Finding Spanning Forests by Broadcasting

Bogdan S. Chlebus
University of Colorado at Denver, U.S.A.

Karol Gołąb
Uniwersytet Warszawski, Poland

Dariusz R. Kowalski
Universytet Warszawski, Poland

Abstract

We consider the problem of finding a spanning forest of a graph by distributed algorithms. If the input graph is weighted, then the goal is to find a minimum-weight spanning forest. The input is distributed among the processors in such a way that each edge is held by one processor. We also consider an adversarial model, in which an adversary controls the timing of obtaining inputs by processors. The processors communicate by broadcasting. The goal is to have the edges of a spanning forest broadcast on the network. The underlying broadcast network is implemented as a multiple-access channel.

We study the optimality of algorithmic solutions, which may depend on the class of algorithms. We consider deterministic versus randomized algorithms, and adaptive versus oblivious ones. We also prove lower bounds, that either justify the optimality of specific algorithms, or show that optimal performance depends on the class of algorithms.

Keywords

distributed algorithm, lower bound, minimum-weight spanning forest, broadcast network, multiple-access channel, randomized algorithm, oblivious algorithm.

1 Introduction

We consider a distributed system in which processing units communicate by broadcasting. The underlying broadcast network is implemented by a multiple-access channel, and processors are called stations. This protocol of communication has the following key property: a message sent at a step is successfully received by all the stations only if there is exactly one station that performs broadcast at this step. We work with the model in which a collision, of the messages sent simultaneously by different stations at a step, results in a feedback allowing all the stations to detect this.

We study the problem of finding a spanning forest of a graph. The input may be either a simple graph, with no weights assigned to edges, or it may have a weight given for each edge, then the goal is to find a minimum-weight spanning forest. The input is distributed among the stations: each edge is held by one station. During the execution of an algorithm, some edges are heard on the channel. We require that *all* the edges of a spanning forest we seek are revealed explicitly by having been heard at some steps.

We consider two possible scenarios of algorithms obtaining the input: static and dynamic. In the static case, the input is provided to the participating stations at the start of an execution. In the dynamic case, there is an adversary that controls the timing when the station holding a given edge is woken up and allowed to join the process of computation.

The questions we address in this paper concern the time efficiency of finding a spanning forest. We compare adaptive and oblivious algorithms, and deterministic with randomized ones, and we prove lower bounds. For a given class of algorithms, the goal is to identify the optimum time performance of an algorithm in the class. When comparing classes of algorithms, the question is whether the optimum performances differ among the classes.

Early work on the multiple-access channel has concentrated on distributed protocols to handle bursty traffic of packets carrying dynamically generated messages. It included the development of the protocols of Aloha and then the Ethernet. A survey by Gallager [5] covers the results up to 1985, recent papers include those by Goldberg, MacKenzie, Paterson and Srinivasan [9], Håstad, Leighton and Rogoff [12], and by Raghavan and Upfal [23]. Multiple-access channel can be viewed as a special case of a multi-hop radio network. It is actually single-hop, since one step is sufficient to have a message go between any two stations. For an overview of work on communication in radio networks, including the single-hop ones, see a survey by Chlebus [2].

In static problems on the multiple-access channel, we assume that the input has been given to the stations at the start of a protocol. One of the most natural such problems is that of selection, where some among the N stations are given messages and the goal is to have any of them heard on the channel. Willard [24] developed a randomized protocol for this problem, that operates in the expected time $O(\log\log N)$, for a channel with collision detection. Solving this problem requires the expected time $\Omega(\log N)$, if detection of collision is not available, as was shown by Kushilevitz and Mansour [15], hence there is an exponential gap between these two models. In the all-broadcast problem we are asked to have all the messages given to some k stations, selected among the N ones, heard on the channel. This can be done deterministically with a logarithmic overhead per message, by an algorithm of Komlós and Greenberg [14], which is optimal as shown by Greenberg and Winograd [11].

Gąsieniec, Pelc and Peleg [8] studied the wakeup problem, which is a dynamic version of the selection problem. The goal again is to have a successful transmission as soon as possible, but the timing of stations joining the protocol is controlled by an adversary. This paper shows how efficiency depends on various levels of synchrony, and it compares randomized and deterministic solutions.

The issues discussed above concerned the communication itself on the multiple-access channel. Some research has also been done concerning distributed algorithmics when the underlying communication is implemented by a multiple-access channel. Martel and Vayda [19, 20] studied the problem of finding the maximum value among those stored at a subset of stations. Chlebus, Kowalski and Lingas [3] considered the problem of performing a set of independent unit tasks, when the stations may fail by crashing.

The most popular model, to study distributed algorithms finding minimum-weight spanning trees, has the processors as nodes of the graph and the communication links are the edges of the graph. This provides a unique combination in which the underlying communication network is also an input. The problem of finding a minimum-weight spanning tree in such a setting was first proposed by Gallager, Humblet and Spira [6]. Awerbuch [1] developed an algorithm working in time $O(n)$, where n is the number of nodes. Garay, Kutten and Peleg [7] found a solution with performance proportional to the diameter, if the diameter is sufficiently large. Faloutsos and Molle [4] studied trade-offs between the time and the number of messages. Lower bounds on the time have been given by Peleg and Rubinovich [22] and by Lotker, Pat-Shamir and Peleg [18].

Other graph problems studied for this model include finding maximal matchings, a deterministic algorithm has been developed by Hańćkowiak, Karoński and Panconesi [13], and edge coloring, a randomized algorithms has been given by Grable and Panconesi [10]. An algorithm finding k-dominating sets was given by Kutten and Peleg [16]. The issues of locality in distributed graph algorithms have been studied by Linial [17], see the book by Peleg [21] for a comprehensive coverage.

2 Technical preliminaries

We consider distributed algorithms performed by stations that communicate over a broadcast network. The computations are synchronous, all the stations have access to a global clock.

Adversaries. The stations are categorized at each step as either *active* or *passive,* and only active stations participate in the computation. A passive station may be *activated* at any step, then it changes its status and becomes active. Such decisions, concerning which stations to activate and when to do this, are made by the *adversary*. In a *static* scenario, the active stations are stipulated at the start of an execution, and this is not changed later by the adversary. If this is the case, then we do not mention the adversary at all. Only if the adversary activates passive stations in the course of an execution, which is the *dynamic* scenario, then we mention this possibility explicitly.

An activated station is aware of being activated at the given step, and can immediately access the number of the step, as counted by a global clock. Many stations may be activated at a step, or none.

Multiple-access channel. The broadcast operation is implemented on a *multiple-access channel.* All the stations attached to it receive the same information from the channel at each step, unless they are passive, this information is said to be *heard* on the channel. The basic property of the channel is that only if exactly one station attempts a broadcast operation then its message is delivered to all the active stations by the end of the step.

Multiple-access channel comes in two variants: either *with* or *without collision detection.* In the former case, if more than one stations broadcast at a step then all the stations can hear the *collision noise* on the channel. This signal is distinct from the *background noise,* which is heard when no station performs a broadcast at a step. In the channel without collision detection, both kinds of noise signals are received as indistinguishable. In this paper we work with the collision-detection variant.

Graphs and algorithms. We consider graph problems, the inputs are simple graphs, possibly with weights assigned to edges. The goal of algorithms is to find a maximal spanning forest of the input graph, of the smallest weight in the case of weighted graphs. A spanning forest is maximal if adding any edge from outside the forest would create a cycle. Throughout the rest of this paper, when we refer to a spanning forest, we always mean a maximal one.

The set of nodes of the input graph is the interval $[1..n]$, where n is the number of nodes known by all the stations. Each station holds a single edge. Additionally, each station is assigned a unique identifier from a contiguous segment of integers starting from 1.

We require that all the edges of the forest sought are *revealed* by being heard on the channel. Notice that, after having revealed the edges of a spanning subgraph that has as many connected components as the input graph, its spanning forest can be determined by some direct rule. For instance, if there are no weights on edges, the algorithm can take the first spanning forest in lexicographic ordering of sets of edges. If there are weights on edges, only spanning forests of the smallest weight are considered.

The stations are activated by an adversary but they halt on their own. The issue of termination and correctness needs to be clarified precisely, since an adversary might activate a number of stations just before the stations already active have decided to terminate. We say that the algorithm *terminates* at the step when some station successfully broadcasts a special *termination signal.* To define correctness, we need to disregard the edges held by the stations activated too late. To this end, let us call the period of the last c steps by termination the *closing c steps,* for any fixed integer $c > 0$. An algorithm is said to be *c-correct,* for a positive integer c, if the output based on the edges revealed by the closing c steps is a spanning forest, of the minimum weight in the case of weighted graphs, for the graph with the edges held by the stations that have been activated before the closing c steps. An algorithm is *correct* if it is c-correct for some integer $c > 0$ and sufficiently large number of nodes n. The number of edges is denoted as m, we count only the edges held by the stations activated before the last c closing steps, where parameter c is some fixed constant, as required in the definition of correctness.

Typically, when a station performs a broadcast, then its message contains only its input edge. An algorithm proceeds as a sequence of *queries,* each specifies the stations that are to attempt to broadcast in the given step. We use the notation in which query Q_i is a list including the ids of stations and/or edges. Station s_k attempts to broadcast its edge on the channel at step i if it is *specified by the query* Q_i, which means that either the id k of station s_k is in Q_i or s_k holds an edge that is in Q_i.

An algorithm is *adaptive* if each query Q_{i+1} depends on the feedback heard on the channel when the preceding queries $Q_1, Q_2, \ldots, Q_i$ were executed. An algorithm is *oblivious* if the queries $Q_1, Q_2, \ldots, Q_i, \ldots$ are all known in advance, and depend only on the number n of nodes of the input graph. Randomized algorithms, that we develop, are also described by queries, but each station specified by a query additionally first performs a random experiment, and then actually performs broadcast only if the result of the experiment was a success. Such experiments are Bernoulli trials, that is, they return either success or failure, which are independent over the steps and the stations, the probability of success at a step may vary over the steps.

3 Simple graphs

The graphs considered in this section do not have weights assigned to the edges. The input is given at the start of an execution, no adversaries are involved.

3.1 Deterministic adaptive algorithm

We start by giving a routine RESOLVE(Q) to resolve conflicts among the set of stations that want to reveal their edges Q at the same time. It is based on the binary-search paradigm. Let us fix a linear ordering I of all the possible edges of a graph with vertices in $[1..n]$, for instance the lexicographic one. First all the stations whose edges are in Q broadcast simultaneously. If this results in silence then Q is empty and RESOLVE is completed. If an edge is heard then set Q is a singleton and the conflict is resolved, so RESOLVE is completed. Otherwise a collision is detected. Let I_1 and I_2 be a partition of I into left and right subintervals, respectively, determined by the median of I. In the next step all the stations in $Q \cap I_1$ broadcast. If this results in a single edge heard on the channel then RESOLVE is done. If there is a collision then $I_1 \cap Q$ is searched recursively. If there is silence then $I_2 \cap Q$ is searched recursively. It takes $O(\log n)$ steps to resolve a conflict and hear the smallest edge in Q.

The algorithm proceeds as follows. It iterates phases in a loop. The set of the input edges are partitioned into three subsets:

A_1: the edges already revealed on the channel;

A_2: the edges that would make a cycle if added to those in A_1;

A_3: the remaining edges.

The correctness of this algorithm follows from the invariant maintained in each iteration: the edges in A_1 make a forest on the set of vertices $[1..n]$. The edges in A_3 are called *waiting,* each of them could be added to A_1 and still the property that A_1 is a forest would be maintained. A station holding a waiting edge is also called waiting. Initially A_1 and A_2 are empty, and A_3 consists of all the input edges. During one iteration, procedure RESOLVE(A_3) is called, and the edge heard on the channel is added to A_1. Each of the remaining waiting stations checks to see if it is now in A_2, and if this is the case then it does not perform any broadcast till the termination. When eventually no waiting edge remains, then the station with the first id broadcasts the termination signal.

The overall cost is $O(n \log n)$. It may happen that $m = o(n \log n)$. To have the algorithm work in time $O(m)$ in such a case, it is modified as follows. The algorithm as described above uses the odd-numbered steps, while in the even-numbered ones the stations broadcast their edges on the channel one by one, in the order of their ids. This shows the following:

Theorem 1 *There is a deterministic adaptive algorithm for the static model which finds a spanning forest of a simple graph in time* $O(\min\{m, n \log n\})$. □

3.2 Randomized algorithm

Next we present a randomized algorithm that finds a spanning forest in a graph in the expected time $O(n)$. The algorithm is similar in structure to the deterministic one, and we use the same terminology. It is performed in a loop, an iteration adds a new waiting edge. The main difference is that the expected time to reveal a waiting edge is constant.

The algorithm uses a variable, denoted as a, that is maintained by all the stations, to approximate the number of waiting stations during an execution of the algorithm. This estimate of the number of waiting edges is updated in each iteration.

1. Set $a := n(n-1)/2$
2. Each waiting station broadcasts its edge with the probability $1/a$, independently over all such stations
3. If silence is heard on the channel then set $a := a/2$ and go to Step 2
4. If collision is heard on the channel then set $a := 2a$ and go to Step 2
5. Each waiting station broadcasts its edge
6. If silence is heard on the channel then terminate
7. Go to Step 2

Figure 1: Randomized algorithm for simple graphs without weights.

A pseudo-code of the algorithm is in Figure 1. The algorithm is correct because as long as the value of a is smaller than 1 then there is still a possibility to add more edges to the current spanning forest.

To show that the algorithm works in time $O(n)$ we apply the method similar to that used by Martel in [19], which in turn relies on the results of Willard [24]. Observe that our first estimation of the number of active stations is an overestimate. As the algorithm proceeds, the number of active stations decreases monotonically, allowing to update the estimate as in [19] (theorem 2.4), which results in an amortization of the number of trials needed to keep the estimate a close to the true number of the waiting edges. We also substitute n for k, as there are at most $n-1$ edges in a spanning forest, and $n(n-1)/2$ for c_1. It follows that our algorithm runs in the expected time $O(n+\log n^2) = O(n)$.

Theorem 2 *There is a randomized algorithm for the static model that finds a spanning forest of a simple graph in the expected time* $O(\min\{m,n\})$. □

3.3 Lower bound for oblivious algorithms

In this subsection we prove a lower bound $\Omega(m)$ on the number of queries required by an oblivious algorithm. This shows that there is a gap in optimum performance between adaptive and oblivious algorithms.

We may assume that all the queries contain at most one station's id of range between 1 and m, since otherwise no edge is heard on the channel. We give a construction that works under the assumption that queries do not contain any id of a station, only edges. The remaining case is omitted, it will be discussed in the final version of the paper.

Suppose $\mathcal{A}$ is an algorithm and m is the number of edges we would like to be the size of an input. We start with a setting in which for each edge in the complete graph K_n there is a station holding this edge. We specify which of these edges should be removed, in such a way that the total number of edges is m. We may assume that the inequalities $2n \leq m \leq \frac{n(n-1)}{4}$ hold.

Let $k = \lfloor \frac{m-n}{2} \rfloor$. We consider queries $Q_1, \ldots, Q_k$ of algorithm $\mathcal{A}$. Let $Q = \bigcup_{i \leq k} Q_i$. Our goal is to construct a graph $G = (V,T)$, such that $V = \{1, \ldots, n\}$, $|T| = m$ and there is a node $v \in V$ such that, for each node $w \in V$ different from v and query Q_i, we have $Q_i \cap T \neq \{(v,w)\}$.

We define graph $G = (V,T)$ that makes $\mathcal{A}$ perform $\Omega(m)$ steps. To this end we use the following procedure:

Phase 0: Initialize $Q := \bigcup_i Q_i$ and $E := Q$. Set Q is fixed and contains all edges that can appear in queries. Set $Q \setminus E$ is dynamic and contains set of *forbidden* edges that we want to remove from the initial graph K_n.

Phase 1: Remove from E those edges that do not isolate any node in graph spanned on Q, but would be revealed successfully, these are the edges that are singleton intersections of the current E and some Q_i. The set $Q \setminus E$ contains the forbidden edges, when this phase has been completed.

> `stop` $:= 0$
> **while** `stop` $= 0$ **do**
>   `stop` $:= 1$
>   **if** there is a Q_i such that $|Q_i \cap E| = 1$
>   and there is no isolated node in $E \setminus Q_i$
>   **then** $E := E \setminus Q_i$ and `stop` $:= 0$

Phase 2: The current set $Q \setminus E$ will not change until the end of the construction. Now we begin to construct the set T, and we will keep increasing it during the next phases.

> set $T \subseteq E$ to a maximal forest in E

Phase 3: After this part set T will have all the intersections with Q_i of size at least two. This can be achieved by adding sufficiently many edges to T, in case any such intersection is of size smaller than two. This is true for all Q_i, except for those that contain less than two edges that are not forbidden, that is, $|Q_i \cap E| < 2$.

> **for** $i := 1$ **to** k **do**
>   **if** $Q_i \cap T = \{e_1\}$ and $|Q_i \cap E| > 1$ **then** $T := T \cup \{e_2\}$,
>   for some $e_2 \in Q_i \cap E$ such that $e_2 \neq e_1$

if $|Q_i \cap T| = 0$ and $|Q_i \cap E| > 1$ **then** $T := T \cup \{e_1, e_2\}$,
for some $e_1, e_2 \in Q_i \cap E$ such that $e_1 \neq e_2$

Phase 4: In this part we increase the number of edges in T to be as large as possible, preferably equal to m.

for $i := 1$ **to** k **do**
if $|T| < m$ and $Q_i \cap T \neq \emptyset$ **then** $T := T \cup X$,
where $X \subseteq Q_i \cap E$ is any set of maximal size
such that the updated T is of size at most m

Phase 5: This is the closing phase, its purpose is to guarantee that the size of the set of edges is sufficiently large.

If $|T| < m$ then add $m - |T|$ edges to T that are not in Q.

The oblivious algorithm $\mathcal{A}$ does not broadcast a spanning forest for graph $G = (V, T)$ during $k = \Omega(m)$ steps.

Theorem 3 *Assume that $m \leq n^2$ and $\mathcal{A}$ is a deterministic oblivious algorithm that always reveals a spanning forest. Then there exists a graph $G_{\mathcal{A}}$, with n nodes, $\Theta(m)$ edges and no isolated nodes, such that algorithm $\mathcal{A}$ requires time $\Omega(m)$ if given graph $G_{\mathcal{A}}$ as the input.* □

4 Weighted graphs

In this section we consider graphs that have a positive weight assigned to each edge. The input is specified at the start of an execution, no adversaries are involved.

4.1 Randomized algorithm

We present a randomized algorithm that finds a minimum-weight spanning forest of a graph in $O(n \log(m/n))$ expected queries.

Throughout this section an edge is said to be *known* if it was broadcast as a minimal edge among the waiting ones. The remaining edges broadcast during the selection are not categorized as known. The *waiting* edges again are those that would not create a cycle if added to the known minimum-weight spanning tree.

The algorithm works in a loop. The goal of each iteration is to find a minimal waiting edge which then becomes known. This is done by selecting a minimal edge using the algorithm presented in [19] as a subroutine and updating the estimate of the number of waiting edges for the next iteration. A pseudocode of

Numbers i, a_i and b_i are global variables used in the algorithm.

1. Each waiting station holding an edge with weight less than b_i broadcasts its edge with its weight with the probability $1/a$, independently over all such stations
2. If there is an edge e with its weight w_e broadcast on the channel, go to Step 9
3. If collision is heard on the channel then set $a := 2a$ and go to Step 1
4. If silence is heard on the channel then set $a := a/2$
5. Each waiting station with weight less than b_i broadcasts its edge with its weight
6. If collision is heard on the channel then set $a := \max\{a, 2\}$ and go to Step 1
7. If silence is heard on the channel go to Step 12
8. Let e be the edge broadcast on the channel and let w_e be its weight
9. Set $i := i + 1$
10. Set $b_i := w_e$, $a_i := a$
11. Go to Step 1
12. Edge e is the minimal waiting edge and it becomes a known edge

Figure 2: Procedure FindMin to find a minimal edge.

the procedure to find a minimal edge is given in Figure 2. This subroutine uses $O(\log N)$ expected queries in each envocation, for N participating stations.

A pseudocode of the algorithms is given in Figure 3. The algorithm is essentially an implementation of the greedy minimum tree finding algorithm. The expected time of the algorithm is of the order of magnitude of the sum of the expected times of executions for all the instances of the procedure FindMin.

The total time needed to re-estimate the number of waiting stations is roughly equal to that of execution of all instances of procedure FindMin. This fact follows from the observation that counter i is incremented only during procedure FindMin and decremented only during re-estimation phases.

Each station may take part in many instances of procedure FindMin. However each station may become passive during only one phase of re-estimation of the number of waiting stations. Furthermore, we can match re-estimation phases with

1. Set $a := n(n-1)/2$
2. Set $i := 0$, $b_i := \infty$, $a_i := a$
3. Use Procedure FindMin to find minimal waiting edge
4. Each waiting station with weight less than b_i broadcasts its edge
5. If silence is heard on the channel and i equals 0 then terminate
6. If silence is heard on the channel then set $i := i-1$ and go to Step 4
7. Set $a := a_i$
8. Go to Step 3

Figure 3: Randomized algorithm for minimal spanning forest.

parts of instances of procedure FindMin in a one-to-one fashion, which allows to estimate the contribution from FindMin by properties of the re-estimation phase.

There are at most $n-1$ re-estimation phases in which m edges become passive (known edges also becomes passive). The total time needed by matched segments of procedure FindMin is thus

$$\mathcal{T} = \sum_{i=1}^{n-1} \log r_i$$

where r_i is the number of stations that become passive during the i-th re-estimation phase.

Function $\mathcal{T}$ attains its maximum for $r_i = \frac{m}{n-1}$. This gives the following:

Theorem 4 *There is a randomized algorithm to find a minimum-weight spanning forest that runs in the expected time* $O(n\log(m/n))$. □

4.2 Lower bound for deterministic algorithms

For each deterministic adaptive algorithm $\mathcal{A}$, we construct a certain weighted graph $G_{\mathcal{A}}$ of n nodes and m edges such that the algorithm requires time $\Omega(m)$ on this graph. We start with any assignment of edges to stations at the very beginning of computation, such that the graph $G=(V,E)$ has no isolated nodes. Our goal is to assign weights to all the edges. Let $A_i(t)$ denote a subset of numbers $\{1/j : j\in\mathbb{N}\}$ corresponding to station i after step t of construction. Initially, for every station i, we set $A_i(0) = \{1/j : j\in\mathbb{N}\}$.

Since there is a one-to-one correspondence between the edges and the stations, we will treat E also as a set of stations. Denote by $E(t)$ the set of stations which have an edge with some weight assigned to it by step t of algorithm $\mathcal{A}$. Each station i from $E(t)$ has just one element in $A_i(t)$. We assume the following invariant after step t: for each station $i \in E \setminus E(t)$, the set $A_i(t)$ is infinite.

Consider step $t+1$ of algorithm $\mathcal{A}$, as determined by step t of the construction. Let $S_0(t+1)$ denote the set of these stations i, that do not broadcast during step $t+1$ of algorithm $\mathcal{A}$ for an infinite number of possible weights from $A_i(t)$. Similarly, let $S_1(t+1)$ denote the set of stations that broadcast in step $t+1$ for infinitely many possible initial values.

The following three cases may hold:

CASE 1: $S_1(t+1) \setminus S_0(t+1) = \emptyset$.
Make set $A_i(t+1)$ to contain all the weights from $A_i(t)$, for which station i does not broadcast during step $t+1$ of algorithm $\mathcal{A}$, for each station $i \in S_0(t+1)$. Hence $E(t+1) = E(t)$ and the invariant holds after step $t+1$. Moreover, there are no new stations broadcasting on the channel, hence we can determine what happens in step $t+1$ by considering only stations in $E(t)$.

CASE 2: $S_1(t+1) \setminus S_0(t+1) = \{i\}$.
We set $A_i(t+1) = \{1/l\}$, where $1/l$ is the minimum weight in $A_i(t)$ for which station i broadcasts in step $t+1$ of algorithm $\mathcal{A}$. For other stations j in $E \setminus E(t)$ we set $A_j(t+1) = A_j(t) \cap (0, 1/l)$. In this case $E(t+1) = E(t) \cup \{i\}$ and the invariant holds after step $t+1$. In step $t+1$, only station i broadcasts, among stations in $E \setminus E(t)$, hence by an argument similar to that used in Case 1 we can decide on what happens in step $t+1$.

CASE 3: $|S_1(t+1) \setminus S_0(t+1)| > 1$.
We choose two different stations $i_1, i_2 \in S_1(t+1) \setminus S_0(t+1)$ and decide on the sets to be $A_{i_1}(t+1) = \{1/l_1\}$ and $A_{i_2}(t+1) = \{1/l_2\}$, similarly as for node i in Case 2. For other stations j in $E \setminus E(t)$ we set $A_j(t+1) = A_j(t) \cap (0, 1/\max(l_1, l_2))$. In step $t+1$ only stations i_1, i_2 broadcast, among stations in $E \setminus E(t)$, hence, by a similar argument as in Case 1, we can decide on the events in step $t+1$.

This construction gives the following result:

Theorem 5 *For each adaptive deterministic algorithm $\mathcal{A}$ that finds a minimum spanning forest, there is a graph $G_{\mathcal{A}}$ such that algorithm $\mathcal{A}$ terminates after $\Omega(m)$ steps.* □

5 Adversarial environment

In this section we consider dynamic graphs. There is an adversary who is able to control the time when the station holding a particular edge is activated. First we prove a lower bound for deterministic adaptive algorithms in this model.

Assume $m > 2n$. We define a strategy of an adversary against a fixed determin-

1. Set $E = \emptyset$. Select a leader by binary search.
2. All the stations update E. The leader broadcasts the contents of E and the interval $[i, j]$ in one message.
3. Each active station with id in $[i, j]$ that holds an edge independent with E broadcasts the edge on the channel.
4. All the stations update set E. Each active station that holds an edge independent with E broadcasts its edge.
5. If silence was not heard at Step 4 then go to Step 2.
6. The leader broadcasts the termination signal.

Figure 4: A deterministic algorithm for an adversarial environment.

istic adaptive c-correct algorithm $\mathcal{A}$, for any fixed positive constant c. Assume that n is sufficiently large, such that $c < \log n$. We partition an execution into stages, each containing $\lfloor \frac{\log n}{2} \rfloor$ steps. In the beginning of each stage, the adversary activates exactly two stations and assigns two suitable edges to them. The chosen stations and edges must satisfy the condition that during the current stage none of these stations will broadcast successfully.

Suppose that stage $k < n/2$ starts and the above condition has been satisfied for all the previous stages. Hence the algorithm has not terminated yet, by the correctness condition and the invariant for stage $k-1$. Moreover, at most $2(k-1)$ stations are active. Now we describe a mechanism of selection. Let us consider all the pairs (i, e), where i is the id of a passive station and e is an edge independent from the graph containing all the edges held by the active stations. There are at least $m - 2(k-1) > n$ possible stations and $n-1$ possible edges. We can select two pairs that perform the same action during stage k, which moreover takes $\lfloor \frac{\log n}{2} \rfloor$ steps.

By the invariants, we can activate at most n stations with independent edges by step $\lfloor \frac{n \log n}{4} \rfloor$, and algorithm $\mathcal{A}$ has not terminated by this step, since in each stage there are active stations that hold independent edges not revealed yet.

This shows:

Theorem 6 *Every deterministic adaptive correct algorithm requires time $\Omega(n \log n)$ to broadcast a spanning forest, if run against a certain adversary.* □

Next we give a deterministic adaptive algorithm whose performance matches this lower bound, its pseudocode is in Figure 4.

The notation and updating interval $[i, j]$ are as follows. The variable E denotes the set of edges revealed on the channel by the current step. The interval $[i, j]$ is chosen by the leader by binary search. Assume for simplicity that m is a power of 2. We start with $[1, m/2]$. If E was updated in the previous iteration in Step 3 or at the beginning of Step 2 then set $[i, j] = [1, m/2]$. Otherwise, if there was silence heard in the last execution of Step 3, take $[j+1, j+(j-i+1)/2]$ as the new interval. If there was a collision heard in the last execution of Step 3, then take $[i, (i+j-1)/2]$ as the new interval.

This shows:

Theorem 7 *There is an algorithm, finding a spanning forest, that is* 2*-correct and terminates in time* $O(n \log n)$ *against any adversary.* $\square$

6 Discussion

We show that adaptive deterministic algorithms are more efficient than oblivious ones. Estimating the optimum performance of a deterministic adaptive algorithm for simple graphs is an open problem. We develop an optimal randomized algorithm for simple graphs without weights. It is an open problem if the performance of this algorithm can be matched by that of a deterministic one.

We develop a randomized algorithm finding a minimum-weight spanning tree of a graph in $O(n \log(m/n))$ steps, and show that any deterministic one requires time $\Omega(m)$. This shows that randomization helps for this problem, for sufficiently many edges. Estimating the optimum performance of finding a maximum-weight spanning forest by a randomized algorithm is an open problem.

References

[1] B. Awerbuch, Optimal distributed algorithms for minimum-weight spanning trees, counting, leader election and related problems, in *Proc.,* 19*th Symp. on Theory of Computing,* pp. 230 - 240, 1987.

[2] B.S. Chlebus, Randomized communication in radio networks, a chapter in “Handbook on Randomized Computing,” P.M. Pardalos, S. Rajasekaran, J.H. Reif, and J.D.P. Rolim, (Eds.), Kluwer Academic Publishers, vol. I, pp. 401 - 456.

[3] B.S. Chlebus, D.R. Kowalski, and A. Lingas, The Do-All problem in broadcast networks, in *Proc.,* 20*th ACM Symp. on Principles of Distributed Computing,* 2001, pp. 117 - 126.

[4] M. Faloutsos, and M. Molle, Optimal distributed algorithm for minimum spanning tree revisited, in *Proc.,* 14*th ACM Symp. on Principles of Distributed Computing,* 1995, pp. 231 - 237.

[5] R.G. Gallager, A perspective on multiaccess channels, *IEEE Trans. on Information Theory,* 31 (1985) 124 - 142.

[6] R.G. Gallager, P. Humblet, and P. Spira, A distributed algorithm for minimum-weight spanning trees, *ACM Transactions on Programming Languages and Sytems,* 5 (1983) 66 - 77.

[7] J.A. Garay, S. Kutten, and D. Peleg, A sub-linear time distributed algorithms for minimum-weight spanning trees, *SIAM J. on Computing,* 27 (1998) 302 - 316.

[8] L. Gąsieniec, A. Pelc, and D. Peleg, The wakeup problem in synchronous broadcast systems, *SIAM J. on Discrete Mathematics,* 14 (2001) 207 - 222.

[9] L.A. Goldberg, P. MacKenzie, M. Paterson, and A. Srinivasan, Contention resolution with constant expected delay, *J. ACM,* 47 (2000) 1048 - 1096.

[10] D.A. Grable, and A. Panconesi, Nearly optimal distributed edge coloring in $O(\log\log n)$ rounds, *Random Structures and Algorithms,* 10 (1997) 385 - 405.

[11] A.G. Greenberg, and S. Winograd, A lower bound on the time needed in the worst case to resolve conflicts deterministically in multiple access channels, *J. ACM,* 32 (1985) 589 - 596.

[12] J. Håstad, T. Leighton, and B. Rogoff, Analysis of backoff protocols for multiple access channels, *SIAM J. on Computing,* 25 (1996) 740 - 774.

[13] M. Hańćkowiak, M. Karoński, and A. Panconesi, A faster distributed algorithm for computing maximal matchings deterministically, in *Proc.,* 18*th ACM Symp. on Principles of Distributed Computing,* 1999, pp. 219 - 228.

[14] J. Komlós, and A.G. Greenberg, An asymptotically nonadaptive algorithm for conflict resolution in multiple-access channels, *IEEE Trans. on Information Theory,* 31 (1985) 303 - 306.

[15] E. Kushilevitz, and Y. Mansour, An $\Omega(D\log(N/D))$ lower bound for broadcast in radio networks, *SIAM J. on Computing,* 27 (1998) 702 - 712.

[16] S. Kutten, and D. Peleg, Fast distributed construction of small k-dominating sets and applications, *J. of Algorithms,* 28 (1998) 40 - 66.

[17] N. Linial, Locality in distributed graph algorithms, *SIAM J. on Computing,* 21 (1992) 193 - 201.

[18] Z. Lotker, B. Pat-Shamir, and D. Peleg, Distributed MST for constant diameter graphs, in *Proc.,* 20*th ACM Symp. on Principles of Distributed Computing,* 2001, pp. 63 - 71.

[19] C.U. Martel, Maximum finding on a multiple access broadcast network, *Information Processing Letters,* 52 (1994) 7 - 13.

[20] C.U. Martel, and T.P. Vayda, The complexity of selection resolution, conflict resolution, and maximum finding on multiple access channels, in *Proc. 3rd Int. Workshop on Parallel Computation and VLSI Theory,* 1988, pp. 401 - 410.

[21] D. Peleg, "Distributed Computing: A Locality-Sensitive Approach," SIAM, Philadelphia, 2000.

[22] D. Peleg, and V. Rubinovich, A near-tight lower bound on the time complexity of distributed MST construction, *SIAM J. on Computing,* 30 (2000) 1427 - 1442.

[23] P. Raghavan, and E. Upfal, Stochastic contention resolution with short delays, *SIAM J. on Computing,* 28 (1998) 709 - 719.

[24] D.E. Willard, Log-logarithmic selection resolution protocols in a multiple access channel, *SIAM J. on Computing,* 15 (1986) 468 - 477.

Bogdan S. Chlebus is with Department of Computer Science and Engineering, University of Colorado at Denver, Campus Box 109, P.O. Box 173364, Denver, CO 80217, USA, and with Instytut Informatyki, Uniwersytet Warszawski, Banacha 2, Warszawa 02-097, Poland.
E-mail: Bogdan.Chlebus@cudenver.edu

Karol Gołąb is with Instytut Informatyki, Uniwersytet Warszawski, Banacha 2, Warszawa 02-097, Poland.
E-mail: karolek@mimuw.edu.pl

Dariusz R. Kowalski is with Instytut Informatyki, Uniwersytet Warszawski, Banacha 2, Warszawa 02-097, Poland, and is currently a postdoc at Département d'Informatique, Université du Québec à Hull, Hull, Québec J8X 3X7, Canada.
E-mail: darek@mimuw.edu.pl