

Sublinear Fully Distributed Partition with Applications

Bilel Derbel · Mohamed Mosbah · Akka Zemmari

© Springer Science+Business Media, LLC 2009

Abstract We present new efficient deterministic and randomized distributed algorithms for decomposing a graph with n nodes into a disjoint set of connected clusters with radius at most $k - 1$ and having $O(n^{1+1/k})$ intercluster edges. We show how to implement our algorithms in the distributed *CONGEST* model of computation, i.e., limited message size, which improves the time complexity of previous algorithms (Moran and Snir in Theor. Comput. Sci. 243(1–2):217–241, 2000; Awerbuch in J. ACM 32:804–823, 1985; Peleg in Distributed Computing: A Locality-Sensitive Approach, 2000) from $O(n)$ to $O(n^{1-1/k})$. We apply our algorithms for constructing low stretch graph spanners and network synchronizers in sublinear deterministic time in the *CONGEST* model.

Keywords Distributed algorithm · Distributed model · Time complexity · Sparse partition · Graph spanner

Some materials presented in this paper appeared in two extended abstracts published in the proceedings of IPDPS06 (20th IEEE International Parallel & Distributed Processing Symposium) [19] and PDCS04 (16th IASTED International Conference on Parallel and Distributed Computing and Systems) [18].

B. Derbel

LIFL, INRIA, Université des Sciences et Technologies de Lille Bâtiment M3, 59655 Villeneuve d'Ascq Cedex, France
e-mail: bilel.derbel@lifl.fr

M. Mosbah (✉) · A. Zemmari

LaBRI, Université Bordeaux I, ENSEIRB, 351 Cours de la Libération, 33405 Talence, France
e-mail: mosbah@labri.fr

A. Zemmari

e-mail: zemmari@labri.fr

1 Introduction

Due to the constant growth of networks, it becomes necessary to find new techniques to handle related global information, to maintain and to update this information in an efficient way. A *Locality-Preserving (LP) network representation* [36] can be considered as an efficient data structure that captures topological properties of the underlying network and helps to design distributed algorithms for many fundamental problems: synchronization [6, 34, 41], Maximal Independent Set (MIS) [4], routing [7], mobile users [8], coloring [35] and other related applications [1, 13, 24, 25, 28]. In order to provide *efficient* solutions for these problems, it is important to construct LP-representations in a distributed way while maintaining good complexity measures.

The main purpose of this paper is to give an overview of some LP-representations of special interest and to show how to construct them efficiently in the distributed setting. More precisely, we focus on one important type of LP-representations called *clustered representations*. The main idea of a clustered representation is to decompose the nodes of a graph into many possibly overlapping regions called clusters. This decomposition allows us to organize the graph in a particular way that satisfies some desired properties. In general, the clusters satisfy two types of qualitative criteria. The first criterion attempts to measure the *locality level* of the clusters. Some parameters like the *radius* or the *size* of a cluster are usually used to measure the locality level of a clustered representation. The second criterion attempts to measure the *sparsity level*. This criterion gives an idea about how the clusters are connected to each others. For instance, in the case of disjoint clusters, the number of intercluster edges is usually used to express the sparsity level. In the case of overlapping clusters, the average/maximum number of occurrences of a node in the clusters is usually used to express the sparsity (or the overlap) of the clustered representation.

In general, the locality and the sparsity levels of a clustered representation are tightly related and often go in an opposite way. For instance, one can take the whole graph to be one cluster C . In this case, the sparsity level is good (the degree of C is 0), but the locality level is bad (the radius of C is the radius of the whole graph). In opposite, one can take a representation in which each node forms a cluster. In this case, the locality level is good (the radius of each cluster is 0), but the sparsity level is bad (the degree of a cluster may be Δ where Δ is the maximum degree of the graph).

The complexity of many applications (using clustered representations as a communication structure) is also tightly related to the sparsity and locality levels. In fact, a good locality level implies in general a low time complexity, and a low sparsity level implies low message/memory complexity. All the clustered representations one can find always attempt to find a good compromise between the sparsity and the locality levels.

1.1 Goals and Related Works

In this paper, we focus on an important clustered representation called *Basic Partition* [36, Chap. 11]. Our interest in this *Basic Partition* comes from its good sparsity-locality compromise. In fact, given an n -node graph, the *Basic Partition* provides a

set of *disjoint connected* clusters such that the radius of a cluster is at most $k - 1$ and the number of intercluster edges is $O(n^{1+\frac{1}{k}})$ where k is a given integer parameter. Our goal is to design time efficient algorithms for constructing a *Basic Partition* of a graph in a distributed model of computation where *nodes can only communicate with their neighbors by exchanging messages of limited size*.

The *Basic Partition* was first used in [3] in order to design efficient network synchronizers. The idea of producing a clustered representation satisfying a good compromise between the locality level and the sparsity level was then studied in [5]. The results of [5] inspired many other applications and generalizations [9, 10, 17]. In particular, Awerbuch *et al.* [9] studied two important types of clustered representations:

1. The first one called *network decomposition* aims at partitioning the network into *disjoint* colored clusters with either *weak* or *strong* small radius and using a small number of colors. For *weak*-network decompositions, a cluster does not necessarily need to be connected and its radius is computed using paths which may shortcut through neighboring clusters. For *strong*-network decompositions, a cluster must be connected and its radius is computed in the network induced by this cluster.
2. The second one called *network covers* constructs a set of *possibly overlapping* clusters with the property that for any node v , there exists a cluster which contains the t -neighborhood of v , i.e., the neighbors at distance at most t from v where t is an integer parameter. The quality of such covers is measured using the strong radius of clusters and the cluster overlap, i.e., the maximum number of clusters a node belongs to.

In addition to design new network decompositions satisfying some desirable properties, many works studied the problem of distributively constructing these representations in an efficient way. For instance, Awerbuch *et al.* [9] gave a deterministic (resp. randomized) distributed algorithm to construct a $(k, t, O(kn^{1/k}))$ -neighborhood cover in $O(tk \cdot 2^{c\sqrt{\log n}} + tk^2 \cdot 2^{4\sqrt{\log n}} \cdot n^{1/k})$ (resp. $O(tk^2 \cdot \log^2 n \cdot n^{1/k})$) time for some constant $c > 0$. A (k, t, d) -neighborhood is a set of possibly overlapping clusters such that (i) the strong radius of a cluster is $O(kt)$, (ii) each node belongs to at most d clusters, and (iii) the t -neighborhood of each node is covered by at least one cluster. Moreover, a remark in [9] claims that it is possible to translate this neighborhood cover into a strong-network decomposition of comparable parameters by using some techniques from [5, 17].

On one hand, the strong radius of the cover constructed in [9] is $2k - 1$ which is worse (by a factor 2) than the one of the *Basic Partition*. On the other hand, the distributed model considered there does not take into account the congestion created at various bottlenecks in the network (see Sect. 3.4 of [9]). In fact, the network model used in [9] is the Linial's *free* model [29, 30] also known as the *LOCAL* model (see [36, Chap. 2]). The *LOCAL* model assumes that nodes can communicate by exchanging messages of *unlimited size*. This assumption focuses on the locality nature of distributed problems, i.e., what can be computed distributively provided that every node knows its whole neighborhood at some distance?

From a practical point of view, since clustered representations are in the basis of many applications, it is crucial to design fast algorithms to construct such representations in practical distributed models. From a more theoretical point of view, it is

also interesting and challenging to design fast algorithms assuming only some weak distributed assumptions, e.g., see [37].

In [34], Moran and Snir gave a distributed algorithm that computes a *Basic Partition* in $O(n)$ time in a distributed model where the size of a message is at most $O(\log n)$ bits, i.e., *CONGEST* model (see [36, Chap. 2]). The algorithm of [34] improves the previous constructions of [3, 41], and allows us to obtain more efficient algorithms for designing network synchronizers γ , γ_1 and γ_2 . The algorithm of [34] is semi-sequential: Each cluster is constructed around some node in a distributed and layered fashion. Nevertheless, the clusters are constructed sequentially. In other words, the clusters are constructed one after the other: at each iteration, a new node is selected and the next cluster is constructed.

Moran and Snir end their paper [34] saying:

[34] *Question:* Are there *truly parallel algorithms* which construct a Basic Partition in *polylogarithmic or sublinear time complexity* in the *CONGEST* model?

1.2 Contribution

In the following, we answer the [34] question. In fact, we give new sparse partition algorithms with $O(n^{1-1/k})$ time complexity, using messages of size at most $O(\log n)$.

More precisely, we give a fully distributed deterministic algorithm `DIST_PART` with no precomputation step. The idea is to let the clusters grow spontaneously in parallel in different regions of the graph, breaking ties using node identities. We give a detailed implementation of algorithm `DIST_PART` using small messages and we analyze its efficiency. The time complexity of algorithm `DIST_PART` is only linear. However, the technique of algorithm `DIST_PART` is used as a *black box* in order to design a new synchronous deterministic algorithm (`SYNC_PART`) with sublinear time complexity. The main idea to break the linear time barrier is to privilege the construction of clusters in the dense region of the graph which allows us to finish the distributed construction in constant time once the graph becomes sparse. This idea is then adapted in order to run in an asynchronous setting and we obtain algorithm `FAST_PART`. Our new asynchronous algorithm is even faster than the synchronous one for many particular graphs.

We also give a randomized distributed algorithm (`ELECT_PART`) which is based on a local election technique (LE_k) in balls of radius k . This k -local election technique is a generalization of the algorithms given in [33] and can be of an independent interest. For general graphs, our randomized construction has the same sublinear time complexity as the deterministic one, but it provides improved bounds for many particular graphs. In fact, the analysis of algorithm `ELECT_PART` enables us to express analytically the degree of parallelism of our construction and to compute the expected number of clusters constructed in parallel.

The basic partition can be applied for designing network covers, network synchronizers and also graph spanners. Hence, we obtain new fast algorithms for all of these applications. For instance, we obtain new $O(n^{1-1/k})$ time deterministic algorithm for constructing optimal spanners with size $O(n^{1+1/k})$ and stretch $2k - 1$

which improves on previous constructions. Fast construction of sparse spanners is a real challenge and has been intensively studied over many years and are of special interest for many useful graph structures such as shortest paths, distance oracles and routing [2, 11, 12, 15, 21–23, 40, 42]. We should note that the best known distributed algorithms for constructing graph spanners with optimal stretch-size tradeoffs are either randomized [12] or use unbounded size messages [20].

One should finally note that at each round of our distributed constructions the number of messages exchanged by nodes can be order of the number of edges which is rather large. In this paper, we focus only on providing time efficient constructions. Improving the message complexity of our algorithms remains an open research field as it will be pointed later.

Outline

In Sects. 2 and 3, we give some definitions and we review the BASIC_PART algorithm for constructing the *Basic Partition* in a semi-sequential manner. In Sect. 4, we give a detailed implementation and analysis of the fully distributed algorithm DIST_PART. In Sects. 5 and 6, we describe algorithms SYNC_PART, FAST_PART and ELECT_PART, and we analyze their time complexity. In Sect. 7, we apply our algorithms to construct sparse graph spanners.

The application of the basic partition to network covers and network synchronizers γ , γ_1 and γ_2 is given in Appendix A. In Appendix B, we also give a constructive analysis of our algorithms in the case of *Circulant graphs* and we obtain logarithmic time complexity.

2 Model and Definitions

We represent a network of n processes by an unweighted undirected connected graph $G = (V, E)$ where V represents the set of processes ($|V| = n$) and E the set of links between them. We consider the distributed model of computation used in [3, 34] and known as the *CONGEST* model. More precisely, we assume that a node can only communicate with its neighbors by sending and receiving messages of size $O(\log(n))$ bits. Each node processes messages received from its neighbors, performs local computations, and sends messages to its neighbors in negligible time. In a synchronous network, all nodes have access to a global clock which generates pulses. A message which has been sent in a given pulse arrives before the next pulse. In a synchronous network, the time complexity of an algorithm is defined as the worst-case number of pulses from the start of the algorithm to its termination. In an asynchronous network, there is no global clock and a message delay is arbitrary but finite. In the latter case, the time complexity is defined as the worst-case number of time units from the start of the algorithm to its termination, assuming that a message delay is at most one time unit (this assumption is introduced only for the purpose of performance evaluation).

A cluster C is a subset of V such that the subgraph induced by C is connected. A cluster is always considered with a *leader* node and a BFS spanning tree rooted at the leader. We also assume that each node v of a graph G has a unique identity Id_v

Fig. 1 Algorithm BASIC_PART [36]

```

1: Set  $\mathcal{C} := \emptyset$ 
2: while  $V \neq \emptyset$  do
3:   Select an arbitrary vertex  $v \in V$ 
4:   Set  $C := \{v\}$ 
5:   while  $|\Gamma(C)| > n^{1/k}|C|$  do
6:      $C := \Gamma(C)$ 
7:   end while
8:   Set  $\mathcal{C} := \mathcal{C} \cup C$  and  $V := V - C$ 
9: end while
10: return  $\mathcal{C}$ 

```

(of $O(\log(n))$ bits). The identity Id_C of a cluster C is defined as the identity of its leader.

For every pair of nodes u and v of a graph G , $d_G(u, v)$ denotes the distance between u and v in G (we also write $d(u, v)$ when G is clear from the context). For any node v of a graph G , $\mathcal{N}(v) = \{u \in V \mid d_G(u, v) \leq 1\}$ denotes the neighborhood of v . For any cluster C of a graph G , $\Gamma(C) = \bigcup_{v \in C} \mathcal{N}(v)$ denotes the neighborhood of C . For any cluster C of a graph G , $Rad(C)$ denotes the radius of the cluster C , i.e., the radius of the subgraph induced by C in G . Similarly, for any set \mathcal{C} of clusters, $Rad(\mathcal{C}) = \max_{C \in \mathcal{C}} Rad(C)$ denotes the radius of \mathcal{C} .

In all our algorithms, clusters are constructed in a layered and concurrent fashion. In other words, a cluster may grow and explore a new layer but it may also lose its last layer. Some clusters may disappear because they lost all their layers and some others may be newly formed. A cluster is called *finished* if it belongs to the final decomposition that we are constructing. A node belonging to a finished cluster is also called finished. A node is called *active* if it does not belong to a finished cluster.

3 A Basic Algorithm for Constructing a Sparse Partition

Let $k \geq 1$ be an integer parameter. Typically, k is taken to be small compared with n ($k \leq \log n$). Let us consider algorithm BASIC_PART (Fig. 1) as given in Peleg's book [36, Chap. 11, p. 130]. Algorithm BASIC_PART was first used in [3] as a data structure for synchronizer γ , then some improvements were given in [34, 41]. The algorithm operates in many phases. At each phase, a node is selected from the set of nodes which are not yet covered by a cluster. Then a new cluster is constructed in many iterations according to the sparsity condition of line 5, i.e., $|\Gamma(C)| > n^{1/k}|C|$. It is important to note that the graph G changes in line 8 of the algorithm and the notations in the while loop correspond to the new graph G obtained after the deletion of the corresponding nodes.

Algorithm BASIC_PART constructs a *Basic Partition*. In fact, we have the following:

Theorem 1 [36] *The output \mathcal{C} of algorithm BASIC_PART is a partition of G which satisfies the following properties:*

1. $Rad(\mathcal{C}) \leq k - 1$ (locality level)
2. There are at most $n^{1+1/k}$ intercluster edges (sparsity level)

Proof On one hand, once the construction of a cluster C is finished, the nodes of C are definitely removed from the graph G . Thus, the clusters constructed by the algorithm are disjoint. On the other hand, the algorithm terminates once no node remains uncovered. Thus, the final output \mathcal{C} is a partition of G .

Using the sparsity condition, if a cluster C adds i layers, then the size of C satisfies $|C| > n^{i/k}$. Hence, a cluster cannot add more than $k - 1$ layers and the first property of the partition holds.

Let $G_{\mathcal{C}}$ be the graph induced by the clusters of the partition \mathcal{C} : the nodes of $G_{\mathcal{C}}$ are the clusters of \mathcal{C} and there is an intercluster edge between two clusters if the clusters are at distance 1 from each others. Now, consider a cluster $C \in \mathcal{C}$. Once the construction of C is finished, there are at most $n^{1/k}|C|$ nodes in G at distance 1. Thus, there will be at most $n^{1/k}|C|$ neighboring clusters that will be constructed after C . Thus, there are at most $n^{1/k}|C|$ intercluster edges that can be added to the graph $G_{\mathcal{C}}$ after the construction of C is finished. Thus, the number of intercluster edges is bounded by $\sum_{C \in \mathcal{C}} n^{1/k}|C|$. Since \mathcal{C} is a partition, $\sum_{C \in \mathcal{C}} |C| = n$ and the second property of the partition holds. \square

There are many distributed implementations of the BASIC_PART algorithm. All of these implementations are semi-sequential. First, they distributively elect a new leader in the network which corresponds to the center of a new cluster. Then, the cluster is constructed in a distributed way by adding the layers in many iterations. The construction of the cluster ends when there are no new layers to add or when the sparsity condition is no longer satisfied. Once the construction of the cluster is finished, a new leader is elected from unprocessed nodes and a new cluster grows up around this leader.

The main difficulty in these algorithms is to *distributively elect* the next leader. In [34], a preprocessing is used to overcome this difficulty. First, a spanning tree T of the graph G is constructed. Then, the next leader is elected by achieving a DFS traversal of T . This technique allows us to improve the complexity bounds of the decomposition: $O(|E|)$ messages and $O(|V|)$ time.

In the next sections, we introduce a new algorithm with no precomputation step and no next leader election step.

4 A Deterministic Fully Distributed Basic Partition Algorithm

4.1 Overview of the Algorithm

The main idea of algorithm DIST_PART is to allow clusters to grow in parallel in different regions. In fact, consider two nodes u and v such that $d_G(u, v) \geq 2k$ where k is the same parameter as that in algorithm BASIC_PART. Then, it is possible to grow two clusters respectively around u and v without any interference. Based on this observation, we initially let each node of the graph be a singleton cluster. Then,

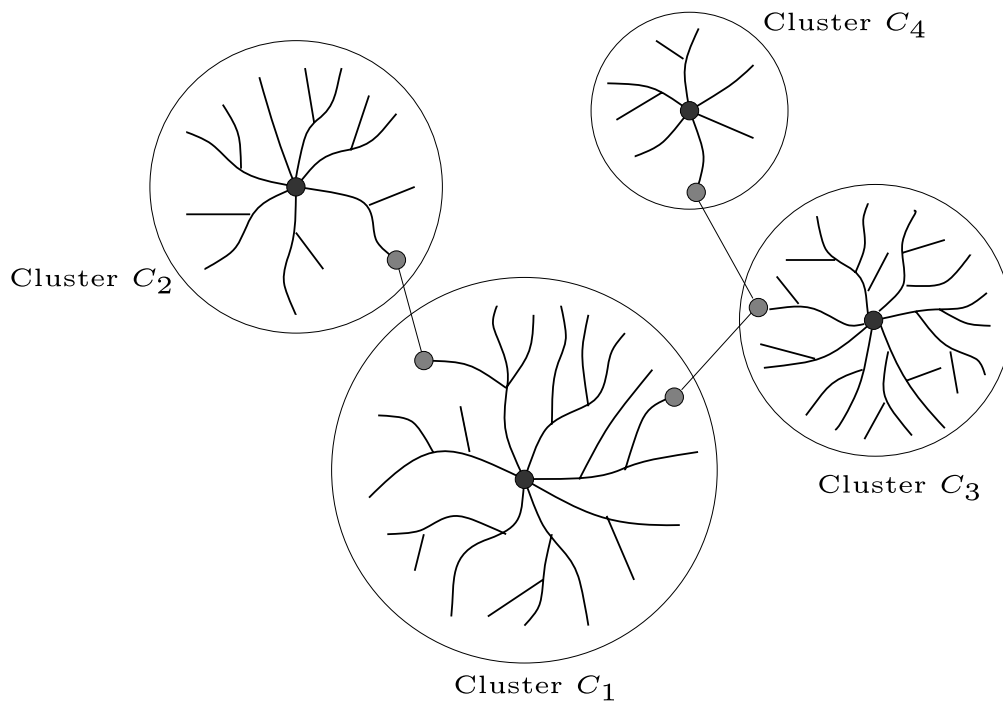


Fig. 2 An example of conflicts between clusters at distance 1 or 2

we allow the clusters to grow spontaneously. The main difficulty here is to guarantee that the clusters do not share any nodes.

We do not avoid cluster collisions but we try to manage the conflicts that can occur. For instance, consider some region of the graph and suppose that some clusters have independently grown as shown in Fig. 2. The clusters cannot add a new layer simultaneously without overlapping. Thus, we make each cluster compete against its neighbors in order to win a new layer. There are two critical situations. Either, a cluster enters in conflict with an adjacent one or with another cluster at distance two. For instance, in the example of Fig. 2, cluster C_1 tries to invade some nodes that belong to cluster C_3 and C_2 at distance 1. Thus, the neighboring cluster C_1 , C_2 and C_3 are in conflicts. Similarly, cluster C_4 tries to invade some nodes in cluster C_3 . Nevertheless, these nodes are also required for the new layer of cluster C_1 . Thus, the two clusters C_1 and C_4 (at distance 2) are also in conflict. To resume, each cluster must compete against all clusters at distance 1 or 2 in order to add a layer. In addition, a layer not satisfying the sparsity condition of algorithm BASIC_PART must be rejected.

In order to manage the conflicts and the cluster growth, we use the following rules:

1. *Exploration Rule*: a cluster is able to add a new layer if its *identifier* is bigger than those of not finished neighboring clusters at distance one or two. If a cluster wins in exploring a new layer then it must apply the *Growth Rule*, otherwise it must apply the *Battle Rule*.
2. *Growth Rule*: If the sparsity condition is satisfied then a cluster adds the last explored layer and tries to apply the *Exploration Rule* once again. Otherwise, the cluster construction is finished and the cluster rejects the last explored layer. The

Fig. 3 Algorithm DIST_PART:
code for a cluster

```

1: continue := True
2: while continue do
3:   execute the Exploration Rule
4:   if success of the Exploration Rule then
5:     add the new layer
6:     execute the Growth Rule
7:     if Non success of the Growth Rule then
8:       reject the last explored layer
9:       switch to a finished cluster
10:      continue := False
11:    end if
12:  else
13:    execute the Battle Rule
14:  end if
15: end while

```

nodes in the rejected layer are re-initialized to singleton clusters with their initial identifiers.

3. *Battle Rule*: a cluster loses its *whole* last layer if at least one neighboring cluster at distance one has successfully applied the *Exploration Rule*. The nodes lost by a cluster are re-initialized to singleton clusters with their initial identifiers.

Based on the three previous rules, we obtain the fully distributed algorithm DIST_PART described in a high level way in Fig. 3.

Remark 1 It is important to choose a unique identifier for each cluster. For instance, the identifier of a cluster can be chosen to be the identity of its leader. This is implicitly assumed in the rest of this section. However, we can also choose the couple $(|C|, Id_v)$ as the identifier of a cluster C with a root v , and the lexicographical order to compare cluster identifiers.

Example Let us consider the concrete example of Fig. 4. We have five clusters 1, 2, 3, 4 and 5 with identities $Id_1 > Id_2 > Id_3 > Id_4 > Id_5$. Assume that the identifier of each clusters corresponds to the identity of its leader node. When a new exploration begins, cluster 1 wins against clusters 5 and 3. Cluster 2 wins against clusters 4 and 5 but loses against cluster 1 which is at distance two. Thus, cluster 2 cannot add a new layer. Cluster 4 loses against both clusters 2 and 3 but it will not be invaded because both clusters 2 and 3 cannot grow. Cluster 3 wins against cluster 4 but loses against cluster 1. Cluster 3 will be invaded by cluster 1 which wins against all clusters at distance two (cluster 5, 2 and 3). Thus, cluster 3 will lose its last layer. The node connecting it with cluster 4 becomes a singleton cluster with its initial identity Id_6 . The node connecting cluster 3 with cluster 1 becomes a leaf in cluster 1. Now, suppose that the sparsity condition for the new enlarged cluster 1 is not satisfied. Then, cluster 1 rejects the last explored layer and its construction is finished. Hence, the nodes in the rejected layer become singleton clusters. Then, the remaining active clusters spontaneously continue new explorations. In our example, both cluster 2 and cluster 3

will succeed their explorations and add a layers. Note that in the other regions of the graph, there are other clusters which are fighting against each others. Hence, many clusters can grow in parallel.

4.2 Detailed Description and Implementation

In this section, we give a complete description of how to implement the three rules of the `DIST_PART` algorithm using message passing. For the clarity of our algorithm, we assume that the identifier of a cluster is the identity of its root. The main difficulty when implementing the three rules of algorithm `DIST_PART` is to coordinate the center of a cluster with its leaves, i.e., nodes at the border of the cluster. On one hand, the center of a cluster cannot see what is happening on the borders of its cluster. Hence, it must always wait for pieces of information from the leaves before making a decision. Symmetrically, the leaves cannot see the global state of their cluster. Hence, they must also wait for information from the center of their cluster.

To apply the three rules, the nodes in a cluster must collaborate. Each node can be in five states *root*, *leaf*, *relay*, *orphan* or *final*. At the beginning, all nodes are orphans and they form *orphan clusters*, i.e., cluster with only one node. If a node is in a *final* state, then it belongs to a finished cluster and thus it does not make any computation. Roughly speaking, if a node v is in a *root* state, then it is the leader and it makes decisions for its cluster. If v is in a *leaf* state, then it tries to invade new nodes and it informs its root. If v is in a *relay* state, then it forwards information from the leaves to the root.

As long as new layers are added (resp., removed) to (resp., from) a cluster, the nodes in the cluster maintain a layered BFS spanning tree. The root of the tree corresponds to the root of the cluster, the leaves of the tree correspond to the leaves of the cluster and the nodes in the interior of the tree correspond to relay nodes. The decisions of adding or removing a layer are broadcasted by the root node according to the information forwarded by the leaves all along the constructed BFS tree. Each time that some new nodes join a cluster, the BFS spanning tree is enlarged by making each new node choose a parent among the leaves of the already constructed tree.

In next paragraphs, we detail the actions to be performed by each node according to its state. Notice that the state of a node can change several times. For instance, the state of a node can be relay at some time, then becomes leaf, after that orphan, and at last final.

Remark 2 In the pseudo-code of our algorithms, a node uses the function *Send* to send a message to a (or some) neighbor(s). The function *Receive* allows a node to receive a message from a (or some) neighbor(s). The receive function is blocking, that is, a node cannot execute the next instruction in the algorithm unless the receive action is terminated, i.e., all messages were arrived.

Root Nodes The algorithm executed by a root node is given in Fig. 5. First, the root verifies if the sparsity condition is satisfied and it informs the leaves (*Growth rule*).

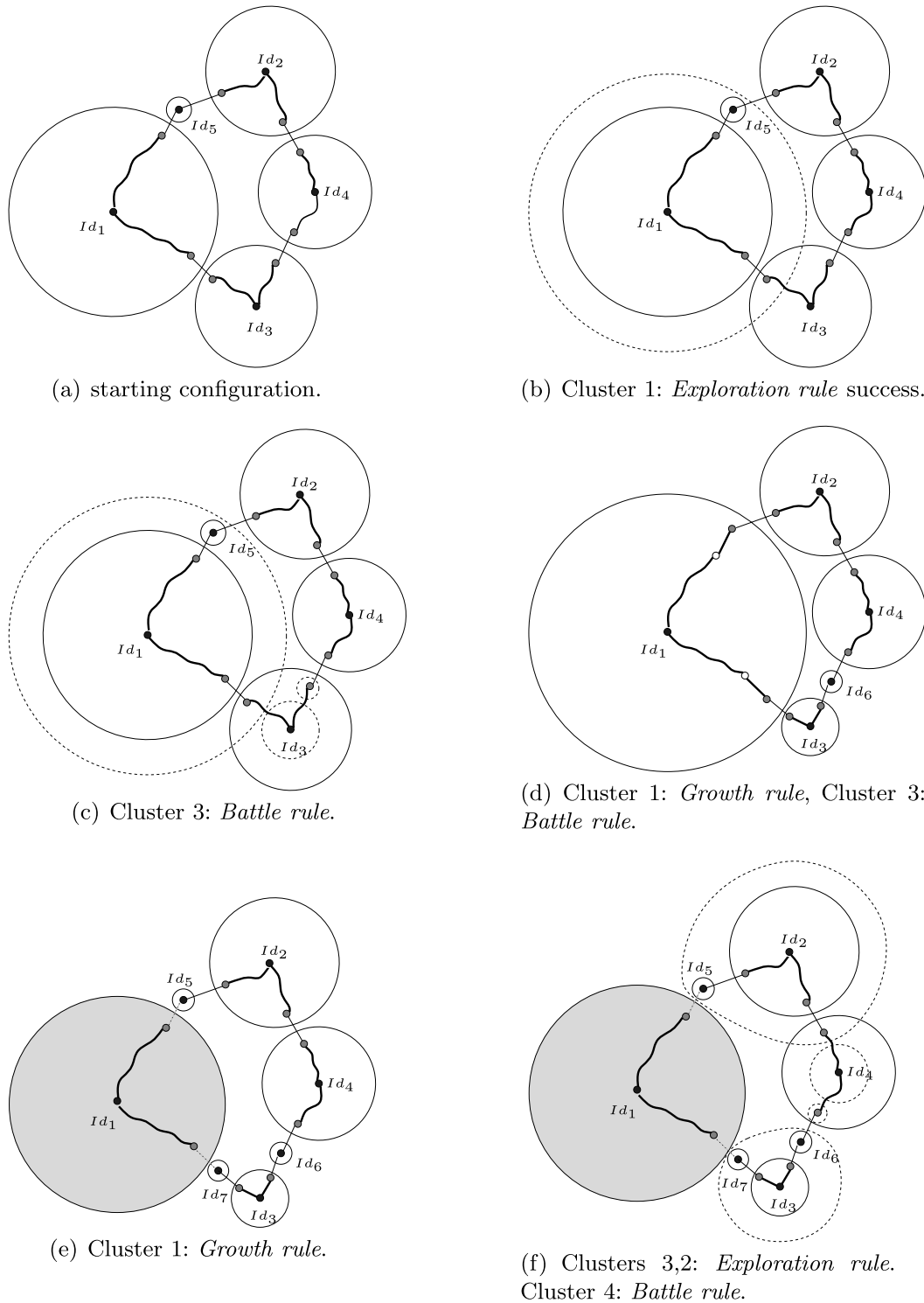


Fig. 4 An example of algorithm DIST_PART

More specifically, if the sparsity condition is satisfied, then the root broadcasts a notification message *NEW* to the leaves in order to begin a new exploration. Otherwise, it broadcasts a *REJECT* message saying that the construction is finished.

```

1 Receive count From Children; Compute  $|\Gamma(C)|$ ;
2 if  $|\Gamma(C)| > n^{1/k}|C|$  then
3   Send NEW To Children ;
4   Receive LOST_WIN_STOPPED From Children;
5   if there exists at least one LOST message then
6     exploration_success := false ;
7   else if all messages are STOPPED then
8     cluster_stopped := true;
9   else
10    exploration_success := true ;
11  if cluster_stopped then
12    Send STOP To Children ; State := Final ;
13  else
14    if exploration_success then
15      Send SUCCESS To Children ;
16       $h := h + 1$ ; /* the radius of the cluster */
17    else
18      Send FAILURE To Children ;
19      Receive BYE_SAFE From Children ;
20      if there exists at least one BYE message then
21        Send DOWN To Children ;
22         $h := h - 1$  ;
23        if  $h = 1$  then State := Orphan ;
24      else
25        Send OK To Children ;
26  else
27     $h := h - 1$ ;
28    State := Final ;
29    Send REJECT To Children ;

```

Fig. 5 DIST_PART: high level code for the *root node* of a cluster C

After broadcasting a *NEW* message, the root waits the response from its leaves. There are three possible cases:

1. The root receives only *STOPPED* messages from its leaves. This means that the leaves did not find new nodes to explore. In this case, the root broadcasts a *STOP* message informing all the nodes that the cluster construction is finished.
2. The root does not receive any *LOST* message, i.e., only *WIN* (or *STOPPED*) messages. This means that the exploration was globally successful. Thus, the root broadcasts a *SUCCESS* message to the leaves. Then, the root waits to learn the size of the new enlarged cluster.
3. The root receives at least one *LOST* message. This means that the exploration was not successful (at least one leaf has lost against a neighboring cluster). Thus, the root informs the leaves by broadcasting a *FAILURE* message. Then, the root waits for the leaves responses. There are two cases:

- At least one leaf is invaded by another cluster. Thus, the root must receive at least one *BYE* message. In this case, the cluster must reject its last layer (*Battle rule*). Hence, it broadcasts a *DOWN* message asking the leaves to become *orphans*.
- All leaves have resisted to neighbor's attacks. Thus, the root must receive only *SAFE* messages, i.e., no neighboring cluster has succeeded in invading the current cluster. In this case, the root broadcasts an *OK* message saying that the cluster is not invaded and asking for a new exploration.

Leaf Nodes The algorithm executed by a leaf is given in Fig. 6.

Remark 3 We remark that a leaf does not always belong to the last layer of a cluster. For instance, a leaf node may have only final neighbors belonging to finished clusters. Hence, it cannot add new nodes to its cluster. Nevertheless, other leaves belonging to the same cluster can continue exploring new nodes. Therefore, the construction of the cluster can continue even if some leaves cannot locally explore new nodes. In order to handle this situation, for each node we use a local variable h which corresponds to its depth in the BFS-spanning tree of its cluster. If $h = 1$ then the leaf belongs to the last layer and it can compete to add new layers, otherwise the leaf cannot explore any new layer.

Since the exploration of a new layer is done before verifying the sparsity condition, whenever a node u becomes a leaf in a new cluster, it sends 1 to its parent v in the new cluster and waits for a message from its parent. The parent node v sends back the number of its children and so on. Thus, by a convergecast process, the root will be able to compute the size of the new cluster and to broadcast its decision to the leaves.

If the leaf u receives a *REJECT* message from its parent, then u leaves its new cluster (*Growth rule*) and it becomes an orphan cluster. Otherwise, u receives a *NEW* message from its parent. This means that a new layer must be explored. If u cannot explore new regions, then u sends back a *STOPPED* message. Otherwise, u begins a new exploration using an election technique in a ball of radius two: First, u sends its cluster identifier to its neighbors. Symmetrically, it waits for the identifiers of the neighboring clusters. Second, u computes the maximum of the neighbor identifiers (including the identifier of its own cluster) and sends it again to the neighbors. Symmetrically, it waits for the maximum identifiers sent by neighboring leaves. If all the identifiers received by u are equal to the identifier of u 's cluster, then u has locally succeed its exploration and it sends back a *WIN* message. Otherwise, u sends back a *LOST* message.

Remark 4 Since the clusters have unique identifiers, then two neighboring leaves can easily decide whether they belong to the same cluster, e.g., when exchanging their identifiers in a new exploration.

Once the exploration is finished, the leaf node u waits for the decision of its root. There are three cases:

```

1  Send 1 To parent ; Receive msg From parent;      /* msg is either NEW or REJECT */
2  if msg = NEW then
3      if there are new nodes to explore then
4          Send  $Id_C$  To neighbors in other active clusters;
5          Receive  $\cup_{C'} Id_{C'}$  From neighbors in other active clusters ;
6           $max_1 :=$  the maximum of  $\cup_{C'} Id_{C'}$  ;
7          if  $max_1 < Id_C$  then
8              Send  $Id_C$  To neighbors in other active clusters;
9          else
10             Send  $max_1$  To neighbors in other active clusters;
11             Receive Ids From neighbors in other active clusters ;
12              $max_2 :=$  the maximum of received Ids;
13             if  $max_2 > Id_C$  then
14                 Send LOST To parent ;
15             else
16                 Send WIN To parent ;
17         else
18             Send STOPPED To parent ;
19         Receive msg From parent;      /* msg is either SUCCESS or STOP or FAILURE */
20         if msg = SUCCESS then
21             Send JOIN To neighbors in other active clusters ;
22             Receive messages From neighbors in other active clusters ;
23             Mark the new Children in the BFS tree of  $C$ ;  $h := h + 1$ ; State := Relay;
24         else if msg = STOP then
25             State := Final ;
26         else
27             Send STAY To neighbors in other clusters ;
28             Receive messages From neighbors in other active clusters ;
29             if there exists at least one JOIN message then
30                 Send BYE To parent ;
31                 Receive msg From parent;      /* the message must be a DOWN message*/
32                 choose a new parent in the new winner cluster;  $h := 1$  ;
33             else
34                 Send SAFE To parent; Receive msg From parent ;
35                 if msg = DOWN then
36                     if  $h = 1$  then State := Orphan ;
37                     if  $h \neq 1$  then  $h := h - 1$  ;
38     else
39         if  $h = 1$  then State = Orphan ;
40         if  $h \neq 1$  then  $h := h - 1$ ; State = Final ;

```

Fig. 6 DIST_PART: high level code for a *leaf node* in a cluster C

1. If u receives a *STOP* message from the root, then none of the leaves can explore new nodes. Thus, u becomes a *final* node, i.e., the construction of the cluster is finished.
2. If u receives an *SUCCESS* message from the root, then all leaves have succeeded their local explorations, i.e., they won against all neighbors at distance 1 or 2. Thus, u sends a *JOIN* message to neighboring leaves asking them to join its cluster. Then, u switches to a relay state.

3. If u receives a *FAILURE* message from the root, then at least one leaf has not succeeded the exploration. Thus, u sends a *STAY* message to neighboring leaves informing them that they will not be invaded by u 's cluster. Then, u waits to know if the neighboring clusters succeeded their explorations. There are two cases:
 - If u receives at least a *JOIN* message from a neighboring leaf (in a different cluster), then it sends back a *BYE* message to its root, waits for an acknowledgment (*DOWN* message) and it joins the new cluster.
 - Otherwise, if none of the neighboring cluster has succeeded in invading the leaf (*STAY* message), the leaf sends back to its root a *SAFE* message. At this stage of the algorithm, the leaves (except those who have received a *JOIN* message) do not know whether their cluster is being invaded or not (only the root globally knows what is happening at its frontiers). Thus, the leaves wait for either an *OK* or a *DOWN* message from the root. If a leaf receives an *OK* message, then it remains in the same cluster and it begins a new exploration once again. Otherwise, it receives a *DOWN* message and it becomes an orphan node.

Orphan Nodes An orphan node acts like a root and like a leaf node. In fact, it makes decisions for its singleton cluster and it fights against neighboring nodes. If an orphan node succeeds an exploration, it becomes a root node in a new cluster of radius 1. If it is invaded by a cluster, it becomes a leaf. Otherwise, it re-tries to invade its neighbors (new exploration). If it has only neighbors belonging to finished clusters, then it switches to a final state. The type of messages that must be sent by an orphan node to neighbors can be deduced from the previous discussion.

Relay Nodes The main role of a relay node is to forward information from the root to the leaves. If a relay node receives a message from its parent, it simply forwards it to its children. If the message is a *REJECT* or a *STOP* message, then the node knows that the cluster construction is finished and it switches to a final state. If the message is a *SUCCESS* message, then the node knows that there is a new layer that will join the cluster. Thus, the depth of the node is incremented by one. If the message is a *DOWN* message then the relay node knows that its cluster was invaded and lost the last layer. In this case, if a relay node belongs to the layer before the last one ($h = 2$) then the relay node becomes a leaf.

On the other hand, if a relay node receives a message from its children, it can deduce which step the leaves are executing (exploration of a new layer: *WIN*, *LOST* or *STOPPED* messages, resistance against neighbors attacks: *OK* or *BYE* messages, and computation of the sparsity condition: integer message). In all cases, the relay node can easily compute what kind of message it must forward to its root.

Remark 5 Each node can easily know which of its neighbors belongs to a finished cluster. It is sufficient to make each node (which becomes final) send a message to its neighbors to inform them. However, we can avoid these extra communication messages as following. When a node v is explored by a cluster C , it uses the *Ids* sent by neighbors in order to compute a set \mathcal{F}_C of neighbors belonging to the layer before the last one. Then, if v receives a *REJECT* message from the root of C , i.e., the sparsity condition for the last layer of C is not satisfied, then v marks its neighbors in

\mathcal{F}_C as finished. Now, consider any edge (u, v) . Suppose that u and v do not belong to the same cluster at the end of the algorithm. W.l.o.g., suppose that u becomes in a final state before v . Thus, the cluster containing u must have explored v . Thus, v must have received a *REJECT* message from its parent. Thus, v can decide that u switched to a final state. The case where nodes u and v ends up in the same cluster is trivial since both the two nodes stop communicating.

Remark 6 Although our algorithm is completely asynchronous, we remark that there is a kind of synchronization in our implementation which is close to the one used in synchronizer γ (see Appendix A). On one hand, the root nodes control the execution of the algorithm and give the starting signal to all the actions of the leaves using the relay nodes. Hence, the actions of nodes inside one cluster are synchronized. On the other hand, the decisions made by the roots in neighboring clusters are synchronized since a root must wait for some information concerning those neighboring clusters. The leaves in different clusters synchronize also their actions to execute the decisions of their roots.

4.3 Analysis of the Algorithm

Theorem 2 *Algorithm DIST_PART terminates.*

Proof From the algorithm description, the cluster having the biggest identifier in the graph always succeeds the *Exploration rule*. Thus, it always succeeds adding new layers until the sparsity condition is violated. Thus, after at most $k - 1$ layers, the nodes in the biggest cluster are in final states. Now, the remaining cluster with the biggest identifier always succeeds its new explorations and so on until all the nodes are in final states. \square

Theorem 3 *Algorithm DIST_PART emulates the BASIC_PART algorithm.*

Proof From the algorithm description, once the construction of a cluster is finished, the cluster cannot be invaded by any other active cluster. Hence, the constructed clusters are disjoint.

In addition, a new layer is added if and only if it verifies the sparsity condition (*Growth rule*). Symmetrically, if a cluster is invaded, then it loses its whole last layer. Hence, the layers of finished clusters satisfy the sparsity condition.

Thus, the constructed partition satisfies the sparsity and locality properties of algorithm BASIC_PART. \square

Theorem 4 *In the worst case, the time complexity of algorithm DIST_PART is:*

$$Time(DIST_PART) = O(n)$$

Proof In the following proof, we consider the clusters in an increasing order of the time of their construction. Let C be a cluster in the final partition \mathcal{C} . Let r be the radius of C . We remark that the construction of a cluster always ends with a sequence

of successive successful explorations, and in all these explorations except for the last one the growth condition holds. Consider the first time t when the cluster C starts to successively completes all its explorations. Let $Time(C)$ be the number of time units from t to the end of C 's construction. In other words, $Time(C)$ is the duration of the successive successful explorations. Let j be the radius of C at time t . For any $i \in \{j, \dots, r\}$, we consider the time when C contains i layers, and we denote by r_{max_i} the maximum radius of the neighboring clusters of C .

In order to decide if a layer is added or not, the cluster C must be traversed at most a constant number of times. In addition, before a node joins a new cluster, it informs its previous root and waits for the acknowledgment of this root. Thus, $Time(C) \leq \sum_{0 < i \leq r} O(i + r_{max_i})$. Using Theorem 1, we have $r \leq k - 1$ and $r_{max_i} \leq k - 1$. Thus, $Time(C) = O(kr)$.

In the worst case, two clusters are never constructed in parallel. Thus,

$$Time(DIST_PART) = \sum_{C \in \mathcal{C}} Time(C)$$

Hence, using the fact that $\sum_{C \in \mathcal{C}} r \leq n$, we get

$$Time(DIST_PART) = O(k \cdot n)$$

The previous analysis is not sufficient to prove the theorem if the parameter k is not a constant. Nevertheless, it gives us a precious remark. In fact, we remark that it can be interesting to take the couple $(Rad(C), Id_v)$ to be the identifier of a cluster C rooted at a node v and the lexicographical order to compare cluster identifiers (which do not change the overall implementation). In this case, we have $r_{max_i} \leq r$. Thus, for the relevant range of $k \leq \log(n)$, we have:

$$|C| \geq n^{r/k} \Rightarrow r \leq \frac{k}{\log(n)} \log(|C|) \Rightarrow r \leq \log(|C|)$$

Thus,

$$\begin{aligned} Time(DIST_PART) &= \sum_{C \in \mathcal{C}} \sum_{0 < i \leq r} O(i + r_{max_i}) \leq \sum_{C \in \mathcal{C}} O(r^2) \\ &\leq \sum_{C \in \mathcal{C}} O(\log(|C|)^2) \\ &\leq \sum_{C \in \mathcal{C}} O(|C|) \end{aligned}$$

Since \mathcal{C} is a partition, the theorem holds. □

Remark 7 Since at each time unit nodes could exchange order of $|E|$ messages in a worst case scenario, the message complexity of our algorithm can be rather large. However, this does not take into account the fact that finished clusters stop communicating with their neighbors. In this paper, we are mainly interested in the time complexity and we will not consider improving the message complexity measure.

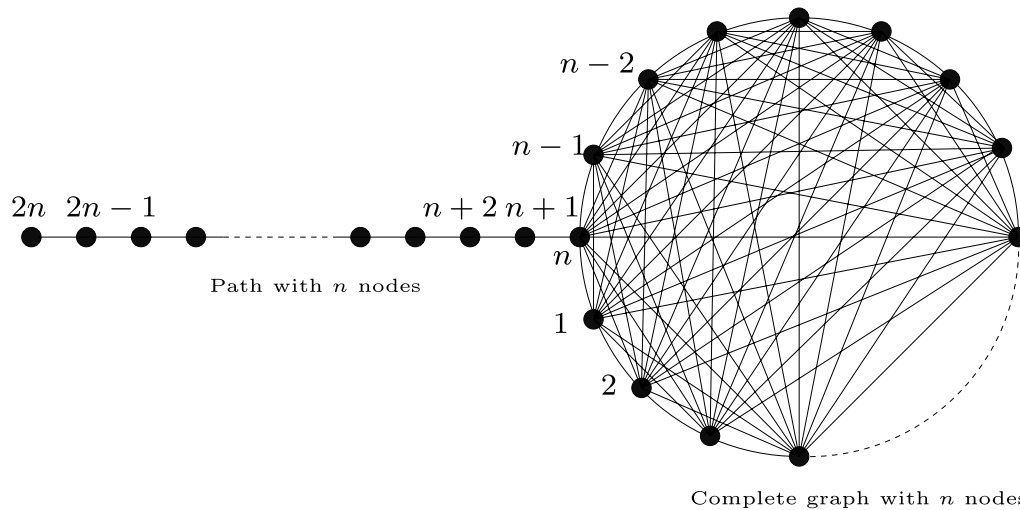


Fig. 7 An example of bad node distribution

Remark 8 One shall remark that our theoretical analysis is still sequential. In fact, in our analysis we consider that clusters are never constructed in parallel. This can actually happen for instance if the graph contains a path with nodes having a decreasing sequence of identities. In Fig. 7, such a bad scenario is illustrated. In fact, at each round of the algorithm execution there will be only one cluster constructed. At the beginning, all nodes but node number $2n$ fail to explore a new layer. Then, node number $2n$ switches to a final state, and only node number $2n-1$ wins the exploration and so on. Thus, it takes $O(n)$ time to terminate the construction. Also in the example of Fig. 7, there will be order of $O(n^2)$ messages exchanged at each new exploration round. In fact, the nodes of the clique will not be able to switch to final states before the last round. Hence, the message complexity is large. Nevertheless, by considering a random permutation of node identities, one can see that the path in Fig. 7 is likely to be broken in many pieces rather quickly allowing more than only one cluster to grow in parallel. This leads to a better time and message complexity in practice. For the general case of any graph, we think that the average complexity of our algorithm can be much better than the worst case bound of Theorem 4. It would be very nice to prove this claim analytically. This could be a hard task since one have to consider any connected graph with n nodes and all possible permutations of node identities.

5 Sublinear Deterministic Distributed Partition

In the following, we show how to improve algorithm `DIST_PART` in order to obtain sublinear time algorithms for constructing a basic partition. First, we describe and analyze a new synchronous algorithm called `SYNC_PART`. Then, we show that the synchrony of the network is not important to achieve a sublinear time construction, and we provide a new asynchronous algorithm called `FAST_PART`.

In the remainder, we denote by V_f the set of finished nodes, i.e., nodes in a finished cluster. Furthermore, we are interested in active nodes in $V - V_f$, hence the degree d_v of a node v is defined as its degree in the graph G_{V-V_f} induced by $V - V_f$.

5.1 A Synchronous Deterministic Algorithm

In this section, we assume that the network is synchronous, i.e., there exists a global clock. At any time t , A_t denotes the set of active nodes (nodes not in V_f at time t), and $R_t = \{v \in A_t \mid d_v > n^{\frac{1}{k}}\}$ denotes the set of active nodes having high enough degrees at time t .

We remark that the sparsity condition for a singleton cluster rooted at some node v is $d_v > n^{\frac{1}{k}}$. Hence, a singleton cluster rooted at some node in $A_t \setminus R_t$ cannot grow any layer. Thus, at any time t , we only let the nodes in R_t compete in order to grow some clusters. Once R_t becomes empty, we just let the remaining active nodes be finished singleton clusters.

The new algorithm SYNC_PART works in two stages. The first stage is performed until time $\mathcal{T} = O(k^2 n^{1-1/k})$ is reached. The second stage begins at time \mathcal{T} and lasts $O(1)$ time units.

In the next paragraphs, we give the details of algorithm SYNC_PART and discuss its correctness and its complexity.

First Stage of the Algorithm During this stage, all nodes execute algorithm DIST_PART with the following additional exploration rules:

- If a node $v \in A_t$ is no longer in R_t , i.e., $v \in A_t \setminus R_t$, then v sets its identity to $-\infty$.
- Singleton clusters rooted at nodes in $A_t \setminus R_t$ do not explore any layer.

Notice that the previous modifications are made only by singleton clusters that do not verify the sparsity condition. We use the same three rules of algorithm DIST_PART to manage the growth of other clusters rooted at any node in R_t .

Let us consider a singleton cluster rooted at $v \in A_t \setminus R_t$. Then, when applying the new rules, v sets its identity to $-\infty$. Hence, v has the lowest identity among all other possible identities. Therefore, node v will not stop the growth of another cluster rooted at a node of R_t . In fact, v can only be a part of other neighboring dense clusters (if it is asked to join). If the neighborhood of v is also in $A_t \setminus R_t$, then the cluster behaves as if it has the lowest identity, i.e., it does not explore any layer. In a practical implementation, a node needs to know whether it is in R_t or not. Since at any moment of algorithm DIST_PART a node is aware of its finished neighbors (see Remark 5), there are no further communications to be done by a node in order to know if it is still in R_t .

Second Stage of the Algorithm At time \mathcal{T} , all remaining active nodes in A_t stop computing and just decide to be finished singleton clusters.

Proposition 1 (Correctness) *Algorithm SYNC_PART emulates algorithm BASIC_PART.*

Lemma 1 *Let C be the cluster with the biggest identity among active nodes at some moment of the algorithm. We need $O(k^2)$ time in the worst case before the construction of C is finished.*

Proof On one hand, the communications performed by the algorithm are done using a broadcast convergecast process inside the BFS spanning tree of each cluster. Since a cluster has a radius at most $O(k)$, a broadcast (or a convergecast) costs at most $O(k)$ time units.

On the other hand, using the *Exploration rule*, cluster C always wins against its neighboring clusters and it always succeeds in exploring new layers. In the worst case, there will be at most $k - 1$ new explored layers. Thus, it takes at most $O(k \cdot k)$ time before the construction of C is finished. \square

Lemma 2 *For any time t either $|A_{t+O(k^2)}| < |A_t| - n^{1/k}$ or $R_{t+O(k^2)} = \emptyset$.*

Proof Consider a given time t . If $R_t = \emptyset$ then $R_{t+O(k^2)} = \emptyset$ (because a finished node will never be active again). In the remainder of the proof we consider the less trivial case where $R_t \neq \emptyset$.

Consider a node v in R_t , i.e., v has more than $n^{1/k}$ active neighbors at time t . Node v must belong to some cluster C . Let u be the root of C . Suppose that u is active at time t , then u must be in R_t (because sparse nodes cannot explore new layers). The cluster having the biggest identity will end up having radius at least 1 and size at least $n^{1/k}$. The vertices of that cluster become inactive and so at least $n^{1/k}$ vertices become inactive. Using Lemma 1, the construction of the cluster having the biggest identity is terminated within at most $O(k^2)$ time. Thus, $|A_{t+O(k^2)}| < |A_t| - n^{1/k}$.

Suppose now that for each node $v \in R_t$, the root of the cluster containing v at time t is inactive. This may happen since it may take some time for a root node to inform the other nodes in its cluster that the construction of the cluster is finished. This information takes at most $O(k)$ time to reach a node v . Thus at time $t' = t + O(k)$, either v becomes inactive or $v \in R_{t'}$ or $v \in A_{t'} \setminus R_{t'}$. Thus, we have two relevant cases:

- No node v becomes in $R_{t'}$. This means that $R_{t'} = \emptyset$ and the second property of the lemma holds.
- At least one node v becomes in $R_{t'}$. By considering the cluster having the biggest identity at time t' and using Lemma 1, we have that $|A_{t'+O(k^2)}| < |A_{t'}| - n^{1/k}$. Since $|A_{t'}| \leq |A_t|$ and $t' + O(k^2) = t + O(k) + O(k^2) = t + O(k^2)$, we can conclude that $|A_{t+O(k^2)}| < |A_t| - n^{1/k}$. Thus, the first property of the lemma holds.

In all cases, either the first property or the second property of the lemma holds. \square

Lemma 3 *For $t = O(k^2 n^{1-1/k})$, $R_t = \emptyset$.*

Proof Using Lemma 2, at the beginning of the algorithm we have $|A_{O(k^2)}| < |A_1| - n^{1/k}$ or $R_{O(k^2)} = \emptyset$. If $R_{O(k^2)} = \emptyset$ then the lemma holds. Otherwise, we have $|A_{O(k^2)}| < |A_1| - n^{1/k}$. Let $t = O(k^2)$. Again by Lemma 2, we have $|A_{t+O(k^2)}| < |A_t| - n^{1/k}$ or $R_{t+O(k^2)} = \emptyset$. If $R_{t+O(k^2)} = \emptyset$ then the lemma holds. Otherwise, we

have $|A_{t+O(k^2)}| < |A_t| - n^{1/k}$. Hence it is easy to see that each $O(k^2)$ time period p , either the set of active nodes decreases by at least $n^{1/k}$ factor or all nodes become sparse. Thus, by a simple induction, after at most $n^{1-1/k}$ periods p either set A becomes empty or set R becomes empty. Since at any time t , $R_t \subseteq A_t$, then we can conclude that $R_{O(k^2n^{1-1/k})} = \emptyset$. \square

Since the first stage of the algorithm costs $\mathcal{T} = O(k^2n^{1-1/k})$ time units and the second one is performed in $O(1)$ time units, we get the following theorem:

Theorem 5 (Time Complexity) *The time complexity of algorithm SYNC_PART is $O(k^2 n^{1-1/k})$.*

Remark 9 We remark that since we are interested in small values of k (typically constant values of k), the time complexity of our algorithm is $O(n^{1-1/k})$. However, we can show that for any k verifying $k < \log n$, the value of \mathcal{T} in the previous algorithm can be chosen to be equal to $O(n^{1-1/k})$. To do that, we slightly modify our algorithm by privileging the growth of clusters having the biggest couple $(Radius, Id)$. The proof of this claim is technically similar to the proof of Theorem 4. In fact, consider a cluster C rooted at some node v in R_t . Consider the cluster C having the biggest couple (r, Id_v) where r is the radius of C and Id_v the identity of v . Similarly to the analysis of Theorem 4, it takes at most $O(r)$ time to explore a new layer. Moreover cluster C contains at least $n^{r/k}$ nodes that will be removed from set A_t once the construction of C is finished. Let $\ell \geq 1$ the radius of C in the final partition. Overall, it costs $O(\ell^2)$ time to construct cluster C . Moreover at least $n^{\ell/k}$ nodes become inactive at the end of the cluster construction. Now, for $k < \log n$, we have that $n^{\ell/k} / \ell^2 = \Omega(n^{1/k})$. Thus, by considering the clusters in an increasing order of their time construction, it is easy to see that the way that our algorithm removes nodes from set A_t is equivalent to removing at least $\Omega(n^{1/k})$ nodes each $O(1)$ time units. Thus, after at most $O(n^{1-1/k})$ time units, either set A_t becomes empty or R_t becomes empty. Thus, after $O(n^{1-1/k})$ time units, no clusters with radius at least 1 can be constructed. Hence, \mathcal{T} can be chosen to be $O(n^{1-1/k})$ and the factor k^2 can be removed in the time complexity.

5.2 An Asynchronous Deterministic Algorithm

Algorithm SYNC_PART uses the property that the system is synchronous to find a bound on the time \mathcal{T} before no nodes can grow a non zero radius cluster. The time \mathcal{T} informs all remaining active nodes that there are no more active dense clusters in the graph. This compels us to wait \mathcal{T} time units even if the input graph is sparse. Furthermore, algorithm SYNC_PART cannot be run in an asynchronous system without using any synchronizers (see, e.g., Appendix A). In the following, we give a new asynchronous algorithm FAST_PART which does not use any global clock. The general idea of the algorithm is to allow sparse clusters to become finished without waiting until pulse \mathcal{T} . Our asynchronous algorithm shows that the key point for speeding up the construction does not rely on the global synchrony of the system, but rather on more local parameters.

Details of the Algorithm Let us call a cluster C *dense*, if C has a radius at least 1 or if the single node v of C verifies $d_v > n^{\frac{1}{k}}$. We also define a *sparse cluster* to be a singleton cluster which is not dense (this corresponds to a node in $A_t \setminus R_t$ in algorithm SYNC_PART).

Algorithm FAST_PART uses the three rules of algorithm DIST_PART with the following modifications:

- A dense cluster can explore a new layer if it has an identity bigger than those of its active dense neighbors at distance one or two.
- A sparse cluster is not allowed to explore a new layer.
- A sparse cluster declares itself finished singleton cluster if:
 - all its neighbors are sparse,
 - or if none of its dense neighbors has succeeded in exploring a new layer.

Using these new rules, a sparse node is allowed to declare itself finished if it is not explored by any neighboring cluster. This occurs if all neighbors are sparse or if the dense neighbors have not succeeded their explorations. This simple idea enables us to improve the time complexity of the previous synchronous algorithm.

It is obvious that the new modifications can be implemented using messages of size at most $O(\log(n))$ using the same techniques than in algorithm DIST_PART. For instance, we can use a couple $(Id, Dense)$ for the cluster identifiers, where *Dense* is a boolean variable indicating whether a cluster is dense or sparse.

Proposition 2 (Correctness) *Algorithm FAST_PART emulates algorithm BASIC_PART.*

Let Λ be the number of clusters of radius at least 1 at the end of algorithm FAST_PART. Then, the following theorem holds:

Theorem 6 (Time Complexity) *The worst case time complexity of algorithm FAST_PART satisfies:*

$$\text{Time}(\text{FAST_PART}) = O(k^2 \Lambda) = O(k^2 n^{1-\frac{1}{k}})$$

Proof The new rules guarantee that a dense cluster is never stopped by a sparse one. In the worst case, no two dense clusters are constructed in parallel. Thus, let us consider the finished dense clusters in a *decreasing* order of their time construction.

The construction of a cluster costs at most $O(k^2)$. Thus, after at most $O(k^2 \Lambda)$ time, it only remains active sparse clusters in the graph. In two rounds, all remaining sparse clusters detect that their neighbors are sparse. Thus, using the new rules, they become finished clusters and the algorithm terminates. Thus, the first part of the theorem holds. In addition, since the cluster are disjoint, it is obvious that for any graph and for any execution of the algorithm, Λ is bounded by $n^{1-\frac{1}{k}}$ which completes the proof. \square

Remark 10 Note that we can apply Remark 9 for the asynchronous algorithm FAST_PART in order to remove the k^2 factor and obtain a $O(n^{1-\frac{1}{k}})$ time complexity.

Remark 11 The bound $O(k^2 \Lambda)$ becomes of special interest in the case of graphs where Λ can be shown to be small compared to $n^{1-\frac{1}{k}}$, e.g., see Appendix B for the case study of Circulant graphs.

Remark 12 A nice property of algorithm FAST_PART is to privilege the clustering of dense regions of the graph. For instance, if we consider a graph with only some few dense regions, e.g., some cliques connected by some paths. Our algorithm will automatically capture the topology of the underlying graph and the clustering will have a high priority at those dense regions. In the example of Fig. 7, algorithm FAST_PART constructs a basic partition in constant time whereas algorithm DIST_PART needs $O(n)$ time.

6 Sublinear Randomized Distributed Partition

Although, the previous deterministic algorithms allow us to construct clusters in parallel, their analysis is still sequential. In this section, we give a new randomized algorithm enabling us to compute a lower bound of the number of clusters constructed in parallel.

6.1 Randomized Local Elections

In [33], a randomized algorithm called L_2 -election (LE_2 for short) is introduced in order to implement distributed algorithms described with Closed Star (CS for short) relabeling systems. Relabeling systems can be considered as a formal tool to describe and to prove distributed algorithms independently of the underlying model of communication. The reader can refer to [14, 26, 31, 32] for a review on the mathematical foundations of relabeling system. Roughly speaking, a distributed computation step in the CS relabeling system formalism consists in relabeling the nodes attached to a star according to a precise relabeling rule. This encodes the fact that local computations done by a node in a distributed environment can be viewed as a function of the states (labels) of its neighbors. The execution of a distributed algorithm is then described as a sequence of relabeling steps. Each relabeling step changes the labels of a star in the graph according to some rule. The relabeling could be executed in parallel in different regions of the graph at the condition that the corresponding stars do not intersect. In this context, algorithm LE_2 [33] is a message passing algorithm used to implement any formal algorithm described using the relabeling system formalism. Algorithm LE_2 works in fact in rounds where at each round some nodes are elected centers of some stars. In other words, at each round, algorithm LE_2 computes some disjoint stars to be relabeled in parallel.

Algorithm LE_2 is based on the following simple idea. At each round, a node chooses a random number. If the number chosen by a node is bigger than the number chosen by its neighbors at distant 2, then the node is elected center of a star. Thus, the stars centered at the elected nodes can be relabeled in parallel since they are disjoint. Now, suppose that we want to construct the basic partition for $k = 2$. By Theorem 1, the radius of a cluster is at most 1. Thus, we remark that we can use algorithm LE_2

```

1: while There exist nodes not in a finished cluster do
2:   (0.) each node selects randomly an identity from a big set of integers.
3:   Stage 1: local election in balls of radius  $k$ 
4:   (1.a) Each node  $v$  not in a finished cluster runs algorithm  $LE_k$ 
5:   Stage 2: reinitialization
6:   (2.a) Each formed cluster  $C$  computes independently the sparsity condition for each layer  $j \leq k$ ,
7:   if  $S$  contains a layer  $j$  violating the sparsity condition then
8:     (2.b)  $C$  releases all layers  $l \geq j$  and becomes a finished cluster,
9:     (2.c) nodes in released layers become singleton clusters.
10:  else
11:    if all neighbors are finished then
12:      (2.d)  $C$  becomes finished.
13:    end if
14:  end if
15:  (2.e) Break all non finished clusters and form new singleton clusters.
16: end while

```

Fig. 8 Algorithm ELECT_PART

to first compute some disjoint stars. Since the elected stars are disjoint, the elected nodes can verify the sparsity of their corresponding stars in parallel without interfering with each other. Thus, whenever a node is elected center of a star, it computes the size of its corresponding star and then it decides to be either a radius 1 finished cluster or a finished singleton cluster. By repeating this process until each node gets clustered, we obtain the basic partition we want to compute.

In [33], the authors studied the number of nodes locally elected by their LE_2 algorithm, and they interpreted that as the degree of parallelism authorized by their algorithm. Thus, by applying the LE_2 algorithm for constructing the basic partition, we can study the number of clusters constructed in parallel in one round and for $k = 2$. Using that study, we can derive new upper bounds on the time needed to cluster all the nodes.

In the following we will argue that the local election algorithm of [33] can be extended to elect nodes which are centers of disjoint balls of radius $k \geq 2$. Our LE_k algorithm is then used as a sub-procedure in algorithm ELECT_PART in order to construct the basic partition. These algorithms are described in next paragraphs.

6.2 Algorithm ELECT_PART

Algorithm ELECT_PART is depicted in Fig. 8 below. It runs in many phases until each node of the graph becomes part of a finished cluster. A phase of the algorithm is executed in two stages.

In the first stage, we construct disjoint balls of radius at most k using algorithm LE_k depicted in Fig. 9. Algorithm LE_k can be viewed as a variant of algorithm DIST_PART. It works in at most k rounds. At each round, a node applies the exploration rule and tries to add a new layer. If the exploration is not successful, then

Fig. 9 Algorithm LE_k : code for a cluster

```

1:  $Round \leftarrow 0$ ;
2: while  $Round < k$  do
3:   execute the Exploration Rule;
4:    $Round \leftarrow Round + 1$ ;
5:   if Non Success of the Exploration Rule then
6:     execute the Battle Rule;
7:   end if
8: end while

```

the node executes the battle rule as in algorithm DIST_PART. Note however that whenever the exploration is successful then the new explored layer is added even if it does not satisfy the sparsity condition. At the end of algorithm LE_k , some nodes will succeed growing a cluster up to some distance $t \leq k$. Nevertheless, some layers of those clusters may not verify the sparsity condition.

The second stage allows us to compute finished clusters and to re-initialize the computations for a new phase. In fact, each cluster in the input of the second phase computes independently whether there is a layer that does not satisfy the sparsity condition (Step 2.a). This can be done distributively using convergecast and broadcast between the root and the leaves. If there exists a layer j violating the sparsity condition then the cluster rejects all layers $l \geq j$ and declares itself finished (Steps 2.b and 2.c). Otherwise, if all its neighbors are finished then the cluster declares itself finished (Step 2.d). This is because the cluster will not be able to grow any more. Finally, the remaining clusters are broken into singleton clusters in order to run a new phase (Step 2.e).

Remark 13 Algorithm LE_k grows balls of radius k whereas a radius $k - 1$ suffices. This allows us to mark edges connecting a cluster with the nodes in the last rejected layer and we avoid the *preferred edge election step* needed for some applications. This step is discussed in Sect. 7.

6.3 Analysis of the Algorithm

In this section, we compute a bound of the expected number of phases needed before algorithm ELECT_PART terminates. The main idea of our analysis is to bound the number of nodes becoming part of a finished cluster in a phase, by using the number of clusters constructed in parallel in each phase.

In the sequel, we say that a node is *locally k -elected* if it succeeds the first stage of algorithm ELECT_PART without losing against any other cluster, i.e., line 6 of algorithm LE_k is never executed by a locally k -elected node. We also use a parameter K such that: $\forall v \in V, \mathcal{N}_{2k}(v) \leq K$, where $\mathcal{N}_{2k}(v) = \{u \in V \mid d(u, v) \leq 2k\}$, i.e., K is an upper bound of the $2k$ -neighborhood of any node.

It is not difficult to show that the probability that a node v is locally k -elected in a given phase is $\Omega(1/\mathcal{N}_{2k}(v))$. On the other hand, if we denote by X the random variable which counts the number of locally k -elected nodes, then X can be written as the sum of n random variables X_v (for each $v \in V$) such that $X_v = 1$ with probability

$q = \Omega(1/\mathcal{N}_{2k}(v))$ and 0 with probability $1 - q$. Hence, by the linearity of the expectation, we obtain $\mathbb{E}(X) = \sum_{v \in V - V_f} q$. Thus, the following lemma is straightforward:

Lemma 4 *The expected number of nodes locally k -elected in a phase is lower bounded by $\Omega(\frac{|V - V_f|}{K})$.*

Theorem 7 *Let T be the time complexity of algorithm ELECT_PART. The expected value of T satisfies:*

$$\mathbb{E}(T) = O\left(k^2 \frac{\log(n)}{\log(\frac{K}{K-1})}\right)$$

Proof Let $i \geq 0$ be a phase of the algorithm and $(G_i)_{i \geq 0}$ the sequence of graphs such that $G_0 = G$ and for all $i \geq 1$, G_i is the graph obtained by removing the nodes (and the corresponding incident edges) belonging to a finished cluster from G_{i-1} . Obviously, G_i is the input graph of phase i .

Let X_i be the random variable which denotes the size of the graph G_i (the number of its nodes) for all $i \geq 0$, and let Y_i be the number of nodes locally k -elected in the i th phase. It is clear from Lemma 4 that we have the following inequality:

$$\mathbb{E}(Y_i | G_i) \geq X(G_i)/K$$

It is also easy to see that $X_{i+1} \leq X_i - Y_i$ for all $i \geq 0$. Thus,

$$\mathbb{E}(X_{i+1} | G_i) \leq X_i - \mathbb{E}(Y_i | G_i) \leq X_i \cdot \left(1 - \frac{1}{K}\right)$$

For $i \geq 0$, we define a new r.v. Z_i by $Z_i = X_i / (1 - \frac{1}{K})^i$. Then, $\mathbb{E}(Z_{i+1} | G_i) \leq Z_i$. Thus, the r.v. Z_i is a super-martingale (see [43]), and then

$$\mathbb{E}(Z_{i+1}) = \mathbb{E}(\mathbb{E}(Z_{i+1} | G_i)) \leq \mathbb{E}(Z_i)$$

A direct application of a theorem from [43, Chap. 9], yields $\mathbb{E}(Z_i) \leq Z_0 = n$. Thus

$$\mathbb{E}(X_i) = \left(1 - \frac{1}{K}\right)^i \mathbb{E}(Z_i) \leq n \left(1 - \frac{1}{K}\right)^i.$$

The algorithm terminates when $V_f = V$, i.e., $X_i = 1$. This implies that i is upper bounded by the ratio $\log(n) / \log(\frac{K}{K-1})$. Since both the first and the second stage of the algorithm take at most $O(k^2)$ time to be finished, the assertion in the theorem is proved. \square

Remark 14 The bound given by Theorem 7 does not take into account the size of the finished clusters at each phase but only the number of clusters constructed in parallel. Furthermore, the number of clusters constructed in parallel is just lower bounded using the variable K which corresponds to the initial graph G and not to the subgraph in the input of each phase. It would be very interesting to take all this features into account in order to get a better bound on the number of phases needed to terminate algorithm ELECT_PART.

6.4 Improvements

In algorithm `ELECT_PART`, sparse nodes also participate in the computations and compete against other nodes in order to grow a ball. This slows down the construction because an elected sparse node will always form a finished singleton cluster. Thus, we can improve algorithm `ELECT_PART` by allowing only dense nodes to compete in order to grow a ball of radius k .

Similarly to algorithm `FAST_PART`, we let a dense node win against a sparse one using a couple $(Id, Dense)$. In other words, we prohibit that a sparse node stops the growth of a dense cluster. We also let a sparse node declare itself finished if it is not invaded by any neighboring cluster, i.e., if dense neighbors lose their explorations or if all neighbors are sparse.

By considering the number of *dense nodes* at each phase and using the same arguments than in Theorem 7, we can find a bound on the expected number of phases needed to terminate the construction. Unfortunately, the theoretical analysis leads to the same bound than in Theorem 7. It is also easy (using the same reasoning than in Theorem 6) to prove that the complexity of the modified algorithm is bounded by $O(k^2\Lambda)$.

This new modified version of algorithm `ELECT_PART` is particularly interesting because it has a sublinear time complexity for general graphs, and at the same time, it allows us to express the high degree of parallelism of our method. For instance, consider a graph G such that $K = O(n^\epsilon)$ with $\epsilon < 1$. This defines a large class of graphs for which we can achieve an improved time complexity, namely $O(\log(n)n^\epsilon)$.

In Appendix B, we show that the expected running time of the modified algorithm is $O(\log(n))$ in the case of *Circulant graphs*.

7 Application to Graph Spanners

In this section, we show how to efficiently construct graph spanner in the *CONGEST* distributed model using our previous algorithms. A subgraph H is an (α, β) -spanner of a graph G if H is a spanning subgraph of G and $d_H(u, v) \leq \alpha \cdot d_G(u, v) + \beta$ for all nodes u, v of G , where $d_X(u, v)$ denotes the distance from u to v in the graph X . The couple (α, β) is called the *stretch* of H , and the *size* of H is the number of its edges.

One immediate application of algorithm `BASIC_PART` is the construction of a $(4k - 3, 0)$ -spanner with $O(n^{1+1/k})$ edges for any n -node graph G . The spanner is obtained by considering the set of edges spanning each cluster and by selecting an inter-cluster edge for each pair of two neighboring clusters. The bounds on the stretch and the size of the spanner are a straightforward consequence of Theorem 1. In order to construct such a spanner *distributively*, we must first construct the basic partition, and second *select an edge between every two neighboring clusters*. However, we can both avoid this additional step of selecting preferred edges and at the same time improve the bound on the spanner size.

In fact, let us consider any cluster C under construction in algorithm `DIST_PART`. Before the construction of C is finished (just after the sparsity condition is no longer

satisfied) and for every neighboring vertex u of C (u is on the last rejected layer of C), we select an edge from u to some v in the last layer of C and we add it to the spanner S . Moreover, we add the BFS spanning tree of each cluster C to the spanner S . It is well known that this idea allows us to construct a $(2k - 1, 0)$ -spanner with $O(n^{1/k})$ edges. This idea is in fact attributed to [27] in [36, Exercise 3, p. 188] and is used in [22] as a first step to construct $(1 + \epsilon, \beta)$ -spanners. The same idea is also used in [34] to improve the complexity of synchronizers γ_1 and γ_2 . The time complexity of the algorithms used in [34, 36] is $O(n)$ and it has not been improved since.

We remark that the last rejected layer is always explored in all the distributed algorithms described in previous sections. Hence, the edges connecting a cluster with nodes in the last rejected layer are implicitly computed by our algorithms without any extra communications. Hence, by the previous discussion, the following results are straightforward:

Theorem 8 *There is a deterministic distributed algorithm that given a graph with n nodes and a fixed integer $k \geq 1$, constructs a $(2k - 1, 0)$ -spanner with $O(n^{1+1/k})$ edges in $O(n^{1-1/k})$ time using messages of length $O(\log n)$.*

Corollary 1 *There is a deterministic algorithm that given a graph with n nodes constructs a $(3, 0)$ -spanner with $O(n^{3/2})$ edges in $O(\sqrt{n})$ time using messages of length $O(\log n)$.*

One can find many papers concerning the construction of graph spanners. In [38, 39, 44], the reader can find excellent and recent reviews on graph spanners.

To our knowledge Theorem 8 provides the best time complexity for constructing $(2k - 1, 0)$ -spanners with $O(n^{1+1/k})$ edges in a *deterministic* manner and *using small messages*. The fastest deterministic algorithm was given very recently in [20]. It constructs $(2k - 1, 0)$ -spanners with $O(kn^{1+1/k})$ edges in $O(k)$ time using messages of polynomial size. Fast *randomized* algorithms using small messages exist. The best one is due to Baswana et al. [11, 12]. The authors there gave a (*Las-Vegas*) randomized algorithm that computes a $(2k - 1, 0)$ -spanner with *expected size* $O(kn^{1+1/k})$ in $O(k)$ time using $O(\log(n))$ size messages. As mentioned in [9], a randomized solution might not be acceptable in some cases, especially for distributed computing applications. In the case of graph spanners, deterministic algorithms that *guarantee* a high quality spanner are more than of a theoretical interest. Indeed, one cannot just run a randomized distributed algorithm several times to guarantee a good spanner, since it is impossible to check efficiently the global quality of the spanner in the distributed model.

8 Open Questions

In this paper, we focus on the time complexity of constructing sparse partitions in the practical *CONGEST* distributed model. One important motivation of our work is to understand and to study the effects of the congestion created by small messages into the overall time complexity of a distributed algorithm.

We left open the following questions:

1. Can we improve the time complexity of our algorithms from $n^{1-1/k}$ to $n^{1/k}$ in the *CONGEST* model? In particular, we remark that, in the case of small k (2, 3, 4, 5), the locality level of the basic partition and the time complexity bound obtained using our technique are better than the bounds one can obtain by both assuming a more powerful distributed model, i.e., unlimited message size, and using techniques from [9]. This observation is intriguing and one can be interested in a lower bound on the time complexity of distributively computing the basic partition. Although, the case $k = 2$ seems hard to improve, we are optimistic that deterministic algorithms with better bounds exist for other values of k .
2. We have studied the sparse partition problem from a *locality* point of view. In other words, we only consider the problem of improving the time complexity. Can we improve the message complexity of our algorithms while maintaining the same time complexity?

Appendix A: Application to Neighborhood Covers and Network Synchronizers

Neighborhood covers can be thought as a generalization of the basic partition. In fact, for any positive integer ρ , a ρ -neighborhood cover of a graph G can be defined as a collection of clusters $\mathcal{C} = \cup C$ such that for every $v \in V$, there exists a cluster $C \in \mathcal{C}$ such that $\mathcal{N}_\rho(v) \subseteq C$ where $\mathcal{N}_\rho(v) = \{u \in V \mid d(u, v) \leq \rho\}$ denotes the ρ -neighborhood of node v in the graph G . Hence, the basic partition described before is a 0-neighborhood cover of G having a low average degree, namely $O(n^{1/k})$. In Peleg's book [36, Chap. 12], one can find a survey on different types of covers obtained on the basis on the basic partition algorithm. In particular, given an initial cover \mathcal{S} , it is shown in [36] how to extend algorithm BASIC_PART to construct a *coarsening* cover \mathcal{T} of \mathcal{S} , that is a cover that subsumed \mathcal{S} . By taking $\mathcal{S} = \bigcup_{v \in V} \mathcal{N}_\rho(v)$, it can be shown that a ρ -neighborhood cover with radius $O(k \cdot \rho)$ and average degree $O(n^{1/k})$ can be constructed on the basis of algorithm BASIC_PART (the proof is by Theorem 12.2.1 of [36]). This type of neighborhood covers is also used in [34] as an auxiliary communication structure to design network synchronizers. More specifically, the authors in [34] described a distributed 1-neighborhood cover algorithm that is used to design a new efficient synchronizer. Based on the extended construction of [36], it is not difficult to extend our basic partition distributed algorithms to construct ρ -neighborhood covers distributively. Since we are mainly interested in the application of covers in the design of network synchronizers, we will briefly outline the modifications to be done to obtain the ρ -neighborhood cover used in [34]. Extending our technique for any ρ is left as an exercise.

A.1 Distributed Construction of 1-Neighborhood Covers

In this section, we extend algorithm DIST_PART in order to cover the 1-neighborhood of each node. We use the same distributed techniques to manage cluster growth. However, we make a cluster explore two layers at the same time instead of only one. At each new exploration, each cluster fights to maintain two layers l_i and l_{i+1} with i the radius of the cluster. The first layer l_i allows the cluster to compute the sparsity condition (the same one than in algorithm DIST_PART). The second layer l_{i+1} (which is

the last explored layer) guarantees that the neighborhoods of all nodes in layer l_i are in the current cluster. There are mainly five important modifications to do:

1. At the beginning of the algorithm, all nodes are orphans. An orphan node first explores *two consecutive* layers before starting computing the sparsity condition.
2. If the sparsity condition is satisfied for layer l_i , then a cluster begins a new exploration, i.e., the leaves in layer l_{i+1} try to invade new nodes. If the new exploration succeeds, then layer l_{i+1} becomes layer l'_{i+1} and the new explored layer becomes the new l_{i+1} layer.
3. If the sparsity condition for layer l_i is not satisfied, then the construction of the cluster is finished. The finished cluster contains not only all layers $l_{j < i}$ but also the two layers l_i and l_{i+1} . Nevertheless, only nodes in layers $l_{j < i}$ are in a final state. The 1-neighborhoods of all nodes in layer l_i are covered by the finished cluster but they do not stop computing yet. In fact, the 1-neighborhoods of nodes in layer l_{i+1} may not be covered by a cluster. Hence, nodes in layer l_i become orphan clusters with identity $-\infty$ in order to allow other clusters to grow and cover the neighborhoods of nodes in layer l_{i+1} . On the other side, nodes in layer l_{i+1} become orphan clusters with their initial identifiers and continue competing in order to grow new clusters.
4. If a new exploration fails, i.e., there is a cluster at distance 1 or 2 (from layer l_{i+1}) with a bigger identifier, then:
 - either the winner lost against another neighboring cluster and the current cluster is not invaded. Hence, the cluster simply retries a new exploration.
 - or the current cluster is invaded and the cluster loses its last layer l_{i+1} . Hence, invaded nodes in layer l_{i+1} become part of the last layer $l_{i_{win}+1}$ of the winner cluster. Nodes in layer l_{i+1} which have not been invaded become orphan nodes and begin a new exploration using their own identifiers. Layer l_i becomes the last layer $l'_{i+1=i}$ and layer l_{i-1} becomes layer l'_{i-1} . Then the cluster begins a new exploration once again.
5. When the construction of a cluster is finished, nodes at distance at least 2 from the border of the cluster, i.e., layers $l_{j \leq i-1}$, switch to final states. In fact, layer l_{i-1} of a finished cluster acts as a barrier that protects the finished cluster from future invasions. Layers $l_{j \leq i-1}$ are usually called the *Kernel* of the cluster.

It is easy to see that the time and message complexity of the extended algorithm increases by only a constant factor due to the computation of the extra layer l_{i+1} . Notice also that it is not difficult to adapt the techniques of algorithms FAST_PART and ELECT_PART in order to obtain sublinear time complexity.

An Example of Cluster Growth In Fig. 10, we give an example of how the cover is constructed. In our example, there are four active clusters: 1, 2, 3 and 4 with identities $Id_1 > Id_2 > Id_3 > Id_4$. We suppose that there is a finished cluster in the neighborhood of cluster 1.

The nodes in layer l_i of the finished cluster (first part of Fig. 10) still participate in the computation with identity $-\infty$, all the nodes in the Kernel of the finished cluster are in a final state. There is also a node in layer l_{i+1} of the finished cluster which belongs to layer l_{i+1} of cluster 1.

Suppose that the layers l_i of the active clusters satisfy the sparsity condition, then these clusters will try to grow. Cluster 2 cannot grow because cluster 1 is at distance two of it and has a bigger identity. Cluster 1 will invade both clusters 3 and 4. Cluster 4 is orphan and it simply joins the last layer of cluster 1. Cluster 3 will lose its last layer l_{i+1} . The invaded nodes of cluster 3 join cluster 1 and the other nodes which have not been invaded become orphan clusters (second part of Fig. 10). Note also that the node with identity $-\infty$ in the finished cluster is invaded by cluster 1. This guarantees that the neighborhood of the children of the $(-\infty)$ -node in the finished cluster is covered by cluster 1.

Once the new exploration is finished, cluster 1 verifies the sparsity condition. If it is satisfied, a new exploration will begin and clusters 2 and 3 will be invaded. Note that nodes in the Kernel of the finished cluster will not be invaded by cluster 1. If the sparsity condition is not satisfied which is the case in the third part of Fig. 10, the construction of cluster 1 is finished. The nodes in layer $l_{i'}$ become orphans with identity $-\infty$. The nodes in layer $l_{i'+1}$ become orphan nodes except those which are already in layer l_i of another finished cluster (those whose neighborhoods are covered). Note that the two finished clusters we have constructed overlap (they have a common edge).

A.2 Application to Network Synchronizers

The basic partition and the 1-neighborhood covers constructed in previous sections are of special interest for designing network synchronizers γ , γ_1 and γ_2 [34]. In the following, we review the basic properties of these synchronizers.

Background Network synchronizers allow us to transform a synchronous algorithm into an asynchronous one. In general, one prefers to design a distributed algorithm in a synchronous model rather than an asynchronous model which is typically harder to grasp and to analyze [36, Chap. 6]. From a practical point of view, network synchronizers provide a uniform methodology for transforming synchronous distributed algorithms into asynchronous ones.

Generally speaking, the basic idea of network synchronizers is to simulate a global clock by using *local pulse generators*. If the local clock pulse of some node v is equal to p , then node v knows that the messages that it has sent at pulse $p - 1$ have reached their destinations. Many simulation techniques were developed in order to guarantee this property:

1. The first basic technique is known as synchronizer α . The general idea of synchronizer α is to send an acknowledgment corresponding to each received message of the original synchronous algorithm. Once, a node receives an acknowledgment of all the original messages corresponding to one pulse, the node informs its neighbors and then it generates the next pulse. This technique leads to an overhead of $O(|E|)$ messages in order to simulate a pulse. Assuming that a message delay is at most $O(1)$ (this is only for performance analysis), it also leads to the theoretical $O(1)$ time overhead per pulse.
2. The second basic technique called synchronizer β assumes a precomputed rooted BFS spanning tree T of G . Only the root of T have a pulse generator which

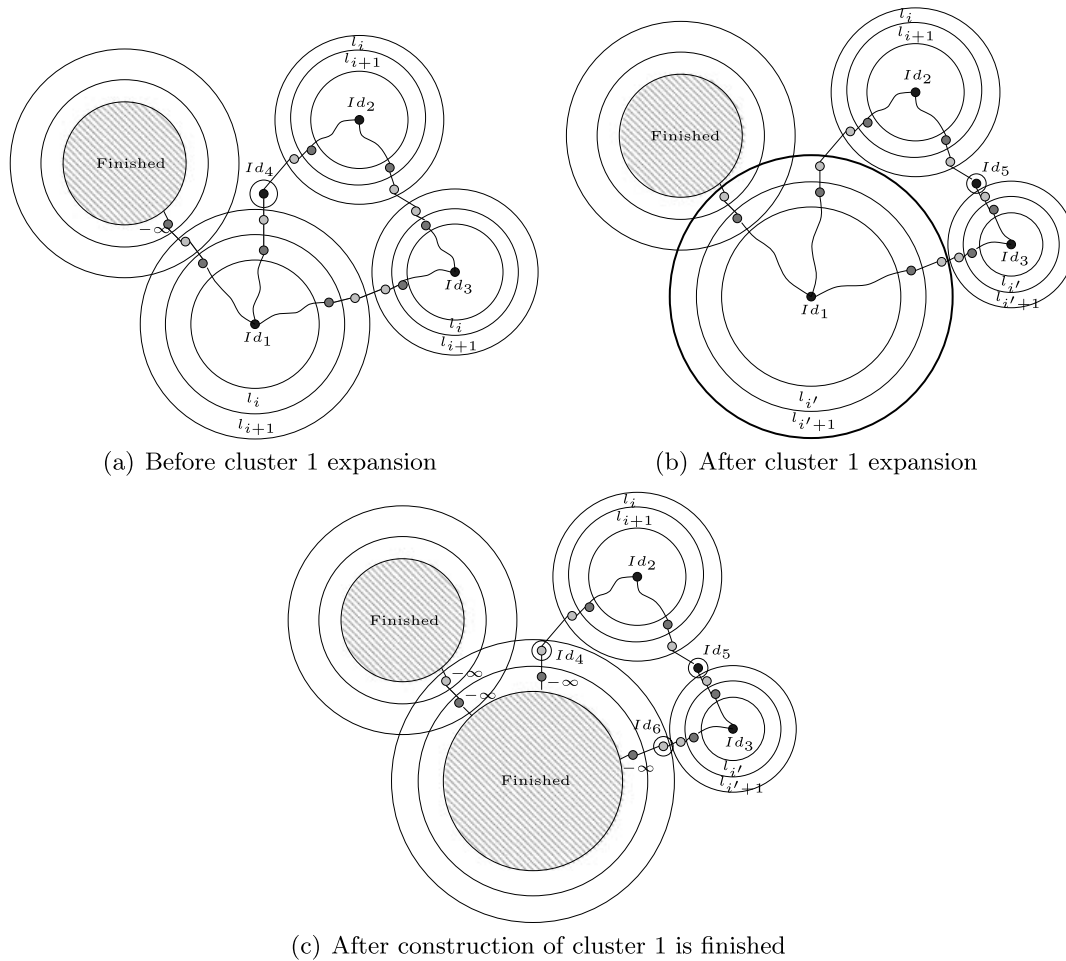


Fig. 10 An example of a cluster expansion for cover needed for γ_2

controls all other nodes. In fact, once a node u receives the acknowledgments of the messages it has sent, the node u is ready for the next pulse and it informs its parent in the tree T . The parents forward this information until it reaches the root of T . Once the root learns that all the nodes are ready for the next pulse, it broadcasts a message saying “*it is time for the next pulse*”! Thus, synchronizer β implies an overhead of $O(|V|)$ messages and $O(D)$ time per pulse, where D is the diameter of G .

3. The third technique is an intermediate technique which provides a good time-message trade-offs. This technique implies three synchronizers γ , γ_1 and γ_2 [34]. All of these three synchronizers use sparse covers. More precisely, synchronizer γ uses the basic partition as an auxiliary communication structure. Synchronizer γ_1 uses a cover based on the basic partition where each edge belongs to at least one cluster. This property is easily obtained by our partition algorithms by simply marking the last rejected layer as part of the cluster. Finally, synchronizer γ_2 uses the 1-neighborhood cover described in the previous section.

A detailed description of synchronizers γ , γ_1 and γ_2 can be found in [34]. In the following, we just outline the basic ideas used in synchronizer γ (the two other synchronizers are based on the same general ideas). First, we assume the following:

1. A partition \mathcal{C} of G is constructed.
2. A rooted BFS spanning tree T_C for each cluster $C \in \mathcal{C}$ is constructed.
3. A set \mathcal{I} of intercluster edges is selected.

To simulate a pulse, we combine the techniques of synchronizers α and β . Roughly speaking, the root of each tree T_C first waits to learn that the nodes in C are ready for the next pulse (which costs $O(|C|)$ messages and $O(\text{Rad}(C))$ time for each cluster). Then, the cluster tries to synchronize with its neighbors using the intercluster edges. More precisely, the root of C broadcasts a notification message all along the tree T_C saying that all the nodes in its cluster are ready. When the leaves of T_C receive the notification message, they forward it to neighboring clusters using the selected intercluster edges (which costs $O(|\mathcal{I}|)$ message and $O(1)$ time). Symmetrically, the leaves receive a notification from their neighboring clusters. When receiving such a notification, they send it back to their root. Once the root receives the notification messages of its neighbors, it sends a message to the nodes in its cluster saying “it is time for the next pulse”. Thus, the global overhead is $O(n + |\mathcal{I}|)$ messages and $O(\text{Rad}(C))$ time per pulse.

Thus, if we take the basic partition as a communication structure, then the global overhead is $O(n^{1+1/k})$ messages and $O(k)$ time per pulse which gives a good compromise compared to synchronizer α and β .

Contribution The previous overhead is essentially optimal according to Lemma 25.1.7 in Peleg’s book [36]. Synchronizers γ , γ_1 and γ_2 have the same performances up to a constant factor. Hence, one remaining challenge is to improve the pre-processing step of constructing the required covers. Using our algorithms, the time complexity of this pre-processing step is reduced from $O(n)$ in previous implementations to $O(n^{1-1/k})$.

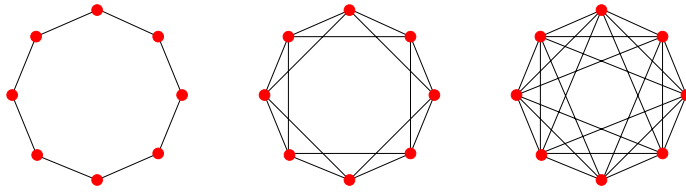
Appendix B: Case Study: Circulant Graphs

In this section, we study the efficiency of algorithms FAST_PART and algorithm ELECT_PART in the case of *Circulant Graphs*. In fact, Circulant Graphs are *dense* enough to be interesting for the algorithm we are studying. They have enough large diameter in order to let the analysis non trivial and constructive. In addition, the analysis given here is interesting from a theoretical point of view. In particular, the proof of Theorem 12 below illustrate the improvements discussed in Sect. 6.4. The reader should also note that this class of graphs was studied in many past works and for different purposes. For instance, it is used in [16] as a basis for the construction of graphs having the small-world property.

Definition 1 A circulant graph $Cir_n(\mathcal{L})$ is a graph of n nodes $\{1, 2, \dots, n\}$ in which a vertex i is adjacent to nodes $(i - j)$ and $(i + j)$ for each i and j in the list \mathcal{L} (see Fig. 11 for an example).

Definition 2 For every parameter ϵ such that $0 < \epsilon \leq 1$, we define the graph Cir_n^ϵ to be the circulant graph $Cir_n(1, 2, \dots, \lfloor \frac{n^\epsilon}{2} \rfloor)$.

Fig. 11 An Example of $Cir_n(\mathcal{L})$ graphs with $n=8$ and $\mathcal{L} \in \{\{1\}, \{1, 2\}, \{1, 2, 3\}\}$



In the sequel, we suppose that $\epsilon > \frac{1}{k}$. In fact, if $\epsilon \leq \frac{1}{k}$ then the graph is already sparse and all our algorithms terminate in $O(1)$ rounds.

Theorem 9 For $k < \log(n)$ and for every graph Cir_n^ϵ , the time complexity of algorithm FAST_PART is bounded by $O(n^{1-\epsilon})$.

Proof For $k < \log(n)$, any constructed cluster has radius at most 1. It can also be shown that $\Lambda \leq 2 \frac{n}{n^\epsilon} = O(n^{1-\epsilon})$. Thus, the theorem follows as a consequence of Theorem 6. \square

Theorem 10 Let T be the time complexity of algorithm ELECT_PART. Then, for every graph Cir_n^ϵ , the expected value of T satisfies:

$$\mathbb{E}(T) = O(k^3 \log(n) n^\epsilon)$$

Proof For any graph Cir_n^ϵ , it is easy to show that $K = 2k n^\epsilon$ (we recall that K is an upper bound of the $2k$ -neighborhood of any node). Thus, $\log(1 - \frac{1}{K}) \leq -\frac{1}{K} = -\frac{1}{2kn^\epsilon}$ and the result follows immediately from Theorem 7. \square

The two Theorems 9 and 10 are immediate consequences of the analysis we have already made for algorithms SYNC_PART and ELECT_PART. In particular, we obtain a time complexity which is better than $O(n^{1-\frac{1}{k}})$. Nevertheless, using a more careful analysis, we obtain the following bounds:

Theorem 11 For every graph Cir_n^ϵ , the expected time complexity T of algorithm ELECT_PART satisfies:

$$\mathbb{E}(T) = O(k^3 \log(n) + kn^{\frac{1}{k}})$$

Theorem 12 Using the improved version of algorithm ELECT_PART described in Sect. 6.4, the expected time complexity T of algorithm ELECT_PART satisfies:

$$\mathbb{E}(T) = O(k^3 \log(n))$$

Proof We prove the previous two theorems in two parts. The first part is common to the two theorems. The technical arguments are similar to those in the analysis made in Theorem 7 but the reasoning is different.

First Part of the Proof Let $i \geq 0$ be a phase of algorithm ELECT_PART and $(G_i)_{i \geq 0}$ be the sequence of graphs such that $G_0 = G$ and for all $i \geq 1$, G_i is the graph obtained by removing the nodes belonging to a finished cluster from G_{i-1} .

Let V_i be the set of nodes having a degree higher than $\lfloor \frac{n^\epsilon}{2} \rfloor$ in phase i . Let X_i be the random variable which denotes the number of nodes in V_i , and let Y_i be the number of nodes from V_i which are locally k -elected in the i th step. The following inequality holds:

$$\mathbb{E}(Y_i | G_i) \geq \frac{X_i}{K}.$$

One can show that if the node v belongs to V_i , then every active neighbor w of v is also in V_i . Hence, we can state the following:

$$\begin{aligned} \mathbb{E}(X_{i+1} | G_i) &\leq X_i - \mathbb{E}(Y_i | G_i) \frac{n^\epsilon}{2} \\ &\leq X_i \left(1 - \frac{n^\epsilon}{2K} \right) \\ &\leq X_i \left(1 - \frac{1}{2 \cdot 2k} \right). \end{aligned}$$

By induction and using the same arguments as in Theorem 7, the expected time such that $X_i = 1$ is bounded by:

$$O\left(k^2 \frac{\log(n)}{\log\left(\frac{4k}{4k-1}\right)}\right).$$

Let us consider the time after which all nodes in the graph have a degree less than $\lfloor \frac{n^\epsilon}{2} \rfloor$ (i.e., the time such that $V_i = 0$). One can show that the remaining nodes are grouped in many connected fragments that can be divided in two types: dense components with more than $n^{\frac{1}{k}}$ nodes and sparse components with no more than $n^{\frac{1}{k}}$ nodes. All these components are disjoint and do not share any node. Thus, the algorithm runs independently on each component.

Let us consider a dense component C_d (i.e., $n^{\frac{1}{k}} < |C_d| < \lfloor \frac{n^\epsilon}{2} \rfloor$). In one phase, there will be exactly one elected node in C_d and the finished cluster constructed around this node will contain the whole component C_d . Thus, in $O(k)$ time, all nodes in C_d become finished.

Second Part of the Proof of Theorem 11 Let us consider a sparse component C_s (i.e., $|C_s| \leq n^{\frac{1}{k}}$). The nodes of such a component have a degree less than $n^{\frac{1}{k}}$. At each phase of algorithm ELECT_PART, there will be exactly one elected node in C_s which forms a finished singleton cluster. Thus, we need at most $O(n^{\frac{1}{k}})$ phases of $O(k)$ time units each before all nodes in C_s become finished.

To conclude, if V_i becomes empty then we need at most $O(kn^{\frac{1}{k}})$ time units before the algorithm terminates and Theorem 11 holds.

Second Part of the Proof of Theorem 12 Let us consider a sparse component C_s , i.e., $|C_s| \leq n^{\frac{1}{k}}$. The nodes of such a component have a degree less than $n^{\frac{1}{k}}$. Thus, using from the improvements of algorithm ELECT_PART in Sect. 6.4, these nodes are allowed to be finished. Hence, in $O(1)$ time, they all become finished singleton clusters.

To conclude, if V_i becomes empty then we need at most $O(k)$ time units before the algorithm terminates and Theorem 12 holds. \square

References

1. Afek, Y., Ricklin, M.: Sparsers: a paradigm for running distributed algorithms. *J. Algorithms* **14**, 316–328 (1993)
2. Althofer, I., Das, G., Dobkin, D., Joseph, D., Soares, J.: On sparse spanners of weighted graphs. *Discrete Comput. Geom.* **9**, 81–100 (1993)
3. Awerbuch, B.: Complexity of network synchronization. *J. ACM* **32**, 804–823 (1985)
4. Awerbuch, B., Goldberg, A.V., Luby, M., Poltkin, S.A.: Network decomposition and locality in distributed computation. In: 30th IEEE Symposium on Foundation of Computer Science (FOCS89), pp. 364–369 (1989)
5. Awerbuch, B., Peleg, D.: Sparse partitions. In 31th Symposium on Foundations of Computer Science (FOCS90), pp. 503–513. IEEE Comput. Soc., Los Alamitos (1990)
6. Awerbuch, B., Peleg, D.: Network synchronization with polylogarithmic overhead. In: 31st IEEE Symposium on Foundations of Computer Science (FOCS90), vol. 2, pp. 503–513 (1990)
7. Awerbuch, B., Peleg, D.: Routing with polynomial communication-space trade-off. *SIAM J. Discrete Math.* **5**, 151–162 (1992)
8. Awerbuch, B., Peleg, D.: Online tracking of mobile users. *J. ACM* **42**, 1021–1058 (1995)
9. Awerbuch, B., Berger, B., Cowen, L.J., Peleg, D.: Fast distributed network decompositions and covers. *J. Parallel Distrib. Comput.* **39**, 105–114 (1996)
10. Awerbuch, B., Berger, B., Cowen, L.J., Peleg, D.: Near-linear time construction of sparse neighborhood covers. *SIAM J. Comput.* **28**(1), 263–277 (1998)
11. Baswana, S., Sen, S.: A simple linear time algorithm for computing a $(2k - 1)$ -spanner of $O(n^{1+1/k})$ size in weighted graphs. In: 30th International Colloquium on Automata, Languages and Programming (ICALP03). Lecture Notes in Computer Science, vol. 2719, pp. 384–396. Springer, Berlin (2003)
12. Baswana, S., Kavitha, T., Mehlhorn, K., Pettie, S.: New constructions of (α, β) -spanners and purely additive spanners. In: 16th Symposium on Discrete Algorithms (SODA05), pp. 672–681. ACM, New York (2005)
13. Belkouch, F., Bui, M., Chen, L., Datta, A.K.: Self-stabilizing deterministic network decomposition. *J. Parallel Distrib. Comput.* **62**, 696–714 (2002)
14. Chalopin, J., Métivier, Y.: A bridge between the asynchronous message passing model and local computations in graphs. In: Mathematical Foundations of Computer Science (MFCS05). Lecture Notes in Computer Science, vol. 3618, pp. 212–223. Springer, Berlin (2005)
15. Cohen, E.: Fast algorithms for constructing t -spanners and paths with stretch t . *SIAM J. Comput.* **28**(1), 210–236 (1998)
16. Comellas, F., Ozón, J., Peters, J.G.: Deterministic small-world communication networks. *Inf. Process. Lett.* **76**(1–2), 83–90 (2000)
17. Cowen, L.J.: On local representations of graphs and networks. Ph.D. Thesis, MIT (1993)
18. Derbel, B., Mosbah, M.: A fully distributed linear time algorithm for cluster network decomposition. In: 16th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS04), pp. 548–553 (2004)
19. Derbel, B., Mosbah, M., Zemmari, A.: Fast distributed graph partition and application. In: 20th IEEE International Parallel & Distributed Processing Symposium (IPDPS06). IEEE Comput. Soc., Los Alamitos (2006)
20. Derbel, B., Gavaille, C., Peleg, D., Viennot, L.: On the locality of distributed sparse spanner construction. In: 27th Symposium on Principle of Distributed Computing (PODC), pp. 273–282 (2008)

21. Elkin, M.: Computing almost shortest paths. In: 20th ACM Symposium on Principles of Distributed Computing (PODC01), pp. 53–62. ACM, New York (2001)
22. Elkin, M., Peleg, D.: $(1 + \epsilon, \beta)$ -spanner constructions for general graphs. *SIAM J. Comput.* **33**(3), 608–631 (2004)
23. Elkin, M., Zhang, J.: Efficient algorithms for constructing $(1 + \epsilon, \beta)$ -spanners in the distributed and streaming models. In: 23rd ACM Symposium on Principles of Distributed Computing (PODC04), pp. 160–168. ACM, New York (2004)
24. Gaber, I., Mansour, Y.: Centralized broadcast in multihop radio networks. *J. Algorithms* **46**, 1–20 (2003)
25. Garay, J.A., Kutten, S., Peleg, D.: A sublinear time distributed algorithm for minimum-weight spanning trees. *SIAM J. Comput.* **27**, 302–316 (1998)
26. Godard, E., Métivier, Y., Muscholl, A.: Characterizations of classes of graphs recognizable by local computations. *Theory Comput. Syst.* **37**(2), 249–293 (2004)
27. Halperin, S., Zwick, U.: Unpublished result (1996)
28. Kutten, S., Peleg, D.: Fast distributed construction of small k -dominating sets and applications. *J. Algorithms* **28**(1), 40–66 (1998)
29. Linial, N.: Distributive graph algorithms—global solutions from local data. In: 28th IEEE Symposium on Foundations of Computer Science (FOCS87), pp. 331–335. IEEE Comput. Soc., Los Alamitos (1987)
30. Linial, N.: Locality in distributed graphs algorithms. *SIAM J. Comput.* **21**(1), 193–201 (1992)
31. Litovsky, I., Métivier, Y., Sopena, E.: Different local controls for graph relabelling systems. *Math. Syst. Theory* **28**, 41–65 (1995)
32. Litovsky, I., Métivier, Y., Sopena, E.: Graph relabelling systems and distributed algorithms. In: *Handbook of Graph Grammars and Computing by Graph Transformation*, vol. 3, pp. 1–56. World Scientific, Singapore (1999)
33. Métivier, Y., Saheb, N., Zemmari, A.: Randomized local elections. *Inf. Process. Lett.* **82**, 313–320 (2002)
34. Moran, S., Snir, S.: Simple and efficient network decomposition and synchronization. *Theor. Comput. Sci.* **243**(1–2), 217–241 (2000)
35. Panconesi, A., Srinivasan, A.: Improved distributed algorithms for coloring and network decomposition. In: 24th ACM Symposium on Theory of Computing (STOC92), pp. 581–592 (1992)
36. Peleg, D.: *Distributed Computing: A Locality-Sensitive Approach*. SIAM Monographs on Discrete Mathematics and Applications (2000)
37. Peleg, D., Rubinfeld, V.: A near-tight lower bound on the time complexity of distributed minimum-weight spanning tree construction. *SIAM J. Comput.* **30**(5), 1427–1442 (2000)
38. Pettie, S.: Low distortion spanners. In: 34th International Colloquium on Automata, Languages and Programming (ICALP), pp. 78–89 (2007)
39. Pettie, S.: Distributed algorithms for ultrasparse spanners and linear size skeletons. In: 27th Symposium on Principle of Distributed Computing (PODC), pp. 253–262 (2008)
40. Roditty, L., Thorup, M., Zwick, U.: Deterministic constructions of approximate distance oracles and spanners. In: 32nd International Colloquium on Automata, Languages and Programming (ICALP), *Lecture Notes in Computer Science* (2005)
41. Shabtay, L., Segall, A.: Low complexity network synchronization. In: 8th International Workshop on Distributed Algorithms, pp. 223–237 (1994)
42. Thorup, M., Zwick, U.: Spanners and emulators with sublinear distance errors. In: 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 802–809 (2006)
43. Williams, D.: *Probability with Martingals*. Cambridge University Press, Cambridge (1993)
44. Woodruff, D.P.: Lower bounds for additive spanners, emulators, and more. In: 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 389–398 (2006)



Broadcast in the rendezvous model[☆]

Philippe Duchon, Nicolas Hanusse, Nasser Saheb, Akka Zemmari*

LaBRI – CNRS – Université Bordeaux I, 351 Cours de la Liberation, 33405 Talence, France

Received 9 July 2004; revised 21 April 2005
Available online 10 March 2006

Abstract

In many large, distributed or mobile networks, broadcast algorithms are used to update information stored at the nodes. In this paper, we propose a new model of communication based on *rendezvous* and analyze a *multi-hop distributed algorithm* to *broadcast* a message in a *synchronous* setting. In the *rendezvous model*, two neighbors u and v can communicate if and only if u calls v and v calls u simultaneously. Thus, nodes u and v obtain a rendezvous at a meeting point. If m is the number of meeting points, the network can be modeled by a graph of n vertices and m edges. At each round, every vertex chooses a random neighbor and there is a rendezvous if an edge has been chosen by its two extremities. Rendezvous enable an exchange of information between the two entities. We get sharp lower and upper bounds on the time complexity in terms of number of rounds to broadcast: we show that, for any graph, the expected number of rounds is between $\ln n$ and $O(n^2)$. For these two bounds, we prove that there exist some graphs for which the expected number of rounds is either $O(\ln(n))$ or $\Omega(n^2)$. For specific topologies, additional bounds are given.

© 2006 Elsevier Inc. All rights reserved.

Keywords: Algorithms and data structures; Distributed algorithms; Graph; Broadcast; Rendezvous model

[☆] A preliminary version of this paper has appeared in the 21st Symposium on Theoretical Aspects of Computer Science (STACS), Montpellier, France, March 2004.

* Corresponding author.

E-mail addresses: duchon@labri.fr (P. Duchon), hanusse@labri.fr (N. Hanusse), saheb@labri.fr (N. Saheb), zemhari@labri.fr (A. Zemmari).

1. Introduction

Among the numerous algorithms to broadcast in a synchronized setting, we are witnessing a new tendency of distributed and randomized algorithms, also called *gossip-based algorithms*: at each instant, any number of broadcasts can take place simultaneously and we do not give any priority to any particular one. In each round, a node chooses a random neighbor and tries to exchange some information. Due to the simplicity of gossip-based algorithm, such an approach provides reliability and scalability. Contrary to deterministic schemes for which messages tend to route in a particular subgraph (for instance a tree), a gossip-based algorithm can be fault-tolerant (or efficient for a dynamic network) since in a strongly connected network, many paths can be used to transmit a message to almost every node.

The majority of results deal with the *uniform random phone call* for which a node chooses a neighbor uniformly at random. However, such a model does not take into account that a given node could be “called” by many nodes simultaneously implying a potential congestion. A more embarrassing situation is the one of the *radio networks* in which a node should be called simultaneously by a unique neighbor otherwise the received messages are in collision. In the *rendezvous model*, every node chooses a neighbor and if two neighbors choose themselves mutually, they can exchange some information. The *rendezvous model* is useful if a physical meeting is needed to communicate as in the case of robots network.

Although the *rendezvous model* can be used in different settings, we describe the problem of broadcasting a message in a network of robots. A robot is an autonomous entity with a bounded amount of memory having the capacity to perform some tasks and to communicate with other entities by radio when they are geographically close. Examples of use of such robots are numerous: exploration [1,7], navigation (see Survey of [17]), capture of an intruder [3], search for information, help to handicapped people or rescue, cleaning of buildings, ... The literature contains many efficient algorithms for one robot and multiple robots are seen as a way to speed up the algorithms. However, in a network of robots [4], the coordination of multiple robots implies complex algorithms. *Rendezvous* between robots can be used in the following setting: consider a set of robots distributed on a geometric environment. Even if two robots sharing a region of navigation (called neighbors) might communicate, they should also be close enough. It may happen that their own tasks do not give them the opportunity to meet (because their routes are deterministic and never cross) or it may take a long time if they navigate at random. A solution consists in deciding on a meeting point for each pair of neighbor robots. If two neighbors are close to a given meeting point at the same time, they have a *rendezvous* and can communicate.

Although there exist many algorithms to broadcast messages, we only deal with algorithms working under a very weak assumption: each node or robot only knows its neighbors or its own meeting points. This implies that the underlying topology is *unknown*. Depending on the context, we might also be interested in *anonymous* networks in which the labeling of the nodes (or history of the visited nodes) is not used. By anonymous, we mean that unique identities are not available to distinguish nodes (processors) or edges (links). In a robot network, the network can have two (or more) meeting points with the same label if the environment contains two pairs of regions that do not overlap. The anonymous setting can be encountered in dynamic, mobile or heterogeneous networks.

1.1. Related works

How to broadcast efficiently a message with a very poor knowledge on the topology of an *anonymous* network ? Depending on the context, this problem is related to the way a “rumor” or an “epidemic” spreads in a graph. In the literature, a node is *contaminated* if it knows the rumor. The broadcast algorithm highly depends on the communication model. For instance, in the *k-ports model*, a node can send a message to at most k neighbors. Thus, our rendezvous model is a 1-port model.

The performance of a broadcast algorithm is measured by the time required to contaminate all the nodes, the amount of memory stored at each node or the total number of messages. In this article, we analyze the time complexity in a synchronous setting of a rendezvous algorithm (although several broadcast algorithms including ours can work in an asynchronous setting, the theoretical time complexity is usually analyzed in a synchronous model).

Many broadcast algorithms exist (see the survey by Hedetniemi et al. [9]) but few of them are related to our model. The closest model is the one of Feige et al. [8]. The authors prove general lower and upper bounds ($\log_2 n$ and $O(n \ln n)$) on the time to broadcast a message *with high probability*¹ in any unknown graph. A contaminated node chooses a neighbor uniformly at random but no rendezvous are needed. In our model, the time complexity increases since a rendezvous has to be obtained to communicate. For a family of small-world graphs and other models (2-ports model but a node can only transmit a given message a bounded number of times), Comellas et al. [6] showed that a broadcast can always be done. A recent work of Karp et al. [11] deals with the *random phone call model*. In each round, each node u chooses another node v uniformly at random (more or less as in [8]) but the transmission of a rumor is done either from the caller to the called node (*push transmission algorithm*) or from the called node to the caller (*pull transmission algorithm*). The underlying topology is the complete graph and they prove that any rumor broadcasted in $O(\ln n)$ rounds needs to send $\omega(n)$ messages on expectation.

However, the results of random call phone [8,11] do not imply the presented results in the rendezvous model:

- The classes of graphs for which the broadcast runs fast or slow are different in the rendezvous model and in the random phone call model. For instance, the lower bound is $\Omega(\ln(n))$ in both models. Now, consider the complete graph, its broadcast time $O(n \ln(n))$ is close to the lower bound in the random phone call model whereas it becomes $\theta(n \ln(n))$ in the rendezvous model.
- We deal with the expected broadcast time. Depending on the topology, this time can be either equal or different to the broadcast time with high probability.

In the radio network setting (n -ports model), some algorithms and bounds exist whether the topology is known or unknown (see the survey of Chlebus [5]). However, the model of communication is different from ours: simultaneously, a node can send a message to all of its neighbors and a node can receive a message if and only if a unique neighbor send a message. Two kinds of algorithms are

¹ High probability means with probability $1 - O(n^{-c})$ for some positive constant c .

proposed in the radio model: with or without collision detection. In our model, there is no problem of collision.

Rendezvous in a broadcast protocol are used in applications like Dynamic Host Configuration Protocol but to the best of our knowledge, the analysis of a *randomized* rendezvous algorithm to broadcast in a network is new. The random rendezvous model was introduced in [15] in which the authors compute the expected number of rendezvous per round in a randomized algorithm. Their algorithm is a solution to implement synchronous message passing in an anonymous network that passes messages asynchronously [18]. Many concurrent programming languages including CSP and Ada use this method to define a communication between pairs of asynchronous processes. Angluin [2] proved that there is no deterministic algorithm for this problem (see the paper of Lynch [12] containing many problems having no deterministic solutions in distributed computing). In [16], the rendezvous are used to elect randomly a leader in an anonymous graph.

1.2. The model

Let $G = (V, E)$ be a connected and undirected graph of n vertices and m edges. For convenience and with respect to the problem of spreading an epidemic, a vertex is *contaminated* if it has received the message sent by an *initial vertex* v_0 .

The model can be implemented in a fully distributed way. The complexity analysis, however based on the concept of *rounds*, is commonly used in similar studies [8,15,16]. In our article, a *round* is the following sequence:

- for each $v \in V$, choose uniformly at random an incident edge;
- if an edge (v_i, v_j) has been chosen by v_i and v_j , there is a rendezvous;
- if there is a rendezvous and if only v_i is contaminated, then v_j becomes contaminated.

T_G is the *broadcast time* or *contamination time*, that is the number of rounds until all vertices of graph G are contaminated. T_G is an integer-valued random variable; in this paper, we concentrate the study on its expectation $\mathbf{E}(T_G)$.

Some remarks can be made on our model. As explained in Section 1, the rendezvous process (the first two steps of the round) keeps repeating forever and could be seen as a way of maintaining connectivity. Several broadcasts can take place simultaneously and we do not give any priority to any one of them, even if we study a broadcast starting from a given vertex v_0 .

We concentrate our effort on $\mathbf{E}(T_G)$ and we do not require that the algorithm finds out when the rumor sent by v_0 has reached all the nodes. However, some hints can be given: we can stop the broadcast algorithm (do not run the third step of the round) using a local control mechanism in each node of the network: if identities of the nodes are available (non anonymous networks), each node keeps into its memory a list of contaminated neighbors for each rumor and when this list contains all the neighbors, the process may stop trying to contaminate them (with the same rumor). If the network is anonymous and the number of nodes n is known, then it is possible to prove that in $O(n^2 \ln(n))$ rounds with high probability, all the neighbors of a contaminated node know the rumor.

In our algorithm, nodes of large degree and a large diameter increase the contamination time. Taking two adjacent nodes v_i and v_j of degrees d_i and d_j respectively, the expected number of rounds

to contaminate v_j from v_i is $d_i d_j$. For instance, take two stars of $n/2$ leaves. Join each center by an edge. In the rendezvous model, the expected broadcast time is $\Theta(n^2)$ whereas in [8]’s model, it will be $\Theta(n \ln(n))$ on expectation and with high probability. Starting from this example, $\mathbf{E}(T_G)$ can easily be upper bounded by $O(n^3)$ but we find a tighter upper bound.

1.3. Our results

The main result of the paper is to prove in Section 2 that for any graph G , $\log_2 n \leq \mathbf{E}(T_G) \leq O(n^2)$. More precisely, for any graph G of maximal degree Δ , $\mathbf{E}(T_G) = O(\Delta n)$. This main result is far from obvious.

In Section 3, we show that there are some graphs for which the expected broadcast time asymptotically matches either the lower bound or the upper bound up to a constant factor. For instance, for the complete balanced binary tree, $\mathbf{E}(T_G) = O(\log_2 n)$ whereas $\mathbf{E}(T_G) = \Omega(n^2)$ for the double star graph (two identical stars joined by one edge). For graphs of bounded degree Δ and diameter D , we also prove in Section 3 that $\mathbf{E}(T_G) = O(D\Delta^2 \ln \Delta)$. This upper bound is tight since for Δ -ary complete trees of diameter D , $\mathbf{E}(T_G) = \Omega(D\Delta^2 \ln \Delta)$. The complete graph was proved [15] to have the least expected number of rendezvous per round; nevertheless, its expected broadcast time is $\Theta(n \ln n)$.

2. Arbitrary graphs

The first section presents some terminology and basic lemmas that are useful for the main results.

2.1. Generalities on the broadcast process

The rendezvous process induces a *broadcast process*, that is, for each nonnegative integer t , we get a (random) set of vertices, V_t , which is the set of vertices that have been reached by the broadcast after t rounds. The sequence $(V_t)_{t \in \mathbf{N}}$ is a *homogeneous, increasing Markov process* with state space $\{U : \emptyset \subsetneq U \subset V\}$. Any state U contains the initial vertex v_0 and the subgraph induced by U is connected. State V is its sole absorbing state; thus, for each graph G , this process reaches state V (that is, the broadcast is complete) in finite expected time.

The transition probabilities for this Markov chain (V_k) depend on the rendezvous model. Specifically, if U and U' are two nonempty subsets of V , the transition probability $p_{U,U'}$ is 0 if $U \not\subseteq U'$, and, if $U \subseteq U'$, $p_{U,U'}$ is the probability that, in a given round, $U' - U$ is the set of vertices not in U that have a rendezvous with a vertex in U . Thus, the *loop probability* $p_{U,U}$ is the probability that each vertex in U either has no rendezvous, or has one with another vertex in U .

In the sequel, what we call the *broadcast sequence* is the sequence of *distinct* states visited by the broadcast process between the initial state $\{v_0\}$ and the final absorbing state V . A *possible broadcast sequence* is any sequence of states that has a positive probability of being the broadcast sequence; this is any sequence $\mathcal{X} = (X_1, \dots, X_m)$ such that $X_1 = V_0 = \{v_0\}$, $X_m = V$, and $p_{X_k, X_{k+1}} > 0$ for all k .

By d_u we denote the degree of vertex u . For a bounded degree graph, Δ is the maximal degree of the graph. By D we denote the diameter of the graph.

If $X_k = V_t$ is the set of the k contaminated vertices at time t then Y_k is the set of remaining vertices. We define the cut C_k as the set of edges that have one endpoint in X_k and the other in Y_k .

For any edge $a = (u, v) \in E$, $\mathbf{P}(a) = (d_u d_v)^{-1}$ (respectively, $\mathbf{P}(\bar{a})$) is the probability that edge a will obtain (respectively, not obtain) a rendezvous at a given round. The product $(d_u d_v)^{-1}$ is also called the *weight* of the edge a .

We also define two values for any set of edges $C \subset E$: $\mathbf{P}(\mathcal{E}_C)$ (respectively, $\mathbf{P}(\overline{\mathcal{E}_C})$) where \mathcal{E}_C is the event of obtaining a rendezvous in a round for at least one edge (respectively, no edge) in C ; and $\pi(C) = \sum_{a \in C} \mathbf{P}(a)$. Since $\pi(C)$ is the expected number of rendezvous in C , it is much easier to deal with in computations. Obviously, $\mathbf{P}(\mathcal{E}_C) \leq \pi(C)$ holds for any C . Lemma 2 provides a *lower bound* for $\mathbf{P}(\mathcal{E}_C)$ of the form $\Omega(\pi(C))$ provided $\pi(C)$ is not too large.

With these notations, for any set of vertices U , $p_{U,U} = 1 - \mathbf{P}(\mathcal{E}_{C_U})$, where C_U is the set of edges that have exactly one endpoint in U (the *cut* defined by the partition $(U, V - U)$).

Lemma 1. *Let C be any given subset of E . For any $a \in E$, we have $\mathbf{P}(a \mid \overline{\mathcal{E}_C}) \geq \mathbf{P}(a)$.*

Proof. Partition C into $C_1 \cup C_2$, where $C_1 = \{e' \mid e' \in C, e' \text{ incident to } a\}$ and $C_2 = C \setminus C_1$. Then we have:

$$\begin{aligned} \mathbf{P}(a \mid \overline{\mathcal{E}_C}) &= \mathbf{P}(a \mid \overline{\mathcal{E}_{C_1}} \wedge \overline{\mathcal{E}_{C_2}}) \\ &= \frac{\mathbf{P}(a \wedge \overline{\mathcal{E}_{C_1}} \mid \overline{\mathcal{E}_{C_2}})}{\mathbf{P}(\overline{\mathcal{E}_{C_1}} \mid \overline{\mathcal{E}_{C_2}})}. \end{aligned}$$

Since $\mathbf{P}(\overline{\mathcal{E}_{C_1}} \mid \overline{\mathcal{E}_{C_2}}) \leq 1$, we have:

$$\mathbf{P}(a \mid \overline{\mathcal{E}_{C_1}} \wedge \overline{\mathcal{E}_{C_2}}) \geq \mathbf{P}(a \wedge \overline{\mathcal{E}_{C_1}} \mid \overline{\mathcal{E}_{C_2}}).$$

Once there is a rendezvous on the edge a , there will be no rendezvous on the edges of C_1 . So we have:

$$\mathbf{P}(a \wedge \overline{\mathcal{E}_{C_1}} \mid \overline{\mathcal{E}_{C_2}}) = \mathbf{P}(a \mid \overline{\mathcal{E}_{C_2}}),$$

yielding

$$\mathbf{P}(a \mid \overline{\mathcal{E}_{C_1}} \wedge \overline{\mathcal{E}_{C_2}}) \geq \mathbf{P}(a \mid \overline{\mathcal{E}_{C_2}}).$$

The edge a being adjacent to none of the edges in C_2 , the fact that there is no rendezvous on this edges does not affect the probability of a rendezvous on the edge a . Therefore:

$$\mathbf{P}(a \mid \overline{\mathcal{E}_{C_1}} \wedge \overline{\mathcal{E}_{C_2}}) \geq \mathbf{P}(a).$$

Thus, for any $a \in E \setminus C$

$$\mathbf{P}(a \mid \overline{\mathcal{E}_C}) \geq \mathbf{P}(a). \quad \square$$

Lemma 2. *For any $C \subset E$, $\mathbf{P}(\mathcal{E}_C) \geq \lambda \min(1, \pi(C))$ with $\lambda = 1 - e^{-1}$ where $e = \exp(1)$.*

Proof. Assume that at time t , we have $V_t = X_k$, that is k nodes are contaminated.

There is no new contaminated vertex (and hence $|V_{t+1}| = k$), if and only if there is no rendezvous in C during one round. Let $\{e_1, e_2, \dots, e_l\}$ denote the set of edges of C . Then

$$\begin{aligned} \mathbf{P}(\overline{\mathcal{E}_C}) &= \mathbf{P}(\overline{e_1} \wedge \overline{e_2} \wedge \dots \wedge \overline{e_l}) \\ &= \mathbf{P}(\overline{e_1})\mathbf{P}(\overline{e_2} \mid \overline{e_1}) \cdots \mathbf{P}(\overline{e_l} \mid \overline{e_1} \wedge \overline{e_2} \wedge \dots \wedge \overline{e_{l-1}}) \\ &= (1 - \mathbf{P}(e_1))(1 - \mathbf{P}(e_2 \mid \overline{e_1})) \cdots (1 - \mathbf{P}(e_l \mid \overline{e_1} \wedge \overline{e_2} \wedge \dots \wedge \overline{e_{l-1}})). \end{aligned}$$

From Lemma 1, we have $\mathbf{P}(e_i \mid \overline{e_1} \wedge \overline{e_2} \wedge \dots \wedge \overline{e_{i-1}}) \geq \mathbf{P}(e_i)$. Hence

$$\mathbf{P}(\overline{\mathcal{E}_C}) \leq \prod_{i=1}^l (1 - \mathbf{P}(e_i))$$

and

$$\mathbf{P}(\mathcal{E}_C) \geq 1 - \prod_{i=1}^l (1 - \mathbf{P}(e_i)).$$

Now, since $1 - x \leq e^{-x}$, this becomes

$$\mathbf{P}(\mathcal{E}_C) \geq 1 - \prod_{i=1}^l e^{-\mathbf{P}(e_i)} = 1 - e^{-\pi(C)}.$$

The function $x \mapsto 1 - e^{-x}$ is increasing and concave, so that $1 - e^{-x} \geq \min(\lambda, \lambda x) = \lambda \min(1, x)$ (with $\lambda = 1 - e^{-1}$) holds for all $x \geq 0$. This proves the lemma. \square

Corollary 3.

$$\frac{1}{\mathbf{P}(\mathcal{E}_C)} \leq \frac{e}{e-1} \max\left(1, \frac{1}{\pi(C)}\right) \leq \frac{e}{e-1} \left(1 + \frac{1}{\pi(C)}\right). \tag{1}$$

Lemma 4. For any graph G , any integer k and any $p \in (0, 1)$, if $\mathbf{P}(T_G > k) \leq p$ then $\mathbf{E}(T_G) \leq k/(1 - p)$.

Proof. Cut the broadcast process into “segments” of k rounds, and consider the “broadcast-or-reset” process such that, at the beginning of each segment when the broadcast has not yet occurred, the set of contaminated vertices is reset to the initial vertex. Let X be the index of the segment in which the broadcast-or-reset process terminates. The hypothesis implies that X is geometrically distributed with parameter at least $1 - p$, so that $\mathbf{E}(X) \leq 1/(1 - p)$.

The broadcast-or-reset process cannot terminate before the broadcast process, so that $T_G \leq kX$. Taking expectations yields

$$\mathbf{E}(T_G) \leq \frac{k}{1 - p}. \tag{2}$$

Since the number of contaminated vertices can be at most doubled at each round, we have the following trivial lower bound. \square

Theorem 5. For any graph G , $T_G \geq \log_2 n$ with probability 1.

2.2. The general upper bound

We will prove the following:

Theorem 6. For any connected graph G with n vertices and maximum degree Δ , the broadcast time T_G satisfies

$$\mathbf{E}(T_G) \leq \frac{e}{e-1}(n-1)(6\Delta+1). \quad (3)$$

The proof of this theorem is a bit involved; we will sketch it before stating and proving a few lemmas.

The probability distribution for the full broadcast time T_G is not known, but, when *conditioned* by the sequence of states visited by the broadcast process, it becomes a sum of independent geometric random variables, for which the parameters are known exactly (Lemma 7). Thus, the conditional expectation of the broadcast time becomes the weight of some trajectory, which is defined as a sum of weights for the visited states. Each individual weight is upper bounded by an expression that only depends on individual rendezvous probabilities (Lemma 2 and Corollary 3), and then a uniform upper bound is obtained for the conditional expectations (Lemma 9); this uniform upper bound then straightforwardly translates into an upper bound for the (unconditional) expected broadcast time.

The next lemma is stated in a more general setting than our broadcasting process.

Lemma 7. Let $(M_t)_{t \in \mathbf{N}}$ be a homogeneous Markov chain with finite state space S and transition probabilities $(p_{x,y})_{x,y \in S}$.

Let $(T_k)_{k \in \mathbf{N}}$ denote the increasing sequence of stopping times defined by

$$\begin{aligned} T_0 &= 0 \\ T_{k+1} &= \inf\{t > T_k : M_t \neq M_{T_k}\}, \end{aligned}$$

and let $(M'_k)_{k \in \mathbf{N}}$ be the “trajectory” chain defined by

$$M'_k = \begin{cases} M_{T_k} & \text{if } T_k < \infty, \\ M'_{k-1} & \text{if } T_k = \infty. \end{cases}$$

Then, for any sequence x_0, \dots, x_N such that $x_{k+1} \neq x_k$ and $p_{x_k, x_{k+1}} > 0$ for $0 \leq k \leq N-1$, conditioned on $M'_k = x_k$ for $0 \leq k \leq N$, $\mathcal{T} = (T_{k+1} - T_k)_{0 \leq k \leq N-1}$ is distributed as a vector of independent geometric random variables with respective parameters $1 - p_{x_k, x_k}$.

Proof. The proof is straightforward. Let $\mathbf{t} = (t_0, \dots, t_{N-1})$ be any vector of positive integers. The event $\{\mathcal{T} = \mathbf{t} \wedge M'_k = x_k, 0 \leq k \leq N\}$ has probability

$$\mathbf{P}(M_0 = x_0) \prod_{i=0}^{N-1} p_{x_i, x_i}^{t_i-1} p_{x_i, x_{i+1}}. \quad (4)$$

Summing over all possible vectors \mathbf{t} , we get for the probability that the trajectory matches x_0, \dots, x_N :

$$\mathbf{P}(M'_k = x_k, 0 \leq k \leq N) = \mathbf{P}(M_0 = x_0) \prod_{i=0}^{N-1} \frac{p_{x_i, x_{i+1}}}{1 - p_{x_i, x_i}}. \tag{5}$$

Dividing (4) by (5) yields

$$\mathbf{P}(T = \mathbf{t} \mid M'_k = x_k, 0 \leq k \leq N) = \prod_{i=0}^{N-1} p_{x_i, x_i}^{t_i - 1} (1 - p_{x_i, x_i}), \tag{6}$$

which is indeed the distribution of a vector of independent geometric variables with the claimed parameters. \square

Corollary 8.

Let \mathcal{V} denote the trajectory of the loopless broadcast process (denoted M' in the statement of Lemma 7). Let $\mathcal{X} = (X_1, \dots, X_m)$ be any possible broadcast sequence, and $\mathcal{C} = (C_1, \dots, C_{m-1})$ the corresponding sequence of cuts. Then

$$\mathbf{E}(T_G \mid \mathcal{V} = \mathcal{X}) = \sum_{k=1}^{m-1} \frac{1}{\mathbf{P}(\mathcal{E}_{C_k})}.$$

Proof. Lemma 7 ensures that, conditioned on $\mathcal{V} = \mathcal{X}$, T_G is distributed as the sum of independent geometric random variables G_1, \dots, G_{m-1} , where G_k has parameter $1 - p_{X_k, X_k} = \mathbf{P}(\mathcal{E}_{C_k})$, which implies expectation $1/\mathbf{P}(\mathcal{E}_{C_k})$. Linearity of expectation yields the claim. \square

Lemma 9. Define the weight of any possible broadcast sequence \mathcal{X} as

$$w(\mathcal{X}) = \sum_{k=1}^{m-1} \frac{1}{\pi(C_k)}. \tag{7}$$

Then

$$w(\mathcal{X}) \leq 6(n - 1)\Delta. \tag{8}$$

Proof. We begin by noting that, since we are looking for a uniform upper bound on the weight, we can assume that $m = n$, which is equivalent to $|X_k| = k$ for all k (recall that in the slowest process, we have at most one new vertex contaminated per round). If such is not the case in a sequence \mathcal{X} , then we can obtain another possible sequence \mathcal{X}' with a higher weight by inserting an additional set X' between any two consecutive sets X_k and X_{k+1} such that $|X_{k+1} - X_k| \geq 2$, with $p_{X_k, X'}$ and $p_{X', X_{k+1}}$ meeting the condition that they are both positive; such an X' always exists, because each edge of every graph has positive probability of being the only rendezvous edge in a given round. This will just add a positive term to the weight of the sequence; thus, the sequence with the maximum weight satisfies $m = n$.

To prove that $\sum_{k=1}^{n-1} 1/\pi(C_k) \leq 6(n - 1)\Delta$, we prove that the integer interval $[1, n - 1]$ can be partitioned into a sequence of smaller intervals, such that, in each interval, the average value of $1/\pi(C_k)$ is at most 6Δ .

Assume that integers 1 to $k - 1$ have been thus partitioned, and let us consider C_k . If $\pi(C_k) \geq 1/(4\Delta)$ (that is, $1/\pi(C_k) \leq 4\Delta < 6\Delta$), we put k into an interval by itself and move on to $k + 1$. We now assume $\pi(C_k) < 1/(4\Delta)$, and set $1/\pi(C_k) = \alpha\Delta$ with $\alpha > 4$.

Let v be the next vertex to be reached by the broadcast after X_k , that is, $\{v\} = X_{k+1} - X_k$. This vertex must have at least one neighbor u in X_k .

Let $d \geq 1$ denote the number of neighbors of v that are in X_k . Each edge incident to v has weight at least $1/(d_v\Delta)$, and d of them are in C_k , so that we have $d/(d_v\Delta) \leq \pi(C_k) = 1/(\alpha\Delta)$, or equivalently,

$$d \leq d_v/\alpha. \quad (9)$$

Thus, $v \in X_{k+1}$ has $d_v - d$ neighbors in $Y_{k+1} = V - X_{k+1}$. Since at most one of them is added to X at each step of the sequence, this means that, for $0 \leq j \leq d_v - d$, Y_{k+1+j} contains at least $d_v - d - j$ neighbors of v . In other words, C_{k+1+j} contains at least $d_v - d - j$ edges that are incident to v , each of which has weight at least $1/(d_v\Delta)$. Consequently,

$$\frac{1}{\pi(C_{k+1+j})} \leq \frac{d_v\Delta}{d_v - d - j} \quad (10)$$

holds for $0 \leq j \leq d_v - d$.

The right-hand side of (10) increases with j , and for $j = \lfloor d_v/4 \rfloor$ (Eq. (9) and $\alpha > 4$),

$$\begin{aligned} \frac{d_v\Delta}{d_v - d - \lfloor d_v/4 \rfloor} &\leq \frac{d_v\Delta}{d_v - 2\lfloor d_v/4 \rfloor} \\ &\leq \frac{d_v\Delta}{\lceil d_v/2 \rceil} \\ &\leq 2\Delta. \end{aligned}$$

Summing (10) over $0 \leq j \leq \lfloor d_v/4 \rfloor$, we obtain

$$\sum_{j=0}^{\lfloor d_v/4 \rfloor} \frac{1}{\pi(C_k + 1 + j)} \leq 2\Delta \left(1 + \frac{d_v}{4}\right). \quad (11)$$

Since $d_v \geq \alpha$, we also have $1/\pi(C_k) \leq d_v\Delta$. Adding this to inequality (11), we now get

$$\begin{aligned} \frac{1}{\pi(C_k)} + \sum_{0 \leq j \leq \lfloor d_v/4 \rfloor} \frac{1}{\pi(C_k + 1 + j)} &\leq \Delta \left(\alpha + 2 + \frac{d_v}{2}\right) \\ &\leq \Delta \left(2 + \frac{3d_v}{2}\right). \end{aligned}$$

There are $2 + \lfloor d_v/4 \rfloor \geq 1 + \frac{d_v}{4}$ terms in the left-hand side of this inequality, so that the average value of $1/\pi(C_i)$, when i ranges over $[k, k + 1 + \lfloor d_v/4 \rfloor]$, is at most

$$\Delta \frac{2 + \frac{3d_v}{2}}{1 + \frac{d_v}{4}} \leq 6\Delta. \quad (12)$$

This concludes the recursion, and the proof. \square

Proof (Theorem 6).

Let \mathcal{X} be any possible broadcast sequence as in Lemma 9. Applying Corollary 3 to $C = C_k$ and summing over k , we get

$$\sum_k \frac{1}{\mathbf{P}(\mathcal{E}_{C_k})} \leq \frac{e}{e-1} \left(n-1 + \sum_k \frac{1}{\pi(C_k)} \right). \tag{13}$$

By Lemma 9, the right-hand side of (13) is at most

$$\frac{e}{e-1} (n-1 + 6\Delta(n-1)) = \frac{e(n-1)(6\Delta+1)}{e-1}. \tag{14}$$

By Lemma 7, the left-hand side of (13) is the conditional expectation of T_G . The upper bound remains valid upon taking a convex linear combination, so that we get, as claimed,

$$\mathbf{E}(T_G) \leq \frac{e(n-1)(6\Delta+1)}{e-1}. \tag{15}$$

Note. It should be clear that the constants are not best possible, even with our method of proof. They are, however, quite sufficient for our purpose, which is to obtain a uniform bound on the expected broadcast time.

The complete characterization of the distribution of T_G seems difficult and is left open. \square

3. Specific graphs

Theorems 5 and 6 provide lower and upper bounds on the expected contamination time for any graph. In this section, we prove that there exists some graphs for which the bounds can be attained.

The well-known coupon-collector problem (that is the number of trials required to obtain n different coupons if each round one is chosen randomly and independently. See [14] for instance) implies the next lemma:

Lemma 10. *For a star S of n leaves, $\mathbf{E}(T_S) = n \ln n + O(n)$.*

3.1. The l -star graphs

An l -star graph S_l is a graph built with a chain of $l+2$ vertices. Then, to each vertex different from the extremities, $\Delta-2$ leaves are added. Let S_l be a l -star graphs with $n = l(\Delta-1) + 2$ vertices. According to Theorem 6, $\mathbf{E}(T_{S_l}) = O(\Delta n) = O(\frac{n^2}{l})$. On the other hand, the expected number of rounds to get a rendezvous between the centers of two adjacent stars is Δ^2 and, therefore, the expected number of rounds for contaminating all the centers is $\Omega(l\Delta^2) = \Omega(n\Delta)$. As a corollary to this result we have

Proposition 11. *There exists an infinite family \mathcal{F} of graphs with n vertices and maximal degree Δ such that, for any $G \in \mathcal{F}$, $\mathbf{E}(T_G) = \Omega(\Delta n)$.*

It follows that the general upper bound $O(n^2)$ given by Theorem 6 is tight for the any l -star graph with $l \geq 2$ constant.

3.2. Matching the lower bound

To prove that the $\Omega(\ln(n))$ bound is tight, we prove an upper bound that only involves the maximum degree Δ and the diameter D .

Theorem 12. *Let G be any graph with maximum degree $\Delta \geq 3$ and diameter D . Then the expected broadcast time in G , starting from any vertex, is at most $4\Delta^2 (\ln 2 + D + D \ln \Delta)$.*

Our proof of this theorem will make use of the following lemma.

Lemma 13. *Fix a constant $p > 0$, and let Z_k denote the sum of k independent geometric random variables with parameter p .*

Then, for any $t \geq k/p$, we have

$$\mathbf{P}(Z_k > t) \leq \exp\left(-\frac{tp}{2} \left(1 - \frac{k}{tp}\right)^2\right).$$

Proof. We will use Hoeffding's inequality, as recalled in [13], Theorem 2.3: if $(X_i)_{1 \leq i \leq t}$ are independent random variables such that $0 \leq X_i \leq 1$ holds with probability 1 for each i , and $X = \sum_{i=1}^t X_i$ has expected value μ , then, for every positive ϵ ,

$$\mathbf{P}(X \leq (1 - \epsilon)\mu) \leq \exp\left(-\frac{\epsilon^2 \mu}{2}\right). \quad (16)$$

Let $(X_i)_{i \geq 1}$ be a sequence of independent Bernoulli trials, each one with probability of success p . Since the index of the first success in such a sequence is geometric with parameter p , the index of the k -th success in this sequence is distributed as Z_k . Thus,

$$\mathbf{P}(Z_k > t) = \mathbf{P}\left(\sum_{i=1}^t X_i < k\right). \quad (17)$$

Here we have $\mu = tp$, so that the right-hand side of (17) is of the form $\mathbf{P}(X \leq (1 - \epsilon)\mu)$ with $\epsilon = 1 - \frac{k}{tp}$, provided $t \geq k/p$. Applying (16) yields the claimed upper bound. \square

Proof (Theorem 12). We prove that the probability for the broadcast time to exceed half of the claimed bound is at most 1/2 and then use Lemma 4.

Let u be the initial vertex for the broadcast. For each other vertex v , pick a path γ_{uv} from u to v with length at most D . Since all degrees are at most Δ , each edge in γ_{uv} has a rendezvous probability at least $1/\Delta^2$. Hence, the broadcast time from u to v along the path γ_{uv} (that is, the time until the first edge has a rendezvous, then the second edge, and so on) is distributed as the sum of independent geometric random variables with parameters equal to the rendezvous probabilities, and is thus stochastically dominated by the sum of D independent geometric random variables with parameter $1/\Delta^2$.

Let T_{uv} denote the time until broadcast reaches v when the initial vertex is u ; Lemma 13 and the above discussion imply that, for any t ,

$$\mathbf{P}(T_{uv} > t) \leq e^{-\frac{t}{2\Delta^2} \left(1 - \frac{D\Delta^2}{t}\right)^2}. \quad (18)$$

Let n denote the number of vertices in G . Moore’s bound ensures that $n - 1 \leq \Delta^D$.

It is routine to check that, if $t > 2\Delta^2(\ln 2 + D + D \ln \Delta)$, then $t(1 - D\Delta^2/t)^2 > 2\Delta^2(\ln 2 + D \ln \Delta) \geq 2\Delta^2 \ln(2n - 2)$. Thus, for each of the $n - 1$ vertices $v \neq u$, we get

$$\mathbf{P}(T_{uv} > t) \leq e^{-\ln(2n-2)} = \frac{1}{2n-2}, \tag{19}$$

so that, summing over v , we get

$$\mathbf{P}(T_u > t) \leq \frac{1}{2}. \quad \square \tag{20}$$

Corollary 14. *There exists an infinite family of graphs \mathcal{F} such that, for any $G \in \mathcal{F}$, $\mathbf{E}(T_G) = O(\ln(|V|))$.*

Proof. For any integers $\Delta \geq 3$ and any h , the complete Δ -ary tree with diameter $2h$ has $\Delta \cdot (\Delta - 1)^{h-1} > (\Delta - 1)^h$ leaves (which implies that $\ln |V| \geq h \ln(\Delta - 1)$), and Theorem 12 states that its expected broadcast time is no larger than

$$4\Delta^2 (\ln 2 + 2h + 2h \ln \Delta) < 8\Delta^2 \left(1 + \frac{\ln \Delta}{\ln(\Delta - 1)}\right) \ln |V| + 4\Delta^2 \ln 2.$$

For any fixed $\Delta \geq 3$, this is $O(\ln |V|)$. \square

3.3. The complete graph

It seems also interesting to point out that the complete graph K_n has the minimal (see [15]) expected rendezvous number in a round:

$$\mathbf{E}(N_{K_n}) = \frac{\binom{n}{2}}{(n-1)^2},$$

which is asymptotically $\frac{1}{2}$. We prove in this section that its expected broadcast time is however $O(n \ln n)$, which is *significantly shorter* than that of the l -star graph with l constant which is $\Omega(n^2)$.

Lemma 15. $\mathbf{E}(T_{K_n}) \leq 2\lambda^{-1}n \ln n + O(n)$.

Proof. We bound the expected contamination time from above, by allowing at most one new contaminated vertex per round. In this new and pessimistic contamination process, we sum up the expected time to increase the number of contaminated vertices by one:

$$E(T_{K_n}) \leq \sum_{k=1}^{n-1} \frac{1}{\mathbf{P}(\mathcal{E}_{C_k})}.$$

Since $\pi(C_k) = \frac{k(n-k)}{(n-1)^2} < 1$, we can apply Lemma 2 with $\mathbf{P}(\mathcal{E}_{C_k}) \geq \lambda\pi(C_k)$. It turns out that

$$E(T_{K_n}) \leq \lambda^{-1}(n-1)^2 \sum_{k=1}^{n-1} \frac{1}{k(n-k)} = \lambda^{-1}(n-1)^2 \frac{2}{(n-1)} H_{n-1}.$$

Since $H_n \sim \ln n + 0.57721\dots + o(1)$ we obtain $E(T_{K_n}) \leq 2\lambda^{-1}(n-1) \ln(n-1) + O(n)$. \square

Moreover, we have:

Lemma 16. *With probability $1 - n^{-1/2}$, $T_{K_n} \geq \frac{1}{2}n \ln n$.*

Proof. Let v_t denote the number of contaminated vertices at round t . Then, we have

$$\mathbf{E}(v_{t+1} \mid v_t = k) = k \left(1 + \frac{n-k}{(n-1)^2} \right)$$

and then

$$\mathbf{E}(v_{t+1} \mid v_t) = v_t \left(1 + \frac{n-v_t}{(n-1)^2} \right) = v_t \left(1 + \frac{n}{(n-1)^2} \right) - \frac{v_t^2}{(n-1)^2} \leq v_t \left(1 + \frac{n}{(n-1)^2} \right),$$

yielding

$$\mathbf{E}(v_t) \leq \left(1 + \frac{n}{(n-1)^2} \right)^t.$$

For any $\alpha < 1$ positive real value, we have $(1 + \frac{n}{(n-1)^2})^t \leq \alpha n$ whenever $t \ln \left(1 + \frac{n}{(n-1)^2} \right) \leq \ln(\alpha n)$. It follows that if $t < t_0(n, \alpha) = (\ln n + \ln \alpha) \frac{(n-1)^2}{n}$ then $\mathbf{E}(v_t) \leq \alpha n$. By the Markov inequality, we have

$$\mathbf{P}(v_t \geq n) = \mathbf{P}(v_t = n) \leq \frac{\mathbf{E}(v_t)}{n} \leq \alpha.$$

By definition, $v_t = n$ if and only if $T_{K_n} \leq t$. Hence

$$\mathbf{P}(T_{K_n} > t) \geq 1 - \alpha$$

and then using again the Markov inequality, we have

$$\mathbf{E}(T_{K_n}) \geq t \mathbf{P}(T_{K_n} > t) \geq (1 - \alpha)t.$$

With $\alpha = \frac{1}{\sqrt{n}}$ and $t = t_0(n, \alpha)$, we obtain:

$$\mathbf{E}(T_{K_n}) \geq \frac{1}{2} \ln n \frac{(n-1)^2}{n} \left(1 - \frac{1}{\sqrt{n}} \right)$$

yielding

$$\mathbf{P} \left(T_{K_n} \geq \frac{1}{2} \ln n \frac{(n-1)^2}{n} \right) \geq 1 - \frac{1}{\sqrt{n}},$$

i.e., with probability $1 - n^{-1/2}$, we have

$$T_{K_n} \geq \frac{1}{2} n \ln n. \quad \square$$

Lemmas 16 and 15 imply:

Proposition 17. $\mathbf{E}(T_{K_n}) = \Theta(n \ln n)$.

3.4. Δ -regular balanced rooted trees with bounded diameter

Lemma 18. *Let G be a Δ -regular balanced complete rooted tree of depth 2. The expected time for the root to contaminate its children is $\Theta(\Delta^2 \ln \Delta)$.*

Proof. This lemma is a variation of the coupon collector problem (See Lemma 10). Let v be the root of tree. The probability of rendezvous in one round for vertex v is $\frac{1}{\Delta}$. Let T_k be the number of rounds required for v to obtain a rendezvous with a new vertex knowing that k is the number of its children already contaminated. This event occurs with probability $\frac{\Delta-k}{\Delta^2}$ and implies $\mathbf{E}(T_k) = \frac{\Delta^2}{\Delta-k}$. Hence, we have

$$\mathbf{E}(T_d) = \sum_{k=0}^{\Delta-1} \mathbf{E}(T_k) = \sum_{k=0}^{\Delta-1} \frac{\Delta^2}{\Delta-k} = \Delta^2 \sum_{k=1}^{\Delta} \frac{1}{k} = \Delta^2 H_{\Delta} = \Theta(\Delta^2 \ln \Delta). \quad \square$$

Theorem 19. *Let G be a Δ -regular balanced complete rooted tree of depth $D/2$ with D even. $\mathbf{E}(T_G) = \Omega(D\Delta^2 \ln \Delta)$.*

Proof. Suppose the broadcast starts from the root v_0 . Let us construct a path $v_0, v_1, v_2, \dots, v_{D/2}$ such that v_i is the last contaminated child of v_{i-1} . T_{v_i} denotes the number of rounds to contaminate v_j by its parent v_{i-1} once v_{i-1} is contaminated. Since $T_G \geq \sum_{i=1}^{D/2} T_{v_i}$ and from Lemma 18, for every $1 \leq i \leq D/2$, $\mathbf{E}(T_{v_i}) = \Theta(\Delta^2 \ln \Delta)$, we have $\mathbf{E}(T_G) \geq \sum_{i=1}^{D/2} \mathbf{E}(T_{v_i}) = \Omega(D\Delta^2 \ln \Delta)$. \square

Theorem 19 proves that there exists a graph for which the upper bound of Theorem 12 is tight.

References

- [1] S. Albers, M. Henzinger, Exploring unknown environments, *SIAM Journal on Computing* 29 (4) (2000) 1164–1188.
- [2] D. Angluin, Local and global properties in networks of processors, in: *Proceedings of the 12th Symposium on theory of computing*, 1980, pp. 82–93.
- [3] L. Barriere, P. Flocchini, P. Fraigniaud, N. Santoro, Capture of an intruder by mobile agents, in: *14th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 2002, pp. 200–209.
- [4] M.A. Bender, D.K. Slonim, The power of team exploration: two robots can learn unlabeled directed graphs, in: S. Soddif (Ed.), *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 75–85.
- [5] B. Chlebus, Handbook on Randomized Computing, in: P.M. Pardalos, S. Rajasekaran, J.H. Reif, J.D.P. Rolim, (eds.), *Randomized communication in radio networks*, Kluwer Academic (2001). <http://citeseer.nj.nec.com/489613.html>.
- [6] F. Comellas, J. Ozón, J.G. Peters, Deterministic small-world communication networks, *Information Processing Letters* 76 (1–2) (2000) 83–90.
- [7] X. Deng, T. Kameda, C.H. Papadimitriou, How to learn an unknown environment i: the rectilinear case, *Journal of the ACM* 45 (2) (1998) 215–245.
- [8] U. Feige, D. Peleg, P. Raghavan, E. Upfal, Randomized broadcast in networks, *Random Structures and Algorithms* 1 (1990).
- [9] S.M. Hedetniemi, S.T. Hedetniemi, A.L. Liestman, A survey of gossiping and broadcasting in communication networks, *Networks* 18 (1988) 319–349.
- [10] M. Habib, C. McDiarmid, J. Ramirez-Alfonsin, B. Reed, *Probabilistic Methods for Algorithmic Discrete Mathematics*, Springer, Berlin, 1998.

- [11] R.M. Karp, C. Schindelhauer, S. Shenker, B. Vocking, Randomized rumor spreading, *IEEE Symposium on Foundations of Computer Science (2000)* 565–574.
- [12] N. Lynch, A hundred impossibility proofs for distributed computing, in: *Proceedings of the 8th ACM Symposium on Principles of Distributed Computing PODC*, ACM Press, New York, NY, 1989, pp. 1–28.
- [13] C. McDiarmid, Concentration, pp. 195–248. In Habib et al. [10], 1998.
- [14] R. Motwani, P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, 1995.
- [15] Y. Metivier, N. Saheb, A. Zemmari, Randomized rendezvous, *Trends in Mathematics (2000)* 183–194.
- [16] Y. Metivier, N. Saheb, A. Zemmari, Randomized local elections, *Information Processing Letters* 82 (2002) 313–320.
- [17] N. Rao, S. Karet, W. Shi, S. Iyengar, *Robot Navigation in Unknown Terrains: Introductory Survey of Non-Heuristic Algorithms*. 1993. <http://citeseer.nj.nec.com/rao93robot.html>.
- [18] G. Tel, *Introduction to Distributed Algorithms*, Cambridge University Press, Cambridge, 2000.



The compactness of adaptive routing tables

Cyril Gavoille *, Akka Zemmari

LaBRI, Université Bordeaux I, France

Abstract

The compactness of a routing table is a complexity measure of the memory space needed to store the routing table on a network whose nodes have been labelled by a consecutive range of integers. It is defined as the smallest integer k such that, in every node u , every set of labels of destinations having the same output in the table of u can be represented as the union of k intervals of consecutive labels. While many works studied the compactness of deterministic routing tables, few of them tackled the adaptive case when the output of the table, for each entry, must contain a fixed number α of routing directions. We prove that every n -node network supports shortest path routing tables of compactness at most n/α for an adaptiveness parameter α , whereas we show a lower bound of $n/\alpha^{O(1)}$.

© 2003 Elsevier B.V. All rights reserved.

Keywords: Compact routing tables; Adaptive routing; Interval routing

1. Introduction

1.1. Generalities

Given a parallel or distributed system, the interconnection network ensures the communication between the processors, the terminal nodes. Each intermediate node has a router, a dedicated co-processor which forwards the messages between processors through the links of the underlying topology. The routers run a distributed algorithm which specifies the way to go from a node of the network to another. This algorithm is described by a routing function.

Once a router receives a message, it looks at its header and checks the destination of the message, and finds the output port that will be used to forward the message towards to next intermediate node up to its destination. The output port is a number local to each router

* Corresponding author.

E-mail addresses: gavoille@labri.fr (C. Gavoille), zemmari@labri.fr (A. Zemmari).

and associated to each link between routers. A standard way to implement such algorithms is to use a *routing table*. To find the output port, the router consults a table which is kept in its local memory. For each destination, this table returns the output port number through which the message can be forwarded.

A simple method to organize this table is to associate to each destination the output port number which can serve it. This method is simple, but it is very memory expensive. It requires $O(n \log d)$ bits to maintain the routing table in each node of degree d , where n is the number of nodes of the underlying graph representing the network.

For a large or growing network, this method is not feasible. It is interesting to look for another method in order to reduce the size of the data structure stored by the routers, and used for the routing task. In the field of compact routing, several methods and strategies were introduced to reduce the router memory size, as separator-based routing schemes [13, 14], hierarchical routing schemes [2,25], prefix routing [3], Boolean routing [9], and interval routing [27,30]. We focus our work on the latter technique that offers a more compact data structure for routing tables.

1.2. The interval routing schemes

The interval routing was introduced by Santoro and Khatib in [27], and extended in [30] by van Leeuwen and Tan. It has been intensively studied in recent years, and an overview can be found in [16]. This method consists of finding a global labelling of the nodes with integers taken from $\{1, 2, \dots, n\}$, and, given a routing table, to group in the smallest set of intervals the destination labels using the same output port in each node. An interval means a set of consecutive integers, the labels 1 and n being considered as consecutive. If there exists a routing table such that each set of destination labels using a same output port can be grouped with at most k intervals, we deal with a k -interval routing scheme for this network, k -IRS for short. A k -IRS can be implemented with $O(kd \log n)$ bits per node by storing the interval boundaries of the destinations. Actually, this naive coding can be slightly compressed into $O(kd \log(n/k))$ bits [16]. In a sense, interval routing is a compact implementation of routing tables. One can hope to store only $O(k)$ integers per node for bounded degree networks using k -IRS, whereas standard routing tables require $O(n)$ integers. The parameter k is called *compactness* of a routing table.

Many works try to determine routing tables with minimum compactness under several assumptions on the quality of the routing measured in term of length of the routes: shortest path routing [12,17,18,20], stretched routing [4,11,24], routing with bounded dilation [6, 15,22,24,28], etc. (cf. [16]). Nevertheless, these works have studied only the deterministic case: for each source-destination pair, the routing table encodes only one routing path. So, the routing path is completely determined by giving the intervals. On the contrary, *adaptive* routing allows to diversify the routing paths. A destination can belongs to more than one set of intervals. For interval routing schemes, this extension has been partially suggested in [29].

1.3. Adaptive routing tables

More precisely, let us define an α -adaptive routing table as a routing table in which every destination can be founded in each router for exactly α different output ports, for some integer $\alpha \geq 1$. Similarly, an α -adaptive k -interval labelling scheme, or k -ILS $_{\alpha}$ for short, is an α -adaptive routing table for which the set of destination labels using the same output port can be grouped into at most k intervals. An ILS $_{\alpha}$ is termed *valid* if for every source-destination pair u, v , with¹ $u \neq v$, there exists in u (and all the other intermediate nodes) *at least one* output port among the α possible ones that induces a route to v . A valid ILS $_{\alpha}$ is called an α -adaptive interval routing scheme, IRS $_{\alpha}$ for short. Therefore, a k -IRS $_{\alpha}$ is simply an α -adaptive routing table of compactness k .

The definition captures adaptiveness of a routing, since at each step the router can select the next edge of the route among² α . This potentiality provides many routing paths, but not necessarily entirely disjoint paths that would require some strong assumptions on the edge-connectivity of the network. In this model some routes may loop. The router has the guarantee that at least one route connects to the destination. The other paths are called *deflecting paths*. They can be used depending on the load of the network, or on every other parameters, in order to improved the traffic. The case $\alpha = 1$ corresponds to the deterministic one (no deflecting paths).

Of course, in practice, for a complete implementation of a routing protocol, a *selection* function must choose one output port among the valid set. The adaptiveness of a routing table of compactness k implies to store in the router a total of $O(kd \log(n/k)) + |S|$ bits of routing information, where $|S|$ represents the number of bits needed to code the selection function S encoding the policy of the router. For instance, a kind of routing policy may consist to choose at random a permutation of the possible paths returned by the router if several³ messages come in the router at a same time (this occurs, for instance, when the messages cannot be stored locally due to physical constraints of the router). In this case $|S|$ is just the size of a pseudo-random generator. A selection function may also provide some priority ordering between the routing paths. In this case it requires to store extra bits, and $|S|$ might be large. In particular S must differentiate routing paths from deflecting paths. In all the cases, our approach consists in splitting the memory requirements of the router in two parts: one required by the routing tables (the term $O(kd \log(n/k))$), and the other part required by the selection function (the term $|S|$).

In this paper, we are not interested in the coding of the selecting function S , but rather in the parameter k , the compactness. This latter parameter depends on the graph topology only, whereas the coding of the selection function may depend on the strategy to optimize the traffic: the links can be chosen at random, or selected according to

¹ In the framework of compact routing a common assumption is that the destination of a message is never its source. The case $u = v$ can be solved by the local processor (assumed having a relatively high computational level) without any communication with its router. This allows to establish more flexible and deeper results in particular for space memory lower bounds.

² As we will see the degree of the node has to be at least α .

³ No more than α .

some load history tables of the links, or predicted from some other arbitrary policies (deadlock-free, ...). We observe that a space complexity measure that would combine both terms suffers of the general $\Omega(n \log d)$ bit/node lower bound (and an $\Omega(n)$ intervals for the compactness) that applies to shortest path deterministic routing tables [18, 19]. Indeed, as we will see more precisely in Section 4.3, an adaptive routing table and a selection function encode together at least a deterministic routing table. However, such a combination does not allow to measure precisely the contribution of each part (for instance, the $\Omega(n)$ -lower bound on the compactness [18] does not apply for shortest path α -adaptive routing tables, cf. Section 4). So, our approach allows to measure the balance between the information needed for the adaptive routing table and the selection function.

1.4. Related works

Previous works on compact and adaptive routing schemes can be founded in [1,2,9,21] for general schemes, and in [10,11,23,26] for interval routing schemes and its generalizations. However, most of these works try to give a compact representation of *all* the shortest paths. Although these schemes extend the deterministic case, they suffer by the fact that many general lower bounds for deterministic routing established in [12,18–21] apply as well for the adaptive case. Indeed, these lower bounds are based on the uniqueness of the shortest paths between specific subset of nodes in some worst-case graphs. Thus, on these graphs all-shortest-path routing would consist to route along one shortest path as in deterministic routing. In essence, all-shortest-path compact routing schemes are not more compact than deterministic shortest path routing schemes. For instance, the asymptotic $n/4$ -lower bound on the compactness for deterministic shortest path IRS applies also for all-shortest-path IRS [18].

1.5. Our results

As we will see in the following, the situation is better thanks to the definition we propose for α -adaptive routing tables (IRS_α), specially whenever $\alpha > 1$ and becomes larger. All previously cited lower bounds does not apply in that case, and moreover we show that n/α intervals per arc suffice for shortest path IRS_α that is already better than the deterministic case whenever $\alpha \geq 4$.

This paper is organized as follows. Section 2 defines more precisely the model of α -adaptive routing tables. In Section 3 we show that every routing tables can be transformed on an α -adaptive routing table with the same set of routes and the same compactness. In particular we show that n/α intervals per arc suffice, even if shortest paths are required. In Section 4 we specifically study more deeply shortest path routing tables, and we show an existential $n/\alpha^{O(1)}$ -lower bound for the compactness, that is asymptotically optimal for constant α . We conclude in Section 5 by some possible extensions and perspectives of this work.

2. Preliminaries

In this paper, the network is modeled by a connected graph $G = (V, E)$, whose set of nodes V represents the routers, and whose set of arcs E the communication links between the routers. We assume that the links are bi-directional, i.e., if $(u, v) \in E$ then $(v, u) \in E$; G is a symmetric digraph.⁴ For every $u \in V$, we denote by $\deg(u)$ the number of neighbors of u corresponding to the common value of in- and out-degree of u . Finally, $\delta(G)$ denotes the minimum degree of G , that is $\delta(G) = \min\{\deg(u) \mid u \in V\}$.

2.1. Definitions

Formally, an *interval labelling scheme* on an n -node G is a pair $(\mathcal{L}, \mathcal{I})$ of functions where $\mathcal{L}: V \rightarrow \{1, \dots, n\}$ is a one-to-one labelling of the nodes, and $\mathcal{I}: E \rightarrow 2^{\mathcal{L}(V)}$ is a labelling of the arcs such that, for every arc $(u, v) \in E$, $\mathcal{L}(w) \in \mathcal{I}(u, v)$ if and only if the route from u to w uses the arc (u, v) .

Moreover, given an integer $\alpha \geq 1$, the pair $(\mathcal{L}, \mathcal{I})$ is an α -*adaptive interval labelling scheme*, ILS_α for short, if for all $u, w \in V$, $w \neq u$, the set

$$\{(u, v) \in E \mid \mathcal{L}(w) \in \mathcal{I}(u, v)\}$$

is of cardinality α . A *valid* ILS_α is called an IRS_α (α -adaptive interval routing scheme or α -adaptive routing table), if it fulfills the connectivity condition: for all $u, w \in V$, $w \neq u$, there exists a sequence $\rho(u, w) = (v_1, \dots, v_t)$ of nodes such that $v_1 = u$ and $v_t = w$, and for every $i \in \{1, \dots, t-1\}$, $\mathcal{L}(w) \in \mathcal{I}(v_i, v_{i+1})$. The sequence $\rho(u, w)$ is called a *routing path* or *route* from u to w , and may not form a simple path in G .

A *shortest path* IRS_α is an IRS_α for which, for any pair u, w , there exists a routing path $\rho(u, w)$ that is a shortest path in G . This definition easily extends to weighted graphs considering paths of minimum cost. We insist on the fact that between u and w there is at least one routing path $\rho(u, w)$ that is a shortest path, although many routing paths might be represented by the labelling. As said before in Section 1.4, the main interest of this condition is to avoid the $n/4$ -lower bound of [18] on the compactness.

Remark. A consequence of the previous definition is that only the graphs of minimum degree at least α support an ILS_α , and thus an IRS_α . A variant of the previous definition to overcome this problem would consist to impose that

$$|\{(u, v) \in E \mid \mathcal{L}(w) \in \mathcal{I}(u, v)\}| = \min\{\alpha, \deg(u)\}.$$

Although all the results we propose in this paper hold for both definitions, for simplicity, only the former definition is considered in the sequel.

⁴ However, many of the results presented in this paper are still valid for nonsymmetric and strongly connected digraphs.

2.2. Compactness

The *compactness* of an $\text{ILS}_\alpha(\mathcal{L}, \mathcal{I})$ is the smallest integer k such that every set $\mathcal{I}(u, v)$ can be represented as the union of at most k intervals of consecutive integers of $\{1, 2, \dots, n\}$ (1 and n being considered as consecutive). Such ILS_α and IRS_α are denoted respectively by $k\text{-ILS}_\alpha$ and $k\text{-IRS}_\alpha$.

Remark. For $\alpha = 1$, all the definitions match with the standard ILS/IRS introduced by [27, 30]. For simplicity, we denote in the sequel IRS for IRS_1 . The labellings we consider in this paper are supposed to be *strict*, i.e., we impose that $\mathcal{L}(u) \notin \mathcal{I}(u, v)$, for every $(u, v) \in E$.

3. A general labelling scheme

We show in this section that every graph G supports a 1-IRS_α for every $\alpha \leq \delta(G)$, the routing paths being not necessary shortest paths. This result can be seen as a generalization of the labelling scheme of [27] (showing that every graph has a 1-IRS), and will be a tool for the remaining of the paper. We denote by $[1, n]$ the set $\{1, 2, \dots, n\}$.

Theorem 3.1. *Let $(\mathcal{L}, \mathcal{I}_A)$ be any $k\text{-IRS}_\alpha$ on an n -node graph $G = (V, E)$ with $\delta(G) \geq \alpha + 1$, and let $Y \subseteq E$ such that every $x \in V$ has at most one neighbor y so that $(x, y) \in Y$. Then, $(\mathcal{L}, \mathcal{I}_A)$ can be transformed in polynomial time into a $k\text{-IRS}_{\alpha+1}$ on G , $(\mathcal{L}, \mathcal{I}_B)$, such that all the routes represented by $(\mathcal{L}, \mathcal{I}_A)$ are preserved in $(\mathcal{L}, \mathcal{I}_B)$, and such that for every $(x, y) \in Y$, $\mathcal{I}_B(x, y) = [1, n] \setminus \{\mathcal{L}(x)\}$.*

Proof. For every $z \in [1, n]$, let us denote $\text{succ}(z)$ (respectively $\text{pred}(z)$) the successor (respectively predecessor) of z in $[1, n]$ modulo n . Formally, $\text{succ}(z) = (z \bmod n) + 1$, and $\text{pred}(z) = (z + n - 2 \bmod n) + 1$. Let us define the following procedure of inputs $(\mathcal{L}, \mathcal{I}_A)$ and Y , and of output $(\mathcal{L}, \mathcal{I}_B)$ satisfying the statement of Theorem 3.1.

For every node x do (possibly in parallel):

- (1) For every $(x, y) \in E$, set $\mathcal{I}_B(x, y) \leftarrow \mathcal{I}_A(x, y)$.
- (2) Set $R \leftarrow [1, n] \setminus \{\mathcal{L}(x)\}$.
- (3) Let (x, y) be the unique arc of Y (if y does not exist go to 4), set $\mathcal{I}_B(x, y) \leftarrow R$, and update $R \leftarrow \mathcal{I}_A(x, y)$.
- (4) While $R \neq \emptyset$ do:
 - (a) Find y and z such that $(x, y) \in E$, $z \in R \setminus \mathcal{I}_B(x, y)$, and either $\text{pred}(z) \in \mathcal{I}_B(x, y)$ or $\text{succ}(z) \in \mathcal{I}_B(x, y)$.
 - (b) Find y' such that $(x, y') \in E$, and $z \in \mathcal{I}_B(x, y')$. Let $[a, b]$ an interval such that $z \in [a, b] \subseteq \mathcal{I}_A(x, y')$.
 - (c) Update $R \leftarrow R \setminus ([a, b] \setminus \mathcal{I}_B(x, y))$.
 - (d) Update $\mathcal{I}_B(x, y) \leftarrow \mathcal{I}_B(x, y) \cup [a, b]$.

Intuitively, the procedure consists on finding a label $z \in R$ such that its predecessor (or successor) is a boundary of some intervals of $\mathcal{I}_B(x, y)$. Then we append $[a, b]$, an interval

containing z , to $\mathcal{I}_B(x, y)$ solving the problem for z (at least). The procedure iterates on the updated version of R .

Let us consider any node x . Let us show that for every i , at the beginning of the i th run of Instruction 4 (at the test $R \neq \emptyset$), the set R fulfills the following property P_i : R contains at most $n - i$ labels, and if $z \in R$ then z appears in α sets \mathcal{I}_B , and otherwise $z = \mathcal{L}(x)$ or z appears in $\alpha + 1$ sets \mathcal{I}_B . In other words, at each loop, R denotes the set of labels that remains to treat.

By induction on i : the first time in Instruction 4, if no arc $(x, y) \in Y$ exists, R is the set of all the labels (except for $\mathcal{L}(x)$), and \mathcal{I}_B is initialized to \mathcal{I}_A . Hence, if there is no arc $(x, y) \in Y$, P_1 is true. Otherwise, after Instruction 3, all labels remaining in R appear exactly in α sets \mathcal{I}_B (the others appear already in $\alpha + 1$ sets by setting $\mathcal{I}_B(x, y) = [1, n] \setminus \{\mathcal{L}(x)\}$). Hence in any cases P_1 is true.

Now, assume the property holds up to the i th loop. To show that P_{i+1} is true, let us first show that Instruction 4(a) is doable, that is the pair (y, z) can be founded: first, if $i = 1$, then it suffices to choose any y such that $(x, y) \notin Y$ and $\mathcal{I}_A(x, y) \neq R$ (it must exist otherwise every label $\neq \mathcal{L}(x)$ would appear in at least $\delta(G) \geq \alpha + 1$ sets \mathcal{I}_A). Then, we can choose any $z \notin \mathcal{I}_A(x, y)$ (thus $z \in R$) so that $\text{pred}(z) \in \mathcal{I}_A(x, y)$ or $\text{succ}(z) \in \mathcal{I}_A(x, y)$.

For $i > 1$, a pair (y, z) exists otherwise, z and $\text{pred}(z)$ (or $\text{succ}(z)$) would appear in the same number of sets \mathcal{I}_B . By Property P_i , $z \in R$ implies $\text{pred}(z)$ or $\text{succ}(z) \in R$ (otherwise they would not appear in the same number of sets \mathcal{I}_B). This implies that $R = [1, n] \setminus \{\mathcal{L}(x)\}$, which is not possible since $|R| \leq n - i < n - 1$ ($i > 1$). So, Instruction 4(a) is doable. Instruction 4(b) is doable since $z \in R$ and by Property P_i z appears in $\alpha \geq 1$ sets \mathcal{I}_B . Instructions 4(c) and 4(d) are doable as well. We remark, that in Instruction 4(c), $|R|$ decreases by at least one element: $[a, b]$ contains at least z . We check that all labels removed from R appears in exactly $\alpha + 1$ sets \mathcal{I}_B . Therefore, P_{i+1} holds.

So, at the end of the last loop ℓ , R is empty and by Property P_ℓ , all the labels appear in $\alpha + 1$ sets \mathcal{I}_B . Taking a union in Instruction 4(d), we guarantee that $\mathcal{I}_A(x, y) \subseteq \mathcal{I}_B(x, y)$, and thus it preserves the routes. It follows that $(\mathcal{L}, \mathcal{I}_B)$ is a valid $\text{ILS}_{\alpha+1}$. Moreover, in Instruction 4(d), because $\text{pred}(z)$ and z are consecutive modulo n , and because $z \in \mathcal{I}_A(x, y)$ and $\text{pred}(z) \in \mathcal{I}_A(x, y)$, we have that the minimum number of intervals to represent $\mathcal{I}_B(x, y)$ never increase and thus is at most the one of $\mathcal{I}_A(x, y)$. So, $(\mathcal{L}, \mathcal{I}_B)$ has compactness at most k , and by Instruction 3, all the arcs of Y have the interval $[1, n] \setminus \{\mathcal{L}(x)\}$. This completes the proof. \square

Remark. We do not precise the time complexity of the previous algorithm because it may depend on the data structure used to code the input IRS (the one achieving the lowest time complexity is not necessary the most compact one). Anyway, using naive interval coding representation of IRS, this time is less than $O(n^4)$, but can easily be reduced to $O(|E|k\alpha)$ with more efficient data structures.

Using a spanning tree T of G , a DFS-based 1-IRS₁ on T (cf. [27]), and applying inductively on α in Theorem 3.1 we have:

Corollary 3.2. *Every graph G such that $\delta(G) \geq \alpha$, supports a 1-IRS_α .*

Whereas for $\alpha = 1$ every graph has a $k\text{-IRS}$ with $k \leq n/2$, for $\alpha > 1$ we show that $k \leq n/\alpha$. More precisely:

Theorem 3.3. *Let $(\mathcal{L}, \mathcal{I})$ be any $k\text{-IRS}_1$ on an n -node graph G , and let $\alpha \leq \delta(G)$. Then, G supports a $k'\text{-IRS}_\alpha$ such that all the routes of $(\mathcal{L}, \mathcal{I})$ are preserved, and such that $k' \leq \min\{k, (n-1)/\alpha\}$.*

Proof. The statement is obvious for $\alpha = 1$. Assume, $\alpha \geq 2$. We build a set Y composed of the arcs assigned with the largest number of intervals, for each node. After the first application of Theorem 3.1, we obtain a $k\text{-IRS}_2$ for G , with the same set of routes, and where the arc with the largest number of intervals (for each node) is now reduced to one. We can re-apply Theorem 3.1 with a new set Y still composed of the arcs assigned with the largest number of intervals, which is hence at most the second largest one in $(\mathcal{L}, \mathcal{I})$. Finally, after a total of $\alpha - 1$ applications of Theorem 3.1 (this is feasible since $\alpha \leq \delta(G)$), we obtain a $k'\text{-IRS}_\alpha$ with the same set of routes where the maximum number of intervals assigned on an arc, k' , is bounded by k and also by the α th largest number of intervals assigned on an arc in $(\mathcal{L}, \mathcal{I})$.

Let x be any node of G . Let $d = \deg(x)$, and let k_1, \dots, k_d be the number of intervals of the sets $\mathcal{I}(x, y)$ (the $k\text{-IRS}_1$ defined on G) for all neighbors y of x . Moreover assume $k_1 \geq \dots \geq k_d$. We have $\sum_{i=1}^\alpha k_i \leq \sum_{i=1}^d k_i \leq n - 1$ ($\alpha \leq d$ and the label of x is not assigned). Thus $k_\alpha \leq (n - 1)/\alpha$. As said before, $k' \leq \min\{k, k_\alpha\}$ completing the proof.

Remark. Theorem 3.1 can be slightly improved to

$$k' \leq \min\{k, (n - 1 - \delta(G))/\alpha + 1\}$$

if all the incident arcs of each node are labelled with non-empty labels (in this case we have $\sum_{i=1}^\alpha k_i \leq n - 1 - (\delta(G) - \alpha)$). This assumption occurs, for instance, for shortest paths routing tables.

4. Shortest path labelling

In this section we are interested in IRS_α for which there exists at least one shortest path (represented by the labelling schemes) for all pairs of nodes. Thanks to Theorem 3.1, many graphs can be identified to support shortest path $k\text{-IRS}_\alpha$. For instance, grid, hypercube, complete graph, cycle, trees, outerplanar graphs, interval graphs, etc., have shortest path 1-IRS , and thus also shortest path 1-IRS_α . Families of graphs having shortest path $O(1)\text{-IRS}$ include torus, k -trees with constant k , planar graphs with a constant number of faces, etc. (see [16] for a complete state of the art).

For every graph G , we define

$$\text{IRS}_\alpha(G) = \min\{k \mid G \text{ has a shortest path } k\text{-IRS}_\alpha\}.$$

Note that the computation of $\text{IRS}(G)$ (for $\alpha = 1$) already involves several difficult optimizations. The decision problems “is $\text{IRS}(G) = 1$?” and “is $\text{IRS}(G) = 2$?” are NP-complete [5,7]. Hereafter, the value $\text{IRS}_\alpha(G)$ is termed *compactness* $_\alpha$ of G .

4.1. Comparison between compactness $_1$ and compactness $_\alpha$

By Theorem 3.1, we have $\text{IRS}_{\alpha+1}(G) \leq \text{IRS}_\alpha(G) \leq \dots \leq \text{IRS}(G)$. It is not a difficult exercise to check that there are graphs that support shortest path 1- IRS_2 , whereas they do not support shortest path 1- IRS_1 (for instance, consider the Petersen graph [17], or the wheel-graph [8]). The next result shows that the difference between the compactness of 1- and α -adaptive routing of a graph can be exponentially large.

Theorem 4.1. For every integer $\delta \geq 0$, there exists a graph G on $2^{\delta+3}$ nodes such that $\text{IRS}_1(G) \geq 2^\delta$ and $\text{IRS}_2(G) \leq 2\delta + 4$.

Proof. We use the construction given in [17] that shows that $\text{IRS}_1(G) \geq n/8$ for some n -node graphs with n a power of two. Here we recall their construction.

For a $p \times q$ Boolean matrix $M = (M_{i,j})$, let $G_M = (V_M, E_M)$ be the graph such that:

- (1) $V_M = \{v_1, \dots, v_p\} \cup \{a_1, \dots, a_q\} \cup \{b_1, \dots, b_q\}$;
- (2) $\{x, y\} \in E_M$ if and only if $(x = a_j$ and $y = b_j)$, $(x = b_j$ and $y = v_i$ and $M_{i,j} = 1)$, or $(x = a_j$ and $y = v_i$ and $M_{i,j} = 0)$.

We have $n = |V_M| = p + 2q$. Roughly speaking, G_M is a two-level graph. The first level consists of edges of type $\{a_j, b_j\}$, and the second one consists of v_i s (a stable) which are connected to a_j or b_j depending on whether $M_{i,j} = 0$ or 1. See Fig. 1 for an example.

For every Boolean matrix M , we denote by \bar{M} the matrix M with every bit complemented. Moreover, if $M = (XY)$, where X and Y are two matrices of same dimensions, we set $\chi(M) = (YX)$, which is the matrix obtained from M by exchanging the columns of X with those of Y . We consider a specific matrix M_δ , $\delta \geq 0$, defined by induction. The

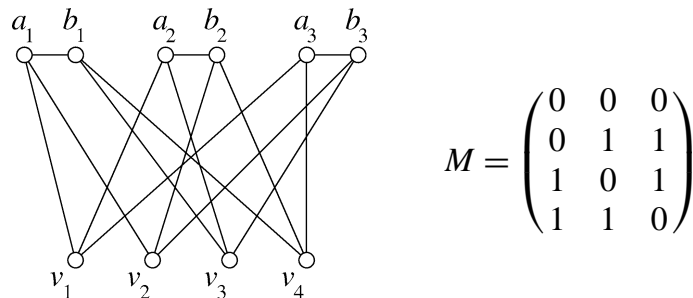


Fig. 1. A graph G_M .

construction of M_δ is summarized by Eq. (4.1).

$$M_0 = \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{pmatrix}, \quad \chi(M_0) = \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{pmatrix}, \quad M_{\delta+1} = \begin{pmatrix} M_\delta & \frac{M_\delta}{\chi(M_\delta)} \\ \chi(M_\delta) & \frac{M_\delta}{\chi(M_\delta)} \end{pmatrix}. \quad (4.1)$$

It is shown in [17] that $\text{IRS}(G_{M_\delta}) \geq 2^\delta$ (roughly speaking, whatever the labelling of the v_i s, the set $\mathcal{I}(a_j, b_j)$ must contain only a particular subset of the v_i s which is made to be hard to represent with intervals). In this case $p = 2^{\delta+2}$ and $q = 2^{\delta+1}$. Thus $n = 2^{\delta+3}$, proving the first part of Theorem 4.1.

Let us show that $\text{IRS}_2(G_{M_\delta}) \leq 2\delta + 4$ for every $\delta \geq 0$. For this purpose, it suffices to define a shortest path IRS_2 on G_{M_δ} , $(\mathcal{L}, \mathcal{I})$, such that for all the arcs $(u, v) \notin \{(a_j, b_j), (b_j, a_j)\}$, $|\mathcal{I}(u, v)| \leq 2\delta + 4$, thus composed of at most $2\delta + 4$ intervals. Indeed, by Theorem 3.1, such a labelling can be transformed into an IRS_2 with the same set of routes such that the edges $\{a_j, b_j\}$ consist of one interval. Therefore it would prove that $\text{IRS}_2(G_{M_\delta}) \leq 2\delta + 4$.

In this proof, we do not optimize the node-labelling, leaving a small space to improve the bound on $\text{IRS}_2(G_{M_\delta})$. Let us choose an arbitrary labelling \mathcal{L} . Since we do not care about the number of intervals on the edges $\{a_j, b_j\}$, let us define B_{M_δ} be the graph G_{M_δ} where the all the edges $\{a_j, b_j\}$ have been removed. First, remark that B_{M_δ} is a $2^{\delta+1}$ -regular bipartite graph. Clearly, B_M is isomorphic to every graph $B_{M'}$, where M' is obtained by complementing some columns of M (this morphism exchanges the roles playing by some a_j s and b_j s), or by permuting some columns (this morphism permutes some edges $\{a_j, b_j\}$). So, for the sake of simplicity, let us set B_δ to be the common graph isomorphic to B_{M_δ} , $B_{\chi(M_\delta)}$, etc. Let $V_1(B_\delta)$ to be the set of the first partition of nodes of B_δ , the a_j s and b_j s, and $V_2(B_\delta)$ as the nodes v_i s of B_δ .

Let us define by induction on δ , $(\mathcal{L}, \mathcal{I})$ on B_δ . $B_{\delta+1}$ consists of two copies of B_δ , B_1 and B_2 , with some extra edges connecting $V_1(B_1)$ to $V_2(B_2)$, and some edges connecting $V_1(B_2)$ to $V_2(B_1)$. Let ψ be the morphism between $V(B_1)$ and $V(B_2)$, and let ϕ_i be the morphism between $V_i(B_1)$ and $V_i(B_2)$ for $i = 1, 2$. For $\delta = 0$, we check that one can find \mathcal{I} such that $|\mathcal{I}(u, v)| \leq 4$ for all arcs (u, v) of B_0 . Let $k = \max |\mathcal{I}(u, v)|$, over all arcs (u, v) of B_δ . Since we do not care about \mathcal{L} , we consider $\mathcal{I}(u, v)$ as a subset of nodes rather than a subset of labels.

We first look at any node $v \in V_2(B_1)$. By induction, assume that $|\mathcal{I}(v, a)| \leq k$ for all $(v, a) \in E(B_1)$. We remark that if $(v, a) \in E(B_1)$, then $(v, \phi_1(a)) \in E(B_{\delta+1}) \setminus E(B_1)$. Setting $\mathcal{I}(v, \phi_1(a)) = \psi(\mathcal{I}(v, a))$ for all $(v, a) \in E(B_1)$, we are able to route from $v \in V_2(B_1)$ to all the nodes of $V(B_{\delta+1}) \setminus \{\phi_2(v)\}$. We add $\phi_2(v)$ to any arc incident of v leading to $\phi_2(v)$ by a shortest path. One can check that the routes are still the shortest, and since the edges (v, a) and $(v, \phi_1(a))$ are distinct, $|\mathcal{I}(v, a)| \leq k + 1$ for all $(v, a) \in E(B_{\delta+1})$.

Then, let us look at any node $a \in V_1(B_1)$. With a similar argument, we can set $\mathcal{I}(a, \phi_2(v)) = \psi(\mathcal{I}(a, v))$ for all $(v, a) \in E(B_1)$. We are able to route from $a \in V_1(B_1)$ to $V(B_{\delta+1}) \setminus \{\phi_1(v), \phi_1(\bar{a})\}$, where \bar{a} is the unique node of $V_1(B_1)$ such that $\{a, \bar{a}\}$ is an edge of $E(G_{M_\delta})$. We add $\phi_1(a)$ and $\phi_1(\bar{a})$ to any arc incident of a allowing shortest route from a . So, $|\mathcal{I}(a, v)| \leq k + 2$ for all $(a, v) \in E(B_{\delta+1})$.

The routing from any $v \in V_2(B_2)$ and any from $V_1(B_2)$ is defined similarly since the graph $B_{\delta+1}$ is the same if B_1 and B_2 are exchanged. In total, for every arc $(u, v) \in E(B_{\delta+1})$, we have $|\mathcal{I}(u, v)| \leq k + 2$, that is at most $2\delta + 4$ for B_δ .

We complete the proof by Theorem 3.1 applied on the edges $\{a_j, b_j\}$. \square

4.2. An upper bound for compactness $_\alpha$

In this section we show that compactness $_\alpha$ of a general n -node graph is not bounded for $\alpha > 1$. Note that for $\alpha = 1$, a tight lower bound exists. It has been shown in [18] that for every graph G , $\text{IRS}(G) \leq n/4 + o(n)$, whereas there exists a worst-case graph G_0 with $\text{IRS}(G_0) \geq n/4 - o(n)$. We first present a general upper bound:

Theorem 4.2. *For every n -node graph G and every $\alpha \leq \delta(G)$,*

$$\text{IRS}_\alpha(G) \leq \frac{1}{\alpha}(n - 1 - \delta(G)) + 1.$$

Proof. It suffices to consider any shortest path k -IRS $_1$ for G (for instance, choosing $k \leq n/4 + o(n)$), and to apply Theorem 3.3 remarking that all the arcs have non-empty labels.

4.3. A lower bound for compactness $_\alpha$

We will show that there exists some worst-case graphs with compactness $_\alpha$ at least $n/\alpha^{O(1)}$. Therefore this shows an asymptotic optimal lower bound for the compactness of shortest path IRS $_\alpha$ with constant α . It is quite complicated to build “by hand” small counter-example G with, for instance, $\text{IRS}_2(G) > 1$. Indeed, we need to argue for such G , that whatever is the node-labelling, whatever are the shortest paths, and mainly, whatever are the deflecting paths, one cannot code the routing table with one interval. The first counter-example with $\text{IRS}_2(G) > 1$ that we are able to build (we will not draw it here) has roughly 10^5 nodes. That is why we present in this paper an existential lower bound only, holding also for unbounded α . We will mainly use the fact that any shortest path α -adaptive routing table combined with a suitable selection function S implements a standard routing table ($\alpha = 1$). So, up to an additive term of $|S|$ one can lower bound the compactness of the α -adaptive routing table thanks to the $\Omega(n \log d)$ bit/node lower bound of [19].

For this purpose, let us present the graph $H_{p,\delta}$ introduced by [19], and defined inductively on p for all integers $p \geq 1$ and $\delta \geq 2$. Let $T_{h,\delta}^i$ be a complete δ -ary tree of height h whose all its leaves are labelled i . For $h = 0$, we set $T_{0,\delta}^i$ as a tree composed of a single node labelled i . For every $m \geq 2$, we define $T_{p,\delta,m}$ as the tree composed of m trees $T_{p-1,\delta}^1, T_{p-1,\delta}^2, \dots, T_{p-1,\delta}^m$, all connected by their root to a single node of degree m . This node is labelled $p + 1$ and forms the root of $T_{p,\delta,m}$. Note that for $m = \delta$, a $T_{p,\delta,\delta}$ tree is isomorphic to a complete δ -ary tree of height p , and thus has δ^p leaves.

$H_{p,\delta}$ has two distinguished subsets of nodes: $A_p = \{1, \dots, p\}$ and $B_p = \{1, \dots, \delta\}^p$ the set of all the words of length p on the alphabet $\{1, \dots, \delta\}$. $H_{1,\delta}$ is isomorphic to $K_{1,\delta}$, where $A_1 = \{1\}$ is reduced to the unique node of degree δ in $K_{1,\delta}$, and where $B_1 = \{1, \dots, \delta\}$ is the set of nodes of degree 1. The $H_{p+1,\delta}$ graph is composed of a copy of $H_{p,\delta}$, a copy of $T_{p,\delta,\delta}$, and of the set of nodes B_{p+1} , connected as follows:

- (1) Every node $u \in B_p$ is connected to the nodes $u_i \in B_{p+1}$, for every $i \in \{1, \dots, \delta\}$;
- (2) Every leaf of $T_{p,\delta,\delta}$ labelled i is connected to exactly δ nodes $u_i \in B_{p+1}$ such that no two leaves are connected to the same node of B_{p+1} (leaving some freedom in the connections).

The set A_{p+1} is composed of the set A_p of $H_{p,\delta}$, and of the root of $T_{p,\delta,\delta}$. See Fig. 2.

For every integer m such that $2 \leq m \leq \delta$, let us define the $H_{p,\delta,m}$ graph composed of a $H_{p,\delta}$ graph, a $T_{p,\delta,m}$ tree, and a set of $m\delta^p$ nodes, $B_p^m = \{1, \dots, \delta\}^p \times \{1, \dots, m\}$. The connections between B_p , B_p^m , and the leaves of $T_{p,\delta,m}$ are similar to the connections in a $H_{p+1,\delta}$ graph excepted that m may be smaller than δ (every $u \in B_p$ is connected to $u_i \in B_p^m$ for every $i \in \{1, \dots, m\}$). The $H_{p,\delta,m}$ graph is an induced subgraph of $H_{p+1,\delta}$. Let us denote by A_p^m the set of modes composed of A_p and of the root of the $T_{p,\delta,m}$ tree.

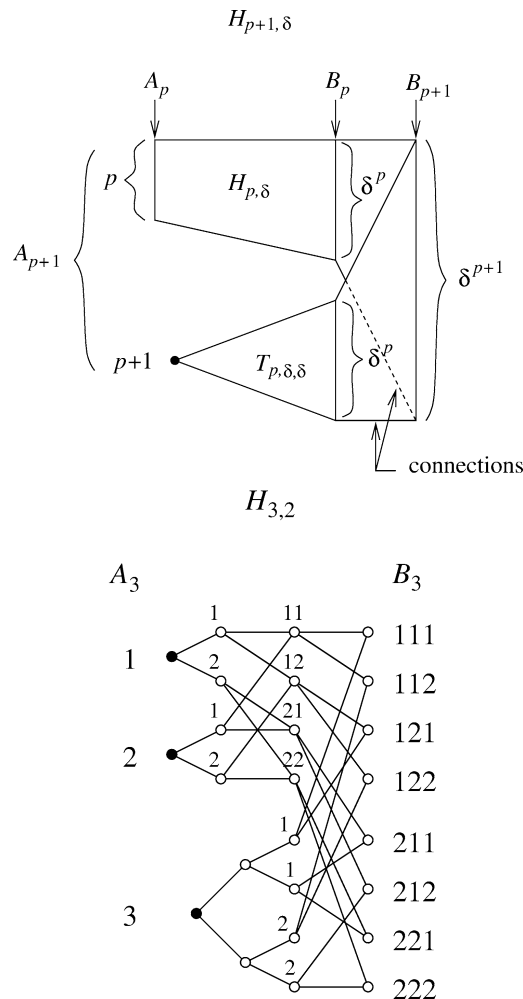


Fig. 2. The recursive construction of the $H_{p+1,\delta}$ graph, and $H_{3,2}$.

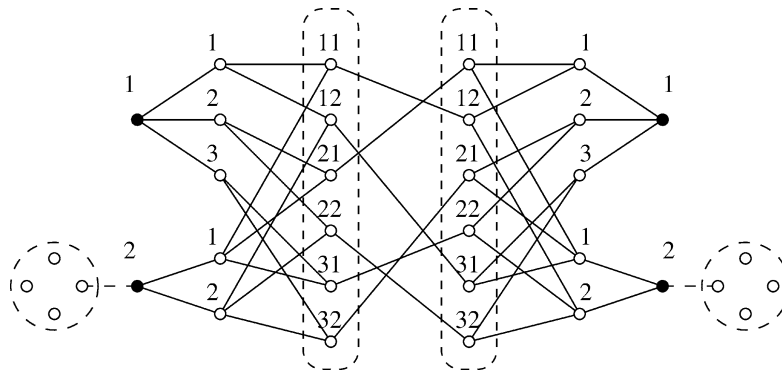


Fig. 3. A graph $G_\pi \in \mathcal{H}_{1,3,2}$, and its modification H_0 : the dashed sets induced a clique in H_0 and are missing from G_π .

Lemma 4.3. For all integers $p \geq 1$, and $\delta, m \geq 2$, $H_{p,\delta,m}$ has at most $2(m+2)\delta^p$ nodes, and $\deg(x) = 2$ if $x \in B_p^m$, $\deg(x) = m$ if x is the root of $T_{p,\delta,m}$, and $\deg(x) \geq \delta$ otherwise.

Given a permutation π of B_p^m , let us denote by G_π the graph composed of two copies of $H_{p,\delta,m}$ whose B_p^m sets are connected by the perfect matching defined by π (see Fig. 3, for an example). Let $\mathcal{H}_{p,\delta,m}$ denote the family composed of all the G_π graphs, for all permutations π of B_p^m . For each $G \in \mathcal{H}_{p,\delta,m}$, we denote by $A(G)$ (respectively $B(G)$) the set of nodes composed of both sets A_p^m (respectively B_p^m) of each copy of $H_{p,\delta,m}$ forming G . The nodes of $A(G)$ are drawn in black on Fig. 3.

In [19], it is shown the following important lemma:

Lemma 4.4 (Gavaille and Perennes [19]). For all integers $p, \delta, m \geq 2$, $m \leq \delta$, and such that $\delta^p \rightarrow +\infty$, there exists a graph $G_0 \in \mathcal{H}_{p,\delta,m}$ such that every shortest path routing table on G_0 has a size of M bits⁵ for a node of $A(G_0)$ such that

$$M \geq \frac{m\delta^p}{2(p+1)} \log(m\delta^p) - O\left(\frac{m\delta^p}{p}\right).$$

This result is based on the uniqueness of the shortest paths between the nodes of $A(G_0)$ and the nodes of $B(G_0)$. In order to prove our result, one transform G_0 into a new graph H_0 such that $\delta(H_0) \geq \delta$, and such that Lemma 4.4 holds for H_0 as well. It consists on connecting all the nodes of B_p^m by a clique in each copy of $H_{p,\delta,m}$ (so making the degree of the nodes of $B(G_0)$ larger than δ in H_0). Then, for the root of both $T_{p,\delta,m}$ trees, we add a clique of $\delta + 1$ nodes and select from them $\delta - m$ nodes that we connect to the root (so making the degree of nodes at least δ , and exactly δ for all the nodes of $A(G_0)$ in H_0). See Fig. 3. In H_0 , the shortest paths between $A(G_0)$ and $B(G_0)$ are not modified, and has $2(\delta + 1)$ more nodes than G_0 .

We are now ready to prove a lower bound on compactness $_\alpha$ of n -node graphs.

⁵ We assume that all logarithms are in base two.

Theorem 4.5. *There exist a constant $c \leq 31$ such that for every n large enough, and for every integer $\alpha \leq (n/18)^{1/(2c)}$, there exists a graph H_0 with at most n nodes such that*

$$\text{IRS}_\alpha(H_0) > \frac{1}{2790} \cdot \frac{n}{\alpha^c}.$$

Proof. From Theorem 4.1, for every n , there exists a graph G with $2^{t+3} \leq n$ nodes and such that $\text{IRS}_1(G) \geq 2^t > n/16$ for $t = \lfloor \log n \rfloor - 3$. Thus the result is true for $\alpha = 1$.

Assume $\alpha \geq 2$. Let us fix $c = 31$, and let $\delta = \alpha^c$. We consider the graph H_0 , the modified graph $G_0 \in \mathcal{H}_{p,\delta,m}$, for some parameters $p, \delta, m \geq 2$ such that $m \leq \delta$ and $\delta^p \rightarrow +\infty$. Let N denote the number of nodes of H_0 . We will fix later the values of p, δ, m as a function of n in order to prove that $N \leq n$. Let $x \in A(G_0)$ be a node of H_0 for which the size of any shortest path routing table is of size at least M (bound given by Lemma 4.4). Note that by construction of H_0 , $\deg(x) = \delta$.

Consider on H_0 any shortest path k - IRS_α , $(\mathcal{L}, \mathcal{I})$. This α -adaptive IRS is an implementation in x of a particular shortest path α -adaptive routing table. This implementation can be done in x with at most $\lceil \delta \log \binom{N}{2k} \rceil$ bits. Indeed, for the δ output ports of x it suffices to store at most k intervals of labels. There is at most $\binom{N}{2k}$ ways to choose k sub-intervals of $[1, N]$. So, a total of $\lceil \delta \log \binom{N}{2k} \rceil$ bits for x , remarking that a sequence of p integers taken from $\{1, \dots, q\}$ can be coded on $\lceil \log(q^p) \rceil$ bits since there are q^p such sequences.

Now, it is easy to transform any shortest path α -adaptive routing tables into a shortest path 1-adaptive routing table, i.e., a standard routing table, adding $\lceil N \log \alpha \rceil$ extra bits per node: for each destination label we specify the output port leading to a shortest path, and there are exactly α possible output ports. Thus, H_0 has a shortest path routing table in x of size at most

$$\left\lceil \delta \log \binom{N}{2k} \right\rceil + \lceil N \log \alpha \rceil$$

using an implementation of the deterministic version of $(\mathcal{L}, \mathcal{I})$. From Lemma 4.4, it turns out for x that:

$$\delta \log \binom{N}{2k} + N \log \alpha + 2 > M. \quad (4.2)$$

We have to prove that H_0 has at most n nodes and that $k \geq n/\alpha^{O(1)}$. Let us fix now p, δ, m , and let us prove that:

$$N \leq n < 4(m + 3 + o(1))\delta^p. \quad (4.3)$$

Let p be the largest integer such that $n \geq 16\delta^p + 2(\delta + 1)$. Clearly, $\delta^p \rightarrow +\infty$ as $n \rightarrow +\infty$. Let $m = \lfloor (n - 2(\delta + 1))/(4\delta^p) \rfloor - 2$. Let us show that p, δ , and m are all greater than 2.

First, $\delta \geq 2$ because $\delta = \alpha^c$, and $\alpha, c \geq 2$. For $p, \alpha \leq (n/18)^{1/(2c)}$ implies $18(\alpha^c)^2 \leq n$, i.e., $n \geq 18\delta^2$. But $18\delta^2 \geq 16\delta^2 + 2(\delta + 1)$ for $\delta \geq 2$, hence the equation $n \geq 16\delta^p + 2(\delta + 1)$ has a solution for $p \geq 2$. For m , since $n \geq 16\delta^p + 2(\delta + 1)$, then $(n - 2(\delta + 1))/(4\delta^p) \geq 4$, and thus $\lfloor (n - 2(\delta + 1))/(4\delta^p) \rfloor - 2 \geq 2$, proving $m \geq 2$.

Let $X = 4(m + 2)\delta^p + 2(\delta + 1)$, and let $Y = n - 2(\delta + 1)$. Note that $m + 2 = \lfloor Y/(4\delta^p) \rfloor$. We have:

$$\begin{aligned} X &= 4\delta^p \left\lfloor \frac{Y}{4\delta^p} \right\rfloor + 2(\delta + 1) \\ &= Y - (Y \bmod 4\delta^p) + 2(\delta + 1) \\ &= n - (Y \bmod 4\delta^p). \end{aligned}$$

Therefore,

$$X \leq n < X + 4\delta^p. \tag{4.4}$$

From Lemma 4.3, the number of nodes of G_0 is at most $2 \cdot 2(m + 2)\delta^p$, and thus the number of nodes of H_0 is $N \leq 4(m + 2)\delta^p + 2(\delta + 1)$, i.e., $N \leq X$. By Eq. (4.4) we have proved that H_0 has at most n nodes, and more precisely that:

$$\begin{aligned} N &\leq n < 4(m + 2)\delta^p + 4\delta^p + 2(\delta + 1) \\ &< 4(m + 3 + o(1))\delta^p \end{aligned}$$

remarking that $2(\delta + 1) = o(\delta^p)$, and proving therefore Eq. (4.3).

In Eq. (4.2), we bound

$$\binom{N}{2k} \leq \binom{n}{2k} \leq \left(\frac{en}{2k}\right)^{2k} \leq t^{en/t},$$

where $t = en/(2k)$. Eq. (4.2) becomes (using $\delta = \alpha^c$, $N \leq n$, and $M \rightarrow +\infty$)

$$\delta \log(t^{en/t}) + n \log \alpha \geq M \tag{4.5}$$

$$\implies \alpha^c en \frac{\log t}{t} + n \log \alpha \geq M \tag{4.6}$$

$$\implies n \left(\frac{\alpha^c e \log t}{t} + \log \alpha \right) \geq M \tag{4.7}$$

$$\implies n\beta \geq M, \tag{4.8}$$

where $\beta = (\alpha^c e \log t)/t + \log \alpha$. From Lemma 4.4 we have a lower bound on M , and plugging in Eq. (4.8) the upper bound on n of Eq. (4.3), we obtain that:

$$\begin{aligned} 4(m + 3 + o(1))\delta^p \beta &> \frac{m\delta^p}{2(p + 1)} \log(m\delta^p) - O\left(\frac{m\delta^p}{p}\right) \\ \implies \beta &> \frac{m \log(m\delta^p)}{8(p + 1)(m + 3 + o(1))} \end{aligned}$$

neglecting the second order term $O(m\delta^p/p)$. We remark that $p, m \geq 2$, thus

$$\begin{aligned} \frac{m \log(m\delta^p)}{8(p + 1)(m + 3 + o(1))} &\geq \frac{2 \log(2\delta^p)}{8(p + 1)(5 + o(1))} > \frac{p \log \delta}{4(p + 1)(5 + o(1))} \\ &> \frac{2 \log \delta}{4 \cdot 3(5 + o(1))} > \frac{\log \delta}{30 + o(1)}. \end{aligned}$$

Replacing β and δ , we obtain:

$$\frac{\alpha^c e \log t}{t} + \log \alpha \geq \frac{c \log \alpha}{30 + o(1)} \quad (4.9)$$

$$\implies \frac{\alpha^c \log t}{t} \geq \frac{1}{e} \left(\frac{c}{30 + o(1)} - 1 \right) \log \alpha \quad (4.10)$$

$$\implies \frac{\alpha^c \log t}{t} \geq \gamma \log \alpha, \quad (4.11)$$

where $\gamma = (c/(30 + o(1)) - 1)/e$. Because $c = 31$, we have $\gamma > 0$ (for n large enough), and since $\log \alpha \geq 1$, it follows that:

$$(4.11) \implies \alpha^c \geq \frac{\gamma t}{\log t} \quad (4.12)$$

$$\implies c \log \alpha \geq \log(\gamma t) - \log \log t. \quad (4.13)$$

Note that $k \geq en/(2t)$. We consider two cases. If $t \leq 2^{35}$, then $k \geq en/2^{36}$. Since $2^{36}/e > 2790\alpha^c$ for $c = 31$ and $\alpha \geq 2$, it follows in this case that:

$$k > \frac{n}{2790\alpha^c}.$$

If $t > 2^{35}$, then we check that (as $\gamma \rightarrow 0.001226\dots$ as $n \rightarrow +\infty$):

$$\log(\gamma t) - \log \log t > \frac{2}{3} \log t$$

thus by Eq. (4.13) $c \log \alpha > \frac{2}{3} \log t$. Bounding $\log t < (3c \log \alpha)/2$, Eq. (4.11) becomes:

$$\frac{\alpha^c 3c \log \alpha}{2t} \geq \gamma \log \alpha$$

$$\implies t \leq \frac{3c}{2\gamma} \alpha^c$$

$$\implies k \geq \frac{e\gamma}{3c\alpha^c} n = \left(\frac{c/(30 + o(1)) - 1}{3c} \cdot \frac{1}{\alpha^c} \right) n$$

$$> \frac{1}{90c\alpha^c} \cdot n \quad (\text{as } n \rightarrow +\infty)$$

$$> \frac{1}{2790} \cdot \frac{n}{\alpha^c}$$

that completes the proof. \square

5. Conclusion

We showed that α -adaptive routing tables on n -node graphs, that are routing tables mapping each destination on exactly α directions, have compactness at most n/α (i.e., require n/α intervals of destination labels per link), computable in polynomial time. We proved also that, if at least one shortest path must be represented, there are n -node graphs for which every α -adaptive routing table has compactness larger than $n/\alpha^{O(1)}$.

In the other side, it is known that if *all* the shortest paths must be represented, then such routing tables require compactness $n/4$ for some worst-case graphs. Therefore, it would be interesting to study the compactness of β -shortest path α -adaptive routing tables, a natural extension of shortest path α -adaptive routing tables, that map each destination on α directions and whose at least β must be on a shortest path. The present paper concerns $\beta = 1$.

We stress also that our $n/\alpha^{O(1)}$ -lower bound is not a serious obstacle for the study of graphs having small compactness, even for $\alpha = 2$. Indeed, due to some large constants in this existential lower bound, the smallest example of graphs we can prove by Theorem 4.5 to have a compactness greater than 1 must have more than 2^{42} nodes. It suggests that the class of graphs supporting shortest path 2-adaptive routing tables is rather large, and it would be interesting to develop this study to various class of concrete networks.

References

- [1] Y. Afek, E. Gafni, M. Ricklin, Upper and lower bounds for routing schemes in dynamic networks, in: 30th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 1989, pp. 370–375.
- [2] B. Awerbuch, A. Bar-Noy, N. Linial, D. Peleg, Compact distributed data structures for adaptive routing, in: 21st Annual ACM Symposium on Theory of Computing (STOC), 1989, pp. 479–489.
- [3] E.M. Baker, R.B. Tan, J. van Leeuwen, Prefix routing schemes in dynamic networks, Technical Report RUU-CS-90-10, Dept. of Computer Science, Utrecht University, 1990.
- [4] T. Eilam, C. Gavoille, D. Peleg, Compact routing schemes with low stretch factor, in: 17th Annual ACM Symposium on Principles of Distributed Computing (PODC), 1998, pp. 11–20.
- [5] T. Eilam, S. Moran, S. Zaks, The complexity of the characterization of networks supporting shortest-path interval routing, in: 4th International Colloquium on Structural Information & Communication Complexity (SIROCCO), Carleton Scientific, 1997, pp. 99–111.
- [6] T. Eilam, S. Moran, S. Zaks, Lower bounds for linear interval routing, *Networks* 34 (1999) 37–46.
- [7] M. Flammini, On the hardness of devising interval routing schemes, *Parallel Processing Lett.* 7 (1997) 39–47.
- [8] P. Fraigniaud, C. Gavoille, Interval routing schemes, *Algorithmica* 21 (1998) 155–182.
- [9] M. Flammini, G. Gambosi, On devising Boolean routing schemes, *Theoret. Comput. Sci.* 186 (1997) 171–198.
- [10] M. Flammini, G. Gambosi, U. Nanni, R.B. Tan, Characterization results of all shortest paths interval routing schemes, in: 5th International Colloquium on Structural Information & Communication Complexity (SIROCCO), Carleton Scientific, July 1998, pp. 201–213.
- [11] M. Flammini, G. Gambosi, S. Salomone, Interval labeling schemes for chordal rings, in: 1st International Colloquium on Structural Information & Communication Complexity (SIROCCO), Carleton Univ. Press, May 1994, pp. 111–124.
- [12] M. Flammini, J. van Leeuwen, A. Marchetti-Spaccamela, The complexity of interval routing on random graphs, *Comput. J.* 41 (1998) 16–25.
- [13] G.N. Frederickson, R. Janardan, Separator-based strategies for efficient message routing, in: 27th Symposium on Foundations of Computer Science (FOCS), May 1986, pp. 428–437.
- [14] G.N. Frederickson, R. Janardan, Space-efficient message routing in c -decomposable networks, *SIAM J. Comput.* 19 (1990) 164–181.
- [15] C. Gavoille, On the dilation of interval routing, *Comput. J.* 43 (2000) 1–7.
- [16] C. Gavoille, A survey on interval routing, *Theoret. Comput. Sci.* 245 (2000) 217–253.
- [17] C. Gavoille, E. Guévremont, Worst case bounds for shortest path interval routing, *J. Algorithms* 27 (1998) 1–25.
- [18] C. Gavoille, D. Peleg, The compactness of interval routing, *SIAM J. Discrete Math.* 12 (1999) 459–473.

- [19] C. Gavoille, S. Pérennès, Memory requirement for routing in distributed networks, in: 15th Annual ACM Symposium on Principles of Distributed Computing (PODC), 1996, pp. 125–133.
- [20] E. Kranakis, D. Krizanc, Lower bounds for compact routing, in: 13th Annual Symposium on Theoretical Aspects of Computer Science (STACS), in: Lecture Notes in Computer Sci., Vol. 1046, Springer, Berlin, 1996, pp. 529–540.
- [21] E. Kranakis, D. Krizanc, J. Urrutia, Compact routing and shortest path information, in: 2nd International Colloquium on Structural Information & Communication Complexity (SIROCCO), Carleton Univ. Press, 1995, pp. 101–112.
- [22] R. Kráľovič, P. Ružička, D. Štefankovič, The complexity of shortest path and dilation bounded interval routing, *Theoret. Comput. Sci.* 234 (2000) 85–107.
- [23] P. Loh, J. Wenge, Adaptive, fault-tolerant, deadlock-free and livelock-free interval routing in mesh networks, in: 2nd IEEE International Conference on Algorithms & Architectures for Parallel Processing (ICAPP), 1996, pp. 348–355.
- [24] L. Narayanan, J. Opatrny, Compact routing on chordal rings of degree four, in: 4th International Colloquium on Structural Information & Communication Complexity (SIROCCO), Carleton Scientific, 1997, pp. 125–137.
- [25] D. Peleg, E. Upfal, A trade-off between space and efficiency for routing tables, *J. ACM* 36 (1989) 510–530.
- [26] P. Ružička, D. Štefankovič, On the complexity of multi-dimensional interval routing schemes, *Theoret. Comput. Sci.* 245 (1999).
- [27] N. Santoro, R. Khatib, Labelling and implicit routing in networks, *Comput. J.* 28 (1985) 5–8.
- [28] S.S.H. Tse, F.C.M. Lau, An optimal lower bound for interval routing in general networks, in: 4th International Colloquium on Structural Information & Communication Complexity (SIROCCO), Carleton Scientific, 1997, pp. 112–124.
- [29] J. van Leeuwen, R.B. Tan, Computer networks with compact routing tables, in: G. Rozenberg, A. Salomaa (Eds.), *The Book of L*, Springer, Berlin, 1986, pp. 259–273.
- [30] J. van Leeuwen, R.B. Tan, Interval routing, *Comput. J.* 30 (1987) 298–307.