Event Extent Estimation*

Marcin Bienkowski[1], Leszek Gąsieniec[2], Marek Klonowski[3], Miroslaw Korzeniowski[3], and Stefan Schmid[4]

[1] Institute of Computer Science, University of Wrocław, Poland
[2] Department of Computer Science, The University of Liverpool, UK
[3] Wroclaw University of Technology, Poland
[4] Deutsche Telekom Laboratories / TU Berlin, Germany

Abstract. This paper studies local-control strategies to estimate the size of a certain event affecting an arbitrary connected subset of nodes in a network. For example, our algorithms allow nodes in a peer-to-peer system to explore the remaining connected components after a Denial-of-Service attack, or nodes in a sensor network to assess the magnitude of a certain environmental event. In our model, each node can keep some extra information about its neighborhood computed during the deployment phase of the network. On the arrival of the event, the goal of the active nodes is to learn the network topology induced by the event, without the help of the remaining nodes. This paper studies the tradeoffs between message and time complexity of possible distributed solutions.

1 Introduction

This paper attends to the problem of how nodes in a network can efficiently learn about (or deal with) the effects of a certain event. We assume that before the event takes place, e.g., during network deployment, nodes have sufficient time to perform certain pre-computations. Then, at some unknown time point, an event activates an arbitrary subset of nodes. We investigate distributed algorithms that allow the activated nodes to gather necessary information about the event in an efficient, cooperative manner — without the help of the remaining nodes. In particular, our algorithms allow these nodes to learn the topology induced by affected nodes.

For example, consider a peer-to-peer network which is hit by a virus spreading along the topology, or which is under a denial-of-service attack. After the attack, the goal of the surviving peers is to learn about the remaining functional peers in their respective connected component, e.g., in order to trigger a best-effort recovery of both data and topology.

Natural disaster detection is another motivation for our model. Today, many observation systems are used to monitor a certain endangered area and to warn about floods, fires, or earthquakes in time, to prevent larger damage. Besides

* Supported by MNiSW grant number N206 257335, 2008-2011, PBZ/MNiSW/07/2006/46.

global solutions like satellite systems, there is a trend towards distributed monitoring with wireless sensor nodes that are distributed in space (see e.g. [3] for detecting seismic events). Our algorithms can be useful in this context: they allow for computing the number of other nodes observing a certain phenomenon.

In our work, we assume that the nodes which are not activated by the event are inactive (or faulty) and *cannot* participate in the communication nor in the computation. This is clear in our peer-to-peer example, as the nodes attacked by the virus may simply be down or crashed. In order to motivate this assumption in a sensor network, consider the situation where nodes are in an energy-parsimonious sleeping mode and only wake up in case of some external physical influence, triggered by the event.

1.1 Model

This work assumes an arbitrary undirected network or graph $G = (V, E)$ of $n = |V|$ nodes and $m = |E|$ edges. We consider a synchronous model where algorithms proceed in rounds and where the event happens at a globally unique time t_0. At this time, a subset of the nodes become *active*, the remaining ones are *inactive*. The goal of the active nodes is to find out about other active nodes.

In our model, we assume that the nodes have sufficient time to perform arbitrary *preprocessing operations* at times $t < t_0$, e.g., to learn the topology. In particular, we assume that nodes choose unique IDs from $\{1, ..., n\}$ in the preprocessing stage. Our algorithms hence work in two stages, where in the first "offline" stage nodes do *preprocessing*, and in the second "event" stage, *the actual event is explored.* During our analysis, we are mainly interested in the complexities of the second stage. A novelty of the problem studied in this paper comes from this division.

Henceforth, by $V' \subseteq V$ we denote the subset of active nodes and by G' subgraph of G induced by V'. We note that G' is not necessarily connected; we call connected components of G' *active components.* Let $s = |V'|$ be the number of active nodes, and let δ denote the *active diameter*, i.e. the maximum diameter of an active component.[1] We aim at designing algorithms in which all active nodes will learn about their active components; in particular, nodes will learn the size of their components. Observe that usually, once there is at least one node $v_{G'} \in G'$ with this knowledge in a component G', $v_{G'}$ can subsequently inform all other nodes in G' along a corresponding spanning tree. Hence, in the following, we sometimes concentrate on this case only, given that sending this information along the spanning tree does not change the asymptotic complexity.

We strive to optimize two criteria: First, our algorithms should have a good reaction time (*time complexity*). We measure the time (i.e., the number of rounds) until all active nodes know the IDs of the other active nodes in their active component. In each round, the nodes can perform arbitrary local computations and communicate with their neighbors in the network.

[1] Observe that the active diameter can be much larger than the diameter. For instance, in a $\sqrt{n} \times \sqrt{n}$-grid, the active diameter can be linear in n.

Second, we want the algorithm to send as few messages as possible (*message complexity*). We assume that in a round, each node can send a (different) message to each neighbor. Each such message costs a unit of energy, independently of the neighbor being active or not. We are interested in the total number of messages sent by all active nodes and our focus is on feasibility, so — in the lines of the $\mathcal{LOCAL}$ model [11] — we do not restrict the size of messages. Notice however that at time t_0 nodes do not know which of their neighbors are active and which are not. They have to deduce this knowledge from the protocol and the communication pattern in subsequent rounds.

1.2 Related Work

There is a considerable scientific interest in distributed monitoring and alarming systems. For instance, there are approaches to detect the boundaries of a toxic leach [6] or monitoring mechanisms to defend against Internet worms [7]. Our work is also related to literature on robustness of overlay networks, where nodes need to reorganize after an attack in an efficient manner [5].

Our paper belongs to the field of local algorithms where computations are performed by repeated interactions of nodes with their neighbors. Our problem formulation is reminiscent of classic problems such as leader election, and indeed, most of our algorithms implicitly solve this problem. However, we can identify at least two interesting and new aspects of our model. First, our algorithms adapt themselves to the environment, in the sense that the runtime and the number of messages is smaller for fewer active nodes. Several papers recently investigated local solutions for global problems for which the runtime depends on the concrete problem input [2], rather than considering the worst-case over all possible inputs: if in a special instance of a problem the input behaves well, a solution can be computed quickly. Second, we assume that only active nodes can participate in the computations, while the number of active nodes is not known in advance. Thus, preprocessing the graph appears to be of limited use, as any precomputed coordination points or infrastructure may not be active later. This poses a higher demand on efficient coordination primitives during runtime.

It is interesting to compare our work to the disaster disclosure problem introduced by Mans et al. [9] — the closest paper to our work. In [9], it is assumed that nodes that did not sense an event can participate in the coordinated exploration of a disaster. For example, this so-called "on-duty model" is meaningful in sensor networks where all nodes are regularly "online" in order to exchange status updates and can start collaborating on demand. In contrast, in the model studied in our work, we try to capture a scenario where certain nodes are not available during the distributed computations after the event; this so-called "off-duty model" has been stated as an open research direction in [9]. Despite the similar nature of the on-duty and the off-duty model, the two problems exhibit a different structure. This is due to the fact that in the on-duty model, nodes can heavily rely on computations done during the pre-processing phase. Indeed, the approach taken in [9] relies on a hierarchy of "leaders" elected during deployment. However, in the off-duty model the use of such local coordinators is out of question: they may be unavailable during the event exploration phase. This has implications both on the design of algorithms as well as the achievable performance.

1.3 Our Contribution

This paper initiates the study of a new model for estimating the size of an event, where the affected nodes need to coordinate without the help of the remaining nodes. The task is non-trivial, because nodes do not know the structure of the active component in advance. On the other hand, we aim to make the runtime and the number of messages dependent on the size of the affected node set and not on the size of the network. In this work, we study the tradeoffs between the runtime and the number of messages needed to solve the task.

We begin our investigations with a case study of one-hop networks (i.e., complete graphs, see Section 2). We describe a natural and simple algorithm family Group, where nodes organize in groups of increasing sizes. Subsequently, we describe a randomized Las Vegas approach Rand where nodes seek to "guess" the number of active nodes in order to coordinate. In expectation, the algorithm requires time $O(\log(n/s))$ and $O(n)$ messages. In Section 3, we complement our insights on clique algorithms with lower bounds. We show that our problem requires an understanding of the intriguing interplay of time and message complexity, and use Turán's theorem together with the concept of primary schedules to show that the product of time and message complexity of any deterministic algorithm is at least $\Omega(n \log \log n)$. This proves that our results for clique are optimal up to logarithmic factors.

Section 4 proceeds to examine arbitrary topologies. We first discuss general graph searching techniques combined with waiting techniques, and then introduce a preprocessing scheme, which allows each node to locally and efficiently detect its active neighbors. The complexity of this scheme depends on a graph's arboricity, i.e., the forest cover size. Given this construction, we present the MinID algorithm with time complexity $O(s \log s)$ and message complexity $O(\alpha s \log s)$, i.e., its performance only depends on the event size s and the graph arboricity α.

Motivated by our results for general graphs, we tackle the important case of planar graphs (Section 5), which are known to have constant arboricity. There the time and message complexities of our algorithm MinID are optimal up to an $O(\log s)$ factor. We also show that the message complexity can be improved: using the Planar Separator Theorem, we construct a graph decomposition approach resulting in an algorithm family k-Sep which yields optimal message complexity and non-trivial time.

2 The Clique

To start our scrutinies and to get acquainted with the model, we consider the case of one-hop networks. Note that in such networks, active nodes form only one connected component. Clearly, here a broadcast by all active nodes is already time-optimal, but it requires $s \cdot (n-1)$ messages. We therefore study a natural class of algorithms called GROUP where nodes organize themselves recursively into groups. Subsequently, we extend our scope to randomized algorithms and study an efficient algorithm RAND which tries to estimate the active set cardinality.

2.1 The Algorithm GROUP

The algorithm GROUP uses an integer parameter $k \in \{2, \ldots n\}$. For simplicity of description, we assume that $\log_k n$ is an integer as well. We further assume that node identifiers are written as $(\log_k n)$-digit strings, where each digit is an integer between 0 and $k-1$. This implicitly creates the following hierarchical partitioning of all nodes into clusters. The topmost cluster (on level $\log_k n$) contains all nodes and is divided into k clusters, each consisting of n/k nodes, where cluster i contains all the nodes whose first digit is equal to i. Each of these clusters is also partitioned into k clusters on the basis of the second digit of identifiers. This partitioning proceeds to leafs, which are 0^{th} level clusters, each containing a single node. We call a cluster *active* if it contains at least one active node.

GROUP works in $\log_k n$ phases, where in the i^{th} phase we are dealing with clusters on level i. We inductively require that at the beginning of the phase, there is a *leader* in each i^{th} level active cluster; the leader knows all the active nodes within its cluster and all these nodes know the leader. Thus, at the end of the phase $\log_k n$, all nodes constitute a single cluster and its leader knows the set of all active nodes.

To study what happens in the i^{th} phase, we concentrate on a single $(i+1)^{th}$ level cluster A. (The procedure is performed in all such clusters independently in parallel.) A consists of k i^{th} level clusters, denoted $A_1, A_2, \ldots, A_k$, which will merge in this phase. Moreover the leader of A is the node with smallest ID amongst leaders of active clusters A_i. The merging procedure comes in two flavors: parallel (PAR) and sequential (SEQ).

In the PAR variant, a phase lasts two rounds. In the first round, the leaders of clusters A_j broadcast a "hello message" to all nodes from these clusters. All the active nodes among them answer with a message to a leader with the smallest identifier, and this node becomes a leader of the $(i+1)^{th}$ level cluster.

In the SEQ variant, the phase lasts for $k+1$ rounds. For $j \leq k$, in the j^{th} round, the leader of cluster A_j broadcasts a hello message to all nodes from A, provided such a message was not sent already. The nodes which hear the message, answer in the next round, and the leader that transmitted the broadcast becomes a leader of the next higher level cluster, the $(i+1)^{th}$ level cluster.

Theorem 1. *Fix any $1 \leq \ell \leq \log n$. Then there exists a parameterization of the algorithm* GROUP *which solves the problem using $O(\ell)$ rounds and $O(n \cdot \ell \cdot \min\{n^{1/\ell}, s\})$ messages and there exists a parameterization, which solves it using $O(\ell \cdot n^{1/\ell})$ rounds and $O(n \cdot \ell)$ messages.*

Proof. We measure the time and message complexity of the GROUP algorithm using variants PAR and SEQ for all levels and choosing $k = n^{1/\ell}$, i.e. $\log_k n = \ell$.

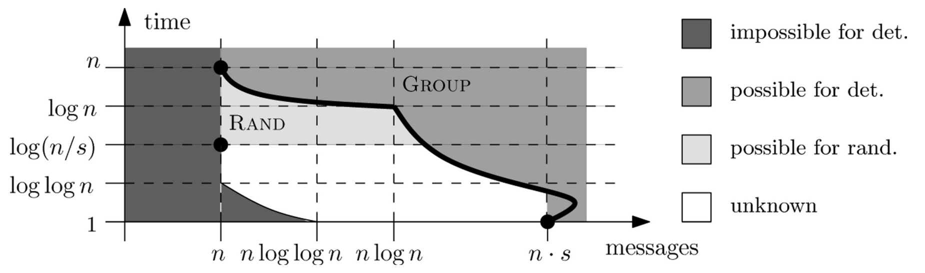

Fig. 1. Performance comparison of different clique algorithms. Black dots represent different algorithms and the black line indicates the trade-offs achievable with the Group algorithm. Darker regions represent infeasibility results.

Clearly, in the Par variant, the algorithm needs $2 \cdot \log_k n = O(\ell)$ rounds. As for the message complexity, we look at a single phase i. In the first round of this phase, each node gets a hello message from at most $\min\{k, s\}$ leaders and sends a reply. Therefore, the algorithm uses at most $2 \cdot \min\{k, s\} \cdot n \cdot \log_k n = O(n \cdot \ell \cdot \min\,\{n^{1/\ell}, s\})$ messages. The variant Seq requires $(k + 1) \cdot \log_k n = O(\ell \cdot n^{1/\ell})$ rounds. Then, in a single phase, each node gets at most one hello message and answers at most once. Thus, the total number of messages transmitted is at most $2 \cdot n \cdot \log_k n = O(n \cdot \ell)$. □

We observe that the best time×message-product is achieved for $\ell = \log n$, in which case Group solves the problem in time $O(\log n)$ using $O(n \log n)$ messages. Note that Group can be regarded as a generalization of two graph search techniques: the extreme cases require 1 round or n messages and correspond to parallel or sequential flooding of the graph by active nodes. These trade-offs are depicted in Figure 1.

2.2 Randomized Cardinality Guessing

In this section, we extend our analysis to randomized approaches. The idea behind our algorithm Rand is to approximately "guess" the number of active nodes. For succinctness of the description, we assume that n is a power of 2. Rand proceeds in $\log n + 1$ phases, numbered from 0 to $\log n$, each consisting of two rounds. In the first round of the i^{th} phase, each node — with probability $p_i = 2^i/n$ — broadcasts a hello message to all other nodes. In the second round active nodes reply. After a phase with a broadcast, the algorithm terminates. The Las Vegas algorithm Rand always solves the problem, as in phase $\log n$ each node performs a broadcast (with probability 1).

Theorem 2. *On expectation,* Rand *terminates in* $O(\log(n/s))$ *rounds and uses* $O(n)$ *messages.*

Proof. Let $k = \lceil \log(n/s) \rceil$, i.e., $2^{k-1} < n/s \le 2^k$. Then, phase k is the first phase in which the broadcast probability of each node reaches $1/s$, i.e., $p_k \in$

$[1/s, 2/s)$. It is sufficient to show that the algorithm makes its first broadcast around phase k, and, in expectation, it makes a constant number of broadcasts.

Let $\mathcal{E}_i$ denote an event that RAND does not finish till phase i (inclusive), i.e., there was no broadcast in phases $1, 2, \ldots, i$. Let τ be a random variable denoting the number of phases of RAND. Then, $\mathbf{E}[\tau] = \sum_{i=1}^{\log n} \Pr[\tau \geq i] = \sum_{i=0}^{\log n - 1} \Pr[\mathcal{E}_i] \leq \sum_{i=0}^{k-1} 1 + \sum_{j=0}^{\log n - k - 1} \Pr[\mathcal{E}_{k+j}]$.

To bound the last term, we first observe that the necessary condition for $\mathcal{E}_i$ is that no node transmits in phase i. Hence, $\Pr[\mathcal{E}_i] \leq (1 - p_i)^s$, and thus for $0 \leq j \leq \log n - k - 1$,

$$\Pr[\mathcal{E}_{k+j}] = \left(1 - \frac{2^{k+j}}{n}\right)^s \leq \left(\frac{1}{\mathrm{e}}\right)^{\frac{2^{k+j}}{n} \cdot s} \leq \mathrm{e}^{-2^j} \ .$$

Therefore, $\mathbf{E}[\tau] \leq k + O(1)$.

Now, we upper-bound the number of transmitted messages. Let X_i be a random variable denoting the number of nodes transmitting in phase i. Then, $\mathbf{E}[X_i | \mathcal{E}_{i-1}] = s \cdot p_i$. The expected total number of transmitted messages is then

$$\begin{aligned}
\mathbf{E}\left[\sum_{i=0}^{\log n} X_i\right] &= \sum_{i=0}^{k+1} \mathbf{E}[X_i | \mathcal{E}_{i-1}] \cdot \Pr[\mathcal{E}_{i-1}] + \sum_{j=1}^{\log n - k - 1} \mathbf{E}[X_{k+j+1} | \mathcal{E}_{k+j}] \cdot \Pr[\mathcal{E}_{k+j}] \\
&\leq \sum_{i=1}^{k+1} s \cdot p_i \cdot 1 \ + \sum_{j=1}^{\log n - k - 1} s \cdot p_{k+j+1} \cdot \mathrm{e}^{-2^j} \\
&\leq 4 \cdot s \cdot p_k + s \cdot p_k \sum_{j=1}^{\infty} \left(2^{j+1} \cdot \mathrm{e}^{-2^j}\right) \\
&= O(s \cdot p_k) = O(1) \ .
\end{aligned}$$

As the expected number of broadcasts is constant, the expected number of messages is $O(n)$. □

3 Lower Bounds

Our bounds from the previous section raise the question of what can and cannot be achieved in distributed event size estimation. Hence, we turn our attention to lower bounds. Clearly, in any graph, an algorithm solving the problem needs $\Omega(\delta)$ rounds, as there exists a pair of active nodes in distance δ and any node has to communicate with both of them. Also, each active node has to send or hear at least one message, and therefore the total communication is at least $\Omega(s)$.

Below we again concentrate on cliques. Fix any deterministic algorithm ALG, and assume that only node i is active. Then i transmits messages in some particular order, which we call a *primary schedule for i*. Note that for any starting set of active nodes, ALG uses the primary schedule for i as long as i does not receive a message from other nodes. For succinctness, we say that ALG *p-sends* a message in round ℓ, meaning that the primary schedule of Alg sends a message in round ℓ. We say that two nodes *p-contact* if one of them p-sends a message to the other. Using an averaging argument, we may find a pair of nodes which p-contact after transmitting many messages.

Lemma 1. *For any deterministic algorithm for the clique and for any subset of k nodes A, there exists a pair of nodes $v, v' \in A$ which p-contact only after either of them p-sends at least $k/2 - 1$ messages.*

Proof. First, we observe that the total number of messages in all primary schedules is at least $\binom{k}{2}$. Otherwise, there would exist a pair of nodes which never p-contact. In effect, if the algorithm is run on an instance where only these two nodes are active, it cannot solve the problem, as none of these nodes can distinguish between instances where the second node is active or inactive.

For simplicity of the description, we assume that messages are p-sent sequentially. The j-th message of node i receives label j. An edge between node i and i' receives the label which is the minimum of the labels of messages sent from i to i' and from i' to i. It is therefore sufficient to show that there exists an edge with label at least $k/2$. Assume the contrary, i.e., all edges have labels of at most $k/2 - 1$. Then the label of any message would be at most $k/2 - 1$, which would imply that the total number of p-sent messages is $k \cdot (\frac{k}{2} - 1) < \binom{k}{2}$. □

By Lemma 1, it is possible, for any given algorithm, to choose two active nodes in a clique, so that they contact after sending $\Omega(n)$ messages. In a similar way, we may show that the $\Omega(n)$-message bound holds for an arbitrary number of active nodes.

Corollary 1. *The number of messages sent by any deterministic algorithm in a clique is at least $\Omega(n)$.*

Theorem 3. *For any fixed s and n, there exists an n-node graph, such that for any algorithm for this graph, there exists an instance of the problem with s active nodes, on which the algorithm performs $\Omega(n)$ message transmissions.*

Proof. Fix any $s \leq n$. If $s \geq n/2$, then the theorem follows immediately in any graph by the trivial $\Omega(s)$ lower bound.

Otherwise, we assume that $s < n/2$ and we construct a graph, in which a choice of $s-2$ active nodes is already fixed in a way which cannot help meeting the remaining two active nodes. The graph is depicted in Figure 2. Its nodes are partitioned into three sets: a chain S_0 of $s-2$ nodes, and sets S_1, S_2 containing $\lfloor (n-s+2)/2 \rfloor$ and $\lceil (n-s+2)/2 \rceil$ nodes, respectively. Sets S_1 and S_2 form a complete bipartite graph.

The algorithm is run on an instance where all $s-2$ nodes of S_0 are active, and exactly one node $v_i \in S_1$ and one node $v_j \in S_2$ is active. We show that there exists a pair of nodes v_i, v_j which contact after sending $\Omega(n)$ messages.

As v_i knows that all nodes from S_0 are active, its primary schedule is not affected if it contacts or is contacted by a node from S_0. This time we consider only nodes from S_1 and S_2 and messages crossing the edges between these two

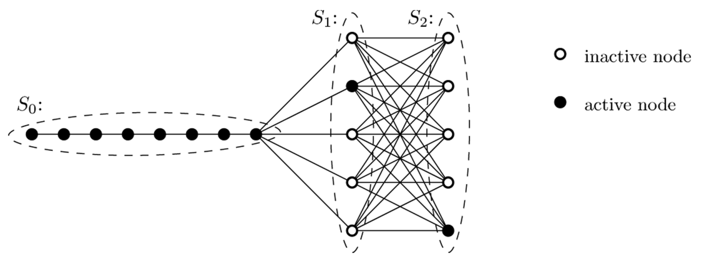

Fig. 2. Illustration of the lower bound for arbitrary number of active nodes

sets. The total number of such messages in all primary schedules have to be at least $|S_1| \cdot |S_2|$. Using the same labeling technique as in the proof of Lemma 1, it is straightforward that there exists an edge between S_1 and S_2 with label at least $f = \frac{|S_1| \cdot |S_2|}{|S_1|+|S_2|}$, as otherwise the total number of messages would be $(|S_1| + |S_2|) \cdot (f - 1) < |S_1| \cdot |S_2|$. Since $f \in \Omega(n)$, the theorem follows. □

The following theorem sheds light on the tradeoff between time and message complexity.

Theorem 4. *Fix any deterministic algorithm* ALG *that solves the problem in a clique using* TIME *rounds and* MSG *messages. Then* TIME·MSG $= \Omega(n \cdot \log \log n)$.

Proof. We assume that $\log \log n \geq 4$. We consider the first t rounds of the nodes' primary schedules, where $t = \log(\log n / \log \log \log n) = \Omega(\log \log n)$.

First, assume that there exists a subset A of $n/2$ nodes, each p-sending less than $n/4$ messages in the first t steps. By Lemma 1, there exists a pair of nodes $v, v' \in A$, which first contact after one of them sends at least $|A|/2 - 1 = n/4 - 1$ messages. Thus, if we start ALG on a graph where only v and v' are active, it takes at least $n/4$ messages and time t.

Hence, in the remaining part of the proof, we assume that there is a set B_0 of at least $n/2$ nodes, each p-sending at least $n/4$ messages within the first t steps.

We create a sequence of sets $\{B_i\}_{i=0}^{t}$, such that B_i is a maximum subset of B_{i-1} with the property that no two nodes of B_i p-send a message to each other in round i. By induction, no node from B_i p-sends a message to another node from B_i within the first i steps. Let $h = \frac{1}{2} \cdot \log \log n$. We show the following property:

Assume that for all $i \leq t-1$, the nodes of B_i p-send in total at most $hn/4$ messages in round i. Then for all $i \leq t$, it holds that $|B_i| \geq \frac{n}{2 \cdot (2h)^{2^i - 1}}$.

We prove the property inductively. The initial case of $i = 0$ holds trivially. Fix any round $i \leq t$. In round $i - 1$ the nodes of B_{i-1} p-sent at most $hn/4$ messages to themselves. Consider a graph on nodes from B_{i-1}, in which an edge exists between a pair of nodes if they contact in round $i - 1$. The average degree in this graph is $(h{\cdot}n)/(2{\cdot}|B_{i-1}|)$ and by Turán's theorem [1], there exists an independent set B_i of size

$$|B_i| \geq \frac{|B_{i-1}|}{1 + \frac{h \cdot n}{2 \cdot |B_{i-1}|}} \geq \frac{|B_{i-1}|^2}{h \cdot n} \geq \frac{n^2}{4 \cdot (2h)^{2^i - 2} \cdot h \cdot n} = \frac{n}{2 \cdot (2h)^{2^i - 1}} \ .$$

In our context, independence means that the nodes of B_i do not p-contact each other in round i.

Finally, we show how the theorem follows by the property above. If there exists a round $i \leq t - 1$ in which nodes of B_i p-send at least $hn/4$ messages, then we run Alg on a graph where only nodes of B_i are active and the theorem follows immediately. Otherwise, B_t contains at least $n/(2 \cdot (2h)^{2^t - 1}) \geq 2$ nodes. Then, if we run Alg on a graph where only nodes of B_t are active, they do not contact within the first t steps and each of them sends at least $n/4$ messages. □

4 Arbitrary Graphs

In this section, we construct algorithms that perform well on arbitrary graphs. First, we note that it is possible to implement distributed *depth/breadth first search* (DFS/BFS) procedures in our environment.

Lemma 2. *A distributed DFS procedure initiated at a single node finishes in time $O(s)$ using $O(n)$ messages. BFS uses $O(\delta)$ rounds and $O(m)$ messages.*

Proof. A BFS procedure is just a simple flooding and its time and message complexities are straightforward.

As for DFS, we fix a starting node. We say that this node holds the "token": the token indicates the node that would be processed in the centralized DFS. This token is a table of current knowledge about all the nodes: nodes are either known to be active, known to be inactive or have unknown state. During our procedure, the token node tries to forward the token to the neighbor which would be next on the DFS tree. This is done as follows. First, the token node "pings" all its neighbors with unknown state and active neighbors respond immediately. Then like in the centralized DFS algorithm the token is passed to any unvisited active node, and if there is none, the token is sent back to the node from which it came. As DFS proceeds along the DFS tree spanning all active nodes in a single component, it takes time $O(s)$. In the process of gaining knowledge each node changes its state just once, so the number of messages is $O(n)$. □

These procedures are useful if there is a predefined leader. Otherwise, we have to start this procedure at all (active) nodes utilizing some level of parallelism.

Lemma 3. *For arbitrary graphs, for any $1 \leq k \leq n$, there exists an algorithm k-*WaitDFS*, which solves the problem in $O(n^2/k)$ rounds using $O(\min\{k, s\} \cdot n)$ messages. There also exist algorithms* ParDFS*, solving the problem in $O(s)$ rounds using $O(s{\cdot}n)$ messages and* ParBFS *which takes time $O(\delta)$ and performs $O(m \cdot s)$ message transmissions.*

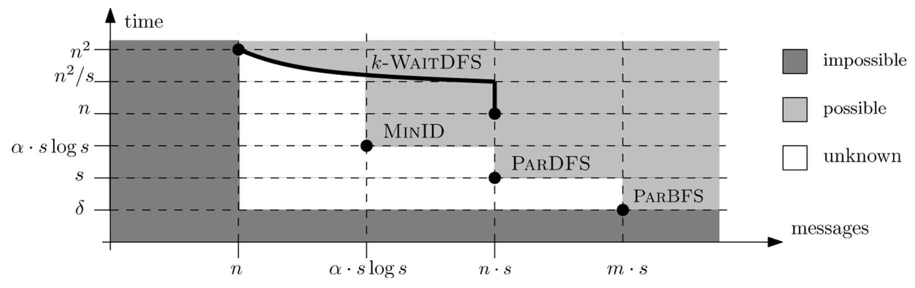

Fig. 3. Performance comparison of different algorithms for arbitrary graphs

Proof. In general, our problem can be solved by running s instances of DFS in parallel. We call this algorithm ParDFS. Obviously, it runs in time $O(s)$ and uses $O(s \cdot n)$ messages.

One way to reduce the number of messages is to have a graded start of the DFS procedures. Of course, as we do not know which nodes are active, we may need to wait for potentially inactive nodes. Concretely, in our algorithm k-WaitDFS, we exploit the fact that the nodes are ordered in the preprocessing stage (i.e. they have IDs from 1 to n). We divide time into $\lceil n/k \rceil$ phases of length $\Theta(n)$. This length is chosen in such a way that for any choice of the active nodes, the worst-case execution of a DFS initiated at any node ends within one phase. In phase i, we define a subset of *busy* nodes. These are nodes with identifiers between $k \cdot (i-1) + 1$ and $k \cdot i$, which have not participated in any DFS so far. All nodes which are active and busy start their DFS procedures, transmitting in total at most $O(\min\{k, s\} \cdot n)$ messages in one phase. In the worst-case, the algorithm finishes after $\lceil n/k \rceil$ phases, i.e., after $O(n^2/k)$ rounds.

If we only care about the time complexity, the optimal algorithm initiated by a single node is a BFS (i.e., flooding). Again, we have to cope with an issue of choosing the node which initiates such a search; in the algorithm ParBFS, all nodes perform a BFS concurrently. □

In the remaining part of this section, we first present a technique for efficient neighborhood discovery and later use it in an algorithm MinID, whose performance is output-sensitive and depends, besides s, only on the arboricity of the graph.

4.1 Neighborhood Discovery

So far, the preprocessing stage was used for assigning identifiers to nodes only. In the following, we assume that the nodes pre-compute a list of neighbors they will contact if they get activated by the event.

We employ the concept of the *arboricity* of an arbitrary graph G which is defined as the minimum number of forests α that are required to cover all edges in G. During preprocessing of the network, we compute respective rooted spanning forests $\mathcal{F} = \{F_1, F_2, .., F_\alpha\}$. We note that this decomposition can be performed in polynomial time [4,10]). For any node v, we define a set $N_v = \{w : \exists\, F_j \in \mathcal{F},\ \text{s.t. } w \text{ is a parent of } v \text{ in } F_j\}$.

In the first round of the event stage, every active node v "pings" all nodes from N_v. At the same time it receives similar probing messages from some of its active neighbors. Pinged active nodes reply in the second round. We observe each active node receives a ping or a reply from each of its active neighbors, and thus learns its active neighborhood.

The neighborhood discovery is performed in two rounds. As each active node v sends $|N_v| \leq \alpha$ test messages followed by the transmission of $|N_v|$ receipts, the total communication complexity is at most $O(\alpha s)$.

4.2 The MinID Algorithm

In the preprocessing phase of MinID, the algorithm assigns identifiers to nodes, discovers the topology, and computes trees $\{F_j\}_{j=1}^{\alpha}$ as described above. In the first two rounds of the event stage, using $O(\alpha s)$ messages, each node learns about its active neighbors. Then a leader election is performed in the way described below. First, we present the algorithm under the assumption that s is known; later we show that this assumption is not critical for our analysis.

The discovery of active components is performed by leader election, where the node with the smallest index distributes its index to everyone else in the component. The algorithm proceeds in $2 \log s$ phases. Initially, each active node v defines its own cluster $C_v = \{v\}$, with v acting as the leader. In due course the number of clusters is reduced, s.t., after at most $2 \log s$ phases a single cluster containing all active nodes in the component is formed. At any time two clusters C_i and C_j are neighbors if there exists an edge (v, w) connecting two active nodes $v \in C_i$ and $w \in C_j$.

We also assume that before entering a new phase each cluster is supported by a spanning tree rooted in the leader. Note that the Euler tour defined on edges of the spanning tree allows to visit all nodes in the cluster in time at most $2s$, e.g., by a token released by the leader. Each cluster C_i is visited by the token three times. During the first visit at each node $v \in C_i$, the token distributes the index i to the entire C_i and all active neighbors of C_i in different clusters. During the second visit, the token collects information about indices of neighboring clusters and it picks the C_j with the smallest index j. If $j < i$, during the third consecutive visit, the token distributes j to all nodes in C_i to inform them that they are now destined for C_j.

Let G_C be a digraph in which the set of nodes is formed of clusters C_i and where there is an arc from C_j to C_i iff nodes of C_i are destined for C_j. A node C_w with in-degree 0 in G_C corresponds to a cluster that during this phase spreads its index to all other clusters reachable from C_w according to the directed connections in G_C. Note also that since the maximum in-degree of nodes in G_C is 1, each cluster with in-degree 1 will receive a new index from exactly one cluster. The process of reindexing is performed by a DFS procedure initiated by the leader in each cluster C_w with in-degree 0 in G_C and it is extended to the nodes of all (not only neighbors) clusters reachable from C_w (according to the connections in G_C). The three visits along Euler tours followed by reindexing take time $O(s)$. The total communication complexity is $O(\alpha s)$, since every edge is traversed a constant number of times.

It remains to show that during two consecutive phases the number of clusters is reduced by half in non-trivial components (containing at least two clusters). Two cases occur. The first case refers to "amalgamation" of clusters where any cluster either delegates its nodes to some other cluster or is a cluster that provides its index to some other clusters. In this case the number of clusters is reduced by half after the execution of a single phase. The second case refers to clusters whose indices form local minima in G_C. If during the first phase such a cluster C_i has a neighbor C_z that chooses to delegate its nodes to some other cluster C_j we know that $j < i$ and j is adopted as the new index of C_z. And since $j < i$ during the next phase C_i will fall into the first case as a cluster that delegates its nodes to some other cluster. Thus, after at most $2 \log s$ phases exactly one cluster resides in each component. Hence, for a single phase, the total time is $O(s \log s)$ and the total communication $O(\alpha s \log s)$.

Finally, recall that the procedure presented above works under the assumption that the value of s is known in advance. Since this is not the case we take an exponentially increasing sequence of upper bounds $2, 4, .., 2^i, .., 2^{\lceil \log n \rceil}$ on s, and run our algorithm assuming for these consecutive powers of two, until the correct bound on s is found. Note that when the algorithm runs with a wrong assumption on the size of the component, the nodes eventually learn that the component is larger than expected. The nodes in clusters that are about to expand too much are informed by their leaders, and the nodes destined for other clusters, if not contacted by the new leader on time, also conclude that the bound on s is inappropriate. Thus, the process is continued until the appropriate bound on s is found and then it is stopped. Therefore in total the time complexity in a component of size s is bounded by $\sum_{i=1}^{\lceil \log s \rceil} O(2^i \cdot \log 2^i) = O(s \log s)$. Similarly, the total communication is $O(\alpha s \log s)$.

Theorem 5. *In a graph G with arboricity α, the deterministic algorithm* MinID *finishes in $O(s \log s)$ rounds using $O(\alpha s \log s)$ messages.*

5 Planar Graphs

Some of our algorithms work much better for planar graphs. The arboricity of a planar graph is 3 [10]. Thus, MinID runs in time $O(s \log s)$ using $O(s \log s)$ messages.

If we run the DFS and BFS procedures (and their variants) after we perform a neighborhood discovery presented in Section 4.1, then we may decrease the number of used messages to $O(s)$. These procedures are performed in a manner that ignores the existence of non-active nodes. In particular, the number of messages used by ParBFS is then $O(s^2)$.

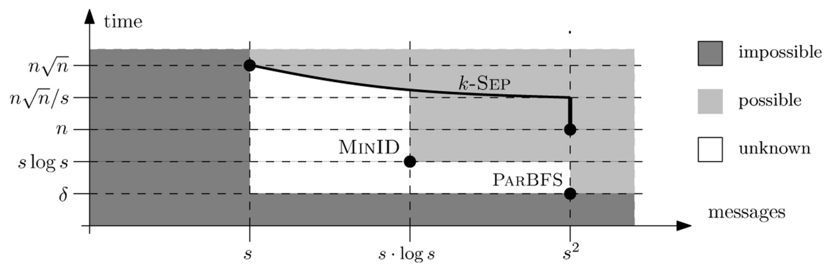

Fig. 4. Performance comparison of different algorithms for planar graphs

Below we present an algorithm k-Sep, which is specially suited for planar graphs. While its runtime is larger than that of MinID, its communication cost is reduced. The performance of the aforementioned algorithms is depicted in Figure 4.

5.1 Hierarchical Decomposition

We start our description from the preprocessing stage. Recall that the planar separator theorem by Lipton and Tarjan [8] enables us to partition any set of nodes V_0 of a planar graph into three disjoint sets: a separator U_0, s.t. $|U_0| \leq c\sqrt{n}$, and sets A_0, B_0 of sizes at most $\frac{2}{3}n$, such that A_0 and B_0 are themselves connected, but there is no edge between them. In the preprocessing stage, this theorem is applied recursively starting from V to produce a binary decomposition tree; if an internal node corresponds to a set $V_0 = U_0 \uplus A_0 \uplus B_0$, then its children correspond to sets A_0 and B_0; each leaf contains a single node.

At the beginning of the event stage, the algorithm Sep performs a neighborhood discovery. Then it proceeds recursively as described below. Sep starts from the root of the tree, which corresponds to three sets $V = V_0 = U_0 \uplus A_0 \uplus B_0$. It initiates $|U_0|$ DFS procedures sequentially, i.e., one after another, starting at vertices of U_0. These procedures are allowed to visit only nodes in V_0; for the execution of each of them we reserve $O(|V_0|)$ rounds. Moreover, a node which already took part in any DFS, does not start its own DFS. Thus, if an active component has a non-empty intersection with U_0, say at a node v, then a DFS in which v participates solves the task in this component. There are possibly other active components. This is where the separating property comes into play: such an active component is contained entirely inside A or inside B, and Sep is run recursively in parallel for these sets.

To bound the number of messages, we observe that each active node participates in exactly one DFS, and thus $O(s)$ messages suffice. To bound the number of rounds, we observe that any separator set contains at most $c\sqrt{n}$ vertices. Further, the execution of a single DFS started within a set V_0 takes time $O(|V_0|)$. Therefore, the number of used rounds is at most $O(c\sqrt{n}) \cdot (n + \frac{2}{3}n + \left(\frac{2}{3}\right)^2 n + \ldots) = O(n \cdot \sqrt{n})$.

Using the same technique of setting the level of parallelism of DFS procedures as in k-WaitDFS, we derive the following theorem.

Theorem 6. *For any $1 \leq k \leq n$, there exists an algorithm k-*Sep*, which solves the problem in $O(n \cdot \sqrt{n}/k)$ rounds, using $O(\min\{s \cdot k\} \cdot s)$ messages.*

Proof. The algorithm k-Sep works essentially in the same way as the original Sep algorithm, but within a single set $V_0 = U_0 \uplus A_0 \uplus B_0$ corresponding to a tree node, it utilizes some level of parallelism in running DFS procedures. Namely, an original Sep algorithm runs the 1-WaitDFS procedure there, whereas k-Sep runs k-WaitDFS, where each of $|U_0|$ DFS procedures is run for $O(|V_0|)$ steps. As in the analysis of the k-WaitDFS algorithm, this gives us at most $O(\min\{k, s\} \cdot s)$ messages and $O((c \cdot \sqrt{n}/k) \cdot |V_0|)$ rounds.

In total, the algorithm also uses at most $O(\min\{k, s\} \cdot s)$ messages, because it stops after a successful DFS. The total number of rounds is then $O(\sqrt{n}/k) \cdot (n + \frac{2}{3}n + \left(\frac{2}{3}\right)^2 n + \ldots) = O(n \cdot \sqrt{n}/k)$. □

References

1. Alon, N., Spencer, J.: The Probabilistic Method. John Wiley, Chichester (1991)
2. Birk, Y., Keidar, I., Liss, L., Schuster, A., Wolff, R.: Veracity Radius: Capturing the Locality of Distributed Computations. In: Proc. 25th ACM Symp. on Principles of Distributed Computing (PODC), pp. 102–111 (2006)
3. Davison, A.: Laptops as Earthquake Sensors. In: MIT Technology Review (2008)
4. Gabow, H.N., Westermann, H.H.: Forests, Frames, and Games: Algorithms for Matroid Sums and Applications. Algorithmica 7(1), 465–497 (1992)
5. Jacob, R., Richa, A., Scheideler, C., Schmid, S., Taeubig, H.: A Polylogarithmic Time Algorithm for Distributed Self-Stabilizing Skip Graphs. In: Proc. 28th ACM Symposium on Principles of Distributed Computing, PODC (2009)
6. Jiang, C., Dong, G., Wang, B.: Detection and Tracking of Region-Based Evolving Targets in Sensor Networks. In: Proc. 14th International Conference on Computer Communications and Networks, ICCCN (2005)
7. Kim, H.-A., Karp, B.: Autograph: Toward Automated, Distributed Worm Signature Detection. In: Proc. 13th Usenix Security Symposium, Security (2004)
8. Lipton, R.J., Tarjan, R.E.: A Separator Theorem for Planar Graphs. SIAM Journal of Applied Mathematics 36, 177–189 (1979)
9. Mans, B., Schmid, S., Wattenhofer, R.: Distributed Disaster Disclosure. In: Proc. 11th Scandinavian Workshop on Algorithm Theory, SWAT (2008)
10. Nash-Williams, C.S.J.: Edge-disjoint spanning trees of finite graphs. J. of London Mathematical Society 36, 445–450 (1961)
11. Peleg, D.: Distributed Computing: A Locality-sensitive Approach, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics (2000)