IMFM
Institute of Mathematics, Physics and Mechanics
Jadranska 19, 1000 Ljubljana, Slovenia

Preprint series
Vol. 53 (2015), 1202
ISSN 2232-2094

BROADCASTING ON CACTUS GRAPHS

Maja Čevnik Janez Žerovnik

Ljubljana, January 19, 2015

Noname manuscript No.
(will be inserted by the editor)

Preprint series, IMFM, ISSN 2232-2094, no. 1202, January 19, 2015

Broadcasting on cactus graphs

Maja Čevnik · Janez Žerovnik

Received: date / Accepted: date

Abstract Broadcasting is the process of dissemination of a message from one vertex (called originator) to all other vertices in the graph. This task is accomplished by placing a sequence of calls between neighboring vertices, where one call requires one unit of time and each call involves exactly two vertices. Each vertex can participate in one call per one unit of time. Determination of the broadcast time of a vertex x in arbitrary graph G is NP-complete. Problem can be solved in polynomial time for trees and some subclasses of cactus graphs. In this paper broadcasting in cactus graphs is studied. An algorithm that determines broadcast time of any originator with time complexity $O(n \log \Delta)$ in k-restricted cactus graph is given. Furthermore, another algorithm which calculates broadcast time for all vertices in k-restricted cactus graph within the same time complexity is outlined. The algorithm also provides an optimal broadcast scheme for every vertex. As a byproduct, broadcast center of a k-restricted cactus graph is computed.

Keywords Broadcasting · Cactus graph · Broadcast time · Broadcast scheme · Broadcast center

1 Introduction

Broadcasting is the process of dissemination of a message from one vertex (called originator) to all other vertices in the graph. This task is accomplished

M. Čevnik
IMFM, Jadranska 19, SI-1000 Ljubljana, Slovenia
Tel.: +386-1-4267177
Fax: +386-1-4267178
E-mail: maja.rotovnik@imfm.si

J. Žerovnik
FME, University of Ljubljana, Aškerčeva 6, SI-1000 Ljubljana, Slovenia
IMFM, Jadranska 19, SI-1000 Ljubljana, Slovenia

by placing a sequence of calls between neighboring vertices, where one call requires one unit of time and each call involves exactly two vertices. Each vertex can participate in one call per one unit of time. The *broadcast time of vertex* u, denoted by $b(u)$, is the minimum number of time units required to complete broadcasting of a message originating at vertex u. The broadcast time of graph G, denoted by $b(G)$ is the maximum over all broadcast times of its vertices, i.e. $b(G) = \max\{b(u) \mid u \in V(G)\}$. The *broadcast center* of a graph G is the vertex v_{BC} which has minimal broadcast time. A *broadcast scheme* of a vertex u is a set of calls that, starting at vertex u, completes the broadcasting in the network (Harutyunyan and Maraachlian, 2008). More precisely, a broadcast scheme of a vertex u is an ordered sequence of calls which provides an optimal visiting order of vertices to complete broadcasting. It is well known that determination of the broadcast time of a vertex x in arbitrary graph G is NP-complete (Garey and Johnson, 2006). A number of papers report on research of approximation or heuristic algorithms to determine the broadcast time of a vertex x in arbitrary graph G (Bar-Noy et al., 1998; Beier and Sibeyn, 2000; Elkin and Kortsarz, 2005, 2006; Feige et al., 1990; Fraigniaud and Vial, 1997a,b, 1999; Harutyunyan and Shao, 2006; Kortsarz and Peleg, 1995; Ravi, 1994; Scheuermann, 1984) Another direction of the research is to design polynomial algorithms that determine the broadcast time of a vertex in special families of graphs. First family of graphs for which was found polynomial algorithm for broadcasting is the trees (Slater et al., 1981). Broadcasting problem is also solved with linear algorithm for some subfamilies of cactus graphs: unicyclic graphs (Harutyunyan and Maraachlian, 2008), necklace graphs (Harutyunyan et al., 2009) and graphs with no intersecting cycles (Harutyunyan and Maraachlian, 2009b). There is also an $O(n \log n)$ broadcast algorithm for fully connected trees (Harutyunyan and Maraachlian, 2009a). Note that cactus graphs are interesting interconnection network topologies as they are a common generalization of trees and ring networks (Ben-Moshe et al., 2012; Elenbogen and Fink, 2007). Furthermore, the cacti are an interesting class of graphs in combinatorial optimization because it is well-known that efficient solutions to many problems can be generalized to cactus graphs, often within the same time complexity (Markov et al., 2012; Zmazek and Žerovnik, 2005).

In this paper we design an algorithm that determines the broadcast time from any originator in arbitrary k-restricted cactus graph and as a side result we give a broadcast scheme of originator. Time complexity of our algorithm is $O(n \log \Delta)$ for any originator. Clearly, a naive approach that computes the broadcast times for all vertices separately and in summary identifies the vertices with minimal broadcast times as centers has time complexity $O(n^2 \log \Delta)$. We show that the naive method can be considerably improved by providing an algorithm for the broadcast time and center of a k-restricted cactus graph that runs in $O(n \log \Delta)$ time. We also determine the broadcast center and broadcast time.

The rest of the paper is organized as follows. In the next section we will give some definitions and a useful representation of a cactus graph. Section 3

Preprint series, IMFM, ISSN 2232-2094, no. 1202, January 19, 2015

presents an algorithm which calculates broadcast time of an arbitrary originator. In Section 4 we provide the algorithm for broadcast time of a cactus. In the final section we summarize the results and discuss possible future work.

2 Preliminaries

2.1 Overview of the main ideas

In this paper we first introduce an algorithm which determines the broadcast time of an arbitrary chosen, but fixed vertex r in a cactus graph G. Given a cactus graph G and a root vertex r, we first run DFS from the root vertex r and label the vertices as they appear in the DFS order: $r = v_0, v_1, ...v_n$. During the DFS run we determine, for each vertex v_i, its father and its cycle root when applicable.

The main task, computing the broadcast time of a vertex will be computed based on the DFS order. First the vertices of G will be visited in the reverse DFS order, and during this phase the temporary broadcast time (to be defined later) will be computed. The computation will locally involve partial results for the subcacti rooted at the vertex currently visited and will depend on the type (tree-like or cycle-like) of the subcacti attached. As locally the time complexity of the algorithm depends on the number of cycles rooted at a vertex, we need the assumption that at most k cycles can meet in one vertex, i.e. we work with k-restricted cacti. Essential is the observation that at the root of G the temporary broadcast time equals the broadcast time.

The broadcast time of a graph G can thus be computed using the DFS order started at all vertices and computed the respective broadcast times. In Section 4 we show that this can be substantially improved. Using the temporary broadcast times achieved when computing the broadcast time of one vertex can be used in one additional phase, this time visiting the vertices in the DFS order, and computing the broadcast times of all vertices.

2.2 Basic definitions

Graph G is an ordered pair $G = (V, E)$, where $V = V(G)$ is the set of *vertices* and $E = E(G)$ is the set of unordered pairs of vertices $(x, y) = xy$ of G called *edges.* As usual, $n = |V(G)|$ and $m = |E(G)|$. Two vertices x and y of graph G are *adjacent* if G contains the edge xy. The *degree* of a vertex x is the number of vertices which are adjacent to x, denoted by $deg(x)$. The maximum and minimum degree of a graph G are respectively denoted by $\Delta(G)$ and $\delta(G)$. A *simple path* from vertex x to vertex y is a finite sequence of distinct vertices $x = x_0, x_1, ..., x_l = y$ such that each pair $x_i x_{i+1}$ is connected by an edge. The *length* of a path is the number of edges on the path. The *distance* $d(x, y)$ between two vertices x and y is the length of a shortest path from x to y. A *connected* graph is a graph such that there exists a path between each pair of vertices. A *cycle* is an induced subgraph which is connected and in which any vertex is of degree two. A *cactus* graph is a connected graph in which any two cycles have at most one vertex in common. Or, equivalently, each edge in a cactus graph meets at most one cycle. A *k-restricted cactus graph* is a cactus graph where no more than k cycles can have more than one vertex in common, or equivalently, a cactus graph in which every vertex is on at most k cycles.

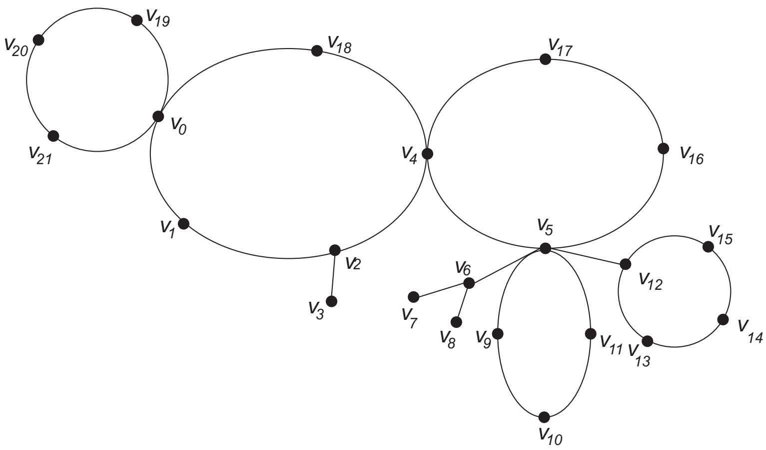

Fig. 1 Example of a k-restricted cactus graph where $k = 2$.

2.3 DFS representation of a cactus graph

It is well-known that cactus graphs can be recognized in linear time (Aho et al., 1974; Brandst, 2000; Hedetniemi et al., 1986). This can be done by a DFS (depth first search) from any vertex of G yielding a DFS representation of a cactus graph, for details see (Zmazek and Žerovnik, 2004). Any cactus graph G can always be represented as a rooted cactus. This means that an arbitrary vertex $v_0 \in V(G)$ is distinguished and called a root of the cactus graph G. All other vertices are indexed as $v_1, v_2, ..., v_{n-1}$ as they appear in a DFS order. In each cycle C of the cactus G there is a unique vertex v_i which is the first vertex of the cycle in the particular DFS order. We call this vertex v_i the root of cycle C. All cycles in a cactus are determined by the root and its successor. Therefore, a rooted cactus can be represented with two arrays F_i and $R_i,\ i = 0, 1, ..., n-1$, where F_i denotes the unique father of vertex v_i and R_i denotes the root vertex of a cycle containing vertex v_i. If v_i is not on a cycle, then $R_i = v_i$ and there is no vertex v_j for which $R_j = v_i$ would hold. If v_i is on a cycle rooted at v_j, then $R_i = v_j \neq v_i$.

Preprint series, IMFM, ISSN 2232-2094, no. 1202, January 19, 2015

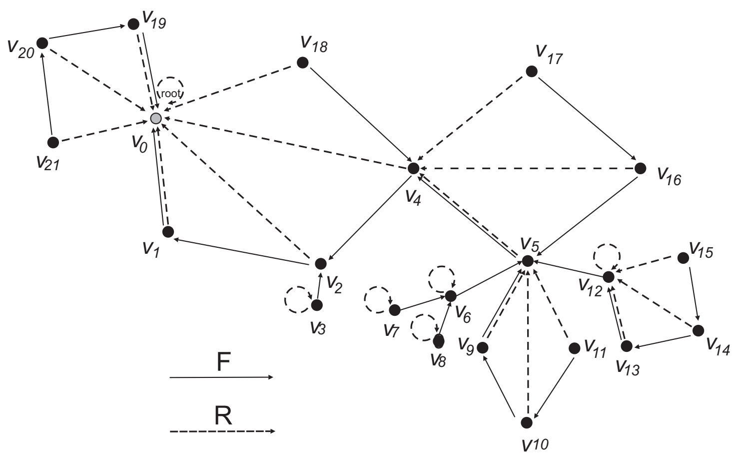

Fig. 2 Pointers F and R on a rooted cactus from Figure 1.

An example of a 2-restricted cactus graph is on Figure 1. The arrays F and R on this graph after a DFS run from vertex 1 are indicated as arcs on Figure 2.

2.4 Tree-like and cycle-like components

We will first study how to compute the temporary broadcast time for two simple structures which we call tree-like components and cycle-like components.

At a vertex u somewhere in the DFS order we may observe several types of vertices that are visited later in the DFS order. First u may lie on a cycle that is rooted by some vertex that appears before u in the DFS order, and there are some later vertices on this cycle. These vertices will be of minor interest because they will be treated when the root of the cycle is considered. Furthermore, there may be some subcacti rooted at u, and we distinguish two types of subcacti that can be attached to u. Loosely speaking, we call them the tree-like and the cycle-like components, because they (from vertex u) either look like a tree or like a cycle. We wish to emphasize that a tree-like component is not necessarily a tree and a cycle-like component is not necessarily a cycle. More precisely, we define

Definition 1 Let uv be an edge which is not on a cycle and let u precede v in the DFS order. The *tree-like component* $\tau_u(v)$ rooted at u and covering v is the subgraph of G that is induced on vertices u, v and all vertices that are closer to v than u in G.

Equivalently, delete the vertex u and let G_v be the connected component of $G - u$ that covers v. The tree-like component rooted at u that covers v is the subgraph of G induced on the vertices $V(G_v) \cup \{u\}$. Example is on Figure 3.

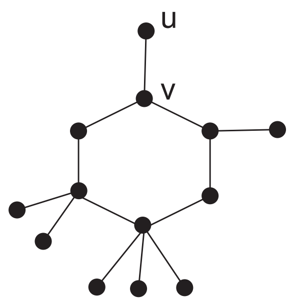

Fig. 3 Example of a tree-like component $\tau_u(v)$. v is the sole neighbor of u.

Definition 2 Let the edge uv lie on a cycle C and let u be the root of cycle C. The *cycle-like component* $\gamma_u(v)$ rooted at u and covering v consists of the cycle C (called the basic cycle) and all the subcacti rooted at vertices $v \in C$, $v \neq u$.

Equivalently, delete the vertex u and let G_v be the connected component of $G - u$ that covers v. The cycle-like component rooted at u that covers v is the subgraph of G induced on vertices $V(G_v) \cup \{u\}$. Example is on Figure 4.

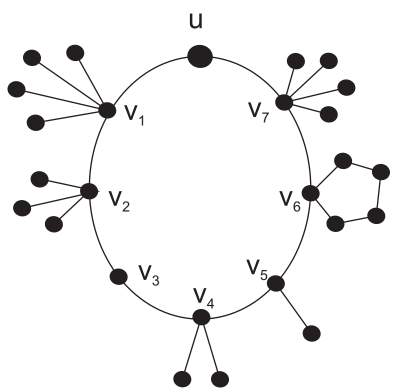

Fig. 4 Example of a cycle-like component $\gamma_u(v_1) = \gamma_u(v_7)$.

3 Temporary broadcast time

In this section we show how the temporary broadcast times are computed. More precisely, given a cactus graph G with the root r, we want to compute the temporary broadcast times for all vertices in G assuming a DFS order of vertices is given. It appears that the temporary broadcast time of the root vertex r is also exactly the broadcast time of vertex r. The temporary broadcast time at vertex u is formally defined as follows:

Definition 3 *Temporary broadcast time of a vertex u* is the number of steps required for u to send message to all its descendants in DFS order except for vertices which are on the same cycle as u and u is not the root of that cycle. Temporary broadcast time of the vertex u is denoted by $T(u)$.

Remark 1 Temporary broadcast time of the root vertex r is the number of steps required to visit all other vertices of a graph and therefore is also the broadcast time of the root vertex r.

When computing the broadcast time of a vertex, we take into account temporary broadcast times of the components (i.e. subcacti) rooted at the vertex. The rules for tree-like components are simple, while the treatment of cycle-like components is somewhat more complicated, because a cycle-like component may either be visited from one, from the other, or from both sides. Several auxiliary values are used to store the information about the cycle-like components that enable the finding of the optimal schedule.

For each vertex u we introduce a queue in which the order of components rooted at u will be stored. This queue consists of a sequence of vertices. Each tree-like component in the queue is presented by a vertex which is the son of u and each cycle-like component is presented either with one or both neighbors of u (depends whether we visit component from one or from both sides).

Definition 4 Visiting queue of the vertex u Q_u is a queue that determines the order in which neighbors of the vertex u will be visited.

Length of the visiting queue Q_u will be at least the number of components of vertex u and at will at most equal the degree of the vertex u. The length depends on how many cycle-like components rooted at u have to be visited from both sides.
Every component of vertex u has a position in queue Q_u. With $Q_u[i]$ we denote the component rooted at u which is stored on $i-th$ place in Q_u. This component will be visited $i-th$ in the row with delay $i-1$.

3.1 Temporary broadcast time of a tree-like component

Observation 1 *Let v be a descendant of the vertex u such that the edge uv is on a tree-like component $\tau_u(v)$ and let $T(v)$ be temporary broadcast time of the vertex v. Then $T(\tau_u(v)) = T(v) + 1$.*

Proof Obvious. □

When all the components rooted at the vertex u are tree-like, the temporary broadcast time of u can easily be found.

Observation 2 *Let $v_1, v_2, \ldots, v_l$ be descendants of the vertex u; let $\tau_u(v_1)$, $\tau_u(v_2), \ldots, \tau_u(v_l)$ be the corresponding tree-like components and $T(v_1)$, $T(v_2)$, $T(v_3)$..., $T(v_l)$ temporary broadcast times of the vertices $v_1, v_2, \ldots, v_l$ respectively. Assume that the descendants are ordered by temporary broadcast time in descending order. Then $T(u) = \max_{i\in\{1,2,\ldots,l\}} \{T(v_i) + i\} + 1$.*

Proof Clear. □

The observation above can be proved by a straightforward generalization of the same fact that holds for trees. Note however that we can not directly use already known theory of broadcasting on trees (Slater et al., 1981), or any other special subclasses of cactus graphs (Harutyunyan et al., 2009; Harutyunyan and Maraachlian, 2008, 2009a,b), because a descendant of the vertex u can be a root of an arbitrary k-restricted subcactus.

3.2 Cycle-like components - more definitions

Let γ be a cycle-like component with the root v_0 and with $v_1, ..., v_l$ as vertices of the basic cycle. We assume that temporary broadcast times of the vertices on the basic cycle $T(v_1), ..., T(v_l)$ and their respectively visiting queues are known.
Before we consider the calculation of a temporary broadcast time of cycle-like component, we introduce some auxiliary notions that will be used in the analysis. (Clearly, $\gamma = \gamma_{v_0}(v_1) = \gamma_{v_0}(v_l)$ because the edges v_0v_1 and v_0v_l define the same component rooted at v_0.)
We first define several auxiliary variables which will be used to explain (and prove) the optimal strategy for broadcasting in minimal time.
When only one cycle is considered, it is important to choose the right strategy how to visit all vertices so that we need minimal number of steps. Namely, at every vertex on the basic cycle we may decide whether to first visit the next vertex on the basic cycle or it is better to visit other descendants before continuing along the cycle. The information on possible strategies for visiting one cycle-like component are useful when we have more components rooted at the same vertex in order to combine optimally the order of visiting the components. In order to study the possible strategies for visiting vertices of one cycle-like component, we will first divide the vertices of the cycle in two parts. We call the part with lower indices "the first half", although this does not imply that it will be visited first. In fact, the whole cycle may be either visited from one side or from both sides, and the starting time for each of the parts may depend on other components that are also rooted at u. This will be explained in detail later.

Definition 5 Let γ be a cycle-like component with the root v_0 and $v_1, v_2, ..., v_l$ vertices on the basic cycle of γ. Then

$$A(\gamma) = A = \max_{1 \leq i \leq \lceil \frac{l}{2} \rceil} \{T(v_i) + i\},$$

$$B(\gamma) = B = \max_{\lceil \frac{l}{2} \rceil + 1 \leq i \leq l} \{T(v_i) + l - i + 1\}.$$

Definition 6 Let γ be a cycle-like component with the root v_0 and $v_1, v_2, ..., v_l$ vertices on the basic cycle of γ. Then $S(\gamma) = S = \min\{S_1, S_2\}$, where $S_1 = \max_{1 \leq i \leq l} \{T(v_i) + i\}$ and $S_2 = \max_{1 \leq i \leq l} \{T(v_i) + l - i + 1\}$.

Roughly speaking, S is the number of steps needed to visit a cycle-like component from one side, while A and B give the number of steps needed to visit one part of the cycle-like component.
When we want to visit vertices of a cycle-like component, we have to decide at each vertex u on the basic cycle whether to first visit the next vertex on the basic cycle and then other descendants of u or it is better to first visit some descendants of vertex u and then go to the next vertex on the basic cycle. The following observations determine optimal visiting strategy for cycle-like components.
We say that we enter cycle-like component from "side A" if we enter component from direction $v_1, v_2, ...$ and similarly we say that we enter cycle-like component from "side B" if we visit cycle-like component from direction $v_l, v_{l-1}, ...$.

Observation 3 *Algorithm 1 returns the last possible vertex on the basic cycle entered from "side A" which (together with its components) still can be visited in t steps.*

Proof Our goal is to visit as many vertices as possible in t steps in order to minimize the remaining part which will have to be visited from the other direction. Therefore, if possible we always go first to the next vertex on the basic cycle and then visit other descendants. In this case, i.e. if we first move to the next vertex on the basic cycle, we can visit other descendants with delay one. From this it is obvious that if temporary broadcast time together with the distance from root is smaller than t, we do not exceed t steps even with delay one. Similarly, if temporary broadcast time of a vertex together with the distance from root exceeds t, it is not possible to visit this vertex and we can conclude that previous vertex on the basic cycle is the last visited vertex. In these cases the algorithm clearly works optimal.
If the temporary broadcast time of a vertex together with a distance from the root is equal t, then we can not first move to the next vertex on a basic cycle without exceeding available number of steps. However in some cases it is possible that we first visit some descendants of the current vertex, then move to the next vertex on the basic cycle and then with additional delay still within t steps visit remaining descendants of the previous vertex. To find out when this is possible, we first look how many components of currently visited vertex

can get delay one and not exceed t and then (if such components exists) we first visit descendants of currently visited vertex for which can not afford delay and then (with some additional delay) move to the next vertex on the basic cycle. This is clearly optimal since we move forward along the basic cycle as soon as component together with delay does not exceed t and therefore continue along basic cycle with smallest possible delay. □

Algorithm 1 Procedure finds how many vertices on basic cycle and their descendants can be visited from "side A" $(v_1, v_2, ..., v_l)$ in t steps. Returns last visited vertex on basic cycle.

```
Input: A cycle-like component and a number of steps available.
i = 1; delay = 1;
while (T(v_i) + delay ≤ t) do
  if (i = l) then
    Return v_l and exit procedure.
    {We visited whole component from one side.}
  end if
  if (T(v_i) + delay < t) then
    delay = delay + 1;
    i = i + 1; {We first move to the next vertex on the basic cycle and then later visit
    vertices in Q_{v_{i-1}}.}
  else
    j = length(Q_{v_i});
    while ((T(Q_{v_i}[j]) + delay + 1 < t)&(j > 0)) do
      j = j − 1; {We search for such position (smallest) in queue that to vertices onward
      it we can add delay one and still do not exceed t.}
    end while
    delay = delay + j + 1;
    i = i + 1;
    {We first visit j vertices in Q_{v_i}. Therefore, delay increases for j.Then we move to the
    next vertex on basic cycle and consequently i increases for one and delay increases
    for one. Then we visit remaining vertices in Q_{v_{i-1}} with new delay.}
  end if
end while
if (T(v_i) + delay > t) then
  Return v_{i-1} and exit procedure.
  {v_{i-1} is the last vertex which can be visited in t steps.}
end if
```

Observation 4 *Algorithm 2 returns the last possible vertex on the basic cycle entered from "side B" which (together with its components) still can be visited in t steps.*

Proof Analogous to Observation 3. Details are omitted. □

Example 1 The graph on Figure 4 has the basic cycle on eight vertices therefore $A = 6$, $B = 5$ and $S = 11$.

Algorithm 2 Procedure finds how many vertices on basic cycle and their descendants can be visited from "side B" $(v_l, v_{l-1}, ..., v_1)$ in t steps. Returns last visited vertex on basic cycle.

```
Input: A cycle-like component and a number of steps available.
i = l; delay = 1;
while (T(v_i) + delay ≤ t) do
   if (i = 1) then
      Return v_1 and exit procedure.
      {We visited whole component from one side.}
   end if
   if (T(v_i) + delay < t) then
      delay = delay + 1;
      i = i − 1; {We first move to the next vertex on the basic cycle and then later visit
      vertices in Q_{v_{i+1}}.}
   else
      j = length(Q_{v_i});
      while ((T(Q_{v_i}[j]) + delay + 1 < t)&(j > 0)) do
         j = j − 1; {We search for such position (smallest) in queue that to vertices onward
         it we can add delay one and still do not exceed t.}
      end while
      delay = delay + j + 1;
      i = i − 1;
      {We first visit j vertices in Q_{uv_i}. Therefore, delay increases for j.Then we move
      to the next vertex on basic cycle and consequently i decreases for one and delay
      increases for one. Then we visit remaining vertices in Q_{v_{i+1}} with new delay.}
   end if
end while
if (T(v_i) + delay > t) then
   Return v_{i+1} and exit procedure.
   {v_{i+1} is the last vertex which can be visited in t steps.}
end if
```

If value S of a cycle-like component is larger as available number of steps then we will have to visit a cycle-like component from both sides. In other words, we will have to divide a cycle-like component to two tree-like components. We have two options how to divide a cycle-like component according to side from which we first enter the component.

We first run Algorithm 1 for available number of steps. Then we have to calculate how large is the remaining part of the cycle-like component. Algorithm 3 calculates how many steps we need to visit remaining vertices of a cycle-like component from "side B" if we know the last visited vertex from "side A".

Remark 2 Considering both possible ways of dividing cycle-like component, it is obviously better to choose the one with smaller remaining part.

We denote the chosen remaining part (smaller tree-like component) with C'' and corresponding first part (bigger tree-like component) with C'.

Algorithm 3 Procedure calculates the number of steps required to visit remaining vertices of cycle-like component from "'side B"' if we know the last visited vertex from "'side A"'. Last visited vertex from "side A" is v_A.

Input: a cycle-like component and last visited vertex from first side.
$t = 0$;
for $(i = A + 1; i \leq l; i + +)$ **do**
  if $(T(v_i) + l - i + 1 > t)$ **then**
    $(t = T(v_i) + l - i + 1)$
  end if
end for
Run Algorithm 2 for t steps and obtain vertex v_B. {Calculates the last visited vertex in t steps from "side B".}
if $B \leq A + 1$ **then**
  $(T(C'') = t)$
else
  $(T(C'') = t + 1)$
end if

Algorithm 4 Procedure calculates the number of steps required to visit remaining vertices of cycle-like component if we know the last visited vertex from "side B". Last visited vertex from "side B" is v_B.

Input: a cycle-like component and last visited vertex from "side B".
$t = 0$;
for $(i = 1; i < B; i + +)$ **do**
  if $(T(v_i)i > t)$ **then**
    $t = T(v_i) + i$
  end if
end for
Run Algorithm 1 for t steps and obtain vertex v_A. {Calculates last vertex still visited in t steps from "side A".}
if $B \leq A + 1$ **then**
  $(T(C'') = t)$
else
  $(T(C'') = t + 1)$
end if

3.3 Temporary broadcast time of a cycle-like component

In this section we will determine temporary broadcast time $T(\gamma)$ of a cycle like component γ rooted at u if we assume that temporary broadcast times of vertices on the basic cycle are known. Because of simpler analysis and due to symmetry we can without loss of generality assume that $A(\gamma) \geq B(\gamma)$. As γ is fixed in this subsection, we will often use A, B, and S instead of $A(\gamma)$, $B(\gamma)$, and $S(\gamma)$. We distinguish two cases $A = B$, and $A > B$.

When the values $A(\gamma)$ and $B(\gamma)$ are equal, it is clear that we need at least $A + 1$ steps to visit all vertices of the component γ, since we have to go into one direction of cycle with delay one. However, $A+2$ steps is always the upper bound for $T(\gamma)$. Clearly, if we follow the rule that for each vertex on the basic cycle always go first to the next vertex on the basic cycle and then visit other descendants, we produce delay at most one. So, for each half of the component we need at most $A+1$ steps and since we go into one side of the component with delay one, $A+2$ steps suffices. Therefore,

Observation 5 *If* $A(\gamma) = B(\gamma)$ *then* $A(\gamma)+1 \leq T(\gamma) \leq A(\gamma)+2$.

Proof Clear. □

Observation 6 *If* $A(\gamma) > B(\gamma)$ *then* $A(\gamma) \leq T(\gamma) \leq A(\gamma)+1$.

Proof It is clear that for visiting all vertices we need at least A steps. The upper bound for the number of required steps is $A+1$, since we can visit the first half of the cycle with at most $A+1$ steps and the second half with $B < A$ steps, hence with delay one we still need at most $A+1$ steps. □

Lemma 1 *Algorithm 5 calculates temporary broadcast time of a cycle-like component.*

Proof Algorithm obviously calculates the number of steps required to visit all vertices of a cycle-like component. If $A = B$ and the calculated value is equal to $A+1$, then by Observation 5, the calculated value is minimal. If calculated value is $A+2$, then we look if we could visit vertices differently and visit all vertices in $A+1$ steps. However, we have already shown that Algorithm 1 and Algorithm 2 work optimal in such manner that they visit as many vertices as possible, therefore any other visiting strategy would give the same or worse result. We can conclude that $A+2$ steps is optimal result.
Arguments for correctness in case where $A > B$ are similar. □

Algorithm 5 Procedure calculates temporary broadcast time of a cycle-like component γ.

```
Input: a cycle-like component γ.
Calculate values A(γ) and B(γ).
if (A = B) then
   v_{i1} =Algorithm1(γ, A + 1), v_{i2} =Algorithm2(γ, A).
   v_{j1} =Algorithm1(γ, A), v_{j2} =Algorithm2(γ, A + 1).
   if (v_{i2} ≤ v_{i1} + 1) or (v_{j2} ≤ v_{j1} + 1) then
      T(γ) = A + 1.
   else
      T(γ) = A + 2.
   end if
else if (A > B) then
   v_{i1} =Algorithm1(γ, A), v_{i2} =Algorithm2(γ, A − 1).
   v_{j1} =Algorithm1(γ, A − 1),v_{j2} =Algorithm2(γ, A).
   if (v_{i2} ≤ v_{i1} + 1) or (v_{j2} ≤ v_{j1} + 1) then
      T(γ) = A.
   else
      T(γ) = A + 1.
   end if
end if
```

3.4 Temporary broadcast time of arbitrary vertex u in k-restricted cactus graph

In the following we will describe how to calculate temporary broadcast time of an arbitrary vertex u in k-restricted graph. We assume that temporary broadcast times of components rooted (or fathered) by u are known.
Let $C_1, C_2, ..., C_l$ be components rooted (or fathered) by vertex u. Since u is a vertex in k-restricted cactus graph, we know that at most k of those components are cycle-like.
With $A(C_i)$ we denote value A of the component C_i. Similarly $B(C_i)$ is the value B of the component C_i and $S(C_i)$ is value S of the component C_i.
Our goal is to find an optimal order in which components of vertex u will be visited.

Definition 7 Assume that components rooted or fathered at the vertex u are sorted descending by value $T : T(C_1) \geq T(C_2), ..., T(C_l)$. Then define

$$M_{min} = \max_{1 \leq i \leq l} \{T(C_i) + i - 1\}. \tag{1}$$

Lemma 2 *To visit all components of the vertex u we need at least $M_{\min}$ steps.*

Proof Since we can enter at most in one component of the vertex u at one time unit, all components except the first one will be visited with some delay. How large will delay of a component be depends on the position of component in the queue. Component C_i which is on $i-th$ place in Q_u will be visited with delay at least $i - 1$. If we look at temporary broadcast times of component together with suitable delays, it is clear that we need at least $M_{\min}$ steps to visit all components rooted by vertex u. □

Remark 3 If all components of the vertex u are tree-like, then $T(u) = M_{\min}$.

Lemma 3 *Let k be the number of cycle-like components rooted at u. To visit all components of a vertex u we need at most $M_{\min} + k$ steps.*

Proof After dividing at most k cycle-like components, the upper bound for the number of steps needed is $\max_{1 \leq i \leq L} \left\{T(\tilde{C}_i) + i - 1\right\}$, where $L \leq l + k$ and $\tilde{C}_*$ are either components C_* or parts of components C_*, possibly reordered. Clearly, $T(\tilde{C}_*) = T(C_*)$ when $\tilde{C}_* = C_*$ and $T(\tilde{C}_*) \leq T(C_*)$ when $\tilde{C}_*$ is a part of C_*. From this we conclude that $\max_{1 \leq i \leq L} \left\{T(\tilde{C}_i) + i - 1\right\} \leq M_{\min} + k$. □

Lemma 4 *Let $C_1, C_2, ..., C_l$ be components rooted at the vertex u and $T(C_1)$, $T(C_2), ..., T(C_l)$ their temporary broadcast times respectively. There is an optimal order in which the components with greater value of T have priority over components with smaller value of T.*

Proof Assume that components rooted at u are sorted in some order $C_1, C_2, ...,$ C_l such that component C_j which is placed after C_i has greater temporary broadcast time. More precisely $T(C_j) > T(C_i)$, $j > i$. We need $T(C_j) + j - 1$ steps to visit all vertices of a component C_j and $T(C_i) + i - 1$ steps to visit all vertices of a component C_i. However if we switch positions of components C_i and C_j, we need $T(C_j) + i - 1$ steps for visit component C_j and $T(C_i) + j - 1$ steps for visit component C_i. Since $i < j$ it follows that $T(C_j) + i - 1 < T(C_j) + j - 1$. It is true that in the same time $T(C_i) + j - 1 > T(C_i) + i - 1$, however we increased smaller component which does not effect negative on temporary broadcast time of vertex u. To conclude we can say that the order in which component with larger temporary broadcast time has priority is at least as good as any other order of components. □

Lemma 5 *Let C_i and C_j be two tree-like components of vertex u such that $T(C_i) = T(C_j)$. Then order of these two components in visiting queue Q_u can be arbitrary.*

Proof Obvious. □

Lemma 6 *Let C_i be a cycle-like component and C_j a tree-like component of vertex u such that $T(C_i) = T(C_j)$. Then there is an optimal order in which the cycle-like component appears before the tree-like component.*

Proof Suppose that a cycle-like component C_{u_i} has to be divided to two tree-like components regardless of the position. Then if we first visit C_j and then divide C_i, we have one step less available to visit vertices from first side (bigger delay). Therefore we can visit less vertices in available number of steps from first side and consequently we get larger remaining part as we would first divide C_i and then visited C_j. Since we tend to get smallest remaining part, it is clearly better to first visit the cycle-like component and then the tree-like component.
When $S(C_j)$ is equal to available number of steps, we first visit the cycle-like component because we can visit the entire component from one side. If we would first visit a tree-like component, then we would have to visit a cycle-like component with additional delay one and therefore we would have to divide it to two tree-like components.
In case when a cycle-like component can be visited without dividing it regardless of the position of those two components, then it is not important which of those components will be visited first. □

If for a vertex u it holds that in its visiting queue there exists a position from which all further components can wait one step longer and still do not exceed available number of steps, we say that in visiting queue there exists "free position".

Remark 4 There can be several free positions in a visiting queue.

When Algorithm 6 ends, we get $T(u)$ and visiting order of components of the vertex u stored in Q_u.

Lemma 7 *$T(u)$ is temporary broadcast time of vertex u.*

Algorithm 6 Algorithm calculates temporary broadcast time of a vertex u.

if u is a leaf or u is on a cycle and has $\delta(u) = 2$ **then**
  $T(u) = 0$. Exit.
else
  Arrange components rooted at the vertex u descending by value T. If more components have equal T, cycle-like components have priority over tree-like components.
  Store current order of components to Q_u.
  $t = M_{\min}$.
  if All components are tree-like **then**
    $T(u) = M_{\min}$. Exit.
  else
    for Each different temporary broadcast time T_s of elements in Q_u (starting with the greatest)) **do**
      $m = t$
      for Every permutation of components with temporary broadcast time T_s **do**
        If necessary divide cycle-like components and insert remaining parts of divided components into Q_u to suitable place.
        Calculate $\max_{v_i, T(v_i)=T_s}\{T(Q_u[i]) + i - 1\}$.
        if $\max_{v_i, T(v_i)=T_s}\{T(Q_u[i]) + i - 1\} > m$ **then**
          $m = \max_{v_i, T(v_i)=T_s}\{T(Q_u[i]) + i - 1\}$.
        end if
      end for
      if $(m > t)$ **then**
        $t = t + 1$;
        Return to Step 9
      end if
    end for
  end if
end if

Proof It is easy to see that if we follow the algorithm written above, we get time in which all components of vertex u can be visited. We need to show that this time is optimal.
If $T(u) = M_{\min}$, our algorithm is obviously optimal. Therefore, assume that $T(u) > M_{\min}$.
As we check all possible orders of cycle-like components and from them choose the best result, other visiting orders would give equal or worse result by lemmas 4, 5 and 6. Therefore we can conclude that algorithm gives the minimum number of steps required to visit all components rooted by u. □

Example 2 Let us calculate temporary broadcast time of vertices v_{21}, v_{18} and v_5 from Figure 1. First, observe that $T(\tau_{v_5}(v_6)) = 3$, $T(\gamma_{v_5}(v_9)) = 2$ and $T(\tau_{v_5}(v_{12})) = 3$ and $T(v_{21}) = 0$ since v_{21} is on a cycle and it is not a root of any component. $T(v_{18}) = 2$ because v_{18} is on a cycle and $\delta(v_{18}) = 2$.
In order to calculate $T(v_5)$ we first arrange components in descending order by T : $\tau_{v_5}(v_6)$, $\tau_{v_5}(v_{12}), \gamma_{v_5}(v_9)$. Note that $\tau_{v_5}(v_6)$ precedes $\tau_{v_5}(v_{12})$ because the first is a cycle-like component while the second is a tree-like component. Hence $M_{\min} = \max\{3, 3+1, 2+2\} = 4$. Then we check if we can visit all components in 4 steps. The first two components clearly can be visited in 4 steps, but the third component is a cycle-like and can not be visited from one side in 4 steps (including delay 2). Therefore we have to divide it into two

tree-like components C' and C'', where $T(C') = 2$, $T(C'') = 1$. Insert these two tree-like components in queue and get: $\tau_{v_5}(v_6), \tau_{v_5}(v_{13}), C', C''$. Now it is easy to see that all components can be visited in 4 steps, therefore $T(v_5) = 4$.

3.5 Time complexity of calculating $T(u)$

In this section we show how to determine the broadcast time of an arbitrary chosen, fixed vertex u of the k-restricted cactus graph and estimate the time complexity.

Computation of the broadcast time of vertex u proceeds as follows. We first run a DFS search from u as a root vertex and denote the vertices as appear in this DFS order with $u = v_0, v_1, ..., v_{n-1}$. Then we calculate temporary broadcast time for every vertex v_i in the k-restricted cactus graph rooted at v_0. We traverse the graph in reverse DFS order so that when we calculate temporary broadcast time of a vertex v_i we already know temporary broadcast times of all of components rooted at v_i. Temporary broadcast time of a vertex v_i, $i \in \{n-1, ..., 1, 0\}$ is calculated using Algorithm 6.

Observation 7 *If u is a root vertex of DFS order in a k-restricted cactus graph, then $b(u) = T(u)$.*

Proof Observation directly follows from Definition 3. □

It is clear that if we follow the order of components in the queue Q_u, we get the broadcast scheme of vertex u. Obviously, broadcast schemes for all vertices provide all information needed for the broadcast scheme of the graph.

Lemma 8 *Time complexity of calculating temporary broadcast time of vertices in a cactus graph is $O(kk!m + n \log \Delta)$.*

In the following proof we will need some more notation. Let $C(G)$ denote a set of all cycle-like components in a k-restricted cactus graph G. For each cycle-like component $C_i \in C(G)$, let d_{C_i} be the length of its basic cycle.

Proof (of Lemma 8) Recall that calculating temporary broadcast time of vertices consists of several major steps:

1. First step is to obtain a representation of rooted cactus graph and indexing vertices in DFS order. It is known (Zmazek and Žerovnik, 2004) that time complexity of this step is $O(n)$.
2. Second step is, for each vertex, to calculate temporary broadcast times of individual tree-like and cycle-like components rooted at the vertex, and then for each vertex calculate lower bound for temporary broadcast time and arrange components rooted at this vertex by temporary broadcast times. In particular,

Preprint series, IMFM, ISSN 2232-2094, no. 1202, January 19, 2015

(a) From Observation 1 we know that temporary broadcast time of a tree-like component can be obtained in constant time if we already know temporary broadcast time of the descendant which is a sole neighbor. Since we visit vertices in reverse DFS order, this temporary broadcast time is known. From this we conclude that for calculating temporary broadcast times of all tree-like components we need at most $O(n)$ time.
(b) Calculating T of a cycle-like component requires more work. For each cycle-like component C_{u_i} we have to calculate values A, B and S. Time complexity of calculating those values depends on the length of the basic cycle. For each cycle like component, time complexity is $O(d_{C_{u_i}})$. Because sum of lengths of all cycles in cactus graph is at most equal to number of edges, we conclude that time complexity of this step is $\sum_{C_{u_i} \in C(G)} O(d_{C_{u_i}}) = O(m)$.
(c) Having computed A, B and S, Algorithm 1 and Algorithm 2 are called twice for every cycle-like component. Time complexity of those algorithms is the same and depends on length of the basic cycle and degrees of vertices on the basic cycle. We need $\sum_{C_j \in C(G)} \sum_{v_i \in C_j} \delta(v_i)$ time for this computation at v_i. Since every vertex can lie in at most k cycle-like components, it holds $\sum_{v_i \in C_j \in C(G)} \delta(v_i) \leq k \sum_{v_i \in V(G)} \delta(v_i)$. From the handshaking lemma we know that $\sum_{v_i \in V(G)} \delta(v_i) = 2m$, so the total time complexity over all vertices is $O(km)$.
(d) For each vertex we have to sort components descending by value T. Thus for each vertex v_i we need at most $\delta(v_i) \log(\delta(v_i))$ steps. The number of all components in graph is at most equal to sum of degrees of vertices. And since we know that $\sum_{v_i \in V(G)} \delta(v_i) = 2m$, we can estimate that $\sum_{v_i \in V(G)} \delta(v_i) \log(\delta(v_i)) \leq 2m \log(\Delta)$. We conclude that time complexity of this part is $O(2m \log \Delta) = O(m \log \Delta)$.
(e) Next step is to calculate value $M_{\min}$ for each vertex. For every vertex the value depends on length of visiting queue. Since the length of visiting queue is at most equal to vertex degree, for each vertex v we need at most $O(\delta(v))$ time. Therefore for all vertices we need at most $\sum_{v \in V(G)} \delta(v)$ time. As already mentioned $\sum_{v \in V(G)} \delta(v) = 2m$, therefore time complexity of this step is $O(m)$.

3. Finally, for every vertex u we calculate temporary broadcast time. Vertex u is a father of some tree-like components and a root to $k_i \leq k$ cycle-like components. Assume that the number of components in queue Q_u is l, where $l \leq \delta(u_i)$. These components have to be arranged in queue by value T descending. Each sorting of components has time complexity $l \log l$. Overall, the work requires $O(n \log \Delta)$ time.
When sorting the components, it can occur that several values have same temporary broadcast time. For each different value T (starting with greatest value) it is necessary to decide which of the cycle-components with that temporary broadcast time will have to be divided. To do that, we check all possible orders of cycle-like components, for each order we divide cycle-like components if necessary and insert remaining parts into Q_u to appropriate

places. Then among all possible orders we choose the best solution. Since vertex u can be the root of k_i cycle-like components, the number of possible orders of components is $k_i!$. Hence for each value of T, Algorithm 1 and Algorithm 4 are called at most $k_i k_i!$ times. Algorithms 1 and 4 are linear and it follows that for deciding which is the best order of cycle-like components for all different values of T in Q_u, we need $O(kk!m)$ time. Hence, for this part we need $O(kk!m + n \log \Delta)$ time.

Summarizing, as in cactus graph $O(m) = O(n)$ and because above analyzed steps are independent, we conclude that time complexity of calculating temporary broadcast times of all vertices is $O(kk!m + n \log \Delta)$. □

Remark 5 For k-restricted cactus graph, k is fixed and therefore $k!$ is constant.

Theorem 8 *Time complexity of calculating temporary broadcast time of vertices in k-restricted cactus graph is $O(n \log \Delta)$.*

Proof Follows from Lemma 8 and Remark 5. □

4 Broadcast time and broadcast center of a *k*-restricted cactus graph

To determine broadcast time of a k-restricted cactus graph, we have to calculate broadcast time of every vertex in the graph. In previous section we already described how to calculate broadcast time of arbitrary but fixed vertex. If we repeat the procedure for every vertex in cactus graph, then maximum over all broadcast times of vertices is the broadcast time of cactus graph. More precisely, $b(G) = \max_{u \in V(G)} \{b(u)\}$.

Observation 9 *Broadcast time can be found in $O(n^2 \log \Delta)$ time.*

Proof Run Algorithm 6 from each vertex and take maximum broadcast time. □

Observation 10 *Broadcast center can be found in $O(n^2 \log \Delta)$ time.*

Proof Run Algorithm 6 from each vertex, take the vertex with smallest broadcast time. □

The time complexity of calculating broadcast time of a k-restricted cactus graph in the naive manner described above is $O(n^2 \log \Delta)$. However, as we will show below, it is possible to calculate broadcast time of cactus graph with lower time complexity.

In continuation we will explain how to calculate broadcast time of every vertex in k-restricted cactus graph using already calculated temporary broadcast times of vertices. We assume that we have all information obtained in the first pass of vertices in reverse DFS order stored and we can use them. We will

traverse the graph in DFS order, so that when calculating broadcast time of the vertex v_i, broadcast times of vertices with smaller indices in DFS order are already known.
When we calculated temporary broadcast time of a vertex v_i, we have calculated how many steps we need to visit vertices and their descendants which were rooted (or fathered) at v_i. In order to calculate broadcast time of vertex v_i we have to calculate how many steps we need to visit remaining part of graph. To do that, we temporary look at the graph from perspective of v_i (regard v_i as temporary root vertex). We can see from this perspective that when we have calculated temporary broadcast time of v_i, we took into consideration all components except one. This is the component which contains original root (father) of the vertex v_i. We will denote this component with $\bar{C}_{v_i}$. Therefore if we want to calculate broadcast time of all vertices, we have to calculate temporary broadcast time of component $\bar{C}_{v_i}$ for each vertex and insert it into visiting queue on suitable place.
To calculate temporary broadcast time of component $\bar{C}_{v_i}$ we have to know temporary broadcast times of vertices in this component. Temporary broadcast time of every vertex $u_j, j \in \{\bar{C}_{v_i}\}$ will be denoted with $\bar{T}(v_j)$.

Definition 8 Assume that v_j is a root (or a father) of v_i in original DFS order. If we remove component which contains vertex v_i from visiting queue Q_{v_j}, we get a new queue. This queue is denoted by $\bar{Q}_{v_j(-v_i)}$.

When we were calculating temporary broadcast time of vertices in k-restricted cactus graph, we saved some important information which we will use to calculate broadcast time of vertices. In particular, we have stored temporary broadcast times and visiting queues of all vertices. In visiting queue of vertex v_j Q_{v_j} is stored list of neighbors of the vertex v_j which determine local broadcast scheme of vertex v_j. It is also known which of neighbors represents tree-like an which cycle-like components and that when two elements of the queue lie on the same component, then this component was a cycle-like component which was divided during process.
Component $\bar{C}_{v_i}$ can be tree-like or cycle-like. Example of the case when the component $\bar{C}_{v_i}$ is tree-like is on Figure 5 and example when the component $\bar{C}_{v_i}$ is cycle-like is on Figure 6.
Since the case when $\bar{C}$ is a tree-like component is easier to study, we will first show how to calculate broadcast time of vertex v_i if $\bar{C}_{v_i}$ is a tree-like component.

Definition 9 Let $\bar{C}_{v_i}$ be a tree-like component and v_j a father of v_i in original DFS order. Then $\bar{T}(v_j)$ is the result of Algorithm 6 on C_{v_i} rooted at v_i.

Remark 6 One could write $\bar{T}(v_j)$ =Algorithm 6(C_{v_i}, v_i).

Let v_j be the father of v_i in the original DFS order, therefore component $\bar{C}_{v_i}$ is tree-like component with root v_j. We have to calculate temporary broadcast

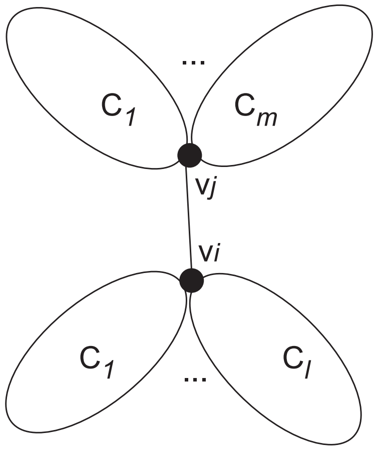

Fig. 5 Case when the component $\bar{C}_{v_i}$ is tree-like (v_j is the father of v_i).

time of component $\bar{C}_{v_i}$. It is clear that $\bar{T}(v_j)$ tells us how many steps we need to visit all vertices in graph except those which have v_i as one of ancestors. Consequently, $T(\bar{C}_{v_i}) = \bar{T}(v_j) + 1$. The next step is to insert this component into new Q_{v_i} and then calculating broadcast time of vertex v_i is simple.

Lemma 9 *Let v_j be the father of vertex v_i and $\bar{C}_{v_i}$ be tree-like component. Then $b(v_i) = \bar{T}(v_j)$.*

Proof We want to calculate $b(v_i)$. Since we temporary look at v_i as a root vertex, v_i is a root of all components also rooted at v_i in original DFS order and a root of a new component $\bar{C}_{v_i}$. It would be necessary to calculate new temporary broadcast time of v_i. However, since we already had original Q_{v_i} which contained visiting order of components rooted at v_i in original DFS order, we have only insert component $\bar{C}_{v_i}$ on appropriate place and then calculate greatest value together with respectively delay in in queue. □

Now we study the case when $\bar{C}_{v_i}$ is a cycle like component. First we introduce some more notation.

Let C_{v_0} be a cycle-like component with root vertex v_0 and the other vertices on the basic cycle $v_1, v_2, ..., v_l$. Then with $\bar{C}_{v_i}$ we denote the component rooted at v_i which has the same basic cycle as component C_{v_0}. More formally,

Definition 10 Let C_{v_0} be a cycle-like component with the root vertex v_0 and vertices on basic cycle $v_1, v_2, ..., v_l$. With $\bar{C}_{v_i}$ we denote a component rooted at v_i for which $V(C_{v_0}) = V(\bar{C}_{v_i})$.

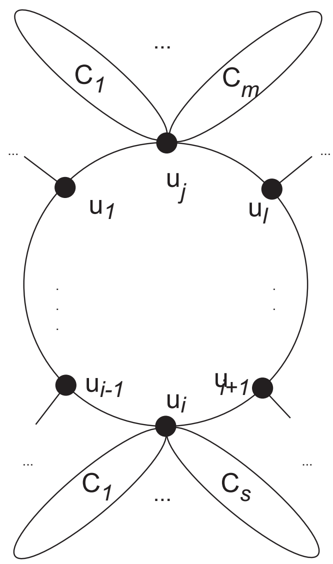

Fig. 6 Case when the component $\bar{C}_{v_i}$ is cycle-like (u_j is the root of u_i).

We obtain the component $\bar{C}_{v_i}$ from C_{v_i} so that we take the basic cycle of the component C_{v_i} and redefine the root vertex. Originaly, the root vertex was v_j while in $\bar{C}_{v_i}$ the root vertex is v_i, another vertex on the basic cycle. Since we want to calculate temporary broadcast time of the component $\bar{C}_{v_i}$ with Algorithm 5, we need to know new temporary broadcast times of vertices on the basic cycle (all but the root). Note that for vertices which are neither the new nor the original root vertex, the temporary broadcast time does not change. Hence $\bar{T}(C_{v_j})$ is the only temporary brodacast time that is not known from the first DFS run. However, it is easy to see that $\bar{T}(C_{v_j})$ is actually broadcast time of the vertex v_j where we exclude component C_{v_i}. Finally, observe that $\bar{T}(C_{v_i})$ is actually the broadcast time of vertex v_i and can be calculated taking into account all the components rooted at v_i. More formally,

Definition 11 Temporary broadcast time of the vertex v_0 of the component $\bar{C}_{v_i}$ is $\bar{T}(v_0) = \max_{1 \leq l \leq \bar{Q}_{v_0(-v_i)}} \{\bar{Q}_{v_0(-v_i)}[l] + l - 1\}$.

Definition 12 Temporary broadcast time of vertex $v_j, j \in \{1, 2, ..., l\}$ on a basic cycle of the component $\bar{C}_{v_i}$ is $\bar{T}(v_j) = T(v_j)$.

Remark 7 Temporary broadcast time of the component $\bar{C}_{v_i}$ can be calculated by Algorithm 5.

When we calculate $T(\bar{C}_{v_i})$, we insert it into a new queue Q_{v_i}. Then broadcast time of a vertex v_i can be easily calculated.

Lemma 10 *Let v_0 be the root of a cycle like component containing v_i. If $\bar{T}(\bar{C}_{v_i})$ is known and inserted into Q_{v_i}, then $b(v_i) =$ is the result of Algorithm 6 on C_i rooted at v_i.*

Proof Analogous as previously. □

In order to calculate broadcast time of all vertices in a k-restricted cactus graph, we pass through the graph in DFS order. Since we already know broadcast time of the root vertex, we start with the next vertex in DFS order. For each vertex v_i we have to first find the component $\bar{C}_{v_i}$ and then regardless of the type of the component (tree-like, cycle-like) calculate $\bar{T}(\bar{C}_{v_i})$. Finally we calculate $b(v_i)$ following Lemma 9 if $\bar{C}_{v_i}$ is a tree-like component and by Remark 7 if $\bar{C}_{v_i}$ is a cycle-like component.

Lemma 11 *Time complexity of finding broadcast time of arbitrary vertex in k-restricted cactus graph is $O(n \log \Delta)$.*

Proof In second DFS pass through a k-restricted cactus graph we have less work than we had in first DFS pass through the graph, therefore time complexity does not change and it is $O(n \log \Delta)$. Details omitted. □

Example 3 We will calculate broadcast time of graph from Figure 1. First, we determine temporary broadcast of vertices in reverse DFS order. When we calculated all values T, we already calculated $b(v_0) = 7$. Now, if we want to calculate $b(v_1)$, we have to calculate $T(\bar{C}_{v_1})$. $\bar{C}_{v_1}$ is a component which is temporary rooted (only for calculating $b(v_1)$) by vertex v_1. If we follow description in Section 4.3.1, we see that $\bar{T}(v_2) = T(v_2) = 1$, $\bar{T}(v_4) = T(v_4) = 5$, $\bar{T}(v_{18}) = T(v_{18}) = 0$ and $\bar{T}(v_0) = b_{v_0(-v_1)} = 2$. Then we can calculate $\bar{T}(C_{v_1}) = 7 = b(v_1)$ since this is the only component of v_1.
Similarly we calculate all other vertices of graph G. Calculated values of temporary broadcast times and broadcast times of vertices are in Table 1.

v_i	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
$T(v_i)$	7	0	1	0	5	4	2	0	0	0	0
$b(v_i)$	7	7	6	7	5	5	6	7	7	6	7
v_i	v_{11}	v_{12}	v_{13}	v_{14}	v_{15}	v_{16}	v_{17}	v_{18}	v_{19}	v_{20}	v_{21}
$T(v_i)$	0	2	0	0	0	0	0	0	0	0	0
$b(v_i)$	6	6	7	8	7	6	6	6	8	9	8

Table 1 Temporary broadcast times and broadcast times of vertices from Figure 1.

Recall that broadcast center of k-restricted cactus graph G is the vertex with minimal broadcast time. More formally, v_{BC} is vertex for which $b(v_{BC}) = \min_{1 \le i \le n}\{b(v_i)\}$ Broadcast time of graph G is

$$b(G) = \max_{1 \le i \le n}\{b(v_i)\}.$$

Corollary 1 *Broadcast time of k-restricted cactus graph can be found in $O(n \log \Delta)$ time.*

Proof Follows from Lemma 11. □

Corollary 2 *Broadcast center of k-restricted cactus graph can be found in $O(n \log \Delta)$ time.*

Proof Since we can calculate broadcast time of every vertex in k-restricted cactus graph in $O(n \log \Delta)$ time, we only have to search for a vertex with smallest broadcast time. This can be done in $O(n)$ time and consequently broadcast center can be found in $O(n \log \Delta)$ time. □

As we can see from the example above, there can exists more than one broadcast center. Vertices v_5 and v_6 are broadcast centers and they lie on the same cycle.

Observation 11 *Broadcast center of $k-$restricted cactus graph is either one vertex or two end vertices of an edge or more vertices which all lie on the same cycle.*

Examples of graphs with more than one broadcast center are on Figure 7 and Figure 8.

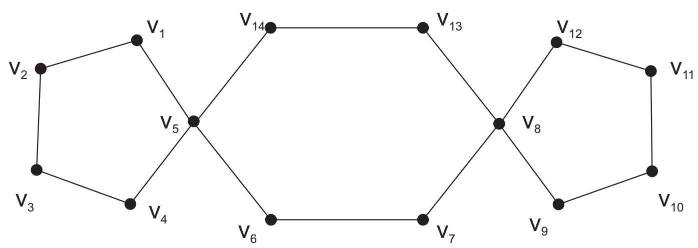

Fig. 7 Vertices which form broadcast center are v_6, v_7, v_{13} and v_{14}. They are on a same cycle, however they are not all adjacent among themselves.

5 Conclusion

In this paper we have designed an algorithm that determines the broadcast time from any originator in arbitrary cactus graph and as a side result it also gives a broadcast scheme of the originator. Time complexity of the algorithm

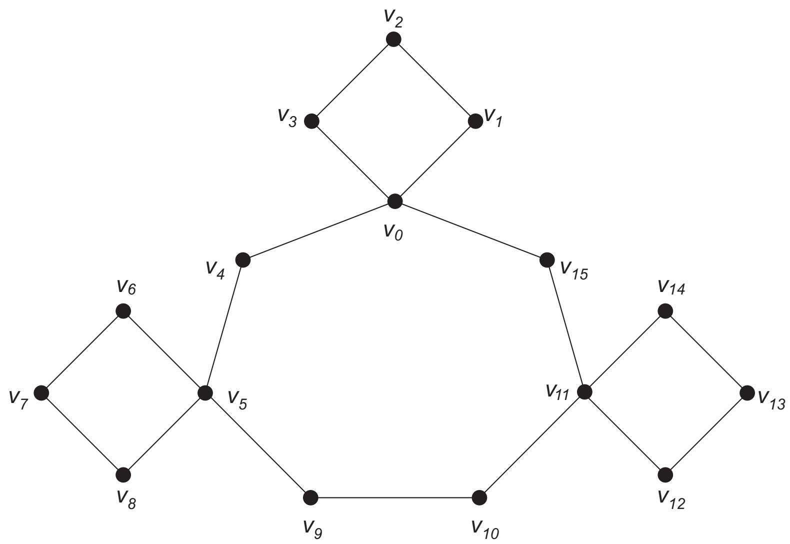

Fig. 8 Vertices which form broadcast center are v_0, v_4, v_{15}. The broadcast center is a path on three vertices on the central cycle.

is $O(kk!m + n \log \Delta)$, which is polynomial on k-restricted cactus graphs. Unfortunately, the algorithm has superpolynomial time complexity on the class of arbitrary cactus graphs. Existence of a polynomial time solution of the problem on cacti thus remains an open problem.

We also provided an algorithm which finds the broadcast time of any vertex of a cactus graph and runs within same time complexity. As a side result, broadcast center of the k-restricted cactus graph and optimal broadcast scheme from every vertex is obtained. Our main result is that this algorithm has the same time complexity as computing broadcast time of only one vertex, i.e. $O(n \log \Delta)$ on k-restricted cactus graphs.

Finally, considering several examples, we observed that the broadcast center of a cactus is always either a vertex, an edge or several vertices on a single central cycle. We did not attempt prove Observation 11, though we believe that the proof is probably not too difficult. However, besides formaly proving the observation, it would be interesting to analyze in more detail the possible configurations of broadcast centers of a cactus graphs.

References

A. V. Aho, J. E. Hopcroft, J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA, (1974).

A. Bar-Noy, S. Guha, J. Naor, B. Schieber. Multicasting in Heterogeneous Networks. Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing, (1998).

R. Beier, J. F. Sibeyn, *A powerful heuristic for telephone gossiping.* Max-Planck-Inst. für Informatik, Bibliothek & Dokumentation, (2000).

B. Ben-Moshe, A. Dvir, M. Segal and A. Tamir, Centdian Computation in Cactus Graphs. *Journal of Graph Algorithms and Applications*, **16(2)**, 199-224, (2012).

A. Brandstädt, V. B. Le, J. P. Spinrad, Graph Classes. A Survey SIAM, Philadelphia, PA, (2000).

B. Elenbogen, J. F. Fink, Distance distributions for graphs modeling computer networks. *Discrete Applied Mathematics*, **155(18)**, 2612-2624, (2007).

M. Elkin, G. Kortsarz, A combinatorial logarithmic approximation algorithm for the directed telephone broadcast problem. *SIAM Journal on Computing*, **35(3)**, 672–689, (2005).

M. Elkin, G. Kortsarz, Sublogarithmic approximation for telephone multicast. *Journal of Computer and System Sciences*, **72(4)**, 648–659, (2006).

U. Feige, D. Peleg, P. Raghavan, E. Upfal, Randomized broadcast in networks. *Random Structures and Algorithms*, **1(4)**, 447–460, (1990).

P. Fraigniaud, S. Vial, Approximation algorithms for broadcasting and gossiping. *Journal of Parallel and Distributed Computing*, **43(1)**, 47–55, (1997).

P. Fraigniaud, S. Vial, Heuristic algorithms for personalized communication problems in point-to-point networks. Proceedings of the 4th Colloquium on Structural Information and Communication Complexity, SIROCCO 97, 240-252, (1997).

P. Fraigniaud, S. Vial, Comparison of heuristics for one-to-all and all-to-all communications in partial meshes. *Parallel Processing Letters*, **9(1)**, 9–20, (1999).

M. Garey, D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, CA, (1979).

H. A. Harutyunyan, B. Shao, An efficient heuristic for broadcasting in networks. *Journal of Parallel and Distributed Computing*, **66(1)**, 68–76, (2006).

H. A. Harutyunyan, G. Laza, E. Maraachlian, Broadcasting in necklace graphs. Proceedings of the 2nd Canadian Conference on Computer Science and Software Engineering C3S2E09, 253–256, (2009).

H. A. Harutyunyan, E. Maraachlian, On broadcasting in unicyclic graphs. *Journal of Combinatorial Optimization*, **16**, 307–322, (2008).

H. A. Harutyunyan, E. Maraachlian, Broadcasting in fully connected trees. International Conference on Parallel and Distributed Systems, ICPADS09, 740-745, (2009).

H. A. Harutyunyan, E. Maraachlian, Linear algorithm for broadcasting in networks with no intersecting cycles. International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA09, 296–301, (2009).

S. T. Hedetniemi, R. Laskar, J. Pfaff, A linear algorithm for finding a minimum dominating set in a cactus. *Discrete Applied Mathematics*, **13**, 287–292, (1986).

G. Kortsarz, D. Peleg, Approximation algorithms for minimum time broadcast. *SIAM Journal on Discrete Mathematics*, **8**, 401–427, (1995).

M. Markov, M. Ionut Andreica, K. Manev, N. Tapus, A linear time algorithm for computing longest paths in cactus graphs. *Serdica Journal of Computing*, **6(3)**, 287–298, (2012).

R. Ravi, Rapid rumor ramification: Approximating the minimum broadcast time. In Foundations of Computer Science. 35th Annual Symposium on Foundation of Computer Science, 202–213, (1994).

P. Scheuermann, G. Wu, Heuristic algorithms for broadcasting in point-to-point computer networks. *IEEE Transactions on Computers*, **100(9)**, 804–811, (1984).

P. J. Slater, E. J. Cockayne, S. T. Hedetniemi, Information dissemination in trees. *SIAM Journal on Computing*, **10**, 692–701, (1981).

B. Zmazek, J. Žerovnik, The obnoxious center problem on weighted cactus graphs. *Discrete Applied Mathematics*, **136**, 377–386, (2004).

B. Zmazek, J. Žerovnik, Estimating the traffic on weighted cactus networks in linear time. International Conference on Information Visualisation, 536–541, (2005).