# An improved algorithm for radio broadcast

Michael Elkin *†    Guy Kortsarz‡

September 4, 2005

**Abstract**

We show that for every radio network $G = (V, E)$ and source $s \in V$, there exists a radio broadcast schedule for $G$ of length $Rad(G, s) + O(\sqrt{Rad(G, s)} \cdot \log^2 n) = O(Rad(G, s) + \log^4 n)$, where $Rad(G, s)$ is the radius of the radio network $G$ with respect to the source $s$. This result improves the previously best-known upper bound of $O(Rad(G, s) + \log^5 n)$ due to Gaber and Mansour [11].

For graphs with constant genus, particularly for *planar* graphs, we provide an even better upper bound of $Rad(G, S) + O(\sqrt{Rad(G, s)} \cdot \log n + \log^3 n) = O(Rad(G, s) + \log^3 n)$.

*Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel, elkinm@cs.bgu.ac.il

†This work was done in the Department of Computer Science, Yale University, New Haven, CT, USA

‡Computer Science department, Rutgers University, Camden, NJ, USA. Email: guyk@crab.rutgers.edu

# 1 Introduction

## 1.1 Definition and motivation

Consider a synchronous network of processors that communicate by transmitting messages to their neighbors, where a processor receives a message in a given step if and only if precisely one of its neighbors transmit. The instance of the radio broadcast problem, called *radio network*, is a pair $(G = (V, E), s)$, $s \in V$, where $G$ is an unweighted undirected $n$-vertex graph, and $s$ is a vertex, called *source*. The objective is to deliver one single message that the source $s$ generates to all the vertices of the graph $G$ using the smallest possible number of communication rounds. The prescription that tells each vertex when it should broadcast is called *schedule*; the *length* of the schedule is the number of rounds it uses, and it is called *admissible* if it informs all the vertices of the graph (see Section 2 for a formal definition).

From practical perspective, the interest in radio networks is usually motivated by their military significance, as well as by the growing importance of cellular and wireless communication (see, e.g., [16, 11, 4]). The radio broadcast is perhaps the most important communication primitive in radio networks, and, consequently, it has been intensively studied starting from mid-eighties [10, 13, 14, 8, 7, 9, 12, 11, 16, 1, 3, 4, 6, 5].

From theoretical perspective, the study of the radio broadcast problem provided researchers with a particularly convenient playground for the study of such broad and fundamental complexity-theoretic issues as the power and limitations of randomization, and of different models of distributed computation [4, 16, 14].

## 1.2 History and our results

One of the most important parameters of an instance $(G, s)$ of the radio broadcast problem is its *radius*, denoted $Rad(G, s)$ or shortly $Rad$, and defined as the maximum distance $dist_G(s, v)$ in the graph $G$ between the source $s$ and some vertex $v \in V$. It is easy to see that any admissible schedule for $(G, s)$ must have at least $Rad(G, s)$ rounds.

A basic open question in this area is to understand how large can be the minimum length of an admissible schedule for an instance $(G, s)$ in terms of $Rad(G, s)$ and $n$. Chlamtac and Weinstein [6] proved the first upper bound of $O(Rad \cdot \log^2 n)$, which was improved to $O(Rad \cdot \log n + \log^2 n)$ soon afterwards by Bar-Yehuda et al. [4].[1] Very recently Kowalski and Pelc [15] provided an alternative proof of this upper bound; their proof has certain *algorithmic* advantages over the one of [4]. Alon et al. [1] have shown a lower bound of $Rad + \Omega(\log^2 n)$. Finally, in SODA 1995 Gaber and Mansour [11] have published an upper bound of $O(Rad + \log^5 n)$, which is the state-of-the-art. See Table 1 for the summary of previous and our results.

[1] In this discussion we consider the relevant results from the *existential* rather than *algorithmic* perspective. How-

| Reference | Type of Bound | The bound |
|---|---|---|
| [6] | Upper | $O(Rad \cdot \log^2 n)$ |
| [4, 15] | Upper | $O(Rad \cdot \log n + \log^2 n)$ |
| [1] | Lower | $Rad + \Omega(\log^2 n)$ |
| [11] | Upper | $O(Rad + \log^5 n)$ |
| This paper | Upper (general graphs) | $Rad + O(\sqrt{Rad} \cdot \log^2 n) =$ $= O(Rad + \log^4 n)$ |
| This paper | Upper (planar graphs) | $Rad + O(\sqrt{Rad} \cdot \log n + \log^3 n) =$ $= O(Rad + \log^3 n)$ |

Table 1: The summary of previous and our *existential* results.

Despite the recent intensive research of the radio broadcast [10, 13, 14, 8, 7, 9, 12], no progress has been made in narrowing the gap between the upper bound of $O(Rad + \log^5 n)$ of Gaber and Mansour [11] and the lower bound of $Rad + \Omega(\log^2 n)$ of Alon et al. [1] during the last decade. In this paper we improve the upper bound and devise an algorithm that returns schedules of length $Rad + O(\sqrt{Rad} \cdot \log^2 n) = O(Rad + \log^4 n)$ for general graphs. For graphs with genus $g$ our algorithm returns schedules of length $Rad + O(\sqrt{Rad} \cdot g^{1/4} \cdot \log n + \log^3 n)$. Particularly, for *planar* graphs the bound is $Rad + O(\sqrt{Rad} \cdot \log n + \log^3 n) = O(Rad + \log^3 n)$. The upper bound of $O(Rad + \log^3 n)$ applies actually to a more general family of graphs with genus $g = O(\log^2 n)$.

**Remark:** Our result is the first upper bound of the form $Rad + f(n)$, where $f(n) = o(Rad)$ for a very wide range of parameters (specifically, for $Rad = \omega(\log^4 n)$).

### 1.3 Proof techniques

Our algorithm for constructing a radio broadcast schedule of length $Rad + O(\sqrt{Rad} \cdot \log^2 n)$ is based on the algorithm of Gaber and Mansour [11]. Both algorithms partition the graph into *layers* (or, *super-levels*, cast to the terminology of [11]), and construct an $(O(\log n), O(\log n))$-partition of every layer (see Sec. 2.6 for the definition) using the algorithm of [17] or [2]. One difference between the algorithms is that in our algorithm the layers are *overlapping* and not *disjoint* as in the algorithm of [11]. This difference, while complicating the analysis, enables us to reduce the additive term from $O(\log^5 n)$ to $O(\log^4 n)$. Both algorithms use the constructed partitions to control the data flow, and use similar procedures for this purpose. In [11] the scheduling of these procedures involves round-robin time sharing, which guarantees that executions of different procedures do not cause collisions. While very convenient for the analysis, this technique carries a price of increasing the length of the constructed schedule by a constant multiplicative factor. Our algorithm is geared to

ever, in fact, both [4] and [6] provide efficient algorithms for constructing schedules of corresponding length.

avoid this unnecessary multiplicative factor in order to achieve a *purely additive* approximation of the optimal schedule. To this end, our algorithm carefully combines different building blocks in such a manner that they cause no collision even though they employ *the same rounds*.

## 2 Preliminaries

We start with introducing some definitions and notations.

**Definition 2.1.** *The set of neighbors of a vertex $v$ in an unweighted undirected graph $G(V, E)$, denoted $\Gamma_G(v)$, is the set $\{u \in V \mid \{v, u\} \in E\}$. For a subset $X \subseteq V$, the set of neighbors of the vertex $v$ in the subset $X$, denoted $\Gamma_G(v, X)$ (or $\Gamma(v, X)$ when the graph $G$ is clear from the context), is the set $\{u \in X \mid \{v, u\} \in E\} = \Gamma_G(v) \cap X$.*

**Notation 2.2.** *1. For a positive integer number $n$, let $[n]$ denote the set $\{1, 2, \ldots, n\}$.*

*2. For an index $i \in \{1, 2\}$, let $\bar{i}$ denote the complementary index $3 - i$.*

**Definition 2.3.** *Let $G = (V, E)$ be an unweighted undirected graph, and $R \subseteq V$ be a subset of vertices. The set of vertices* informed *by $R$, denoted $Inf(R)$, is $Inf(R) = \{v \mid \exists! x \in R \;\; s.t. \;\; v \in \Gamma_G(x)\}$ (the notation $\exists! x$ stands for "there exists a unique $x$"). For a singleton set $R = \{x\}$, $Inf(R) = Inf(\{x\}) = Inf(x) = \Gamma_G(x)$.*

A sequence of vertex sets $\Pi = (R_1, R_2, \ldots, R_q)$, $q = 1, 2, \ldots$, is called a *radio broadcast schedule* (henceforth referred as a *schedule*) if $R_{i+1} \subseteq \bigcup_{j=1}^{i} Inf(R_j)$ for every $i = 1, 2, \ldots, q-1$. Intuitively, this condition means that the vertices that send a message in certain round have to be informed in one of the previous rounds.

The set of vertices *informed by a schedule* $\Pi$, denoted $Inf(\Pi)$, is $Inf(\Pi) = \bigcup_{R \in \Pi} Inf(R)$.

Given a graph $G = (V, E)$ and a vertex $s \in V$, a schedule $\Pi$ is *admissible* with respect to $(G, s)$ if $R_1 = \{s\}$ and $V = Inf(\Pi)$.

The *length* of the schedule $\Pi = (R_1, R_2, \ldots, R_q)$ is $|\Pi| = q$.

An instance of the *radio broadcast problem* $\mathcal{G}$ is a pair $(\bar{G} = (\bar{V}, \bar{E}), s)$, where $\bar{G}$ is a graph, and $s \in \bar{V}$ is a vertex. The goal is to compute an admissible schedule $\Pi$ of minimal length. The *value* of an instance $\mathcal{G}$ of the radio broadcast problem is the length of the shortest admissible schedule $\Pi$ for this instance.

For any schedule $\Pi = (R_1, R_2, \ldots, R_q)$, the set $R_i$ is called the *ith round* of $\Pi$, $i = 1, 2, \ldots, q$. If a vertex $v$ belongs to the set $R_i$, we say that $v$ is *scheduled to broadcast* on round $i$ by $\Pi$.

**Definition 2.4.** *For a rooted tree $(T = (V_T, E_T), s)$, $s \in V_T$, the* parent *of a vertex $v \in V_T$, $v \neq s$, denoted $parent_T(v)$, is the unique vertex $u$ such that $\{u, v\} \in E_T$, and $u$ belongs to the path between $s$ and $v$ in $T$. The vertex $v$ is also said to be a* child *of the vertex $u$ in the rooted tree $(T, s)$. For a subset $U \subseteq V_T$, let $Children_T(U)$ denote the set of all the children of vertices of the set $U$ in the tree $T$, i.e., $Children_T(U) = \{v \mid \exists u \in U \;\; s.t. \;\; u = parent_T(v)\}$.*

*For a rooted tree $(T, s)$, the* rank *of a vertex $v \in V_T$ is defined as follows. If $v$ is a leaf, its rank is 0. For an internal vertex $v$, let $R$ be the set of ranks of all the children of the vertex $v$, and let $r$ be the maximum value in $R$. If there is exactly one child $w$ of $v$ with rank $r$, then rank of the vertex $v$ is defined to be $r$; otherwise, it is defined to be $r+1$.*

*For a graph $G = (V, E)$, and a simple path $P$ connecting a pair $u, w \in V$ of vertices in $G$, the* length *of $P$ is the number of edges it contains. The (unweighted)* distance *between $u$ and $w$ in $G$, denoted $dist_G(u, w)$, is the length of the shortest path $P$ connecting $u$ and $w$ in $G$.*

*For a vertex $v \in V_T$, let $\ell_T(v)$ (or $\ell(v)$, when the tree $T$ can be deduced from the context) denote the* level *of the vertex $v$ in the rooted tree $(T, s)$, that is, the distance between the vertices $s$ and $v$ in $T$.*

*The* radius *of a rooted tree $(T, s)$, denoted $Rad(T, s)$, is* $\max\{\ell(v) \mid v \in V_T\}$. *For a graph $G = (V, E)$, and a designated vertex $s$, the* radius of the graph $G$ with respect to the vertex $s$, *denoted $Rad(G, s)$, is the radius of any breadth-first-search (henceforth, BFS) spanning tree of the graph $G$ rooted in the vertex $s$. The* diameter *of a graph $G = (V, E)$, denoted $Diam(G)$, is the maximal distance between a pair of vertices $u, w \in V$. For a subset $C \subseteq V$ of the vertices, the* diameter *of $C$, denoted $Diam_G(C)$, is the maximum distance in $G(C)$ between a pair of vertices in $C$, where $G(C) = (C, E(C))$, $E(C) = \{\{u, w\} \in E \mid u, w \in C\}$ is the* subgraph *of $G$* induced *by the subset $C$.*

**Lemma 2.5.** *For an $n$-vertex graph rooted tree $(T, s)$ and a vertex $v \in V_T$, $0 \le rank(v) \le \log n$.*

**Proof:** Follows directly from the definition. ∎

**Definition 2.6.** *For a graph $G = (V, E)$ and a pair of positive integer numbers $\kappa$ and $\alpha$, a $(\kappa, \alpha)$-*partition *(called also* Linial-Saks decomposition*) $\mathcal{C}$ of the graph $G$ is a collection of disjoint subsets of $V$, called* clusters, *that satisfy*

1. *For every cluster $C \in \mathcal{C}$, $Diam(C) \le \kappa$.*
2. *Consider the super-graph $\tilde{G} = (\tilde{V}, \tilde{E})$, $\tilde{V} = \mathcal{C}$, $\tilde{E} = \{\{C, C'\} \mid \exists v \in C,\ v' \in C' \ \ s.t. \ \ \{v, v'\} \in E\}$. The super-graph $\tilde{G}$ is $\alpha$-*colorable, *i.e., there exists a coloring $\chi : \mathcal{C} \to [\alpha]$ such that for every super-edge $\{C, C'\} \in \tilde{E}$, necessarily $\chi(C) \ne \chi(C')$.*

We remark that in our algorithms we use many $(\kappa, \alpha)$-partitions of *intersecting* sets. Consequently, although clusters of the same partition are disjoint, clusters of different partitions may (and actually do) intersect.

**Theorem 2.7.** *[17, 2] For every $n$-vertex graph $G = (V, E)$, there exists an $(O(\log n), O(\log n))$-partition $\mathcal{C}$ of the graph $G$. Furthermore, this partition is computable deterministically within time polynomial in $n$, and also, the coloring $\chi$ that is associated with the partition $\mathcal{C}$ and satisfies the property 2 of Definition 2.6 is also computable within the same time bounds.*

Let `LSPart` denote the procedure that forms an $(O(\log n), O(\log n))$-partition of the input graph

$G$. This procedure outputs the partition $\mathcal{C}$ and the coloring $\chi$.

**Definition 2.8.** *For an $n$-vertex rooted tree $(T = (V_T, E_T), s)$, a* semi-spanning path-forest *$F$ of $T$ is a collection of simple paths that satisfy*

1. *Every path $P \in F$, $P = (V_P, E_P)$, is contained in the tree $T$, i.e., $V_P \subseteq V_T$, $E_P \subseteq E_T$, and, furthermore, if $V_P = (v_1, v_2, \ldots, v_k)$ then either $v_1$ is an ancestor of $v_k$ in the tree $T$ or vice versa.*

2. *The paths of the forest $F$ are vertex-disjoint.*

3. *For every vertex $v \in V_T$, the unique path that connects the vertices $s$ and $v$ in the tree $T$ contains at most $\log n$ edges that do not belong to the set $\bigcup_{P \in F} E_P$ (such edges will be henceforth referred to as the* non-edges of the forest *$F$).*

**Lemma 2.9.** *For every $n$-vertex rooted tree $(T, s)$, a semi-spanning path-forest $F$ of the tree exists, and can be constructed within time polynomial in $n$.*

**Proof:** Consider the edge set $E' = \{\{v, w\} \in E_T \mid rank(v) = rank(w)\}$. Observe that the edge set $E'$ decomposes into a collection of vertex-disjoint paths. Denote this collection by $F$. It is easy to see that $F$ is a semi-spanning path-forest of the input tree. ∎

# 3 Preprocessing

Our algorithm consists of two major parts, the *preprocessing* algorithm, and the algorithm that actually constructs the schedule. While neither of the algorithms is executed offline, and running them efficiently is equally important, the preprocessing algorithm is somewhat detached from the specifics of the radio broadcast problem, and this motivates its name.

In this section we describe our preprocessing algorithm, which is essentially a variant of the analogous part of the algorithm due to Gaber and Mansour [11]. However, unlike the algorithm of [11] our algorithm forms *overlapping layers* and not disjoint ones, and this difference affects the implementation. In terms of its effect on the length of the resulting schedule, this different preprocessing algorithm enables us to reduce the additive term from $O(\log^5 n)$ in the result of [11] to $O(\log^4 n)$.

The input of the radio broadcast problem is a graph $G = (V, E)$, and a source vertex $s \in V$. Let $Rad$ denote $Rad(G, s)$. Assume that $Rad$ is $\Omega(\log^2 n)$. For graphs with smaller radius schedules of length $O(\log^3 n)$ are provided by the algorithms of Bar-Yehuda et al. [4] and Kowalski and Pelc [15]. (For a deterministic algorithm one should use the algorithm of [15], as the algorithm of [4] is randomized.)

The first step of the preprocessing algorithm is to construct a BFS spanning tree $T$ of the graph $G$, rooted in the source $s$. The second step of the algorithm is to cover the vertex set $V$ by $x$

subsets, called *layers*, where $x = 1, 2, \ldots$ is a positive integer parameter, $x \le Rad/3$. The specific value of $x$ will be determined later. Let $d$ denote $\lfloor Rad/x \rfloor$. Note that $d \ge 3$. We assume, without loss of generality, that $Rad$ is divisible by $x$. Otherwise, the graph can be modified so that its radius becomes divisible by $x$. This modification would increase the radius of the graph by at most an additive term of $x$, which will be accounted for at the end of our analysis.

For a vertex $v$, let the *level* of the vertex $v$, denoted $\ell(v)$ be the distance in $G$ between the source $s$ and the vertex $v$.

The next steps of the preprocessing algorithm are summarized below. The subroutine `FormTree` will be described in full detail later.

`Preprocressing`$(G, s, T, x)$

1. For every $j \in [x]$ set $V_j \leftarrow \{v \mid (j-1)d \le \ell(v) \le jd\}$.
2. For every $v \in V$ s.t. $\ell(v) \equiv 0 \pmod{d}$ form the *super-set* of $v$, $S(v)$, to be the set of all descendents of $v$ in $T$ that satisfy $\ell_T(w) \le \ell_T(v) + d$.
3. Form $\tilde{G} = (\tilde{V}, \tilde{E})$, with $\tilde{V} = \{S(v) \mid \ell(v) \equiv 0 \pmod{d}\}$, $\tilde{E} = \{\{S(v), S(v')\} \mid (S(v) \cap S(v') \neq \emptyset)$ or $(\exists w \in S(v), w' \in S(v')$ s.t. $\{w, w'\} \in E$ and $\ell(v) = \ell(v'))\}$.
4. For every $j \in [x]$, set $\tilde{V}_j \leftarrow \{S(v) \mid S(v) \in \tilde{V}, \ell(v) = (j-1)d\}$; $\tilde{E}_j \leftarrow \tilde{E}(\tilde{V}_j)$; $\tilde{G}_j = (\tilde{V}_j, \tilde{E}_j)$;
5. For every $j \in [x]$, $(\mathcal{C}_j, \chi_j) = \mathtt{LSPart}(\tilde{G}_j)$; $\mathcal{C} \leftarrow \bigcup_{j=1}^{x} \mathcal{C}_j$;
6. $(\tau, M) = \mathtt{FormTree}(\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_x)$, where $M = \{(u(C), w(C)) \mid C \in \mathcal{C}$ and $(u(C), w(C))$ is the messenger-representative pair of $C\}$ is the set containing the messenger-representative pair for every cluster $C \in \mathcal{C}$.
7. $\mathtt{return}(\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_x, \chi_1, \chi_2, \ldots, \chi_x, \tau, \{(u(C), w(C)) \mid C \in \mathcal{C}\})$.

For each index $j \in [x]$, the vertex set $V_j$ formed in step 1 will be henceforth referred to as the *jth layer* of the graph $G$. Note that $V = \bigcup_{j=1}^{x} V_x$, and that each pair $V_j$, $V_{j+1}$ for $j \in [x-1]$ of consecutive layers intersect in the set of all vertices of level $j \cdot d$. Also, as $d \ge 3$, non-consecutive layers do not intersect.

Consider the set $\tilde{V}_j$ of super-sets $S(v)$ with $\ell(v) = (j-1)d$. Note that

$$V_j \;=\; \bigcup \{S(v) \mid v \text{ s.t. } \ell(v) = (j-1)d\} \;,$$

and that this is a union of disjoint sets. Hence, the only edges in $\tilde{E}_j$ are between super-sets $S(v)$, $S(v')$ such that there exists an edge $(w, w') \in E$ with $w \in S(v)$, $w' \in S(v')$. However, by definition of $\tilde{E}$ (step 3), for a pair of vertices $v$, $v'$ with *different* levels $\ell(v) < \ell(v')$, $\{S(v), S(v')\} \in \tilde{E}$ holds if and only if $\ell(v') = \ell(v) + 1$ and $S(v) \cap S(v') = \{v'\}$.

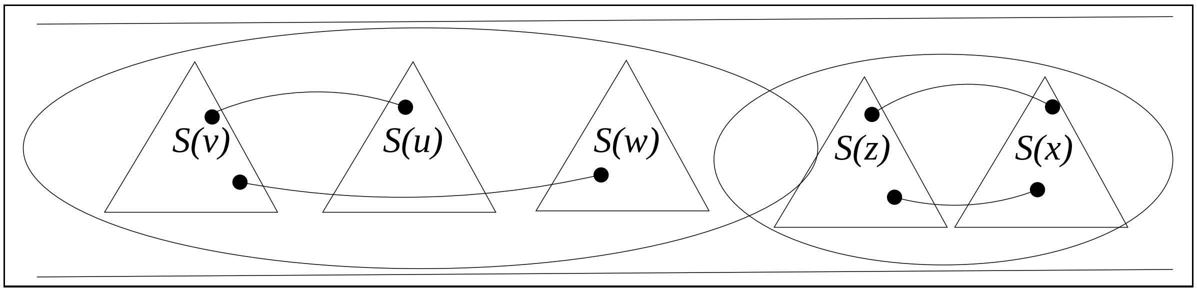


Figure 1: Forming clusters out of super-sets of a particular layer $j$. The super-sets are depicted by triangles, and the clusters are depicted by ovals.

In step 5, for each index $j \in [x]$, super-sets of the $j$th layer are merged into an $(O(\log n), O(\log n))$-partition $\mathcal{C}_j$. Observe that Theorem 2.7 guarantees that each cluster $C \in \mathcal{C}_j$ has diameter $O(\log n)$ with respect to the super-graph $\tilde{G}_j$, and so, the diameter of the cluster $C_G = \bigcup_{S(v)\in C} S(v)$ is $O(d \log n)$ (since $Diam_G(S(v)) \leq 2d$). With a slight abuse of notation we will henceforth denote by $C$ both the collection of subsets $S(v)$ that were used to form the cluster $C$, and the set $C_G = \bigcup_{S(v)\in C} S(v)$. See Figure 1 for illustration.

Note that every cluster $C \in \mathcal{C}_j$ contains one or more vertex $v$ of level $\ell_T(v) = (j-1)d$. Also, viewing clusters as collections of subsets $S(v)$, different clusters $C, C' \in \mathcal{C}_j$ cannot possibly intersect in a subset $S(v)$, because $\mathcal{C}_j$ is a *partition* of the super-graph $\tilde{G}_j$. Consequently, viewing clusters as collections of vertices of the graph $G$, since the sets $S(v) \in \tilde{V}_j$ are disjoint, the clusters $C, C' \in \mathcal{C}_j$ are disjoint as well. However, clusters $C_j \in \mathcal{C}_j$, $C_{j+1} \in \mathcal{C}_{j+1}$ may intersect in a subset $S = C_j \cap C_{j+1} \subseteq V$ of vertices of level $jd$. Since $d \geq 3$, it is easy to see that two clusters $C_j \in \mathcal{C}_j$, $C_{j'} \in \mathcal{C}_{j'}$ with $|j - j'| \geq 2$ cannot possibly intersect.

The partition $\mathcal{C}_j$ will be referred to as the *$j$th layer* of the super-graph $\tilde{G}$. Note that the first layer $\mathcal{C}_1$ of the super-graph $\tilde{G}$ contains only one single cluster $C(s) = S(s)$.

We next describe Procedure `FormTree`. The Procedure accepts as input the BFS tree $T$ of the input graph $G$, and all the $x$ layers $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_x$ of the super-graph $\tilde{G}$. It assigns *ranks* to all the clusters of the collection $\mathcal{C}$, constructs a BFS spanning tree $\tau$ rooted in the cluster $C(s)$ of the graph $\tilde{G}$, and chooses a pair of designated vertices, called the *messenger* and the *representative* for every cluster $C \in \mathcal{C}$. These three tasks are performed simultaneously in the following way.

`FormTree` initializes the tree $\tau$ to be an empty tree. Then, it assigns rank 0 to every cluster $C \in \mathcal{C}_x$. Next, in every cluster $C \in \mathcal{C}_x$, the algorithm picks an arbitrary leaf $w \in C$ of the tree $T$ and designates it to serve as the *representative* of the cluster $C$. In addition, in every cluster $C \in \mathcal{C}_x$, the algorithm designates the ancestor $u$ of the representative $w$ that has level $\ell_T(u) = (x-1) \cdot d$ to serve as the *messenger* of the cluster $C$. Then the algorithm initializes the set $\mathcal{US}$ of *uncovered* clusters to be equal to $\mathcal{C}_x$. (These clusters are uncovered with respect to the *next* iteration of the procedure.)

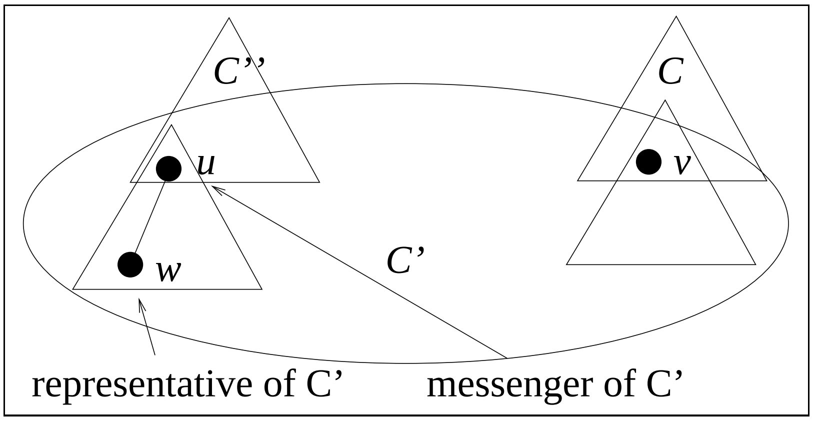


Figure 2: Even though $C'$ intersects $C$, the set $\mathcal{N} = \mathcal{N}(C)$ is empty. The reason is that the messenger $u$ of $C'$ belongs to a different cluster $C''$ of $\mathcal{C}_{j-1}$, and not to $C$.

After the algorithm finishes with the layer $j = 2, 3, \ldots, x$, it proceeds to layer $j-1$. Specifically, it passes over the clusters $C \in \mathcal{C}_{j-1}$ in an arbitrary order. For every cluster $C \in \mathcal{C}_{j-1}$ it does the following. Let $\mathcal{N} = \mathcal{N}(C) \subseteq \mathcal{US}$ be the subset of clusters $C'$ whose intersection with the cluster $C$ contains the messenger $u$ of the cluster $C'$.

If $\mathcal{N} = \emptyset$ then the algorithm picks a vertex $w \in C$ of maximum level $\ell(w)$, designates it to serve as the *representative* of $C$; then designates its unique ancestor $u \in C$ that satisfies $\ell(u) = (j-2)d$ to serve as the messenger of $C$; finally, it assigns $C$ rank 0.

It may happen that $\mathcal{N} = \emptyset$ but $C$ does intersect with a cluster $C' \in \mathcal{C}_j$. In this case the messenger of $C'$ belongs to some different cluster $C'' \in \mathcal{C}_{j-1}$ that also intersects $C'$. See Figure 2.

For each cluster $C' \in \mathcal{N}$, `FormTree` adds the super-edge $\{C, C'\}$ into the tree $\tau$, and sets $C$ to be the parent of $C'$ in the tree $\tau$. Let $\mathcal{R}$ be the set of ranks of all the children of the cluster $C$. Let $r = \max\{r' \mid r' \in \mathcal{R}\}$. If there exists exactly one cluster $C' \in \mathcal{N}$ with rank $r$, then the messenger $w$ of the cluster $C'$ becomes the *representative* of the cluster $C$. In this case the algorithm assigns the cluster $C$ rank $r$. Otherwise, an arbitrary cluster $C' \in \mathcal{N}$ is chosen, and its messenger $w$ becomes the representative of the cluster $C$. In this case `FormTree` assigns the cluster $C$ rank $r+1$. Next, `FormTree` designates the ancestor $u \in C$ of the representative $w$ of the cluster $C'$ that satisfies $\ell_T(u) = (j-2) \cdot d$ to serve as the messenger of the cluster $C$. Finally, in both cases, the set $\mathcal{N}$ is removed from the set $\mathcal{US}$ (i.e., $\mathcal{US} \leftarrow \mathcal{US} \setminus \mathcal{N}$), and the algorithm enters into another iteration of the internal loop by picking another cluster of layer $j-1$.

It is not hard to see that this procedure indeed builds a tree $\tau$ rooted in the cluster $C(s)$, and this tree spans the entire collection $\mathcal{C}$ of clusters. Furthermore, note that $\tau$ is a BFS spanning tree of the super-graph $\tilde{G}$. Note also that for every cluster $C \in \mathcal{C}$, the rank of the cluster $C$ in the tree $\tau$ is exactly equal to the rank that was assigned by this procedure. Also, the messenger of the cluster $C(s) = S(s)$ is inevitably the source $s$.

Let $\tilde{E}_{\mathcal{F}}$ be the subset of super-edges $\tilde{e} = \{C, C'\}$ of the tree $\tau$ that satisfy $rank(C) = rank(C')$. Note that the set $\tilde{E}_{\mathcal{F}}$ decomposes into a path-forest $\mathcal{F}$, and, furthermore, $\mathcal{F}$ is a semi-spanning path-forest. In addition, the path-forest $\mathcal{F}$ satisfies one more important property.

**Lemma 3.1.** *Consider a path $\mathcal{P} \in \mathcal{F}$, $\mathcal{P} = (C_1, C_2, \ldots, C_k)$, where $C_1$ is the ancestor of $C_k$ in the tree $\tau$. For each index $i \in [k]$, let $u_i$ and $w_i$ denote the messenger and the representative of the cluster $C_i$, respectively, and let $P_i$ denote the path between them in the tree $T$. Then the paths $P_i$ can be concatenated to form a single path $P = P_1 \cdot P_2 \cdot \ldots \cdot P_k$ in the tree $T$, the vertex $u_1$ is an ancestor of the vertex $w_k$ in the tree $T$, and $u_i = w_{i-1}$ for every index $i \in [k] \setminus \{1\}$.*

Note that the number of distinct subsets $S(v)$ is at most $n$, and therefore, $|\mathcal{C}| \le |\tilde{V}| \le n$, as well. Also, for every cluster $C \in \mathcal{C}$, the path between the root cluster $C(s)$ and the cluster $C$ in the tree $\tau$ contains at most $\log |\mathcal{C}| \le \log n$ non-edges of the forest $\mathcal{F}$.

Also, consider two clusters $C, C' \in \mathcal{C}_j$, for some index $j \in [x]$. Let $u$ (resp., $u'$) be the messenger of the cluster $C$ (resp., $C'$), and $w$ (resp., $w'$) be its representative. Recall that $\mathcal{C}_j$ is an $(O(\log n), O(\log n))$-partition of the graph $\tilde{G}_j = (\tilde{V}_j, \tilde{E}_j)$, and $C, C' \in \tilde{V}_j$. Let $\chi_j$ be the coloring associated with this partition, and suppose that $\chi_j(C) = \chi_j(C')$. It implies that there exists no edge $e = \{z, z'\} \in E$, with $z \in C$, $z' \in C'$. Recall also that $|\{\chi_j(C) \mid C \in \mathcal{C}_j\}| \le \alpha = O(\log n)$.

## 4 Constructing the schedule

Our algorithm for constructing the schedule (henceforth Procedure `ConstrSched`) accepts as input the description of the layers $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_x$, the colorings $\chi_1, \chi_2, \ldots, \chi_x$ of the layers, the tree $\tau$, the path-forest $\mathcal{F}$, and the messenger-representative pair of vertices for every cluster. All this is the output of the preprocessing algorithm.

We start with overviewing Procedure `ConstrSched`. There are a few basic ideas. The first one is that the message can be propagated very efficiently along a path $P$ that belongs to the path-forest $\mathcal{F}$. Moreover, since the clusters on each layer are colored with a small number of colors, we will show that it is possible to propagate messages efficiently over different paths $P$ of the path-forest $\mathcal{F}$ in parallel. Some delay is, however, incurred here, but we will show that this delay contributes only an *additive term* to the total length of the schedule (rather than a *multiplicative* factor). We will also show that this contribution (in terms of the number of rounds) is of the same order of magnitude as the maximum number of colors that are used for coloring one of the layers, that is, $O(\log n)$.

If the path-forest $\mathcal{F}$ were to cover the entire cluster tree $\tau$, the description above would suffice for constructing an admissible schedule. Even though it is not the case, fortunately, in some sense, the forest $\mathcal{F}$ does cover "almost" the entire cluster tree. More specifically, for each cluster $C$ of this tree, the path between the root cluster $C(s)$ and $C$ contains only a small number ($O(\log n)$) of non-edges of $\mathcal{F}$. On these non-edges, indeed, greater delays are incurred, but we show that since there are not many of them, their overall contribution to the length of the resulting schedule is not large.

Procedure `ConstrSched` uses two major building blocks, namely, Procedures `Vert_Broadcast` and `Hor_Broadcast`. These procedures are described in Sections 5 and 6. The two procedures are analogous to Procedures *Broadcast_Through* and *Broadcast_All* of [11]. However, our algorithm does not have an analogue to Procedure *Superlevel_to_Superlevel* of [11], because our preprocessing algorithm forms *overlapping layers*, and not disjoint ones as the algorithm of [11] does. Therefore, in our algorithm the delivery of the message between consequent layers is done via the boundary vertices that participate in both layers.

We remark that while our preprocessing algorithm and the analogous part of the algorithm of [11] have a similar nature, our algorithm for constructing the schedule (Procedure `ConstrSched`) is substantially different from the analogous part of the algorithm [11], which is called there the "main algorithm". The difference concerns mostly the way that different building blocks are combined into a single schedule. The main algorithm of [11] uses a round-robin technique of time sharing for running different procedures, and this way it ensures that there is no collisions between the executions of different procedures. This technique gives rise to a relatively simple analysis, but it carries a price of increasing the length of the schedule by a constant multiplicative factor. Our algorithm is geared to avoid this unnecessary multiplicative factor in order to achieve a *purely additive* approximation of the optimal schedule. To this end, our algorithm carefully combines different building blocks in such a manner that they cause no collision even though they employ *the same rounds.*

# 5 Procedure `Vert_Broadcast`

The simplest setting in which Procedure `Vert_Broadcast` can be invoked is on a cluster $C \in \mathcal{C}$, assuming that the messenger $u$ of $C$ is informed. In this case the procedure propagates the message along the shortest path in the cluster $C$ between the messenger $u$ and the representative $w$ of $C$. Observe that this process of propagating the message results in a schedule of length at most $d$.

The more elaborated, and by far more common, scenario in which Procedure `Vert_Broadcast` is invoked is as follows. Let $\mathcal{S} = \{C^{(1)}, C^{(2)}, \ldots, C^{(k)}\} \subseteq \mathcal{C}_j$ be a collection of clusters of layer $j$, for some index $j \in [x]$. Let $u^{(i)}, w^{(i)} \in C^{(i)}$ be the messenger and the representative of the cluster $C^{(i)}$, respectively, for each index $i \in [k]$. Suppose that all the messengers $u^{(1)}, u^{(2)}, \ldots, u^{(k)}$ are informed. Procedure `Vert_Broadcast`, if it is invoked on $\mathcal{S}$, needs to propagate the message to the representatives $w^{(1)}, w^{(2)}, \ldots, w^{(k)}$. For each index $i \in [k]$, let $P^{(i)}$ denote the shortest path between the messenger $u^{(i)}$ and the representative $w^{(i)}$. This path $P^{(i)}$, denoted also as $P(C^{(i)})$, will be henceforth referred to as the *chosen path* of the cluster $C^{(i)}$. Let $\chi_j$ be the coloring associated with layer $j$ of the collection $\mathcal{C}$. Let $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_\alpha$ be the partition of the subset $\mathcal{S}$ into a disjoint union of color classes, that is, $\bigcup_{p=1}^{\alpha} \mathcal{S}_p = \mathcal{S}$, with $\mathcal{S}_p = \{C \in \mathcal{S} \mid \chi_j(C) = p\}$ for each index $p \in [\alpha]$. Note that some color classes $\mathcal{S}_p$ may be empty. Observe that for an index $p \in [\alpha]$, and for a pair of distinct clusters $C, C' \in \mathcal{S}_p$, the edge set $E$ of the graph $G$ contains no edge $\{z, z'\}$ with $z \in C$ and $z' \in C'$. In particular, the same property holds for the vertex sets $V_P$ and $V_{P'}$ of the paths $P$ and $P'$ respectively. Recall that the number $\alpha$ of color classes is $O(\log n)$.

For a path $P = (u = u_1, u_2, \ldots, u_{d'} = w)$, $d' \leq d$, between the messenger $u$ and the representative $w$ of some cluster $C$, and an index $\ell \in [d']$, the vertex $u_\ell$ is called the *$\ell$th vertex* of the path $P$.

We next describe the structure of the schedule $\Pi_{VL}$ that Procedure `Vert_Broadcast` forms whenever it is invoked in this scenario. On the first round of this schedule, the first vertices of the paths $P(C)$ for different clusters $C \in \mathcal{S}_1$ broadcast. (In a situation like this, when the broadcast of the vertex $u_1$ was originated as part of processing the cluster $C$, we say that the vertex $u_1$ *$C$-broadcasts*.)

More generally, for a round $\ell \in [d]$, on the $\ell$th round of the schedule $\Pi_{VL}$, the $\ell$th vertices $u_\ell$ of the chosen paths $P = P(C)$ broadcast, for every cluster $C \in \mathcal{S}_1$. (If one or more of these paths are not long enough to have the $\ell$th vertex, the set of broadcasting vertices on round $\ell$ will not contain a vertex from this specific path; if none of these paths contains $\ell$th vertex, the round will be replaced by a *void* (that is, empty) round.) In addition, for a round $\ell = 4, 5, \ldots, d+3$, on the $\ell$th round of the schedule $\Pi_{VL}$, the $(\ell-3)$rd vertices $u_{\ell-3}$ of the chosen paths $P(C)$ broadcast, for all the clusters $C \in \mathcal{S}_2$. More generally, for an index $p \in [\alpha]$, a round $\ell = 3(p-1)+1, 3(p-1)+2, \ldots, 3(p-1)+d$, on the $\ell$th round of this schedule the $(\ell - 3(p-1))$th vertices $u_{\ell-3(p-1)}$ of the chosen paths $P(C)$ broadcast, for all the clusters $C \in \mathcal{S}_p$.

Note that this way the broadcast in clusters of $\mathcal{S}_2$ starts three rounds after the broadcast starts in clusters of $\mathcal{S}_1$, and broadcast in clusters of $\mathcal{S}_3$ starts three rounds after it starts in clusters of $\mathcal{S}_2$, etc.. It is not hard to see (and we will soon argue it formally) that since the messengers of all these clusters are at the same distance from $s$, this small delay prevents collisions between different broadcasts.

This completes the description of the schedule $\Pi_{VL}$. We next turn to its analysis.

Observe that the broadcast in clusters $C$ and $C'$ of the same color cannot interfere with each other, because, as we have seen, there are no edges in $G$ between vertices of $V(P(C))$ and vertices vertices $V(P(C'))$.

Consider a pair of clusters $C, C' \in \mathcal{C}_j$ with $\chi_j(C) = p$, $\chi_j(C') = q$, and $p \neq q$. Consider some round $\ell = 1, 2, \ldots, d + 3(\alpha - 1)$. Let $v \in C$, $v' \in C'$ be a pair of vertices that are scheduled to $C$-broadcast and $C'$-broadcast on round $\ell$, respectively. It follows that $v$ is the $(\ell - 3(p-1))$th vertex of the path $P(C)$, and that $v'$ is the $(\ell - 3(q-1))$th vertex of the path $P(C')$. Recall that $C, C' \in \mathcal{C}_j$. It follows that $\ell_T(v) = (j-1)d + \ell - 3(p-1)$, and $\ell_T(v') = (j-1)d + \ell - 3(q-1)$, and so, since $p \neq q$ are both integer, $|\ell_T(v) - \ell_T(v')| \geq 3$. In other words, $|dist_G(s,v) - dist_G(s,v')| \geq 3$, and so $dist_G(v,v') \geq 3$. Hence, the broadcasts of the vertices $v$ and $v'$ do not cause a collision, as required.

Hence, executions of Procedure `Vert_Broadcast` do not cause collisions between vertices that participate in the same execution. The length of the schedule $\Pi_{VL}$ is $\tilde{d} = d + 3(\alpha - 1)$. By padding the schedules with as many void rounds as required we guarantee that every execution of Procedure `Vert_Broadcast` forms a schedule of length precisely $\tilde{d}$. This homogeneity of the lengths of schedules is crucial for our analysis. Note also that our description of Procedure `Vert_Broadcast` assumes that all the clusters in which the broadcast should be conducted are known in advance, and no cluster ever joins "on the fly". In our analysis we will show that our algorithm invokes Procedure `Vert_Broadcast` in such a way that this property is indeed satisfied.

We also remark that all the procedures that our algorithm employs accept as input an additional input parameter, called the *starting round*. This parameter is a positive integer number that determines the index of the first non-empty round of the schedule that the procedure should construct. In other words, if according to the above description Procedure `Vert_Broadcast` is supposed to return a schedule $\Pi = (R_1, R_2, \ldots, R_t)$ on a certain input, then whenever it is invoked on the same input with a starting round parameter $t'$ it would return the schedule $\epsilon_{t'-1} \cdot \Pi$, where $\epsilon_{t'-1}$ stands for a schedule that contains $t' - 1$ void rounds, and $\cdot$ stands for concatenation. For an execution $\varphi$ of one of the procedures that are used by our algorithm, its starting round will be denoted by $I(\varphi)$.

# 6 Procedure `Hor_Broadcast`

Consider a collection of clusters $\mathcal{S} = \{C^{(1)}, C^{(2)}, \ldots, C^{(k)}\} \subseteq \mathcal{C}_j$, where $j \in [x]$ is an index. Suppose that all the representatives $w^{(1)}, w^{(2)}, \ldots, w^{(k)}$ of these clusters are informed, and the goal is to inform all the vertices of the set $\bigcup_{C \in \mathcal{S}} C$. In other words, while Procedure `Vert_Broadcast` propagates the message to vertices of the next layer (with respect to the layer of the collection on which it was invoked), Procedure `Hor_Broadcast` distributes the message among the vertices of the current layer. This is done using Algorithm `Anonymous_Broadcast` due to Bar-Yehuda et al. [4]. Specifically, we use the following result.

**Theorem 6.1.** *[4] Given an $n$-vertex graph $\hat{G} = (\hat{V}, \hat{E})$, and a subset $\hat{U} \subseteq \hat{V}$ of informed vertices, there exists a schedule of length $K \cdot (D \log n + \log^2 n)$ that delivers the message to all the vertices of the graph, where $K$ is a universal constant and $D = \max_{v \in \hat{V}} \min_{u \in \hat{U}} dist_G(u,v)$.*

We remark that for a cluster $C \in \mathcal{S}$, its diameter in the super-graph $\tilde{G}$ is $O(\log n)$, and each vertex $S(v)$ from which the super-vertices of $\tilde{G}$ were formed is itself a subset of diameter at most $2d$ (in the graph $G$). Hence, $D \leq \max\{Diam_G(C) \mid C \in \mathcal{S}\} = O(d \log n)$, and so the length of the schedule that is constructed by Algorithm `Anonymous_Broadcast` is $O(d \log^2 n)$. Let $c$ denote the universal constant hidden by the $O$-notation. For the sake of homogeneity of the lengths of schedules constructed by Procedure `Hor_Broadcast`, whenever the resulting schedule is of length $L < cd\log^2 n$, it is padded with $cd\log^2 n - L$ empty rounds. By Theorem 6.1, this schedule informs all the vertices of the set $\bigcup_{C\in\mathcal{S}} C$.

To prevent collisions between simultaneous executions of Procedure `Hor_Broadcast` in adjacent layers we modify the procedure in the following way. Whenever the input collection of clusters $\mathcal{S}$ is contained in an *even* (respectively, *odd*) layer $\mathcal{C}_j$ (i.e., its index $j$ is even (resp., odd)), the procedure constructs a schedule that employs only even (resp., odd) rounds. The odd (resp., even) rounds are left empty. This raises the length of the schedules constructed by Procedure `Hor_Broadcast` to $2c \cdot d\log n$. Let $a$ denote the constant $2c$. (This is an example of using the round-robin time sharing technique. In the algorithm of [11] it is used also for avoiding collisions between executions of different procedures, and between different executions of the same procedure that accept as input collections of clusters of the same layer.)

# 7 Procedure `Period`

In this section we combine Procedures `Vert_Broadcast` and `Hor_Broadcast` described in the previous sections, and describe Procedure `Period`, which is the major component of our algorithm for constructing the schedule.

This procedure is invoked in the same scenario as Procedure `Vert_Broadcast`, i.e., when it is given a collection $\mathcal{S} \subseteq \mathcal{C}_j$ of clusters of layer $j$, for some index $j \in [x]$, whose messengers are all informed. Let $I$ denote its starting round parameter. It also uses the cluster tree $\tau$, and the path-forest $\mathcal{F}$ that can be considered as global variables.

The formal description of Procedure `Period` follows. Right below it follows its verbal description. (Procedures `SubPeriod` and `SchedUnion` are described below.)

`Period`$(\mathcal{S}, I)$;

1. If $\mathcal{S} = \emptyset$ return $\emptyset$;
2. $\Pi_0 \leftarrow$ `SubPeriod`$(\mathcal{S}, I)$;
3. $\mathcal{S}'' \leftarrow \{C \in \mathcal{S} \mid \exists C' \text{ s.t. } C = \mathit{parent}_\tau(C') \text{ and } \{C, C'\} \in \mathcal{F}\}$;
4. $\mathcal{S}' \leftarrow \{C' \mid \mathit{parent}_\tau(C') \in \mathcal{S}''\}$; $\Pi_1 \leftarrow$ `Period`$(\mathcal{S}', I + \tilde{d})$;
5. $\hat{\mathcal{S}} \leftarrow \mathit{Children}_\tau(\mathcal{S}) \setminus \mathcal{S}'$; $\Pi_2 \leftarrow$ `Period`$(\hat{\mathcal{S}}, I + 4\tilde{d} + ad \cdot \log^2 n)$;
6. $\Pi \leftarrow$ `SchedUnion`$(\{\Pi_0, \Pi_1, \Pi_2\})$;
7. return $(\Pi)$;

`Subperiod`$(\mathcal{S}, I)$;

1. $\Pi_{VL} \leftarrow$ `Vertical_Broadcast`$(\mathcal{S}, I)$;
2. $\Pi_{HL} \leftarrow$ `Horizontal_Broadcast`$(\mathcal{S}, I + 2\tilde{d})$;
3. return$(\Pi_{VL} \cdot \epsilon_{\tilde{d}} \cdot \Pi_{HL} \cdot \epsilon_{2\tilde{d}})$;

Procedure `Period` starts with invoking Procedure `Vert_Broadcast` on the collection $\mathcal{S}$ with the same starting round parameter $I$. (Step 1 of Procedure `SubPeriod`.) Let $\Pi_{VL}$ denote the resulting schedule. Then it invokes Procedure `Hor_Broadcast` on the same collection $\mathcal{S}$, this time with starting round parameter $I + 2\tilde{d}$. (Step 2 of Procedure `SubPeriod`.) Let $\Pi_{HL}$ denote the resulting schedule. (Note that this value the starting round parameter of the invocation of `Hor_Broadcast` ensures that $\Pi_{VL}$ ends at least $\tilde{d}$ rounds before $\Pi_{HL}$ starts. Note also that after the termination of $\Pi_{HL}$ all the representatives of the clusters $C \in \mathcal{S}$ are informed.) Then, the schedules $\Pi_{VL}$ and $\Pi_{HL}$ are used to form the schedule $\Pi_0$. (Step 3 of Procedure `SubPeriod` and Step 2 of Procedure `Period`.) Specifically, the first $\tilde{d}$ rounds of $\Pi_0$ are filled by $\Pi_{VL}$, then $\tilde{d}$ void rounds follow, the next $ad\log^2 n$ rounds are filled by $\Pi_{HL}$, and, finally, $\Pi_0$ is padded with $2\tilde{d}$ void rounds. Hence $|\Pi_0| = 4\tilde{d} + ad\log^2 n$. Observe that $\Pi_0$ delivers the message from the set of the messengers of the clusters $C \in \mathcal{S}$ to all the vertices of the set $\bigcup_{C\in\mathcal{S}} C$.

In addition, Procedure `Period` invokes itself recursively twice, as will be described below. Let $\mathcal{S}'' \subseteq \mathcal{S}$ be a collection of clusters $C \in \mathcal{C}_j$ that have a child (with respect to the tree $\tau$) $C'$ such that the super-edge $\{C, C'\}$ belongs to the path-forest $\mathcal{F}$. Let $\mathcal{S}' = \{C' \mid parent_\tau(C') \in \mathcal{S}''\}$. These two sets are computed in steps 3 and 4. Note that as $\mathcal{S} \subseteq \mathcal{C}_j$, it follows that $\mathcal{S}' \subseteq \mathcal{C}_{j+1}$. Let $W''$ be the set of representatives of the clusters from the collection $\mathcal{S}''$. Observe that, by construction, each representative $w \in W''$ of a cluster $C \in \mathcal{S}''$ is the messenger of the cluster $C' \in \mathcal{S}'$ that satisfies $parent_\tau(C') = C$. It follows that after the schedule $\Pi_{VL}$ that is returned by the invocation of Procedure `Vert_Broadcast` terminates, all the messengers of the clusters $C' \in \mathcal{S}'$ are informed.

Procedure `Period` invokes itself recursively on the collection $\mathcal{S}' \subseteq \mathcal{C}_{j+1}$, and sets the starting round parameter of this recursive invocation to be $I + \tilde{d}$. Note that this choice of starting round parameter ensures that the schedule constructed by this recursive invocation starts after the schedule $\Pi_{VL}$ terminates.

Let $\hat{\mathcal{S}} \subseteq \mathcal{C}_{j+1}$ be the collection of clusters of layer $j+1$ that do not belong to the collection $\mathcal{S}'$, and whose messengers are informed by the schedule that is constructed by the execution of Procedure `Hor_Broadcast` on input $\mathcal{S}$. The last step of Procedure `Period` is to invoke itself recursively on the collection $\hat{\mathcal{S}} \subseteq \mathcal{C}_{j+1}$ with starting round parameter $I + 4\tilde{d} + a \cdot d\log^2 n$. (The rationale here is that the schedule $\Pi_{HL}$ terminates before the round $I + 4\tilde{d} + a \cdot d\log^2 n$, and so, all the messengers of clusters of $\hat{\mathcal{S}}$ are already informed at this point.)

Note that one of the collections $\hat{\mathcal{S}}$, $\mathcal{S}'$, or both of them, could be empty. This would lead to an invocation of Procedure `Period` on an empty collection; in this case it returns an empty schedule. This is, in fact, the termination condition of the recursion. (Step 1 of Procedure `Period`.) See Figure 3 for the schematic description of Procedure `Period`.

After the recursive invocations are over, the original execution of Procedure `Period` has three schedules $\Pi_0$, $\Pi_1$ and $\Pi_2$. The schedule $\Pi_0$ is obtained by processing the collection $\mathcal{S}$ directly, that is, invoking on it Procedures `Vert_Broadcast` and `Hor_Broadcast` and concatenating them in the way that was described above (along with introducing the void rounds). The schedules $\Pi_1$ and $\Pi_2$ are returned by the first and the second recursive invocations of Procedure `Period`, respectively (on collections $\mathcal{S}'$ and $\hat{\mathcal{S}}$, respectively). These three schedules are unified into a single schedule $\Pi$ that the execution returns via an invocation of Procedure `SchedUnion` (step 6 of Procedure `Period`). This procedure operates in the following way. Let $L = \max\{|\Pi_0|, |\Pi_1|, |\Pi_2|\}$, and let $L_i = |\Pi_i|$, for $i \in \{0, 1, 2\}$. For each index $i \in \{0, 1, 2\}$ such that $L_i < L$, pad the schedule $\Pi_i$ with $L - L_i$ void rounds. For $i \in \{0, 1, 2\}$, let $\Pi'_i$ denote the obtained schedules. Now set $\Pi$ to be the *round-wise* union of the schedules $\Pi'_0$, $\Pi'_1$, and $\Pi'_2$. I.e., let $\Pi'_i = (R_1^{(i)}, R_2^{(i)}, \ldots, R_L^{(i)})$, $i \in \{0, 1, 2\}$. Then $\Pi = (\bigcup_{i=0}^2 R_1^{(i)}, \bigcup_{i=0}^2 R_2^{(i)}, \ldots, \bigcup_{i=0}^2 R_L^{(i)})$ is the schedule returned by the execution. (Henceforth we refer to this operation of unifying schedules of not necessarily equal lengths as *schedule-union*. In this terminology, the schedule $\Pi$ is just the schedule-union of schedules $\Pi_0$, $\Pi_1$ and $\Pi_2$. Also, the schedule $\Pi_0$ is the schedule-union of the schedules $\Pi_{VL}$ and $\Pi_{HL}$.)

The formal description of Procedure `SchedUnion` follows.

`SchedUnion`$(\{\Pi_1, \Pi_2, \ldots, \Pi_j\})$;

1. $L \leftarrow \max\{|\Pi_1|, |\Pi_2|, \ldots, |\Pi_j|\}$;
2. For every $i = 1, \ldots, j$ do
   $L_i \leftarrow |\Pi_i|$; $\Pi'_i \leftarrow \Pi_i \cdot \epsilon_{L-L_i}$;
3. For every $\ell = 1, 2, \ldots, L$ do
   $R_\ell \leftarrow \bigcup_{i=1}^j R_\ell^{(i)}$,
   where $\Pi'_i = (R_1^{(i)}, R_2^{(i)}, \ldots, R_L^{(i)})$;
4. return$(\Pi = (R_1, R_2, \ldots, R_L))$;

# 8 The Combined Algorithm

We start with providing some intuition for the rest of the algorithm.

Consider the algorithm that just invokes Procedure `Period` on input $\{C(s)\}$ with starting round parameter 1. Intuitively, this can be seen as the "first approximation" of our algorithm. However,

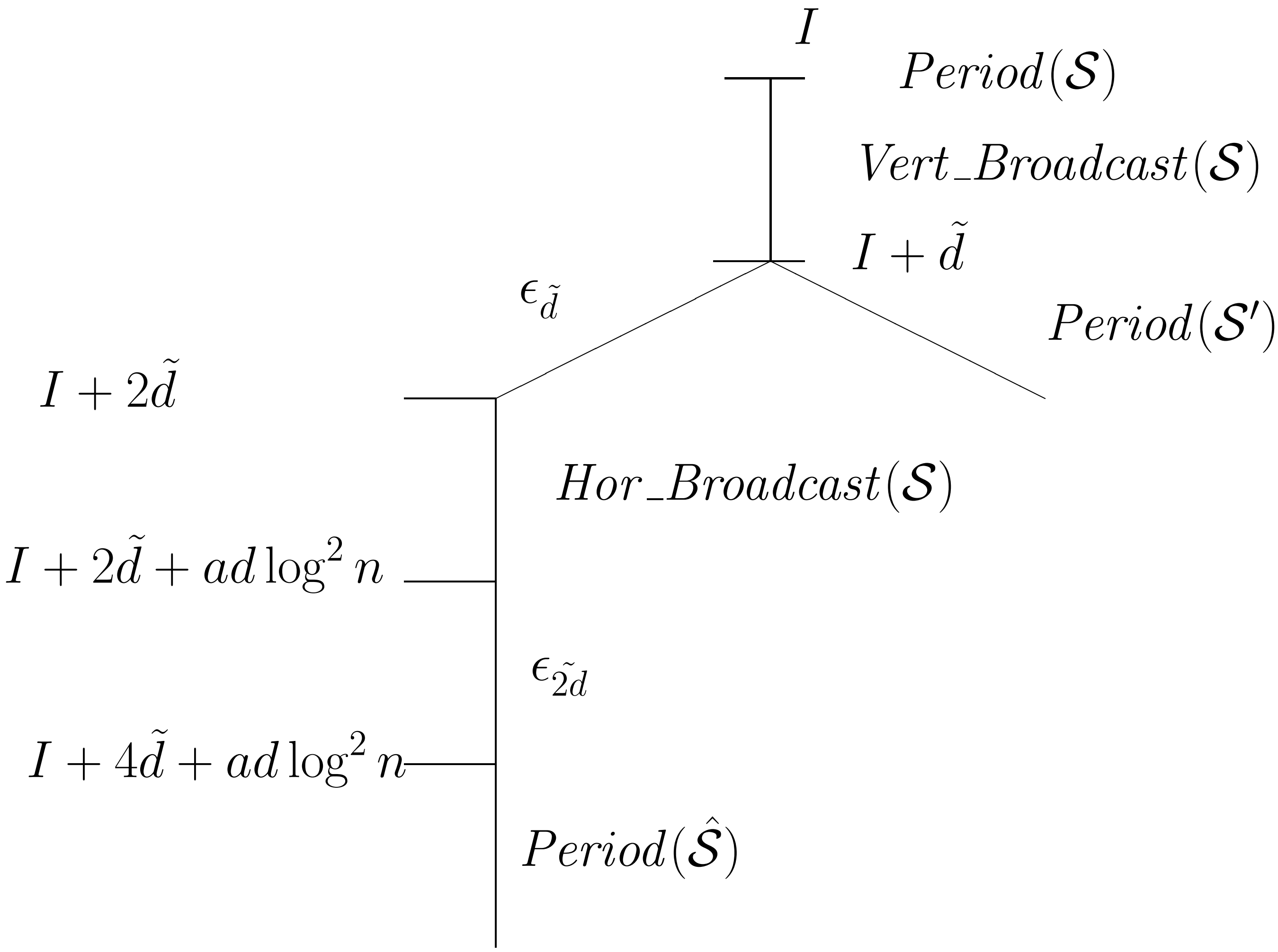


Figure 3: This scheme represents the synchronization between different schedules formed by the invocation `Period`($\mathcal{S}$). The time axis is depicted by vertical lines, and two diagonal lines are used to represent schedules that occupy the same rounds. Going from the top part of the scheme down: $I$ is the starting round of the schedule formed by invocations `Period`($\mathcal{S}$) and `Vert_Broadcast`($\mathcal{S}$). $I+\tilde{d}$ is the starting round of the recursive invocation `Period`($\mathcal{S}'$). $I+2\tilde{d}$ is the starting round of the schedule formed by `Hor_Broadcast`($\mathcal{S}$), and $I+2\tilde{d}+ad\log^2 n$ is a round by which this schedule is guaranteed to terminate. Finally, $I+4\tilde{d}+ad\log^2 n$ is the starting round of the recursive invocation `Period`($\hat{\mathcal{S}}$).

this simplified version of the algorithm is problematic for reasons that we are going to discuss next.

The invocation `Period`$(\{C(s)\}, 1)$ results in multiple invocations of Procedure `SubPeriod` on many various input collections with various starting round parameters. We will show in Section 9 that to a large extent, these invocations do not "interfere" with each other. (Intuitively, two invocations *interfere* with each other if the schedule produced by one of them prevents from the schedule produced by another to "fulfill its mission", that is, to inform a particular set of vertices that this invocation is supposed to inform. The formal definition of this notion can be found in Section 9 (Definition 9.3), but it is not necessary for purposes of this discussion.)

However, some invocations do interfere with each other, and we will see that those are precisely pairs of invocations of the type `SubPeriod`$(\mathcal{S}_1, I_1)$, `SubPeriod`$(\mathcal{S}_2, I_2)$ with $\mathcal{S}_1$ $\mathcal{S}_2$ being subsets of one particular layer $\mathcal{C}_j$, $j \in [x]$, and whose starting round parameters $I_1$ and $I_2$ are precisely *equal* (that is, $I_1 = I_2$). To resolve this problem we modify the algorithm such that each time it "wants" to invoke `SubPeriod` separately on two such subsets $\mathcal{S}_1$ and $\mathcal{S}_2$, it would instead invoke `SubPeriod` on $\mathcal{S}_1 \cup \mathcal{S}_2$ with the same starting round parameter $I_1 = I_2$. For implementing this modification we build the tree of recursive invocations of `Period`$(\mathcal{C}_1, 1)$, and on each layer $j \in [x]$ identify all the problematic recursive invocations, and merge them along the described above lines.

In the rest of this section we formalize this intuition.

Consider the invocation $\varphi_s$ of Procedure `Period` on the input collection that contains only the single cluster $C(s) \in \mathcal{C}_1$ with starting round parameter equal to 1. We next modify the schedule $\Pi$ that is obtained by this invocation to construct the resulting schedule $\Pi^*$ of our algorithm.

Let $\Phi$ be the set of all the recursive invocations of Procedure `Period` that were originated directly or indirectly by the invocation $\varphi_s$. Let $E_\Phi$ be the set of ordered pairs of invocations $(\varphi, \varphi') \in \Phi^2$, $\varphi \neq \varphi'$, such that the invocation $\varphi$ recursively invokes the invocation $\varphi'$. Observe that the graph $T_\Phi = (\Phi, E_\Phi)$ is a directed binary tree with all the arcs oriented from the root towards the leaves. Consider an internal vertex (that is, not a leaf) $\varphi \in \Phi$ of this tree. This vertex corresponds to an invocation of Procedure `Period` on an input collection $\mathcal{S} \subseteq \mathcal{C}_j$ of clusters, for some index $j \in [x-1]$. Since $\varphi$ is not a leaf of the tree $T_\Phi$, it follows that $\varphi$ has two children in $T_\Phi$. These children correspond to the invocations $\varphi'$ and $\hat{\varphi}$ of Procedure `Period` on input collections $\mathcal{S}'$ and $\hat{\mathcal{S}}$, respectively. (See steps 4 and 5 of Procedure `Period`.) The invocation $\varphi'$ (resp., $\hat{\varphi}$) will be referred as the *first* (resp., *second*) *child-invocation of the invocation* $\varphi$.

For each $\varphi \in \Phi$, let $\mathcal{S}(\varphi)$ (respectively, $I(\varphi)$) be the input collection of clusters (resp., the input starting round parameter) of the invocation $\varphi$. For $j \in [x]$, let $\Phi_j$ be the set of vertices $\varphi$ of the tree $T_\Phi$ that have layer $j$, i.e., satisfy $\ell_{T_\Phi}(\varphi) = j$.

We also need a piece of notation: we use the symbol $\bigsqcup$ to denote a disjoint union of sets, i.e., $\mathcal{Z} = \bigsqcup_{j=1}^{\infty} \mathcal{Z}_j$ means that $\mathcal{Z} = \bigcup_{j=1}^{\infty} \mathcal{Z}_j$, and that $\mathcal{Z}_j \cap \mathcal{Z}_k = \emptyset$ for all $j \neq k$.

**Lemma 8.1.** *Let $\mathcal{C}$ be the collection of clusters computed by the preprocessing algorithm. Then*

$\mathcal{C} = \bigsqcup_{\varphi \in \Phi} \mathcal{S}(\varphi)$.

**Proof:** We will prove by induction on $j \in [x]$ that $\mathcal{C}_j = \bigsqcup_{\varphi \in \Phi_j} \mathcal{S}(\varphi)$. This will imply the statement of the lemma as $\mathcal{C} = \bigsqcup_{j=1}^{x} \mathcal{C}_j$, and $\Phi = \bigsqcup_{j=1}^{x} \Phi_j$.

Induction base ($j = 1$): Obvious, since $\mathcal{C}_1 = \{\mathcal{C}(s)\}$, and $\Phi_1 = \{\varphi_s\}$.

Induction step: Let $j \in [x-1]$.

By the induction hypothesis, $\mathcal{C}_j = \bigsqcup_{\varphi \in \Phi_j} \mathcal{S}(\varphi)$. Consider some cluster $C' \in \mathcal{C}_{j+1}$. Let $C = parent_\tau(C')$ be its parent in the cluster tree $\tau$. Observe that $C \in \mathcal{C}_j$. Hence there exists an invocation $\varphi \in \Phi_j$ such that $C \in \mathcal{S}(\varphi)$. Let $\varphi'$ and $\hat{\varphi}$ be the children invocations of $\varphi$ (its children in the tree $T_\Phi$), and let $\mathcal{S}' = \mathcal{S}(\varphi')$ and $\hat{\mathcal{S}} = \mathcal{S}(\hat{\varphi})$ be their respective input collections.

By step 5 of Procedure `Period`, $Children_\tau(\mathcal{S}) = \mathcal{S}' \cup \hat{\mathcal{S}}$, and $\mathcal{S}' \cap \hat{\mathcal{S}} = \emptyset$. Consequently, since $C' \in Children_\tau(C) \subseteq Children_\tau(\mathcal{S}(\varphi))$, it follows that $C' \in \bigcup_{\varphi \in \Phi_{j+1}} \mathcal{S}(\varphi)$, and so $\mathcal{C}_{j+1} \subseteq \bigcup_{\varphi \in \Phi_{j+1}} \mathcal{S}(\varphi)$. Since $\mathcal{S}(\varphi) \subseteq \mathcal{C}_{j+1}$ for every invocation $\varphi \in \Phi_{j+1}$ (this can be seen by a straightforward induction on $j$ as well), it follows that $\mathcal{C}_{j+1} = \bigcup_{\varphi \in \Phi_{j+1}} \mathcal{S}(\varphi)$.

The fact that this is a disjoint union follows directly from the following three observations: (1) $\tau$ is a tree. (2) $\mathcal{C}_j = \bigsqcup_{\varphi \in \Phi_j} \mathcal{S}(\varphi)$ (3) For each $\varphi \in \Phi_j$, $Children_\tau(\varphi) = \mathcal{S}(\varphi') \cup \mathcal{S}(\hat{\varphi})$, and $\mathcal{S}(\varphi') \cap \mathcal{S}(\hat{\varphi}) = \emptyset$, where $\varphi'$ and $\hat{\varphi}$ are the two recursive invocations of `Period` from $\varphi$. ∎

Lemma 8.1 implies that different invocations of Procedure `Period` cover the entire cluster tree $\tau$, and therefore, the entire input graph. But we still need to show that these invocations do not interfere with each other. (See the discussion in the beginning of this section, and Definition 9.3.) In Section 9 we will show that the only problematic case is the one when two different invocations $\varphi_1$ and $\varphi_2$ of Procedure `Period` are invoked on two different subcollections $\mathcal{S}_1, \mathcal{S}_2 \subseteq \mathcal{C}_j$, for some index $j \in [x]$, with the same starting parameter $I = I(\varphi_1) = I(\varphi_2)$. Even though by Lemma 8.1, the sets $\mathcal{S}_1$ and $\mathcal{S}_2$ are disjoint, there might be edges between them, and those edges may cause collisions.

To overcome this difficulty, we do not use the schedule $\Pi$ returned by the invocation `Period`$(\mathcal{C}_1, 1)$, but rather construct another schedule $\Pi^*$ in which no two invocations $\varphi_1$ and $\varphi_2$ as above may appear. Intuitively, the idea is to *unify* $\varphi_1$ and $\varphi_2$ into a single invocation $\varphi_3$ of Procedure `Period` on input $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$ with the starting round parameter $I_3 = I$. For $\varphi_1$ and $\varphi_2$ as above, let `Unify`$(\{\varphi_1, \varphi_2\})$ be the subroutine that computes $\mathcal{S}$ and $I$ as described above, and let $\varphi_3$ be the invocation of Procedure `Period` on $\mathcal{S}$ and $I$. The input set of Procedure `Unify` may contain any number of invocations, as far as they all have mutually disjoint input collections of clusters of the same level, and the same starting round parameter $I$. The formal description of Procedure `Unify` is provided below.

Unify($\Psi$);

1. $\mathcal{S} \leftarrow \bigcup_{\varphi\in\Psi} \mathcal{S}(\varphi)$;

2. $I \leftarrow I(\varphi)$ for some $\varphi \in \Psi$;

3. return$((\mathcal{S}, I))$;

The following lemma shows that Procedure Unify can be safely used.

**Lemma 8.2.** *Let $\varphi_1, \varphi_2, \varphi_3$, $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3$, $I_3 = I_1 = I_2 = I$, be as above. For an index $i \in [3]$, let $\varphi'_i$ (respectively, $\hat{\varphi}_i$) be the first (resp., second) child-invocation of Procedure* Period, *and let $\mathcal{S}'_i$ (resp., $\hat{\mathcal{S}}_i$) be its input collection. Then $\mathcal{S}'_3 = \mathcal{S}'_1 \cup \mathcal{S}'_2$, and $\hat{\mathcal{S}}_3 = \hat{\mathcal{S}}_1 \cup \hat{\mathcal{S}}_2$.*

**Proof:** The sets $\mathcal{S}_1$ and $\mathcal{S}_2$ are disjoint subsets of vertices of level $j$ of the tree $\tau$. Recall also that $\mathit{Children}_\tau(\mathcal{S}_i) = \mathcal{S}'_i \cup \hat{\mathcal{S}}_i$, $\mathcal{S}'_i \cap \hat{\mathcal{S}}_i = \emptyset$, for $i \in [3]$. Finally, notice that $\mathcal{S}'_i = \{C' \mid C = \mathit{parent}_\tau(C') \in \mathcal{S}_i$ and $(C, C') \in \mathcal{F}\}$, where $\mathcal{F}$ is the path-forest $\mathcal{F}$. The lemma now follows from the definitions. ▮

Obviously, Lemma 8.2 can be generalized to any number of invocations $\varphi_1, \varphi_2, \ldots, \varphi_\ell$ by a straightforward induction on $\ell$. It follows that Procedure Unify can be applied to the invocations of $\Phi$ as many times as one wishes, and the resulting set $\Phi'$ of invocations will still satisfy the assertion of Lemma 8.1 (i.e., $\mathcal{C} = \bigsqcup_{\varphi\in\Phi'} \mathcal{S}(\varphi)$). Moreover, this way we can guarantee also that the set $\Phi'$ will not contain two invocations $\varphi_1, \varphi_2 \in \Phi'$ that satisfy: (1) Their input collections $\mathcal{S}_1$, $\mathcal{S}_2$ are contained in $\mathcal{C}_j$ for some index $j \in [x]$. (2) Their starting round parameters $I_1$ and $I_2$ are equal ($I_1 = I_2$).

Observe also that analogously, under the same conditions, one can unify invocations $\varphi_1$ and $\varphi_2$ of Procedure SubPeriod. Moreover, the invocation $\varphi_3$ that is obtained by such a unification satisfies the assertion of Lemma 8.2. Also, like with invocations of Procedure Period, the same statement generalizes to an arbitrary (possibly greater than 2) number of invocations.

We now turn to constructing our final schedule $\Pi^*$. Consider again the directed binary tree $T_\Phi = (\Phi, E_\Phi)$. For each level $j \in [x]$, Procedure Merge partitions $\Phi_j$ into $k_j \geq 1$ subsets $\Phi_j^{(1)}, \Phi_j^{(2)}, \ldots, \Phi_j^{(k_j)} \subseteq \Phi_j$, $\bigsqcup_{\ell=1}^{k_j} \Phi_j^{(\ell)} = \Phi_j$, such that all invocations from a subset $\Phi_j^{(\ell)}$, $\ell \in [k_j]$, have the same starting round parameter $I_j^{(\ell)}$. Moreover, for two distinct indices $\ell, \ell' \in [k_j]$, $\ell \neq \ell'$, $I_j^{(\ell)} \neq I_j^{(\ell')}$. Then Merge computes collections $\mathcal{S}_j^{(\ell)} = \bigcup_{\varphi_j^{(\ell)}} \mathcal{S}(\varphi)$, and sets $I_j^{(\ell)} = I(\varphi)$ for some $\varphi \in \Phi_j^{(\ell)}$, for every $\ell \in [k_j]$. (Those are essentially the invocations of Procedure Unify on the sets $\Phi_j^{(\ell)}$ for $\ell \in [k_j]$.)

Once the external loop (over the different values of the index $j$) is over, Merge computes schedules $\Pi_j^{(\ell)} = \texttt{SubPeriod}(\mathcal{S}_j^{(\ell)}, I_j^{(\ell)})$ for every $j \in [x]$ and $\ell \in [k_j]$. Finally, Merge computes the schedule-union of all these schedules $\Pi_j^{(\ell)}$ using Procedure SchedUnion (see Section 7), and returns the resulting schedule $\Pi^*$.

The formal description of Procedure `Merge` follows.

---

`Merge`$(T_\Phi = (\Phi, E_\Phi))$;

1. For $j \in [x]$ do

   1a. Compute the partition $\Phi_j^{(1)}, \Phi_j^{(2)}, \ldots, \Phi_j^{(k_j)}$ of $\Phi_j$ that satisfies

      i. $\bigcup \Phi_j^{\ell} = \Phi_j$;

      ii. $\forall \varphi, \varphi' \in \Phi_j^{(\ell)}$, $\ell \in [k_j]$, $I(\varphi) = I(\varphi')$;

      iii. $I(\Phi_j^{(\ell)}) \neq I(\Phi_j^{(\ell')})\ \ \forall \ell \neq \ell'$, $\ell, \ell' \in [k_j]$;

   1b. For $\ell \in [k_j]$ do $(\mathcal{S}_j^{(\ell)}, I_J^{(\ell)}) \leftarrow$ `Unify`$(\Phi_j^{(\ell)})$; $\Pi_j^{(\ell)} \leftarrow$ `SubPeriod`$(\mathcal{S}_j^{(\ell)}, I_j^{(\ell)})$;

2. $\Pi^* \leftarrow$ `SchedUnion`$(\{\Pi_j^{(\ell)} \mid j \in [x], \ell \in [k_j]\})$;

3. return$(\Pi^*)$;

---

To provide some intuition for Procedure `Merge`, we remark that the schedule $\Pi$ constructed by `Period`$(\mathcal{C}_1, 1)$ can be also constructed in the following way. For each invocation $\varphi \in \Phi$ with input collection $\mathcal{S}(\varphi)$ and starting round parameter $I(\varphi)$, let $\Pi(\varphi)$ be the schedule returned by the invocation `SubPeriod`$(\mathcal{S}(\varphi), I(\varphi))$. Then $\Pi$ is equal to the schedule-union of the set $\{\Pi(\varphi) \mid \varphi \in \Phi\}$. (This claim can be easily proven by induction on the depth of $T_\Phi$.) Procedure `Merge` constructs the schedule $\Pi^*$ similarly to this indirect way of constructing the schedule $\Pi$, but it also merges different invocations with equal starting round parameters.

Our algorithm can be now summarized in the following way.

---

`Broadcast`$(G, s, x)$;

1. Construct a BFS spanning tree $T$ of $G$ rooted in $s$;

2. $(\mathcal{C}_1, \ldots, \mathcal{C}_x, \chi_1, \ldots, \chi_x, \tau, \{(u(C), w(C)) \mid C \in \mathcal{C}\} \leftarrow$ `Preprocessing`$(G, s, T, x)$;

3. Compute the directed binary invocation tree $T_\Phi = (\Phi, E_\Phi)$ of the invocation `Period`$(C(s), 1)$ (see the beginning of this section for more details);

4. $\Pi^* \leftarrow$ `Merge`$(T_\Phi)$;

5. return$(\Pi^*)$;

---

**Lemma 8.3.** *The running time of the algorithm is polynomial in $n$.*

**Proof:** Consider the cluster tree $\tau$. Our algorithm starts with invoking Procedure `Period` on $\mathcal{C}_1 = \{C(s)\}$. Then `Period` invokes itself recursively on two disjoint subsets $\mathcal{S}'$ and $\hat{\mathcal{S}}$ of $\mathcal{C}_2$, such that $\mathcal{S}' \cup \hat{\mathcal{S}} = \mathcal{C}_2$, $\mathcal{S}' \cap \hat{\mathcal{S}} = \emptyset$. Moreover, in general, for each invocation $\varphi$ of Procedure `Period` on an input collection $\mathcal{S}$ of clusters that results in two recursive invocations $\varphi'$ and $\hat{\varphi}$ of `Period` on collections $\mathcal{S}'$ and $\hat{\mathcal{S}}$, these collections ($\mathcal{S}'$ and $\hat{\mathcal{S}}$) are disjoint, and also $\mathcal{S}' \cup \hat{\mathcal{S}} = \mathit{Children}(\mathcal{S})$.

Hence, a straightforward induction on the level of the specific cluster $C$ in the tree $\tau$ shows that every cluster $C \in \mathcal{C} = \bigcup_{j=1}^{x} \mathcal{C}_j$, there is at most one invocation $\varphi$ of Procedure `Period` whose input collection $\mathcal{S}$ contains the cluster $C$. Therefore, the overall number of (recursive) invocations of Procedure `Period` throughout the invocation `Period`($\{C(s)\}$) is at most $|\mathcal{C}|$.

Since every cluster $C$ of $\mathcal{C}$ contains at least one vertex that belongs to no other cluster $C' \in \mathcal{C}$ (recall that $d \geq 3$), it follows that $|\mathcal{C}| \leq n$, and so at most $n$ recursive invocations of Procedure `Period` occur during the invocation of the algorithm. Since every invocation of Procedure `Period` (disregarding its child-invocations) requires running time at most polynomial in $n$, it follows that the running time of the entire algorithm is polynomial in $n$ as well. ▮

# 9 Analysis

In this section we argue that our algorithm constructs a short broadcast schedule that informs all the vertices of the input graph $G$.

We start with the following lemma that is proven by induction on the indices $j \in [x]$ of layers $\mathcal{C}_j$ of the super-graph $\tilde{G}$.

**Lemma 9.1.** *The starting rounds of invocations $\varphi$ and $\gamma$ of Procedures* `Period` *and* `Vert_Broadcast`*, respectively, on input collections $\mathcal{S} \subseteq \mathcal{C}_j$ are of the form*

$$I(\varphi) \;=\; I(\gamma) \;=\; 1 + \tilde{d} \cdot (j-1) + h(3 \cdot \tilde{d} + a \cdot d \log^2 n) \;,$$

*for some positive integer $h \in [j-1]$. The starting rounds of invocations $\delta$ of Procedure* `Hor_Broadcast` *on such a collection are of the form*

$$I(\delta) \;=\; 1 + \tilde{d} \cdot (j+1) + h(3 \cdot \tilde{d} + a \cdot d \log^2 n) \;,$$

*for some positive integer $h \in [j-1]$.*

**Proof:** Consider the simpler version of our algorithm that invokes Procedure `Period` on $\mathcal{C}_1$ with starting round parameter equal to 1. Note that as Procedure `Merge` replaces some invocations of Procedure `Period` with other invocations of the same procedure with the *same starting round parameter*, it is enough to prove the lemma for this simpler version of our algorithm.

**The induction base:** For layer $j = 1$, the only invocation of Procedure `Vert_Broadcast` (on the input collection $\{C(s)\}$) has starting round 1, and the same is true for Procedure `Period`. The only invocation of Procedure `Hor_Broadcast` (on the same singleton input collection) starts at round $1 + 2\tilde{d}$, proving the induction base.

**The induction step:** For an index $j \in [x]$, let $\texttt{Period}_j$ (resp., $\texttt{Vert\_Broadcast}_j$; $\texttt{Hor\_Broadcast}_j$) denote the set of all the invocations of Procedure `Period` (resp., `Vert_Broadcast`; `Hor_Broadcast`) on different collections $\mathcal{S} \subseteq \mathcal{C}_j$ that occur during the algorithm.

For an invocation $\varphi \in \texttt{Period}_j$, let $\gamma(\varphi)$ (resp., $\delta(\varphi)$) denote the invocation of Procedure `Vert_Broadcast` (resp., `Hor_Broadcast`) that is invoked by $\varphi$. Let $\varphi'$ and $\hat{\varphi}$ denote the first and second child-invocations of the invocation $\varphi$ (note that $\varphi', \hat{\varphi} \in \texttt{Period}_{j+1}$).

By the induction hypothesis, since $\varphi \in \texttt{Period}_j$,

$$I(\varphi) = 1 + \tilde{d} \cdot (j-1) + h(\varphi) \cdot (3 \cdot \tilde{d} + a \cdot d \log^2 n) \ ,$$

where $h(\varphi) \in [j-1]$. Also, by construction,

$$\begin{aligned} I(\varphi') &= I(\varphi) + \tilde{d} \ , \ I(\hat{\varphi}) \ = \ I(\varphi) + 4\tilde{d} + a \cdot d \log^2 n \ , \\ I(\gamma(\varphi)) &= I(\varphi) \ , \ I(\delta(\varphi)) \ = \ I(\varphi) + 2 \cdot \tilde{d} \ , \\ I(\gamma(\varphi')) &= I(\varphi') \ , \ I(\delta(\varphi')) \ = \ I(\varphi') + 2 \cdot \tilde{d} \ , \\ I(\gamma(\hat{\varphi})) &= I(\hat{\varphi}) \ , \ I(\delta(\hat{\varphi})) \ = \ I(\hat{\varphi}) + 2 \cdot \tilde{d} \ . \end{aligned}$$

Observe that $\varphi \in \texttt{Period}_j$, $\varphi', \hat{\varphi} \in \texttt{Period}_{j+1}$, $\gamma(\varphi'), \gamma(\hat{\varphi}) \in \texttt{Vert\_Broadcast}_{j+1}$, and $\delta(\varphi'), \delta(\hat{\varphi}) \in \texttt{Hor\_Broadcast}_{j+1}$. Hence,

$$\begin{aligned} I(\gamma(\varphi')) &= I(\varphi') \ = \ I(\varphi) + \tilde{d} \\ &= 1 + \tilde{d} \cdot j + h(\varphi) \cdot (3\tilde{d} + a \cdot d \log^2 n) \ , \end{aligned}$$

with $h(\varphi) \in [j-1] \subseteq [j]$.

Analogously,

$$I(\gamma(\hat{\varphi})) \ = \ I(\hat{\varphi}) \ = \ I(\varphi) + 4\tilde{d} + a \cdot d \log^2 n \ = \ 1 + \tilde{d} \cdot j + h(\gamma(\hat{\varphi})) \cdot (3\tilde{d} + a \cdot d \log^2 n) \ ,$$

with $h(\gamma(\hat{\varphi})) = h(\varphi) + 1$. Since $h(\varphi) \in [j-1]$, it follows that $h(\gamma(\hat{\varphi})) \in [j]$.

Also,

$$\begin{aligned} I(\delta(\varphi')) &= I(\varphi') + 2\tilde{d} \ = \ I(\varphi) + 3\tilde{d} \\ &= 1 + \tilde{d} \cdot (j+2) + h(\varphi) \cdot (3\tilde{d} + a \cdot d \log^2 n) \ , \end{aligned}$$

and $h(\varphi) = h(\delta(\varphi)) \in [j-1] \subseteq [j]$.

Finally,

$$\begin{aligned} I(\delta(\hat{\varphi})) &= I(\hat{\varphi}) + 2\tilde{d} \ = \ I(\varphi) + 6\tilde{d} + a \cdot d \log^2 n \\ &= 1 + \tilde{d} \cdot (j+2) + (h(\varphi) + 1) \cdot (3\tilde{d} + a \cdot d \log^2 n) \ , \end{aligned}$$

and so $h(\delta(\hat{\varphi})) = h(\varphi) + 1 \in [j]$, as required.

To summarize, $I(\gamma(\varphi'))$, $I(\gamma(\hat{\varphi}))$, $I(\delta(\varphi'))$ and $I(\delta(\hat{\varphi}))$ are all of required form. (Note that $\gamma(\varphi'), \gamma(\hat{\varphi}) \in \texttt{Vert\_Broadcast}_{j+1}$, and $\delta(\varphi'), \delta(\hat{\varphi}) \in \texttt{Hor\_Broadcast}_{j+1}$, and the required forms for the staring rounds of invocations of Procedures `Vert_Broadcast` and `Hor_Broadcast` are different.)
▮

**Notation 9.2.** *For an invocation $\varphi$ of one of the procedures that are used by our algorithm, let $\Pi(\varphi)$ denote the schedule returned by this procedure.*

**Definition 9.3.** *Consider two invocations $\varphi_1$, $\varphi_2$ of procedures that are used in our algorithm. These two invocations are said to* interfere with each other *if there exists an index $\ell = 1, 2, \ldots$, and a pair of (not necessarily distinct) vertices $u, w \in V$ at distance at most 2 one from another in the graph $G = (V, E)$, such that the vertex $u$ is scheduled to broadcast on round $\ell$ by the schedule $\Pi(\varphi_1)$ and the vertex $w$ is scheduled to broadcast on round $\ell$ by the schedule $\Pi(\varphi_2)$.*

**Lemma 9.4.** *Different invocations of Procedures* `Vert_Broadcast` *and* `Hor_Broadcast` *do not interfere with each other.*

**Proof:** Consider two invocations $\varphi_1$ and $\varphi_2$. For $i \in \{1, 2\}$, let $j_i$ be the *layer* of the invocation $\varphi_i$, that is, the index that satisfies $\varphi_i \in \texttt{Vert\_Broadcast}_{j_i} \cup \texttt{Hor\_Broadcast}_{j_i}$. Observe that if $|j_1 - j_2| \geq 2$ then the invocations $\varphi_1$ and $\varphi_2$ cannot interfere with each other. (Recall that $d \geq 3$.) Hence, from now on we assume that either the layers of the invocations $\varphi_1$ and $\varphi_2$, denoted $\ell(\varphi_1)$ and $\ell(\varphi_2)$, are equal, or they differ by 1.

The discussion splits into two cases. The first case is $\ell(\varphi_1) = \ell(\varphi_2) = j$, for some index $j \in [x]$, and the second case is $\ell(\varphi_1) = j$, $\ell(\varphi_2) = j+1$ (without loss of generality), for some index $j \in [x-1]$.

1. $(\ell(\varphi_1) = \ell(\varphi_2) = j)$
   The discussion splits again, this time into three cases, depending on whether $\varphi_1$, $\varphi_2$ are both invocations of Procedure `Vert_Broadcast`, or both of them are invocations of Procedure `Hor_Broadcast`, or one of them is an invocation of Procedure `Vert_Broadcast` and the other is an invocation of Procedure `Hor_Broadcast`.

   (a) $(\varphi_1, \varphi_2 \in \texttt{Vert\_Broadcast}_j)$
   In this case the starting rounds $I(\varphi_1)$, $I(\varphi_2)$ of the two invocations satisfy

$$\begin{aligned} I(\varphi_1) &= 1 + \tilde{d} \cdot (j-1) + h(\varphi_1) \cdot (3\tilde{d} + a \cdot d \log^2 n) , \\ I(\varphi_2) &= 1 + \tilde{d} \cdot (j-1) + h(\varphi_2) \cdot (3\tilde{d} + a \cdot d \log^2 n) , \end{aligned}$$

   where $h(\varphi_1), h(\varphi_2) \in [j-1]$.
   If $h(\varphi_1) = h(\varphi_2)$ then $I(\varphi_1) = I(\varphi_2)$, but by construction, this cannot happen to two distinct invocations of the same procedure that have input collections of the same layer. (Recall that in this case the input collections $\mathcal{S}(\varphi_1), \mathcal{S}(\varphi_2) \in \mathcal{C}_j$ of these two invocations would be merged into a single collection by Procedure `Merge`, and Procedure `Vert_Broadcast` would be invoked on a unified collection that would contain the set $\mathcal{S}(\varphi_1) \cup \mathcal{S}(\varphi_2)$.)

If $h(\varphi_1) \neq h(\varphi_2)$ then since both $h(\varphi_1)$ and $h(\varphi_2)$ are integer, it follows that $|I(\varphi_1) - I(\varphi_2)| \geq 3\tilde{d} + a \cdot d \log^2 n$. Suppose, without loss of generality, that $I(\varphi_1) < I(\varphi_2)$. The length of the schedule $\Pi(\varphi_1)$ that is constructed by the invocation $\varphi_1$ is $\tilde{d} < I(\varphi_2) - I(\varphi_1)$, and so, the invocations $\varphi_1$ and $\varphi_2$ do not interfere with each other.

(b) ($\varphi_1, \varphi_2 \in \texttt{Hor\_Broadcast}_j$)
In this case the starting rounds of the two invocations satisfy

$$\begin{aligned} I(\varphi_1) &= 1 + \tilde{d} \cdot (j+1) + h(\varphi_1) \cdot (3\tilde{d} + a \cdot d \log^2 n) \ , \\ I(\varphi_2) &= 1 + \tilde{d} \cdot (j+1) + h(\varphi_2) \cdot (3\tilde{d} + a \cdot d \log^2 n) \ , \end{aligned}$$

where $h(\varphi_1), h(\varphi_2) \in [j-1]$.
If $h(\varphi_1) = h(\varphi_2)$ then $I(\varphi_1) = I(\varphi_2)$, and by the same argument as in Case 1a, this implies that $\varphi_1 = \varphi_2$, and we are done.
If $h(\varphi_1) \neq h(\varphi_2)$ then $|I(\varphi_1) - I(\varphi_2)| \geq 3\tilde{d} + a \cdot d \log^2 n$. Since $\max\{|\Pi(\varphi_1)|, |\Pi(\varphi_2)|\} = a \cdot d \log^2 n < |I(\varphi_1) - I(\varphi_2)|$, the invocations $\varphi_1$ and $\varphi_2$ would not interfere with each other in this case as well.

(c) ($\varphi_1 \in \texttt{Vert\_Broadcast}_j$, $\varphi_2 \in \texttt{Hor\_Broadcast}_j$)
In this case

$$\begin{aligned} I(\varphi_1) &= 1 + \tilde{d} \cdot (j-1) + h(\varphi_1) \cdot (3\tilde{d} + a \cdot d \log^2 n) \ , \\ I(\varphi_2) &= 1 + \tilde{d} \cdot (j+1) + h(\varphi_2) \cdot (3\tilde{d} + a \cdot d \log^2 n) \ , \end{aligned}$$

where $h(\varphi_1), h(\varphi_2) \in [j-1]$.
If $h(\varphi_1) = h(\varphi_2)$ then $I(\varphi_2) - I(\varphi_1) = 2\tilde{d}$. The length of the schedule $\Pi(\varphi_1)$ that is constructed by the invocation $\varphi_1$ is $\tilde{d} < I(\varphi_2) - I(\varphi_1)$, and so, in this subcase we are done.
If $h(\varphi_1) < h(\varphi_2)$ then $I(\varphi_1) < I(\varphi_2)$ too, and the difference $I(\varphi_2) - I(\varphi_1)$ is even greater than in the subcase $h(\varphi_1) = h(\varphi_2)$.
Finally, consider the subcase $h(\varphi_1) > h(\varphi_2)$. Since $h(\varphi_1)$ and $h(\varphi_2)$ are both integer, it follows that

$$\begin{aligned} I(\varphi_1) - I(\varphi_2) &= (h(\varphi_1) - h(\varphi_2)) \cdot (3\tilde{d} + a \cdot d \log^2 n) - 2\tilde{d} \\ &\geq \tilde{d} + a \cdot d \log^2 n \ > \ 0 \ , \end{aligned}$$

and so, the starting round of the invocation $\varphi_2$ is at least by $\tilde{d} + a \cdot d \log^2 n$ smaller than the starting round of the invocation $\varphi_1$. The length of the schedule $\Pi(\varphi_2)$ is $a \cdot d \log^2 n$, and so, this schedule terminates $\tilde{d}$ rounds before the starting round of the invocation $\varphi_1$, implying that the two invocations do not interfere with each other.

2. ($\ell(\varphi_1) = j$, $\ell(\varphi_2) = j+1$)

(a) ($\varphi_1 \in \texttt{Vert\_Broadcast}_j$, $\varphi_2 \in \texttt{Vert\_Broadcast}_{j+1}$)

In this case

$$\begin{aligned} I(\varphi_1) &= 1+\tilde{d}\cdot(j-1)+h(\varphi_1)\cdot(3\tilde{d}+a\cdot d\log^2 n)\ , \\ I(\varphi_2) &= 1+\tilde{d}\cdot j+h(\varphi_2)\cdot(3\tilde{d}+a\cdot d\log^2 n)\ , \end{aligned}$$

where $h(\varphi_1) \in [j-1]$, $h(\varphi_2) \in [j]$.

If $h(\varphi_1) = h(\varphi_2)$ then $I(\varphi_2) - I(\varphi_1) = \tilde{d}$, which is precisely the length of the schedule $\Pi(\varphi_1)$, and so, the schedule $\Pi(\varphi_1)$ terminates exactly one round before the starting round of the invocation $\varphi_2$.

If $h(\varphi_1) \neq h(\varphi_2)$ then

$$|I(\varphi_2)-I(\varphi_1)| \ \geq\ 2\tilde{d}+a\cdot d\log^2 n\ >\ \max\{|\Pi(\varphi_1)|, |\Pi(\varphi_2)|\} = \tilde{d}\ .$$

and, therefore, the two invocations do not interfere wittth each other.

(b) ($\varphi_1 \in \texttt{Vert\_Broadcast}_j$, $\varphi_2 \in \texttt{Hor\_Broadcast}_{j+1}$)

In this case

$$\begin{aligned} I(\varphi_1) &= 1+\tilde{d}\cdot(j-1)+h(\varphi_1)\cdot(3\tilde{d}+a\cdot d\log^2 n)\ , \\ I(\varphi_2) &= 1+\tilde{d}\cdot(j+2)+h(\varphi_2)\cdot(3\tilde{d}+a\cdot d\log^2 n)\ , \end{aligned}$$

where $h(\varphi_1) \in [j-1]$, $h(\varphi_2) \in [j]$.

If $h(\varphi_1) = h(\varphi_2)$ then $I(\varphi_2)-I(\varphi_1) = 3\tilde{d} > 0$, and since $|\Pi(\varphi_1)| = \tilde{d}$, and $I(\varphi_1) < I(\varphi_2)$, we are done.

If $h(\varphi_1) \neq h(\varphi_2)$ then $|I(\varphi_2) - I(\varphi_1)| \geq a\cdot d\log^2 n$, and since $\max\{|\Pi(\varphi_1)|, |\Pi(\varphi_2)|\} = a\cdot d\log^2 n$, the schedule with smaller starting round will terminate at least one round before the beginning of the schedule with the larger starting round.

(c) ($\varphi_1 \in \texttt{Vert\_Broadcast}_{j+1}$, $\varphi_2 \in \texttt{Hor\_Broadcast}_j$)

In this case

$$\begin{aligned} I(\varphi_1) &= 1+\tilde{d}\cdot j+h(\varphi_1)\cdot(3\tilde{d}+a\cdot d\log^2 n)\ , \\ I(\varphi_2) &= 1+\tilde{d}\cdot(j+1)+h(\varphi_2)\cdot(3\tilde{d}+a\cdot d\log^2 n)\ , \end{aligned}$$

where $h(\varphi_1) \in [j]$, $h(\varphi_2) \in [j-1]$.

If $h(\varphi_1) = h(\varphi_2)$ then $I(\varphi_2) = I(\varphi_1) + \tilde{d}$, and, in particular, $I(\varphi_1) < I(\varphi_2)$. Since $|\Pi(\varphi_1)| = \tilde{d}$, the schedule $\Pi(\varphi_1)$ terminates at least one round before $I(\varphi_2)$, that is, the round when the schedule $\Pi(\varphi_2)$ starts.

If $h(\varphi_1) \neq h(\varphi_2)$ then $|I(\varphi_1) - I(\varphi_2)| \geq 2\tilde{d} + a\cdot d\log^2 n \geq \max\{|\Pi(\varphi_1)|, |\Pi(\varphi_2)|\}$, and we are done.

(d) ($\varphi_1 \in \texttt{Hor\_Broadcast}_j$, $\varphi_2 \in \texttt{Hor\_Broadcast}_{j+1}$)
Recall that the schedule that is constructed by an invocation $\delta \in \texttt{Hor\_Broadcast}_j$ employs only rounds $\ell$ that satisfy $\ell \equiv j \pmod 2$. Hence the invocation $\varphi_1$ employs only rounds with even indices and the invocation $\varphi_2$ employs only rounds with odd indices, or vice versa. Hence, the two invocations do not interfere with each other.

▮

**Lemma 9.5.** *The length of the schedule $\Pi^*$ that is constructed by our algorithm is at most $Rad + O(d \cdot \log^3 n + x \cdot \log n)$, and it informs all the vertices of the graph.*

**Proof:** Consider a vertex $v \in V$. Let $C \in \mathcal{C}$ be one of the clusters that contain the vertex $v$. Let $\mathcal{P}$ be the (unique) path between the root cluster $C(s)$ and the cluster $C$ in the tree $\tau$. Observe that $\mathcal{P}$ contains at most $x$ clusters, and, therefore, at most $x-1$ super-edges. Recall that since $\mathcal{F}$ is a *semi-spanning* path-forest, the path $\mathcal{P}$ contains at most $\log n$ super-edges that do not belong to $\mathcal{F}$ (henceforth termed as *non-edges* of $\mathcal{F}$).

Consider the unique representation of the path $\mathcal{P}$ as the concatenation $\mathcal{P} = \mathcal{P}_1 \cdot \tilde{e}_1 \cdot \mathcal{P}_2 \cdot \tilde{e}_2 \cdot \ldots \cdot \mathcal{P}_r \cdot \tilde{e}_r \cdot \mathcal{P}_{r+1}$, where $r \in [\log n]$, and for each index $i \in [r]$, $\mathcal{P}_i$ is a (possibly empty) path that is contained in one of the paths of the forest $\mathcal{F}$, and the super-edges $\tilde{e}_1, \tilde{e}_2, \ldots, \tilde{e}_r$ are non-edges of the path-forest $\mathcal{F}$. Also, the cluster $C(s)$ is one of the endpoints of the path $\mathcal{P}_1$, if $\mathcal{P}_1 \neq \emptyset$, and if $\mathcal{P}_1 = \emptyset$ then $C(s)$ is one of the endpoints of the super-edge $\tilde{e}_1$. Analogously, the cluster $C$ is one of the endpoints of the path $\mathcal{P}_{r+1}$ if the latter is not empty, and if it is empty then the cluster $C$ is one of the endpoints of the super-edge $\tilde{e}_r$.

For each index $i \in [r+1]$, let $p_i = |\mathcal{P}_i|$. Let $\mathcal{P}_i = (C_1^{(i)}, C_2^{(i)}, \ldots, C_{p_i}^{(i)})$, and for each index $j \in [p_i]$, let $u_j$, $w_j$ denote the messenger and the representative of the cluster $C_j^{(i)}$, respectively. Recall that by Lemma 3.1, $w_1 = u_2,\ w_2 = u_3,\ \ldots, w_{p_i-1} = u_{p_i}$, and the vertex $w_{p_i}$ is a descendent of the vertex $u_1$ with respect to the BFS spanning tree $T$ of the graph $G$.

Consider an invocation of Procedure `Period` on some collection $\mathcal{S}$ that contains the cluster $C_1^{(i)}$. Suppose that this invocation has starting round parameter $I$. By construction, the procedure invokes itself recursively with starting round parameter $I + \tilde{d}$ rounds on a collection $\mathcal{S}'$ of clusters that contains the cluster $C_2^{(i)}$. The latter recursive invocation again invokes itself recursively with starting round parameter $I + 2\tilde{d}$ on a collection $\mathcal{S}''$ of clusters that contains the cluster $C_3^{(i)}$, etc. To summarize, this line of recursive invocations constructs a partial schedule of length $p_i \cdot \tilde{d}$ that delivers the message from the messenger $u_1$ of the cluster $C_1^{(i)}$ to the messenger $u_{p_i}$ of the cluster $C_{p_i}^{(i)}$.

For each non-edge $\tilde{e}_i = \{C', C''\}$, $i \in [r]$, of the path-forest $\mathcal{F}$, the message is delivered from the messenger of the cluster $C'$ to all the other vertices of this cluster (including the messenger of the cluster $C''$ that belongs to the intersection $C' \cap C''$) via a schedule that is constructed by an invocation of Procedure `Period` on a collection $\mathcal{S}$ of clusters that contains the cluster $C'$.

Procedure `Period` starts with invoking Procedure `Vert_Broadcast` on the same input collection $\mathcal{S}$. This invocation constructs a schedule of length $\tilde{d}$ that delivers the message from the messenger $u'$ of the cluster $C'$ to its representative $w'$. Then this schedule is padded with $\tilde{d}$ empty rounds, and then Procedure `Period` concatenates the resulting schedule with the schedule that is returned by the invocation of Procedure `Hor_Broadcast` on the same input collection $\mathcal{S}$ (but with starting round parameter greater by $2\tilde{d}$ than the one that Procedure `Period` accepted as a part of its input). The latter schedule is of length $a \cdot d \log^2 n$, and, in particular, it delivers the message from the representative $w'$ to all the other vertices of the cluster $C'$ including the messenger $u''$ of the cluster $C''$. (Recall that $\tilde{e}_i = \{C', C''\} \in \tau$, and $C' = parent_\tau(C'')$. Hence, by construction, the messenger $u''$ of $C''$ belongs to $C'$.) Finally, this schedule is padded with extra $2\tilde{d}$ empty rounds.

To summarize, for each non-edge $\tilde{e}_i = \{C', C''\}$, $i \in [r]$, of the path-forest $\mathcal{F}$, Procedure `Period` constructs a schedule of length $4\tilde{d} + a \cdot d \log^2 n$ that delivers the message from the messenger $u'$ of the cluster $C'$ to the messenger $u''$ of the cluster $C''$.

Altogether, it follows that our algorithm constructs a schedule that delivers the message from the messenger $s$ of the cluster $C(s)$ to the messenger $u$ of the cluster $C$, and this schedule is of length

$$\sum_{i=1}^{r+1} p_i \tilde{d} + r \cdot (4\tilde{d} + a \cdot d \log^2 n) \;=\; \tilde{d} \sum_{i=1}^{r+1} p_i + r(4\tilde{d} + a \cdot d \log^2 n) \; .$$

Observe that $|\mathcal{P}| = \sum_{i=1}^{r+1} p_i + r$. Hence, the length of this schedule is equal to $\tilde{d}|\mathcal{P}| + r \cdot (3\tilde{d} + a \cdot d \log^2 n)$. Finally, the delivery of the message from the messenger $u$ of the cluster $C$ to the vertex $v$ (and to all the other vertices of this cluster $C$) is done via a schedule of length $4\tilde{d} + a \cdot d \log^2 n$ that is constructed by an invocation of Procedure `Period` on a collection of clusters that contains the cluster $C$.

Hence, altogether, the schedule that is constructed by our algorithm delivers the message from the source $s$ to the vertex $v$ no later than on round $\tilde{d} \cdot (|\mathcal{P}| + 1) + (r+1) \cdot (3\tilde{d} + a \cdot d \log^2 n)$. Observe that the tree $\tau$ is of depth $x$, and so $|P| \le x$. Also, $r \le \log n$, $\alpha = O(\log n)$, and $\tilde{d} = d + O(\log n)$. Hence, the length of the schedule is at most

$$\begin{aligned} &(d + O(\log n))(x+1) + O(\log n)(O(d) + O(\log n) + O(d \log^2 n)) \;= \\ &d \cdot x + O(x \cdot \log n + d \cdot \log^3 n) \;\le\; Rad + O(x \cdot \log n + d \cdot \log^3 n) \; . \end{aligned}$$

∎

Recall that by assuming that $x$ divides $Rad$ we could lose at most an additive term of $x$ in our bound on the length of the schedule. Hence, the overall length of $\Pi^*$ is at most $(Rad + O(x \cdot \log n + d \cdot \log^3 n)) + x = Rad + O(x \cdot \log n + d \cdot \log^3 n)$, i.e., this term may contribute only to the constant hidden by the $O$-notation. The main result now follows as $d = \lfloor Rad/x \rfloor$.

**Corollary 9.6.** *For an input $(G = (V,E), s)$, $s \in V$, with radius $Rad \ge \log^2 n$, running our algorithm with input parameter $x = \sqrt{Rad} \cdot \log n$ results in a radio broadcast schedule of length*

$Rad + O(\sqrt{Rad} \cdot \log^2 n)$.

**Remark:** To obtain an algorithmic analogue to Corollary 9.6, we observe that all the components of our algorithm except Algorithm `Anonymous_Broadcast` can be implemented in deterministic polynomial time in a straight-forward way. Algorithm `Anonymous_Broadcast` is due to Bar-Yehuda et al. [4], and it can be implemented in randomized polynomial time. Consequently, our entire algorithm can be implemented in randomized polynomial time.

However, recently Kowalski and Pelc [15] devised a deterministic counter-apart of the algorithm of Bar-Yehuda et al. [4]. Specifically, the algorithm of [15] constructs schedules of length $O(D \cdot \log n + \log^2 n)$ in deterministic polynomial time. Consequently, using the algorithm of [15] as a subroutine instead of the algorithm of [4] derandomizes our algorithm while maintaining the same upper bound on the length of constructed schedules.

Note that for graphs of radius $Rad = \Omega(\log^4 n)$, the algorithm provides a radio broadcast schedule of length $O(Rad)$. Observe also that in the range $Rad < \log^2 n$ the result of [4] provides a better upper bound of $O(Rad \log n + \log^2 n) = O(\log^3 n)$.

## 10 An improved bound for graphs with small genus

In this section we describe a simplified algorithm and its analysis that provide an improved upper bound on the length of an optimal radio schedule for graphs with small genus.

We start with defining *genus*.

**Definition 10.1.** *An* embedding *of an $n$-vertex graph $G = (V, E)$ into a surface $\mathcal{S} \subseteq \mathtt{R}^{\mathtt{n}^2}$ is a pair $(f_V, f_E)$ of mappings such that*

1. *$F_v : V \to \mathcal{S}$ is a one-to-one mapping from the set $V$ of vertices to the surface $\mathcal{S}$.*
2. *$f_E$ maps each edge $e = \{u, w\}$ of the graph $G$ into a continuous curve that connects $f_V(u)$ with $f_V(w)$.*
3. *For every two edges $e = \{u, w\}, e' = \{u', w'\} \in E$, with $e \cap e' = \emptyset$, the curves $f_E(e)$ and $f_E(e')$ do not intersect.*
   *For every two edges $e = \{u, w\}, e' = \{u, w'\} \in E$ that are incident to the same vertex $u$, the curves $f_E(e)$ and $f_E(e')$ intersect only in $f_V(u)$.*

*The* geometric genus *of a surface $\mathcal{S}$ is the largest number of non-intersecting simple closed curves that can be drawn on $\mathcal{S}$ without separating it. (See [18].)*

*The* genus *of a graph $G$ is the minimum number $\gamma$ such that there exists an embedding of $G$ into a surface $\mathcal{S} \subseteq \mathtt{R}^{\mathtt{n}^2}$ of geometric genus $\gamma$.*

The algorithm starts with forming the layers $V_1, \ldots, V_x$ exactly as in Section 3. Then the algorithm partitions each layer $V_j$ into a collection of disjoint super-sets, and this is also done exactly as in Section 3. As previously, the diameter of each of these super-sets is at most $2d$.

As in the algorithm of Section 3, the super-sets in adjacent layers may intersect. For $i = 1, 2, \ldots, x$, let $\tilde{G}_i$ denote the graph induced by the super-sets of layer $i$. That is, the set of vertices of $\tilde{G}_i$ is the set of super-sets of layer $i$, and there is an edge between two super-sets $C$ and $C'$ if and only if there is an edge between some vertex $v \in C$ and $v' \in C'$ in the original graph $G$. Let $\chi_i$ denote the chromatic number of $\tilde{G}_i$, and let $\chi = \max\{\chi_i \mid i = 1, 2, \ldots, x\}$.

Next, we form the tree of super-sets. In this version of the algorithm we do not build clusters out of super-sets, but rather consider each super-set as a separate cluster. Hence, each cluster has a unique parent cluster (unless it is the root of the tree), as it has exactly one vertex in its top level, and this vertex belongs to the bottom level of exactly one of the clusters of the higher layer. Consequently, in this version of the algorithm there is no need to build the tree $\tau$ and to rank its vertices in parallel, but rather we build the tree and then rank it in the standard way. (That is, the leaves are assigned rank 0. A non-leaf vertex $v$ is assigned rank $r$ where $r$ is the maximum rank of its children if there is a unique child of $v$ that has rank $r$; otherwise it is assigned rank $r + 1$.)

The algorithm of Section 4 for constructing the schedule is now applicable. It gives rise to a schedule of length

$$(d + O(\chi)) \cdot x + (d \cdot O(\log n) + O(\log^2 n)) \cdot \log n \ . \tag{1}$$

(This can be shown by the same argument as in the proof of Lemma 9.5.)

Set

$$x \ = \ \min\{Rad, \sqrt{\frac{Rad}{\chi}} \log n\} \ . \tag{2}$$

1. [Case 1: $(Rad < \frac{\log^2 n}{\chi})$ ]
   In this case (2) implies $x = Rad$, and so $d = 1$. Hence (1) implies that the length of the schedule is $O(Rad \cdot \chi + \log^3 n) = O(\log^3 n)$.

2. [ Case 2: $(Rad \geq \frac{\log^2 n}{\chi})$ ]
   In this case $x = \sqrt{\frac{Rad}{\chi}} \log n$, $d = \frac{\sqrt{Rad \cdot \chi}}{\log n}$. Hence (1) implies that the length of the schedule is $Rad + O(\sqrt{Rad \cdot \chi} \cdot \log n + \log^3 n)$.

   (a) [Case 2.1 $(Rad \geq \chi \cdot \log^2 n)$]
   In this case $Rad \geq \sqrt{Rad \cdot \chi} \cdot \log n$, and consequently, the length of the schedule is $O(Rad + \log^3 n)$.

   (b) [Case 2.2 $(\frac{\log^2 n}{\chi} \leq Rad < \chi \cdot \log^2 n)$]
   In this case $\sqrt{Rad \cdot \chi} \cdot \log n \leq \log^3 n$, and consequently, the length of the schedule is $Rad + O(\chi \cdot \log^2 n + \log^3 n)$.

Particularly, when $\chi = O(\log n)$ we get an upper bound of $O(Rad + \log^3 n)$ that is better than our bound of $O(Rad + \log^4 n)$ (that, however, applies to general graphs).

Graphs of genus $g$ have arboricity $O(\sqrt{g})$, and graphs with arboricity $a$ are $2a$-colorable. So graphs of genus $g$ are $O(\sqrt{g})$-colorable. Observe that the genus of each graph $\tilde{G}_i$ is at most the genus $g$ of the original graph $G$, and so $\chi = \max\{\chi_i \mid i = 1, 2, \ldots, x\} = O(\sqrt{g})$. Particularly, from case 2, for $Rad \geq \log^2 n \geq \frac{\log^2 n}{\chi}$, the length of the schedule is at most $Rad + O(\sqrt{Rad} \cdot g^{1/4} \cdot \log n + \log^3 n)$. In particular, for graphs of constant genus (including planar graphs and graphs that exclude small minors) our upper bound is $Rad + O(\sqrt{Rad} \cdot \log n + \log^3 n) = O(Rad + \log^3 n)$.

We remark that this upper bound is *existential* for graphs with small chromatic number, and we do not have a polynomial-time algorithm that achieves this bound because a direct implementation of our algorithm would require to color an arbitrary graph with a near-optimal number of colors. However, for graphs of bounded arboricity (such as *planar* graphs) our bound is *constructive*, because those graphs can obviously be efficiently colored with a near-optimal number of colors.

## Acknowledgements

We are very grateful to an anonymous referee whose remarks helped us to improve the presentation of this paper.

## References


[1] N. Alon, A. Bar-Noy, N. Linial, and D. Peleg. A lower bound for radio broadcast. *Journal of Computer and System Sciences*, 43:290–298, 1991.

[2] B. Awerbuch, B. Berger, L. Cowen, and D. Peleg. Fast network decomposition. In *Proc. 11th ACM Symp. on Principles of Distr. Comp.*, pages 161–173, 1992.

[3] R. Bar-Yehuda, O. Goldreich, and A. Itai. Efficient emulation of single-hop radio network with collision detection on multi-hop radio network with no collision detection. *Distributed Computing*, 2(5):67–72, 1991.

[4] R. Bar-Yehuda, O. Goldreich, and A. Itai. On the time-complexity of broadcast in multi-hop radio networks: An exponential gap between determinism and randomization. *J. Comput. Syst. Sci.*, 1(45):104–126, 1992.

[5] I. Chlamtac and S. Kutten. A spatial-reuse TDMA/FDMA for mobile multihop radio networks. In *Proc. IEEE INFOCOM*, pages 389–394, 1985.

[6] I. Chlamtac and Weinstein. The wave expansion approach to broadcasting in multihop radio networks. In *Proc. IEEE INFOCOM*, pages 874–881, 1987.

[7] B. Chlebus, L. Gasieniec, A. Gibbons, A. Pelc, and W. Rytter. Deterministic broadcasting in ad hoc radio networks. *Distributed Computing*, 1(15):27–38, 2002.

[8] M. Chrobak, L. Gasieniec, and W. Rytter. Fast broadcasting and gossiping in radio networks. In *Proc. 41st Symp. on Foundations of Computer Science*, pages 575–581, 2000.

[9] A. F. Clementi, A. Monti, and R. Silvestri. Distributed broadcast in radio networks of unknown topology. *TCS*, 3(302):337–364, 2003.

[10] A. Czumaj and W. Rytter. Broadcasting algorithms in radio networks with unknown topology. In *Proc. 44th Symp. on Foundations of Computer Science*, pages 492–501, 2003.

[11] I. Gaber and Y. Mansour. Broadcast in radio networks. In *Proc. of the 6th ACM-SIAM Symp. on Discr. Algorithms*, pages 577–585, 1995.

[12] P. Indyk. Explicit constructions of selectors and related combinatorial structures, with applications. In *Proc. 15th Symp. on Discr. Algorithms*, pages 697–704, 2002.

[13] D. Kowalski and A. Pelc. Deterministic broadcasting time in radio networks of unknown topology. In *Proc. 43th Symp. on Foundations of Computer Science*, pages 63–72, 2002.

[14] D. Kowalski and A. Pelc. Broadcasting in undirected ad hoc radio networks. In *Proc. 16th ACM Symp. on Principles of Distr. Comp.*, pages 73–82, 2003.

[15] D. Kowalski and A. Pelc. Centralized deterministic broadcasting in undirected multi-hop radio network. In *Random-Approx 2004*, pages 171–182, 2004.

[16] E. Kushilevitz and Y. Mansour. An $\omega(d \log(n/d))$ lower bound for broadcast in radio networks. *SIAM J. of Computing*, 27(3):702–712, 1998.

[17] N.Linial and M. Saks. Decomposing graphs into regions of small diameter. In *Proc. 2nd ACM-SIAM Symp. on Discr. Alg.*, pages 320–330, 1991.

[18] V. Prasolov. *Intuitive Topology.* Amer. Math. Soc., 1995.