs. A generalization to disconnected graphs is detailed in Section 1.4.6. The set of all partitions p of \mathcal{G} is called \mathcal{P} . In our model the parts of a partition are identified to the clusters. The weight of edge (i, j) , $\forall (i, j) \in \mathcal{E}$ is

¹In a connected graph there exists a path between any pair of vertices.

noted $w_{i,j}$. The weight $w_{i,j}$ is a dimensionless quantity associated with the quality of link (i, j) . For example it can be a number of transmissions required to achieve a target packet error rate (assuming an ARQ scheme). The set of groups \mathcal{O} is defined as $\{\mathcal{O}_1, \dots, \mathcal{O}_T\}$ with T the number of groups. Let us note m_t the size of group \mathcal{O}_t . Each node belongs to only one group, i.e. $\forall t_1 \neq t_2, \mathcal{O}_{t_1} \cap \mathcal{O}_{t_2} = \emptyset$, thus $\sum_{t=1}^T m_t = N$, and $\cup_{t=1}^T \mathcal{O}_t = \mathcal{V}$. A partition p of \mathcal{G} contains N_c clusters noted \mathcal{C}_k with $k \in \{1, \dots, N_c\}$. The size, or number of members, of cluster \mathcal{C}_k is n_k . Each node belongs to only one cluster, i.e., $\forall k \neq \ell, \mathcal{C}_k \cap \mathcal{C}_\ell = \emptyset$, thus $\sum_{k=1}^{N_c} n_k = N$, and $\cup_{k=1}^{N_c} \mathcal{C}_k = \mathcal{V}$. The diameter² of the induced subgraph of the cluster \mathcal{C}_k is noted d_k .

1.4 Case of additive metrics

In this section we define the network cost function J_0 used to measure and compare the performance of clustering solutions in clustered ad hoc networks. More precisely this function measures the quality of a network partition using end-to-end path calculations with additive metrics (e.g., delay), and takes into account the fact that inter-cluster and intra-cluster communications have different costs. As a byproduct, this function is very useful for evaluating the benefit expected through the use of operational group information for obtaining the clustering solution.

1.4.1 Definitions

Before analyzing separately the cases of J_0^L and J_0^X , let us introduce definitions and concepts common to both functions.

To assess the quality of a partition p of \mathcal{G} we define a function J_0 which represents the average cost of communications between all pairs of nodes $(i, j) \in \mathcal{V}^2$. It is defined as:

$$J_0(p) := \frac{1}{N} \sum_{(i,j) \in \mathcal{V}^2} \pi_{j|i} \cdot J_0(p, i, j), \quad (1.1)$$

where the factor $1/N$ embodies the fact that all nodes i have equal probability to transmit, $J_0(p, i, j)$ is the transmission cost between node i and node j , and $\pi_{j|i}$ is the probability that node i chooses node j as a destination. By convention $\pi_{i|i} = 0$, and $\sum_{j \in \mathcal{V}} \pi_{j|i} = 1, \forall i \in \mathcal{V}$.

Finding the best clustering of the network is equivalent to finding the set of best partitions \mathcal{P}' defined as:

$$\mathcal{P}' := \arg \min_{p \in \mathcal{P}} J_0(p). \quad (1.2)$$

1.4.2 Case of J_0^L

In this section the shortest paths are calculated using the link weights information only.

1.4.2.1 Definitions

Let us now explain how $J_0^L(p, i, j)$ is elaborated. Firstly, we assume that the routing process selects the shortest paths to establish the communications in the network. The shortest path between node i and node j is defined here as the set $\mathcal{S}_L(i, j) = ((i, i_1), (i_1, i_2), \dots, (i_{K-1}, i_K), (i_K, j))$

²The diameter of a graph is the length of the longest shortest path between any pair of its vertices.

for which the cumulated weights along this path, $h_L(i, j)$, defined as:

$$h_L(i, j) := \sum_{(i', j') \in \mathcal{S}_L(i, j)} w_{i', j'}, \quad (1.3)$$

is minimum. Note that $\mathcal{S}_L(i, j)$ is independent of the partition p . Fig. 1.1 provides an example of shortest path $\mathcal{S}_{1,4}$ between source node 1 and destination node 4.

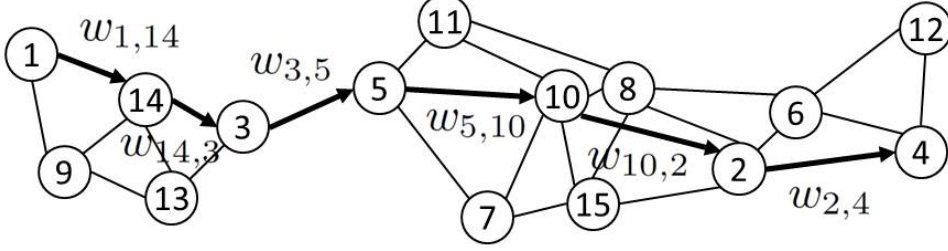


Figure 1.1: Example of multi-hop shortest path.

In clustered networks, inter-cluster communications can be implemented using a medium access control (MAC) different from the one used for intra-cluster communications. Within a cluster a RA node optimizes locally the RRA on behalf of the cluster members. Conversely, inter-cluster RRA is done in a more distributed way (e.g., among the RAs of neighbor clusters) and is thus more difficult to optimize. Therefore, we reasonably assume that the costs of intra-cluster and inter-cluster communications are different. Consequently a path $\mathcal{S}_L(i, j)$ is split into the two subsets $\hat{\mathcal{S}}_L(p, i, j)$ of intra-cluster links and $\tilde{\mathcal{S}}_L(p, i, j)$ of inter-cluster links, leading respectively to the cumulated weights $\hat{h}_L(p, i, j)$ and $\tilde{h}_L(p, i, j)$:

$$\hat{h}_L(p, i, j) := \sum_{(i', j') \in \hat{\mathcal{S}}_L(p, i, j)} w_{i', j'}, \quad (1.4)$$

$$\tilde{h}_L(p, i, j) := \sum_{(i', j') \in \tilde{\mathcal{S}}_L(p, i, j)} w_{i', j'}. \quad (1.5)$$

Following (1.3)-(1.4)-(1.5), we have:

$$h_L(i, j) = \hat{h}_L(p, i, j) + \tilde{h}_L(p, i, j).$$

For example in Fig. 1.2, $\hat{h}_L(p, 1, 4) = w_{1,14} + w_{14,3} + w_{5,10} + w_{2,4}$ and $\tilde{h}_L(p, 1, 4) = w_{3,5} + w_{10,2}$, with $p = \{\{1, 3, 9, 13, 14\}, \{5, 7, 8, 10, 11, 15\}, \{2, 4, 6, 12\}\}$.

In order to account for the difference between intra and inter-cluster communications, we define the cost from node i to node j as the weighted sum of $\hat{h}_L(p, i, j)$ and $\tilde{h}_L(p, i, j)$:

$$J_0^L(p, i, j) := \hat{\gamma} \cdot \hat{h}_L(p, i, j) + \tilde{\gamma} \cdot \tilde{h}_L(p, i, j), \quad (1.6)$$

where $\hat{\gamma} > 0$ is the cost associated with the intra-cluster communications, and $\tilde{\gamma} \geq \hat{\gamma}$ the cost associated with the inter-cluster communications. Note that (1.4)-(1.5)-(1.6) clearly show that J_0^L is applicable to additive metrics.

From (1.1), noting $\gamma := \tilde{\gamma}/\hat{\gamma}$ and summing (1.6) over all pairs of nodes we get:

$$J_0^L(p) = \frac{\hat{\gamma}}{N} \sum_{(i,j) \in \mathcal{V}^2} \pi_{j|i} \cdot (\hat{h}_L(p, i, j) + \gamma \cdot \tilde{h}_L(p, i, j)). \quad (1.7)$$

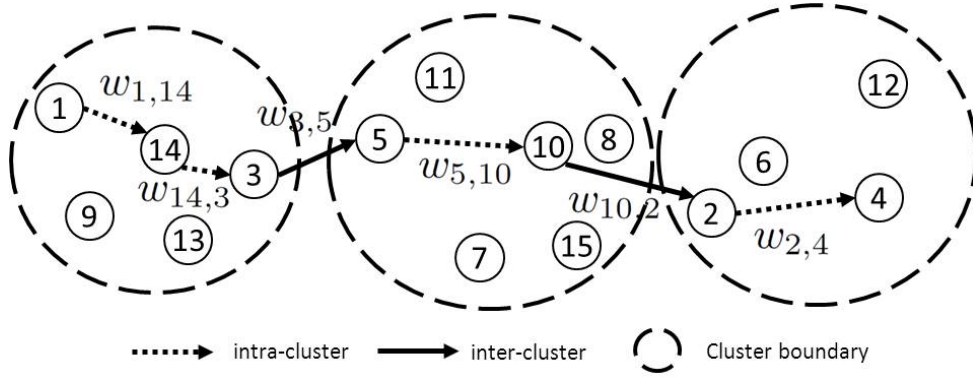


Figure 1.2: Multi-hop shortest path with cluster boundaries.

One can remark that the factor $\hat{\gamma}/N$ in (1.7) plays no role in the optimization problem of (1.2).

Because $h_L(i, j)$ is independent of the way the network is clustered, it is useful to make it appear in (1.7) leading to:

$$J_0^L(p) = A + (\gamma - 1) \cdot B(p), \quad (1.8)$$

with

$$A := \frac{\hat{\gamma}}{N} \sum_{(i,j) \in \mathcal{V}^2} \pi_{j|i} \cdot h_L(i, j),$$

$$B(p) := \frac{\hat{\gamma}}{N} \sum_{(i,j) \in \mathcal{V}^2} \pi_{j|i} \cdot \tilde{h}_L(p, i, j).$$

So far, the expressions are functions of probabilities $\pi_{j|i}$ which makes the model very general and applicable to several contexts. Particular values for $\pi_{j|i}$ are considered in Section 1.4.5 in order to take into account the hierarchical organization of some ad hoc networks.

1.4.2.2 Theoretical results and additional constraints

Thanks to (??) we can prove the following result.

Result 1.1 *Considering J_0^L , the solutions of the problem in (1.2) are:*

1. If $\gamma = 1$, $\mathcal{P}' = \mathcal{P}$, and $\forall p \in \mathcal{P}$, $J_0^L(p) = A$.
2. If $\gamma > 1$, $\mathcal{P}' = \{\mathcal{V}\}$.

The first part of Result 1.1 means that when $\gamma = 1$, the way clusters are built is not important. This case is not interesting in practice since there is always a difference of cost between intra and inter-cluster communications. When $\gamma > 1$, the best and trivial solution is to build one cluster corresponding to the whole network. This solution is not acceptable because of the constraints related to the size of the clusters. If the cluster is too large, then the RRA becomes complex and the goal of seeking simple intra-cluster RRA cannot be fulfilled. Consequently, we add constraints on the clusters and define \mathcal{P}^c the subset of valid partitions as follows:

$$\mathcal{P}^c = \{p \in \mathcal{P} \text{ s.t. } p \text{ satisfies } C_1, C_2, C_3\}, \quad (1.9)$$

with

$$\begin{aligned} C_1 &: \mathcal{C}_k \text{ is connected } \forall k \in \{1, 2, \dots, N_c\}, \\ C_2 &: n_{min} \leq n_k \leq n_{max} \forall k \in \{1, 2, \dots, N_c\}, \\ C_3 &: d_k \leq d_{max} \forall k \in \{1, 2, \dots, N_c\}. \end{aligned}$$

Firstly, C_1 ensures that each cluster is a connected subgraph of \mathcal{G} , allowing intra-cluster communication between all the cluster members. Then C_2 forces the number of cluster members to be neither too small nor too large, which makes sense from a RRA point of view. Finally C_3 prevents nodes from the same cluster from being too far (in number of hops) from the RA (in charge of RRA). The values of the parameters n_{min} , n_{max} and d_{max} depend on the RRA process.

Now, our goal is to find the set of partitions \mathcal{P}_L^* solving the following problem:

$$\mathcal{P}_L^* = \arg \min_{p \in \mathcal{P}^c} J_0^L(p). \quad (1.10)$$

The two following results detail some properties of \mathcal{P}_L^* .

Result 1.2 *The partial ordering between J_0^L values of different partitions does not depend on γ .*

Proof In (??), A does not depend on the partition whereas B does. Consequently (??) can be written:

$$J_0^L(\gamma, p) = A + (\gamma - 1)B(p). \quad (1.11)$$

Let p_1 and p_2 two different partitions of \mathcal{P} such that:

$$\begin{aligned} J_0^L(\gamma, p_1) &< J_0^L(\gamma, p_2), \\ \Leftrightarrow A + (\gamma - 1)B(p_1) &< A + (\gamma - 1)B(p_2), \\ \Leftrightarrow B(p_1) &< B(p_2). \end{aligned}$$

The last equation does not involve γ , which proves the result. ■

Result 1.2 induces the following result:

Result 1.3 *\mathcal{P}_L^* is independent of γ .*

Proof Application of Result 1.2 for optimal partitions. ■

1.4.3 Case of J_0^X

In this section we consider that both link weights and cluster structure are available to calculate the shortest paths. We proceed like in Section 1.4.2 to study the case of J_0^X .

1.4.3.1 Definitions

Let us define graph $\mathcal{G}(p)$ as the set of nodes \mathcal{V} connected by edges in \mathcal{E} whose weights are $\hat{\gamma} \cdot w_{i,j}$ if (i, j) is an intra-cluster link, and $\tilde{\gamma} \cdot w_{i,j}$ if (i, j) is an inter-cluster link. The shortest path between nodes i and j in $\mathcal{G}(p)$ is denoted by $\mathcal{S}_X(p, i, j)$, with cost $h_X(p, i, j)$. This shortest

path can be split into its intra-cluster part $\hat{\mathcal{S}}_X(p, i, j)$ and its inter-cluster part $\tilde{\mathcal{S}}_X(p, i, j)$ with respective costs $\hat{h}_X(p, i, j)$ and $\tilde{h}_X(p, i, j)$:

$$h_X(p, i, j) := \hat{h}_X(p, i, j) + \tilde{h}_X(p, i, j), \quad (1.12)$$

with:

$$\hat{h}_X(p, i, j) := \sum_{(i', j') \in \hat{\mathcal{S}}_X(p, i, j)} \hat{\gamma} \cdot w_{i', j'} \quad (1.13)$$

$$\tilde{h}_X(p, i, j) := \sum_{(i', j') \in \tilde{\mathcal{S}}_X(p, i, j)} \tilde{\gamma} \cdot w_{i', j'}. \quad (1.14)$$

Similarly to what we did for (1.6) we define $J_0^X(p, i, j)$ as follows:

$$J_0^X(p, i, j) := h_X(p, i, j). \quad (1.15)$$

Here $\hat{\gamma}$ and $\tilde{\gamma}$ are already included in $\hat{h}_X(p, i, j)$ and $\tilde{h}_X(p, i, j)$ respectively, which justifies that $J_0^X(p, i, j)$ is equal to $h_X(p, i, j)$.

From (1.1), and summing (1.15) over all pairs of nodes we get:

$$J_0^X(p) := \frac{1}{N} \sum_{(i, j) \in \mathcal{V}^2} \pi_{j|i} \cdot J_0^X(p, i, j). \quad (1.16)$$

1.4.3.2 Theoretical results

Let us now prove that Result 1.1 applicable to J_0^L is also valid for J_0^X . First we need the following two intermediate results.

Result 1.4 $\forall p \in \mathcal{P}$ with $p \neq \{\mathcal{V}\}$, when $\gamma > 1$ there is at least one pair of nodes $(i, j) \in \mathcal{V}^2$ such that $h_X(p, i, j) > h_X(\{\mathcal{V}\}, i, j)$.

Proof Because $p \neq \{\mathcal{V}\}$, partition p has at least two clusters. Let us choose clusters \mathcal{C}_1 and \mathcal{C}_2 and two 1-hop neighbor nodes i and j such that $i \in \mathcal{C}_1$ and $j \in \mathcal{C}_2$, and the shortest path $\mathcal{S}_X(\{\mathcal{V}\}, i, j)$ from i to j is link (i, j) : $h_X(\{\mathcal{V}\}, i, j) = \hat{\gamma} \cdot w_{i, j}$. Consider the shortest path $\mathcal{S}_X(p, i, j)$ from i to j in $\mathcal{G}(p)$.

Case 1: If $\mathcal{S}_X(p, i, j) = (i, j)$ then $h_X(p, i, j) = \tilde{\gamma} \cdot w_{i, j}$ and because $\tilde{\gamma} > \hat{\gamma}$, we have $h_X(p, i, j) > h_X(\{\mathcal{V}\}, i, j)$.

Case 2: If $\mathcal{S}_X(p, i, j) \neq (i, j)$ then let nodes $\{i_1, i_2, \dots, i_{K-1}, i_K\} \subset \mathcal{V}$, with $i_1 = i$ and $i_K = j$ such that $\mathcal{S}_X(p, i, j) = (i_1, \dots, i_K)$. $\exists \alpha \in \{1, \dots, i_K - 1\}$ such that nodes i_α and $i_{\alpha+1}$ belong to two different clusters. Because link $(i_\alpha, i_{\alpha+1})$ belongs to a shortest path in $\mathcal{G}(p)$, it is also a shortest path between i_α and $i_{\alpha+1}$, and $h_X(p, i_\alpha, i_{\alpha+1}) = \tilde{\gamma} \cdot w_{i_\alpha, i_{\alpha+1}}$. Consequently for all network path $\mathcal{S}(i'_1, i'_{K'}) = (i'_1, i'_2, \dots, i'_{K'-1}, i'_{K'})$ from $i'_1 = i_\alpha$ to $i'_{K'} = i_{\alpha+1}$, the cost of $\mathcal{S}(i'_1, i'_{K'})$ is written:

$$\hat{\gamma} \sum_{(u, v) \in \hat{\mathcal{S}}(i'_1, i'_{K'})} w_{u, v} + \tilde{\gamma} \sum_{(u, v) \in \tilde{\mathcal{S}}(i'_1, i'_{K'})} w_{u, v} \geq \tilde{\gamma} \cdot w_{i_\alpha, i_{\alpha+1}},$$

with $\hat{\mathcal{S}}(i'_1, i'_{K'})$ and $\tilde{\mathcal{S}}(i'_1, i'_{K'})$ the set of intra-cluster and inter-cluster links in path $\mathcal{S}(i'_1, i'_{K'})$ respectively. Multiplying by $\hat{\gamma}/\tilde{\gamma} < 1$ we get:

$$\begin{aligned} & \hat{\gamma}^2/\tilde{\gamma} \sum_{(u,v) \in \hat{\mathcal{S}}(i'_1, i'_{K'})} w_{u,v} + \hat{\gamma} \sum_{(u,v) \in \tilde{\mathcal{S}}(i'_1, i'_{K'})} w_{u,v} > \hat{\gamma} \cdot w_{i_\alpha, i_{\alpha+1}}, \\ \Leftrightarrow & \hat{\gamma} \left[\sum_{(u,v) \in \hat{\mathcal{S}}(i'_1, i'_{K'})} w_{u,v} + \sum_{(u,v) \in \tilde{\mathcal{S}}(i'_1, i'_{K'})} w_{u,v} \right] > \hat{\gamma} \cdot w_{i_\alpha, i_{\alpha+1}}, \end{aligned}$$

which proves that link $(i_\alpha, i_{\alpha+1})$ is a shortest path between nodes i_α and $i_{\alpha+1}$ in \mathcal{G} , with $h_X(p, i_\alpha, i_{\alpha+1}) = \hat{\gamma} \cdot w_{i_\alpha, i_{\alpha+1}}$. Because $\tilde{\gamma} > \hat{\gamma}$, we have $h_X(p, i_\alpha, i_{\alpha+1}) > h_X(\{\mathcal{V}\}, i_\alpha, i_{\alpha+1})$. ■

Result 1.5 $\forall p \in \mathcal{P}$ with $p \neq \{\mathcal{V}\}$, when $\gamma > 1$ there is no pair of nodes $(i, j) \in \mathcal{V}^2$ such that $h_X(p, i, j) < h_X(\{\mathcal{V}\}, i, j)$.

Proof Because $\gamma > 1$ the weight of any link in $\mathcal{G}(p)$ is greater or equal than its weight in \mathcal{G} . Consequently a shortest path between two nodes i and j in \mathcal{G} has a cost lower than or equal to the one of a shortest path between i and j in $\mathcal{G}(p)$. ■

Result 1.6 Considering J_0^X , the solutions of the problem consisting in (1.2) are:

1. If $\gamma = 1$, $\mathcal{P}' = \mathcal{P}$,
2. If $\gamma > 1$, $\mathcal{P}' = \mathcal{V}$.

Proof

1. Suppose $\gamma = 1$. Then $\tilde{\gamma} = \hat{\gamma}$. This means that for any pair of nodes $(i, j) \in \mathcal{V}^2$, $\mathcal{S}_X(p, i, j) = \mathcal{S}_X(\{\mathcal{V}\}, i, j)$, $\forall p \in \mathcal{P}$, which entails $h_X(p, i, j) = h_X(\{\mathcal{V}\}, i, j)$. Consequently $J_0^X(p)$ does not depend on p and any partition $p \in \mathcal{P}$ is a best partition.
2. Suppose $\gamma > 1$. Let us consider the partition $\{\mathcal{V}\}$. Let us suppose that there is a partition $p \neq \{\mathcal{V}\}$ of \mathcal{G} such that $J_0^X(p) \leq J_0^X(\{\mathcal{V}\})$. Thanks to Result 1.4 there is at least one pair of nodes (i, j) such that $h_X(p, i, j) > h_X(\{\mathcal{V}\}, i, j)$, meaning $J_0^X(p, i, j) > J_0^X(\{\mathcal{V}\}, i, j)$. Thanks to Result 1.5 there is no pair of nodes (i', j') such that $h_X(p, i', j') < h_X(\{\mathcal{V}\}, i', j')$. This means that $J_0^X(p, i', j') \geq J_0^X(\{\mathcal{V}\}, i', j'), \forall (i', j') \in \mathcal{V}^2$. Consequently:

$$J_0^X(p) = J_0^X(p, i, j) + \sum_{\substack{(i', j') \in \mathcal{V}^2 \\ (i', j') \neq (i, j)}} J_0^X(p, i', j') > J_0^X(\{\mathcal{V}\}).$$

Like in Section 1.4.2.2, our goal becomes to find the set \mathcal{P}_X^* of partitions solving the following problem:

$$\mathcal{P}_X^* = \arg \min_{p \in \mathcal{P}^c} J_0^X(p), \quad (1.17)$$

with \mathcal{P}_c defined in (1.9).

Here is now a result illustrating a difference between J_0^X and J_0^L , proving that Result 1.3 does not apply to J_0^X .

Result 1.7 \mathcal{P}_X^* is dependent of γ .

Proof Fig. 1.3 and Fig. 1.4 define the partitions $p_1 = \{\{1\}, \{2, 5, 6\}, \{3, 4, 7, 8\}\}$ and $p_2 = \{\{1, 6\}, \{2, 3, 5\}, \{4, 7, 8\}\}$ of an 8 node network. The inner and outer colors indicates group and cluster membership respectively. In this network: *i*) the nodes $\{1, 2, 3, 5, 6\}$ and $\{4, 7, 8\}$ belong to group 1 and 2 respectively, and *ii*) $w_{i,j} = 1, \forall (i, j) \in \mathcal{E}$. Let $n_{max} = 4, \alpha = 1.0$ and $\hat{\gamma} = 1$. When $\gamma = 2$ we obtain $p_1 \in \mathcal{P}_X^*$ and $p_2 \notin \mathcal{P}_X^*$, and when $\gamma = 6$ we obtain $p_1 \notin \mathcal{P}_X^*$ and $p_2 \in \mathcal{P}_X^*$. Because $\alpha = 1.0$, the value of J_0^X only depends on the number of intra-cluster and inter-cluster links required for intra-group communications. Within p_1 and p_2 the costs of intra-group communications are respectively equal to $c(p_1) = 7\tilde{\gamma} + 14\hat{\gamma}$ and $c(p_2) = 6\tilde{\gamma} + 18\hat{\gamma}$. Thus $c(p_1) \leq c(p_2) \Leftrightarrow \tilde{\gamma} \leq 4\hat{\gamma}$. To illustrate this result we have included Table 1.1, which details the value of J_0^X for p_1 and p_2 with $\gamma \in \{2, 4, 6\}$. When $\gamma = 4, J_0^X(p_1) = J_0^X(p_2)$.

Partition	$\gamma = 2$	$\gamma = 4$	$\gamma = 6$
p_1	2.000	2.875	3.750
p_2	2.063	2.875	3.625

Table 1.1: $J_0^X(p_1)$ and $J_0^X(p_2)$ for $\gamma \in \{2, 4, 6\}$.

Thus, in $\mathcal{G}(p)$, a partition optimal for a given γ may become suboptimal for another value of γ , which concludes the proof. ■

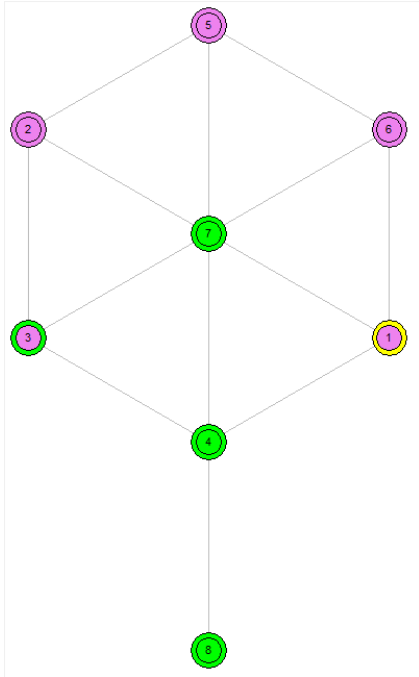


Figure 1.3: Partition p_1 .

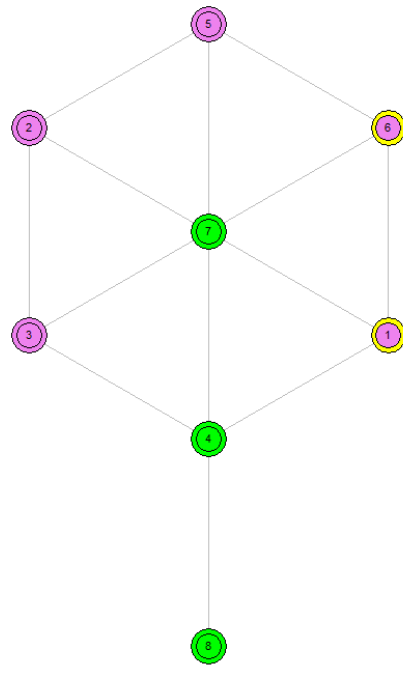


Figure 1.4: Partition p_2 .

1.4.4 Examples for practical J_0^X

In this section, we discuss how to use the generic cost function J_0^X with practical QoS metrics through two examples.

Firstly, J_0^X can be interpreted as an end-to-end delay. In that context the link weights $w_{i,j}$ can be interpreted as the average number of transmissions required on node i to achieve a successful reception on node j (using an ARQ protocol) and γ can be viewed as the multiplicative coefficient of the delay induced by the difference of efficiency between inter-cluster and intra-cluster RRA. The value of $J_0^X(p, i, j)$ from (1.15) should be interpreted as the duration needed for node i to send successfully to node j one unit of traffic. Let us take the example of a TDMA based MAC in which the intra-cluster delay to send data $\hat{\gamma}$ is equal to one 100 ms MAC frame while the inter-cluster delay $\tilde{\gamma}$ is twice this value. Assuming a uniform traffic between all nodes, each destination node j receives an equal portion $\pi_{j|i} = 1/14$ from node i . Considering the shortest path between nodes 1 and 4 of Fig. 1.2 and applying (1.6) with $w_{i,j} = 1 \forall (i, j) \in \mathcal{E}$, we get $J_0^X(p, 1, 4) = 800$ ms. The value of $J_0^X(p)$ in (??) is the average duration for a node to send successfully one unit of traffic to all nodes. Using the same example of Fig. 1.1 and Fig. 1.2 we get $J_0^X(p) = 376$ ms.

Secondly, J_0^X can be used to measure a network resource consumption, which we want to minimize in order to maximize the throughput forwarded in the network. Using the same amount of radio resources, the larger the SNR of the link, the larger the amount of information that can be transmitted. Therefore to transmit the same amount of data, the use of good links leads to the consumption of less network radio resources. From that perspective the link weights $w_{i,j}$ can be interpreted as the inverse of the spectral efficiency achievable on link (i, j) , related to the best modulation and coding scheme (MCS) usable on the link. To render $w_{i,j}$ dimensionless, it must be divided by the spectral efficiency of a reference MCS. Compared to intra-cluster RRA that can be centralized on RA nodes, inter-cluster RRA is a distributed process. Consequently the channel state indications (CSI) used by inter-cluster have less accuracy than CSI used by intra-cluster RRA. Therefore, intra-cluster RRA can use larger efficiency MCS than inter-cluster RRA, thus justifying $\gamma > 1$.

1.4.5 Application of J_0 to structured networks

1.4.5.1 Definition

We now consider that operational groups exist and that the traffic is structured according to these groups. To capture this fact, we consider that the probability that one node communicates with a node of the same group is equal to $\alpha \in [0, 1]$ and thus the probability that one node communicates with a node in another group is equal to $1 - \alpha$. Since a node of group \mathcal{O}_t can communicate to $m_t - 1$ nodes in the same group, the probability to reach one of these nodes is equal to $\alpha/(m_t - 1)$. The number of nodes of the other groups with which this node can communicate is equal to $N - m_t$ with a corresponding probability of $(1 - \alpha)/(N - m_t)$. Thus, we have:

$$\pi_{j|i} := \begin{cases} \frac{\alpha}{m_t - 1} & \text{if } j \in \mathcal{O}_t, \\ \frac{1 - \alpha}{N - m_t} & \text{otherwise,} \end{cases} \quad (1.18)$$

with $(i, j) \in \mathcal{V}^2$ and $i \in \mathcal{O}_t$.

1.4.5.2 Theoretical result

Result 1.8 *The partial ordering between J_0^X values of different partitions depends on α .*

Proof Let us define $J_0^X(p, i)$ as:

$$J_0^X(p, i) = \sum_{j \in \mathcal{V}} \pi_{j|i} \cdot J_0^X(p, i, j). \quad (1.19)$$

Summing all (1.15) with $j \in \mathcal{V}$, replacing $\pi_{j|i}$ by its value as defined in (1.18) and remembering that $i \in \mathcal{O}_t$, we get:

$$\begin{aligned} J_0^X(p, i) &= \frac{\alpha}{m_t - 1} \sum_{j \in \mathcal{O}_t} (\hat{h}_X(p, i, j) + \tilde{h}_X(p, i, j)) + \frac{1 - \alpha}{N - m_t} \sum_{j \notin \mathcal{O}_t} (\hat{h}_X(p, i, j) + \tilde{h}_X(p, i, j)), \\ &= \frac{\alpha(N - m_t)A_1(i) + (1 - \alpha)(m_t - 1)B_1(i)}{(m_t - 1)(N - m_t)}, \end{aligned} \quad (1.20)$$

with:

$$\begin{aligned} A_1(i) &:= \sum_{j \in \mathcal{O}_t} (\hat{h}_X(p, i, j) + \tilde{h}_X(p, i, j)), \\ B_1(i) &:= \sum_{j \notin \mathcal{O}_t} (\hat{h}_X(p, i, j) + \tilde{h}_X(p, i, j)). \end{aligned}$$

Equation (1.20) can be written:

$$\begin{aligned} J_0^X(p, i) &= \frac{1}{(m_t - 1)(N - m_t)} \cdot \left(\alpha \left[(N - m_t)A_1(i) - (m_t - 1)B_1(i) \right] + (m_t - 1)B_1(i) \right), \\ &= \alpha \cdot C(i) + D(i), \end{aligned}$$

with

$$\begin{aligned} C(i) &:= \frac{(N - m_t) \cdot A_1(i) - (m_t - 1) \cdot B_1(i)}{(m_t - 1)(N - m_t)}, \\ D(i) &:= \frac{(m_t - 1) \cdot B_1(i)}{(m_t - 1)(N - m_t)}. \end{aligned}$$

Summing all $J_0^X(p, i)$ with $i \in \mathcal{V}$ allows to write $J_0^X(p)$ as follows:

$$J_0^X(p) = \alpha \cdot C + D, \quad (1.21)$$

with

$$\begin{aligned} C &:= \sum_{i \in \mathcal{V}} C(i), \\ D &:= \sum_{i \in \mathcal{V}} D(i). \end{aligned}$$

In (1.21) both $C(i)$ and $D(i)$ depend on the partition, and $J_0^X(p)$ can be written:

$$J_0^X(\alpha, p) = \alpha \cdot C(p) + D(p). \quad (1.22)$$

Let p_1 and p_2 two different partitions of \mathcal{P} and let us select a specific value $\alpha = \alpha_1$ for which we have:

$$J_0^X(\alpha_1, p_1) < J_0^X(\alpha_1, p_2). \quad (1.23)$$

We now show that there exist values of $\alpha \neq \alpha_1$ which contradicts the order of (1.23) and thus prove the theorem. From (1.22) and (1.23) we have:

$$\begin{aligned} J_0^X(\alpha_1, p_1) &< J_0^X(\alpha_1, p_2), \\ \Leftrightarrow \alpha_1 \cdot C(p_1) + D(p_1) &< \alpha_1 \cdot C(p_2) + D(p_2), \\ \Leftrightarrow \alpha_1(C(p_1) - C(p_2)) &< D(p_2) - D(p_1). \end{aligned} \quad (1.24)$$

If $C(p_1) - C(p_2) > 0$ then (1.24) can be written:

$$\alpha_1 < \alpha_2 \text{ with } \alpha_2 = \frac{D(p_2) - D(p_1)}{C(p_1) - C(p_2)}. \quad (1.25)$$

Thus, we can deduce that for $\alpha_1 \geq \alpha_2$ we have $J_0^X(\alpha_2, p_1) \geq J_0^X(\alpha_2, p_2)$ which proves the assertion. If $C(p_1) - C(p_2) < 0$, the proof is the same. ■

1.4.5.3 Examples

In order to better understand J_0^X , Fig. 1.5 and Fig. 1.6 show two partitions p_1 and p_2 . In both figures nodes $\{7, 8, 9\}$ belong to the same yellow group. In Fig. 1.5 nodes $\{7, 8, 9\}$ belong to the same yellow cluster, when in Fig. 1.6 nodes $\{7, 9\}$ belong to the yellow cluster, and node 8 to the green cluster. Let us consider that the main part of the traffic is exchanged within groups. In p_1 all group members are included in the same clusters, but this is not the case in p_2 . Consequently, $J_0^X(p_1)$ should be lower (thus better) than $J_0^X(p_2)$.

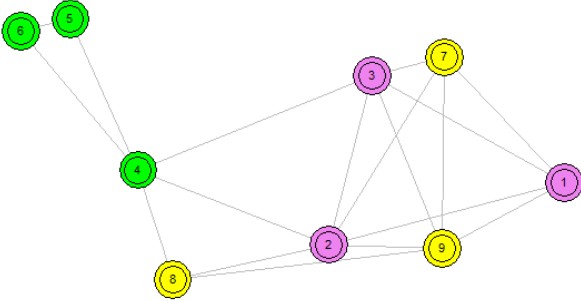


Figure 1.5: p_1 : 3 clusters partition.

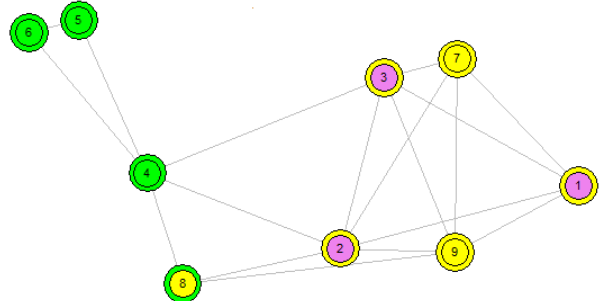


Figure 1.6: p_2 : 2 clusters partition.

However, due to Result 1.8, for different values of α two different partitions p_1 and p_2 may have their order reversed w.r.t. J_0^X . Considering the parameters $\hat{\gamma} = 1.0$, $\tilde{\gamma} = 2.0$ and $w_{i,j} = 1.0 \forall (i, j) \in \mathcal{E}$, Fig. 1.7 shows the values of $J_0^X(p_1)$ and $J_0^X(p_2)$ for $\alpha \in \{0.5, 0.6, 0.667, 0.7, 0.8, 0.9\}$. If $\alpha < 0.667$ then $J_0^X(p_1) > J_0^X(p_2)$, and the partial ordering is inverted if $\alpha > 0.667$.

In this example when the amount of inter-cluster traffic increases (i.e., α decreases), then the benefit of building only two clusters exceeds the benefit of gathering all members of the yellow group in the same cluster.

1.4.5.4 Limitation of J_0^L w.r.t. J_0^X

One may wonder about the difference between J_0^L and J_0^X w.r.t. the partition quality assessment. Let us take the example of the network of Fig. 1.8 whose partition is denoted by p , with $w_{i,j} = 1, \forall (i, j) \in \mathcal{E}$. Let us consider the shortest paths $S_L^1(p, 2, 28) = (2, 14, 20, 27, 25, 10, 28)$ and

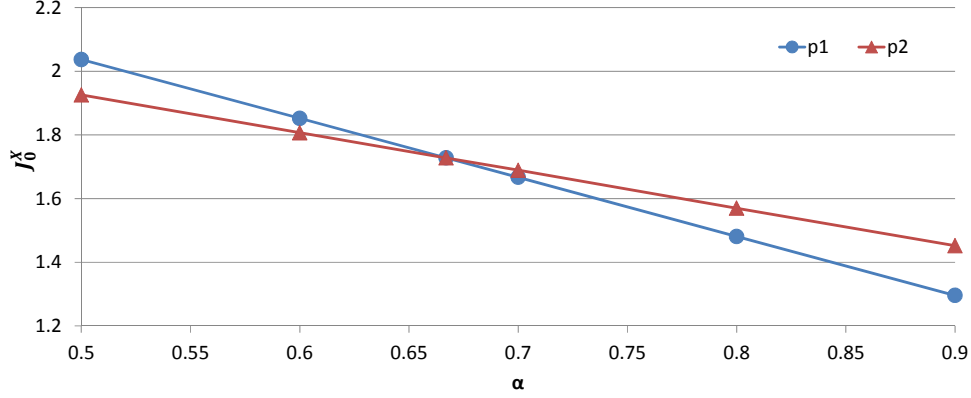


Figure 1.7: $J_0^X(p)$ versus α for 2 different partitions p_1 and p_2 .

$S_L^2(p, 2, 28) = (2, 8, 20, 27, 25, 10, 28)$ between nodes 2 and 28. The number of inter-cluster links in $S_L^1(p, 2, 28)$ and $S_L^2(p, 2, 28)$ are four - (2, 14), (14, 20), (20, 27) and (27, 25) - and two - (20, 27) and (27, 25) - respectively. Consequently, the value of $J_0^L(p, 2, 28)$ depends on whether the selected shortest path is $S_L^1(p, 2, 28)$ or $S_L^2(p, 2, 28)$. Conversely, when $J_0^X(p, 2, 28)$ is calculated, the selected shortest path is always $S_X(p, 2, 28) = S_L^2(p, 2, 28)$.

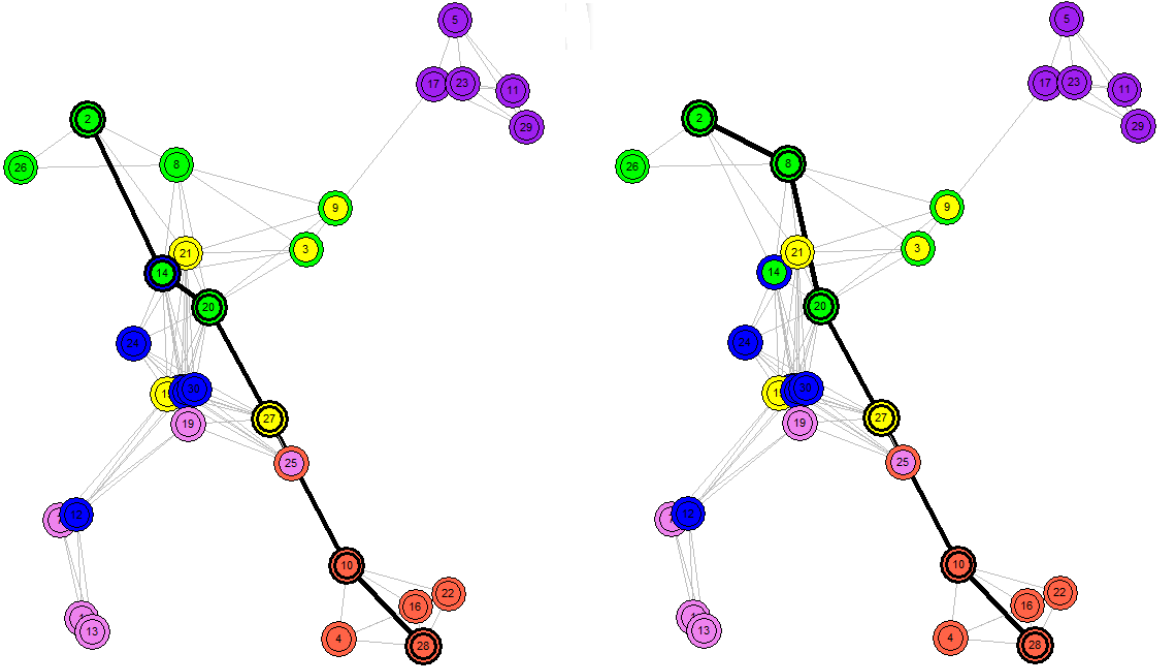


Figure 1.8: Examples of shortest paths between nodes 2 and 28.

This example shows that two different calculations of J_0^L may lead to different values because *i)* multiple shortest paths may exist between any pair of nodes, and *ii)* if more than one such path exists, because J_0^L does not take into account the cluster structure, the shortest paths may include different numbers of inter-cluster links. Thus, to get reliable results, the cluster structure must be used when calculating the cost associated with a network partition: J_0^X must be used instead of J_0^L .

To assess the error performed when using J_0^L instead of J_0^X we have used a k-shortest-path algorithm from Eppstein [37] to calculate all shortest paths between all pairs of nodes $(i, j) \in \mathcal{V}$. Then for each pair of nodes (i, j) we have calculated all $J_0^L(p, i, j)$ and $J_0^X(p, i, j)$ values. Fig. 1.9 plots J_0^X and the average, lowest and highest possible values of J_0^L for the partition of Fig. 1.8, for $\gamma \in \{2, 5\}$ and $\alpha \in \{0.5, 0.6, 0.7, 0.8, 0.9\}$.

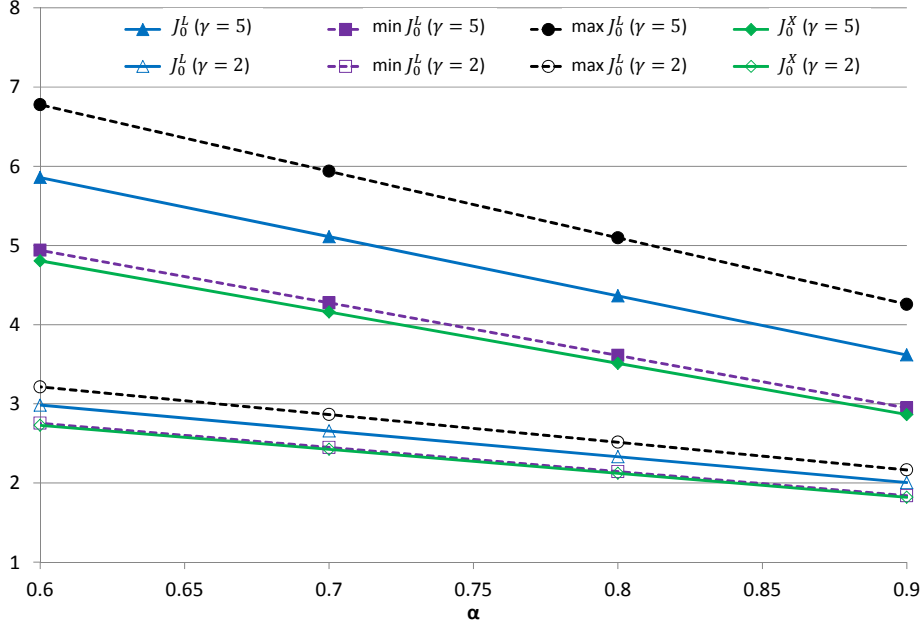


Figure 1.9: $J_0^X(p)$ and $J_0^L(p)$ versus α .

The first conclusion about this figure is the difference between the lowest and highest values of J_0^L , which increases from about 15% to about 30% when the value of γ is modified from 2 to 5. Another interesting result is about J_0^X which is lower than the lowest value of J_0^L : about 1% lower if $\gamma = 2$ and about 3% lower if $\gamma = 5$. To understand this surprising result, one must note that J_0^X may use paths which have a larger number of hops than J_0^L , and still have a lower cost.

1.4.6 Generalization to disconnected networks

In this section the calculation of $J_0(p)$ must be modified to take into account two aspects:

- Network nodes may be split into several connected components and,
- Some members from the same groups can be present in different connected components.

Let us call $\mathcal{J}_0(p)$ the adaptation of $J_0(p)$ to disconnected networks. Let us suppose that the number of connected components in the network is Q , and let us denote by \mathcal{Q}_ν the ν th connected component, with $\nu \in \{1, \dots, Q\}$. Let us denote by p_ν the partition of \mathcal{Q}_ν , with $p := \cup_{\nu \in \{1, \dots, Q\}} p_\nu$. Let us remark that no clustering scheme can enable a communication between two different connected components \mathcal{Q}_{ν_1} and \mathcal{Q}_{ν_2} . It is thus legitimate to ignore the absence of such communication and to separate the calculation of $\mathcal{J}_0(p)$ into a weighted sum of values $\mathcal{J}_0^\nu(p_\nu)$, each associated with a connected component:

$$\mathcal{J}_0^\nu(p_\nu) := \frac{1}{|\mathcal{Q}_\nu|} \sum_{(i,j) \in \mathcal{Q}_\nu^2} \pi_{\nu,j|i} \cdot \mathcal{J}_0(p_\nu, i, j), \quad (1.26)$$

with:

$$\mathcal{J}_0(p_\nu, i, j) := \begin{cases} J_0^L(p_\nu, i, j) & \text{defined by (1.6) to calculate } \mathcal{J}_0^L(p_\nu), \\ J_0^X(p_\nu, i, j) & \text{defined by (1.15) to calculate } \mathcal{J}_0^X(p_\nu), \end{cases}$$

and

$$\pi_{\nu, j|i} := \begin{cases} \frac{\alpha}{|\mathcal{O}_{\nu, k}| - 1} & \text{if } j \in \mathcal{O}_{\nu, k}, \\ \frac{1 - \alpha}{|\mathcal{Q}_\nu| - |\mathcal{O}_{\nu, k}|} & \text{otherwise,} \end{cases}$$

with $(i, j) \in \mathcal{Q}_\nu^2$, $\mathcal{O}_{\nu, k} := \mathcal{O}_t \cap \mathcal{Q}_\nu$, and $i \in \mathcal{O}_{\nu, k}$.

Then we define $\mathcal{J}_0(p)$ as the average of $\mathcal{J}_0^\nu(p)$ over the connected component size:

$$\mathcal{J}_0(p) := \frac{1}{N} \sum_{\nu=1}^Q |\mathcal{Q}_\nu| \cdot \mathcal{J}_0^\nu(p_\nu). \quad (1.27)$$

When the network is composed of several connected components, (1.27) must be used instead of (1.7) or (1.16) to avoid underestimating the cost of the network.

1.5 Numerical analysis

1.5.1 Assessment methodology

A first result we want to show is that the groups have an impact on the quality of the clustering solution. To do this, we find the set \mathcal{P}^u of optimal partitions when the traffic pattern does not depend on the groups, then we determine if these partitions are still good when the traffic pattern becomes dependent on the groups. A traffic independent of the groups is equivalent to $\pi_{j|i} = \frac{1}{N-1}$, $\forall (i, j) \in \mathcal{V}^2$, $i \neq j$, hence:

$$\mathcal{P}^u := \arg \min_{p \in \mathcal{P}^c} J_0^X(p) \Big|_{\pi_{j|i} = \frac{1}{N-1}, \forall (i, j) \in \mathcal{V}^2, i \neq j}.$$

To determine if the partitions $p \in \mathcal{P}^u$ are still good when the traffic pattern becomes dependent on the groups according to (1.18), we calculate the following metric:

$$\delta_u(\alpha) := \frac{J_0^u - J_0^*}{\bar{J}_0^* - J_0^*}, \quad (1.28)$$

with J_0^u the highest value of J_0^X for all the partitions in \mathcal{P}^u when J_0^X is calculated with $\pi_{j|i}$ defined as in (1.18), $\bar{J}_0^* := \max_{p \in \mathcal{P}^c} J_0^X(p)$, and $J_0^* := \min_{p \in \mathcal{P}^c} J_0^X(p)$. Let us call J_0^X *interval* the interval $[J_0^*, \bar{J}_0^*]$. The term $\delta_u(\alpha)$ measures the performance loss obtained by not taking into account the dependence of traffic patterns to groups during cluster building. It takes its values in $[0, 1]$ when J_0^u goes through the J_0^X interval: 0 is associated with the best partitions, and 1 with the worst partitions.

Assuming that taking the group structure into account when building clusters is of interest, new algorithms have to be designed. In order to show that it is not an easy task (which is the subject of the subsequent chapters), we propose hereafter a naive approach called one group-one cluster and denoted by 1G1C. The simplest way to follow would be to *force* all the members of a group to belong to the same cluster. However, due to the constraints it is not always possible. Therefore we propose the following procedure: *i*) for each group, find the node with the highest

degree in the subgraph of \mathcal{G} induced by the members of this group and build a cluster including this node and its 1-hop neighbors; *ii*) for each nodes not yet member of any cluster, attach it to an existing cluster while making sure that C_3 is satisfied, otherwise build a new singleton cluster when C_3 is not satisfied. The obtained partition is denoted by p_g . Using the same idea as (1.28), the performance of the aforementioned naive algorithm is studied through the metric:

$$\delta_g(\alpha) := \frac{J_0^X(p_g) - J_0^*}{\bar{J}_0^* - J_0^*}.$$

Note that this procedure may lead to clusters with a number of nodes less than n_{min} , thus not satisfying C_2 .

1.5.2 Simulation setup

1.5.2.1 Number of partitions of a graph

Before introducing our network model, let us introduce some complexity aspects when enumerating all partitions in \mathcal{P}^c . Firstly, the number of possible partitions in N_c clusters of a graph \mathcal{G} with N vertices is equal to the Stirling number of the second kind [25] $S(N, N_c) := \frac{1}{N_c!} \sum_{k=0}^{N_c} (-1)^{N_c-k} \binom{N_c}{k} k^N$. The total number of possible partitions is the N th Bell number $B_N := \sum_{k=0}^N S(N, k)$. This number increases very quickly with the graph size N . The first ten Bell numbers for $N = 1$ to 10 are 6, 116, 1 928, 27 665, 364 472, 4 547 586, 54 670 463, 639 838 113, 7 338 610 159, 82 857 366 967, which means that an numeration of all partitions \mathcal{P} of a graph is impossible unless the graph consists of very few vertices.

Fortunately only the partitions in \mathcal{P}^c are acceptable, which significantly reduces their number. However, very quickly $|\mathcal{P}^c|$ becomes very large, thus to cope with the limited amount of processing power and time, we have only considered networks with a limited number of nodes N and edges M .

1.5.2.2 Node deployment model

The networks considered in this chapter are structured networks, whose nodes are organized in groups. They are generated using the following procedure. Firstly, T nodes are placed randomly in a $d \times d$ square area following a uniform distribution. Each of those nodes is the first node of each group $\mathcal{O}_t, \forall t \in \{1, 2, \dots, T\}$, called V_k . Then all members of each group \mathcal{O}_t are placed within the disk centered on V_k , using polar coordinates (ρ, θ) . The radius ρ is a random variable following the probability density function (pdf):

$$f_\rho(d) = \begin{cases} \frac{2}{d_1+d_2} & \text{if } 0 < d \leq d_1, \\ \frac{2(d_2-d)}{(d_1+d_2)(d_2-d_1)} & \text{if } d_1 < d \leq d_2, \\ 0 & \text{otherwise.} \end{cases} \quad (1.29)$$

The angle θ is a random variable following a uniform pdf in $[-\pi, +\pi)$. When θ leads to a node outside of the simulated area, another value is calculated.

Fig. 1.10 and 1.11 provides an example of the pdf (1.29) and cumulative distribution function (cdf) of ρ when $d_1 = 4000$ and $d_2 = 8000$.

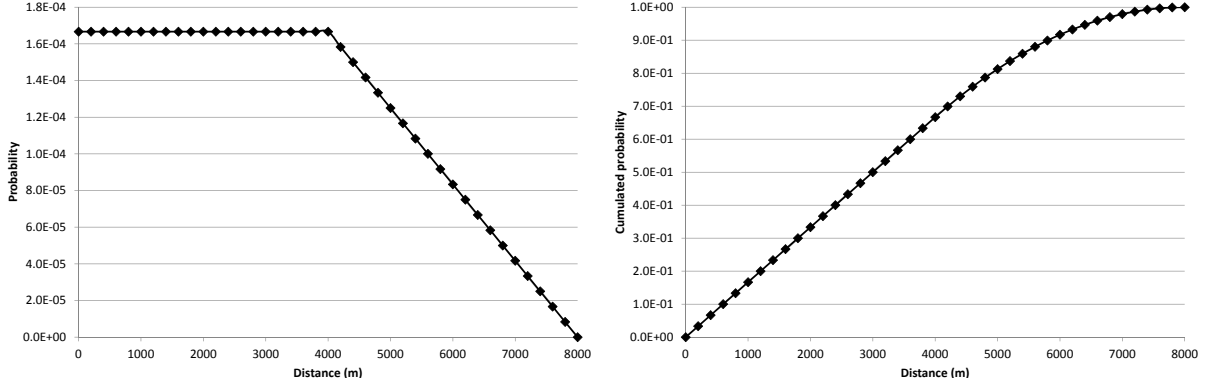


Figure 1.10: pdf of group member distance from the first group member. Figure 1.11: cdf of group member distance from the first group member.

1.5.2.3 Network examples and simulation parameters

Random networks were generated using parameters $N = 30$, $d_1 = 4000$, $d_2 = 8000$, $T = 6$, $r = 5000$ and $d = 20000$. Fig. 1.12 to Fig. 1.15 display some of these networks, ordered by increasing number of edges. In these figures the color indicates group membership. In Fig. 1.12, the yellow group is composed of nodes 3, 9, 15, 21, 27. This group is not a connected component of the whole network because node 21 is not a neighbor of any other member of the group. The blue group is composed of nodes 6, 12, 18, 24, 30. It is a 2-hops diameter connected sub-graph of the whole network. The violet group is composed of nodes 1, 7, 13, 19, 25 is a 3-hops diameter connected sub-graph of the whole network. In Fig. 1.13, all sub-graphs induced by operational groups are connected.

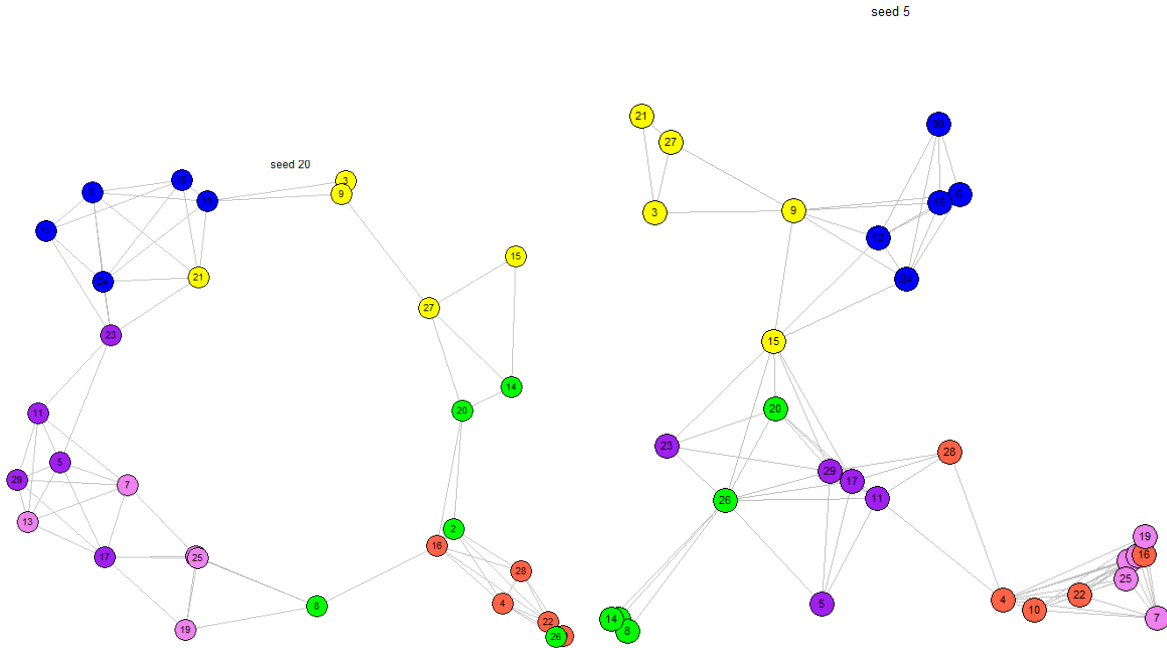


Figure 1.12: 30 nodes network with 146 edges. Figure 1.13: 30 nodes network with 182 edges.

Following the discussion in Section 1.5.2.1, and using the network model described in Section 1.5.2.2, we implemented an algorithm to enumerate all partitions of a graph subject to

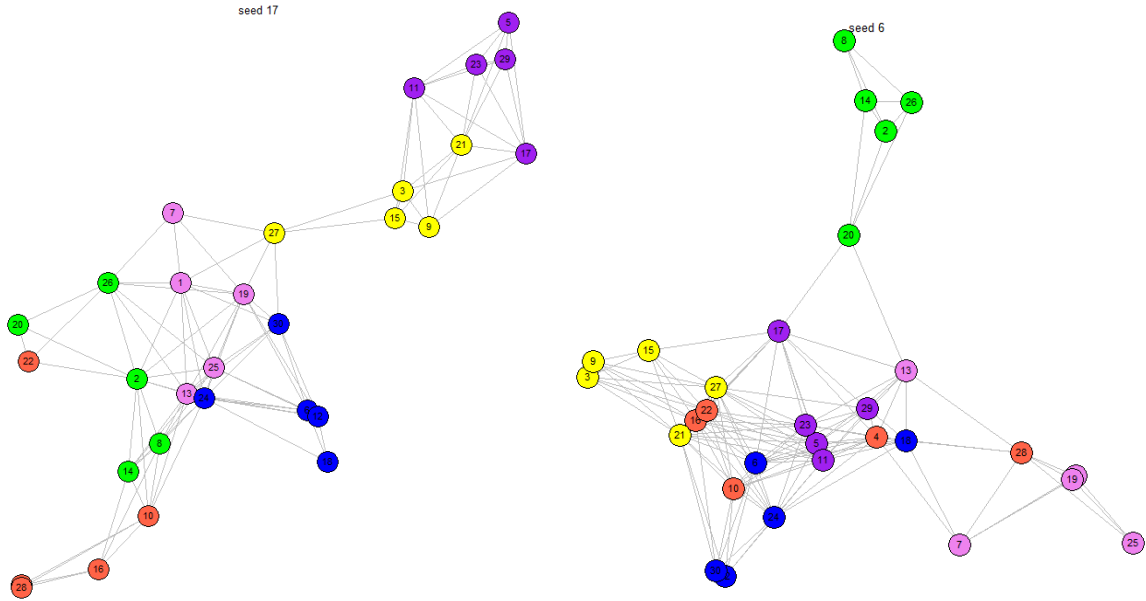


Figure 1.14: 30 nodes network with 202 edges. Figure 1.15: 30 nodes network with 268 edges.

connectivity, size and diameter constraints. It is described in appendix A. This algorithm was used with the four networks of Fig. 1.12 to Fig. 1.15, with $n_{min} = 4$, $n_{max} = 8$ and $d_{max} = 2$. Table 1.2 displays the number of partitions satisfying the constraints, and the number of seconds required to calculate them (using optimized C/C++ software running on a 3.5 GHz core i5 processor). More than 9 days were required to solve the network of Fig. 1.15.

Fig.	Edges	Partitions	Time (s)
1.12	146	8 584	102
1.13	182	56 228	2 079
1.14	202	10 519 184	28 951
1.15	268	979 970 630	795 113

Table 1.2: Complexity for various networks.

Thus, to limit the amount of time required to enumerate partitions in \mathcal{P}^c , in this section the number of nodes has been set to $N = 30$ and the number of edges has been limited to 230. In addition the following parameter values have been used: $d_1 = 4000$, $d_2 = 8000$, $T = 6$, $r = 5000$ and $d = 20000$. Using these parameters, the number of edges of the 100 random networks used during our simulations lies in $[146, 222]$ with an average value of 183 edges. Their network diameter lies in $[5, 12]$ and is equal to 7.19 on average. Among the 100 random networks, the minimum and maximum number of valid partitions are 672 and 49 647 650.

For the sake of simplicity, each $w_{i,j}$ has been set constant and equal to 1, $\forall(i, j) \in \mathcal{E}$, and γ has been set to 2. Finally the constraint parameters are: $n_{min} = 4$, $n_{max} = 8$ and $d_{max} = 2$. All the curves of Section 1.5.3 have been obtained by averaging over 100 networks.

1.5.3 Results

The cdf of $\delta_u(\alpha)$ are plotted in Fig. 1.16.

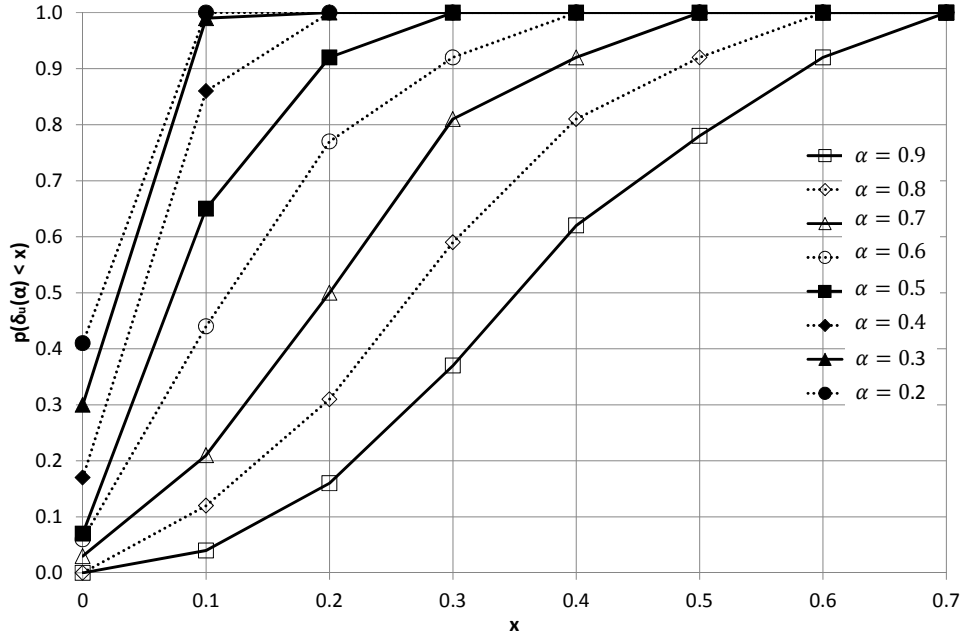


Figure 1.16: cdf of $\delta_u(\alpha)$.

The first information provided by these curves is that the partitions $p \in \mathcal{P}^u$ that are the best when the traffic pattern does not depend on the groups, do not remain the best partitions when the traffic pattern depends on the group: $\mathcal{P}^u \neq \mathcal{P}_X^*$. For example when $\alpha = 0.2$, only 40% of the partitions $p \in \mathcal{P}^u$ are also in \mathcal{P}_X^* and this proportion decreases to about 0% when $\alpha = 0.8$ or 0.9. An additional result is that the partitions $p \in \mathcal{P}^u$ are not only no longer the best, but also become quite bad when the value of α is high enough. For example when $\alpha = 0.7$, only 50% of these partitions have a J_0^X value in the first 20% of the J_0^X interval. Fig. 1.16 shows that when traffic flows are concentrated within groups, the group membership should be taken into account during cluster building. This result has been achieved thanks to the introduction of the benchmark network cost function J_0^X .

Fig. 1.17 illustrates the benefit of operational clustering, using the same example of the TDMA based MAC as in Section 1.4.4. This figure plots, for one particular network, the histogram of the average delay between nodes of the same group, calculated over all groups. Thanks to J_0^X , for each value of α the best partitions $p \in \mathcal{P}_X^*$ have been found and the associated delays have been determined. Depending on α the best partitions are different thus the intra-group delays are different. Fig. 1.17 shows that when α increases, the average delay decreases. Additionally, the larger α is, the larger is the number of pairs of nodes in the same group with a minimum delay of 100 ms. This is a practical example of the benefit provided to the end-user when the group structure is taken into account.

Another result achieved thanks to J_0^X is that the naive clustering strategy consisting in trying to build one cluster per group is not the best way to build clusters. Out of the 100 networks clustered using this heuristic, only 56 satisfied the cluster size constraints, the remaining ones included clusters with only 1 or 2 nodes. Fig. 1.18 plots the cdf of $\delta_g(\alpha)$ for these 56 networks. This figure shows that the need for a clustering solution more clever than the naive one increases when α decreases. For example when $\alpha = 0.9$, about 80% of these partitions have a J_0^X value

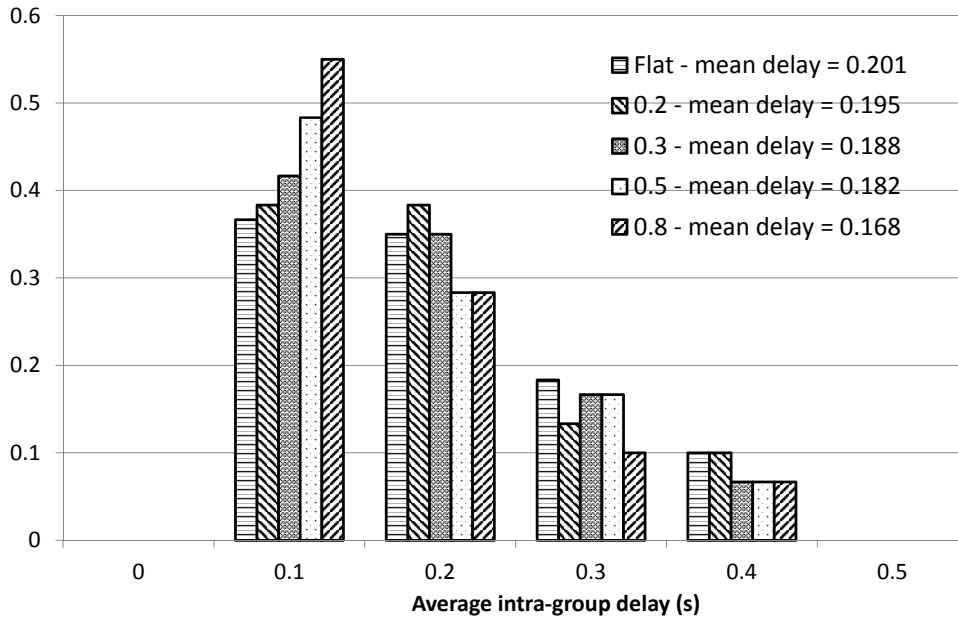


Figure 1.17: End-to-end intra-group delay histogram for several values of α .

in the first 10% of the J_0^X interval. These 80% are reduced to about 55% when $\alpha = 0.7$, and are close to 0% when $\alpha < 0.5$.

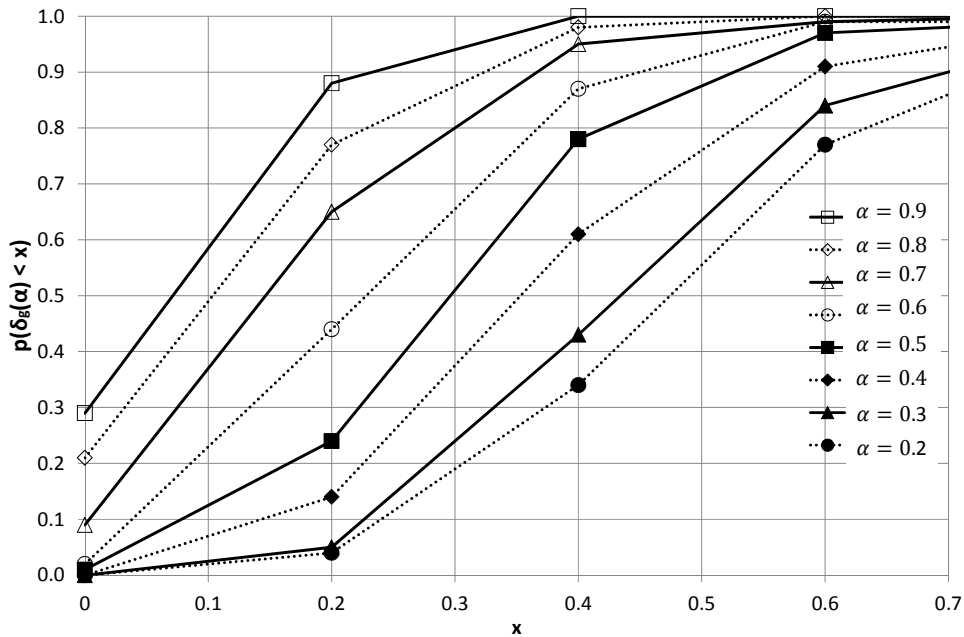


Figure 1.18: cdf of $\delta_g(\alpha)$.

1.6 Extending J_0^X to take cluster size into account

A limit of J_0^X is related to the fact that its minimal (i.e. best) value is when the whole network is partitioned into a single cluster. This is a problem because a key hypothesis when defining J_0^X

is $\tilde{\gamma} \geq \hat{\gamma}$, which is valid only if the cluster size is not too large. To handle this limit, we derive from J_0^X a new network cost function \bar{J}_0^X which takes into account the fact that the efficiency of RRA within a cluster decreases with the number of its members.

Note: in this section we extend J_0^X only, knowing that extending J_0^L could be done similarly without any difficulty.

1.6.1 Definition

Let us define graph $\bar{\mathcal{G}}(p)$ as the set of nodes \mathcal{V} connected by edges in \mathcal{E} whose costs are $\hat{\gamma} \cdot w_{i,j} \cdot \xi(i,j)$ if (i,j) is an intra-cluster link, with $\xi(i,j) \geq 1$ a factor used to take into account cluster size, and $\tilde{\gamma} \cdot w_{i,j}$ if (i,j) is an inter-cluster link. The shortest path between nodes i and j in $\bar{\mathcal{G}}(p)$ is denoted by $\bar{\mathcal{S}}_X(p,i,j)$, with cost $\bar{h}_X(p,i,j)$. This shortest path can be split into its intra-cluster part $\hat{\mathcal{S}}_X(p,i,j)$ and its inter-cluster part $\tilde{\mathcal{S}}_X(p,i,j)$ with respective costs $\hat{h}_X(p,i,j)$ and $\tilde{h}_X(p,i,j)$:

$$\hat{h}_X(p,i,j) := \sum_{(i',j') \in \hat{\mathcal{S}}_X(p,i,j)} \hat{\gamma} \cdot w_{i',j'} \cdot \xi(i',j') \quad (1.30)$$

$$\tilde{h}_X(p,i,j) := \sum_{(i',j') \in \tilde{\mathcal{S}}_X(p,i,j)} \tilde{\gamma} \cdot w_{i',j'}. \quad (1.31)$$

We define $\bar{J}_0^X(p,i,j) := \bar{h}_X(p,i,j)$, and define $\bar{J}_0^X(p)$ like in (1.1):

$$\bar{J}_0^X(p) := \frac{1}{N} \sum_{(i,j) \in \mathcal{V}^2} \pi_{j|i} \cdot \bar{J}_0^X(p,i,j). \quad (1.32)$$

The difference between $\bar{J}_0^X(p,i,j)$ and $J_0^X(p,i,j)$ lies in the factor $\xi(i,j)$. When $\hat{\gamma} \cdot w_{i,j} \cdot \xi(i,j) > \tilde{\gamma} \cdot w_{i,j}$, the cost of gathering nodes i and j in the same cluster is larger than the cost of putting them in two different clusters. The function $\xi(i,j)$ models the decrease of RRA efficiency when the cluster size increases.

To determine the shape of $\xi(i,j)$, the following ideas may be considered:

- Given a slotted MAC, the number of slots per unit of time is fixed,
- The amount of slots needed per cluster member is limited. When a cluster member gets at least this number then it is considered as satisfied. Otherwise its dissatisfaction increases with the number of missing slots,

Let us consider two nodes i and j in the same cluster \mathcal{C}_k . As an example, the function $\xi(i,j)$ can be defined as follows:

$$\xi(i,j) := \begin{cases} 1, & \text{if } n_k \leq n_{thr} \\ \frac{1}{\lambda} \left[\frac{n_k}{n_{thr}} + \lambda - 1 \right] & \text{otherwise.} \end{cases} \quad (1.33)$$

with $\lambda > 0$ and n_{thr} a target cluster size. The function $\xi(i,j)$ increases linearly with the cluster size. Its value is equal to 2 when $n_k = (\lambda + 1) \cdot n_{thr}$.

1.6.2 Example

Let us consider a MAC frame including a signaling part and a data communications part, whose duration is d_{frame} . The number of time slots n_{data} in the data communication part of the MAC

frame is fixed. Let us denote by q_{min} the number of slots per second needed by a cluster member to satisfy the user QoS. Let us consider a simple channel unaware equal radio resource allocation scheme in a cluster \mathcal{C}_k of size n_k . To ensure that the number of slots per second per cluster member is greater or equal than q_{min} we have:

$$\frac{n_{data}}{d_{frame} \cdot n_k} \geq q_{min} \Leftrightarrow n_k \leq \frac{n_{data}}{d_{frame} \cdot q_{min}}.$$

Let us suppose that beyond this cluster size, the QoS degrades rapidly because of the throughput decrease. We now introduce the percentage of throughput degradation δ that can be tolerated by the application. The highest cluster size n_{hig} can be calculated as follows:

$$q_{min} \cdot (1 - \delta/100) = \frac{n_{data}}{d_{frame} \cdot n_{hig}} \Leftrightarrow n_{hig} = \frac{n_{data}}{d_{frame} \cdot q_{min} \cdot (1 - \delta/100)} \quad (1.34)$$

To calculate the λ parameter of (1.33), we decide that when the cluster is n_{hig} , then the intra-cluster communication cost is doubled, i.e. $\xi(i, j) = 2$. We thus write:

$$\frac{1}{\lambda} \left[\frac{n_{hig}}{n_{thr}} + \lambda - 1 \right] = 2 \Leftrightarrow \lambda = \frac{n_{hig}}{n_{thr}} - 1. \quad (1.35)$$

For example let us consider a MAC frame with $d_{frame} = 100$ ms $n_{data} = 20$, and the case whereby $q_{min} = 10$, $n_{thr} = 20$ and $\delta = 20\%$. We use (1.34) to write:

$$n_{hig} = \frac{n_{data}}{d_{frame} \cdot q_{min} \cdot (1 - \delta/100)} = \frac{20}{0.1 \cdot 10 \cdot (1 - 20/100)} = 25.$$

Using (1.35) we have $\lambda = 25/20 - 1 = 0.25$.

1.6.3 Numerical illustration

To assess numerically the difference between J_0^X and \bar{J}_0^X let us consider the network of Fig. 1.19. This network has 20 nodes with $m_t = 4$, and $T = 5$. The number of partitions of this network is very large and cannot be easily calculated. To decrease this number, we have considered only partitions including connected clusters whose size is greater or equal than two.

This number of remaining partitions is still quite large (equal to 2 477 973 765), a few days were required to calculate the values of J_0^X and \bar{J}_0^X for all these partitions. Fig. 1.20 plots the minimum values of J_0^X and \bar{J}_0^X (multiplied by the network size N) versus the maximum cluster size. The parameters common to J_0^X and \bar{J}_0^X are $\gamma = 2.0$ and $\alpha = 0.9$. To calculate \bar{J}_0^X , the value of n_{thr} has been set to five and the one of λ to 0.4. When the maximum cluster size is no greater than five, J_0^X and \bar{J}_0^X are equal. Beyond this value, $\bar{J}_0^X > J_0^X$.

Fig. 1.21 plots only J_0^X , whose minimum occurs when the whole network is a single cluster. The "wave" shape of the curve when $n_{max} \geq 8$ is explained by the group size value $m_t = 4$: when the maximum cluster size is a multiple of four, then full groups can be included in the same cluster and the number of inter-cluster links used for intra-group communication is reduced to zero, thus minimizing J_0^X .

1.7 Case of concave metrics: the throughput example

In this section we follow the methodology used with J_0 to define T_0 usable with the throughput metric. We extend J_0^X only, knowing that extending J_0^L could be done similarly.

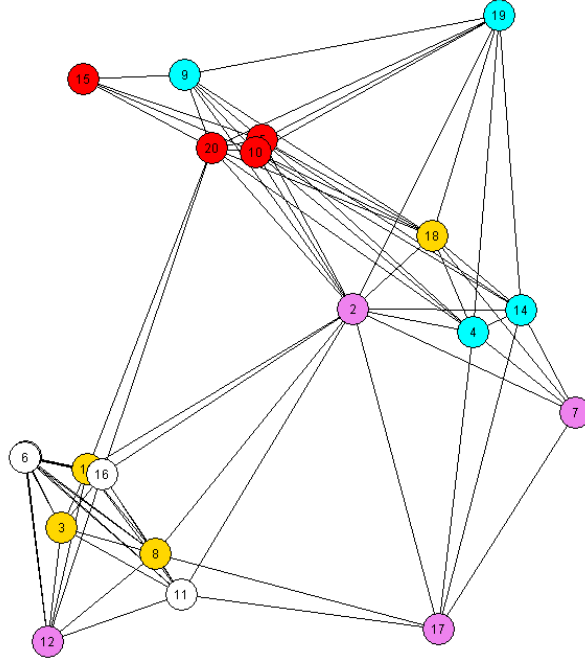


Figure 1.19: Network used to compare J_0^X and \bar{J}_0^X .

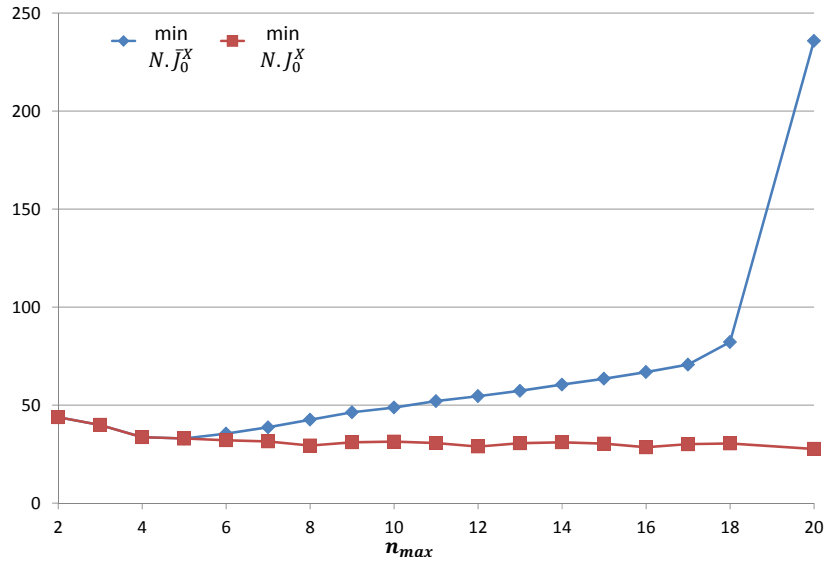


Figure 1.20: Minimum values of J_0^X and \bar{J}_0^X vs. maximum cluster size n_{max} .

1.7.1 Definition

Let us start from (1.15):

$$J_0^X(p, i, j) = \sum_{(i', j') \in \tilde{S}_X(p, i, j)} \hat{\gamma} \cdot w_{i', j'} + \sum_{(i', j') \in \tilde{S}_X(p, i, j)} \tilde{\gamma} \cdot w_{i', j'}.$$

To use throughput instead of additive metric, $J_0^X(p, i, j)$ must be modified to take into account that throughput is a concave metric. This means that the throughput metric associated with a network path is the minimum throughput value along the links of this path, instead of being

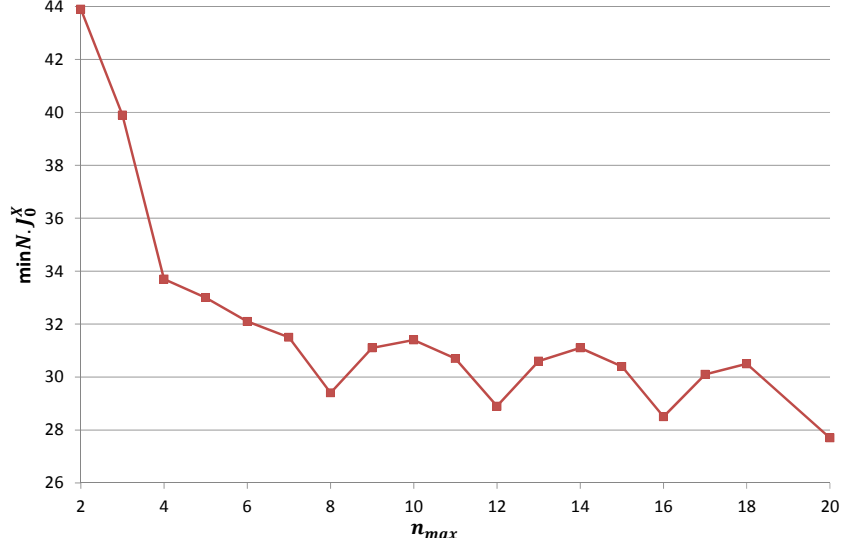


Figure 1.21: Minimum values of J_0^X vs. maximum cluster size, with group size equal to four.

the cumulated value. The graph $\mathcal{G}'(p)$ is defined by its vertices \mathcal{V} and its set of edges $\mathcal{E}_T(p)$ whose weights are $\hat{\eta} \cdot w_{i,j}$ and $\tilde{\eta} \cdot w_{i,j}$ for intra and inter-cluster links respectively, with $\hat{\eta}$ and $\tilde{\eta} < \hat{\eta}$ the throughput efficiencies achieved by intra and inter-cluster RRA, and $\eta = \tilde{\eta}/\hat{\eta} (< 1)$. Let define as $\mathcal{W}_X(p, i, j)$ the shortest widest path [38] from node i to node j in $\mathcal{G}'(p)$. Let us split this path $\mathcal{W}_X(p, i, j)$ into two subsets $\hat{\mathcal{W}}_X(p, i, j)$ and $\tilde{\mathcal{W}}_X(p, i, j)$, the sets of its intra and inter-cluster links respectively. The throughput $T_0^X(p, i, j)$ that can be achieved between node i and node j is defined as:

$$T_0^X(p, i, j) := \min \left(\hat{\eta} \cdot \min_{(i',j') \in \hat{\mathcal{W}}_X(p,i,j)} w_{i',j'}, \tilde{\eta} \cdot \min_{(i',j') \in \tilde{\mathcal{W}}_X(p,i,j)} w_{i',j'} \right) \quad (1.36)$$

$$:= \hat{\eta} \left[\min \left(\min_{(i',j') \in \hat{\mathcal{W}}_X(p,i,j)} w_{i',j'}, \eta \cdot \min_{(i',j') \in \tilde{\mathcal{W}}_X(p,i,j)} w_{i',j'} \right) \right] \quad (1.37)$$

Similarly to (1.1), we now introduce the metric $T_0^X(p)$:

$$T_0^X(p) := \frac{1}{N} \sum_{(i,j) \in \mathcal{V}^2} \pi_{j|i} \cdot T_0^X(p, i, j). \quad (1.38)$$

Finding the best clustering of the network from T_0^X perspective is equivalent to finding the set of partitions \mathcal{P}_T^X defined as:

$$\mathcal{P}_T^X := \arg \max_{p \in \mathcal{P}} T_0^X(p) \quad (1.39)$$

When $\eta < 1$ one may wonder about the quality of the single cluster network partition $\{\mathcal{V}\}$. Using a similar argument as for Result 1.6 page 23, it is possible to demonstrate that $\{\mathcal{V}\}$ is the partition leading to the highest T_0^X value.

1.7.2 Shortest widest path

Let us recall the difference between the shortest path and the shortest widest path algorithms. In a graph \mathcal{G} , for each link $(i, j) \in \mathcal{E}$, let $b_{i,j}$ and $d_{i,j}$ the available bandwidth and the propagation

delay of this link, respectively. If $(i, j) \notin \mathcal{E}$ then $b_{i,j} = 0$ and $d_{i,j} = \infty$. The length of the shortest-path $\mathcal{S}(i, j) = (i, i_2, i_3, \dots, i_{K-1}, j)$ from node i to node j is denoted by $D_{i,j}$ and is defined as the sum of propagation delays along this path. The shortest path algorithm from Dijkstra calculates all shortest paths from node i to all nodes $j \neq i$. It is defined in Table 1.3.

1	Set $L = \{i\}$, $D_{i,i} = 0$, and $D_{i,j} = d_{i,j}, \forall j \neq i$.
2	While True:
3	Find set $K \subset \mathcal{V} \setminus L$ such that $\forall k \in K, D_{i,k} = \min_{j \notin L} D_{i,j}$.
4	If $K \neq \emptyset$ then :
5	Choose $k \in K$ randomly.
6	Set $L = L \cup \{k\}$.
7	For all j such that $d_{k,j} < +\infty$:
8	Set $D_{i,j} = \min(D_{i,j}, D_{i,k} + d_{k,j})$.
9	End For .
10	Else Exit While loop.
11	End If .
12	End While .

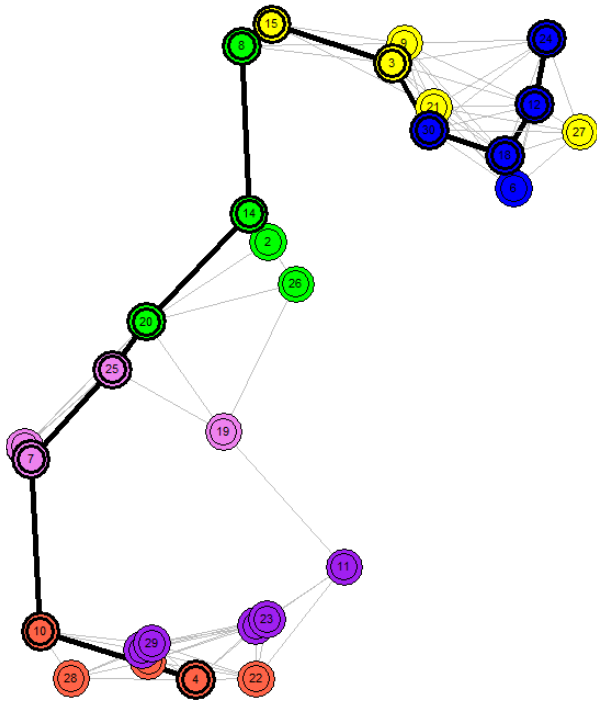
Table 1.3: Dijkstra shortest path algorithm.

$B_{i,j}$ and $D_{i,j}$ are the width and the length of the chosen shortest-widest path from node i to node j . By convention, $B_{i,j} = \infty$ and $D_{i,i} = 0$. The algorithm in Table 1.4 [38] allows to find a widest path from node i to node j , and the associated bandwidth $B_{i,j}^*$. The widest path found by this algorithm is not the shortest widest path. To find the shortest widest path from i to j the Dijkstra algorithm must be used using the set of links $\mathcal{E}' = \mathcal{E} \setminus \{(u, v) \in \mathcal{E} | b_{u,v} < B_{i,j}^*\}$. Without this second step the widest path found is not a shortest one and depends on the source, as illustrated in Fig. 1.22, Fig. 1.23 and Fig. 1.24.

1	Set $L = \{i\}$.
2	Set $B_{i,i} = \infty$, $D_{i,i} = 0$, $B_{i,j} = b_{i,j}$ and $D_{i,j} = d_{i,j}, \forall j \neq i$.
3	While True:
4	Find set $K \subset \mathcal{V} \setminus L$ such that $\forall k \in K, B_{i,k} = \max_{j \notin L} B_{i,j}$.
5	If $K \neq \emptyset$ then :
6	Find k such that $D_{i,k} = \min_{j \in K} D_{i,j}$.
7	Set $L = L \cup \{k\}$.
7	For all $j \notin L$:
9	Set $B_{i,j} = \max(B_{i,j}, \min(B_{i,k}, b_{k,j}))$.
10	End For .
11	Else Exit While loop.
12	End If .
13	End While .

Table 1.4: Modified Dijkstra widest path algorithm.

WP 4->10->7->25->20->14->8->15->3->30->18->12->24



WP 24->3->8->14->20->25->7->10->28->17->16->4

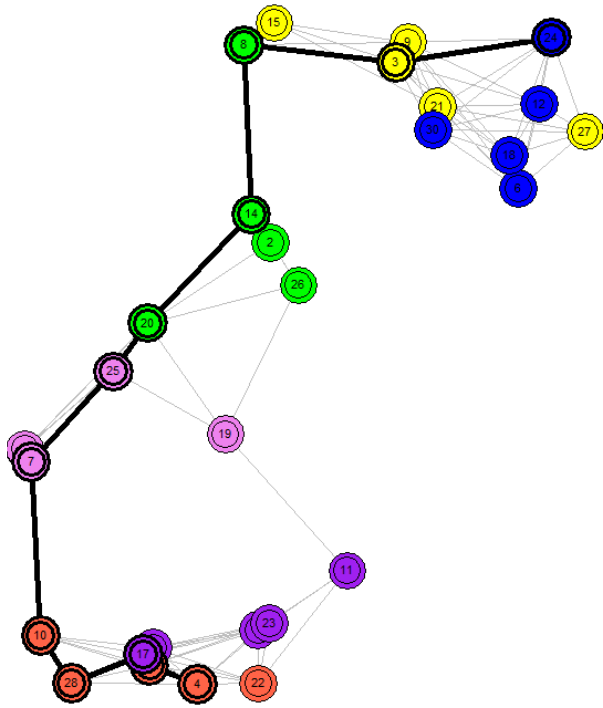


Figure 1.22: Example of a widest path from node 4 to node 24.

Figure 1.23: Example of a widest path from node 24 to node 4.

SWP 4->10->7->25->20->14->8->3->24

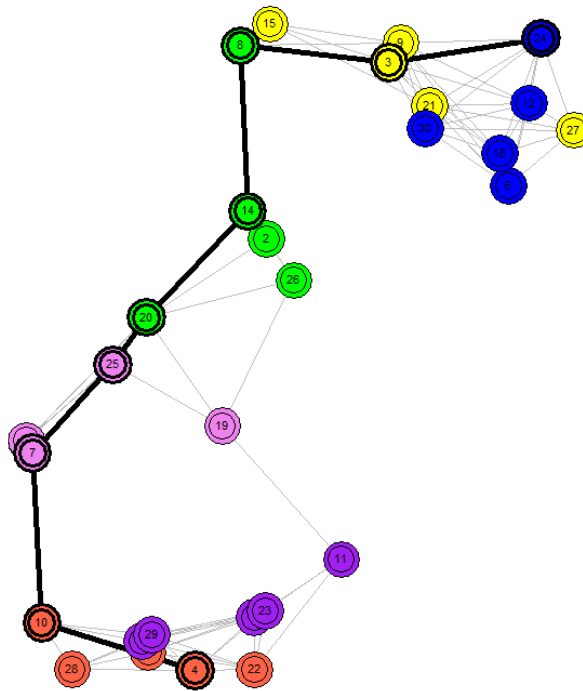


Figure 1.24: Example of a shortest widest path between nodes 4 and 24.

1.8 Conclusions

As first result, this chapter introduces the cost functions J_0^L and J_0^X , which can be used as a benchmark to compare different clustering solutions. These functions measure the quality of a network partition using end-to-end path calculations for additive metrics. They take into account the fact that inter-cluster and intra-cluster communications have different costs and are flexible enough to cover both cases when the traffic distribution depends on the groups or not. During the elaboration of J_0^X , it has been shown that routing should take into account the cluster structure to find good network paths and offer the best QoS to the user traffic.

A second result achieved in this chapter is that making use of group information for clustering leads to better network performance. This result was illustrated with the practical application of J_0^X to intra-group delays. Thanks to this new cost function it has also been shown that using a simple naive approach consisting of building one cluster per group does not lead to the best network performance. This justifies the need for more advanced clustering solutions using group information.

Finally J_0^X was extended in two ways. Firstly, to better take cluster size into account, and avoid that $\{\mathcal{V}\}$ be the best partition; the resulting function is denoted by \bar{J}_0^X . Secondly, to cope with the throughput concave metric; the resulting function is denoted by T_0^X .

The metrics introduced here are used in Chapters 2 and 3 to assess the performance of new distributed clustering algorithms.

Chapter 2

Distributed clustering algorithm with operational groups

The work presented in this chapter has been partially published in [39–41].

2.1 Introduction

Public safety and military networks are organized into a hierarchical structure leading to the existence of *operational groups* (e.g., squad, section, etc.). In those networks, nodes exhibit group mobility behavior. Additionally, the traffic is strongly dependent on the network hierarchical organization, being mostly concentrated within operational groups. For these two reasons, the clustering solution should take into account operational group information in order to provide a better end-to-end QoS and to improve network stability. As demonstrated in Chapter 1 the trivial solution consisting in building one cluster per group is not acceptable. Thus to cope with structured networks based on operational groups, specific clustering solutions are required.

Numerous distributed clustering schemes have been proposed in the literature. These algorithmic solutions first select Cluster Head (CH) nodes, and then the other nodes affiliate to them, leading to the different clusters. A weight is commonly associated with each node and the nodes with the highest weights in a neighborhood are selected to be the CH nodes. The weights can be the node identifier, the node degree, the remaining battery power, metrics related to radio measurements etc., or a combination of them [1, 8, 9]. To get the node weight, other solutions rely on the knowledge of nodes' location and speed, obtained thanks to, for instance, a GPS device [10]. In a second step, when CH nodes have been selected, non-CH nodes affiliate to CH nodes depending on their weight or any other metric.

Only a few papers consider group information for building the clusters. The authors in [15] introduce the type-based clustering algorithm (TCA). This clustering scheme associates a stability factor to each node and selects as CH the nodes that have the highest stability factor in a radio neighborhood. The stability factor takes group membership (identified thanks to the IP subnet of each node) into account. A limitation of TCA lies in the fact that two CH nodes cannot be neighbors. A direct consequence in dense networks is the formation of clusters with a large number of members. Two distributed clustering algorithms, GDMAC [8] and VOTE [9], are well-known to handle the size of clusters but they do not take into account the group structure.

We have identified a clear gap in the ad hoc network clustering literature since there is no

clustering solution suited to structured network that properly take cluster size into account. To fill this gap we first consider extensions of existing CH-based clustering solutions. We find that the performance gains achieved thanks to these extensions are inherently limited by the algorithms from which they are derived. A challenge to overcome when designing CH-based algorithms is how to calculate the node weights used to elect the CH nodes. In structured networks, it make sense to assign the weight of a node based on its group. The drawback of this approach is that comparing the weight of nodes from different groups is very difficult. Also non-CH nodes may choose their CH based on its weight, which could depend on the current state of its cluster. For example the CH of a large cluster could have a greater weight than a CH of a small cluster. Unfortunately, this kind of strategy leads to a lot of instability in the cluster structure. We thus decided to follow a clean slate approach to design a new distributed clustering algorithm suited to structured networks, and search for algorithms that do not involve any CH. This choice allows us to form clusters whose members are peers, and thus avoids the aforementioned problem of allocating a weight to the nodes. In addition, since the CH can be a single point of failure within a cluster, our algorithm does not suffer from this problem.

Section 2.2 introduces the network model. Before going into the definition of the new distributed clustering algorithm, Section 2.3 details several new cluster cost functions and assesses their capability to be used to find good partitions with respect to the J_0 metric, taken as a reference. Then Section 2.4 explains how, starting from the best cost function identified in Section 2.3, we designed our distributed clustering algorithm. The main content of this chapter is in Section 2.5 where the Distributed Clustering with Operational Groups (DCOG) algorithm is detailed. Finally, Section 2.6 is devoted to simulation results and analysis.

2.2 Network model

In this chapter the network model of Section 1.3 is extended introducing the following notations. The cluster group diversity o_k is the number of groups with at least one member in cluster \mathcal{C}_k . The index of the group with the highest number of members in cluster \mathcal{C}_k is o_k^* . The number of members of group \mathcal{O}_t in cluster \mathcal{C}_k is $m_{t,k}$. Let $\mathcal{I}(\mathcal{C}_k) := \{t | m_{t,k} \neq 0, t \in \{1, \dots, T\}\}$ the set of the indices of groups with at least one member in cluster \mathcal{C}_k .

To simplify notations in this chapter the cost functions J_0^X and T_0^X from Chapter 1 are denoted by J_0 and T_0 .

2.3 First step from the global network cost function to a distributed clustering solution

2.3.1 Definition of simple cost functions

The function J_0 (J_0^X defined in Section 1.4) requires a global knowledge of the network and cannot be used during a distributed clustering process. Thus, we now define new cluster cost functions c_i that can be calculated by each cluster based on the knowledge available locally. They must take into account that:

- Nodes of the same group should belong to the same cluster, and

- The number of nodes in the clusters should not exceed a maximum value, associated with the number of nodes best handled by the radio resource allocation feature within a cluster.

Let us introduce the global network cost function J_i written as a sum of the cluster costs over all the clusters of the network:

$$J_i := \sum_{k=1}^{N_c} c_i(\mathcal{C}_k), \quad (2.1)$$

with $c_i(\mathcal{C}_k)$ the cost of cluster \mathcal{C}_k associated with J_i .

In this section our goal is to identify some good criteria that could be used in a distributed clustering process. We define several J_i by their corresponding c_i , and calculate their value for each partition of \mathcal{G} . We retain as best partitions the ones that minimize J_i .

The first cost function is:

$$c_1(\mathcal{C}_k) := o_k.$$

Minimizing function J_1 leads to minimizing the cluster group diversity across all clusters, meaning that the clusters are more or less equal to the groups.

The second and third functions are c_2 and c_3 , which take into account group information in a different way:

$$c_2(\mathcal{C}_k) := -\frac{m_{o_k^*,k}}{m_{o_k^*}},$$

$$c_3(\mathcal{C}_k) := (m_{o_k^*} - m_{o_k^*,k})^2.$$

The idea of these two cost functions is to find in each cluster the group o_k^* with the highest number of members $m_{o_k^*,k}$, and to compare it to the total number of members in this group $m_{o_k^*}$. Having these numbers close to each other is another way of having clusters more or less equal to the groups.

In addition to using group related metrics, the last two functions c_4 and c_5 introduce a new variable that is important from a radio resource allocation point of view: the size of the clusters. They are defined as follows:

$$c_4(\mathcal{C}_k) := (n_{max} - n_k + 1) \cdot o_k,$$

$$c_5(\mathcal{C}_k) := (n_{max} - n_k + 1) \cdot (m_{o_k^*} - m_{o_k^*,k} + 1).$$

The +1 in c_4 and c_5 corresponds to the fact that if the size of the cluster is maximum, we still want to take group information into account.

2.3.2 Performance assessment

Let us define \mathcal{P}_i^* , $\forall i \in \{0, 1, 2, 3, 4, 5\}$, as the set of the optimal partitions for J_i :

$$\mathcal{P}_i^* := \arg \min_{p \in \mathcal{P}^c} J_i(p).$$

A function J_i , $i \neq 0$, could be considered as good as J_0 if it yields the same best partitions: $\mathcal{P}_i^* = \mathcal{P}_0^*$. However this never happens because none of the J_i is as good as J_0 . Therefore, our

first criterion to assess if a function J_i is good or not is whether it can be used to identify some of the best partitions w.r.t. J_0 , i.e., if $\mathcal{P}_i^* \cap \mathcal{P}_0^* \neq \emptyset$.

To evaluate such capability, we have simulated 100 networks using the setup described in Section 1.5.2. Each cell of Table 2.1 contains the number of networks $n_{\alpha,i}$ for which $\mathcal{P}_i^* \cap \mathcal{P}_0^* \neq \emptyset$. The last two lines of the table are the sums of $n_{\alpha,i}$ over several values of α for a given i . In this table the cells are colored in order to help differentiating good J_i from bad J_i , for a given α value (which is the probability that one node communicates with a node of the same group, see Section 1.4.5). Let $i_\alpha^* = \arg \max_{i \in \{1,2,3,4,5\}} n_{\alpha,i}$ and $\bar{i}_\alpha^* = \arg \min_{i \in \{1,2,3,4,5\}} n_{\alpha,i}$. The green color identifies $J_{i_\alpha^*}$, i.e., the J_i function that succeeds in finding at least one best partition in the largest number of networks. Likewise the red color identifies $J_{\bar{i}_\alpha^*}$. Let $n_\alpha = (n_{\alpha,i_\alpha^*} + n_{\alpha,\bar{i}_\alpha^*})/2$. If $n_\alpha \leq n_{\alpha,i} < n_{\alpha,i_\alpha^*}$ then the cell is colored in blue, and if $n_{\alpha,\bar{i}_\alpha^*} < n_{\alpha,i} < n_\alpha$, then it is colored in orange.

α	J_1	J_2	J_3	J_4	J_5
0.2	1	0	4	8	4
0.3	3	0	8	15	8
0.4	5	0	13	21	14
0.5	11	2	28	26	22
0.6	15	4	32	25	25
0.7	31	17	41	23	25
0.8	63	34	56	30	31
0.9	86	55	61	25	28
$\sum_{\alpha=0.2}^{\alpha=0.9}$	215	112	243	173	157
$\sum_{\alpha=0.2}^{\alpha=0.6}$	35	6	85	95	73

Table 2.1: Number of networks with $\mathcal{P}_i^* \cap \mathcal{P}_0^* \neq \emptyset$ (number of trials = 100).

A first remark is that all J_i achieve larger values of $n_{\alpha,i}$ when α increases. This is expected because an increasing α values means that group membership has a larger impact on the user traffic load, thus taking into account groups should help in finding better partitions. The penultimate line can be used to establish a partial ordering between all J_i considering all values of α : $J_3 > J_1 > J_4 > J_5 > J_2$. The function J_3 is the best only for α values from 0.5 to 0.7, and offers consistently good performance for all values of α . The function J_2 is consistently the worst. The case of the second best function J_1 is very interesting: when groups have a low impact on J_0 values its performance is mediocre, but improves with increasing group impact, finally becoming the best. The same kind of argument, reversed, can be made also for J_4 and J_5 : their performance are good for low α values, but degrade as α value increases (J_4 even becoming the worst when $\alpha \geq 0.8$). Summing the numbers over $\alpha \in \{0.2, 0.3, 0.4, 0.5, 0.6\}$ leads to a different partial ordering between all J_i , as shown in the last line: $J_4 > J_3 > J_5 > J_1 > J_2$. This result hints that the J_i should be selected depending on the importance of the groups on traffic.

The fact that the proposed cost functions succeed in finding some best partitions w.r.t. J_0 is a good result. However, we now prove that this criterion is not enough. Let us for example consider a cluster cost function c_{i_0} leading to a network cost function J_{i_0} such that any partition is a best partition: $\mathcal{P}_{i_0}^* = \mathcal{P}^c$. In that case, for any random network we have $\mathcal{P}_{i_0}^* \cap \mathcal{P}_0^* = \mathcal{P}_0^* \neq \emptyset$.

In Table 2.1 such a function would lead to a value $n_{\alpha,i} = 100$ and would be identified as the best. Let us define $\bar{\mathcal{P}}_i^*, \forall i \in \{0, 1, 2, 3, 4, 5\}$, as the set of the worst partitions for J_i :

$$\bar{\mathcal{P}}_i^* := \arg \max_{p \in \mathcal{P}^c} J_i(p).$$

We have $\mathcal{P}_{i_0}^* \cap \bar{\mathcal{P}}_0^* = \bar{\mathcal{P}}_0^*$, meaning that J_{i_0} also identifies as best partitions the partitions that are the worst from the point of view of J_0 . We thus need another criterion to assess the quality of a cluster cost function c_i .

We now define this second criterion, following a methodology similar to the one of Section 1.5.1. Let us write p_j^i , with $j \in \{1, 2, \dots, |\mathcal{P}_i^*|\}$, the members of \mathcal{P}_i^* , and define $J_{0,i}^*$ as the highest J_0 value of the partitions in \mathcal{P}_i^* (the worst according to J_0):

$$J_{0,i}^* := \max_{p \in \mathcal{P}_i^*} J_0(p).$$

Let us write \bar{p}_j^i , with $j \in \{1, 2, \dots, |\bar{\mathcal{P}}_i^*|\}$, the members of $\bar{\mathcal{P}}_i^*$, and define $\bar{J}_{0,i}^*$ as the lowest J_0 value of the partitions in $\bar{\mathcal{P}}_i^*$ (the best according to J_0):

$$\bar{J}_{0,i}^* := \min_{p \in \bar{\mathcal{P}}_i^*} J_0(p).$$

Let us take the example of Fig 2.1, where $\mathcal{P}_i^* = \{p_1^i, p_2^i, p_3^i, p_4^i, p_5^i\}$ with $J_{0,i}^* = J_0(p_3^i)$, and $\bar{\mathcal{P}}_i^* = \{\bar{p}_1^i, \bar{p}_2^i, \bar{p}_3^i, \bar{p}_4^i\}$ with $\bar{J}_{0,i}^* = J_0(\bar{p}_2^i)$.

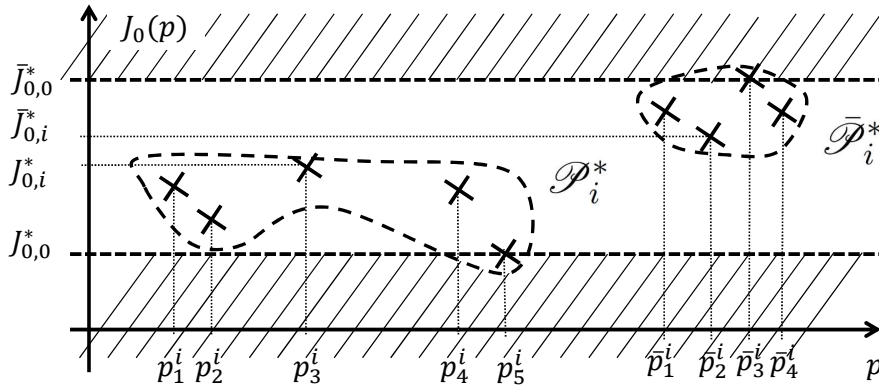


Figure 2.1: Metrics used to assess J_i : example to illustrate $J_{0,0}^*$, $J_{0,i}^*$, $\bar{J}_{0,i}^*$ and $\bar{J}_{0,0}^*$.

To measure how good is a function J_i at finding partitions with good J_0 values, we consider the quantity $J_{0,i}^* - J_{0,0}^*$. We also use $\bar{J}_{0,i}^* - J_{0,0}^*$ to measure if J_i is capable to find partitions with bad J_0 values. Because these values depend on the network trial, $J_{0,i}^* - J_{0,0}^*$ and $\bar{J}_{0,i}^* - J_{0,0}^*$ must be normalized. We now introduce the values δ_i^* and $\bar{\delta}_i^*$ normalized in $[0, 1]$:

$$\begin{cases} \delta_i^* := \frac{J_{0,i}^* - J_{0,0}^*}{J_{0,0}^* - J_{0,0}^*}, \\ \bar{\delta}_i^* := \frac{\bar{J}_{0,i}^* - J_{0,0}^*}{J_{0,0}^* - J_{0,0}^*}. \end{cases} \quad (2.2)$$

In order for J_i to be a good substitute to J_0 , δ_i^* must be as close to 0 as possible. If the value of $\bar{\delta}_i^*$ is close to 1, it means that J_i can also find very bad J_0 partitions.

The cumulated distribution functions of δ_i^* are plotted in Fig. 2.2. For example when $\alpha = 0.2$, the probability that δ_4^* is less than 0.3 is 0.6, i.e., 60% of partitions considered as best by J_4 have a corresponding J_0 value which is in the first 30% of the J_0 values interval.

These figures show that when α increases from 0.2 to 0.6, the partial ordering between the J_i remains constant: $J_5 > J_4 > J_3 > J_1 > J_2$. This result is different from the one of Table 2.1. This means that J_4 is the best if the capability of finding best partitions w.r.t. J_0 is the most important, but that J_5 is the best in probability: it is likely that any partition found as best using J_5 will have a lower value than a partition found as best using J_4 .

The function J_5 remains the best for all values of α no greater than 0.8. It is only when $\alpha = 0.9$ that J_1 becomes the best and J_5 the second best. Also, J_3 identified as best J_i in Table 2.1 is never the best in Fig. 2.2. This confirms the importance of choosing a good criterion to select the J_i . In view of its capacity to deliver greater details than our first criterion, we consider our second criterion as the most relevant.

The frequency diagrams of $\bar{\delta}_i^* - \delta_i^*$ have been drawn in Fig. 2.3 and Fig. 2.4 for J_5 , identified as the globally best J_i (except for $\alpha = 0.9$). The x values on the x-axis, indicate values of $\bar{\delta}_i^* - \delta_i^*$ in the $[x - 0.2, x]$ interval. When strictly positive these values quantify, the larger the better, the fact that J_i finds as its best partitions ($p_i^* \in \mathcal{P}_i^*$), partitions that have better J_0 values than the partitions that J_i finds as its worst partitions ($\bar{p}_i^* \in \bar{\mathcal{P}}_i^*$). Negative values are associated with the inverse undesired behavior. The y value indicates the number of networks. For example in the frequency diagram associated with $\alpha = 0.2$ of Fig. 2.3, the y value of the bar at $x = 0.2$ is 34. This means that for 34 of the 100 random networks, all partitions identified as the best by J_5 have better J_0 values, by a factor in $[0, 0.2]$, than all partitions identified as the worst by J_5 .

Fig. 2.3 and Fig. 2.4 show the capability of J_5 to discriminate between good and bad partitions w.r.t. J_0 . It is lower with low value of α and improves when α increases. As soon as $\alpha \geq 0.5$, it is always the case.

The results obtained so far allow to determine that if group membership is essential w.r.t. the traffic load (i.e. high values of α), J_1 is the best function. However, J_5 is always good $\forall \alpha$, indicating that cluster size is also important. This makes sense because the number of inter-cluster communications also depend on the number of clusters, i.e., on cluster size. Thus, in the following we use the cost function c_5 to design a distributed clustering algorithm.

2.4 Towards a distributed clustering solution

This section replicates in chronological order the steps that led to the definition of the cost function used in the distributed clustering scheme proposed in this chapter, as well as the way to handle it. Accordingly with the analysis performed in Section 2.3, we chose c_5 as initial cost function. Concerning the algorithm describing how the clusters are modified, we initialize it with each node being a singleton cluster. Let us now detail some of the steps that led to the definition of the distributed clustering solution of Section 2.5.

2.4.1 First intermediate solution

The principle of the first algorithm was to split time in rounds, and to divide each round in two steps: the decision making and its application. We found in Section 2.3 that the global cost function J_5 allowed to select good partitions. Consequently, to perform the first step of the

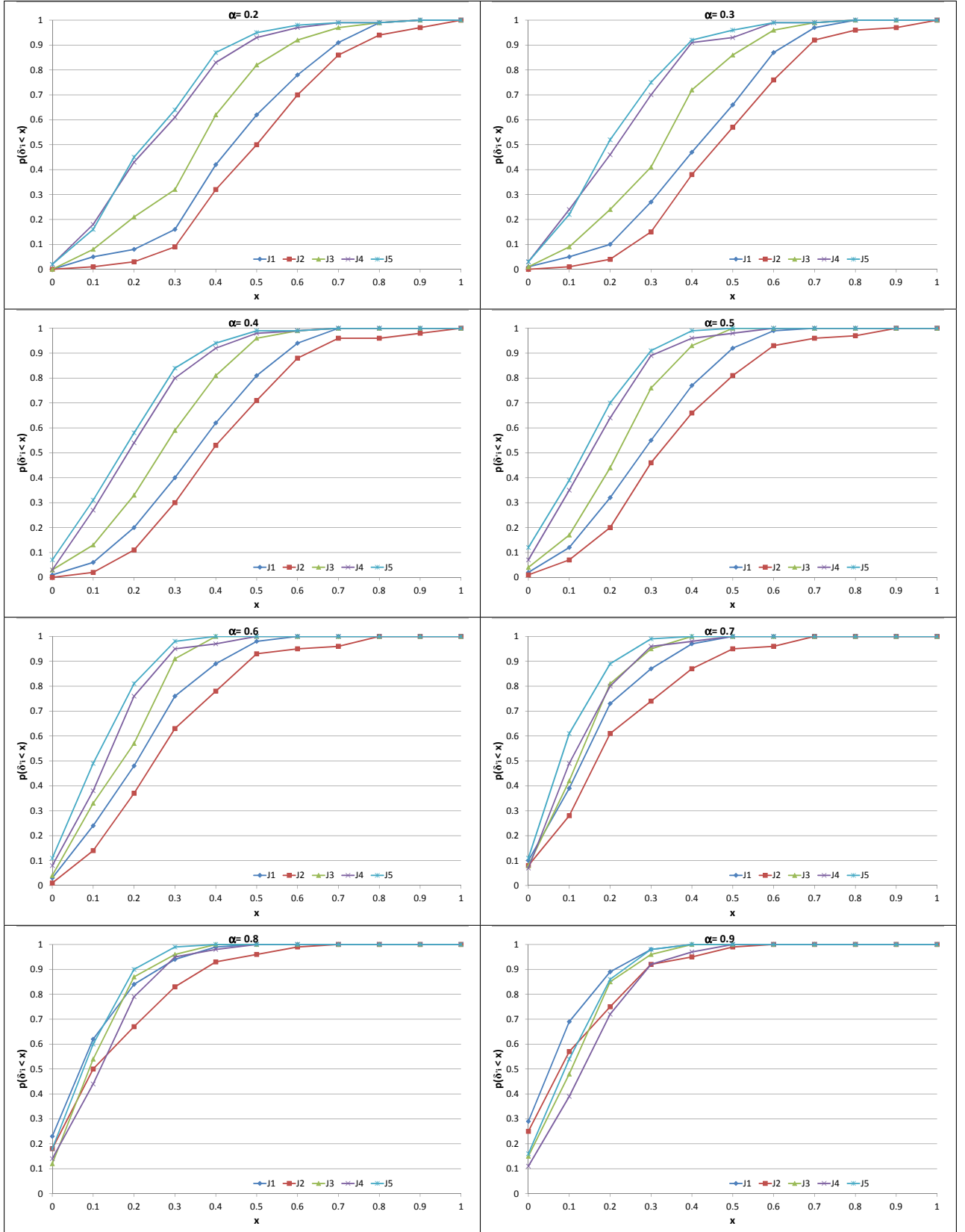


Figure 2.2: cdf of δ_i^* vs. α .

algorithm we first chose the cluster cost function c_5 derived from J_5 :

$$c_5(\mathcal{C}_k) := (n_{max} - n_k + 1) \cdot (m_{o_k^*} - m_{o_k^*,k} + 1).$$

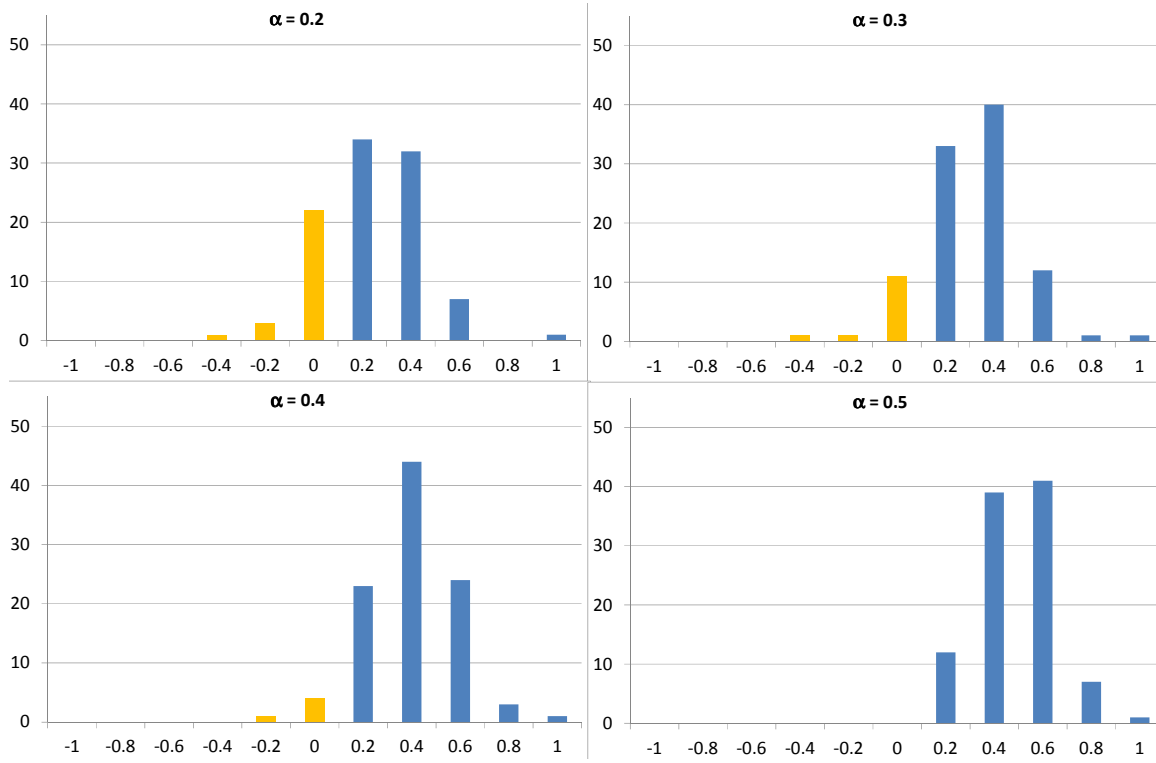


Figure 2.3: $\bar{\delta}_5^* - \delta_5^*$ frequency diagrams for $\alpha \in \{0.2, 0.3, 0.4, 0.5\}$.

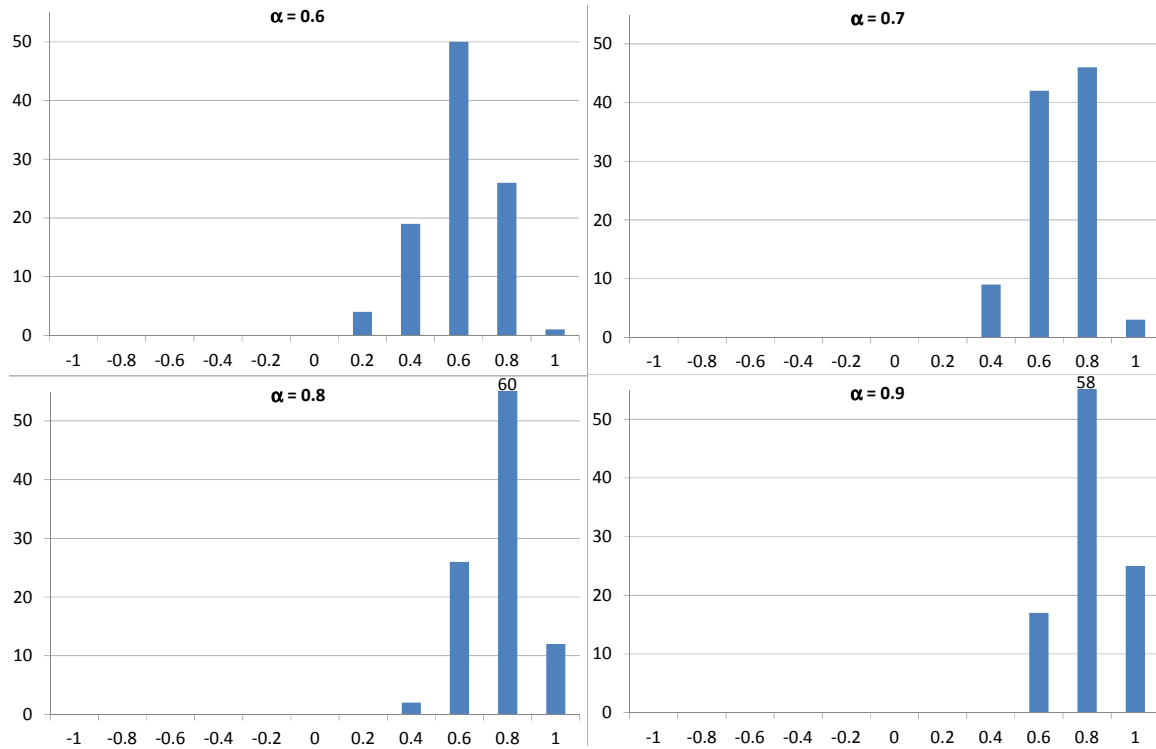


Figure 2.4: $\bar{\delta}_5^* - \delta_5^*$ frequency diagrams for $\alpha \in \{0.6, 0.7, 0.8, 0.9\}$.

2.4.1.1 Step 1: decision making

During step 1 in each cluster \mathcal{C}_k each node i checks if it is valuable to move from its current cluster to each of its neighboring cluster \mathcal{C}_ℓ . To check this, the loss induced by node i leaving

its cluster is calculated:

$$\text{loss}(i, \mathcal{C}_k) := c(\mathcal{C}_k \setminus \{i\}) - c(\mathcal{C}_k). \quad (2.3)$$

Then the gain induced by the arrival of node i in \mathcal{C}_ℓ is calculated:

$$\text{gain}_1(i, \mathcal{C}_k, \mathcal{C}_\ell) := c(\mathcal{C}_\ell \cup \{i\}) - c(\mathcal{C}_k). \quad (2.4)$$

with c the chosen cluster cost function, i.e. c_5 . We decided that a node could leave its cluster \mathcal{C}_k to join another cluster \mathcal{C}_ℓ if:

$$\text{gain}_1(i, \mathcal{C}_k, \mathcal{C}_\ell) > -\text{loss}(i, \mathcal{C}_k). \quad (2.5)$$

If more than one neighbor cluster \mathcal{C}_ℓ exist, and that, at the same time satisfy (2.5) and maximize the $\text{gain}_1(i, \mathcal{C}_k, \mathcal{C}_\ell)$, then one is randomly chosen among them and node i is selected to join \mathcal{C}_ℓ . Otherwise node i does not leave its cluster \mathcal{C}_k .

In addition to this condition, to allow a node to join a neighbor cluster, the connectivity, size and diameter constraints must also be satisfied.

2.4.1.2 Step 2: decision application

At the end of step 1, it has been decided that some nodes should leave their current cluster to join a neighboring one. We call this an inter-cluster nodes swap, or more shortly a swap. During step 2, the selected swap is performed.

2.4.1.3 Behavior of first intermediate solution

Several problems were identified with this solution:

- When multiple groups have the same number of members in a cluster \mathcal{C}_k , if this number is also the highest among all groups in \mathcal{C}_k , then o_k^* has not a unique value. If in addition the groups do not have the same size (multiple values for $m_{o_k^*, k}$), then the value of $c_5(\mathcal{C}_k)$ is implementation dependent.
- Notwithstanding this problem, when multiple groups have at least one member in a cluster, only the one with the highest number of members in the cluster is taken into account by J_5 . The problem is similar for all J_i from Section 2.3.

2.4.2 Second intermediate solution

To solve both problems identified in the previous solution, we modified the cost function in order to take into account all groups with at least one member in the cluster.

2.4.2.1 Modification of the cost function

We defined the new following cost function:

$$c'_5(\mathcal{C}_k) := \sum_{t \in \mathcal{I}(\mathcal{C}_k)} \left[1 - \frac{m_{t,k}}{m_{min}} \right] + 1_{|\mathcal{I}(\mathcal{C}_k)| \leq \Omega} \left[\Omega - |\mathcal{I}(\mathcal{C}_k)| \right], \quad (2.6)$$

with $m_{min} := \arg \min_{t \in \{1, \dots, N_g\}} m_t$ the size of the smallest group, and Ω a constant.

In the first term, m_{min} has been introduced to favor the formation of clusters including the groups that have the biggest size. The rationale to justify the second term is that the cluster size being limited, only a limited number of groups ($\leq \Omega$) should be represented in any cluster. Otherwise a penalty is applied to the cost of the considered cluster .

2.4.2.2 Behavior of second intermediate solution

The loss induced by a node i leaving a cluster \mathcal{C}_k is the same as the gain induced by the arrival of node i in any cluster \mathcal{C}_ℓ . Consequently, the algorithm remains in its initial state.

2.4.3 Third intermediate solution

2.4.3.1 Modification of the cost function

To solve the aforementioned problem, we modified c'_5 . We replaced the linear function $1 - m_{t,k}/m_{min}$ used in (2.6) by function f defined as:

$$f(m_{t,k}) := \frac{m_{t,k}(m_{t,k} + 1)}{m_{min}(m_{min} + 1)}. \quad (2.7)$$

Thanks to (2.7) a gain equal to $n + 1$ is achieved when in a cluster the number of members of a group increases from n to $n + 1$. Let us notice that the following property holds:

$$f(m_{t,k} + 1) - f(m_{t,k}) < f(m_{t,k} + 2) - f(m_{t,k} + 1).$$

Thanks to this property, it is always better from a cost perspective to add members of a group to a cluster already including the highest number of members of this group.

We thus defined the following cost function:

$$c''_5(\mathcal{C}_k) := \sum_{t \in \mathcal{I}(\mathcal{C}_k)} \left[1 - \frac{m_{t,k}(m_{t,k} + 1)}{m_{min}(m_{min} + 1)} \right] + 1_{|\mathcal{I}(\mathcal{C}_k)| \leq \Omega} \left[\Omega - |\mathcal{I}(\mathcal{C}_k)| \right]. \quad (2.8)$$

In Chapter 3, we introduce a class of functions which includes (2.7) as a specific case.

2.4.3.2 Modification of the decision making step

Because $c(\mathcal{C}_k)$ appears in both $\text{gain}_1(i, \mathcal{C}_k, \mathcal{C}_\ell)$ and $\text{loss}(i, \mathcal{C}_k)$, a node in group \mathcal{O}_t may leave its cluster \mathcal{C}_k to join cluster \mathcal{C}_ℓ , even if there are more members of \mathcal{O}_t in \mathcal{C}_k than in \mathcal{C}_ℓ . To fix this issue, we defined the new gain $\text{gain}_2(i, \mathcal{C}_\ell)$ induced by the arrival of node i in cluster \mathcal{C}_ℓ :

$$\text{gain}_2(i, \mathcal{C}_\ell) := c(\mathcal{C}_\ell \cup \{i\}) - c(\mathcal{C}_\ell). \quad (2.9)$$

Also, when using the algorithm of Section 2.4.1.1, the loss induced by a node leaving its singleton cluster is equal to the gain induced by this node joining a cluster not yet including any node of its group. To allow a cluster to include members of several groups the loss associated with the vanishing of a singleton cluster was set to zero.

2.4.3.3 Behavior of third intermediate solution

Some clustering now happens: provided that the constraints allow it, the nodes of the same groups join together into the same cluster, forming one cluster per group. In addition to this, we would like that if the maximum cluster size allows it, more than one group can be included in a single cluster.

2.4.4 Fourth intermediate solution

2.4.4.1 Modification of the cost function

We found out that in (2.8) the indicator function was unnecessary and thus we removed it. We also modified the cost function to make it normalized in $[0, 1]$. The next cost function was defined as:

$$c_5'''(\mathcal{C}_k) := 1 - \frac{1}{n_{max}} \sum_{t \in \mathcal{I}(\mathcal{C}_k)} m_{t,k} \cdot \frac{m_{t,k}(m_{t,k} + 1)}{m_t(m_t + 1)}. \quad (2.10)$$

In this equation, multiplying the fraction $[m_{t,k}(m_{t,k} + 1)]/[m_t(m_t + 1)]$ by $m_{t,k}$ favors the completion of the biggest groups.

2.4.4.2 Modification of the decision making step

Instead of allowing single nodes to be swapped between clusters, we now allow sets of nodes to change cluster together. More precisely, within a cluster all members of the same group may leave their current cluster to join a neighboring cluster. This modification allows to gather members from multiple groups in the same cluster. Also, to allow a set of nodes to move to a neighboring cluster, the connectivity, size and diameter constraints must be satisfied in both the source and the destination cluster.

2.4.4.3 Behavior of fourth intermediate solution

This solution seemed to succeed in building adequate clusters for a variety of networks. Yet a last improvement to simplify the decision making step could be done.

2.4.5 Final solution

In Section 2.4.3, the decision making step was modified to set to zero the loss associated with the vanishing of a singleton cluster. This policy had also to be applied when a non-singleton cluster vanishes because all its members from the same group move to another cluster.

Instead of handling these cases in a specific way, the cost function of (2.10) could be modified in order to favor the formation of big clusters. The new function became:

$$c_6(\mathcal{C}_k) := 1 - \left[\frac{n_k(n_k + 1)}{n_{max}(n_{max} + 1)} \cdot \epsilon + \frac{1}{n_{max}} \sum_{t \in \mathcal{I}(\mathcal{C}_k)} m_{t,k} \cdot \frac{m_{t,k}(m_{t,k} + 1)}{m_t(m_t + 1)} \cdot (1 - \epsilon) \right], \quad (2.11)$$

with ϵ chosen in order to either favor (ϵ close to 0) the formation of clusters including complete groups, or to only favor (ϵ close to 1) large clusters.

In Chapter 3 we detail the properties that must be fulfilled by cluster cost functions usable to perform distributed clustering, (2.11) being only a specific example.

2.4.6 Simulations

Since the final cost function c_6 has been defined in (2.11), we now assess its performance using the same methodology as described in Section 2.3. We set $n_{max} = 8$, $\alpha = 0$ and $\epsilon = 0$ (in order to only take into account group membership). Simulations have been performed for 100 networks, and for each network the best partition w.r.t. J_6 has been found and compared to

the partitions with the lowest and highest J_0 values. Fig. 2.5 details the simulation results, including the ones for the functions J_1 , J_2 , J_3 , J_4 and J_5 from Section 2.3.

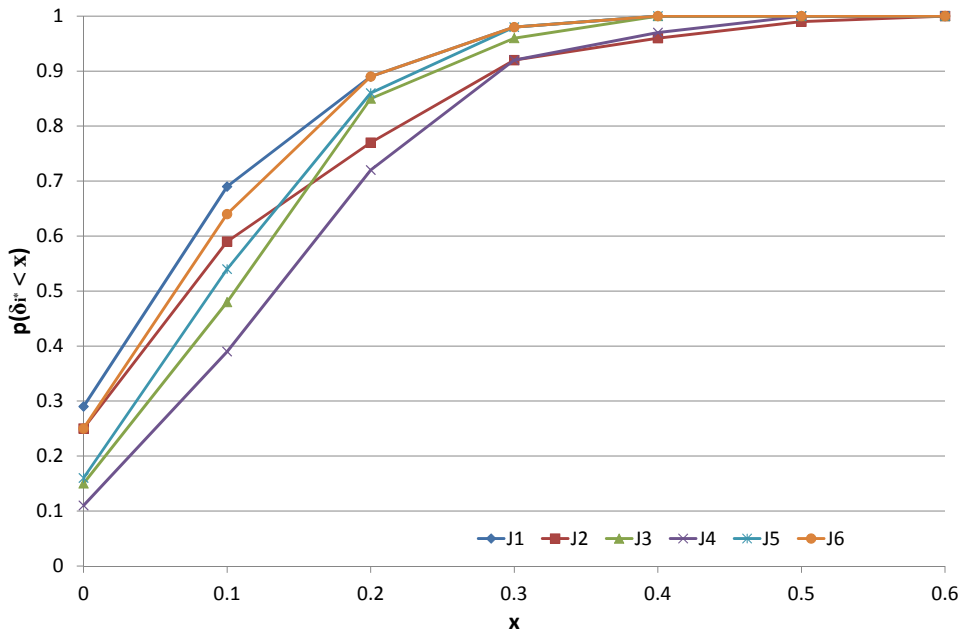


Figure 2.5: cdf of δ_i^* for $\alpha = 0.9$.

The partitions obtained thanks to J_6 are nearly as good as the one obtained thanks to J_1 , which is the best for $\alpha = 0.9$. This achievement foretells good results for our distributed algorithm using c_6 . Now that the methodology that we used to define our distributed clustering algorithm has been detailed, we can introduce the algorithm itself.

2.5 Distributed Clustering with Operational Group

In this section we start by introducing the principles of DCOG, starting from a generic cluster cost function c . Then, we detail the DCOG algorithm based on the results of Section 2.4, i.e., a way to compare node swaps between clusters, and a cluster cost function requiring only local cluster information. Finally we explain how DCOG is adapted to node mobility.

2.5.1 Principles of DCOG

The DCOG algorithm is a distributed clustering algorithm that is run continuously by all network nodes in order to build clusters satisfying the two following properties: any cluster should both *i*) include nodes of the same operational groups (as much as possible), and *ii*) include the largest number of nodes, while satisfying the following constraints: it must be connected, its size must be less or equal to n_{max} , and its diameter must be less or equal to d_{max} . Note that for (*i*), all the nodes of a group may not be able to be gathered into one single cluster due to the constraints.

Conversely to most of the clustering algorithms that can be found in the literature, DCOG algorithm does not need to resort to the notion of CH node. Instead, DCOG manages a trading process between the clusters, each cluster evaluating periodically the opportunity to give away

some of its nodes to a neighbor cluster. Once an opportunity is found, the swap is implemented until the next opportunity.

In order to evaluate if a swap of nodes is possible between two clusters, we introduce two different metric functions: a cluster one (noted c) that gives a score depending on the nodes inside the cluster, and a gain one (noted g) that is a function of c for both clusters involved (details are given in the subsequent sections). The decision to let the swap to happen is based upon the enforcement of the two following conditions: *i*) $g > 0$, and *ii*) the constraints are met.

2.5.2 Functions related to DCOG

In this section we denote by \mathcal{C} the set of all possible clusters.

The cluster cost function used in DCOG is an application that associates a real value in $[0, 1]$ to a cluster \mathcal{C}_k , defined as:

Definition 2.1 *The DCOG cluster cost function c is defined as:*

$$\begin{aligned} c : \mathcal{C} &\mapsto [0, 1] \\ \mathcal{C}_k &\mapsto c(\mathcal{C}_k), \end{aligned}$$

such that $c(\emptyset) = 1$, and $c(\mathcal{C}_k) = 0$ when \mathcal{C}_k fulfills the targeted properties.

Function c is used to define the gain g , and specific implementation is given in Section 2.5.5.

As described previously, DCOG uses a function g to evaluate the benefits of swapping a set of nodes $\{u_i\}$ from one cluster \mathcal{C}_k to a cluster \mathcal{C}_ℓ , with $k \neq \ell$. A gain function is defined as:

Definition 2.2 *The DCOG cluster gain function g is defined, $\forall k, \forall \ell$, with $k \neq \ell$ as:*

$$\begin{aligned} g : \mathcal{V} \times \mathcal{C} \times \mathcal{C} &\mapsto \mathbb{R} \\ (\{u_i\}, \mathcal{C}_k, \mathcal{C}_\ell) &\mapsto g(\{u_i\}, \mathcal{C}_k, \mathcal{C}_\ell) \end{aligned}$$

with $g(\{u_i\}, \mathcal{C}_k, \mathcal{C}_\ell) := [c(\mathcal{C}_\ell) - c(\mathcal{C}_\ell \cup \{u_i\})] - [c(\mathcal{C}_k \setminus \{u_i\}) - c(\mathcal{C}_k)]$.

The gain $g(\{u_i\}, \mathcal{C}_k, \mathcal{C}_\ell)$ results from the difference of: *i*) the gain associated with the arrival of nodes $\{u_i\}$ in \mathcal{C}_ℓ previously introduced in Section 2.4.3.2 and equal to $c(\mathcal{C}_\ell) - c(\mathcal{C}_\ell \cup \{u_i\})$, and *ii*) the loss associated with the departure of nodes $\{u_i\}$ from \mathcal{C}_k and already defined in Section 2.4.1.1, equal to $c(\mathcal{C}_k \setminus \{u_i\}) - c(\mathcal{C}_k)$.

To make sure that any inter-cluster nodes swap decided by DCOG is beneficial, we impose the following property:

Property 2.1 *With DCOG, an inter-cluster nodes swap is allowed only if the associated gain is strictly positive.*

The action of the DCOG clustering algorithm is to only allow swaps that induce a strictly positive gain. To understand the consequence of this property of DCOG, we now introduce the concept of network cost function as the sum of the costs of all clusters in the network. The evolution of the network cost during DCOG execution is instrumental to prove its convergence to a stable partition.

Let us denote by $\mathcal{C}_k(t)$ and $p(t)$ the cluster \mathcal{C}_k and partition p at time t , respectively. Omitting the subscript i , let us consider the cost function c associated to the global cost function J defined in (2.1).

Result 2.1 *If between time t_n and t_{n+1} , the only swap that happened is the departure of nodes $\{u_i\}$ from cluster \mathcal{C}_k towards cluster \mathcal{C}_ℓ , then the following equality holds:*

$$J(p(t_{n+1})) = J(p(t_n)) - g(\{u_i\}, \mathcal{C}_k(t_n), \mathcal{C}_\ell(t_n)). \quad (2.12)$$

Proof Before nodes $\{u_i\}$ move from cluster \mathcal{C}_k to cluster \mathcal{C}_ℓ we have: $J(p(t_n)) = c(\mathcal{C}_k(t_n)) + c(\mathcal{C}_\ell(t_n)) + \sum_{j \neq k, j \neq \ell} c(\mathcal{C}_j(t_n))$. At time t_{n+1} we have:

$$\begin{aligned} J(p(t_{n+1})) &= c(\mathcal{C}_k(t_{n+1})) + c(\mathcal{C}_\ell(t_{n+1})) + \sum_{j \neq k, j \neq \ell} c(\mathcal{C}_j(t_{n+1})) \\ &= c(\mathcal{C}_k(t_n) \setminus \{u_i\}) + c(\mathcal{C}_\ell(t_n) \cup \{u_i\}) + \sum_{j \neq k, j \neq \ell} c(\mathcal{C}_j(t_n)) \\ &= c(\mathcal{C}_k(t_n) \setminus \{u_i\}) + c(\mathcal{C}_\ell(t_n) \cup \{u_i\}) + J(p(t_n)) - c(\mathcal{C}_k(t_n)) - c(\mathcal{C}_\ell(t_n)) \\ &= J(p(t_n)) - \{[c(\mathcal{C}_\ell(t_n)) - c(\mathcal{C}_\ell(t_n) \cup \{u_i\})] - [c(\mathcal{C}_k(t_n) \setminus \{u_i\}) - c(\mathcal{C}_k(t_n))]\} \\ &= J(p(t_n)) - g(\{u_i\}, \mathcal{C}_k(t_n), \mathcal{C}_\ell(t_n)). \end{aligned}$$

■

Result 2.2 *If between time t_n and t_{n+1} , the only swap that happened is the departure of nodes $\{u_i\}$ from cluster \mathcal{C}_k towards cluster \mathcal{C}_ℓ , then the following inequality holds:*

$$J(p(t_{n+1})) < J(p(t_n)).$$

Proof Immediate consequence of Property 2.1 and Result 2.1. ■

2.5.3 Detailed DCOG description

The DCOG algorithm which builds the clusters is described in Table 2.2.

It is split in two steps after which existing clusters satisfy the connectivity, size, and diameter constraints. The first one (lines 1-22) lets each cluster determine if some of its members should leave the cluster and join another neighboring one. During step 2 (lines 22-35), nodes are swapped between clusters as determined in previous step. Step 1 involves nodes of a single cluster, and step 2 involves nodes from a cluster and some of its neighboring clusters. In addition, the clustering algorithm is performed independently by all clusters. This means that the decisions made during step 1 by some nodes $\{u_i\}$ of a cluster \mathcal{C}_k to join a cluster \mathcal{C}_ℓ may no longer make sense because for example step 2 of cluster \mathcal{C}_ℓ has been executed before step 2 of cluster \mathcal{C}_k , and \mathcal{C}_ℓ has been modified. Consequently, in step 2, before modifying clusters it is verified if the decided swaps still make sense. These two steps are repeated until there is no more any possible cluster modification.

During step 1, lines 3-4 ensure that a cluster \mathcal{C}_k will still satisfy connectivity and diameter constraints after some nodes $\{u_i\}$ have joined a neighboring cluster. Lines 6-8 check that if nodes $\{u_i\}$ were to join cluster \mathcal{C}_ℓ , then size, connectivity, and diameter constraints would still be satisfied. Lines 9-16 are used to find the set of neighboring clusters for which the gain achieved when nodes $\{u_i\}$ leave cluster \mathcal{C}_k is maximal. Lines 17-21 check if there are some nodes $\{u_i\}$ that could join a neighboring cluster. If this is the case then one neighboring cluster is selected randomly among the neighboring clusters inducing the maximum gain.

<p>Step 1</p> <p>1 Set $\mathcal{M} = \emptyset$, $\mathcal{L} = \emptyset$ and $g = 0$</p> <p>2 For each set of nodes $\{u_i\}$ members of the same group in \mathcal{C}_k do:</p> <p>3 If $\mathcal{C}_k \setminus \{u_i\}$ is not connected, go to line 2. End If.</p> <p>4 If $d(\mathcal{C}_k \setminus \{u_i\}) > d_{max}$, go to line 2. End If.</p> <p>5 For each cluster \mathcal{C}_ℓ neighbor of cluster \mathcal{C}_k do:</p> <p>6 If $\mathcal{C}_\ell \cup \{u_i\} > n_{max}$, go to line 5. End If.</p> <p>7 If $\mathcal{C}_\ell \cup \{u_i\}$ is disconnected, go to line 5. End If.</p> <p>8 If $d(\mathcal{C}_\ell \cup \{u_i\}) > d_{max}$, go to line 5. End If.</p> <p>9 Calculate $g(\{u_i\}, \mathcal{C}_k, \mathcal{C}_\ell)$ thanks to (2.2).</p> <p>10 If $g(\{u_i\}, \mathcal{C}_k, \mathcal{C}_\ell) > g$ then:</p> <p>11 Set $g = g(\{u_i\}, \mathcal{C}_k, \mathcal{C}_\ell)$.</p> <p>12 Set $\mathcal{L} = \{\mathcal{C}_\ell\}$.</p> <p>13 Else if $g(\{u_i\}, \mathcal{C}_k, \mathcal{C}_\ell) = g$ then:</p> <p>14 Set $\mathcal{L} = \mathcal{L} \cup \mathcal{C}_\ell$.</p> <p>15 End If.</p> <p>16 End For.</p> <p>17 If $g = 0$ then nodes $\{u_i\}$ remain in cluster \mathcal{C}_k.</p> <p>18 Else then</p> <p>19 Choose randomly $\mathcal{C}_\ell \in \mathcal{L}$.</p> <p>20 Set $\mathcal{M} = \mathcal{M} \cup (\{u_i\}, \mathcal{C}_\ell, g)$.</p> <p>21 End If.</p> <p>22 End For.</p>
<p>Step 2</p> <p>23 For each $(\{u_i\}, \mathcal{C}_\ell, g) \in \mathcal{M}$ considered in decreasing g values</p> <p>24 If $\mathcal{C}_k \setminus \{u_i\}$ is not connected, go to line 23. End If.</p> <p>25 If $\mathcal{C}_\ell = \emptyset$, go to line 23. End If.</p> <p>26 If $\mathcal{C}_\ell \cup \{u_i\} > n_{max}$, go to line 23. End If.</p> <p>27 If $d(\mathcal{C}_\ell \cup \{u_i\}) > d_{max}$, go to line 23. End If.</p> <p>28 Calculate $g(\{u_i\}, \mathcal{C}_k, \mathcal{C}_\ell)$ thanks to (2.2).</p> <p>29 If $g(\{u_i\}, \mathcal{C}_k, \mathcal{C}_\ell) \leq 0$, go to line 23. End If.</p> <p>30 If \mathcal{C}_ℓ is available then:</p> <p>31 Set $\mathcal{C}_\ell = \mathcal{C}_\ell \cup \{u_i\}$.</p> <p>32 Set $\mathcal{C}_k = \mathcal{C}_k \setminus \{u_i\}$.</p> <p>33 Exit For loop.</p> <p>34 End If.</p> <p>35 End For.</p>

Table 2.2: Dynamic clustering algorithm applied to cluster \mathcal{C}_k .

During step 2, only at most one set of nodes $\{u_i\}$ may leave cluster \mathcal{C}_k , preferably the one inducing the highest gain. Lines 24-29 check that a swap decided in step 1 is still authorized.

The first condition is that the cluster source of the leaving nodes must remain connected. Also the size and diameter constraints must still be satisfied if the leaving nodes are included in the destination cluster. Finally the gain induced by the swap must still be positive. Line 30 checks that the destination cluster \mathcal{C}_ℓ is ready to accept new nodes, i.e., it is not currently running the clustering algorithm (i.e., running step 1, or step 2 with another cluster). If all those conditions are satisfied, then the source and destination clusters are modified accordingly and line 33 ensures that only one swap will be performed with cluster \mathcal{C}_k as source. Otherwise, clusters \mathcal{C}_k and \mathcal{C}_ℓ are left unchanged.

Note: convenient initial conditions are when all nodes form their own singleton clusters.

Fig. 2.6.a illustrates a swap search by DCOG within a 16 node network organized into four clusters, in which all nodes are in range. The group of each node is identified by its shape. Clusters boundaries are indicated by solid lines. During the execution of DCOG, the cluster \mathcal{C}_1 evaluates the swap gain induced by the two nodes 1 and 2 from the circle group joining each of the neighboring clusters. Among these three possible swaps, it is the one inducing the maximal strictly positive gain which is selected, i.e., the one where the two nodes join \mathcal{C}_2 . The cluster structure resulting from this swap is depicted in Fig. 2.6.b.

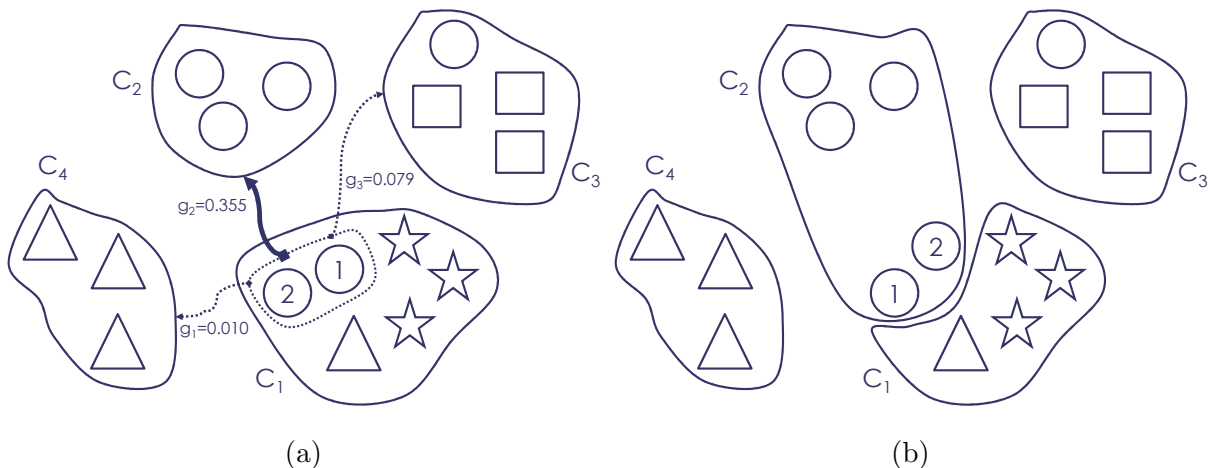


Figure 2.6: (a) Potential swaps and associated gains during DCOG operation in cluster \mathcal{C}_1 for its members $\{1, 2\}$ from the circle group. (b) Swap execution: formation of a cluster with five members from the circle group.

2.5.4 DCOG convergence

Result 2.3 *For a fixed topology, the DCOG algorithm defined in Table 2.2 converges in a finite number of iterations to a stable clustering structure, i.e., when there is no more any possible swap with strictly positive gain.*

Proof Any swap selected in step 1 and executed in step 2 implies a decreasing of the network cost function $J(p(t))$ thanks to Result 2.2. Consequently DCOG cannot choose a partition that has already been selected, which prevents loops. Furthermore since there is a finite number of partitions, the algorithm converges to a stable partition after a finite number of iterations. ■

2.5.5 DCOG cluster cost function

A cost is associated with each cluster. The goal of the clustering algorithm is to build clusters whose costs are minimum. The work of Section 2.3 has allowed to identify that both the number of groups with at least one member in a cluster, and the cluster size, should be used to build good clusters. Therefore, a cost function associated with a cluster must fulfill the following goals (also used in Chapter 3):

Goal 1 (\mathbf{G}_1) *Build clusters whose size is maximal, i.e., equal to n_{max} .*

Goal 2 (\mathbf{G}_2) *Build clusters including the highest number of members from the same group.*

The work of Section 2.4 has identified (2.11) as an example of cost function achieving these two goals. This cost function can be written:

$$c(\mathcal{C}_k) := 1 - \left[f_{n_{max}}(n_k) \cdot \epsilon + \frac{1 - \epsilon}{n_{max}} \sum_{t \in \mathcal{I}(\mathcal{C}_k)} m_{t,k} \cdot f_{m_t}(m_{t,k}) \right], \quad (2.13)$$

with $\epsilon \in [0, 1]$ selected to favor either \mathbf{G}_1 or \mathbf{G}_2 , and function $f_n(m)$ defined as:

$$f_n(m) := \frac{1}{n(n+1)} \cdot m(m+1). \quad (2.14)$$

In Chapter 3 we generalize the expression of (2.14) and prove how ϵ should be set to favor \mathbf{G}_2 over \mathbf{G}_1 . Here we can only state that to favor \mathbf{G}_2 over \mathbf{G}_1 , a small value of ϵ must be selected.

Note that if the size n_k of cluster \mathcal{C}_k is equal to n_{max} , and cluster \mathcal{C}_k is composed of full groups only (i.e., $\forall t \in \mathcal{I}(\mathcal{C}_k), m_{t,k} = m_t$), then $c(\mathcal{C}_k) = 0$.

2.5.6 Adaptation to mobility

Because of node mobility, some clusters may no longer satisfy the connectivity or diameter constraint. In this case, the procedure detailed in Table 2.3 splits those clusters in sets gathering members of the same groups. In Table 2.3, lines 4-7 find connected components and lines 9-15 find subcomponents satisfying the diameter constraint. Finally, lines 16-18 create new clusters as required. Note that line 13 makes sure that if a cluster connectivity or diameter check fails in line 1, it will be split in sets of nodes from the same groups. Consequently, after execution of this procedure, all clusters in the network satisfy the connectivity and diameter constraints.

Referring to line 11, a way to enforce diameter constraint when $d_{max} = 2$ is to split collection of nodes $\{u_i^j\}$ thanks to the following heuristic: *i*) gather the node with the highest degree and all its neighbors, and *ii*) build connected components with remaining nodes. Note that this heuristic does reach its goal only if the algorithm of Table 2.3 is invoked often enough w.r.t. the node mobility.

2.6 Numerical results

2.6.1 Performance metrics

To assess DCOG performance, the following metrics have been selected:

- The time needed to reach a stable clustered network, measured in number of time units.

1	If C_k is not connected or $d(C_k) > d_{max}$ then:
2	Let $\mathcal{M} = \emptyset$.
3	For each set of nodes $\{u_i\}$ members of the same group in C_k do:
4	Let $\mathcal{C} = \{\{u_i\}\}$.
5	If $\{u_i\}$ is not connected then:
6	Split $\{u_i\}$ into its m connected components $\{\{u_i^1\}, \dots, \{u_i^m\}\}$.
7	Set $\mathcal{C} = \{\{u_i^1\}, \dots, \{u_i^m\}\}$.
8	End If.
9	For each set of nodes $\{u_i^j\}$ in \mathcal{C} do:
10	If $d(\{u_i^j\}) > d_{max}$ then:
11	Split $\{u_i^j\}$ into its m' subcomponents $\{\{v_i^1\}, \dots, \{v_i^{m'}\}\}$ each one satisfying the diameter constraint.
12	Set $\mathcal{M} = \mathcal{M} \cup \{\{v_i^1\}, \dots, \{v_i^{m'}\}\}$.
13	Else set $\mathcal{M} = \mathcal{M} \cup \{u_i^j\}$.
14	End If.
15	End For.
16	For each set of nodes $\{v_i\}$ in \mathcal{M} do:
17	Create a new cluster with $\{v_i\}$.
18	End For.
19	End If.

Table 2.3: Adaptation to node mobility for cluster C_k .

- The cluster size. The target cluster size is n_{max} .
- The cluster group diversity (CGD), i.e., the average number of groups per cluster with at least one member within the cluster. Ideally all members of a group should be in the same cluster and this number should not be larger than the cluster size divided by the group size.
- The group cluster diversity (GCD), i.e., the average number of clusters per group including at least one member of the group. This metric should have a low value, meaning that the members of a group tend to be in the same cluster.
- The application level performance measured using J_0^X as defined in Section 1.4.3. Here its values are comparable to the average end-to-end delays from all nodes to all nodes. This means that the lower the J_0^X value, the better. Two salient features of this metric are that *i*) it uses two different costs for inter-cluster ($\tilde{\gamma}$) and intra-cluster ($\hat{\gamma}$) communications; and *ii*) it can be used to concentrate traffic within groups thanks to its α parameter. The J_0^X parameter values are: $\hat{\gamma} = 1$, $\tilde{\gamma} = 2$ and $\alpha = 0.9$. This means that the inter-cluster communication cost is twice the one of intra-cluster communication, and that 90% of the traffic is exchanged between members of the same groups.
- The application level performance measured using T_0^X as defined in Section 1.7. Its values are comparable to average bandwidth on the widest path from all nodes to all nodes. Thus the larger the T_0^X value, the better. The T_0^X parameters are: $\hat{\eta} = 1$, $\tilde{\eta} = 0.5$ and $\alpha = 0.9$.

Except in Section 2.6.7.2, all simulation results provided in this section are average values over 100 different random networks.

2.6.2 Reference clustering algorithms

We have selected as reference the two distributed clustering protocols GDMAC [8] and VOTE [9] because they allow to adapt the number of clusters to the network density. We have also defined extensions of these protocols in order to take into account the group structure of the network.

2.6.2.1 GDMAC

The goal of the distributed clustering protocol GDMAC is to build stable clusters in presence of node mobility. In this protocol CH nodes are elected first, and then non-CH nodes affiliate to a neighbor CH, leading to the clusters. To do that, a weight (depending on the context) is associated with each node. Within a radio neighborhood, nodes with the highest weights are chosen as CH nodes. To increase stability in presence of mobility GDMAC introduces the K parameter, whose value is equal to the number of CH nodes that are allowed to be neighbors of a $(K + 1)$ th CH node. A non-CH node affiliates to the CH node within its neighborhood whose weight is the highest. In order to obtain stable clusters, a non-CH node remains affiliated to its current CH unless there is a CH node in its neighborhood whose weight exceeds the one of the current CH node by at least a positive lower-bound denoted by H . The GDMAC algorithm is defined in Table 2.4. Note that after execution of lines 1-17, some nodes may have decided to join the cluster for a node which itself has made the same kind of decision. Consequently the former nodes will believe to be in a cluster that does not exist in reality. This is handled by these nodes the next time they perform the GDMAC algorithm.

In the GDMAC paper [8], the weight is allocated randomly. In this work, we prefer to use the node identifier as node weight, like in [1]. We refer to this approach as GDMAC-std. To extend GDMAC so as to take group membership into account, we propose to calculate the node weights in a different way, and also to modify the way non-CH nodes affiliate to their CH. This leads to two new versions of the GDMAC, denoted by GDMAC-new1 and GDMAC-new2.

GDMAC-new1: the weight used is the stability factor defined in [15]. In that original paper the stability factor has been introduced for clustering structured networks but is associated with a very simple algorithm. Here we thus propose to associate this stability factor with GDMAC. The stability factor of a node is a linear combination of the average relative speed with its neighbors, the average distance with its neighbors, the average number of neighbors, and its remaining energy. The three mentioned averages are weighted averages. In order to take into account the group structure, lower weights in the average are used for neighbors that are members of the same group as the one of the current node. The TCA protocol from [15] can be seen as a particular case of GDMAC-new1 by setting the GDMAC parameter K to zero. We call it TCA-std.

GDMAC-new2: it is an extension of GDMAC-new1 where we modify the non-CH node affiliation strategy. Indeed, a non-CH node affiliates, if possible, to a CH node that is also member of its group instead of choosing it with respect to its weight.

According to the proposed modifications, GDMAC-new1 and GDMAC-new2 are expected to be better suited to the context of structured networks.

```

1 If local node is CH then:
2   Find list of neighbor CH with larger weight than node  $i$ .
3   If the number of such neighbors is strictly greater than  $K$ 
   then:
4     Local node chooses one of the  $K$  neighbor CH with the
     highest weights as its new CH.
5   Else local node remains CH.
6   End If.
7 Else:
8   If local node CH is no longer a neighbor or is no longer a
   CH then:
9     Find the list of local node's neighbor CH.
10  Else:
11  Find the list of local node's neighbor CH whose weight
    exceeds by at least  $H$  the weight of local node CH.
12  End If.
13  If this list is not empty then:
14  Local node chooses randomly as CH among the  $K+1$  ones
    with the highest weight.
15  Else:
16  If current CH is no longer a neighbor CH then:
17  Local node becomes CH.
18  End If.
19  If current CH is still a neighbor CH then:
20  Local node keeps its current CH as CH.
21  End If.
22  End If.
23 End If.

```

Table 2.4: GDMAC algorithm.

2.6.2.2 VOTE

Similarly to GDMAC, VOTE [9] selects some CH nodes based on their weight, and then non-CH nodes affiliate to a neighbor CH node. In VOTE, the weight of each node is called its *vote* and is a linear combination of the normalized degree and the battery remaining time. The main difference between GDMAC and VOTE does not lay in the weight definition but in the way each CH node manages its cluster size: VOTE limits the cluster size to n_{max} and GDMAC does not handle it. When the number of nodes affiliated to a CH node is equal to $n_{max} - 1$, non-CH nodes refrain to affiliate to this CH. Nevertheless, in the case of simultaneous affiliations a cluster may include more than n_{max} members. Then, the concerned CH randomly rejects as many members as required to satisfy the cluster size constraint. The VOTE algorithm is defined in Table 2.5.

In order to take into account the group structure we propose here to apply the algorithm VOTE (denoted by VOTE-new) by using the stability factor as the weight. Initial VOTE defined in [9] is hereafter denoted as VOTE-std.

```

1 If local node is CH then:
2   Find list of neighbor CH with larger weight than node  $i$ ,
   whose cluster size is lesser than the maximum size.
3   If this list is non empty then:
4     Local node chooses as its CH its neighbor CH with the
     highest weight.
5   Else local node remains CH.
6   End If.
7 Else:
8   If local node CH is no longer a neighbor, or is no longer a
   CH, or local node has been excluded from the cluster of its
   previous CH then:
9     Find the list of local node's neighbor CH whose cluster
     size is lesser than the maximum size.
10  Else:
11  Find the list of local node's neighbor CH whose weight is
     strictly greater than the one of local node CH, and whose
     cluster size is lesser than the maximum size.
12  End If.
13  If this list is not empty then:
14    Local node chooses as its CH its neighbor CH with the
     highest weight.
15  Else:
16    If current CH is no longer a neighbor CH then:
17      Local node becomes CH.
18    End If.
19    If current CH is still a neighbor CH then:
20      Local node keeps its current CH as CH.
21    End If.
22  End If.
23 End If.

```

Table 2.5: VOTE cluster-head selection algorithm.

Note: like for GDMAC-new2, we defined VOTE-new2 which was intended to improve VOTE-new such as a non CH node affiliates preferentially to a CH that is also member of its group. However, the convergence of VOTE-new2 was not ensured because of bad interactions between this affiliation scheme and the cluster size maintenance process. Thus we dismissed VOTE-new2.

2.6.3 Simulation setup

In this section we describe the general simulation setup.

2.6.3.1 Network setup

The simulated network has N nodes. The group size is constant: $\forall t, m_t = 10$. The communication range of a node is $d_{ref} = 250$ distance units. The nodes are deployed in a square area whose side is 1500 distance units long, following a modified RPGM model [42], different from the one in Section 1.5.2.2 because it does not handle mobility. At the beginning of the simulation a randomly located virtual center $A_t(0)$ is associated with each group t , and all group t members are deployed randomly in a disk of radius d_{ref} centered at $A_t(0)$. In mobile networks, time is split in intervals of fixed duration Λ seconds. At time $k \cdot \Lambda$ (i.e., the beginning of interval number $k \in \mathbb{N}$), the virtual center $A_t(k+1)$ of group t is randomly located, and the coordinates of group t members at time $(k+1) \cdot \Lambda$ are randomly chosen in the disk of radius d_{ref} centered at $A_t(k+1)$. Then during time interval $[k \cdot \Lambda, (k+1) \cdot \Lambda]$, each node i follows a uniform rectilinear motion with a speed limited to a maximum v_{max} between its coordinates at time $k \cdot \Lambda$ and time $(k+1) \cdot \Lambda$. Note: when new coordinates are selected for a virtual center $A_t(k)$, if the distance between $A_t(k)$ is smaller than d_{ref} from the deployment area boundary, then a new location is drawn to make sure that no group member is placed outside of the deployment area.

Fig. 2.7 depicts an example of random network with $N = 100$ nodes, such that nodes in the same group share the same color.

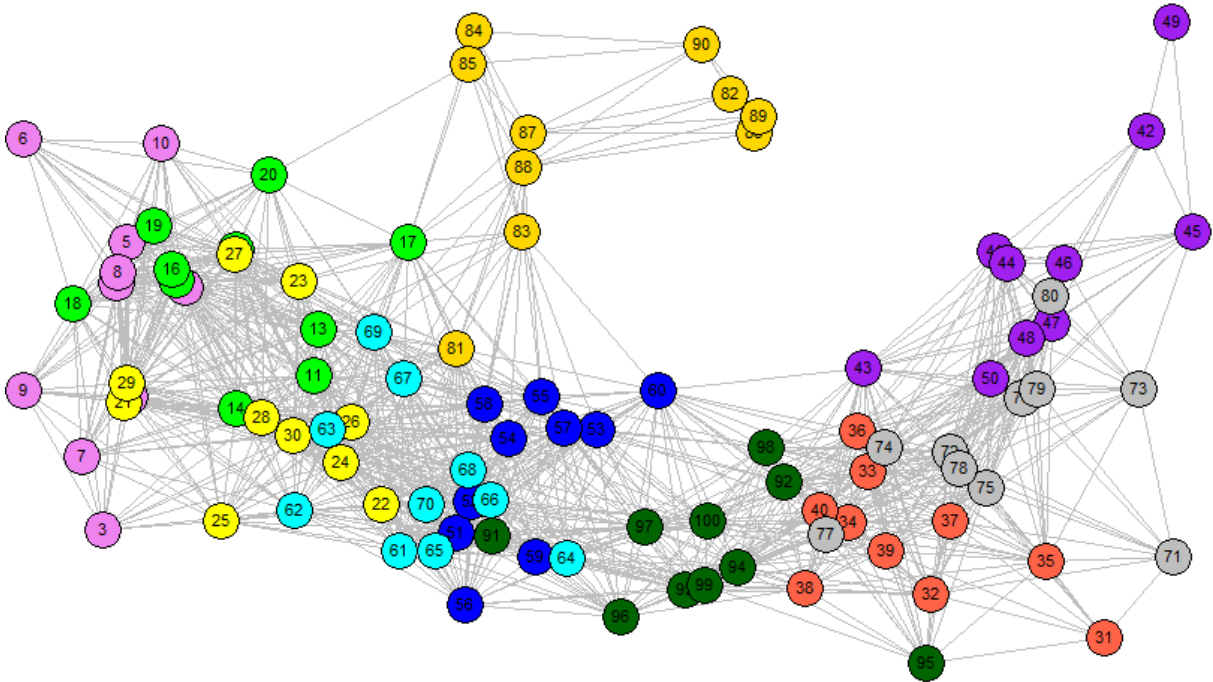


Figure 2.7: Example of random structured network with $N = 100$ nodes.

2.6.3.2 Clustering schemes setup

The clustering algorithms parameters common to all simulations are now detailed. We consider $H = 10$ for GDMAC-std, and $H = 30$ for GDMAC-new1 and GDMAC-new2. Due to the difficulty to set its value, the values of parameter K are selected in Section 2.6.3.3. The stability factor of a node is calculated using weighted linear combination of the average relative speed and distance between this node and its neighbors (with equal weights 0.5), averaged on five samples.

In these average values, group membership is taken into account by allocating a weight (denoted by λ in [15]) equal to 0.5 to nodes in the same group as the local node, and equal to 1 to other nodes. For both VOTE and DCOG, $n_{max} = 20$. Finally, the value of DCOG utility function (2.13) ϵ parameter is set to a value small enough to make sure that whatever the cluster size, increasing the number of members of a group in a cluster always takes precedence over increasing its size.

Fig. 2.8 shows an example of the network from Fig. 2.7 clustered with DCOG. In this figure the nodes in the same cluster share the same outer circle color.

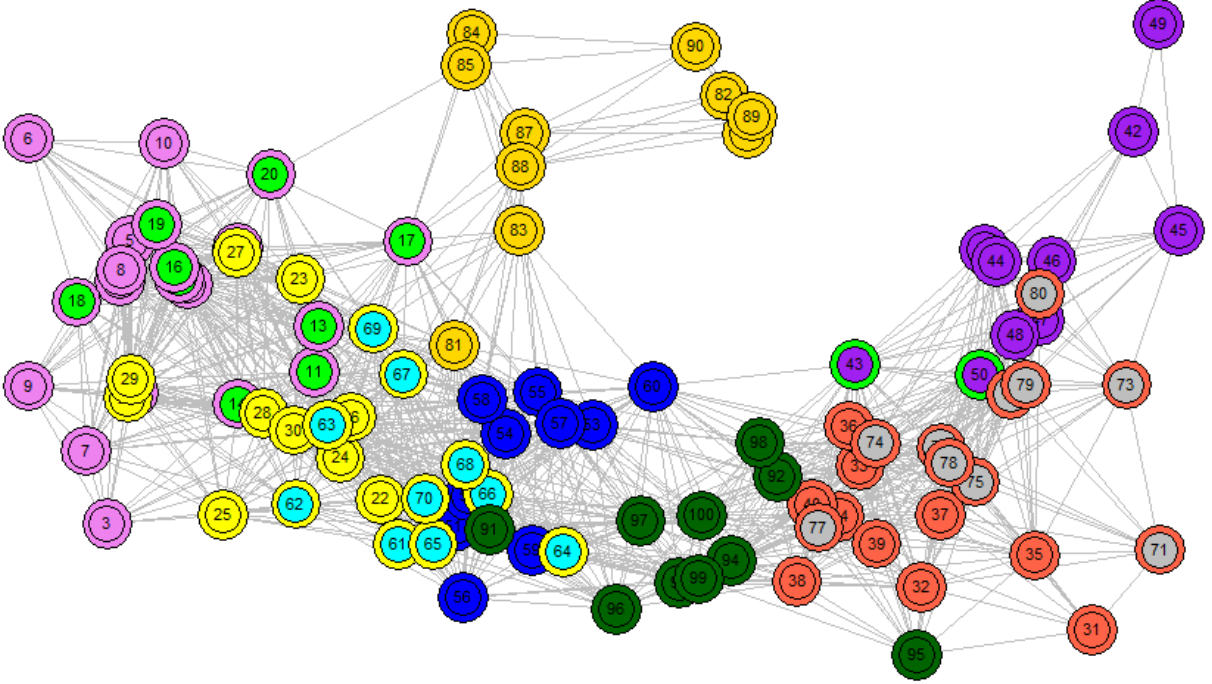


Figure 2.8: Example of DCOG action on a random structured network with $N = 100$ nodes.

2.6.3.3 GDMAC K parameter values

Whereas DCOG and VOTE have a maximum cluster size parameter, GDMAC does not and leads to clusters with large size differences: in the same geographical area very small clusters coexist with very large clusters. Therefore to ensure a meaningful comparison with VOTE and DCOG, K is set to reduce cluster size variability. For each GDMAC extension and each network size, we have determined through simulations the best value of K as indicated in Table 2.6. The selected value is the lowest one leading to an average highest cluster size no greater than n_{max} . GDMAC-std succeeds in filling this criterion only for 100 and 200 nodes. Consequently static networks GDMAC-std simulation results are not provided in Section 2.6.6.

2.6.4 Asynchronous and synchronous modeling for DCOG

To simulate DCOG we defined two models: *i*) a fully distributed model with no synchronization between the clusters, and *ii*) a simpler distributed synchronous model. Concerning metrics unrelated to time, we show that both models lead to the same performance. This allows us to perform subsequent simulations using the synchronous model.

Nodes	100	200	300	400	500
GDMAC-std	2	5	-	-	-
GDMAC-new1	2	3	5	7	9
GDMAC-new2	1	2	4	6	8
Nodes	600	700	800	900	1000
GDMAC-new1	11	13	17	20	22
GDMAC-new2	11	13	17	19	23

Table 2.6: Values of K parameter.

2.6.4.1 Distributed asynchronous model

Let us first define our fully distributed asynchronous model. In this model a cluster can be in one of the following six states:

- *Deciding*: when step 1 of DCOG is being performed (see Table 2.2) and a decision is made to select of set of nodes that will leave the cluster to join a neighboring one. The duration of this state is δ_d .
- *Leaving*: when step 2 of DCOG is being performed and the set of nodes selected in *Deciding* state potentially leave their cluster to join the chosen neighboring one. The duration of this state is δ_l .
- *Checking*: when the cluster checks if connectivity, diameter and size constraints are still satisfied. The duration of this state is δ_c .
- *Adapting*: when, because of node mobility the constraints are no longer satisfied and adaptation to mobility must be performed . The duration of this state is δ_a .
- *Waiting*: when the cluster is idle, ready to accept new nodes. The duration of this state is δ_w , exponentially distributed with parameter λ .
- *Joining*: when some nodes are joining the cluster. The duration of this state is δ_j .

The asynchronous DCOG state machine is described in Fig. 2.9. In this figure, lines illustrate the state transitions, and the text associated with each transition details the duration required for this transition. In this model, all the clusters are always supposed to know the internal information about their neighbor clusters.

To better understand this state diagram, the Fig. 2.10 to 2.12 detail the different state transitions using time diagrams. The Fig. 2.10 details the transition from *Deciding* to *Waiting* state.

The Fig. 2.11 details the two cases when a cluster \mathcal{C}_k leaves the *Deciding* state and some of its nodes want to join a neighboring cluster \mathcal{C}_ℓ . Firstly (① in Fig. 2.11), if cluster \mathcal{C}_ℓ state is *Waiting* when cluster \mathcal{C}_k enters the *Leaving* state, then cluster \mathcal{C}_ℓ state becomes *Joining* and the selected nodes leave \mathcal{C}_k and join \mathcal{C}_ℓ . Note that in that case, both clusters \mathcal{C}_k and \mathcal{C}_ℓ remain in their respective state *Leaving* and *Joining* during the same duration δ_l . Secondly (② in Fig. 2.11), if cluster \mathcal{C}_ℓ state is not *Waiting*, then cluster \mathcal{C}_k state becomes *Leaving* but the selected nodes remain in \mathcal{C}_k .

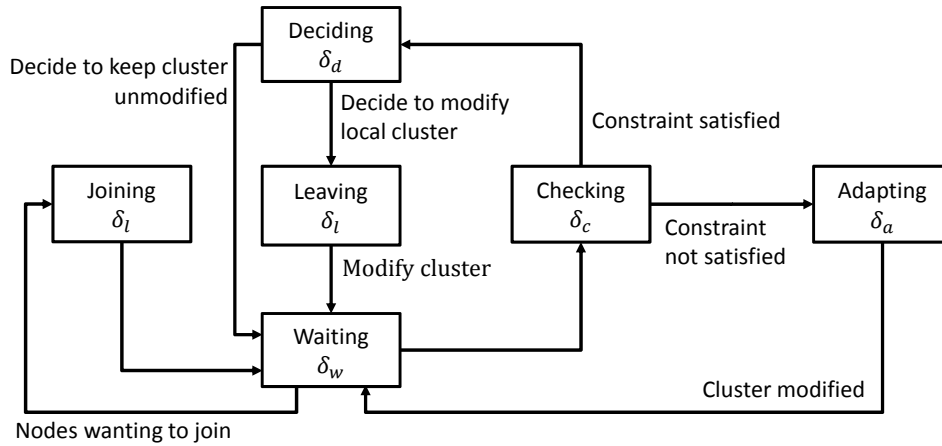


Figure 2.9: Asynchronous DCOG state machine.

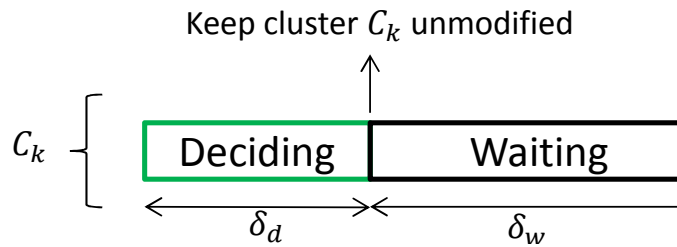


Figure 2.10: Asynchronous DCOG: transition from *Deciding* to *Waiting* state.

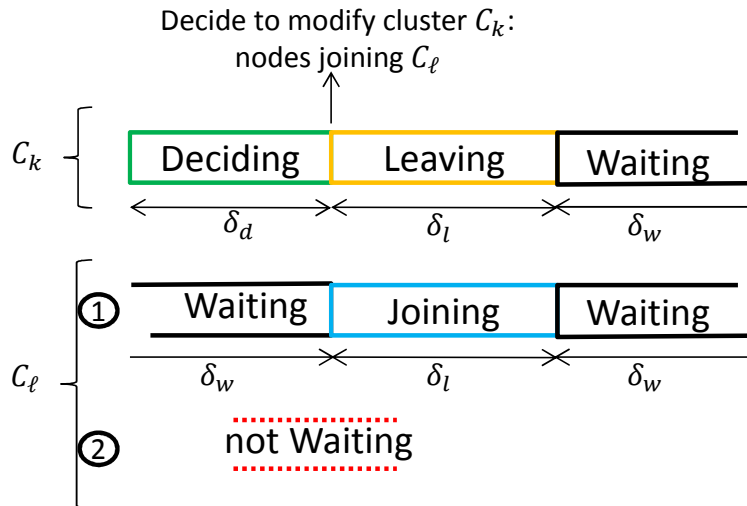


Figure 2.11: Asynchronous DCOG: cases when a cluster leaves *Deciding* state and some of its nodes want to join a neighboring cluster.

Fig. 2.12 details what happens each time a cluster leaves the *Waiting* state to check if the constraints are still satisfied. In the positive case (① in Fig. 2.12), then the cluster state becomes *Deciding*, otherwise (② in Fig. 2.12), *Adapting*.

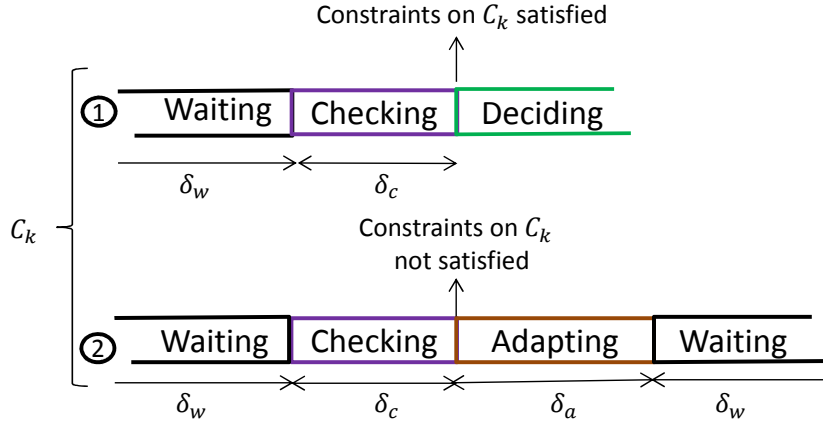


Figure 2.12: Asynchronous DCOG: cases when the cluster constraints are checked.

This asynchronous model is used to verify the convergence of DCOG through simulation in a fully distributed setting. This model requires several parameters (δ_d , δ_l , δ_c , δ_a and λ). Consequently, as soon as DCOG convergence has been verified then we use a simpler and faster (from the point of view of simulation time) synchronous model to measure its performance.

2.6.4.2 Distributed synchronous model

In the synchronous model, time is split in rounds. At the beginning of a round, all clusters are supposed to know the internal information about their neighbor clusters. During a round all clusters perform the following actions:

- Apply sequentially to all clusters the algorithm of Table 2.3 to adapt to mobility.
- Apply sequentially step 1 of Table 2.2 to all clusters not modified in this round due to mobility.
- Apply sequentially step 2 of Table 2.2 to all clusters not modified in this round due to mobility.

2.6.5 Asynchronous DCOG simulation results in static networks

Simulations of asynchronous DCOG have been performed with $N = 100$ static nodes. Due to the lack of mobility, when a node enters the *Checking* state, the constraints are always satisfied, leading to the *Deciding* state.

Firstly, the values of asynchronous DCOG parameters δ_d , δ_l and δ_a have been set to 2 time units, and δ_c to 1 time unit. Table 2.7 provides the results of the simulation of 100 random networks for each value of the λ parameter selected in $\{0.5, 1, 2, 5, 10, 20, 30, 40, 50\}$. The last line whose cell with λ value equal to "sync" provides the results when DCOG is simulated using its synchronous model.

The main result from Table 2.7 is that all metrics are nearly independent of the value of λ . Moreover, the values from asynchronous and synchronous models are similar. The largest difference concerns the cluster size which is always smaller than 3.5%. Concerning the application layer metrics J_0^X and T_0^X , they are within 0.3% of the values obtained with the synchronous model.

With the asynchronous model, the clusters are considered stable when at least 100 time units have elapsed since at least one cluster entered the *Leaving* state. The time required to converge

λ	Cluster size			CGD	GCD	J_0^X	T_0^X
	avg	std dev	max				
0.5	11.42	4.00	18.73	1.27	1.12	1.61	5.49
1	11.34	4.12	18.92	1.26	1.12	1.61	5.50
2	11.27	4.14	18.84	1.25	1.12	1.61	5.48
5	11.28	4.12	18.88	1.25	1.11	1.61	5.49
10	11.34	4.18	19.04	1.26	1.12	1.61	5.50
20	11.38	4.16	19.04	1.26	1.12	1.61	5.51
30	11.37	4.13	18.81	1.26	1.12	1.61	5.50
40	11.32	4.12	18.97	1.26	1.12	1.61	5.49
50	11.35	4.15	18.85	1.28	1.13	1.61	5.49
sync	11.04	4.12	18.87	1.24	1.12	1.61	5.49

Table 2.7: Asynchronous DCOG simulation results, with $\delta_d = \delta_l = \delta_a = 2$ and $\delta_c = 1$.

to a stable cluster structure is plotted in Fig. 2.13. For example, in this figure, when $\lambda = 0.5$ the average convergence time is 300.84 time units, meaning that at event time greater or equal than 400.84, no cluster had entered state *Leaving* for 100 time units. With the synchronous model, the convergence time is in rounds, the clusters are considered stable when no modification has been performed during two consecutive rounds.

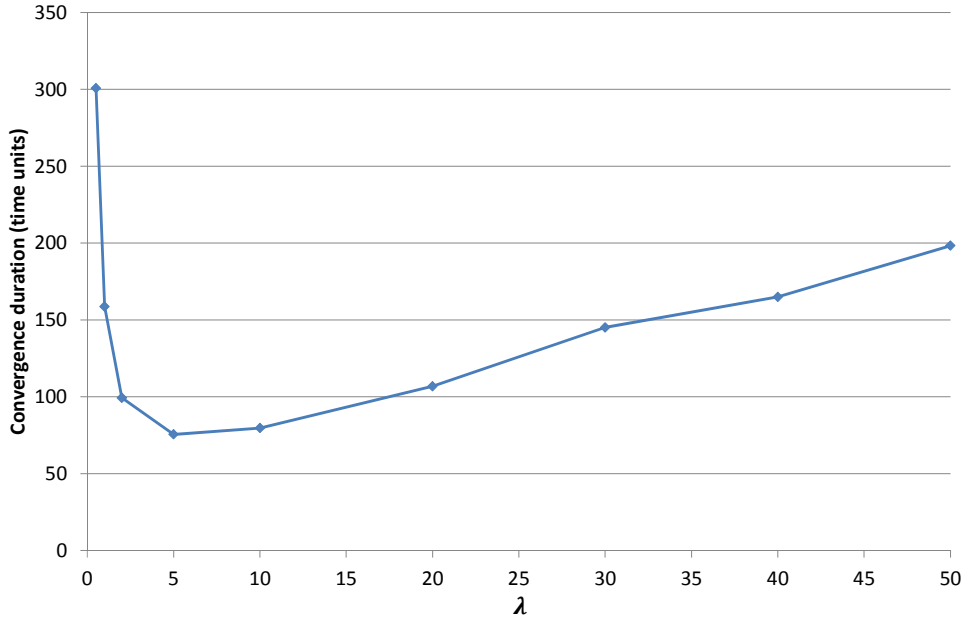


Figure 2.13: Asynchronous DCOG convergence time vs. λ , with $\delta_d = \delta_l = \delta_a = 2$ and $\delta_c = 1$.

The slope of the curve in Fig. 2.13 can be explained as follows:

- When $\lambda < 2$, the *Waiting* state duration is short. Therefore, when some nodes from a cluster \mathcal{C}_k want to join a cluster \mathcal{C}_ℓ , it is usually in a state different from *Waiting*, thus canceling the inter-cluster swap. This explains why the convergence duration decreases when λ increases.
- Conversely, when $\lambda \geq 2$, the clusters are usually in the *Waiting* state, thus allowing inter-

cluster swaps. As λ value increases, the time in the *Waiting* state also increases and clusters need more time to enter the *Deciding* state to build the clusters, thus increasing the time required to reach stable clusters.

To assess the influence of the simulation parameters, simulations have been performed with all parameters values multiplied by 3: $\delta_d = \delta_l = \delta_a = 6$, $\delta_c = 3$ and $\lambda \in \{1.5, 3, 6, 15, 30, 60, 90, 120, 150\}$. Simulation results are provided in Table 2.8. Again, all metrics are nearly independent of the value of λ . The maximum difference with the value achieved using synchronous model is 3.4% for average cluster size, 3.3% for CGD and 4.5% for GCD. The differences for application level metrics J_0^X and T_0^X are less than 0.5%.

λ	Cluster size			CGD	GCD	J_0^X	T_0^X
	avg	std dev	max				
1.5	11.42	4.00	18.73	1.27	1.12	1.61	5.49
3	11.34	4.12	18.92	1.26	1.12	1.61	5.50
6	11.27	4.14	18.84	1.25	1.12	1.61	5.48
15	11.28	4.12	18.88	1.25	1.11	1.61	5.49
30	11.34	4.20	19.04	1.26	1.12	1.61	5.50
60	11.34	4.17	19.01	1.27	1.13	1.61	5.50
90	11.31	4.13	18.79	1.29	1.14	1.61	5.49
120	11.18	4.14	18.80	1.28	1.15	1.61	5.47
150	11.10	4.13	18.70	1.28	1.17	1.62	5.47
sync	11.04	4.12	18.87	1.24	1.12	1.61	5.49

Table 2.8: Asynchronous DCOG simulation results, with $\delta_d = \delta_l = \delta_a = 6$ and $\delta_c = 3$.

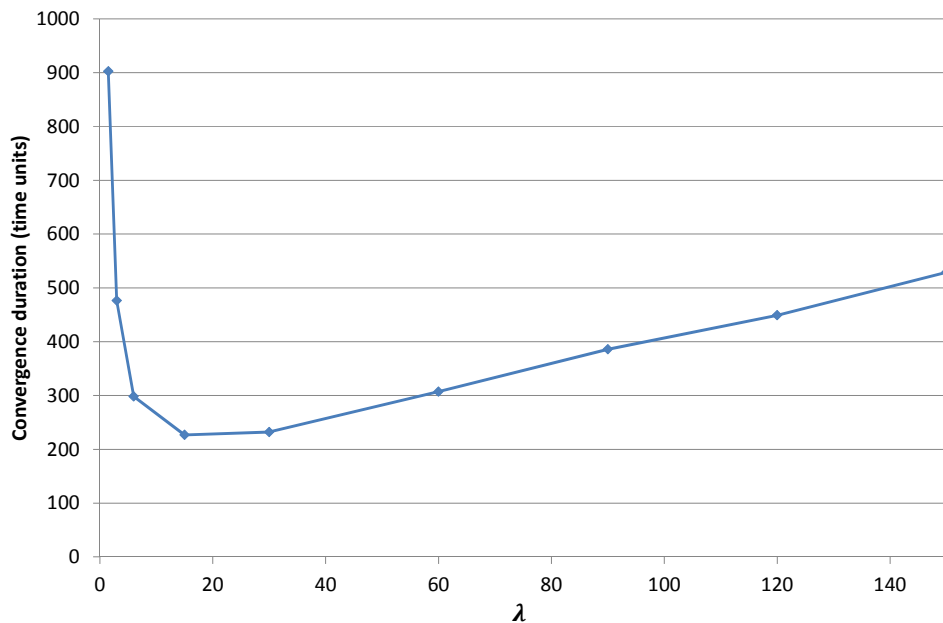


Figure 2.14: Asynchronous DCOG convergence time vs. λ , with $\delta_d = \delta_l = \delta_a = 6$ and $\delta_c = 3$.

Convergence time is plotted in Fig. 2.14. Comparing Fig. 2.14 and Fig. 2.13 reveals that

both curves have similar shape, which was expected. It can be noted that there is an optimal value of λ for which the convergence time is the shortest. This optimal value could be used to design a radio access scheme ensuring the fastest convergence of the clustering algorithm.

The proof of DCOG convergence is independent of the considered model (asynchronous or synchronous). Additionally, the results of this section show that there are only small differences between the results of asynchronous DCOG and synchronous DCOG simulations. Consequently in the following, in order to speed them up, the simulations are performed using the synchronous DCOG model described Section 2.6.4.2.

Let us now describe the synchronous model used to simulate the reference clustering algorithms of Section 2.6.2. Initially all nodes are CH and nodes learn about their neighbors. Then every round, the following actions are performed: *i*) update each node state (CH or non-CH, selected CH) using the appropriate algorithm, and *ii*) for each node update the knowledge of its neighbors' state.

2.6.6 Performance in static networks

This section is devoted to the analysis of the static networks simulation results.

2.6.6.1 Node average degree

Simulations have been performed for an increasing number of nodes. The size of the deployment area is constant, therefore the average node degree increases with N . The Fig. 2.15 shows the node average degree as well as the 5th and 95th percentiles when N increases from 100 to 1000. The average number of 1-hop neighbors increases linearly from a moderate density of about 19 to a high density of about 145.

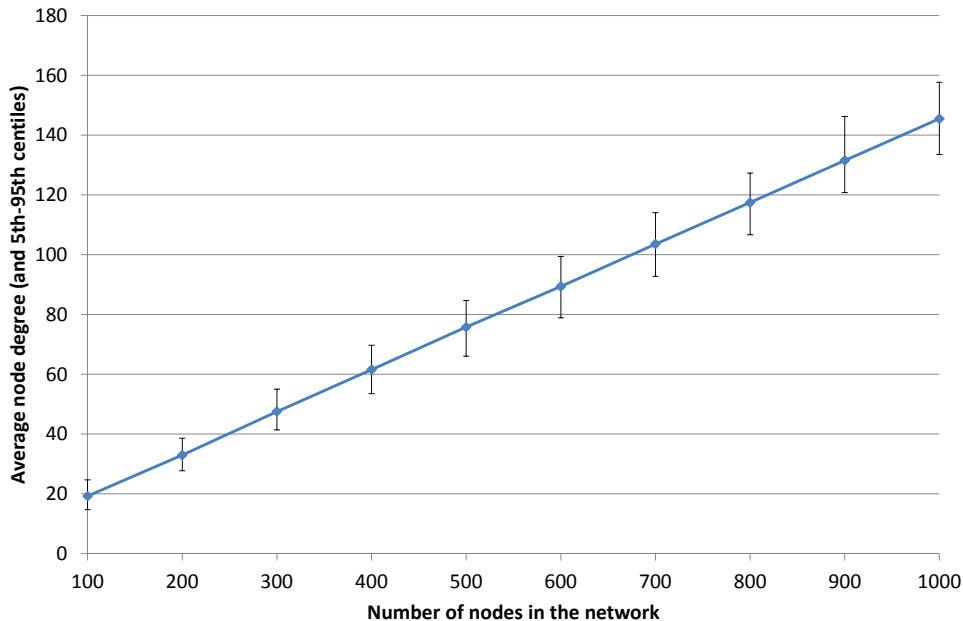


Figure 2.15: Average degree and 5th-95th percentiles in static networks vs. N .

It is interesting to note that when N increases, the ratio of 5th-95th interval width to average degree decreases, as shown in Table 2.9. This means that when N increases, two different

N	100	200	300	400	500	600	700	800	900	1000
Ratio	0.52	0.33	0.29	0.26	0.25	0.23	0.21	0.18	0.19	0.17
Diameter	7.51	7.25	7.23	7.23	7.20	7.22	7.25	7.26	7.25	7.26

Table 2.9: Ratio of 5th-95th interval width to average node degree and network diameter vs. N .

random networks become more and more similar. This table also shows that the average network diameter is nearly independent of the network size.

2.6.6.2 Convergence

The number of rounds needed to reach a stable cluster structure depending on the number of nodes in the network is illustrated in Fig. 2.16.

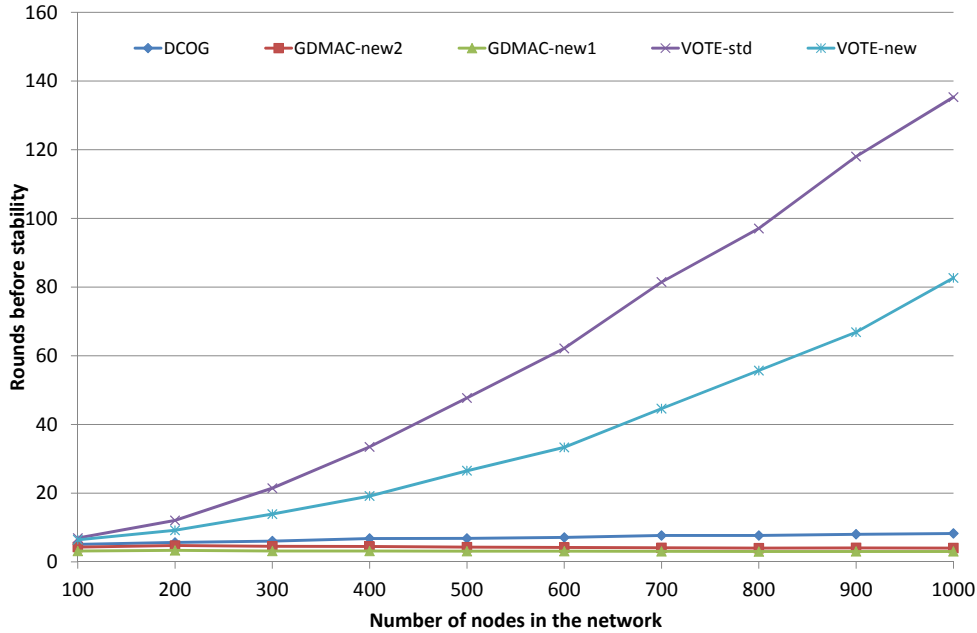


Figure 2.16: Convergence duration in static networks vs. the number of nodes.

The three algorithms GDMAC-new1, GDMAC-new2 and DCOG converge much faster than VOTE-std and VOTE-new. The two VOTE-based algorithms need an increasing amount of time when the number of nodes increases. This happens because when too many nodes are members of a cluster, the CH rejects as many nodes (chosen randomly) as required to reduce this number to the limit n_{max} . Those nodes then simultaneously join neighboring clusters, which usually cause these clusters to exceed their maximum size, causing additional nodes being excluded from these clusters (the highest duration before convergence required by VOTE-std in a 100 node network is 222 rounds).

Fig. 2.17 zooms on GDMAC-new1, GDMAC-new2 and DCOG. GDMAC-new1 needs about 3 rounds to converge, when GDMAC-new2 needs 4 or 5 rounds. The additional time needed by GDMAC-new2 is due to its affiliation procedure slightly more complex than the one of GDMAC-new. Compared to VOTE-std and VOTE-new the GDMAC variants do not manage the cluster size and do not suffer from the problem associated with the exclusion of nodes from

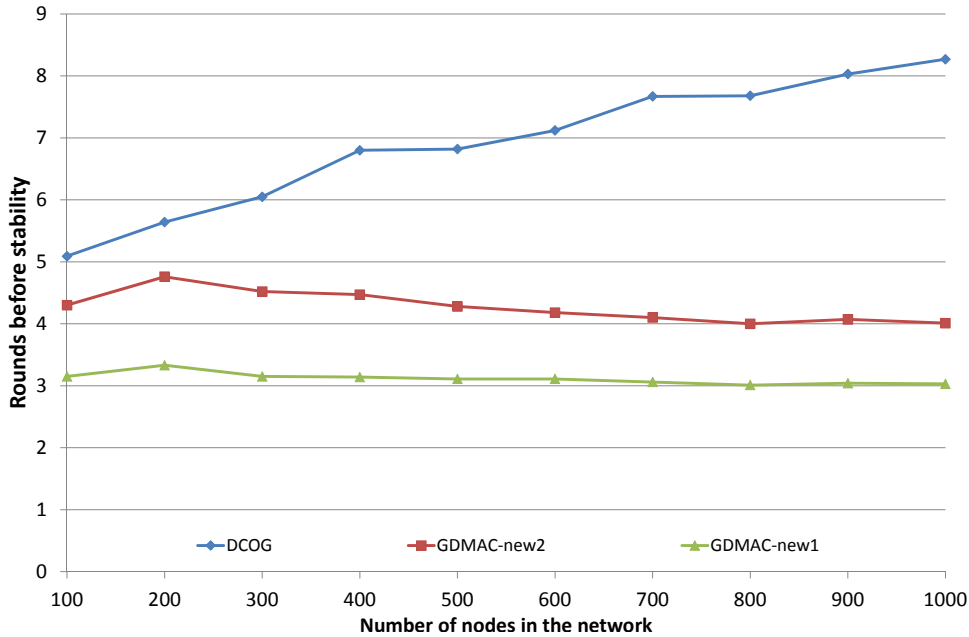


Figure 2.17: Convergence duration (DCOG, GDMAC-new1 and GDMAC-new2) in static networks with 100 to 1000 nodes.

too large clusters. The time needed by DCOG to converge is slightly larger than for the GDMAC variants, and increases with the number of nodes from about 5 to 8 rounds. This is justified by the greater complexity of DCOG, which is mitigated by its capacity to implement node cluster swaps involving multiple nodes.

2.6.6.3 Cluster size

Fig. 2.18 plots the average number of members in a cluster with respect to the number of nodes N in the network.

Because TCA-std only allows one CH node in a neighborhood, the cluster size is expected to increase with the number of nodes. This is exactly what can be seen in Fig. 2.18. This behavior is undesired because in clusters whose maximum size is controlled, it is not possible to ensure efficient radio resource allocation. Consequently, even if TCA-std is the only reference [15] from the state of the art taking into account group membership¹, we no longer consider it in the remainder of our work. Thanks to the appropriate choice of GDMAC K parameter, GDMAC and VOTE-based algorithms as well as DCOG yield clusters whose maximum size is almost the same. However, their average cluster size are substantially different.

Both GDMAC-new1 and GDMAC-new2 limit the number of neighbor CH to $K + 1$, therefore their average cluster sizes are similar. However, their average clusters size are quasi-independent of the network size, and are much smaller than n_{max} . This means that these protocols create too many clusters. This is the drawback of using the parameter K to limit the cluster size, which has not been introduced by the authors of GDMAC for that purpose. A more detailed analysis shows that GDMAC-new2 builds more balanced clusters than GDMAC-new1. This is due to their different affiliation procedure. With GDMAC-new1, non-CH nodes tend to affiliate

¹We did not succeed in implementing [16].

to their neighbor CH with the highest weight leading to large clusters. Conversely, only a few nodes affiliate to the CH with lower weights, leading to small clusters. GDMAC-new2 ensures that non-CH nodes affiliate to a neighbor CH of the same group, if any. Thanks to this rule, if only one CH of a given group exists in a radio neighborhood, this CH has at least as many cluster members as the number of its neighbors belonging to its group.

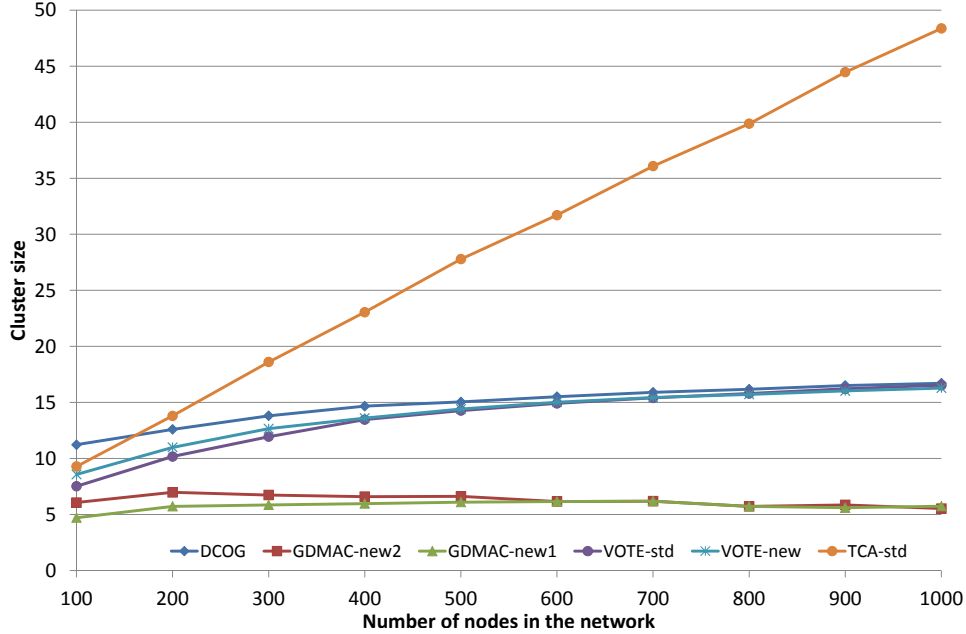


Figure 2.18: Average cluster size in static networks vs. the number of nodes.

VOTE-std and VOTE-new determine the number of CH dynamically, and build clusters whose size is limited by n_{max} . This leads to clusters of similar size, increasing with the number of nodes in the network. Nevertheless the number of small clusters is still large. Even if VOTE-std and VOTE-new build clusters with larger average size than GDMAC-new1 and GDMAC-new2, the cluster size variability is still large.

Finally concerning DCOG, the cluster size starts at 11.2 for 100 nodes, and increases with the number of nodes in the network to reach 16.7 for 1000 nodes. A more detailed analysis shows that DCOG builds clusters whose size are usually around 10 or around 20. Also, the number of maximum size clusters increases with the number of nodes. To understand this behavior, the CGD and GCD must be analyzed, which is done in the next section of this document.

2.6.6.4 Cluster group and group cluster diversities

Following the definitions in Section 2.6.1, the GCD and CGD have been evaluated and plotted vs. the number of nodes in Fig. 2.19 and Fig. 2.20 respectively.

First let us analyze the results for the three algorithms GDMAC-new1, VOTE-std and VOTE-new which lead to the highest (i.e. worst) values. None of these algorithms take the group membership into account to choose the CH they affiliate to, leading to a large numbers of groups per cluster. As expected, the values also increase with the number of nodes in the network.

- GDMAC-new1: the CGD increases from 2 to 5 and the GCD increases from 4 to nearly

8.5. It is interesting to note that the larger the number of nodes, the closer the CGD is to the cluster size itself. This means that on average, each cluster includes only one member of each operational group. Also, when the number of nodes increases, the number of clusters also increases which raises the probability that the members of each group are split among different clusters. This is confirmed by larger GCD values.

- VOTE-std and VOTE-new: a similar analysis as the one performed for GDMAC-new1 can be done. The difference is that they both build larger clusters. Consequently they lead to a larger average CGD value, but a smaller GCD one. This is confirmed by Fig. 2.19 and Fig. 2.20.

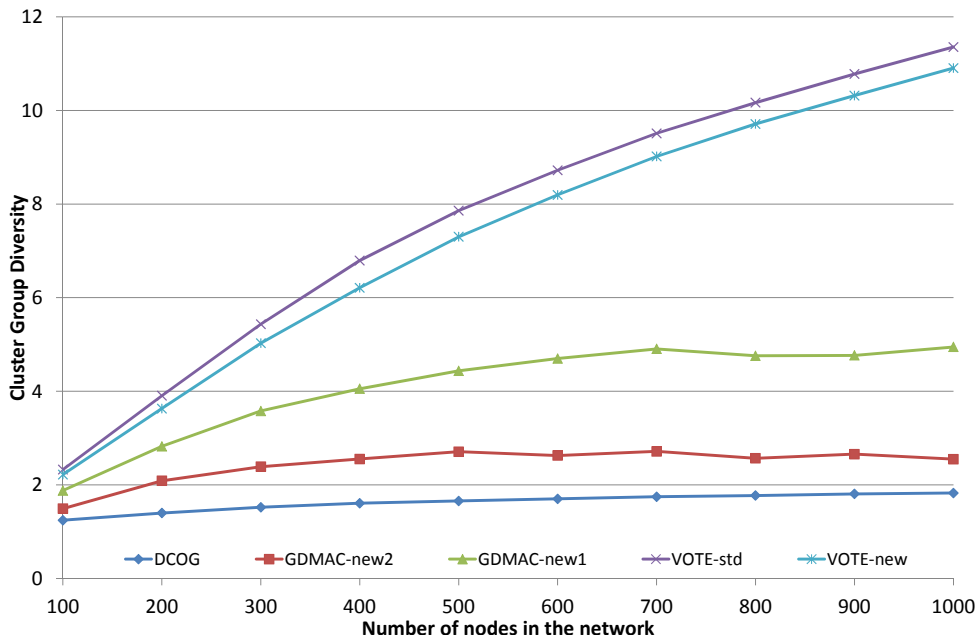


Figure 2.19: Cluster group diversity in static networks vs. the number of nodes.

Let us now study the GDMAC-new2 and DCOG metrics, which are clearly better.

- Thanks to its affiliation procedure GDMAC-new2 succeeds in attaining lower values for both metrics: the CGD increases from to 1.5 to only 2.5, and the GCD increases from 2.5 to nearly 4.5. These results must be put in perspective with the fact that GDMAC-new2 builds small clusters, as GDMAC-new1.
- DCOG achieves the lowest CGD, which lies between 1 and 2. This means that the clusters built by DCOG usually contain members from only one or two operational groups. This is confirmed by GCD which is always close to 1, meaning that DCOG succeeds in collecting nearly all the members of one or more groups in the same cluster. This conclusion is true independently of the number of nodes. In addition to its cluster size management feature, this property is one of the main strengths of DCOG. Let us now interpret in a different way the reason why the DCOG cluster size metric increases with the number of nodes. When N increases, the average node degree becomes larger, thus the probability that two entire groups are included in a single cluster increases, which logically raises the average cluster size.

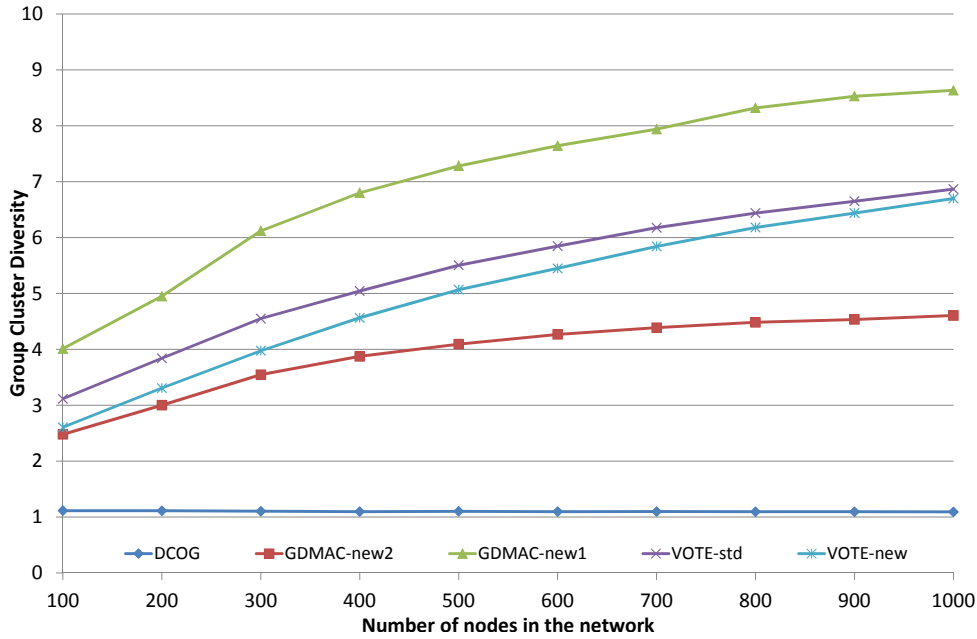


Figure 2.20: Group cluster diversity in static networks vs. the number of nodes.

2.6.6.5 Additional insights on DCOG performance

Different group size m_t have been simulated, keeping constant the maximal cluster size $n_{max} = 20$ and leading to different m_t/n_{max} ratios. To ensure that all groups are complete, the number of nodes in the network has been slightly adjusted, leading to the $(N, m_t, m_t/n_{max})$ 3-tuples listed in Table 2.10.

N	105	100	104	102
m_t	15	10	8	6
m_t/n_{max}	1.3	2	2.5	3.3

Table 2.10: N , m_t values for various m_t/n_{max} ratios.

The distribution of cluster size for all ratios m_t/n_{max} is plotted in Fig. 2.21. When the ratio is 1.3, 81% of the clusters formed by DCOG are as large as a group, and 16% are smaller. When the ratio is equal to 2 or 2.5 then DCOG mainly builds cluster whose size is either equal to the group size or equal to twice the group size. This behavior is justified by \mathbf{G}_2 , which focuses on forming clusters with the highest possible number of members from the same groups. For the same reason, when the ratio is equal to 3.3, DCOG mainly creates cluster whose size is once, twice or thrice the group size. When the ratio is greater than two, the probability to build clusters including a whole group is larger than building clusters including two (or three) entire groups. To understand this, one should remember that the nodes are deployed randomly in groups, and that there is no guarantee that the members of different groups can be gathered in a cluster satisfying the connectivity, size and diameter constraints. We explain in the same way the fact that DCOG also sometimes builds clusters whose size is not a multiple of the group size.

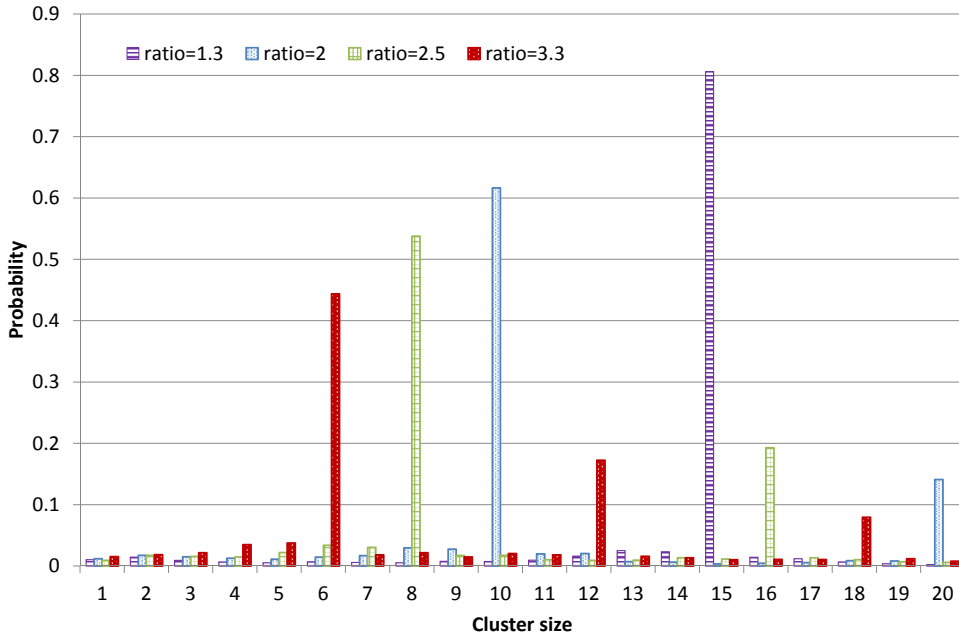


Figure 2.21: Cluster size distribution vs. m_t/n_{max} ratio.

2.6.6.6 Application level performance: average end-to-end delay

The values of J_0^X as defined in Section 1.4.3 are plotted in Fig. 2.22 for GDMAC-new1, GDMAC-new2, VOTE-std and VOTE-new. A box plot is drawn to detail the statistical results associated with DCOG. In this box plot the lowest bar is the lowest value, the second lowest is the 5th percentile, then the box boundaries provide the first quartile, median and third quartile. The second highest bar is the 95th percentile, and the highest bar is the highest value. The average value is identified by a disk.

Firstly, except for VOTE-std and VOTE-new which have similar J_0^X values, the difference between the different clustering solutions are significant. To understand why, let us recall that the metric J_0^X is related to the average end-to-end delay from all nodes to all nodes. To calculate J_0^X we have set the link weight $w_{i,j}$ using the following expression from page 52 of [43]:

$$w_{i,j} := c_{max} + [1 - \Upsilon(i,j)]^{-1} + c_{max}/[\Upsilon(i,j)^{c_{max}} - 1],$$

with $c_{max} = 4$ the maximum number of transmissions considering that a type I hybrid ARQ is used, and $\Upsilon(i,j)$ the packet error rate on link (i,j) , which depends on $\Gamma(i,j)$ the signal noise ratio (SNR) on the link. In our simulations a link (i,j) exists only if $\Gamma(i,j) > 0$ dB, for which $\Upsilon(i,j) = 0.13$. The consequence is that $w_{i,j} \in [1, 1.15]$, meaning that the delay is never significantly greater than 1. In fact it is slightly greater than 1 only for a small number of links which tend to be avoided during the shortest path calculations performed to get the J_0^X value. Therefore the end-to-end delay strongly depends on the number of inter-cluster links, thus on the clustering solution.

Secondly, the DCOG J_0^X values are significantly lower than the ones of the other clustering schemes. Let us define \triangleleft_J a comparison operator between two clustering schemes clu_1 and clu_2 such as $clu_1 \triangleleft_J clu_2$ if the J_0^X value achieved by clu_1 is greater than the one of clu_2 . Comparing

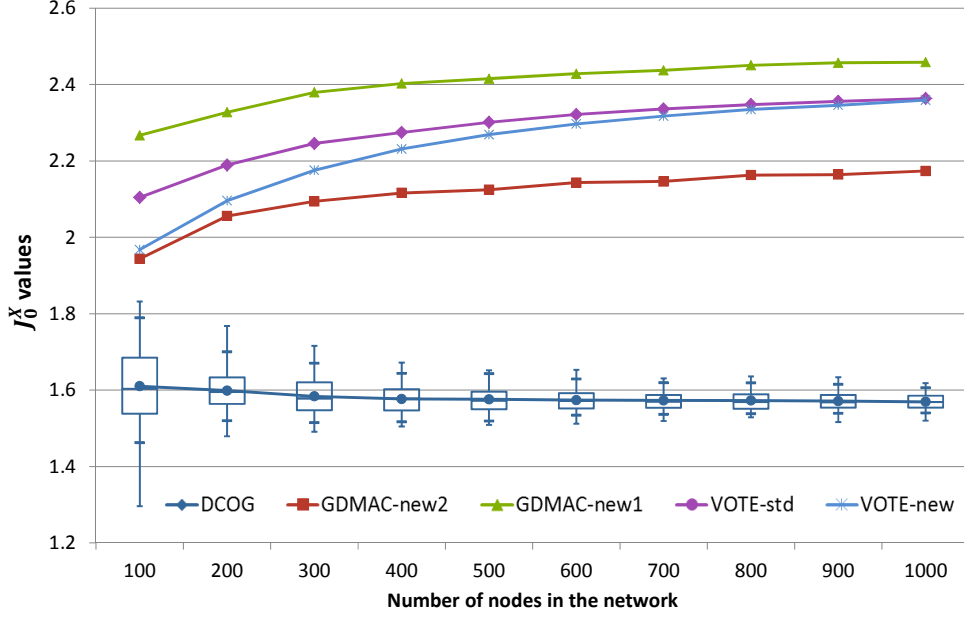


Figure 2.22: J_0^X values in static networks vs. the number of nodes.

the five clustering solutions we have:

$$\text{GDMAC-new1} \triangleleft_J \text{VOTE-std} \triangleleft_J \text{VOTE-new} \triangleleft_J \text{GDMAC-new2} \triangleleft_J \text{DCOG}.$$

DCOG achieves a gain in [29%, 36%] vs. GDMAC-new1, [23%, 34%] vs. VOTE-std, [18%, 33%] vs. VOTE-new, and [17%, 28%] vs. GDMAC-new2. J_0^X decreases when the amount of inter-cluster traffic decreases, which is the case also when the CGD and GCD decrease. This is the reason why DCOG, whose CGD and GCD are the lowest, achieves the best performance here. The same justification holds to justify why GDMAC-new2 is the second best. It is interesting to note that the two VOTE variants, that manage the cluster size, have lower J_0^X performance than GDMAC-new2. This allows to conclude that it is more important to gather in the same clusters the members of the same operational groups than to handle cluster size. But both are important, and this justifies the good performance of DCOG.

To better estimate the quality of DCOG, let us remark that no partition p leads to a highest J_0^X value than the partition consisting in as many singleton clusters as nodes in the network. In this case any link is an inter-cluster link, and its contribution to J_0^X is maximum (cf. Section 1.4.3). Similarly, even if the cluster \mathcal{V} is likely to violate the constraints, a convenient lower bound for J_0^X is associated to the network partition $\{\mathcal{V}\}$. During our simulations we calculated both bounds and found that whatever the number of nodes in the network, $J_0^X(p) \in [1.4, 2.8]$, $\forall p \in \mathcal{P}^c$. The J_0^X values achieved thanks to DCOG are close to 1.6, and are thus not far from the theoretical lower bound equal to 1.4 (knowing that the lowest J_0^X value achieved by a partition satisfying the constraints is somewhere in the interval [1.4, 1.6]).

Finally, the box plot in Fig. 2.22 shows that the distribution of J_0^X becomes more and more narrow as the number of nodes increases from 100 to 600-700 and then remains stable. This is explained thanks to the result of Section 2.6.6.1, showing that the network diameter remains quasi-unchanged when the N increases, leading to similar route lengths in the network, and thus similar J_0^X values independently of the number of nodes.

2.6.6.7 Application level performance: average end-to-end bandwidth

In a similar way as for the values of J_0^X , the values of T_0^X defined in Section 1.7 are plotted in Fig. 2.23.

A first remark is that, even if DCOG achieves the best result, the performance of all solutions are much closer than when measured with J_0^X . To calculate T_0^X , related to the average throughput from all nodes to all nodes, we have set the link weight $w_{i,j}$ equal to the link capacity, defined as follows:

$$w_{i,j} := \log_2[1 + \Gamma(i, j)].$$

Contrarily to the link delay used with J_0^X , the link capacity may vary a lot depending on the SNR. For example if $\Gamma(i, j) = 0$ dB then the capacity is equal to 1.0, but if $\Gamma(i, j) = 10$ dB (which is not a large value) then the capacity is increased to 3.3. Nevertheless, the main difference between T_0^X and J_0^X is the use of a $\min()$ operation instead of a sum to calculate $T_0^X(p, i, j)$. Due to this $\min()$ operation, the contribution of the source-destination pair (i, j) to T_0^X is given by the link with the lowest bandwidth. A single bad link between source i and destination j hurts T_0^X performance a lot more than J_0^X performance. Because DCOG takes into account the group membership and not the link quality, it is expected that its advantage over the other algorithms from a T_0^X perspective would be smaller than from a J_0^X perspective.

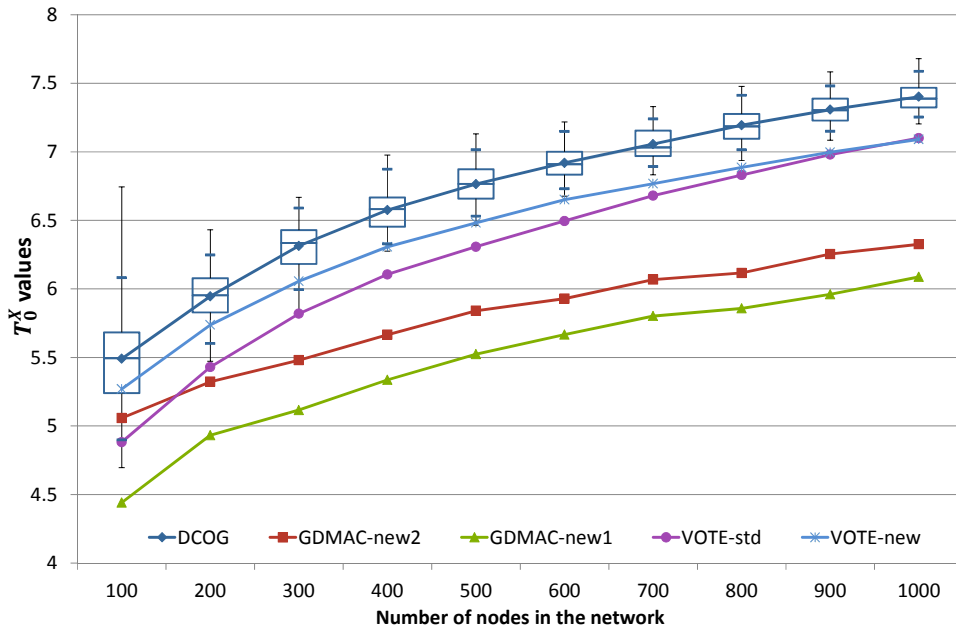


Figure 2.23: T_0^X values in static networks vs. the number of nodes.

Let us define \triangleleft_T a comparison operator between two clustering schemes clu_1 and clu_2 such as $\text{clu}_1 \triangleleft_T \text{clu}_2$ if the T_0^X value achieved by clu_1 is lower than the one of clu_2 . A second remark is that the partial ordering between the clustering algorithms is modified w.r.t. the one of the J_0^X metric:

$$\text{GDMAC-new1} \triangleleft_T \text{GDMAC-new2} \triangleleft_T \text{VOTE-std} \triangleleft_T \text{VOTE-new} \triangleleft_T \text{DCOG}.$$

DCOG still achieves the best performance but GDMAC-new2 is now the second worst solution instead of being the second best. Concerning capacity, it is more important to build bigger

clusters which reduces the number of inter-cluster links than collecting together in the same cluster members of the same groups. This is better achieved by the VOTE variants than by the GDMAC variants. Because DCOG takes into account both cluster size and group membership, it succeeds in reducing the most the number of inter-cluster links, thus achieving the best performance w.r.t. the T_0^X metric.

2.6.7 Performance in dynamic networks

2.6.7.1 Simulation setup

In this section each simulation concerns networks with $T = 10$ and $m_t = 10$. The node mobility model is the aforementioned RPGM group mobility model [42], modified such as to make sure that no node ever moves with a speed greater than a maximum v_{max} . To assess performance in mobility, simulations have been run for varying maximum node speed $v_{max} \in \{1, 3, 5, 7.5, 10, 12.5, 15, 20\}$.

A simulation is split in two phases: the transient phase and the steady phase. When there is no mobility, the steady phase begins when nothing change any more, i.e., concerning our simulations in Section 2.6.6, after the cluster structure has become stable. In presence of node mobility, the focus is on the performance during the steady state, and the metrics measured during the transient phase (also called warmup period) must be ignored. The results of Section 2.6.6.2 show that when $N = 100$, the clustering algorithms converge in less than 10 rounds. Therefore we set the duration of the warmup period to 10 rounds.

Let us now justify our choice for the simulation duration. A metric that can be used to assess the amount of mobility is the average link duration. Because of the group mobility model, intra-group links duration is larger than the global average link duration. The numbers in Table 2.11 and Table 2.12 detail the average link durations and the average durations of intra-group links, versus the maximum node speed and the simulation duration. These numbers are plotted in Fig. 2.24. A first analysis of these numbers reveals that when the maximum

Max. speed → Sim. duration ↓	1	3	5	7.5	10	12.5	15	20
200	158	102	73.8	55.5	45.6	39.5	35.5	30.3
400	242	127	85.2	61.6	49.9	42.7	38.2	32.1
600	290	138	90.0	64.0	52.2	44.3	39.7	33.1
800	322	145	92.4	65.9	53.6	45.4	40.6	33.7
1000	344	150	93.8	67.4	54.5	46.1	41.0	34.0
1200	359	155	95.2	68.7	-	-	-	-
1400	370	158	96.1	69.8	-	-	-	-

Table 2.11: Average link durations.

speed is low, the intra-group link duration strongly depends on the simulation duration. This is illustrated in Fig. 2.24.a by the different plateaus occurring when the maximum speed is 1 and 3. On the other hand, the average link duration dependence on simulation duration is weaker, as shown in Fig. 2.24.b. An important property of the simulation is that its results should not be dependent on its duration. The above analysis shows that this is not doable in a

Max. speed → Sim. duration ↓	1	3	5	7.5	10	12.5	15	20
200	198	194	175	137	108	91.8	82.6	73.6
400	396	382	305	189	133	109	96.8	85.7
600	594	566	401	213	146	119	105	92.0
800	792	746	472	230	156	125	110	95.7
1000	990	922	525	245	162	129	113	98.0
1200	1190	1090	565	252	-	-	-	-
1400	1390	1260	597	261	-	-	-	-

Table 2.12: Average intra-group link durations.

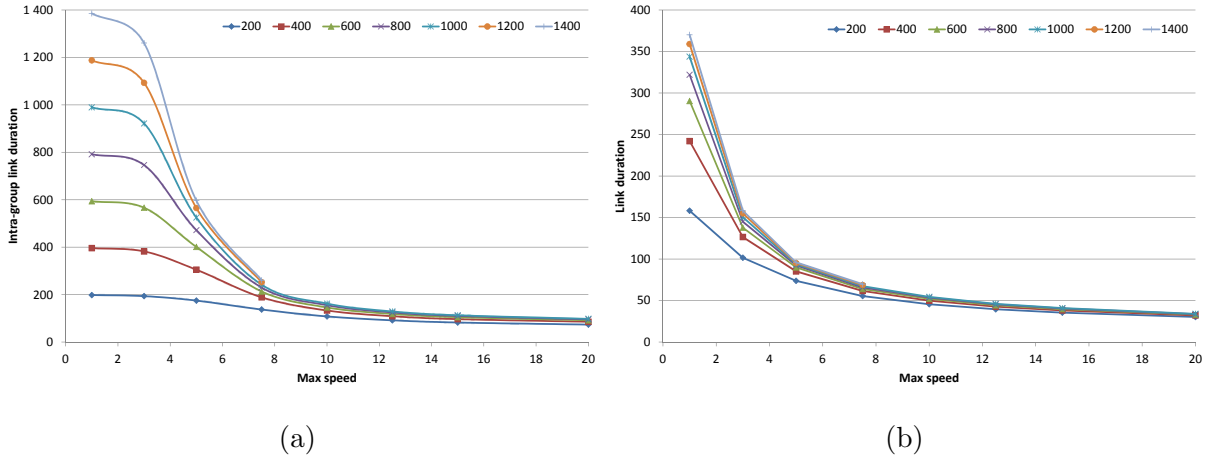


Figure 2.24: Link durations for various simulation durations increasing from 200 to 1400 time units. (a) Intra-group links. (b) Intra-group and inter-group links.

reasonable time for intra-group links when the maximum speed is very low. Table 2.13 provides the increase of link durations when the simulation duration is increased by 200. These numbers are in percentage. For example the number 53.0 in the line associated with simulation duration 400 and maximum speed 1 m/s is calculated by dividing the number 242 which is the link duration when simulation duration is 400 (in Table 2.11) by the number 158 which is the link duration when the simulation duration is 200 (same table). Thus 53.0 means that increasing from 200 to 400 the simulation duration leads to a 53% increase in link duration. In this table, the cells have been filled in green as soon as this percentage increase is lower than 10%. This condition is satisfied for all considered maximum speed if the simulation duration is greater or equal than 1000. Consequently, we choose to set the simulation duration to 1000 rounds.

2.6.7.2 Single simulation results

In mobility conditions, the average node degree is similar to the one during the simulations in static networks, i.e. equal to 20 as in Fig. 2.15. Nevertheless, the instantaneous node degree is different from its average value. To illustrate the instantaneous behavior of the simulation, we plot several relevant metrics vs. time in Fig. 2.25 to Fig. 2.29. These figures are the result of the simulation of a scenario with $N = 100$ and $v_{max} = 10.0$.

Max. speed → Sim. duration ↓	1	3	5	7.5	10	12.5	15	20
400	53.0	24.6	15.4	11.1	9.46	8.09	7.52	6.01
600	19.9	8.89	5.65	3.88	4.54	3.83	4.02	3.21
800	10.8	5.30	2.73	2.92	2.84	2.45	2.18	1.62
1000	6.82	3.68	1.50	2.37	1.52	1.46	1.01	1.10
1200	4.42	3.10	1.43	1.87	-	-	-	-
1400	3.08	2.09	0.97	1.63	-	-	-	-

Table 2.13: Average link durations difference.

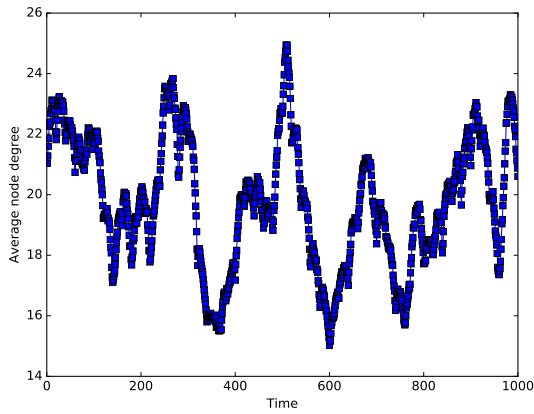


Figure 2.25: Average node degree vs. time.

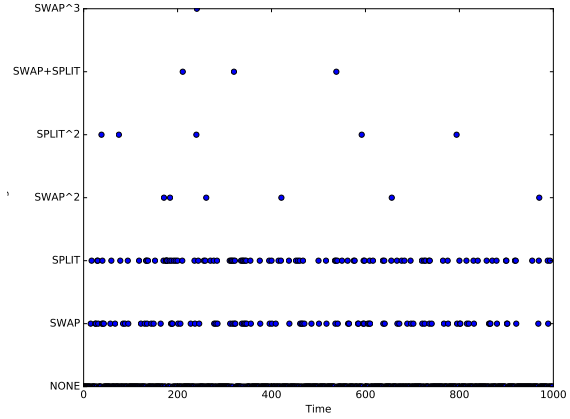


Figure 2.26: Clustering events vs. time.

As depicted in Fig. 2.25, the average node degree does not remain constant, showing large topological modifications along time.

Fig. 2.26 shows how many and when clustering events happened during the simulation. These events may be *i*) a swap between two clusters, or *ii*) a cluster split. A cluster that is split is destroyed and leads to a number of new clusters. Fig. 2.26 shows that during a simulation multiple such events may happen simultaneously: either no event (line NONE), one swap or one split (lines SWAP and SPLIT respectively), two swaps (line SWAP²), two splits (line SPLIT²), one swap and one split (line SWAP+SPLIT) or even three swaps (line SWAP³).

Fig. 2.27 and Fig. 2.28 depict average cluster size, CGD and GCD. The GCD is equal to 1 during 99.0% of the simulation, meaning that the groups are nearly always included in a single cluster. Consequently, during 99.0% of the simulation the cluster size curve is the enlarged version of the CGD curve by a factor of the group size, 10.

Finally Fig. 2.29 and Fig. 2.30 provide application level performance with J_0^X and T_0^X metrics. As expected (see Section 2.6.6.6 and Section 2.6.6.7) the former is less volatile than the latter. No correlation seems to exist between any of these curves and the ones of Fig. 2.25, Fig. 2.27, or Fig. 2.28.

2.6.7.3 Simulation statistical results

To assess the stability of the cluster structure built by the different clustering solutions, usually metrics such as *average cluster duration*, or *average cluster head life time*, etc. are used. These

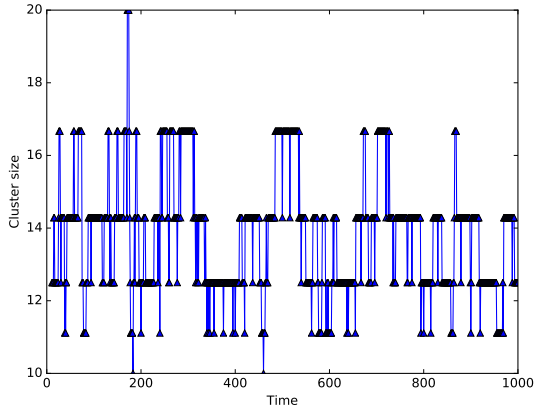


Figure 2.27: Average cluster size vs. time.

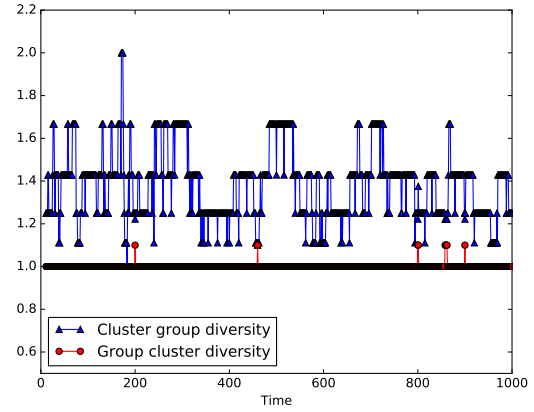


Figure 2.28: Average cluster group and group cluster diversities vs. time.

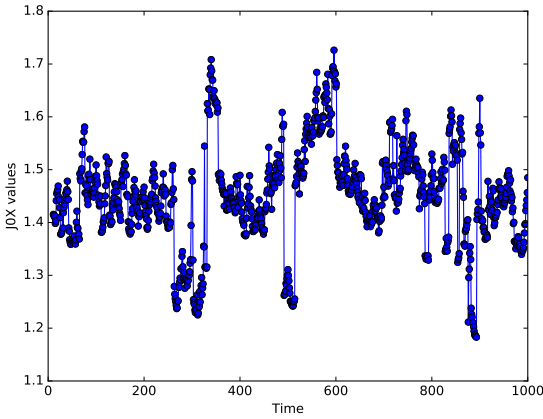


Figure 2.29: J_0^X values vs. time.

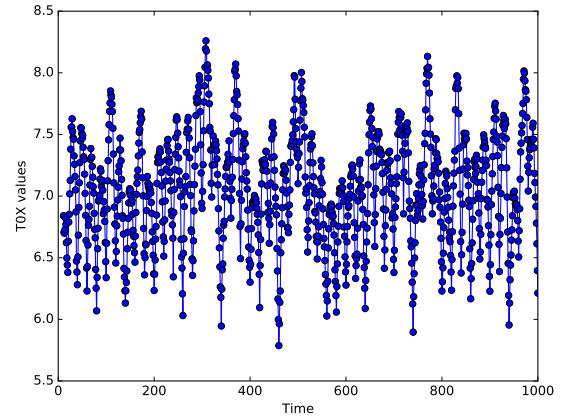


Figure 2.30: T_0^X values vs. time.

metrics cannot be used to perform a fair comparison between DCOG and the reference clustering algorithms because DCOG defines a cluster through its members and does not have any CH. From the point of view of DCOG, when one or more nodes leave a cluster to join a neighboring one, then both source and destination clusters are modified. Consequently, in this section we define as stability metric the ratio of the number of simulation rounds when no cluster was modified to the simulation time. Values of this ratio close to 1 indicate stable clusters, conversely, values close to zero indicate unstable clusters.

The plot of this metric in Fig. 2.31 shows that VOTE-std, VOTE-new and GDMAC-std are highly unstable, having less than 50% stability as soon as the node maximum speed exceeds 3 distance units per simulation round. Thanks to their small average cluster size and their use of group information GDMAC-new1 and GDMAC-new2 succeeds in keeping a stability greater than 50% up to a 5 distance units per simulation round. Finally, DCOG achieves by far the best stability, ensuring more than 60% stability even when the node maximum speed is as high as 20 distance units per simulation round. A first rationale for this good performance is the quick convergence property of DCOG, as shown in Section 2.6.6.2. This is achieved also thanks to a GCD whose values are always very close to 1, indicating that all members of a group are usually in the same cluster. Combined with the fact that nodes follow a group mobility pattern, a DCOG cluster is very stable.

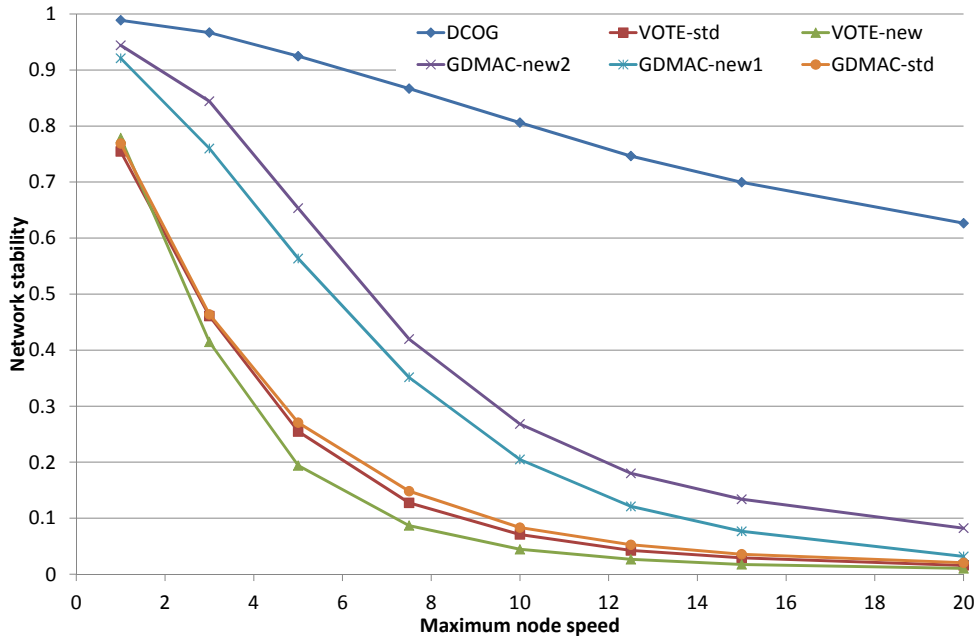


Figure 2.31: Mobile networks stability depending on maximum speed.

2.7 Conclusions

In this chapter we detailed our methodology to design the novel distributed clustering algorithm DCOG suited to structured ad hoc networks such as public safety or military networks.

Firstly, thanks to an exhaustive partition enumeration we found a cluster cost function suitable to identify good partitions, and thus good clusters. Secondly, we explained how from a centralized algorithm, we modified this cluster cost function and elaborated a decision making process, which led us to the distributed clustering algorithm that we called DCOG. The key feature of DCOG is its capability to build size limited clusters including entire groups, without resorting to CH nodes. Thirdly, we formalized our algorithm, and proved its theoretical convergence.

Our simulations in static and mobile conditions showed that thanks to DCOG, it is now possible to operate large scale dense networks based on groups and ensure good performance with respect to end-to-end delay and network stability. The comparison with existing solutions showed that our solution outperforms the ones from the literature. Another interesting feature of DCOG is its small number of parameters compared to other solutions, such as GDMAC, which makes it easier to use.

In Chapter 3, DCOG is revisited within the coalitional game theory framework. This enables us to characterize its final state after convergence, to suggest more general cost functions and gain insights on the design parameters. Moreover, this framework allows us to find a new algorithm for unstructured ad hoc networks that outperforms the conventional ones.

Chapter 3

Distributed clustering and coalitional game theory

Part of the work presented in this chapter is under preparation for future publication [44].

3.1 Introduction

Coalition game theory is the branch of game theory used to study the behavior of players when they cooperate among themselves [17]. Coalitional games involve a set of players who want to cooperate by forming *coalitions* in order to improve their positions in the game. In such games the coalitions are formed based on their *value*, which quantifies their worth in the game. Within a coalition the players receive a *benefit* thanks to their membership, which directly depends on the coalition value. When the value of a coalition can be divided between its members using any division rule, the game is said to be with *transferable utility* (TU). Otherwise the game is said to be with *non transferable utility* (NTU).

Numerous coalition formation algorithms have been proposed to solve many different problems. However some commonalities can be found. A first set of solutions involves the notion of *switch operation* in which a player leaves its current coalition and joins another one. Another family of proposals is based on the notion of *coalition merge and split*. In both cases, the benefit achieved thanks to the chosen algorithm must be positive to really improve the situation of the involved players. Usually the proposed algorithms own stability properties, such as for example Nash-stability [45], that guarantee the stability of the partition resulting from its execution.

Let us consider [20], where roadside units (RSU) form coalitions to diversify the classes of data they transmit to the vehicles within a VANET, in order to increase their revenue. In this paper the game is with TU, and the value of a coalition is shared between its members such that any player obtains a benefit no less than its benefit when acting non-cooperatively. In this paper, similarly to what is chosen in numerous other papers, the initial partition of the network is formed by singletons. The algorithm is synchronous: during each coalition formation iteration, the players are invoked sequentially. In the first phase each player i firsts looks for potential switch operations, i.e., leaving its current coalition C_k and joining another one C_ℓ , such that $C_\ell \cup \{i\} \succ_i C_k$, with \succ_i a preference relation between coalitions. In the second phase, for each player i , if one switch operation has been found in phase 1, then player i leaves its current coalition C_k to join C_ℓ . The algorithm is repeated until a Nash-stable partition is achieved, which

is proved to always happen thanks to the fact that a graph has a finite number of partitions, and that the same partition is never visited twice by a player.

In [19] the players are small cell base stations (SBS) that cooperate to share their radio resources (OFDMA subchannels), in order to deal with the downlink co-tier interference suffered by the small cell user devices from neighboring SBSs. The SBSs form overlapping coalitions, and dedicate part of their frequency resources to each coalition they belong to. The benefit received by a coalition member is equal to the fraction of frequency resource that this members gets from its coalition multiplied by the coalition value, which is its achieved sum-rate. The game is with TU. The authors define the preference relations $\succ_{\mathcal{C}}$ and $\succ_{\mathcal{I}}$ to compare two coalitional structures w.r.t. the SBS benefits and the coalition values. Given the current coalition structure, thanks to these relations and for each subchannel, each SBS decides if this subchannel should be allocated to another coalition: first if possible an existing one using the preference relation $\succ_{\mathcal{C}}$, and otherwise a new one using the preference relation $\succ_{\mathcal{I}}$. When a subchannel is moved from one coalition to another, an history set is maintained in order to prevent this subchannel to be moved back later in the same coalition. This process is repeated until convergence to a stable coalition structure.

The authors in [18] detail an algorithm suited to cognitive networks with primary and secondary users (SU). Thanks to their algorithm, SUs form disjoint coalitions in order to perform collaborative sensing while maximizing utility in terms of detection probability and accounting for a false alarm cost. Here the game is with NTU, the benefit of each member of the coalition being equal to the coalition value. The algorithm is synchronous: during each coalition formation stage, coalitions are invoked sequentially to perform first merge then split operations. In the merge phase, the coalition with the highest value merges with a nearby coalition, then the merged coalition merges again, etc., until there is no more any benefit to merge the resulting coalitions. During this phase, a coalition \mathcal{C}_k may merge with a nearby coalition \mathcal{C}_ℓ only if $\{\mathcal{C}_k \cup \mathcal{C}_\ell\} \succ \{\mathcal{C}_k, \mathcal{C}_\ell\}$, with \succ a comparison relation between collections of coalitions governed by the Pareto order. Following the merge process, the coalitions are subject to split operations. A coalition $\{\mathcal{C}_k, \mathcal{C}_\ell\}$ may split into the two coalitions \mathcal{C}_k and \mathcal{C}_ℓ only if $\{\mathcal{C}_k, \mathcal{C}_\ell\} \succ \{\mathcal{C}_k \cup \mathcal{C}_\ell\}$. With this relation, coalitions merge (split) only if at least one SU is able to strictly improve its benefit through this merge (split) without decreasing the benefits of other SU. The proposed algorithm is proved to converge to a \mathbb{D}_{hp} -stable partition, i.e., no player in this coalition are interested in leaving it through any merge-and-split operation to form other partitions.

Device-to-device (D2D) communications are the subject of [23] which proposes a distributed resource management scheme to jointly solve the problem of resource sharing mode selection and spectrum sharing. Here sharing mode can be *i*) cellular mode, *ii*) dedicated D2D mode and *iii*) hybrid mode where D2D reuse the resources of some cellular links. Players are either single cellular users involved in communication with the eNB, or pairs of cellular users involved in D2D communications. The game is with NTU, the benefit of a player being its achieved rate. The proposed algorithm is based on a succession of merge-and-split operations, using a preference relation \succ governed by the Pareto order to compare collection of coalitions.

The authors of [22] propose a distributed algorithm to achieve cooperative communications in ad hoc networks (with multiple sources and one destination, similarly to a WSN) in order to increase the achievable rate. Within a coalition the communications are performed over two phases: the broadcasting and the cooperation phases. During each slot of the broadcasting phase,

one coalition member performs broadcast transmission, while the other coalition members are listening. Then during the cooperation phase, all coalition members transmit a linearly coded signal of all signals received during the previous broadcasting phase, and the destination performs multiuser detection to extract the per node signal. The benefit of each coalition member is equal to its achievable rate, and the value of a coalition is the sum of its members benefits. The game is thus with NTU.

These examples prove that, when some benefit can be achieved thanks to cooperation, coalitional game theory can be used advantageously to solve numerous problems arising in the wireless communication field. In light of this, it seems natural to apply coalitional game theory to cluster building in wireless ad hoc networks. Within this chapter we thus reinterpret the DCOG algorithm from Chapter 2 in the context of this theory. This allows us to design a generic clustering algorithm which is *i*) applicable to any type of wireless mobile ad hoc network, and is *ii*) parameterized according to some utility functions and heuristics. Our analysis leads us to a specific family of utility functions and to several heuristics suitable for structured networks, and also allows us to deal with the case of unstructured networks.

Within this chapter, Section 3.2 introduces a generic clustering algorithm for mobile ad hoc networks based on coalitional game theory. It is generic in the sense that it is agnostic of the network structure. This generic algorithm is then adapted to structured and unstructured networks in Section 3.3 and Section 3.4 respectively. Finally Section 3.5 is devoted to simulation results and analysis.

3.2 Generic clustering algorithm based on coalition formation theory

In this section we begin by introducing the coalition formation framework that is used to derive the algorithm. We then detail the algorithm that is split into two specific procedures and prove its convergence.

3.2.1 Useful definitions related to coalition formation game theory

The coalition formation games in this chapter involve a set of players \mathcal{N} , also denoted by \mathcal{V} in the graph theory context of the two previous chapters. Let us now recall some definitions from the coalition game theory [17] that are instrumental in the derivation of our clustering algorithm, starting with the notions of coalition and *coalition structure*.

Definition 3.1 A coalition structure (or coalition partition) is defined as the set $p := \{\mathcal{C}_1, \dots, \mathcal{C}_{N_c}\}$ where $\mathcal{C}_k \subseteq \mathcal{N}$ are disjoint coalitions verifying $\cup_{k=1}^{N_c} \mathcal{C}_k = \mathcal{N}$.

Within a coalition structure we associate three different quantities to each coalition \mathcal{C}_k :

- The *utility* $u(\mathcal{C}_k) \geq 0$ quantifies the worth of the coalition, with $u(\emptyset) = 0$. The expression of the utility is a design parameter that depends on the goal of the coalitional game.
- The *cost* $c(\mathcal{C}_k)$ quantifies the cost of cooperation. In this thesis, we use the cost to account for the constraints imposed to the coalitions. As a consequence, we put: $c(\mathcal{C}_k) = 0$ if all constraints are satisfied and $c(\mathcal{C}_k) = +\infty$ otherwise.

- The *value* $v(\mathcal{C}_k)$ is defined as the difference between the utility achieved thanks to the cooperation and the cost of cooperation: $v(\mathcal{C}_k) = u(\mathcal{C}_k) - c(\mathcal{C}_k)$. Note that $v(\mathcal{C}_k) = u(\mathcal{C}_k)$ when the constraints are satisfied, $v(\mathcal{C}_k) = -\infty$ otherwise.

The utility and the cost of a coalition depend on the type of network (unstructured or structured). Examples of utility functions are detailed in Sections 3.3 and 3.4. In this thesis we consider coalition formation games in characteristic form, i.e., when the value of a coalition only depends on its members [17].

A transfer of nodes from one coalition to another is called a *switch operation*:

Definition 3.2 A switch operation $\sigma_{k,\ell}(\mathcal{P})$ is defined as the transfer of players \mathcal{P} from $\mathcal{C}_k \in p$ to $\mathcal{C}_\ell \in p \cup \{\emptyset\}$, $\sigma_{k,\ell}(\mathcal{P}) : \mathcal{C}_k \mapsto \mathcal{C}_k \setminus \mathcal{P}$, and $\mathcal{C}_\ell \mapsto \mathcal{C}_\ell \cup \mathcal{P}$.

Note 1: if $\mathcal{C}_\ell = \emptyset$, then $\sigma_{k,\ell}(\mathcal{P})$ leads to the formation of a new coalition \mathcal{P} , thus increasing by one the number of coalitions. In that case, the switch operation is noted $\sigma_{k,\emptyset}(\mathcal{P})$.

Note 2: if $\mathcal{P} = \mathcal{C}_k$, then $\sigma_{k,\ell}(\mathcal{P})$ leads to the merge of \mathcal{C}_k with \mathcal{C}_ℓ , thus decreasing by one the number of coalitions.

To determine if a switch operation improves the coalition structure, we now define the *switch operation gain*.

Definition 3.3 The switch operation gain $g(\sigma_{k,\ell}(\mathcal{P}))$ associated with $\sigma_{k,\ell}(\mathcal{P})$ is defined as:

$$g(\sigma_{k,\ell}(\mathcal{P})) := r_{\mathcal{P}}(\mathcal{C}_\ell \cup \mathcal{P}) - r_{\mathcal{P}}(\mathcal{C}_k), \quad (3.1)$$

with $r_{\mathcal{P}}(\mathcal{C}_k)$ defined as:

$$r_{\mathcal{P}}(\mathcal{C}_k) := v(\mathcal{C}_k) - v(\mathcal{C}_k \setminus \mathcal{P}). \quad (3.2)$$

The $r_{\mathcal{P}}(\mathcal{C}_k)$ quantity can be interpreted as the added value of having players \mathcal{P} in coalition \mathcal{C}_k . Let us now define the *preference relation* used by the players to compare two switch operations.

Definition 3.4 The preference relation \succ is defined as a complete and transitive binary relation between two switch operations $\sigma_{k,\ell}(\mathcal{P}_i)$ and $\sigma_{k',\ell'}(\mathcal{P}_j)$ such that:

$$\sigma_{k,\ell}(\mathcal{P}_i) \succ \sigma_{k',\ell'}(\mathcal{P}_j) \Leftrightarrow g(\sigma_{k,\ell}(\mathcal{P}_i)) > g(\sigma_{k',\ell'}(\mathcal{P}_j)). \quad (3.3)$$

Similarly, we also define \succeq as: $\sigma_{k,\ell}(\mathcal{P}_i) \succeq \sigma_{k',\ell'}(\mathcal{P}_j) \Leftrightarrow g(\sigma_{k,\ell}(\mathcal{P}_i)) \geq g(\sigma_{k',\ell'}(\mathcal{P}_j))$.

Using our notations, the *Nash-stability* [45] of a partition can be defined as:

Definition 3.5 A partition $p = \{\mathcal{C}_1, \dots, \mathcal{C}_{N_c}\}$ is Nash-stable if $\forall \mathcal{C}_k \in p$, for any node $i \in \mathcal{C}_k$, $g(\sigma_{k,\ell}(\{i\})) \leq 0$ for all $\mathcal{C}_\ell \in p \cup \{\emptyset\}$.

When the partition p is Nash-stable, it means that there exists no single node switch operation with a strictly positive gain.

3.2.2 Generic coalition formation algorithm for clustering

In this section we propose a generic, distributed and asynchronous coalition formation algorithm to cluster the network. To do that, we consider the coalition game theory formation framework, identifying coalitions as clusters, and players as nodes. In the sequel, we preferably use the

cluster/node terminology. We note n_k the size (or number of members) of cluster \mathcal{C}_k , and $d(\mathcal{C}_k)$ the diameter of its induced subgraph.

Implementation of ad hoc networks includes protocols that enable nodes to discover their neighbors and to exchange information between them. These protocols are either specific to the communication system providers or taken from standards such as the IEEE 802.15.4 one, designed for wireless personal area networks. In our work, we assume to use the capabilities provided by such protocols to share information between nodes such as cluster membership, link qualities, etc. This allows to disseminate useful information inside clusters and between neighbor clusters to implement the clustering algorithm.

The proposed algorithm is based upon comparisons of switch operation gains. It is *distributed* since the decision to perform a switch operation is done at the node level. Moreover, the decision-making time points are assumed not to be coordinated between nodes and thus the algorithm is *asynchronous*. However, when a node is evaluating the possibility to perform a switch operation, it may consider other nodes of its current cluster. To account for the asynchronism of the decision-making instants, nodes that are involved in a switch operation are set in a *busy* status. Nodes that are not in the *busy* status are said *available*. Before implementing a switch operation, availability of the nodes involved are checked. Switch operations are done between neighbor clusters; two clusters are neighbors if at least one node in one cluster is a neighbor of at least one node in the other cluster. In order to ensure that communications in between cluster members are possible, we seek clusters for which their subgraph is connected. This constraint is taken into account through the cost c . Notice that other constraints may be added, like the maximal cluster size (see Section 3.2.4).

When the network is static, we can prove that the algorithm converges to a stable solution where all constraints are satisfied. When nodes are mobile, the network topology changes over time. As a consequence, cluster fulfilling the constraints at one time may not satisfy them after some time. Thus, the algorithm needs to cope with this situation and to react to find a new cluster formation that respects the constraints. Our generic clustering algorithm can then be summarized as follows. As soon as a node starts a decision-making process, it first checks if the constraints of its current cluster are satisfied. If the constraints are fulfilled, it applies the procedure \mathbf{P}_1 to operate the best switch operation, described in Section 3.2.2.1. If not, it applies another procedure \mathbf{P}_2 , described in Section 3.2.2.2. Each procedure includes a selection of candidate nodes for switch operations, which is done according to common sense rules referred to as *heuristics* and noted \mathcal{H}_1 for \mathbf{P}_1 and \mathcal{H}_2 for \mathbf{P}_2 .

3.2.2.1 Procedure when constraints are fulfilled (\mathbf{P}_1)

When a node i starts a decision-making procedure and detects that its cluster does not satisfy the constraints, it triggers the procedure \mathbf{P}_1 in order to search for a strictly positive gain switch operation and implement it. The principle of this procedure run at each node i (summarized in Table 3.1) splits into the three following successive steps:

1. **Selection of candidate switch operations (lines 1-9).** We first build the potential candidate sets of nodes denoted by $\{\mathcal{P}_{1,i}(a)\}_{a=1}^{A_1}$ according to \mathcal{H}_1 . We assume that the heuristic \mathcal{H}_1 returns at least one element, the node i itself, hence $A_1 \geq 1$. For each potential candidate set, the switch operation gain is evaluated against all the neighbor clusters. We keep as candidates the switch operations with strictly positive gain.

2. **Selection of one best switch operation (line 11).** If the set of candidate switch operations is not empty, a best switch operation is selected among those with the largest switch operation gain. Otherwise, the procedure is terminated.
3. **Implementation of the switch operation (lines 12-14).** Firstly, we check if the identified switch operation can be performed, i.e., if all the nodes involved (actually $\mathcal{C}_k \cup \mathcal{C}_{\ell^*}$) are available. Then, if the condition is met, the selected switch operation is performed, otherwise it is dropped.

```

// Selection of candidate switch operations
1 Set  $\mathcal{M} = \emptyset$ .
2 Apply heuristic  $\mathcal{H}_1$  to node  $i$  to get  $\{\mathcal{P}_{1,i}(a)\}_{a=1}^{A_1}$ , with  $\mathcal{P}_{1,i}(a) \in \mathcal{C}_k$ 
and  $A_1 \geq 1$ .
3 For each  $a \in \{1, \dots, A_1\}$  do:
4   For each  $\mathcal{C}_\ell$  neighbor of  $\mathcal{C}_k$  do:
5     If  $g(\sigma_{k,\ell}(\mathcal{P}_{1,i}(a))) > 0$  then:
6       Set  $\mathcal{M} = \mathcal{M} \cup \sigma_{k,\ell}(\mathcal{P}_{1,i}(a))$ .
7     End If.
8   End For.
9 End For.
10 If  $\mathcal{M} \neq \emptyset$  then:
// Selection of one best switch operation
11 Find  $(a^*, \ell^*)$  such that  $\sigma_{k,\ell^*}(\mathcal{P}_{1,i}(a^*)) \succeq \sigma_{k,\ell}(\mathcal{P}_{1,i}(a))$ ,
                                                                     $\forall \sigma_{k,\ell}(\mathcal{P}_{1,i}(a)) \in \mathcal{M}$ .

// Implementation of the switch operation
12 If the nodes involved in  $\sigma_{k,\ell^*}(\mathcal{P}_{1,i}(a^*))$  are all available then:
13   The nodes in  $\mathcal{P}_{1,i}(a^*)$  join  $\mathcal{C}_{\ell^*}$ .
14 End If.
15 End If.

```

Table 3.1: Procedure \mathbf{P}_1 at node $i \in \mathcal{C}_k$ of the generic clustering algorithm when constraints are satisfied.

3.2.2.2 Procedure when constraints are not fulfilled (\mathbf{P}_2)

When a node i starts a decision-making procedure and detects that its cluster does not satisfy the constraints, it triggers a procedure to change the cluster topology looking for a new partition matching the constraints. The procedure chooses among three different actions: *i*) do nothing, *ii*) some members of the cluster (including node i) join another cluster, or *iii*) some members of the cluster (including node i) form a new cluster. Action (*i*) happens when no candidate switch operation is identified. The principle of this procedure run at each node i (summarized in Table 3.2) splits into the three following successive steps:

1. **Selection of candidate switch operations (lines 1-10).** We first build the potential candidate sets of nodes denoted by $\{\mathcal{P}_{2,i}(a)\}_{a=1}^{A_2}$ according to \mathcal{H}_2 . When the potential can-

didate set is not empty, we evaluate the value $r_{\mathcal{P}}$ in (3.2) of the merge of all the potential candidate sets against all the neighbor clusters and keep those with strictly positive gains.

2. **Selection of one best switch operation (line 11).** The best switch operation selection is one among those with the largest $r_{\mathcal{P}}$ value.
3. **Implementation of the switch operation (lines 12-16).** Firstly, we check if the identified switch operation can be performed, i.e., if all the nodes involved (actually $\mathcal{P}_{2,i}(a^*) \cup \mathcal{C}_{\ell^*}$) are available. If this condition is met, then the selected switch operation is performed, otherwise nodes available in $\mathcal{P}_{2,i}(a)$ form a new cluster.

```

// Selection of potential candidate switch operations
1 Apply heuristic  $\mathcal{H}_2$  to node  $i$  to get  $\{\mathcal{P}_{2,i}(a)\}_{a=1}^{A_2}$ , with  $\mathcal{P}_{2,i}(a) \in \mathcal{C}_k$ 
and  $A_2 \geq 0$ .
// Selection of candidate switch operations
2 If  $A_2 > 0$  then:
3   For each  $a \in \{1, \dots, A_2\}$  do:
4     Set  $\mathcal{M} = \{\sigma_{k,0}(\mathcal{P}_{2,i}(a))\}$ .
5     For each  $\mathcal{C}_{\ell}$  neighbor of  $\mathcal{C}_k$  do:
6       If  $r_{\mathcal{P}_{2,i}(a)}(\mathcal{C}_{\ell} \cup \mathcal{P}_{2,i}(a)) > 0$  then:
11        Set  $\mathcal{M} = \mathcal{M} \cup \sigma_{k,\ell}(\mathcal{P}_{2,i}(a))$ .
8       End If.
9     End For.
10  End For.
// Selection of one of the best switch operation
11 Find  $(a^*, \ell^*)$  such that  $r_{\mathcal{P}_{2,i}(a^*)}(\mathcal{C}_{\ell^*} \cup \mathcal{P}_{2,i}(a^*)) \geq r_{\mathcal{P}_{2,i}(a)}(\mathcal{C}_{\ell} \cup$ 
 $\mathcal{P}_{2,i}(a)), \forall \sigma_{k,\ell}(\mathcal{P}_{2,i}(a)) \in \mathcal{M}$ .
// Implementation of the switch operation
12 If the nodes involved in  $\sigma_{k,\ell^*}(\mathcal{P}_{2,i}(a^*))$  are all available then:
13   Nodes  $\mathcal{P}_{2,i}(a^*)$  join  $\mathcal{C}_{\ell^*}$ .
14 Else:
15   The nodes in  $\mathcal{P}_{2,i}(a^*)$  that are available form a new cluster.
16 End If.
17 End If.

```

Table 3.2: Procedure \mathbf{P}_2 at node $i \in \mathcal{C}_k$ of the generic clustering algorithm when constraints are not satisfied.

Although \mathbf{P}_2 is organized in the same lines as in \mathbf{P}_1 , its differs along the following features:

- Conversely to \mathcal{H}_1 , \mathcal{H}_2 may return an empty set for the potential candidate since moving node i (and some of its neighbors) may not improve the fulfillment of the constraints.
- Switch operations are selected using the value of $r_{\mathcal{P}}$ in (3.2) instead of the switch operation gain g in (3.1). The reason is the following: when at least one constraint in \mathcal{C}_k is not fulfilled, then $v(\mathcal{C}_k) = -\infty$ and $r_{\mathcal{P}}(\mathcal{C}_k)$ is undefined. As a consequence, the switch operation gain cannot be used. However if there is one \mathcal{C}_{ℓ} such that \mathcal{C}_{ℓ} and $\mathcal{C}_{\ell} \cup \mathcal{P}$ satisfy the

constraints, $r_{\mathcal{P}}(\mathcal{C}_\ell \cup \mathcal{P})$ still exists. Remembering that $r_{\mathcal{P}}(\mathcal{C}_\ell \cup \mathcal{P})$ quantifies the gain associated with the arrival of \mathcal{P} in \mathcal{C}_ℓ , it seems thus relevant to use this quantity instead of g .

- When the nodes are not available to implement the selected switch operation, \mathbf{P}_2 creates a new cluster instead of dropping the switch. The rationale to do that is because dropping the switch would not resolve any of the constraints infringement, whereas creating a new cluster reduces the number of nodes that belong to a cluster not fulfilling the constraints.
- Conversely to \mathbf{P}_1 , the cluster \mathcal{C}_k may not satisfy the constraints after performing \mathbf{P}_2 , for instance because no switch operation is performed. The constraint infringement resolution may occur later by the other nodes of the cluster, when running \mathbf{P}_2 .

The important design parameters of this algorithm are: *i*) the heuristics \mathcal{H}_1 and \mathcal{H}_2 to select the candidate sets of nodes $\mathcal{P}_{1,i}(a)$ and $\mathcal{P}_{2,i}(a)$, and *ii*) the utility function u and the associated cost function c used to calculate switch operation gains (either for g or $r_{\mathcal{P}}$). Various heuristics, cluster utilities and costs can be chosen depending on the type of network. Several examples are given in Section 3.3 dedicated to structured networks and in Section 3.4 to unstructured networks.

3.2.3 Convergence properties

When the network topology is fixed and the algorithm is initialized by clusters fulfilling the constraints (the simplest being to set each node as a singleton cluster), then the following result holds:

Result 3.1 For a fixed network topology and starting from any initial partition p_0 of \mathcal{N} for which the clusters satisfy the constraints, the cluster formation algorithm maps to a sequence of switch operations which converges in a finite number of iterations to a final partition p_f .

Proof See Appendix B.1. ■

Since the candidate set of nodes returned by \mathcal{H}_1 includes the node i itself, we directly get:

Result 3.2 The final partition p_f achieved in Result 3.1 is Nash-stable.

3.2.4 On the cluster constraints and utility function

As mentioned in the introduction, we impose some constraints to the clusters in order to ease the RRA process. In addition to the connectivity constraint, we also impose a maximum number of nodes per cluster noted n_{max} and a maximum cluster diameter noted d_{max} . These constraints are noted by:

- $\rho_1(\mathcal{C}_k)$: the induced subgraph of \mathcal{C}_k is connected (already mentioned in § 3 of Section 3.2.2),
- $\rho_2(\mathcal{C}_k)$: $n_k \leq n_{max}$,
- $\rho_3(\mathcal{C}_k)$: $d(\mathcal{C}_k) \leq d_{max}$,

and the cost within a cluster \mathcal{C}_k is defined as:

$$c(\mathcal{C}_k) := \iota(\rho_1(\mathcal{C}_k)) + \iota(\rho_2(\mathcal{C}_k)) + \iota(\rho_3(\mathcal{C}_k)), \quad (3.4)$$

with $\iota(\text{condition}(\mathcal{C}_k)) := 0$ if $\text{condition}(\mathcal{C}_k)$ is satisfied, $+\infty$ otherwise.

For the same reasons, we want to build clusters that include as many members as possible (limited by the size constraint n_{max}). This is the goal \mathbf{G}_1 already defined in Section 2.5.5:

Goal 1 (\mathbf{G}_1) *Build clusters whose size is maximal, i.e., equal to n_{max} .*

To achieve \mathbf{G}_1 , the utility function should always allow the merge of two clusters, as long as the constraints are satisfied. Let us call this property Condition 3.1, that can be expressed as:

Condition 3.1 The cluster utility function must verify $g(\sigma_{k,\ell}(\mathcal{C}_k)) > 0$, $\forall(\mathcal{C}_k, \mathcal{C}_\ell) \in \mathcal{P}^2$, $k \neq \ell$, as long as $\mathcal{C}_k \cup \mathcal{C}_\ell$ satisfies the constraints.

In addition, Condition 3.1 ensures that the algorithm can be initialized with all the nodes as singleton clusters without being blocked in this configuration, which is not guaranteed for any utility function.

3.3 Clustering algorithm for structured mobile ad hoc networks

In this section we specify the generic clustering algorithm to the structured network case by designing dedicated utility function and heuristics. Hereafter, we need the following additional notation: let $m_{t,k}$ be the number of members of group \mathcal{O}_t in cluster \mathcal{C}_k .

3.3.1 Utility function

As discussed in the Introduction of this document, in structured networks, the members of the same group exchange the main part of their traffic within their group, and intra-cluster links benefit from a RRA which is more efficient than the one associated with the inter-cluster links. Therefore, as much as possible, we want to collect the members of the same group into a single cluster. This is the goal \mathbf{G}_2 already defined in Section 2.5.5:

Goal 2 (\mathbf{G}_2) *Build clusters including the highest number of members from the same group.*

As already seen in Section 3.2.4, we also want to achieve \mathbf{G}_1 . We now show that we can define for each of the two goals a specific utility function adapted to the goal. Therefore, we suggest to define a utility function for structured networks as a linear combination of these two utility functions:

$$u_{st}(\mathcal{C}_k) := u_1(\mathcal{C}_k) \cdot \epsilon + u_2(\mathcal{C}_k) \cdot (1 - \epsilon), \quad \epsilon \in (0, 1), \quad (3.5)$$

where

- u_1 is a utility function adapted to \mathbf{G}_1 and which thus only depends on the cluster size:

$$u_1(\mathcal{C}_k) := f_1(n_k), \quad (3.6)$$

- u_2 is a utility function adapted to \mathbf{G}_2 and which thus only depends on the number of nodes per group in the cluster:

$$u_2(\mathcal{C}_k) := \sum_{t \in \mathcal{I}(\mathcal{C}_k)} f_{2,t}(m_{t,k}), \quad (3.7)$$

with $\mathcal{I}(\mathcal{C}_k)$ defined in Section 2.2.

We now want to determine relevant functions for f_1 and $f_{2,t}$. Regarding f_1 , we use the following result.

Result 3.3 Let two different clusters \mathcal{C}_k and \mathcal{C}_ℓ satisfying the constraints, and a set of nodes $\mathcal{P} \subset \mathcal{C}_k$ such that

- i) $|\mathcal{C}_\ell \cup \mathcal{P}| > |\mathcal{C}_k|$ and
- ii) $\mathcal{C}_\ell \cup \mathcal{P}$ and $\mathcal{C}_k \setminus \mathcal{P}$ satisfy the constraints.

If f_1 is strictly convex then u_1 defined by (3.6) verifies $g(\sigma_{k,\ell}(\mathcal{P})) > 0$.

Proof See Appendix B.2. ■

As a corollary, f_1 also verifies Condition 3.1 (apply Result 3.3 with $\mathcal{P} = \mathcal{C}_k$). Consequently, we propose to select a strictly convex function f_1 to achieve \mathbf{G}_1 . Note that this result also holds for unstructured networks since it depends only on the cluster size. As the simplest strictly convex function is the second-order monomial, we propose to use

$$f_1(n_k) := \frac{n_k^2}{n_{max}^2}, \quad (3.8)$$

where n_{max}^2 normalizes u_1 in $[0, 1]$.

Regarding $f_{2,t}$, we use the following result.

Result 3.4 Let two different clusters \mathcal{C}_k and \mathcal{C}_ℓ satisfying the constraints such that

- i) \mathcal{P} is a subset of $\mathcal{O}_t \cap \mathcal{C}_k$,
- ii) $m_{t,\ell} + |\mathcal{P}| > m_{t,k}$, and
- iii) $\mathcal{C}_\ell \cup \mathcal{P}$ and $\mathcal{C}_k \setminus \mathcal{P}$ satisfy the constraints.

If $f_{2,t}$ is strictly convex then u_2 defined by (3.7) verifies $g(\sigma_{k,\ell}(\mathcal{P})) > 0$.

Proof See Appendix B.3. ■

Consequently, the strict convexity of $f_{2,t}$ ensures that a switch operation leading to a larger number of members of the same group in the receiving cluster than in the departing cluster has a strictly positive gain. It thus contributes to achieving \mathbf{G}_2 .

The same way as for f_1 , we select the simplest second-order monomial function for $f_{2,t}$:

$$f_{2,t}(m_{t,k}) := \frac{m_{t,k}^2}{T \cdot m_t^2}, \quad (3.9)$$

where $T \cdot m_t^2$ normalizes u_2 in $[0, 1]$.

The utility function previously defined holds the following property.

Result 3.5 The utility function defined as (3.5) with (3.6)-(3.9) satisfies condition 3.1.

Proof See Appendix B.4. ■

Notice that, using (3.5) with (3.6)-(3.9) allows to model the problem of cluster building as a non-transferable utility coalitional game in which the benefit of each cluster member is equal to the utility of the cluster.

We now discuss the design of ϵ to control the trade off between \mathbf{G}_1 and \mathbf{G}_2 . Let us first state the following result.

Result 3.6 Let three different clusters satisfying the constraints \mathcal{C}_k , \mathcal{C}_ℓ and \mathcal{C}_q , each of them having some nodes belonging to the group \mathcal{O}_t . Let $\mathcal{U}_q \subset \mathcal{C}_q$ that contains some nodes only of group \mathcal{O}_t with $n_{\mathcal{U}} := |\mathcal{U}_q| \leq m_{t,q}$. We assume that $\mathcal{C}_k \cup \mathcal{U}_q$ and $\mathcal{C}_\ell \cup \mathcal{U}_q$ satisfy the constraints. Assuming that $\forall t, |\mathcal{O}_t| \leq n_{max}$ and using the utility function (3.5) with (3.6)-(3.9), the following properties hold:

- i)* If $m_{t,k} = m_{t,\ell}$ and $|\mathcal{C}_k| > |\mathcal{C}_\ell|$, then $g(\sigma_{q,k}(\mathcal{U}_q)) > g(\sigma_{q,\ell}(\mathcal{U}_q))$, $\forall \epsilon$.
- ii)* If $m_{t,k} = m_{t,\ell}$ and $|\mathcal{C}_k| = |\mathcal{C}_\ell|$, then $g(\sigma_{q,k}(\mathcal{U}_q)) = g(\sigma_{q,\ell}(\mathcal{U}_q))$, $\forall \epsilon$.
- iii)* If $m_{t,k} > m_{t,\ell}$, and $\forall |\mathcal{C}_k|, \forall |\mathcal{C}_\ell|$, then $g(\sigma_{q,k}(\mathcal{U}_q)) > g(\sigma_{q,\ell}(\mathcal{U}_q))$ as soon as:

$$\epsilon < \epsilon^* := \frac{1}{1 + T}.$$

Proof See Appendix B.5. ■

Firstly, when the number of nodes belonging to group \mathcal{O}_t in cluster \mathcal{C}_k and \mathcal{C}_ℓ are equal, Result 3.6 tells that, whatever the value of ϵ : *i)* the algorithm selects the switch of \mathcal{U}_q towards the cluster with the largest number of nodes, thus fulfilling \mathbf{G}_1 , *ii)* if cluster \mathcal{C}_k and \mathcal{C}_ℓ have the same number of nodes, both switch operations are even. Secondly, when the number of nodes belonging to group \mathcal{O}_t in cluster \mathcal{C}_k and \mathcal{C}_ℓ are different, let say $m_{t,k} > m_{t,\ell}$, then, for any $\epsilon < \epsilon^*$, the algorithm always selects the switch of $\mathcal{U}_q \subset \mathcal{O}_t$ towards the cluster which has the largest number of nodes of group \mathcal{O}_t , i.e., \mathcal{C}_k , regardless of the size of the clusters, and more specifically even if $|\mathcal{C}_k| < |\mathcal{C}_\ell|$. In that condition, it thus always fulfills \mathbf{G}_2 over \mathbf{G}_1 .

3.3.2 Heuristics for node selection

We assume that node i running the algorithm in cluster \mathcal{C}_k belongs to group \mathcal{O}_t . For structured networks, the heuristic should select the set that gathers the maximum number of members of group \mathcal{O}_t , including node i . We also want this set to fulfill the constraints. Let us denote this set by \mathcal{L} . Thus, in order to build \mathcal{L} , we first identify i^* the node among the neighbors of i in group \mathcal{O}_t that has the highest degree. Then, we select the set of nodes including i^* that has the largest cardinality and which respects the constraints. Note that when the diameter constraint is equal to two, \mathcal{L} is obtained by considering all the neighbors of i^* (which includes i by construction). This approach is used for the following heuristics used in procedures \mathbf{P}_1 and \mathbf{P}_2 .

3.3.2.1 In procedure \mathbf{P}_1

In order to assess the effect of the choice of the heuristic, we propose three heuristics \mathcal{H}_1^{sth} , with $h \in \{1, 2, 3\}$ that return the sets $\{\mathcal{P}_{1,i}(a)\}_{a=1}^{A_1}$. They are defined as:

- $\mathcal{H}_1^{\text{st1}}$: $A_1 = 2$ with $\mathcal{P}_{1,i}(1) = \{i\}$ and $\mathcal{P}_{1,i}(2) = \mathcal{L}$.
- $\mathcal{H}_1^{\text{st2}}$: $A_1 = 1$, $\mathcal{P}_{1,i}(1) = \mathcal{L}$. Note that this heuristic does not satisfy the condition for Nash stability.
- $\mathcal{H}_1^{\text{st3}}$: $A_1 = 1$ and $\mathcal{P}_{1,i}(1) = \{i\}$.

We expect that the most complex, $\mathcal{H}_1^{\text{st1}}$, leads to the best results.

3.3.2.2 In procedure \mathbf{P}_2

We note the heuristic $\mathcal{H}_2^{\text{st}}$ that returns the set $\{\mathcal{P}_{2,i}(a)\}_{a=1}^{A_2}$. It is defined as: $A_2 = 1$ and $\mathcal{P}_{2,i}(1) = \mathcal{L}$.

3.4 Clustering algorithm for unstructured mobile ad hoc networks

In this section we specify the generic clustering algorithm to the unstructured network case by designing dedicated utility function and heuristics.

3.4.1 Utility function

In unstructured networks, we target clusters offering good throughput for intra-cluster communications. This can be obtained by gathering nodes with high link capacities between each other. We refer this goal as \mathbf{G}_3 :

Goal 3 (\mathbf{G}_3) Build clusters with high link capacities.

Let us define $\kappa(i, j) := \log_2(1 + \Gamma(i, j))$ the *capacity* of link (i, j) , with $\Gamma(i, j)$ the SNR at node j when node i transmits. $\Gamma(i, j)$ can be either the instantaneous SNR in Gaussian channels, or the average SNR in random channels. In order to achieve \mathbf{G}_3 , we propose to define the utility function as the sum capacity of all the intra-cluster links:

$$u_{\text{un}}(\mathcal{C}_k) := \sum_{i \in \mathcal{C}_k} \sum_{j \in \mathcal{C}_k | (i, j) \in \mathcal{E}} \kappa(i, j). \quad (3.10)$$

This utility function also achieves \mathbf{G}_1 since it holds the following result:

Result 3.7 The utility function defined in (3.10) satisfies Condition 3.1.

Proof See Appendix B.6. ■

Notice that using (3.10), the clustering problem can be modeled as a non-transferable utility coalitional game in which the value of a coalition cannot be divided in any manner among the members of the coalition.

3.4.2 Heuristics for node selection

3.4.2.1 \mathcal{H}_1 in procedure \mathbf{P}_1

We note $\mathcal{H}_1^{\text{un}}$ the corresponding heuristic that returns the set $\{\mathcal{P}_{1,i}(a)\}_{a=1}^{A_1}$. Since in unstructured networks there is not any particular reason to select any other node than itself, i.e., $\{i\}$, $\mathcal{H}_1^{\text{un}}$ is defined as follows: set $A_1 = 1$ with $\mathcal{P}_{1,i}(1) = \{i\}$.

3.4.2.2 \mathcal{H}_2 in procedure \mathbf{P}_2

We note $\mathcal{H}_2^{\text{un}}$ the corresponding heuristic that returns the set $\{\mathcal{P}_{2,i}(a)\}_{a=1}^{A_2}$. Here, at least one constraint in the selected cluster is not satisfied, e.g., connectivity or diameter. We identify three different situations for node $\{i\}$: *i*) it has a lot of neighbors and it could be valuable that it forms a new cluster with its neighbors, *ii*) it has very few neighbors in the cluster meaning that it is presumably very likely to be involved in the constraints violation. It would be then beneficial that it leaves the cluster, *iii*) it is not in case *(i)* or *(ii)* and we change nothing. Considering the extreme situations for *(i)* (the most connected) and *(ii)* (the least connected), we propose the following heuristic:

- If node i has the highest degree within its induced cluster subgraph, then set $A_2 = 1$ with $\mathcal{P}_{2,i}(1)$ defined as all cluster members that are in the $\lfloor d_{max}/2 \rfloor$ -hop neighborhood of i . When the diameter constraint is equal to two, $\mathcal{P}_{2,i}(1)$ includes node i and all its 1-hop neighbors within the cluster, i.e., $\mathcal{P}_{2,i}(1) = \{i\} \cup \{j \in \mathcal{C}_k | (i, j) \in \mathcal{E}\}$.
- If node i has the lowest degree within its induced cluster subgraph, then set $A_2 = 1$ with $\mathcal{P}_{2,i}(1) = \{i\}$.
- In the other cases, node i chooses to remain in the cluster and to wait for the action of other cluster members, and $A_2 = 0$.

Notice that there are as many choices for the heuristics as one can imagine. The heuristics proposed here can thus be changed. However, the heuristic selection may take into account the computation complexity.

3.5 Numerical results

In this section we simulate numerically our proposed algorithms (COG and CLQ) and analyze them deeply. In case of structured ad hoc networks, COG is compared to the naive algorithm 1G1C defined in Section 1.5.1, whose goal is to force all members of a group to be in the same cluster as soon as the constraints can be satisfied. If not, simple mechanisms are carried out to satisfy the constraints. In case of unstructured ad hoc networks, CLQ is compared to three standard clustering algorithms entitled LCC [7], VOTE [9], and SECA [13]. Static and mobile configurations are tested.

3.5.1 Simulation setup

The nodes are deployed randomly in a 1.5 km x 1.5 km square area. The node deployment and mobility models are explained in Sections 3.5.2 and 3.5.3 for the structured and the unstructured networks, respectively. No radio communication is possible between two nodes i and j separated by a distance $d_{i,j}$ greater than the radio range $d_{\text{ref}} = 250$ m. The SNR associated with the link (i, j) is given by $\Gamma(i, j) := -40 \log(d_{i,j}/d_{\text{ref}})$. The diameter constraint on the cluster is $d_{max} = 2$. All the results are obtained through averaging over 100 different random networks. The simulation duration is fixed to 5000 s.

3.5.1.1 Distributed asynchronism model

We specify here the way distributed and asynchronous decision-making is handled in the simulation. Remember that a node $i \in \mathcal{C}_k$ can be either in a *Waiting* or *Busy* (i.e., not available) state. According to these states, the simulation behaves as the following:

- **Waiting state:** node i is not involved in current operations done by the clustering algorithm. However, it is learning information needed to run the clustering algorithm. It stays in this state during δ_w s which is randomly chosen according to an exponential distribution with parameter λ . It may leave this state in the two following cases:

1. After δ_w s, it checks its cluster constraints. Then two cases may occur:

- (a) If the constraints are satisfied, then it applies the procedure \mathbf{P}_1 of the clustering algorithm described in Table 3.1. If it decides to move to another cluster, it then switches to the state *Busy* to account for the time needed to operate this change.
- (b) If the constraints are not satisfied, then it applies the procedure \mathbf{P}_2 of the clustering algorithm described in Table 3.2. If it decides to perform a change in the cluster it then switches to the state *Busy* to account for the time needed to operate this change.

2. When the node i becomes involved in a cluster modification decided by another node j .

- **Busy state:** it means that the node is involved in a cluster modification initiated by itself or another node. The duration of this state is deterministic and set equal to δ_b s in order to account for the time spent to exchange information between nodes inside the cluster. Notice that if nodes involved in this cluster modification are in the *Busy* state (because already involved in other cluster modifications simultaneously), then this cluster modification is canceled and the node i goes back to the *Waiting* state.

The above description can be modeled as a state machine as depicted in Fig. 3.1. Notice that we assume that the time required for running the procedures \mathbf{P}_1 and \mathbf{P}_2 is zero. Unless otherwise stated, $\lambda = 5$ and $\delta_b = 0.5$.

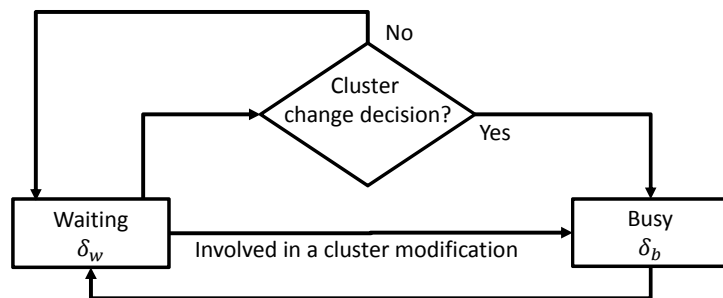


Figure 3.1: State machine model for asynchronous clustering algorithm.

3.5.2 Case of structured ad hoc networks

In this section we present the simulation results associated with COG. We set the group size $m_t = 10$. We denote by COG n the algorithm when the cluster size constraint is $n_{max} = n$.

Unless specified otherwise, the heuristic used in Table 3.1 is \mathcal{H}_1^{st1} . The maximum number of groups T is equal to 100, when $N = 1000$. Consequently, using Result 3.6, we set $\epsilon = 10^{-5} < \epsilon^* = 1/(1 + T) \simeq 9.910^{-3}$ in (3.5), to favor \mathbf{G}_2 over \mathbf{G}_1 .

In structured networks, the nodes are deployed randomly like described in Section 2.6.3.1. To better visualize the type of networks we consider here, Table 3.3 details the average node degrees vs. N . The network density ranges from low/medium (in 100 node networks) to high (in 1000 node networks) values.

N	100	200	300	400	600	800	1000
Degree	19.9	33.58	47.51	61.58	89.72	117.8	145.7

Table 3.3: Average node degree vs. N in structured networks.

3.5.2.1 Static networks

First let us study the case when nodes are static. In this section we thus assess the performance of procedure \mathbf{P}_1 in Table 3.1.

3.5.2.1.1 Cluster-based vs. node-based clustering algorithm execution

The two algorithms DCOG from Chapter 2 and COG are very close in their definition. A noticeable difference lies in the entity running the algorithm. In the case of DCOG the decision making is done at cluster level, when COG is executed by each individual node. It is thus legitimate to wonder about the performance difference between them. To make both algorithms comparable we modified the DCOG simulation to use the cost function $1 - u_{st}(\mathcal{C}_k)$ instead of (2.13), and ran 100 simulations of static random networks with $N = 100$ nodes and $n_{max} = 20$. Fig. 3.2.a plots the values of the network cost function J_0^X defined in Chapter 1 and achieved by both algorithms, with parameters $\alpha = 4$, $\hat{\gamma} = 1$ and $\tilde{\gamma} = 2$. In Fig. 3.2.b we plot the differences between these values. Both cluster-based and node-based approaches lead to very close results, the difference being in $[-3.9\%, 4.2\%]$.

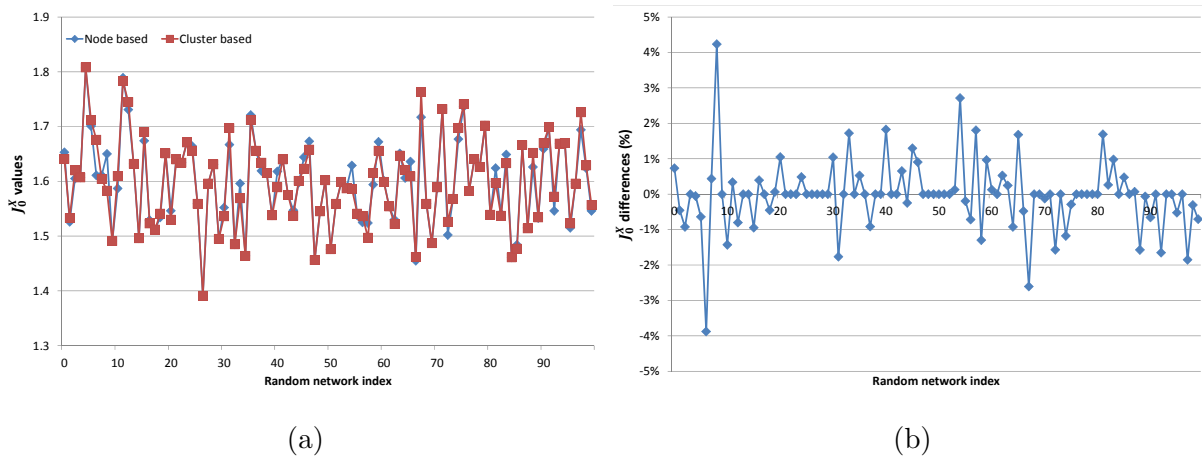


Figure 3.2: Cluster-based DCOG and node-based COG 20 algorithms in 100 structured networks with $N = 100$. (a) J_0^X values. (b) J_0^X value differences.

3.5.2.1.2 Technical metrics: group cluster diversity, cluster size, singletons and cluster group diversity.

In our random networks, the induced subgraphs of the groups do not always satisfy the topology constraints, thus forcing these groups to be split between multiple clusters, and thus leading to GCD (defined in Section 2.6.1) values greater than 1, as shown in Fig. 3.3. For all clustering algorithms, the GCD is only slightly greater than 1, meaning that each group is usually included in a single cluster and confirming that COG achieves \mathbf{G}_2 .

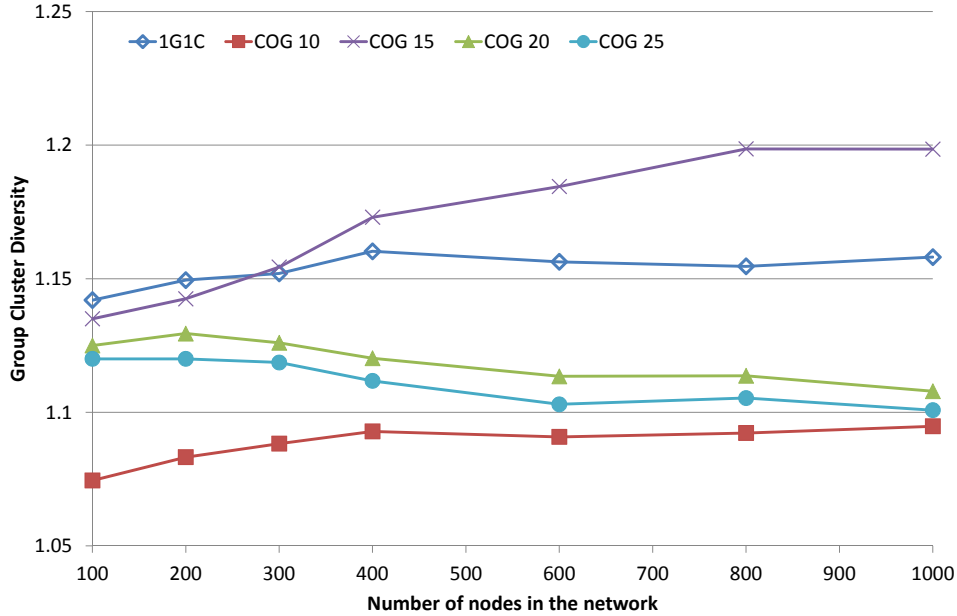


Figure 3.3: Group cluster diversity vs. N in static structured networks.

Fig. 3.4 shows the average cluster size. The curves for 1G1C and COG 10 are nearly the same, and they both build clusters smaller in average than the group size. This is due to topology constraints which sometimes prevent that all members of the same groups be inserted in the same cluster. Additionally COG 10 builds slightly larger clusters than 1G1C. During the distributed process of COG, several switch operations are required to aggregate the members of a group in a single cluster. Depending on the order in which the different nodes perform their switch operations, the constraints may prevent the formation of clusters with full groups and lead to a larger number of clusters, i.e., a smaller average cluster size. To grasp the details of what happens let us consider the example of an 8 node network composed of two equal size groups $\mathcal{O}_1 := \{1, 2, 3, 4\}$ and $\mathcal{O}_2 := \{5, 6, 7, 8\}$, with a maximum cluster size equal to 4. In this example, unless specified otherwise, the cluster constraints are always satisfied. At time t_0 the partition $p(t_0)$ of this network is: $p(t_0) := \{\{1, 2, 3\}, \{4\}, \{5, 6, 7\}, \{8\}\}$. Let us suppose that *i*) at time $t_1 > t_0$ node 4 is the first to execute COG 4 after time t_0 and that *ii*) node 4 cannot join the other members of \mathcal{O}_1 because of the cluster diameter constraint. Then node 4 joins nodes $\{5, 6, 7\}$, members of \mathcal{O}_2 , leading to the partition $p(t_1) := \{\{1, 2, 3\}, \{4, 5, 6, 7\}, \{8\}\}$. If node 8 at $t_2 > t_1$ is the first to execute COG 4 after time t_1 , it will not join the other members of \mathcal{O}_2 because they already belong to a maximum size cluster. Thus node 8 will join nodes $\{1, 2, 3\}$ and the final partition will be $p(t_2) := \{\{1, 2, 3, 8\}, \{4, 5, 6, 7\}\}$. In $p(t_2)$ the average cluster size is 4. Conversely, during the centralized process of 1G1C, as long as the constraints are satisfied

all members of the same group are always inserted in the same cluster. In the same example, 1G1C builds the partition $p := \{\{1, 2, 3\}, \{4\}, \{5, 6, 7, 8\}\}$, because node 4 cannot be in the same cluster as $\{1, 2, 3\}$ due to the cluster diameter constraint. In p the average cluster size is equal to 2.67, smaller than the one in $p(t_2)$.

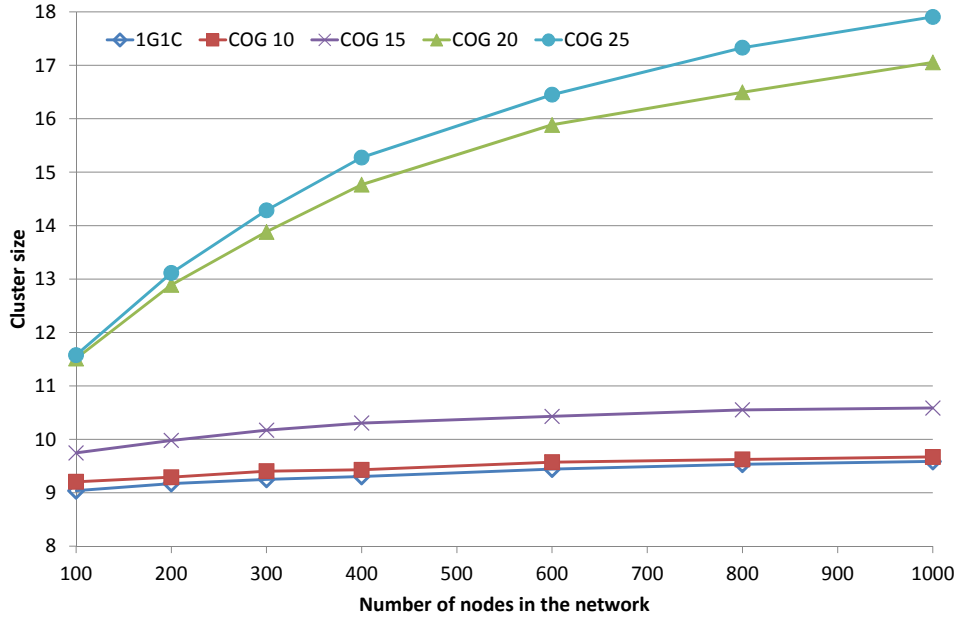


Figure 3.4: Cluster size vs. N in static structured networks.

Increasing the maximum cluster size to 15 does not allow to include two whole groups in a cluster. Therefore, due to our choice of ϵ which favors \mathbf{G}_2 over \mathbf{G}_1 , we expect COG 15 to build clusters whose average size are similar to the ones achieved by COG 10. This is the case, even if COG 15 builds clusters whose size exceeds the one of COG 10 by a number in $[0.5, 1.0]$. This is justified by the capacity of COG 15 to "catch" the singleton clusters, i.e., merge them with clusters that already include one full group. To illustrate this capability, let us come back to the previous example, at time t_1 : $p(t_1) := \{\{1, 2, 3\}, \{4, 5, 6, 7\}, \{8\}\}$. Let us consider what happens with COG 4 and COG 6 at time t_2 if $\{1, 2, 3, 8\}$ does not satisfy the constraints:

- **COG 4:** Node 8 remains alone and the final partition is $p_{\text{COG4}} := \{\{1, 2, 3\}, \{4\}, \{5, 6, 7, 8\}\}$, with average cluster size 2.67 and 1 singleton.
- **COG 6:** Node 8 joins nodes $\{4, 5, 6, 7\}$ and the final partition is $p_{\text{COG6}} := \{\{1, 2, 3\}, \{4, 5, 6, 7, 8\}\}$, with average cluster size 4 and no singleton.

When the number of nodes N increases, the network topology becomes denser, and the probability that COG builds clusters with two full groups becomes larger, thus achieving \mathbf{G}_1 and \mathbf{G}_2 . This is the reason why the average cluster size curves for COG 20 and COG 25 are increasing functions of N . The same explanations as for COG 10 and COG 15 justifies why COG 25 builds cluster whose average size is greater than the one of COG 20.

The average number of singletons is plotted in Fig. 3.5.a. The centralized simple heuristic 1G1C builds the highest number of singletons. More importantly, these curves confirm the argument made about the cluster size metric: COG 15 and COG 25 minimize this number. Thus, to reduce the number of singletons the value of n_{max} should be set such as to be slightly greater than a multiple of the group size.

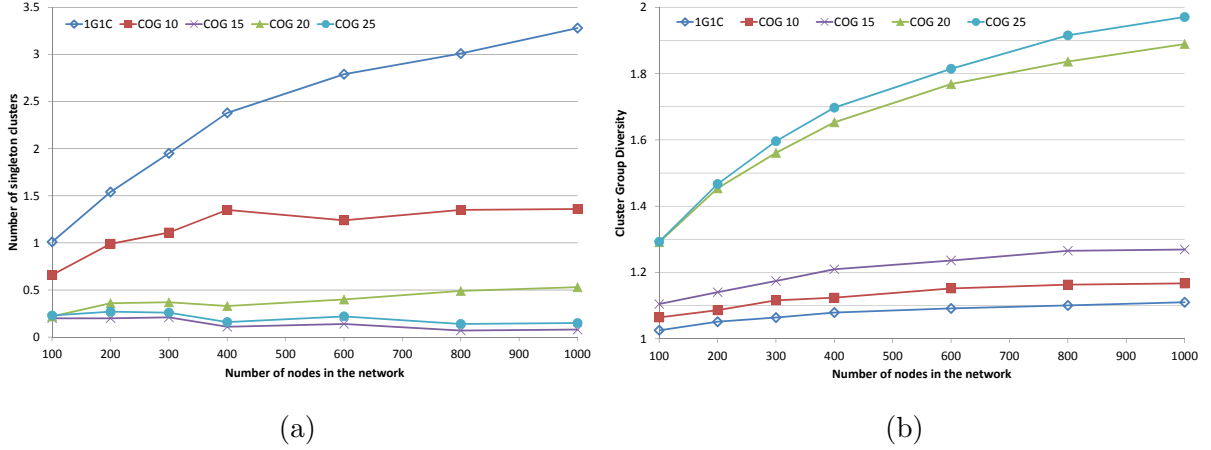


Figure 3.5: Static structured networks. (a) Number of singletons vs. N . (b) CGD vs. N .

The cluster group diversity (CGD) is the average per cluster number of groups with at least one member within the cluster. It appears that the curves of CGD in Fig. 3.5.b look similar to the ones showing the average cluster size in Fig. 3.4. Within a cluster, the cluster size divided by the group size and multiplied by the GCD is equal to the CGD. We know that the GCD is close to 1 and that the group size is equal to 10. Thus Fig. 3.5.b may be seen as reduction of Fig. 3.4 by a factor of 10.

3.5.2.1.3 Application level performance

Fig. 3.6 shows the application level performance measured with J_0^X for COG 10, COG 15, COG 20 and COG 25 with heuristic \mathcal{H}_1^{st1} (like for all simulation results in this section).

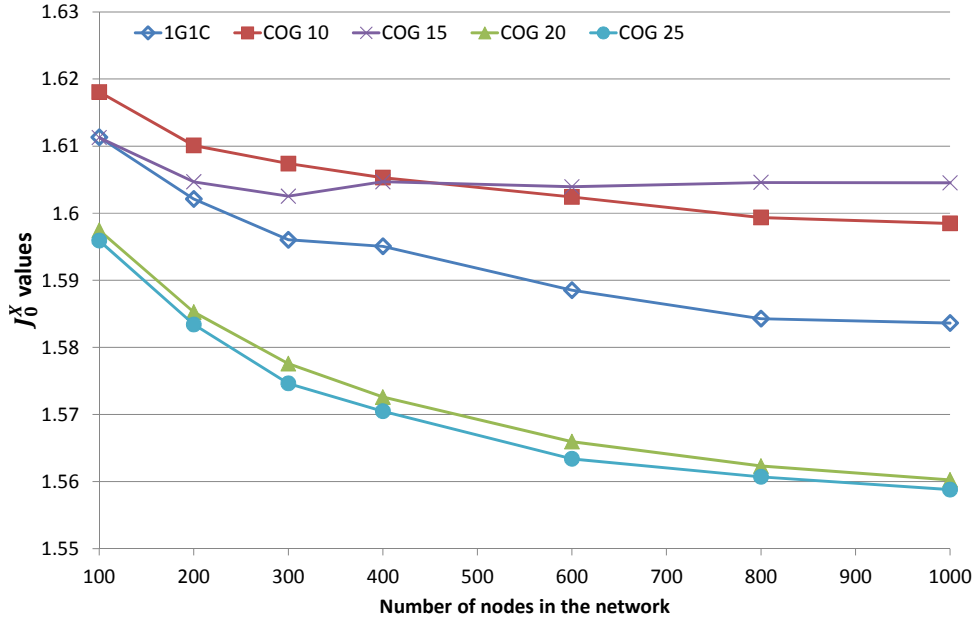


Figure 3.6: J_0^X vs. N in static structured networks.

Whatever the maximum cluster size, COG achieves J_0^X values close to the ones of 1G1C. The justification of the slightly lower performance of COG 10 and COG 15 is the same as discussed in Section 3.5.2.1.2: it is due to the COG distributed execution dynamic. When the

cluster can include two entire groups, then the J_0^X values associated with COG are smaller, and thus better. This small gain, in $[0.014, 0.025]$, results from *i*) a reduction of inter-cluster traffic between members of different groups now in the same cluster, and *ii*) the reduction of the number of inter-cluster links used by communications between members of different groups residing in different clusters. To investigate further the reduction in number of inter-cluster links, let p_π^A the number of paths with π inter-cluster links in the partition formed by algorithm A. Let us define $r_\pi^A := 1 - p_\pi^A/p_\pi^{1G1C}$, the decrease in the number of paths with π inter-cluster links achieved by algorithm A w.r.t. 1G1C. Fig. 3.7 plots the values of r_π^A achieved with the algorithms COG 10, COG 15, COG 20 and COG 25, within $N = 300$ node networks. This figure confirms our expectations. When two groups can be included in a single cluster (A is COG 20 or COG 25), then *i*) the number of paths with more than two inter-cluster links is decreased (by more than 30%), and logically *ii*) the number of paths with zero or one such link increases (by 1700% and 1800%, i.e., a multiplicative factor p_0^A/p_0^{1G1C} close to 20).

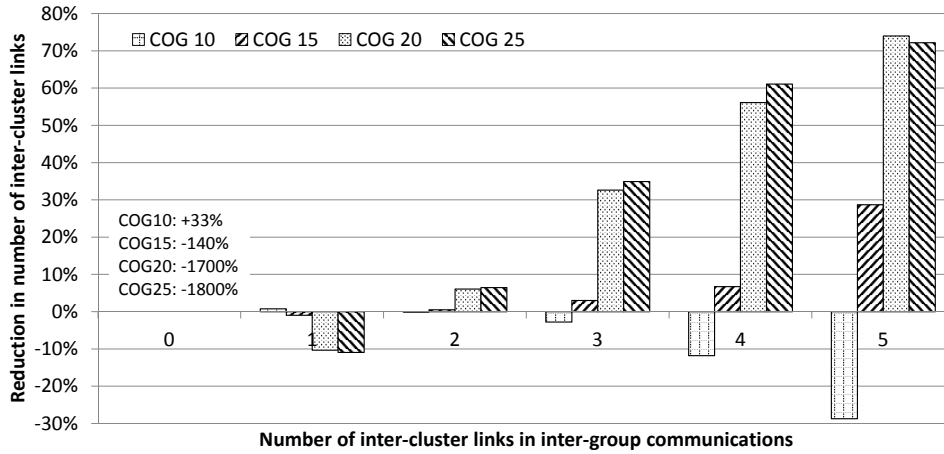


Figure 3.7: Reduction in the number of inter-cluster links for inter group communications vs. 1G1C in static structured networks ($N = 300$).

In Fig. 3.8 we plot the values of J_0^X for COG 25 with heuristics \mathcal{H}_1^{st1} , \mathcal{H}_1^{st2} and \mathcal{H}_1^{st3} , along with the ones for 1G1C. The best results are achieved thanks to \mathcal{H}_1^{st1} . It makes sense given that the output of \mathcal{H}_1^{st1} includes the ones of \mathcal{H}_1^{st2} and \mathcal{H}_1^{st3} . It is interesting to note that \mathcal{H}_1^{st3} and 1G1C curves are nearly the same. An analysis of the clusters formed with \mathcal{H}_1^{st3} reveals that they are composed of only one group, like 1G1C, COG 10 and COG 15 with \mathcal{H}_1^{st1} . Thus using \mathcal{H}_1^{st3} does not allow to achieve \mathbf{G}_1 . To compare \mathcal{H}_1^{st1} and \mathcal{H}_1^{st2} , let us now analyze the complexity of COG.

3.5.2.1.4 Complexity

The computational complexity of COG is related to the number of operations in the double-loop of lines 3-9 and 4-8 in Table 3.1, i.e., the number of $\sigma_{k,\ell}(\mathcal{P}_{1,i}(a))$ gain calculations. The most complex operation when calculating a switch operation gain is to check if the diameter constraint is satisfied, which may require as many breadth first searches¹ (BFS) as cluster members. When running the BFS algorithm on a graph, for each vertex there is one operation per neighbor of this vertex. We denote each such operation by "BFS operation". Fig. 3.9 shows the number of

¹Breadth first search is a simple graph theory algorithm.

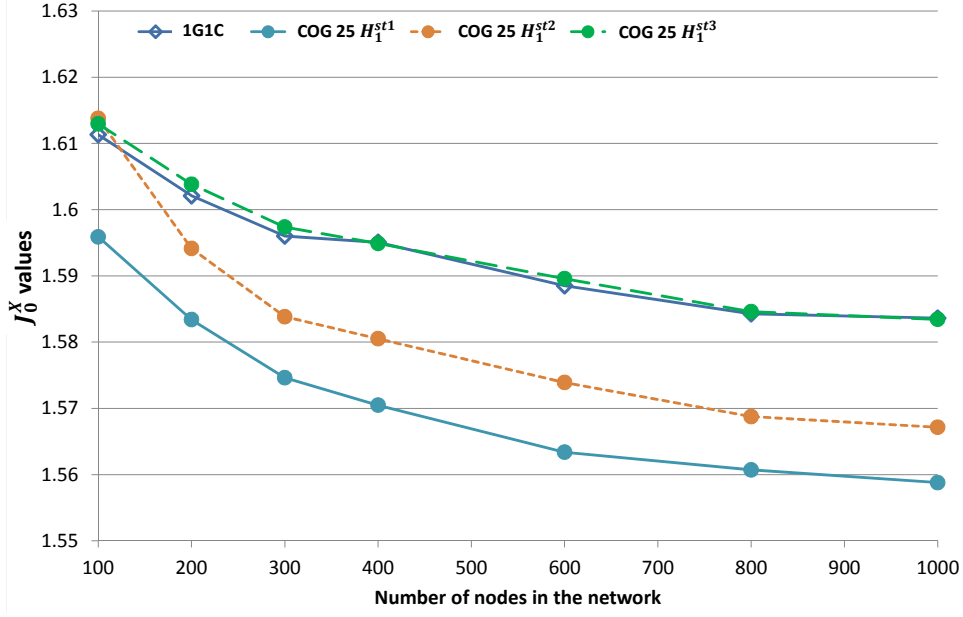


Figure 3.8: J_0^X for COG 25 with \mathcal{H}_1^{st1} , \mathcal{H}_1^{st2} and \mathcal{H}_1^{st3} vs. N in static structured networks.

BFS operations for the COG 10, COG 15, COG 20 and COG 25 with heuristic \mathcal{H}_1^{st1} (like for all simulation results in this section), and for COG 25 with \mathcal{H}_1^{st2} and \mathcal{H}_1^{st3} .

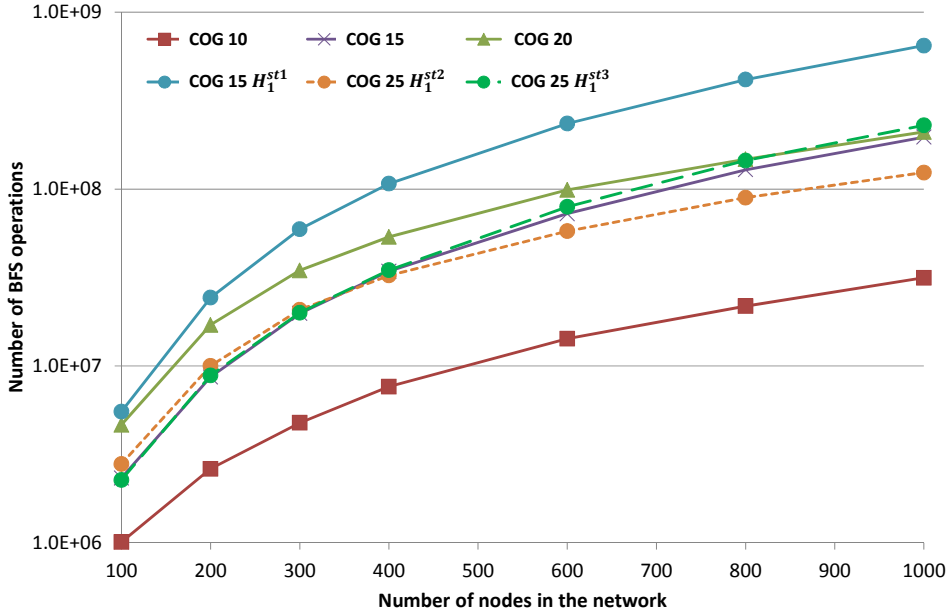


Figure 3.9: Number of breadth first search operations vs. N in static structured networks.

As expected, the number of BFS operations consistently increases with N , and with n_{max} . It is more fruitful to compare the complexity of the three heuristics used with COG 25. The one inducing the highest processing load is logically \mathcal{H}_1^{st1} . Remembering that in Fig. 3.8, the performance of \mathcal{H}_1^{st1} is the best, we confirm that achieving the best performance induces the largest complexity. Additionally, Fig. 3.8 and Fig. 3.9 show that \mathcal{H}_1^{st2} leads to less BFS operations than \mathcal{H}_1^{st1} , at the price of a decreased J_0^X performance. This illustrates the trade off between complexity and performance in the selection of the heuristics.

3.5.2.2 Mobile networks

We now consider networks with moving nodes. In this section, we thus assess the performance of both procedure \mathbf{P}_1 in Table 3.1 and \mathbf{P}_2 in Table 3.2.

In mobile networks, the duration of the simulation warmup phase must be chosen carefully. We measure the variations of the node average degree, plotted in Fig. 3.10. In view of this figure we set the duration of the warmup period to 500 s.

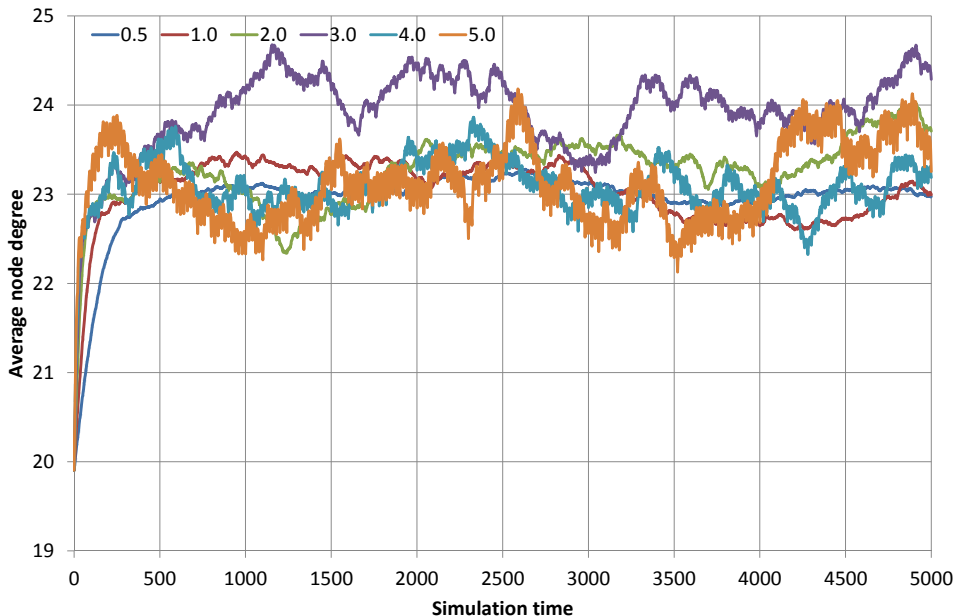


Figure 3.10: Average node degree vs. time for structured networks with $N = 100$, and for various v_{max} values.

In Fig. 3.11, we plot the cluster life time versus the maximum speed of nodes for different values of n_{max} . When n_{max} is smaller than twice the group size, the clusters are composed of a single group and so are very stable: the clusters formed at the beginning of the simulation are nearly never modified. This is an expected consequence of our node deployment scheme which ensures that within a group, the nodes are almost always at two radio hops. When n_{max} is at least equal to twice the group size, then some clusters include two groups. During the simulation, the groups move independently, leading to two kinds of cluster modifications: *i*) switch operations resulting from procedure \mathbf{P}_1 , and *ii*) mobility adaptations performed during procedure \mathbf{P}_2 to enforce the cluster constraints. When the node speed increases, this number of modifications also increases, thus reducing the cluster life time.

Fig. 3.12 shows the number of cluster modifications, i.e., the switch operations (the 'COG xx SO' curves) and the mobility adaptations (the 'COG xx MA' curves). This figure confirms that the clusters formed by COG 10 and COG 15 are nearly never modified (this is not visible in Fig. 3.12 because of the difference of scale between the number of cluster modifications for COG 10 and COG 15 w.r.t. to COG 20 and COG 25). Let us now analyze the behavior of COG 20 and COG 25. During the permanent phase of our simulations, a switch operation happens when two clusters, each composed of one group, merge into a two-group cluster. Subsequently, because the groups move independently, a mobility adaptation occurs and usually the two-group cluster splits back into the two previous one-group clusters. When this happens the total number of

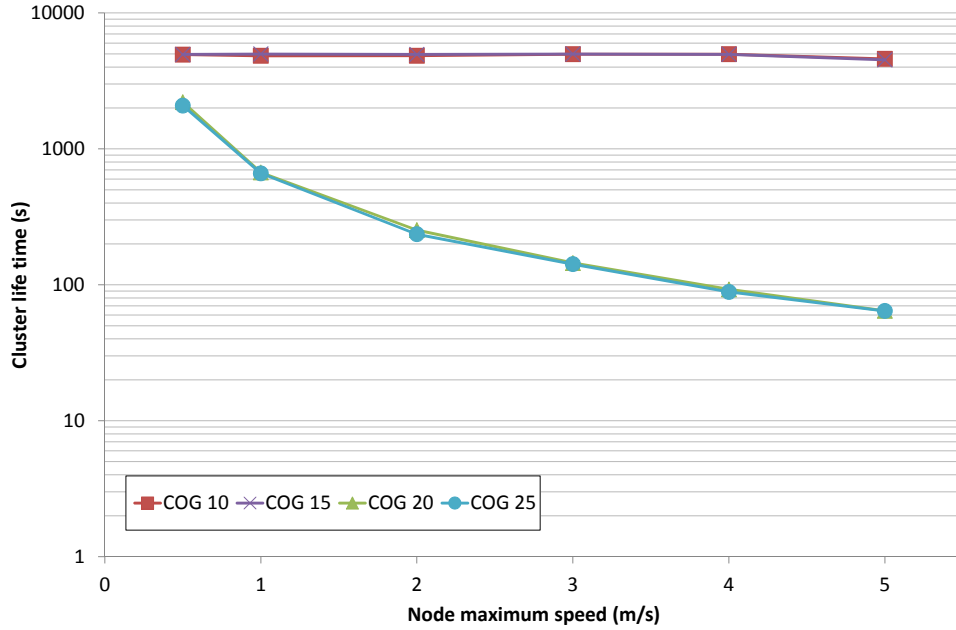


Figure 3.11: Cluster life time vs. maximum node speed in mobile structured networks.

clusters is increased by one, allowing a later switch operation to merge one-group clusters. In some cases a third cluster, composed of one group, is in the vicinity, and one group from the two-group cluster joins the third cluster. When this happens, the number of clusters is unchanged and no switch operation to merge one-group clusters happens, only a mobility adaptation to split them. This last situation happens more frequently when v_{max} increases, which explains why the number of mobility adaptations is slightly larger than the one of switch operations.

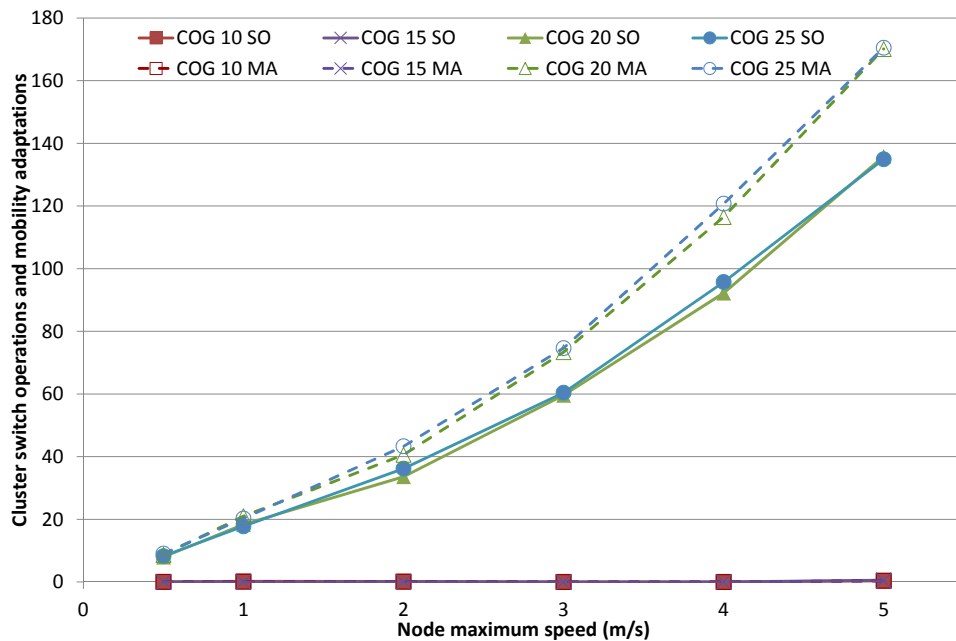


Figure 3.12: Number of cluster modifications vs. maximum node speed in mobile structured networks.

3.5.2.3 Conclusions on structured networks

Thanks to our simulations we verified that when the topology allows it, COG always form clusters composed of entire groups, thus achieving \mathbf{G}_2 and leading to excellent intra-group communication performance. When COG is configured to form clusters composed of only one single group, we showed that *i*) in static networks it leads to nearly the same clusters as the centralized heuristic 1G1C, and that *ii*) in presence of node mobility, COG clusters are stable. When multiple groups are allowed then COG also achieves \mathbf{G}_1 , improving inter-group communication performance, at the price of an increased complexity and of less stability in presence of mobility.

We also established that the transition from the DCOG algorithm, run at cluster level, to COG, run at node level, did not decrease the performance. We also assessed the effect of various heuristics for node selection, which allowed us to highlight the trade off existing between complexity and performance. These results are important milestones in the path to an implementation of a clustering protocol suited to structured networks.

As a conclusion, allowing clusters to include more than one group improves inter-group communications, at the expense of cluster stability. This should be done only when the network mobility is expected to be low. Conversely, if mobility is high then the maximum cluster size should be set to a value slightly larger than the group size. This guarantees cluster stability and reduces the number of singleton clusters.

3.5.3 Case of unstructured ad hoc networks

In this section, we consider the algorithm CLQ, and compare it to the following existing algorithms: the old but very well-known LCC [7], VOTE which forces the cluster size to be less than a target threshold [9], and the recent SECA which takes the link quality into account [13]. Unless otherwise stated, $n_{max} = 20$. The design parameters for SECA (defined in [13]) are $w_{cd} = 0.2$, $w_M = 0.4$, $w_{SL} = 0.4$ and $q_d = 0.1$. Also, SECA considers a radio link $(i, j) \in \mathcal{E}$ as being strong if $d_{i,j} < d_{ref}/2$.

The nodes are deployed randomly following a uniform distribution. When there is node mobility, the coordinates of any node i are updated once per second, following a uniform rectilinear motion with a speed limited to v_{max} . Each node moves between waypoints, whose coordinates are updated once every $MP = 20$ s. To better visualize the type of networks we consider in this section, Table 3.4 details the average node degrees vs. N . When compared to Table 3.3, Table 3.4 shows that the difference between average degrees in structured and unstructured networks is about 5. This gap is justified by the group model used for structured networks, which

N	100	200	300	400	600	800	1000
Degree	13.70	27.94	42.12	56.29	84.47	112.4	140.4

Table 3.4: Average node degree vs. N in unstructured networks.

places all members of the same group in a disk whose radius is twice the radio range d_{ref} , and thus increases the average node degree.

3.5.3.1 Static networks

First let us study the case when nodes are static. For illustration purpose, Fig. 3.13 depicts the result achieved by the different clustering algorithms for a 600 node network. In this figure the cluster membership is indicated by the color and shape of the nodes.

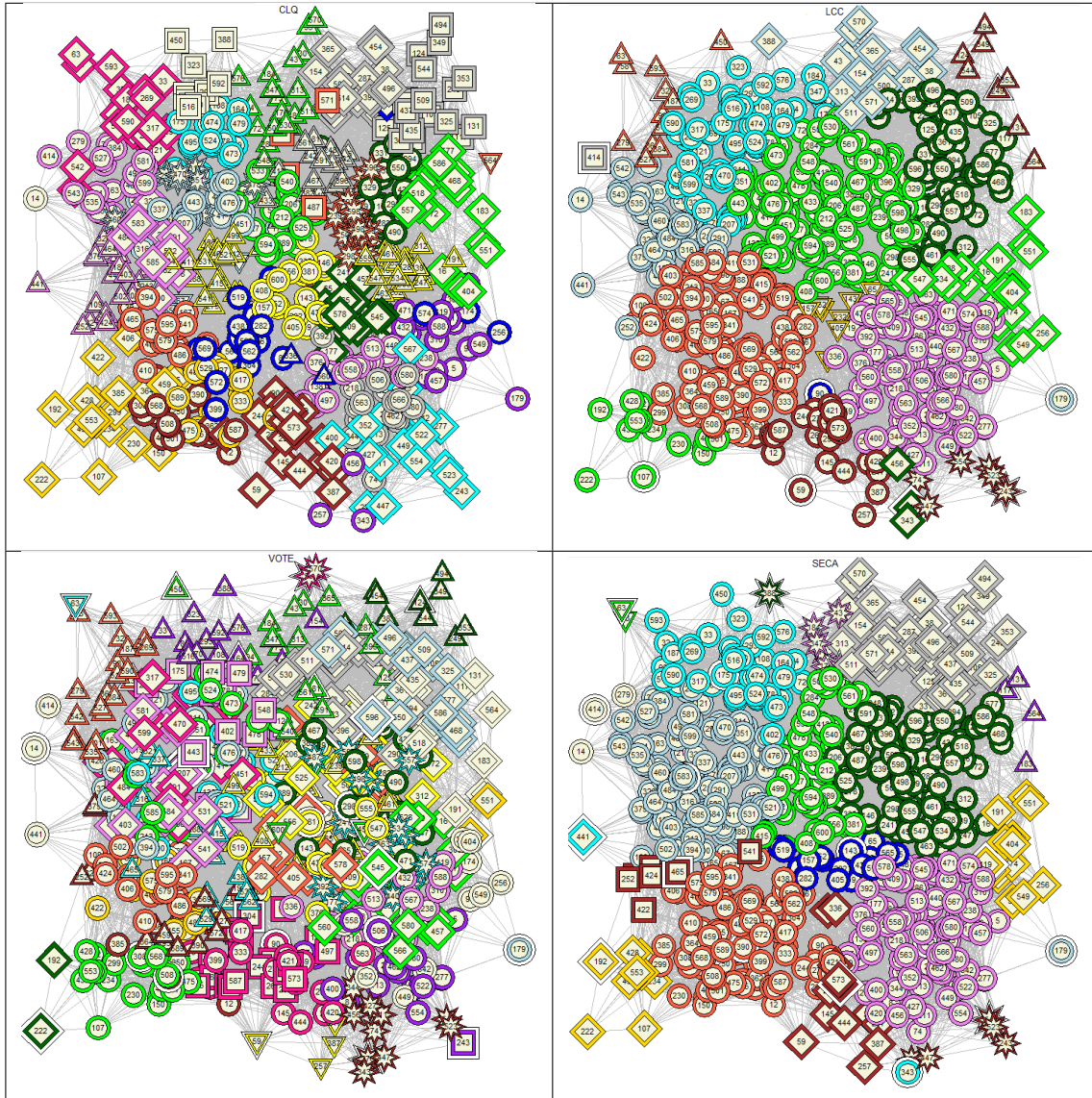


Figure 3.13: Example of 600 node network clustered with CLQ, LCC, VOTE and SECA (from left to right, top to bottom).

3.5.3.1.1 Duration required for CLQ convergence to a stable cluster structure

In Fig. 3.14.a, we plot the duration required to stabilize the cluster structure. The proposed coalition formation algorithm requires more time than the reference algorithms. This larger convergence duration is mainly due to the duration of the *Busy* state. Indeed, for LCC, VOTE, and SECA, this duration has been assumed to be zero since the clustering protocol can be easily done thanks to messages directly exchanged between the concerned cluster member and its already-chosen CH. With CLQ there is no CH, which requires the nodes to perform peer-to-peer message exchanges, leading to longer delays. This delays are modeled by the δ_b long *Busy* state.

Fig. 3.14.b plots the numbers of switch operations that are *i*) performed in line 13 of Table 3.1, or *ii*) canceled because in line 12 some nodes involved in $\sigma_{k,\ell^*}(\mathcal{P}_{1,i}(a^*))$ are not available. For case (*i*), this number increases linearly with N , whereas for case (*ii*) it has a quadratic shape, as highlighted by the dotted black curve. This last result can be interpreted in the following manner: when the node density increases, the probability of collision, i.e., that some nodes involved in the switch operation are not available, also increases. It is a well known property of the MAC based on random access which we have taken into account with our *Waiting-Busy* model.

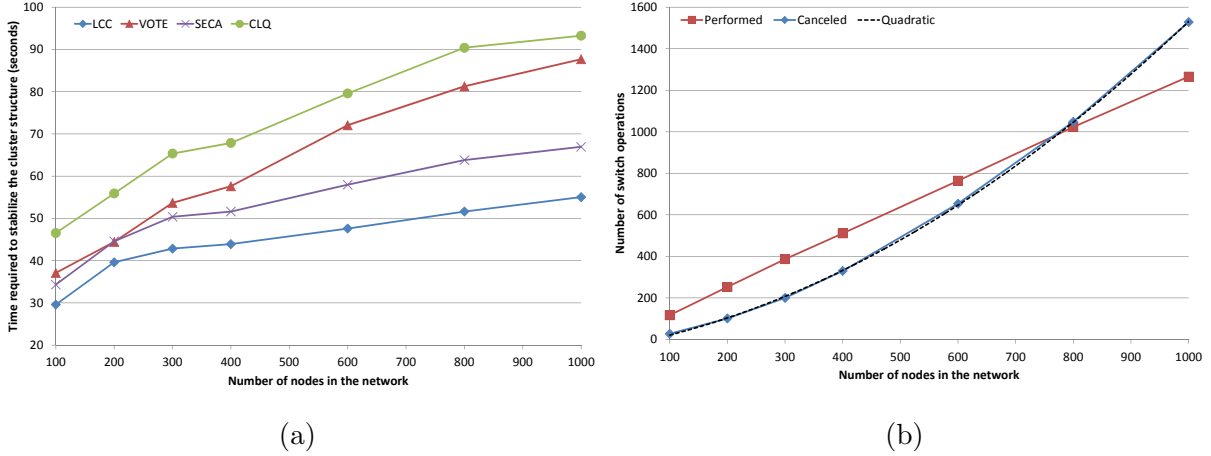


Figure 3.14: Static unstructured networks. (a) Time required to reach a stable cluster structure vs. N . (b) Number of performed and canceled switch operations vs. N .

In order to assess the influence of the parameter δ_b on CLQ, we plot Fig. 3.15 which shows the time required by CLQ to reach stable clusters and the number of performed and canceled switch operations in $N = 100$ networks with varying δ_b . Fig. 3.15.a shows the that duration

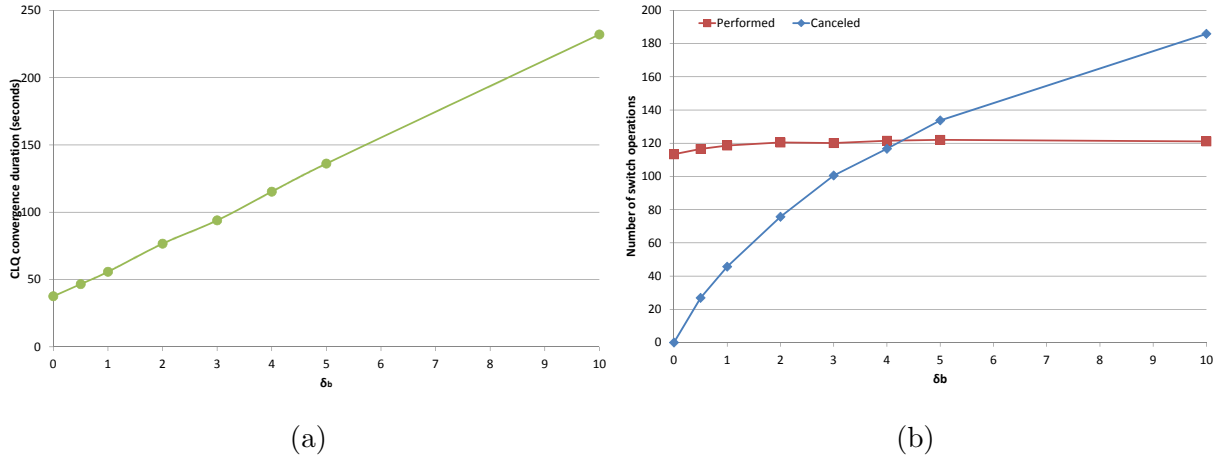


Figure 3.15: Static unstructured networks with $N = 100$. (a) Time required by CLQ to reach a stable cluster structure vs. δ_b . (b) Number of performed and canceled switch operations vs. δ_b .

needed by CLQ to reach stable clusters increases in a linear manner w.r.t. δ_b . Fig. 3.15.b shows that the number of performed switch operations remains more or less constant, independently of δ_b . The number of canceled switch operations increases abruptly, indicating that the number of collisions modeled by our model increases with δ_b . When designing the MAC, if we succeed in

decreasing δ_b (by optimizing the protocol), we will be able to obtain almost the same convergence duration as the existing algorithms plotted in Fig. 3.14.a.

3.5.3.1.2 Cluster size

In Fig. 3.16.a, we plot the cluster size vs. N . It is a non-decreasing function of the number of nodes in the networks. More precisely, the cluster size is limited to $n_{max} = 20$ for CLQ and VOTE, and increases linearly for LCC and SECA (that do not control the cluster size). The proportion of singleton clusters is shown in Fig. 3.16.b. The proposed CLQ builds the lowest number of singletons, which is of great interest since singleton clusters are inefficient for network performance. Table 3.5 details the proportion of n_{max} size cluster vs. N for the two algorithms VOTE and CLQ. CLQ builds the highest number of maximum size clusters, thus achieving \mathbf{G}_1 better than VOTE.

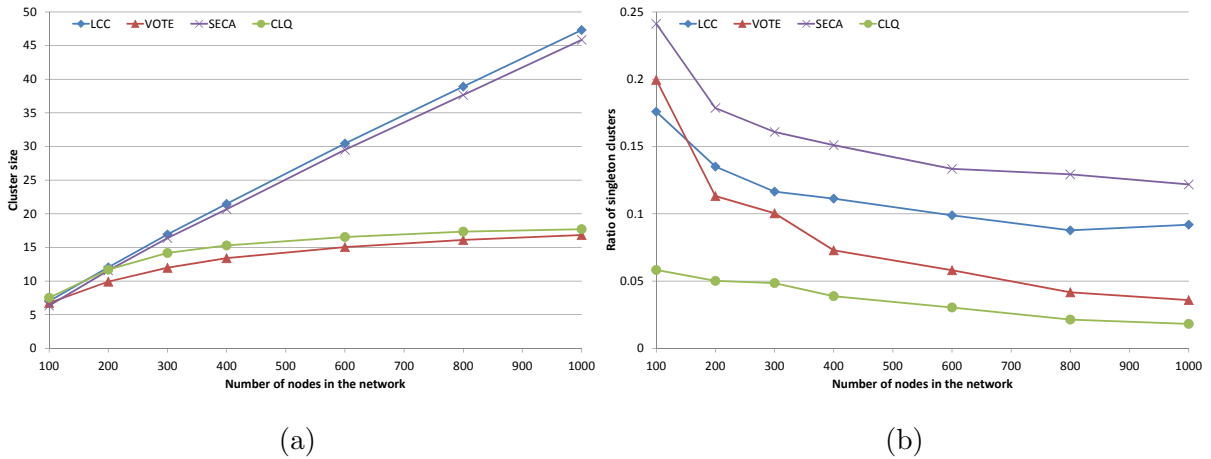


Figure 3.16: Static unstructured networks. (a) Cluster size vs. N . (b) Proportion of singleton clusters vs. N .

Nodes	100	200	300	400	600	800	1000
VOTE	8.71%	22.18%	33.93%	44.01%	56.65%	65.10%	70.87%
CLQ	4.15%	26.70%	46.00%	57.91%	71.68%	78.82%	81.87%

Table 3.5: Proportion of n_{max} size clusters vs. N in unstructured networks.

In Fig. 3.17, we show the distribution of the cluster size with $N = 300$. In this figure, the notation ' $>$ ' on the x axis means that the cluster size is strictly greater than 20. CLQ builds the smallest number of singletons and the largest number of clusters with n_{max} members. Moreover CLQ always satisfies the cluster size constraints ($n_{max} = 20$) whereas LCC and SECA lead to very large clusters. For example when $N = 1000$, the average size of the clusters formed by LCC is 47, with a lot of 200 node clusters. These clusters are not manageable anymore and contradict the objective of clustering.

3.5.3.1.3 Intra-cluster link capacities

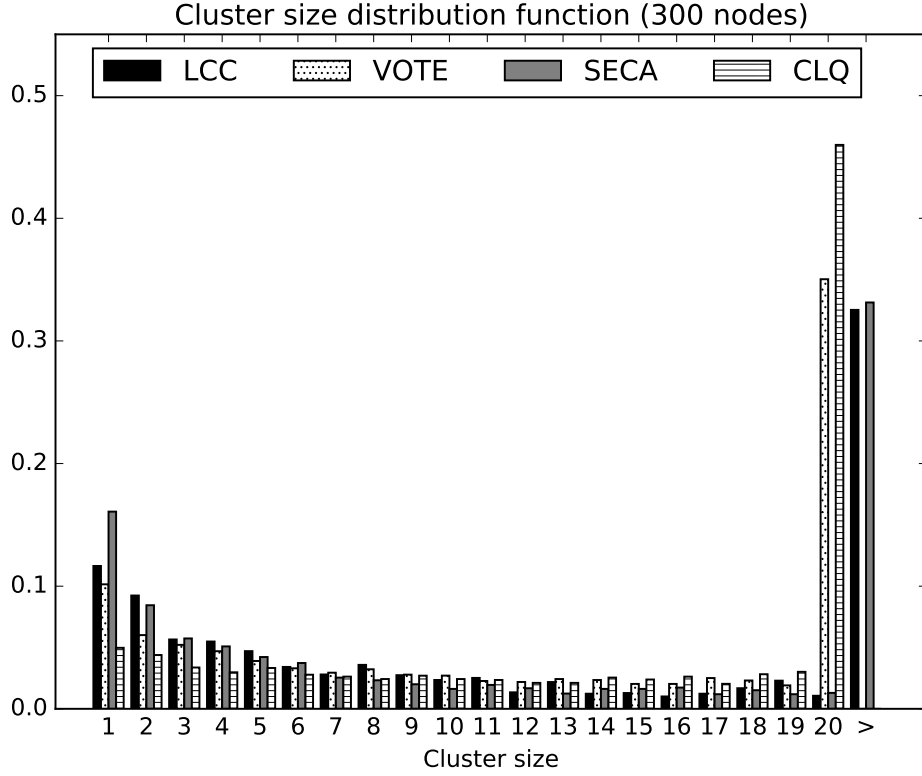


Figure 3.17: $p(|\mathcal{C}_k|)$ vs. $|\mathcal{C}_k|$ in static unstructured networks ($N = 300$).

In Fig. 3.18, we plot the intra-cluster link capacity vs. N . We observe that CLQ always ensures the highest link capacity thus attaining \mathbf{G}_3 .

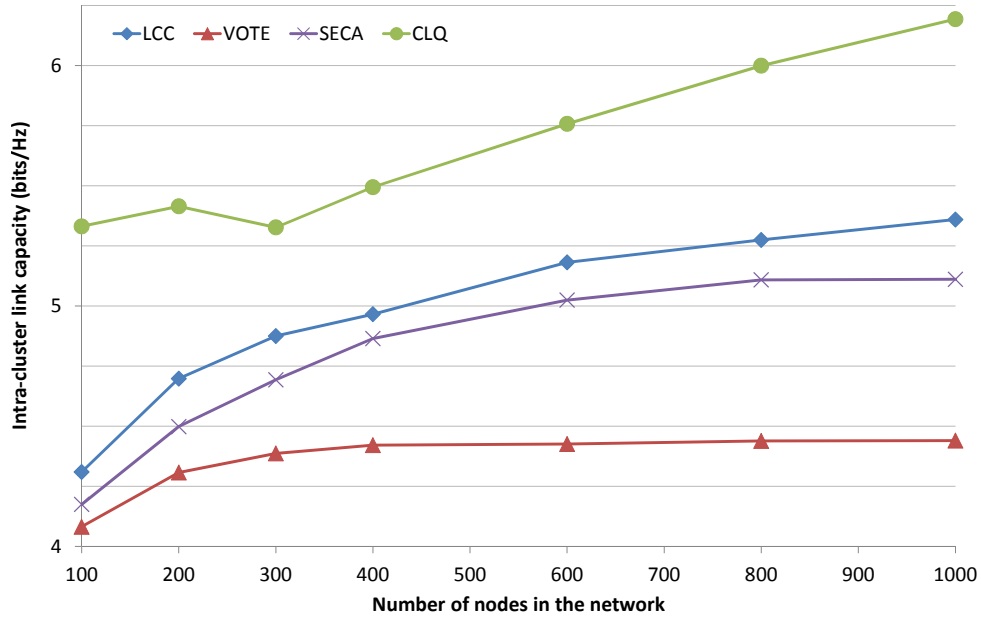


Figure 3.18: Intra-cluster link capacity vs. N in static unstructured networks.

While it does not take link capacities into account, LCC is the second best algorithm. The rationale of this unexpected result resides in the details on the metric calculation. The intra-cluster link capacity for a given number of nodes is the average of 100 such metrics, one per

random network. When calculating the metric for one network, it is first calculated per cluster and then it is averaged. During this last averaging, all clusters are considered equally, meaning that small and large clusters contribute at the same height, when the latter ones usually have links with a much lower capacity than the former. When the size of the clusters is widely spread, such as for LCC, the metric thus overestimates the real intra-cluster link capacity. The only algorithm from the state of the art designed to take link quality into account, SECA, achieves a slightly worse performance than LCC and thus fails to achieve good intra-cluster link capacity. Finally, VOTE is concerned only by the cluster size, leading to clusters whose members are spread on a large area. Two implications follow from this: firstly the intra-cluster link capacities are low, and secondly, there is a lot of cluster overlap.

3.5.3.2 Mobile networks

Let us now consider mobile networks. To determine the duration of the warmup phase we followed the same method as for structured networks. Fig. 3.19 and Fig. 3.20 plot the node average degree vs. time for $v_{max} \in \{0.25, 0.5, 0.75, 1, 1.5, 2, 3, 4\}$. In Fig. 3.20.b all curves remain within a small interval (whose width depends on v_{max}). Consequently $\forall v_{max}$, we set the duration of the warmup phase to 500 s, except for $v_{max} = 0.25$ for which we selected 1000 s.

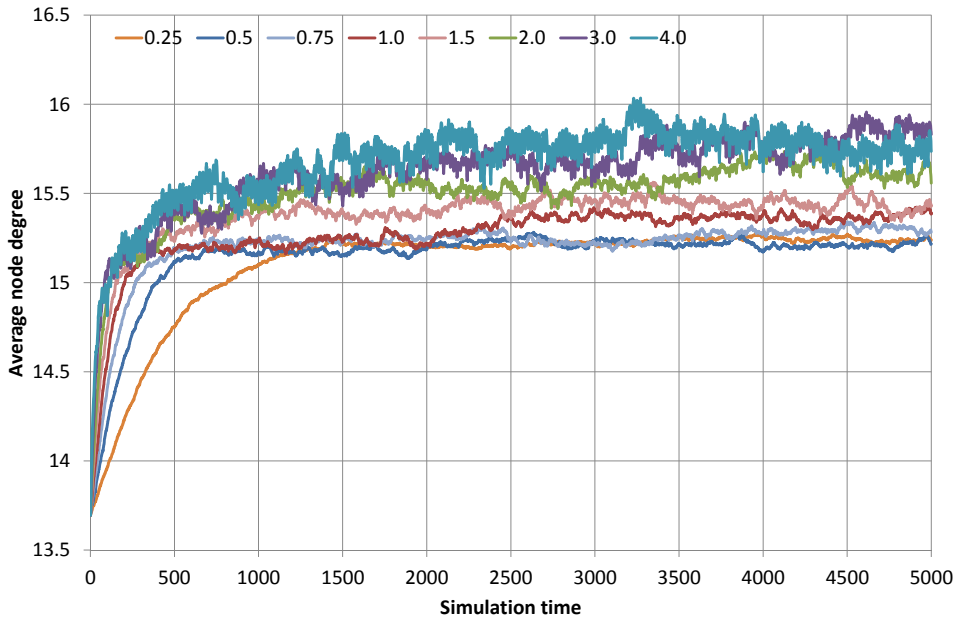


Figure 3.19: Average node degree vs. time for unstructured networks with $N = 100$, and for various v_{max} values.

We expected that small clusters would be more stable than large ones. Consequently we simulated CLQ and VOTE for two maximum cluster size: $n_{max} \in \{10, 20\}$. We denote by CLQ n and VOTE n when the cluster size constraint is $n_{max} = n$. The cluster life time vs. maximum node speed v_{max} is plotted in Fig. 3.21.

These curves first confirm that building smaller clusters improve their stability. The cluster life time associated with CLQ 10 is on average twice the one induced by CLQ 20 (this gain decreases with the node speed). VOTE also benefits from a smaller cluster size: when $n_{max} = 10$, then the cluster life time is on average 40% greater than if $n_{max} = 20$. Additionally, regardless

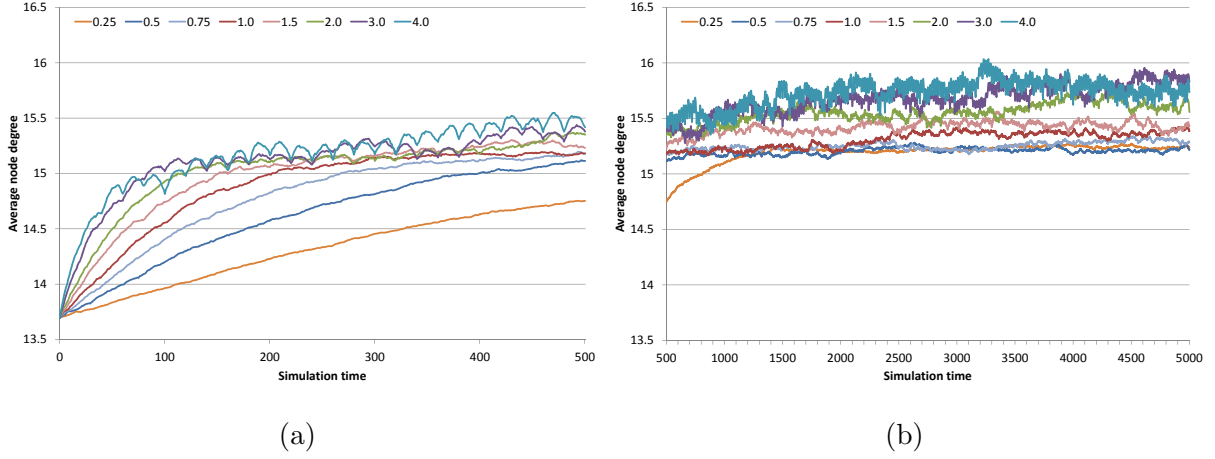


Figure 3.20: (a) Average degree during simulation transient state. (b) Average degree during simulation steady state.

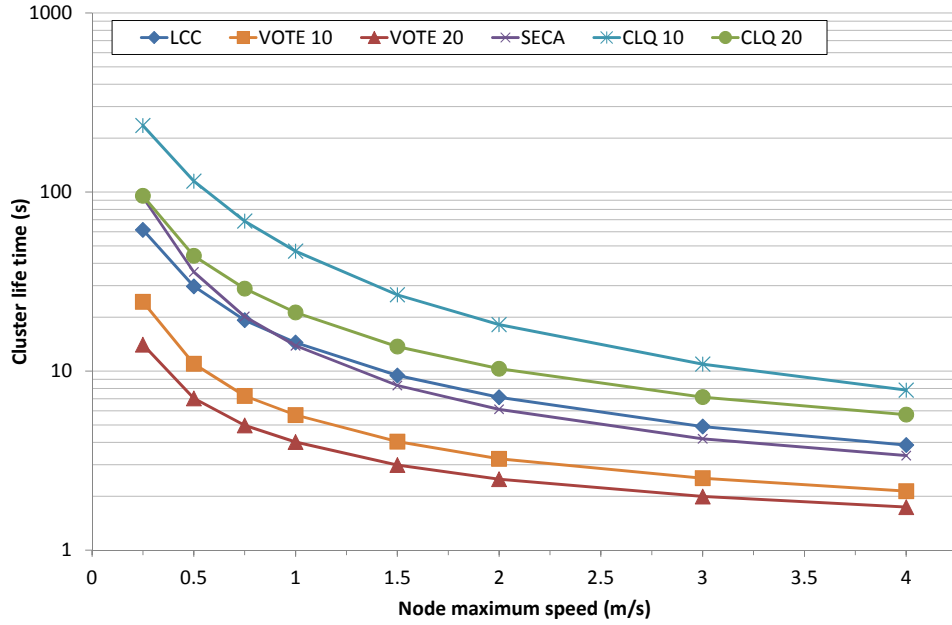


Figure 3.21: Cluster life time vs. maximum node speed in mobile structured networks.

of v_{max} , the most stable clusters are obtained thanks to CLQ. The algorithms LCC and SECA achieve similar performance, and VOTE 10 and VOTE 20 are the worst performers.

To better understand the hierarchy between the different CH-based algorithms, let us analyze the three major reasons that lead to a cluster modification: *i*) when a non-CH node becomes CH, *ii*) when a CH loses its CH state, and *iii*) when a non-CH node affiliates to a new CH. Fig. 3.22.a displays the number of times a node loses its CH state. This number is nearly the same (the difference is $< 0.6\%$) as the number of 'Becoming CH' events (because on average when a node loses its CH role, then another node becomes CH). Fig. 3.22.b plots the number of times when a non-CH node affiliates to a new CH.

LCC leads to the lowest number of cluster modification events thanks to the two following properties: *i*) the CH nodes are selected using a criterion that depends on the topology only marginally (a node becomes CH if its identifier is the lowest in its neighborhood), and *ii*) a CH

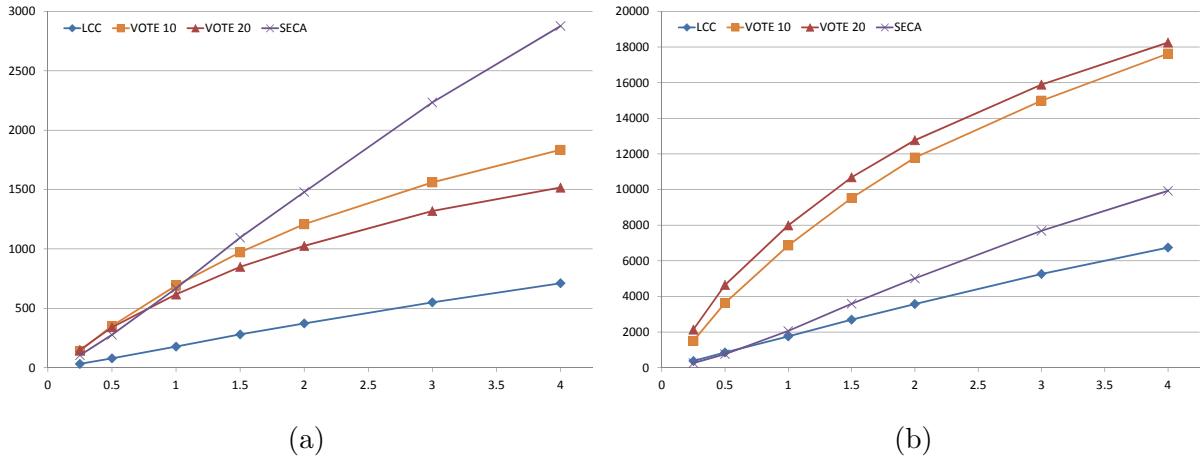


Figure 3.22: (a) Number of 'Losing CH' events vs. v_{max} . (b) Number of 'Changing CH' events vs. v_{max} .

node remains CH unless another CH becomes neighbor, in which case one of the two neighboring CH loses this state. In contrast, VOTE builds its clusters using a criterion strongly dependent of the topology: a node becomes CH if its degree is the highest one in its neighborhood. Thus a node easily gains or loses the CH state depending on the local changing topology, which also leads to a lot of non-CH nodes changing clusters. The last algorithm SECA, selects as CH the nodes with highest *quality value* defined as a combination of node degree, node mobility and link quality. As shown in Fig. 3.22.a, this criterion is even less stable than node degree only. However, SECA implements a mechanism to let a non-CH node become a CH only when its quality value exceeds, by a sufficient margin, the quality value of its current CH. This mechanism is a very effective stabilizing factor, as shown in Fig. 3.22.b, and justifies the better performance of SECA w.r.t. VOTE.

Coming back to CLQ, Fig. 3.23 plots the ratio of the mobility adaptations decided by procedures \mathbf{P}_2 to the number of switch operations decided by procedures \mathbf{P}_1 , showing that the former number is small when compared to the latter. This result is a consequence of the utility function (3.10). For example, on the one hand, when a node $i \in \mathcal{C}_k$ is moving away from the other members \mathcal{C}_k , the quality of its links with its neighbors in \mathcal{C}_k is reduced. On the other hand, if this node is also approaching from another cluster \mathcal{C}_ℓ , then the quality of its links with the \mathcal{C}_ℓ members is improved. Consequently, it is likely that before node i leads to a breach in the constraints in \mathcal{C}_k , procedure \mathbf{P}_1 finds a switch operations with strictly positive gain leading to the departure of node i from \mathcal{C}_k to join \mathcal{C}_ℓ . This is a positive property considering that the heuristic for node selection used by procedure \mathbf{P}_1 in unstructured networks leads to gentler cluster modifications than the one used by procedure \mathbf{P}_2 .

Therefore, the proposed algorithm which has been designed for building compact clusters (i.e., clusters whose links have a high capacity, thus involving nodes close to each other) that are robust to mobility, achieves the best performance.

3.5.3.3 Conclusions on unstructured networks

We compared the proposed algorithm CLQ with the three clustering schemes LCC, VOTE and SECA from the literature.

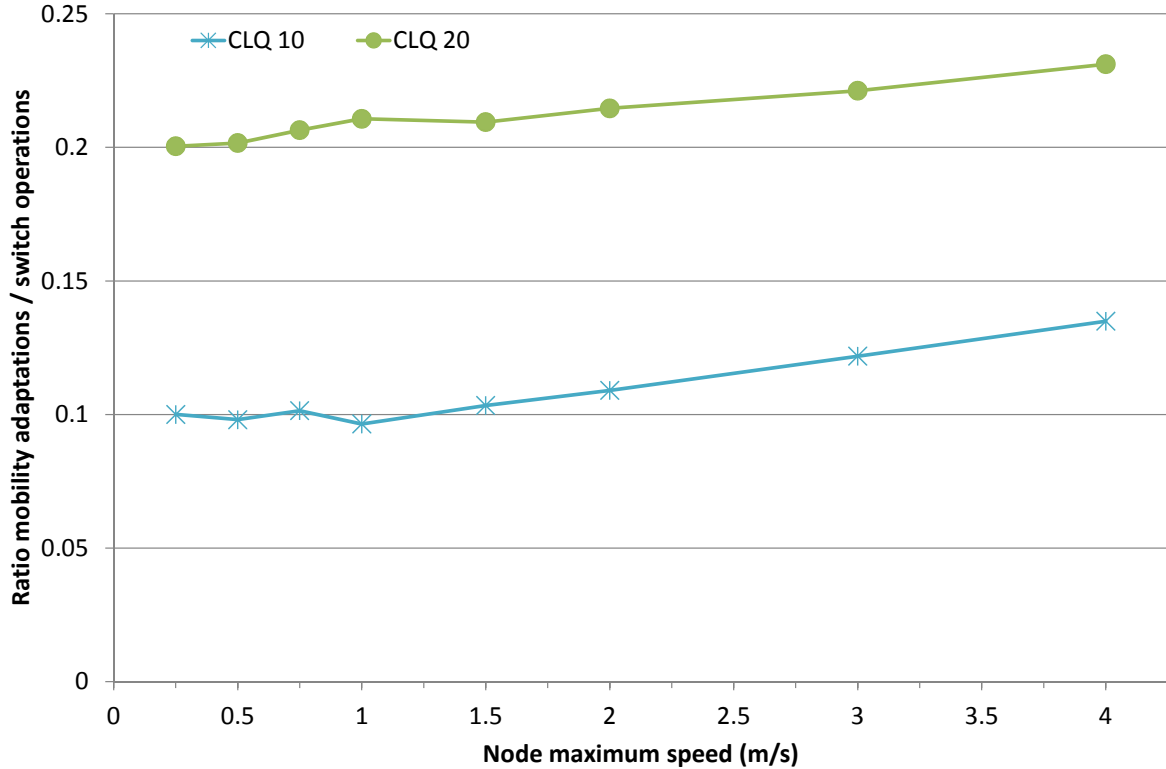


Figure 3.23: Ratio of mobility adaptations to cluster modifications vs. maximum node speed in mobile unstructured networks.

In static configurations our simulations verified that when the topology is dense enough, CLQ builds clusters whose size is at the same time no greater and the closest to n_{max} , thus achieving \mathbf{G}_1 . Another result concerns the number of singleton clusters, which is minimized by CLQ. Our algorithm also maximizes the intra-cluster link capacities, thus attaining \mathbf{G}_3 .

In mobile networks, the good properties of CLQ leads to the most stable clusters. An additional interesting feature of this algorithm lies in its anticipation of the cluster constraints violations, which allows smooth transitions in cluster memberships.

3.6 Conclusions

In this chapter, DCOG was revisited within the coalitional game theory framework, which enabled us to introduce a generic clustering algorithm for mobile ad hoc networks. Using this fully distributed algorithm a node can decide to perform switch operations to leave its current cluster and join a neighboring one, alone or in the company of some other members from its cluster. This decision is made in order to strictly increase the social welfare of the network, thus guaranteeing convergence to a Nash-stable partition.

This algorithm is agnostic of the network structure and is configurable through the choice of a proper utility function and of two heuristics for node selection. We proved its Nash-stability and defined the main condition to be satisfied by any utility function in order to build clusters whose size is maximal.

We then adapted this algorithm to structured networks, suggesting a family of utility func-

tions and several heuristics for node selection suitable for this type of network. In addition to building clusters whose size is maximal, this algorithm, called COG, also aims at gathering all members of the same group in a single cluster. We showed thanks to numerous simulations that COG outperforms a centralized heuristic building one cluster per group. We assessed the effect of various heuristics for node selection, and highlighted the trade off existing between complexity and performance.

We also applied our generic algorithm to unstructured networks, called CLQ, with the aim of building maximum size compact clusters. We verified that CLQ outperforms other clustering solutions from the literature in both static and mobile conditions.

General Conclusions and Perspectives

Conclusions

The work carried out in this thesis took place in the context of ad hoc networks, which are self-organizing networks used in various contexts such as public safety or military networks. The organizational strategy studied in this thesis is called node clustering, and consists in forming sets of nodes, denoted by clusters, in order to introduce hierarchy in the network, and thus improve its scalability. The main objective was to design and analyze clustering algorithms tailored to unstructured networks, for which the state of the art is plentiful, but still presents some defects, and to structured networks, which are very few covered by the literature. Due to the lack of metrics to assess the performance of clustering solutions at the application level, part of the thesis was devoted to the proposal and analysis of global cost functions, that we then used to compare our clustering algorithms with various schemes from the literature.

In Chapter 1, we defined several cost functions which can be used as a benchmark to compare different clustering solutions. These functions measure the quality of a network partition using end-to-end path calculations with additive metrics such as end-to-end delay. They take into account the fact that inter-cluster and intra-cluster communications have different costs, and are flexible enough to cover both cases when the traffic distribution depends on the groups or not. We used the new cost functions to show that the routing should take into account the cluster structure to find good network paths and offer the best QoS to the user traffic. Thanks to a rigorous numerical evaluation, we also verified that in structured networks, the cluster must be formed based on the group membership, and that advanced clustering solutions using group information were required. Finally, we extended our cost functions to the throughput concave metric.

In Chapter 2, we detailed our methodology to design the novel distributed clustering algorithm DCOG suited to structured ad hoc networks, whose key characteristics are its capability to build size limited clusters including full groups, without resorting to CH nodes. We also established the theoretical convergence of this new clustering algorithm. We performed numerous simulations in small and large networks, with low and high densities, and in static and mobile conditions. The comparisons with existing solutions showed that our solution outperforms the ones from the literature, especially with respect to the global cost functions defined in Chapter 1.

In Chapter 3, DCOG was revisited within the coalitional game theory framework, which enabled us to introduce a generic clustering algorithm for mobile ad hoc networks. This algorithm is agnostic of the network structure, and is configurable through the choice of a proper utility function and of two heuristics for node selection. We proved its Nash-stability, and defined the main condition to be satisfied by any utility function in order to build clusters whose size is maximal. We then adapted this algorithm to structured networks, suggesting a family of utility functions and several heuristics for node selection suitable for this type of network. In addition to building clusters whose size is maximal, this algorithm, called COG, aims at gathering all members of the same group in a single cluster. We showed thanks to numerous simulations that COG outperforms a centralized heuristic building one cluster per group, and highlighted the trade off existing between complexity and performance. We also applied our generic algorithm to unstructured networks, called CLQ, with the aim of building maximum size compact clusters. We verified its better performance with respect to other solutions from the literature.

Perspectives

The part of this thesis devoted to the performance measurement of clustering solutions has raised up several issues that would deserve to be addressed in future research.

To begin with, the proposed network cost functions adapted to additive metrics neglect the costs associated to multi-user radio access. This simplification is only valid if the available bandwidth is large with respect to the one required to forward the traffic flows, which is true only for a limited amount of applications. Introducing a MAC resource sharing model in these network cost functions would improve their relevance and applicability.

Concerning the metric based on link capacity, we know that real systems cannot offer a continuous capacity but use a discrete set of modulation and coding schemes. Coping with this limitation of real communication systems could be interesting.

In Chapter 3, we defined separate algorithms suitable for structured and unstructured networks, each one pursuing its own goals. An algorithm suited to hybrid structured/unstructured networks could bring significant added value.

In this thesis, the MAC and physical layers underlying the clustering algorithms are modeled with a simple *Waiting/Busy* automaton. Implementing a more realistic model, or a real protocol on top of an existing radio access scheme, would also be very interesting.

Finally, we tried to cast all the existing CH based clustering algorithms in the framework of our generic clustering algorithm. We did not succeed in this attempt but we gained some insights in the process and are convinced that the framework of coalitional game theory is large enough to achieve this goal.

Appendices

Appendix A

Brute force graph partitioning

In this section we describe an algorithm to find all partitions of a graph \mathcal{G} . First the procedure $\text{RIP}()$ is described which finds all the integer partitions of a number N , then procedure $\text{FPWIP}()$ to find all the partitions of a graph knowing an integer partition of its number of vertices.

To find all the partitions of a graph \mathcal{G} , the procedure $\text{RIP}()$ must be modified in order to invoke the procedure $\text{FPWIP}()$ each time a new integer partition of $|\mathcal{G}|$ is found.

A.1 Integer partitioning

Knowing two integers N and K , the problem is to partition N into the sum of K strictly positive integers $(n_k)_{k=0}^{K-1}$ such that:

1. $\sum_{k=0}^{K-1} n_k = N$, and
2. All permutations of (n_k) are considered equivalent.

Table A.1 describes a recursive algorithm that can solve the integer partition problem.

1	$\text{RIP}(i, (n_k))$
2	$n_{K-1} = n_{K-1} - 1$
3	if $n_{K-2} < n_{K-1}$ then:
4	repeat for $j = i$ and $j = i - 1, j > 0$
5	if $(n_{j-1} < n_j)$ and $(n_{j-1} < n_{K-1} - 1)$ then:
6	$(m_k) = (n_k)$
7	$m_{j-1} = n_{j-1} + 1$
8	$\text{RIP}(j, (m_k))$

Table A.1: Recursive Integer Partitioning (RIP) algorithm.

The input parameters of the recursive procedure RIP are: (1) a possible (n_k) , and (2) the index i of the n_k to be increased.

Let $(n_0) = \{n_k | n_k = 1, \forall k \in \{0, 1, \dots, K - 2\}\}$ and $n_{K-1} = N - K + 1$. Initially the procedure is called with (n_0) as first possible (n_k) and $K - 1$ as initial index. The algorithm builds a tree rooted in (n_0) . Each time it is called, the procedure checks if it is possible to find a new (n_k) solution by reducing by one the value of n_{K-1} and adding 1 either to n_{i-1} or n_{i-2} .

In the former case the same tree branch is continued, and in the latter case a new sub-tree (as well as a new branch, the trunk of the new sub-tree) with current solution as root is started.

Example: looking for the number of ways to partition $N = 8$ into the sum of $K = 4$ positive integer values. (n_0) is defined as $[1\ 1\ 1\ 6]$. As detailed in Table A.2, the solutions are: $[1\ 1\ 1\ 5]$, $[1\ 1\ 2\ 4]$, $[1\ 1\ 3\ 3]$, $[1\ 2\ 2\ 3]$ and $[2\ 2\ 2\ 2]$. Along a branch, the solutions are always updated through the addition of 1 to the corresponding index.

<p>RIP(3, [1 1 1 6])</p> <p>(2) $n_3 = 5 \Rightarrow [1\ 1\ 1\ 5] \Rightarrow$ branch 0</p> <p>(4) $j = 3$</p> <p>(5-8) Possible to +1 to index 2 of [1 1 1 5] \Rightarrow RIP(3, [1 1 2 5])</p> <p>Recursion depth 1 (index = 3):</p> <p>(2) $n_3 = 4 \Rightarrow [1\ 1\ 2\ 4]$</p> <p>(4) $j = 3$</p> <p>(5-8) Possible to +1 to index 2 of [1 1 2 4] \Rightarrow RIP(3, [1 1 3 4])</p> <p>Recursion depth 2 (index = 3):</p> <p>(2) $n_3 = 3 \Rightarrow [1\ 1\ 3\ 3]$</p> <p>(4) $j = 3$</p> <p>(5) Not possible to +1 index 2 of [1 1 3 3]</p> <p>(4) $j = 2$</p> <p>(5) Not possible to +1 index 1 of [1 1 3 3]</p> <p>(4) $j = 2$</p> <p>(5-8) Possible to +1 to index 1 of [1 1 2 4] \Rightarrow RIP(2, [1 2 2 4])</p> <p>Recursion depth 2 (index = 2):</p> <p>(2) $n_3 = 3 \Rightarrow [1\ 2\ 2\ 3] \Rightarrow$ branch 1</p> <p>(4) $j = 2$</p> <p>(5) Not possible to +1 index 1 of [1 2 2 3]</p> <p>(4) $j = 1$</p> <p>(5-8) Possible to +1 to index 0 of [1 2 2 3] Recursion depth 3 (index = 2):</p> <p>(2) $n_3 = 2 \Rightarrow [2\ 2\ 2\ 2] \Rightarrow$ branch 2</p> <p>(4) $j = 1$</p> <p>(5) Not possible to +1 index 0 of [2 2 2 2]</p> <p>(4) $j = 2$</p> <p>(5) Not possible to +1 index 1 of [1 1 1 5]</p>

Table A.2: Using RIP to partition 8 into the sum of 4 positive integer values.

A.2 Partitions of a graph

Table A.3 details the recursive procedure FPWIP (Find Partition With Integer Partition) to find all the partitions of a graph \mathcal{G} knowing one integer partition of the number N of vertices in \mathcal{G} . FPWIP() builds the partitions making sure that all parts satisfy the connectivity, diameter and size constraints. It works on lists of vertices sorted by increasing identifiers order. The parameters of FPWIP() are:

- *partition*: the partition being built, initialized to empty.
- *part*: the partition part being built, initialized to empty.
- *pIndex*: the index of the part being built, initialized to 0.
- *numV*: the number of vertices missing in the current part, initialized to the current part target size.
- *freeVertices*: list of vertices not yet included in the current partition *partialPartition*, initialized to all vertices.
- *vIndex*: among the vertices *freeVertices*, some may have been dismissed by previous calls to FPWIP(), when the constraints were checked. The *vIndex* thus points to the first vertex in this list that has not yet been assessed by the procedure and that is currently the first candidate to be included in the current part being built.
- *IP*: integer partition of the number of vertices (constant).
- *LV*: list of vertices of graph \mathcal{G} to be partitioned (constant).

The `len()` function associates the number of vertices in a list of vertices of \mathcal{G} . The `diam()` function returns the diameter of a graph. The `graph()` function associates the sub-graph of \mathcal{G} induced by a set of vertices of \mathcal{G} . The `distance(\mathcal{G} , \mathcal{L} , V)` function returns the maximum distance calculated within a graph \mathcal{G} between any vertex in a list of vertices \mathcal{L} and a single vertex V . The `updateFreeVertices($L1$, $L2$, min)` function builds a new list with the vertices of list $L2$ that are not in list $L1$ and whose identifier is strictly greater than a minimum.

Line (2) makes sure that the number of vertices missing in the partition part being built is not greater than the remaining candidate vertices. If not the current partition cannot be completed and the last vertex included must be replaced by another one (line (3)).

Line (4) checks if the current part is currently complete. In that case line (5) checks that the current part satisfies the connectivity and diameter constraints. If not the current partition cannot be completed and the last vertex included must be replaced by another one (line (6)). At line (7), the current part is added to the partition. After line (8) a new part is going to be built, thus the index of the part being built is increased by one. Line (9) checks if the partition is completed. If this is not the case then lines (10-13) handle the two cases when the new part to build and the one just built have the same size or not. Line (14) determines the size of the new part to build and line (15) recursively calls FPWIP().

If the test at line (4) determined that the current part is still partial then line (17) iteratively tries to extend it with the next candidate vertex (whose index is incremented at line (18)). If the distance between any vertex in the part being built and the candidate vertex is greater than the distance constraint, then ignore this candidate vertex. Otherwise, line (21) iteratively calls FPWIP() considering as new current part the current part extended with the candidate vertex.

```

1 FPWIP(partition, part, pIndex, numV, freeVertices, vIndex, IP, LV):
2   if [ numV > [ len(freeVertices) - vIndex ] ] then:
3       return
4   else if (numV = 0) then:
5       if [ [ graph(part) is not connected ] or [ diam(graph(part)) > dmax ] ] then:
6           return
7       partition = partition + part
8       pIndex = pIndex + 1
9       if [ pIndex < len(IP) ] then:
10          if (IP[pIndex-1] = IP[pIndex]):
11              freeVertices = updateFreeVertices(part, freeVertices, part[0])
12          else:
13              freeVertices = updateFreeVertices(partition, LV, 0)
14          numV = IP[pIndex]
15          FPWIP(partition, [], pIndex, numV, freeVertices, 0, IP, LV)
16   else:
17       for each vertex in freeVertices list starting at index vIndex:
18           vIndex = vIndex + 1
19           if [ distance(graph(LV), part, vertex) ≤ dmax ]:
20               extendedPart = part + vertex
21               FPWIP(partition, extendedPart, pIndex, numV - 1,
                       freeVertices, vIndex, IP, LV)

```

Table A.3: Algorithm to find partitions of a graph.

A.2.1 Trace of the algorithm

Table A.4 displays a partial trace of FPWIP(), called using the following parameters: FPWIP(partition=[], part=[3], pIndex=0, numV=1, freeVertices=[1,2,3,4,5,6,7,8], vIndex=3), with IP = [2,2,3], LV = [1,2,3,4,5,6,7,8] and the graph the one of Fig. A.1. This trace details how the partial partition [3 4] [5 8] is found by FPWIP(), starting with vertex 3.

Note: in this example, FPWIP() determines that part [1 2 6 7] does not satisfy the maximum diameter constraint and that partial partition [3 4] [5 8] is not valid. The first valid partition found by this call of FPWIP() is [3 4] [6 7] [1 2 5 8].

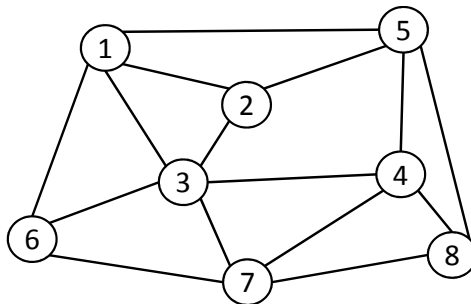


Figure A.1: Example of graph to be partitioned.

```

FPWIP(partition=[], part=[3], pIndex=0, numV=1,
      freeVertices=[1,2,3,4,5,6,7,8], vIndex=3)
(17-18) vertex = 4, vIndex = 4
(19) distance(LV, [3], 4) OK
(21) FPWIP(partition=[], part=[3,4], pIndex=0, numV=0,
      freeVertices=[1,2,3,4,5,6,7,8], vIndex=4)
(5) [3,4] connected and diam([3,4]) OK
(7) partition = [3,4]
(8) pIndex = 1
(10) IP[0] = IP[1] : yes
(11) freeVertices = [5,6,7,8]
(14) numV = 2
(15) FPWIP(partition=[3,4], part=[], pIndex=1, numV=2,
      freeVertices=[5,6,7,8], vIndex=0)
(17-18) vertex = 5, vIndex = 1
(19) distance(LV, [], 5) OK
(21) FPWIP(partition=[3,4], part=[5], pIndex=1,
      numV=1, freeVertices=[5,6,7,8], vIndex=1)
      freeVertices=[5,6,7,8], vIndex=0)
(17-18) vertex = 6, vIndex = 2
(19) distance(LV, [5], 6) OK
(21) FPWIP(partition=[3,4], part=[5,6], pIndex=1,
      numV=0, freeVertices=[5,6,7,8], vIndex=2)
(5) [5,6] disconnected
(17-18) vertex = 7, vIndex = 3
(19) distance(LV, [5], 7) OK
(21) FPWIP(partition=[3,4], part=[5,7], pIndex=1,
      numV=0, freeVertices=[5,6,7,8], vIndex=3)
(5) [5,7] disconnected
(17-18) vertex = 8, vIndex = 4
(19) distance(LV, [5], 8) OK
(21) FPWIP(partition=[3,4], part=[5,8], pIndex=1,
      numV=0, freeVertices=[5,6,7,8], vIndex=4)
(5) [5,8] connected and diam([5,8]) OK
(7) partition = [3,4,5,8]
(8) pIndex = 2
(10) IP[1] = IP[2] : NO
(13) freeVertices = [1,2,6,7]
(14) numV = 4
(15) FPWIP(partition=[3,4,5,8], part=[],
      pIndex=2, numV=4, freeVertices=[1,2,6,7]
      vIndex=0)
...

```

Table A.4: Partial trace of FPWIP() to partition the graph of Fig. A.1.

Appendix B

Proofs of Chapter 3

B.1 Proof of Result 3.1

Firstly, since the clusters of the initial partition fulfill the constraints, \mathbf{P}_1 is operated at the first decision-making time of the algorithm. After \mathbf{P}_1 execution, the resulting clusters satisfy the constraints by construction. Thus, by recurrence, the algorithm will always run \mathbf{P}_1 . Let us now define the social welfare Ψ of a partition $p = \{\mathcal{C}_1, \dots, \mathcal{C}_{N_c}\}$ as $\Psi(p) := \sum_{k=1}^{N_c} v(\mathcal{C}_k)$. Let us consider the n th decision-making occurrence of \mathbf{P}_1 . For the sake of clarity we omit the index n for all functions and variables except for p_n and $\Psi_n := \Psi(p_n)$. Before the n th switch operation the cluster structure of the network is $p_{n-1} = \{\mathcal{C}_1, \dots, \mathcal{C}_{N_c}\}$. Let $\sigma_{k,\ell}(\mathcal{P})$ be the n th selected switch operation. We get $\Psi_n = \Psi_{n-1} + g(\sigma_{k,\ell}(\mathcal{P}))$. Since we consider switch operations with only strictly positive gain, we have $\Psi_n > \Psi_{n-1}$. After any switch operation the social welfare strictly increases, meaning that the same partition can never be visited twice. Furthermore, since there is a finite number of partitions, the algorithm converges to a final partition p_f after a finite number of iterations.

B.2 Proof of Result 3.3

Let us note $x := n_k$, $y := n_\ell$ and $z := |\mathcal{P}|$. Because $n_\ell + |\mathcal{P}| > n_k$, we have $x < y + z$. The function f_1 is a strictly convex function of the cluster size, thus we apply the chordal slope lemma with $x - z < x < y + z$:

$$\frac{f_1(x) - f_1(x - z)}{x - (x - z)} < \frac{f_1(y + z) - f_1(x - z)}{y + z - (x - z)} < \frac{f_1(y + z) - f_1(x)}{y + z - x}.$$

Because $x < y + z$, we have $x - z < y$ and we apply again the chordal slope lemma with $x - z < y < y + z$:

$$\frac{f_1(y) - f_1(x - z)}{y - (x - z)} < \frac{f_1(y + z) - f_1(x - z)}{y + z - (x - z)} < \frac{f_1(y + z) - f_1(y)}{y + z - y}.$$

Consequently, we have:

$$\begin{aligned} \frac{f_1(x) - f_1(x - z)}{z} < \frac{f_1(y + z) - u(y)}{z} &\Rightarrow f_1(x) - f_1(x - z) < f_1(y + z) - f_1(y), \\ &\Rightarrow u_1(\mathcal{C}_k) - u_1(\mathcal{C}_k \setminus \mathcal{P}) < u_1(\mathcal{C}_\ell \cup \mathcal{P}) - u_1(\mathcal{C}_\ell). \end{aligned} \quad (\text{B.1})$$

Because we assume that all the constraints are satisfied $v = u_1$. Then applying (3.2) to (B.1) we get: $r_{\mathcal{P}}(\mathcal{C}_k) < r_{\mathcal{P}}(\mathcal{C}_\ell \cup \mathcal{P})$, which from (3.1) concludes the proof.

B.3 Proof of Result 3.4

If $f_{2,t}$ is strictly convex then noting $x := m_{t,k}$, $y := m_{t,\ell}$ and $z := |\mathcal{P}|$, we can do as in proof of Result 3.3 to get:

$$f_{2,t}(m_{t,k}) - f_{2,t}(m_{t,k} - |\mathcal{P}|) < f_{2,t}(m_{t,\ell} + |\mathcal{P}|) - f_{2,t}(m_{t,\ell}). \quad (\text{B.2})$$

Let us define A and B such that:

$$A := \sum_{\substack{t' \in \mathcal{I}(\mathcal{C}_k) \\ t' \neq t}} f_{2,t'}(m_{t',k}), \text{ and } B := \sum_{\substack{t' \in \mathcal{I}(\mathcal{C}_\ell) \\ t' \neq t}} f_{2,t'}(m_{t',\ell}).$$

Eq. (B.2) can be modified as follows:

$$\begin{aligned} (\text{B.2}) &\Leftrightarrow \left[A + f_{2,t}(m_{t,k}) \right] - \left[A + f_{2,t}(m_{t,k} - |\mathcal{P}|) \right] < \left[B + f_{2,t}(m_{t,\ell} + |\mathcal{P}|) \right] - \left[B + f_{2,t}(m_{t,\ell}) \right] \\ &\Leftrightarrow u_2(\mathcal{C}_k) - u_2(\mathcal{C}_k \setminus \mathcal{P}) < u_2(\mathcal{C}_\ell \cup \mathcal{P}) - u_2(\mathcal{C}_\ell). \end{aligned} \quad (\text{B.3})$$

Because we assume that all the constraints are satisfied, $v = u_2$. Applying (3.2) to (B.3) we get: $r_{\mathcal{P}}(\mathcal{C}_k) < r_{\mathcal{P}}(\mathcal{C}_\ell \cup \mathcal{P})$, which from (3.1) concludes the proof.

B.4 Proof of Result 3.5

Let us consider two clusters \mathcal{C}_k and \mathcal{C}_ℓ such that $\mathcal{C}_k \cup \mathcal{C}_\ell$ satisfies the constraints. Let us calculate the gain $g(\sigma_{k,\ell}(\mathcal{C}_k))$ associated with the merge of \mathcal{C}_k with \mathcal{C}_ℓ . We have:

$$g(\sigma_{k,\ell}(\mathcal{C}_k)) = \frac{2\epsilon \cdot n_\ell \cdot n_k}{n_{max}^2} + (1 - \epsilon) \cdot A(\mathcal{C}_k, \mathcal{C}_\ell),$$

with

$$A(\mathcal{C}_k, \mathcal{C}_\ell) := \sum_{t \in \mathcal{I}(\mathcal{C}_k \cup \mathcal{C}_\ell)} f_{2,t}(m_{t,k} + m_{t,\ell}) - \sum_{t \in \mathcal{I}(\mathcal{C}_\ell)} f_{2,t}(m_{t,\ell}) - \sum_{t \in \mathcal{I}(\mathcal{C}_k)} f_{2,t}(m_{t,k}).$$

Let us define $\mathcal{J} := \mathcal{I}(\mathcal{C}_k) \cap \mathcal{I}(\mathcal{C}_\ell)$. When $\mathcal{J} = \emptyset$ then $A(\mathcal{C}_k, \mathcal{C}_\ell) = 0$ and $g(\sigma_{k,\ell}(\mathcal{C}_k)) > 0$. When $\mathcal{J} \neq \emptyset$ then we have:

$$\begin{aligned} A(\mathcal{C}_k, \mathcal{C}_\ell) &= \sum_{t \in \mathcal{J}} f_{2,t}(m_{t,k} + m_{t,\ell}) + \sum_{t \in \mathcal{I}(\mathcal{C}_k \setminus \mathcal{J})} f_{2,t}(m_{t,k}) + \sum_{t \in \mathcal{I}(\mathcal{C}_\ell \setminus \mathcal{J})} f_{2,t}(m_{t,\ell}), \\ &\quad - \left[\sum_{t \in \mathcal{J}} f_{2,t}(m_{t,\ell}) + \sum_{t \in \mathcal{I}(\mathcal{C}_\ell \setminus \mathcal{J})} f_{2,t}(m_{t,\ell}) \right] - \left[\sum_{t \in \mathcal{J}} f_{2,t}(m_{t,k}) + \sum_{t \in \mathcal{I}(\mathcal{C}_k \setminus \mathcal{J})} f_{2,t}(m_{t,k}) \right]. \end{aligned}$$

Thus $A(\mathcal{C}_k, \mathcal{C}_\ell)$ can be written as a sum: $A(\mathcal{C}_k, \mathcal{C}_\ell) = \sum_{t \in \mathcal{J}} F(t)$, with $F(t) := f_{2,t}(m_{t,k} + m_{t,\ell}) - f_{2,t}(m_{t,k}) - f_{2,t}(m_{t,\ell})$. For all $t \in \mathcal{J}$:

$$F(t) = \frac{2 \cdot m_{t,\ell} \cdot m_{t,k}}{T \cdot m_t^2} > 0.$$

Since $A(\mathcal{C}_k, \mathcal{C}_\ell)$ is a sum of terms that are strictly greater than zero, then $A(\mathcal{C}_k, \mathcal{C}_\ell) > 0$, and $g(\sigma_{k,\ell}(\mathcal{C}_k)) > 0$.

B.5 Proof of Result 3.6

We study the sign of $\Delta := g(\sigma_{q,k}(\mathcal{U}_q)) - g(\sigma_{q,\ell}(\mathcal{U}_q))$ as a function of ϵ . From (3.1), we have

$$\Delta = r_{\mathcal{U}_q}(\mathcal{C}_k \cup \mathcal{U}_q) - r_{\mathcal{U}_q}(\mathcal{C}_\ell \cup \mathcal{U}_q).$$

Since the constraints are fulfilled, $v = u$, then using (3.2) we get:

$$r_{\mathcal{U}_q}(\mathcal{C}_x \cup \mathcal{U}_q) = u(\mathcal{C}_x \cup \mathcal{U}_q) - u(\mathcal{U}_q),$$

where x stands for k or ℓ .

From utility (3.5), and (3.6)-(3.9), we obtain:

$$u(\mathcal{C}_x) = \frac{n_x^2}{n_{max}^2} \epsilon + \frac{(1-\epsilon)}{T \cdot m_t^2} (K + m_{t,x}^2),$$

where K is a constant that depends on the groups different from \mathcal{O}_t . Likewise, we get:

$$u(\mathcal{C}_x \cup \mathcal{U}_q) = \frac{(n_x + n_{\mathcal{U}})^2}{n_{max}^2} \epsilon + \frac{(1-\epsilon)}{T \cdot m_t^2} [K + (m_{t,x} + n_{\mathcal{U}})^2].$$

After simplification, we obtain:

$$r_{\mathcal{U}_q}(\mathcal{C}_x \cup \mathcal{U}_q) = \frac{n_{\mathcal{U}}(2n_x + n_{\mathcal{U}})}{n_{max}^2} \epsilon + \frac{(1-\epsilon)n_{\mathcal{U}}}{T \cdot m_t^2} (2m_{t,x} + n_{\mathcal{U}}),$$

which leads to:

$$\Delta = D_1 \cdot \epsilon + D_2(1 - \epsilon),$$

with

$$D_1 := \frac{2n_{\mathcal{U}}(n_k - n_\ell)}{n_{max}^2},$$

$$D_2 := \frac{2n_{\mathcal{U}}}{T \cdot m_t^2} (m_{t,k} - m_{t,\ell}).$$

The study of the sign of Δ leads to the following three cases:

Case 1: $m_{t,k} = m_{t,\ell}$ and $n_k > n_\ell$. Then $D_2 = 0$ and $D_1 > 0$. Consequently $\Delta > 0$ whatever the value of ϵ .

Case 2: $m_{t,k} = m_{t,\ell}$ and $n_k = n_\ell$. Then $\Delta = 0$ whatever the value of ϵ .

Case 3: $m_{t,k} \neq m_{t,\ell}$. We assume without loss of generality that $m_{t,k} > m_{t,\ell}$ and thus $D_2 > 0$. If $n_k \geq n_\ell$, $D_1 > 0$ and $\Delta > 0$ regardless of the value of ϵ . If $n_k < n_\ell$, then $D_1 < 0$. We can write $\Delta = (D_1 - D_2)\epsilon + D_2$ which is a linear function of ϵ with negative slope. Thus $\Delta > 0 \Leftrightarrow \epsilon < \epsilon_0 := \frac{D_2}{D_2 - D_1}$. We now search the parameters $n_k, n_\ell, m_t, m_{t,k}, m_{t,\ell}, n_{\mathcal{U}}$, leading to the smallest value of ϵ_0 denoted by ϵ^* . When D_2 is fixed, we have to minimize D_1 , which is obtained by setting $n_k = 0$, $n_\ell = n_{max}$, leading to $\epsilon_0 = \frac{D_2}{D_2 + 2n_{\mathcal{U}}/n_{max}}$. Minimizing ϵ_0 is then equivalent to minimizing D_2 , which is obtained by setting $m_{t,k} = 1$, $m_{t,\ell} = 0$ and $m_t = n_{max}$, leading to $\epsilon^* = \frac{1}{1+T}$.

B.6 Proof of Result 3.7

Let clusters \mathcal{C}_k and \mathcal{C}_ℓ , such that $\mathcal{C}_k \cup \mathcal{C}_\ell$ satisfies the constraints. Let us first calculate the utility of $\mathcal{C}_k \cup \mathcal{C}_\ell$. From (3.10), we have:

$$\begin{aligned}
 u_{\text{un}}(\mathcal{C}_k \cup \mathcal{C}_\ell) &= \sum_{i \in \mathcal{C}_k} \sum_{j \in \mathcal{C}_k \cup \mathcal{C}_\ell | (i,j) \in \mathcal{E}} \kappa(i,j) + \sum_{i \in \mathcal{C}_\ell} \sum_{j \in \mathcal{C}_k \cup \mathcal{C}_\ell | (i,j) \in \mathcal{E}} \kappa(i,j), \\
 &= \sum_{i \in \mathcal{C}_k} \left[\sum_{j \in \mathcal{C}_k | (i,j) \in \mathcal{E}} \kappa(i,j) + \sum_{j \in \mathcal{C}_\ell | (i,j) \in \mathcal{E}} \kappa(i,j) \right] + \sum_{i \in \mathcal{C}_\ell} \left[\sum_{j \in \mathcal{C}_k | (i,j) \in \mathcal{E}} \kappa(i,j) + \sum_{j \in \mathcal{C}_\ell | (i,j) \in \mathcal{E}} \kappa(i,j) \right], \\
 &= u_{\text{un}}(\mathcal{C}_k) + u_{\text{un}}(\mathcal{C}_\ell) + 2 \cdot \sum_{i \in \mathcal{C}_k} \sum_{j \in \mathcal{C}_\ell | (i,j) \in \mathcal{E}} \kappa(i,j).
 \end{aligned}$$

Using (3.1) and (3.2) the gain $g(\sigma_{k,\ell}(\mathcal{C}_k))$ associated with the merge of \mathcal{C}_k with \mathcal{C}_ℓ is thus written:

$$g(\sigma_{k,\ell}(\mathcal{C}_k)) = 2 \cdot \sum_{i \in \mathcal{C}_k} \sum_{j \in \mathcal{C}_\ell | (i,j) \in \mathcal{E}} \kappa(i,j).$$

Since $\kappa(i,j) > 0$, we deduce that $g(\sigma_{k,\ell}(\mathcal{C}_k)) > 0$ which concludes the proof.

Bibliography

- [1] M. Gerla and J. Tsai, “Multicluster, Mobile, Multimedia Radio Network,” *Journal of Wireless Networks*, vol. 1, no. 3, pp. 255–265, July 1995.
- [2] X. Zhang, H. Su, and H. h. Chen, “Cluster-based Multi-channel Communications Protocols in Vehicle Ad Hoc Networks,” *IEEE Wireless Communications*, vol. 13, no. 5, pp. 44–51, October 2006.
- [3] T. Chen, H. Zhang, G. M. Maggio and I. Chlamtac, “CogMesh: A Cluster-Based Cognitive Radio Network,” in *2nd IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks*, April 2007, pp. 168–178.
- [4] A. Asterjadhi, N. Baldo, and M. Zorzi, “A cluster formation protocol for cognitive radio ad hoc networks,” in *IEEE European Wireless Conference*, April 2010, pp. 955–961.
- [5] J. Sucec and I. Marsic, “Hierarchical Routing Overhead in Mobile Ad Hoc Networks,” *IEEE Transactions on Mobile Computing*, vol. 3, no. 1, pp. 46–56, March 2004.
- [6] H. Claussen, L. T. W. Ho and L. G. Samuel, “An Overview of the Femtocell Concept,” *Bell Labs Technical Journal*, vol. 13, no. 1, pp. 221–245, 2008.
- [7] C. C. Chiang, H. K. Wu, W. Liu, and M. Gerla, “Routing In Clustered Multihop, Mobile Wireless Networks With Fading Channel,” in *IEEE Singapore International Conference on Networks*, 1997.
- [8] R. Ghosh and S. Basagni, “Mitigating the Impact of Node Mobility on Ad Hoc Clustering,” in *Wireless Communications and Mobile Computing*, vol. 8, no. 3, March 2008, pp. 295–308.
- [9] F. Li, S. Zhang, W. Wang, X. Xue and H. Shen, “Vote-based Clustering Algorithm in Mobile Ad Hoc Networks,” in *International Conference on Information Networking (ICON)*, Busan, South Korea, February 2004.
- [10] Y. Zhang and J. Ng, “A Distributed Group Mobility Adaptive Clustering Algorithm for Mobile Ad Hoc Networks,” in *IEEE International Conference on Communications (ICC)*, Beijing, China, May 2008, pp. 3161–3165.
- [11] X. Gao, “A mobility based clustering algorithm in MANETs utilizing Learning Automata,” in *32nd IEEE Chinese Control Conference (CCC)*, July 2013, pp. 6398–6402.
- [12] N. Keerthipriya and R. S. Latha, “Adaptive cluster formation in MANET using particle swarm optimization,” in *3rd IEEE International Conference on Signal Processing, Communication and Networking (ICSCN)*, March 2015, pp. 1–7.

- [13] X. Tan, Z. Xiong and Y. He, “Signal Attenuation-aware Clustering in Wireless Mobile Ad Hoc Networks,” in *Journal of Networks*, vol. 8, no. 4, April 2013, pp. 796–803.
- [14] M. Cai, L. Rui, D. Liu, H. Huang and X. Qiu, “Group Mobility Based Clustering Algorithm for Mobile Ad Hoc Networks,” in *17th IEEE Asia-Pacific Network Operations and Management Symposium (APNOMS)*, August 2015, pp. 340–343.
- [15] H. Wu, Z. Zhong, and L. Hanzo, “A Cluster-Head Selection and Update Algorithm for Ad Hoc Networks,” in *IEEE International Conference GLOBECOM*, December 2010, pp. 1–5.
- [16] D. Camara, C. Bonnet and N. Nikaein, “Topology Management for Group Oriented Networks,” in *IEEE International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC)*, Honolulu, Hawaii, September 2010, pp. 2739–2744.
- [17] W. Saad, Z. Han, M. Debbah, A. Hjørungnes and T. Basar, “Coalitional Game Theory for Communication Networks,” *IEEE Signal Processing Magazine*, vol. 26, no. 5, pp. 77–97, September 2009.
- [18] —, “Coalitional Games for Distributed Collaborative Spectrum Sensing in Cognitive Radio Networks,” in *IEEE INFOCOM*, April 2009, pp. 2114–2122.
- [19] Z. Zhang, L. Song, Z. Han and W. Saad, “Coalitional Games with Overlapping Coalitions for Interference Management in Small Cell Networks,” *IEEE Transactions on Wireless Communications*, vol. 13, no. 5, pp. 2659–2669, May 2014.
- [20] W. Saad, Z. Han, A. Hjørungnes, D. Niyato and E. Hossain, “Coalition Formation Games for Distributed Cooperation Among Roadside Units in Vehicular Networks,” *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 1, pp. 48–60, January 2011.
- [21] F. Chiti, R. Fantacci, E. Dei and Z. Han, “Context Aware Clustering in VANETs: A game Theoretic Perspective,” in *IEEE International Conference on Communications (ICC)*, June 2015, pp. 6584–6588.
- [22] M. W. Baidas and A. B. MacKenzie, “Altruistic Coalition Formation in Cooperative Wireless Networks,” *IEEE Transactions on Communications*, vol. 61, no. 11, pp. 4678–4689, November 2013.
- [23] Y. Cai, H. Chen, D. Wu, W. Yang and L. Zhou, “A Distributed Resource Management Scheme for D2D Communications Based on Coalition Formation Game,” in *IEEE International Conference on Communications Workshops (ICC)*, June 2014, pp. 355–359.
- [24] A. K. Jain, M.N. Murty, and P.J. Flynn, “Data Clustering: A Review,” *ACM Computing Surveys*, vol. 31, no. 3, pp. 264–323, September 1999.
- [25] S. Fortunato, “Community Detection in graphs,” *Physics Reports*, vol. 486, pp. 75–174, 2010.
- [26] J. Chen, C. Richard and A. H. Sayed, in *23rd European Conference on Signal Processing (EUSIPCO)*.

- [27] M. E. J. Newman, and M. Girvan, “Finding and Evaluating Community Structure in Networks,” *Physical Review E*, vol. 69, February 2004.
- [28] G. Herbiet and P. Bouvry, “SHARC: Community-based Partitioning for Mobile Ad Hoc Networks Using Neighborhood Similarity,” in *IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks (WoWMoM)*, June 2010, pp. 1–9.
- [29] —, “On the Generation of Stable Communities of Users for Dynamic Mobile Ad Hoc Social Networks,” in *International Conference on Information Networking (ICOIN)*, January 2011, pp. 262–267.
- [30] S. Dabideen, V. Kawadia and S.C. Nelson, “CLAN: An Efficient Distributed Temporal Community Detection Protocol for MANETs,” in *IEEE International Conference on Mobile Ad Hoc and Sensor Systems*, October 2014, pp. 91–99.
- [31] U.N. Raghavan, R. Albert, and S. Kumara, “Near Linear Time Algorithm to Detect Community Structures in Large-scale Networks,” *Physical Review E*, vol. 76, p. 036106, September 2007.
- [32] R. Massin, C. J. Le Martret and P. Ciblat, “A Network Cost Function for Clustered Ad Hoc Networks: Application to Group Based Systems,” in *25th IEEE International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, Washington, D.C., September 2014.
- [33] E.W. Dijkstra, “A Short Introduction To The Art Of Programming,” *EWD316*, pp. 67–73, 1971.
- [34] T. Clausen and P. Jacquet, “Optimized Link State Routing Protocol (OLSR),” *RFC 3626*, October 2003.
- [35] A. Munaretto, H. Badis, K. Al Agha and G. Pujolle, “A Link-state QoS Routing Protocol for Ad Hoc Networks,” in *4th International Workshop on Mobile and Wireless Communications Network*, 2002, pp. 222–226.
- [36] R. Baumann, S. Heimlicher, M. Strasser and A. Weibel, “A Survey on Routing Metrics TIK Report 262,” 2007.
- [37] D. Eppstein, “Finding the k Shortest Paths,” *SIAM J. Computing*, vol. 28, no. 2, pp. 652–673, 1998.
- [38] Z. Wang and J. Crowcroft, “Quality of Service Routing for Supporting Multimedia Applications,” *Selected Areas In Communications*, September 1996.
- [39] R. Massin, C. J. Le Martret and P. Ciblat, “Distributed Clustering Algorithms in Group-Based Ad Hoc Networks,” in *23rd European Signal Processing Conference (EUSIPCO)*, September 2015.
- [40] —, “Un Algorithme de Clusterisation Distribué pour les Réseaux Ad Hoc Structurés,” in *25th French GRETSI*, September 2015.

- [41] —, “Distributed Clustering Algorithm in Dense Group-Based Ad Hoc Networks,” in *16th IEEE Mediterranean Ad Hoc Networking Workshop (MedHocNet)*, Vilanova i la Geltru, Spain, June 2016.
- [42] X. Hong, M. Gerla, G. Pei, and C. Chiang, “A Group Mobility Model for Ad hoc Wireless Network,” in *ACM International Workshop on Modeling and Simulation of Wireless and Mobile Systems (MSWiM)*, August 1999, pp. 53–60.
- [43] A. Le Duc, C.J. Le Martret and P. Ciblat, *PhD Dissertation: Performance Closed-form Derivations and Analysis of Hybrid ARQ Retransmission Schemes in a Cross-layer Context*, February 2010.
- [44] R. Massin, C. J. Le Martret and P. Ciblat, “A Coalition Formation Game for Distributed Node Clustering in Mobile Ad Hoc Networks,” *IEEE Transactions on Wireless Communications, under preparation for*, 2016.
- [45] A. Bogomolnaia, and M. O. Jackson, “The Stability of Hedonic Coalition Structures,” *Games and Economic Behavior*, vol. 38, p. 2002, 1998.