Theoretical Computer Science 621 (2016) 22–36

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Constructing minimum changeover cost arborescenses in bounded treewidth graphs

Didem Gözüpek [a,1], Hadas Shachnai [b], Mordechai Shalom [c,d,*,2], Shmuel Zaks [b]

[a] Department of Computer Engineering, Gebze Technical University, Kocaeli, Turkey
[b] Department of Computer Science, Technion, Haifa 3200003, Israel
[c] TelHai Academic College, Upper Galilee, 12210, Israel
[d] Department of Industrial Engineering, Bogaziçi University, Istanbul, Turkey

ARTICLE INFO

Article history:
Received 30 June 2015
Received in revised form 9 December 2015
Accepted 16 January 2016
Available online 27 January 2016
Communicated by P. Widmayer

Keywords:
Reload cost
Changeover cost
Cactus graphs
Bounded treewidth graphs
Network design

ABSTRACT

Given an edge-colored graph, an internal vertex of a path experiences a reload cost if it lies between two consecutive edges of different colors. The value of the reload cost depends only on the colors of the traversed edges. The reload cost concept has important applications in dynamic networks, such as transportation networks and dynamic spectrum access networks. In the *minimum changeover cost arborescence (*MinCCA*)* problem, we seek a spanning tree of an edge-colored graph, in which the sum of reload costs of all internal vertices, starting from a given root, is minimized. In general, MinCCA is known to be hard to approximate within factor $n^{1-\epsilon}$, for any $\epsilon > 0$, on a graph of n vertices.
We first show that MinCCA can be optimally solved in polynomial-time on cactus graphs. Our main result is an optimal polynomial-time algorithm for graphs of bounded *treewidth*, thus establishing the solvability of our problem on a fundamental subclass of graphs. Our results imply that MinCCA is *fixed parameter tractable* when parameterized by treewidth and the maximum degree of the input graph.

1. Introduction

1.1. Background

Reload cost in an edge-colored graph refers to the cost incurred when crossing a vertex along a path where the incident edges have distinct colors. The value of the reload cost depends on the colors of the crossed edges. Two common models relate to this concept: the *reload cost* model, and the *changeover cost* model. In the former, the cost is proportional to the amount of commodity flowing through the edges, whereas in the latter, the cost is independent of the amount of commodity.

The reload cost concept has important applications in transportation networks and in dynamic spectrum access networks. Different modes of transportation can be represented by different edge colors. Loading and unloading of cargo at

* Corresponding author.
E-mail addresses: didem.gozupek@gtu.edu.tr (D. Gözüpek), hadas@cs.technion.ac.il (H. Shachnai), cmshalom@telhai.ac.il (M. Shalom), zaks@cs.technion.ac.il (S. Zaks).

[1] This work is supported by the Scientific and Technological Research Council of Turkey (TUBITAK) under grant no. 113E567.
[2] The work of this author is supported by programme 2221 of the Scientific and Technological Research Council of Turkey (TUBITAK).

http://dx.doi.org/10.1016/j.tcs.2016.01.022
0304-3975/

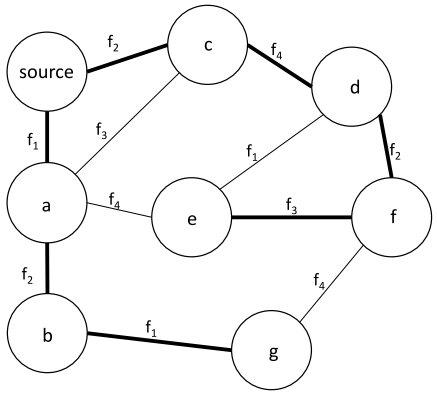


Fig. 1. An application for MinCCA: Energy-efficient broadcasting of control traffic in a multi-hop cognitive radio network.

transfer points has a significant cost, which can be represented using reload costs [21]. Dynamic spectrum access networks aim to address the inefficient spectrum usage in wireless networks by having intelligent wireless devices that dynamically and opportunistically use the frequency bands that are temporarily unused by the licensed owners. Available spectrum resources change dynamically and the intelligent wireless devices, called cognitive radios, need to sense the environment and dynamically adapt their communication parameters according to the network and user demands, as well as time-varying spectrum availability [1,20]. Unlike other wireless networks, dynamic spectrum access networks use a wide range of spectrum. Therefore, the assigned frequency bands can be significantly far away from each other. Consequently, switching from one frequency band to another in a dynamic spectrum access network has a significant cost in terms of delay and energy consumption [3,5,15]. The concept of reload cost has applications also in telecommunication and in energy distribution networks. In communication networks, the cost of switching between different technologies or service provider networks corresponds to reload cost. In energy distribution networks, transfer of energy from one type of carrier to another leads to energy loss corresponding to reload cost.

1.2. Related work

While reload costs naturally arise in numerous applications, the resulting optimization problems received little attention so far. The notion of reload cost was introduced in [21], where the authors considered the problem of finding a spanning tree having minimum diameter with respect to reload cost. In particular, they proved that this problem cannot be approximated within any polynomial-time computable function $f(n)$ on a graph of n vertices. They also showed that the problem cannot be approximated within ratio better than 3, even on graphs with maximum degree 5, and proposed a polynomial-time algorithm for graphs having maximum degree 3. These results were complemented in [10] by showing that, for arbitrary reload costs, the problem cannot be approximated within factor better than 2, even on graphs with maximum degree 4. For reload costs satisfying the triangle inequality, the problem cannot be approximated within ratio better than $5/3$.

The paper [12] considers the *minimum reload cost cycle cover* problem, which is to find a set of vertex disjoint cycles spanning all vertices, i.e., a 2-factor, with minimum total reload cost. The authors prove that the problem is NP-hard in the strong sense and inapproximable within factor $1/\epsilon$ for any $\epsilon > 0$, even when the number of colors is 2 and the reload costs are symmetric in addition to satisfying the triangle inequality. The paper also presents some integer programming formulations and computational results.

In [14], the authors study the problem of finding a path, trail, or walk of minimum total reload cost between two given vertices. They consider (a)symmetric reload costs and reload costs with(out) triangle inequality. In particular, they identify numerous polynomial and NP-hard cases. They prove that the case of walks is solvable in polynomial time, as earlier pointed out in [21], and re-proved later for directed graphs in [2].

The paper [13] studies the problem of finding a spanning tree that minimizes the sum of reload costs over the paths between *all* pairs of vertices. The authors give a proof of NP-hardness, as well as some integer programming formulations for this problem. The paper [2] presents various path, tour, and flow problems related to reload costs. One of the studied problems is *minimum reload cost path-tree*, which is to find a spanning tree that minimizes the total reload cost from a source vertex to all other vertices. The paper shows that the problem is hard to approximate within any polynomial-time computable function on directed graphs.

The paper [11] studies a closely related yet different problem, called *minimum changeover cost arborescence* (MinCCA). The goal is to find a spanning tree that minimizes the sum of the reload costs due to traversing all internal vertices, where the sum is independent of the amount of commodity traversing the edges. This problem is the focus of our study. Fig. 1 depicts an example application scenario, i.e., energy-efficient broadcasting of control traffic in a multi-hop cognitive radio network, where each wireless link is associated with a frequency that is determined according to the spectrum availabilities. The source node aims to form a broadcasting tree to send its control traffic to all other nodes. Each control traffic flow at

each vertex causes an energy consumption, whose value depends on the frequencies of the incident links that are traversed. Energy consumption cost of switching between frequency f_i and f_j is $|j-i|$ unit(s) $\forall i, j$. The MinCCA problem corresponds to finding a broadcasting tree from the source node to all other nodes, such that the total energy consumption due to frequency switching is minimized. The bold edges indicate the optimum solution of the MinCCA problem in this instance, which has value 7, since the changeover cost incurred at nodes a, b, f is 1, and at nodes c, d is 2.

The authors in [11] consider directed graphs and show that even on graphs with bounded degree and reload costs adhering to the triangle inequality, MinCCA is hard to approximate within factor $\beta \log\log(n)$ for $\beta > 0$, when there are only 2 colors, and $n^{\frac{1}{3}-\epsilon}$, for any $\epsilon > 0$, when there are 3 colors. In [16], we derived several inapproximability results, as well as polynomial-time algorithms for some special cases of MinCCA. In particular, we showed that MinCCA is polynomial-time solvable for the special case of cactus graphs where the number of cycles each vertex can belong to is bounded by some constant. In this work, we extend this result to *any* cactus graphs, as well as to the fundamental class of graphs of bounded treewidth.

1.3. Our contribution

In this paper, we study the *minimum changeover cost arborescence* problem on *undirected* graphs. In solving MinCCA, our algorithms for cactus graphs and for graphs with bounded treewidth make non-trivial use of dynamic programming. To the best of our knowledge, the complexity of MinCCA in graphs of bounded treewidth is studied here for the first time. Moreover, except for the results in [16], we are not aware of any previous studies of MinCCA on special graph classes. Given the hardness results for general graphs, an important contribution of this paper is in establishing the polynomial solvability of MinCCA on a fundamental subclass of graphs. Our results imply that MinCCA is *fixed parameter tractable* when parameterized by treewidth and the maximum degree of the input graph.[3]

2. Preliminaries

Graphs, digraphs, trees, forests:
Given an undirected graph $G = (V(G), E(G))$ and a subset $U \subseteq V(G)$ of the vertices of G, $\delta_G(U) \stackrel{def}{=} \{uu' \in E(G) : u \in U, u' \notin U\}$ is the *cut* of G determined by U, i.e., the set of edges of G that have exactly one end in U. In particular, $\delta_G(v)$ denotes the set of edges incident to v in G, and $d_G(v) \stackrel{def}{=} |\delta_G(v)|$ is the *degree* of v in G. The *minimum and maximum degrees* of G are defined as $\delta(G) \stackrel{def}{=} \min\{d_G(v) | v \in V(G)\}$ and $\Delta(G) \stackrel{def}{=} \max\{d_G(v) | v \in V(G)\}$, respectively. We denote by $N_G(U)$ (resp., $N_G[U]$) the *open* (resp., *closed*) *neighborhood* of U in G. $N_G(U)$ is the set of vertices of $V(G) \setminus U$ that are adjacent to a vertex of U, and $N_G[U] \stackrel{def}{=} N_G(U) \cup U$. When there is no ambiguity about the graph G we omit it from the subscripts. For a subset of vertices $U \subseteq V(G)$, $G[U]$ denotes the subgraph of G *induced* by U. Given a tree T and two vertices $v_1, v_2 \in V(T)$, we denote by $P_T(v_1, v_2)$ the path between v_1 and v_2 in T.

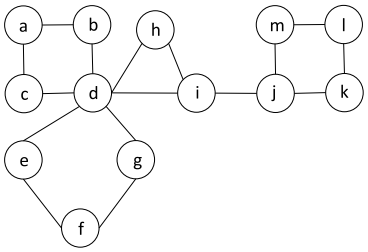
We use similar notations for digraphs. The sets of incoming arcs and outgoing arcs of a vertex v are denoted by $\delta_G^{(in)}(v)$ and $\delta_G^{(out)}(v)$, respectively. We denote by $d_G^{(in)}(v)$ and $d_G^{(out)}(v)$ the *in and out degrees* of v. The degree of a vertex in a digraph is its degree in the underlying graph, i.e., $d_G(v) = d_G^{(in)}(v) + d_G^{(out)}(v)$. A vertex v is a *source* (resp., *sink*) of a digraph G if $d_G^{(in)}(v) = 0$ (resp., $d_G^{(out)}(v) = 0$). We denote the set of sources and the set of sinks of a graph G by $SRC(G) \stackrel{def}{=} \left\{v \in V(G) | d_G^{(in)}(v) = 0\right\}$ and $SNK(G) \stackrel{def}{=} \left\{v \in V(G) | d_G^{(out)}(v) = 0\right\}$, respectively. $N_G^{(in)}(U)$ is the set of vertices of $V(G) \setminus U$ having an outgoing arc to a vertex of U, and $N_G^{(out)}(U)$, $N_G^{(in)}[U]$, $N_G^{(out)}[U]$ are defined similarly. The *inbound induced subgraph* $G[U, in]$ is the subgraph of G that consists of all arcs incoming to a vertex of U and all vertices that are endpoints of these arcs.

A digraph is a *rooted tree* or *arborescence* if its underlying graph is a tree and it contains a *root* vertex which is a source with a directed path to every other vertex. We denote by $root(T)$ the unique source vertex of the rooted tree T. Every other vertex $v \neq root(T)$ has in degree 1. In this case we denote by $in_T(v)$ the unique arc of $\delta_T^{(in)}(v)$. We term the other endpoint of $in_T(v)$ as the *parent* of v and denote it by $parent_T(v)$. We say that v is a *child* of $parent_T(v)$ in T.

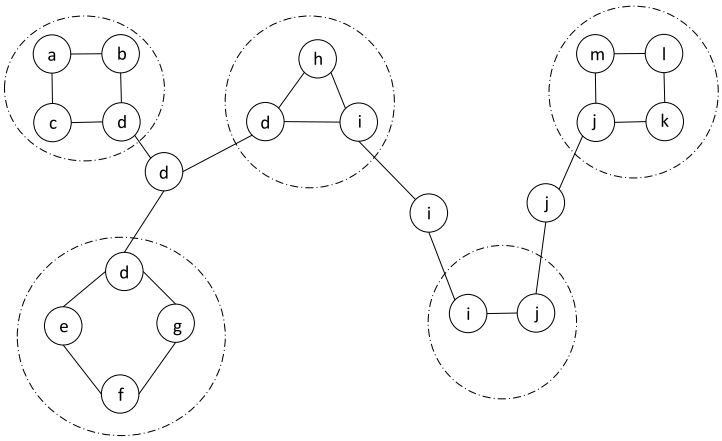
Given two graphs G and G', their union is $G \cup G' \stackrel{def}{=} (V(G) \cup V(G'), E(G) \cup E(G'))$. A *rooted forest* is the disjoint union of rooted trees. Clearly, the number of sources of a rooted forest is equal to the number of the trees. We use the tree notation also for forests, whenever appropriate. For a set A and an element x, we use $A + x$ (resp., $A - x$) as a shorthand for $A \cup \{x\}$ (resp., $A \setminus \{x\}$).

Biconnected components and block trees:
A *vertex separator* of graph is a set of vertices whose removal (along with its incident edges), increases the number of connected components of the graph. A *cut vertex* of a graph is a vertex separator consisting of a single vertex. Such a vertex

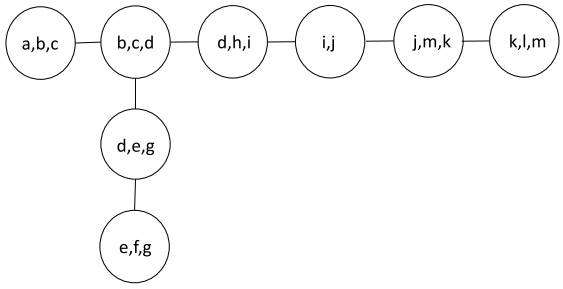
[3] We give the precise definition in Section 2.



(a) A cactus graph.



(b) Block tree of the cactus graph in Figure 2a.



(c) Small tree decomposition of the cactus graph in Figure 2a.

Fig. 2. A cactus graph, its block tree and its small tree decomposition.

is also termed an *articulation point* or *separation vertex* in the literature. A *bridge* in a connected graph is an edge whose removal disconnects the graph. A graph with no cut vertices is called *biconnected* or *nonseparable*. A maximal biconnected subgraph of a graph is called a *biconnected component* or a *block* [9]. Any connected graph G can be decomposed in linear time into a tree whose set of vertices consists of: a) the biconnected components of G, and b) its cut vertices. This tree is termed the *block tree* or *superstructure* of G. The set of edges of the block tree consists of edges connecting every cut vertex to the blocks it belongs to [9,18].

Tree decompositions and treewidth:
A *tree decomposition* of a graph G is a tree $\mathcal{T}$, where $V(\mathcal{T}) = \{B_1, B_2, \ldots\}$ is a set of subsets (called *bags*) of $V(G)$, such that the following three conditions are met:

1. $\bigcup V(\mathcal{T}) = V(G)$.
2. For every edge $uv \in E(G)$, $u, v \in B_i$ for some bag $B_i \in V(\mathcal{T})$.
3. For every $B_i, B_j, B_k \in V(\mathcal{T})$ such that B_k is on the path $P_{\mathcal{T}}(B_i, B_j)$, $B_i \cap B_j \subseteq B_k$.

The *width* $\omega(\mathcal{T})$ of a tree decomposition $\mathcal{T}$ is defined as the size of its largest bag minus 1, i.e., $\omega(\mathcal{T}) = \max\{|B| \mid B \in V(\mathcal{T})\} - 1$. The *treewidth* of a graph G, denoted as $tw(G)$, is defined as the minimum width among all tree decompositions of G. When the treewidth of the input graph is bounded, many efficient algorithms are known for problems that are in general NP-hard. For instance, cactus graphs have treewidth 2. Fig. 2 illustrates a cactus graph, its block tree and its tree decomposition with width 2.

A tree decomposition $\mathcal{T}$ of G is *small* if no proper subtree of $\mathcal{T}$ is a tree decomposition of G. In this work, we consider only small tree decompositions. Clearly, this does not affect the definition of treewidth, i.e., there is a small tree decomposition $\mathcal{T}$ of G such that $\omega(\mathcal{T}) = tw(G)$. This statement is true because if a tree decomposition $\mathcal{T}$ is not small, we can replace it with a proper subtree of it until such a subtree does not exist, i.e., the tree decomposition we obtain is small. Since every bag of the resulting tree $\mathcal{T}'$ is a bag of $\mathcal{T}$, we have $\omega(\mathcal{T}') \leq \omega(\mathcal{T})$. For a subtree $\mathcal{T}'$ of a tree decomposition $\mathcal{T}$ of a graph G, we denote by $\mathcal{T}'^*$ the set of vertices of G that are in some bag of $\mathcal{T}'$, i.e., $\mathcal{T}'^* \stackrel{def}{=} \cup V(\mathcal{T}')$. The following are well known properties of tree decompositions, which are included here for the sake of completeness.

Proposition 1. *Let $\mathcal{T}$ be a tree decomposition of a graph G and B be a bag of $\mathcal{T}$. Also, let $\mathcal{T}_1, \mathcal{T}_2, \ldots$ be the subtrees obtained by the removal of B from $\mathcal{T}$. Then*

(i) *B is a vertex separator of G and it divides the graph into at least $d_{\mathcal{T}}(B)$ components $G_1, G_2, \ldots$ where $G_i = G[\mathcal{T}_i^* \setminus B]$.*
(ii) *Every vertex at distance 1 to a subgraph G_i is in $B \cap B_i$ where B_i is the unique neighbor of B in $\mathcal{T}_i$.*

Proof. (i) See, e.g., [19] p. 575. (ii) Let u be a vertex at distance 1 to some subgraph G_i. Then clearly $u \in B$ because B is a separator. Let v be a neighbor of u in G_i. The edge uv is in some bag B' in $\mathcal{T}_i$. Then u is in B' and also in all bags between B and B'. In particular, u is in the neighbor of B on this path, i.e. in B_i. Therefore, $u \in B \cap B_i$. □

Reload and changeover costs:
We follow the notation and terminology of [21] where the concept of reload cost was defined. We consider edge colored graphs G, where the colors are taken from a finite set X and $\chi : E(G) \to X$ is the *coloring function*. Given a coloring function χ, we denote by E_x^χ, or simply by E_x the set of edges of E colored x. $G_x = (V(G), E(G)_x)$ is the subgraph of G having the same vertex set as G, but only the edges colored x.

The costs are given by a non-negative function $c : X^2 \to \mathbb{N}_0$ satisfying

i) $c(x_1, x_2) = c(x_2, x_1)$ for every $x_1, x_2 \in X$.
ii) $c(x, x) = 0$ for every $x \in X$.

At this point we note that our algorithms do not rely on any one of the above properties of the cost function and they will work on other cost functions, too. We assume without loss of generality that the minimum non-zero cost value is 1 and we denote the maximum cost value as $C_{max} \stackrel{def}{=} \max_{x_1, x_2 \in X} c(x_1, x_2)$. The cost of traversing two incident edges e_1, e_2 is $c(e_1, e_2) \stackrel{def}{=} c(\chi(e_1), \chi(e_2))$.

We say that an instance satisfies the *triangle inequality*, if (in addition to the above) the cost function satisfies

iii) $c(e_1, e_3) \leq c(e_1, e_2) + c(e_2, e_3)$ whenever e_1, e_2 and e_3 are incident to the same vertex.

The changeover cost of a path $P = (e_1 - e_2 - \ldots - e_\ell)$ of length ℓ is $c(P) \stackrel{def}{=} \sum_{i=2}^{\ell} c(e_{i-1}, e_i)$. Note that $c(P) = 0$ whenever $\ell \leq 1$.

We extend this definition to trees as follows: Given a directed tree T rooted at r, (resp., an undirected tree T and a vertex $r \in V(T)$), for every outgoing edge e of r (resp., incident to r) we define $prev(e) = e$. For every other edge, $prev(e)$ is the edge preceding e on the path from r to e. The changeover cost of T with respect to r is $c(T, r) \stackrel{def}{=} \sum_{e \in E(T)} c(prev(e), e)$. When there is no ambiguity about the vertex r, we denote $c(T, r)$ by $c(T)$.

Problem statement:
The MinCCA problem aims to find a spanning tree rooted at r with minimum changeover cost [11]. Formally,

MinCCA (G, X, χ, r, c)
Input: A graph $G = (V, E)$ with an edge coloring function $\chi : E \to X$, a vertex $r \in V$ and a reload cost function $c : X^2 \to \mathbb{N}_0$.
Output: A spanning tree T of G.
Objective: Minimize $c(T, r)$.

Parameterized complexity:
In parameterized complexity theory, the complexity of an algorithm is expressed as a function of both the size n of the input and a parameter k depending on the input. A problem is called *fixed parameter tractable* (FPT) if it can be solved in time $f(k) \cdot p(n)$, where f is a function depending solely on k and p is a polynomial in n. A problem is in XP if it can be solved in time $n^{f(k)}$. We show that MinCCA when parameterized by $tw(G)$ is in XP, and that it is FPT when parameterized by $tw(G) + \Delta(G)$.

Algorithm 1 FindMinCCACactus(G, X, χ, r, c).

Initialization:
1: $V(G) \leftarrow V(G) + r'$.
2: $E(G) \leftarrow E(G) + rr'$.
3: $B_r \leftarrow G[\{r, r'\}]$.
4: $\mathcal{T} \leftarrow$ the block tree of G rooted at B_r.

Dynamic Programming:
5: **for all** vertices U in a postorder traversal of $\mathcal{T}$ except B_r **do**
6: **if** U is a cut vertex s of G **then**
7: Let $B_1, \ldots, B_k$ be the children of s in $\mathcal{T}$.
8: **for all** $e \in \delta_{B(s)}(s)$ **do**
9: $OPT(s, e) = \sum_{i=1}^{k} OPT(B_i, e)$.
10: **else** ▷ U is a block B of G
11: **for all** $e' \in \delta_{B(s(B))}(s(B))$ **do**
12: Let $s_1, \ldots, s_k$ be the cut vertices of G in B except $s(B)$.
13: $OPT(B, e') = \min_{T' \text{ is a spanning tree of } B} \left(c(T' + e') + \sum_{i=1}^{k} OPT(s_i, in_{T'}(s_i))\right)$.

Find Optimum:
14: **return** $OPT(r, rr')$.

3. Bounded treewidth graphs

In this section, we present a polynomial-time algorithm for MinCCA for any graph whose treewidth is bounded by some constant. As a warm up, in Section 3.1 we start with an optimal polynomial-time algorithm for cactus graphs, for which the treewidth is at most 2 [7] (see Fig. 2). In Section 3.2, we generalize this algorithm to obtain an optimal algorithm for graphs of bounded treewidth. Our algorithms make non-trivial use of dynamic programming.

3.1. An algorithm for cactus graphs

In this section, we present a polynomial-time algorithm for cactus graphs. A *cactus* graph is a graph in which any pair of simple cycles have at most one vertex in common; in other words, every edge in a cactus graph belongs to at most one simple cycle. An alternative definition of a cactus graph states that a graph is a cactus if every block of it is either an edge or a cycle [17]. We will start by observing that a solution of MinCCA has a very simple structure when the input graph is a cactus. Exploiting this simple structure, we devise a dynamic programming algorithm that traverses the block tree of the graph in a bottom-up manner, and computes at each block a small set of (in fact one or two) optimal solutions. In the following paragraph, we describe the structure of the solution.

Let (G, X, χ, r, c) be an instance of MinCCA, and let $\mathcal{T}$ be the block tree of G. Consider a spanning tree T of G rooted at r and a block B of $\mathcal{T}$. Let also B_r be an arbitrary block containing r. It is easy to see that $T[B]$ is a spanning tree of B. The root r_B of this tree is the cut vertex that is adjacent to B on the path from B_r to B in $\mathcal{T}$. When G is a cactus graph, B is either an edge or a cycle. When B is an edge, there is only one possible tree spanning B, i.e., the tree consisting of the unique edge of B where the direction is from r_B outwards. When B is a cycle there are $|V(B)|$ possible spanning trees, one per every edge of B whose removal from B leads to a spanning tree of B. Therefore, in order to compute a spanning tree, and in particular a minimum changeover cost tree T of a cactus graph G, one has to choose one edge per each cycle C of G to be absent in T.

The main difficulty in finding an optimal solution is that the decisions about which edges of the cycle should be in the optimal solution are not independent of each other. We now observe that the decision in one cycle affects the decision in another cycle only when the two cycles intersect (in a cut vertex). Let s be a cut vertex separating two cycles C_1, C_2 where C_1 is closer to r than C_2. Let e_{11} and e_{12} be the edges incident to s in C_1. Clearly, $in_T(s) \in \{e_{11}, e_{12}\}$, and the decision on C_2 is affected only by the value of $in_T(s)$. Once this edge is given, we can make a correct decision on C_2. This observation suggests a dynamic programming algorithm that will compute an optimal solution based on two different assumptions, i.e. $in_T(s) = e_{11}$ and $in_T(s) = e_{12}$.

The pseudocode of this algorithm (FindMinCCACactus) is given in Algorithm 1. For ease of exposition, we add a vertex r' to G together with the edge rr'. Then i) r is a cut vertex, and ii) B_r is the block consisting of the edge rr'. We direct $\mathcal{T}$ so that B_r is its root. This is done in lines 1–4.

We now proceed with the details of bottom-up traversal of the block tree $\mathcal{T}$ that is done by the loop in line 5. A vertex of $\mathcal{T}$ is either a cut-vertex of G or a block of G. For a cut vertex s, we denote by $B(s)$ the block $parent_{\mathcal{T}}(s)$ and for a block B we denote by $s(B)$ the cut vertex $parent_{\mathcal{T}}(B)$. Consider a cut vertex s and an edge e in $B(s)$ incident to s. We denote by $OPT(s, e)$ the cost of an optimal tree that starts with the edge e and spans all the blocks in the subtree of s. Similarly, for a block B and an edge e incident to $s(B)$ in $B(s(B))$, we denote by $OPT(B, e)$ the cost of an optimal tree that starts with the edge e and spans all the blocks in the subtree of B.

When the algorithm encounters a cut vertex s with children $B_1, \ldots, B_k$ in $\mathcal{T}$ during its bottom-up traversal, it uses the following simple fact that follows from the definitions (lines 7–9).

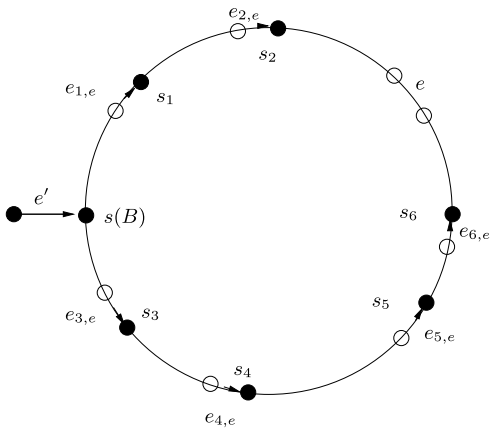


Fig. 3. A cycle block of a cactus graph together with an edge e' of a potential spanning tree entering this block and an edge e of the cycle that does not belong to the spanning tree.

$$OPT(s, e) = \sum_{i=1}^{k} OPT(B_i, e).$$

In the rest of the discussion, we describe the behavior of Algorithm 1 when it encounters a block B during its traversal. Recall that our goal is to compute the best tree spanning all vertices in B and in all blocks traversed so far, assuming that this spanning tree is connected to the solution by a given edge e'. We consider two disjoint cases that are also complementary by the definition of a cactus graph.

- B consists of an edge e: In this case one endpoint of e is $s(B)$. Let s_1 be the other endpoint of e (which is also a cut vertex). Moreover, s_1 is an endpoint of e'. Then

  $$OPT(B, e') = c(e', e) + OPT(s_1, e).$$

- B is a cycle C: Consult Fig. 3 for the discussion of this case. Let $s_1, \ldots, s_k$ be the cut vertices of G in C different from $s(B)$. For an arbitrary edge $e \in E(C)$, consider the path $C - e$ and the subpath of $C - e$ between $s(B)$ and s_i. Let $e_{i,e}$ be the edge incident to s_i on this path. Then, assuming that there is an optimal spanning tree T with $T[B] = C - e$, we have

  $$OPT(B, e') = c(C - e + e') + \sum_{i=1}^{k} OPT(s_i, e_{i,e}).$$

  Since $T[B] = C - e$ for at least one edge e of C, we conclude that

  $$OPT(B, e') = \min_{e \in E(C)} \left(c(C - e + e') + \sum_{i=1}^{k} OPT(s_i, e_{i,e}) \right).$$

In fact, the above two cases can be combined into one case by observing that in both cases we actually consider all spanning trees of B with the addition of the edge e'. Lines 11–13 of Algorithm 1 use this observation to treat both cases in a unified way. This unification leads to the main idea of the algorithm for bounded treewidth graphs presented in Section 3.2. As for the running time, we note that the number of vertices of the block tree is linear in the input size, and at each such vertex the number of operations is at most quadratic in the size of the block or linear in the degree of the cut vertex. Therefore, the running time of the algorithm is polynomial in the input size.

Theorem 1. MinCCA *is solvable in polynomial time for graphs where the number of spanning trees in every block is polynomial, and in particular for cactus graphs.*

3.2. An algorithm for bounded treewidth graphs

In order to develop a dynamic programming algorithm similar to that given in Section 3.1, we need to show how a solution (i.e., a spanning tree) can be obtained from partial solutions. Clearly, these partial solutions are forests. In order to proceed in a recursive manner, we have to analyze the decomposition of a forest into sub-forests as implied by a tree decomposition. We start by introducing notations that we use in this section. We proceed by showing some simple facts

Table 1
Frequently used symbols and notations.

Symbol	Meaning
G	The graph of the input instance.
r	The root vertex of G.
X	The set of available colors.
χ	The edge-coloring of G.
$\mathcal{T}$	A small tree decomposition of G.
B_r	The bag of $\mathcal{T}$ containing r.
$SRC(F)$	The set of sources of the forest F.
$\mathcal{T}_B$	The descendants of the bag B in $\mathcal{T}$ (including itself).
B^*	The set of vertices of all bags in $\mathcal{T}_B$.
$\mathcal{F}(B)$	The set of all forests that span B and some neighbors of B such that these neighbors are their sources.
$\mathcal{F}(B^*)$	The set of all forests that span B^* and some neighbors of B^* such that these neighbors are their sources.
$\mathcal{F}(F)$	The set of all forests whose vertices are the sources of the forest F.
$F\|_U$	The partial order obtained by restricting the partial order F to the set U.
$F' \sim (F, f)$	(F, F', f are forests and $f \in \mathcal{F}(F)$)
	$F' \cup F$ is a forest and the partial order that this forest induces on the sources of F is a subgraph of f.
$c_U(F)$	The partial cost of the forest F that takes into account only the cost of edges joining vertices of U.
$OPT(B, F, f)$	($F \in \mathcal{F}(B^*)$, $f \in \mathcal{F}(F)$)
	The minimum cost of a forest in $\mathcal{F}(B^*)$ that extends F and induces the partial order f on its sources.

related to these definitions and give an overview of the proof as well as the algorithm. Finally, we describe the algorithm and its proof.

More definitions and notations: Let G be the input graph and let $\mathcal{T}$ be a small tree decomposition of G with the smallest width. For simplicity, we direct $\mathcal{T}$ so that it is rooted at some bag B that contains r. All trees and forests considered in this section are rooted. The ancestor-descendant relation between the vertices of a rooted forest is clearly a partial order. We refer to a forest also as the partial order it induces. Two forests on the same set of vertices are equal if and only if the partial orders they induce are equal. Given a forest F and $U \subseteq V(F)$, $F|_U$ is a forest with vertex set U and uu' is an arc of $F|_U$ if and only if u is the first vertex in U on the path from u' to the root of its tree in F. In other words, the partial order $F|_U$ is the partial order F restricted to U.

For a nonempty and proper subset U of vertices of G, $\mathcal{F}(U)$ is the set of all non-trivial forests F with the following properties: a) sources of F are absent in U but are neighbors (in F) of some vertex in U; b) F spans all vertices in U. Formally,

$$\mathcal{F}(U) \stackrel{def}{=} \{F \text{ is a forest of } G : U \subseteq V(F) \subseteq N_G[U], SRC(F) = V(F) \setminus U, \delta(F) > 0\}.$$

For a bag $B \in V(\mathcal{T})$ we denote by $\mathcal{T}_B$ the subtree of $\mathcal{T}$ consisting of B and its descendants. The set $B^* \stackrel{def}{=} \cup \mathcal{T}_B$ consists of all the vertices in the bags of $\mathcal{T}_B$. Clearly, $B \subseteq B^*$. Given a bag B of $\mathcal{T}$, and a forest $F \in \mathcal{F}(B)$ we define $\mathcal{F}(F)$ as follows:

$$\mathcal{F}(F) \stackrel{def}{=} \left\{f \text{ is a forest of } G : V(f) = SRC(F), SRC(f) = SRC(F) \setminus B^*\right\}.$$

Note that F is not an empty forest since it spans B; thus, it has at least one source, implying that $V(f) \neq \emptyset$, which in turn implies that $SRC(f) \neq \emptyset$.

Let $F \in \mathcal{F}(B)$, and $f \in \mathcal{F}(F)$. We say that a forest F' is compatible with F and f, and denote as $F' \sim (F, f)$ if $F \cup F'$ is a forest and $(F \cup F')\big|_{V(f)}$ is a subgraph of f.

Table 1 summarizes the notations used in this section.

Proposition 2.

$$SRC(F) \cap U \subseteq SRC(F|_U).$$

Moreover, if $SRC(F) \subseteq U$ *then*

$$SRC(F) = SRC(F|_U).$$

Proof. To see the first inequality, let $u \in SRC(F) \cap U$. As u is minimal in the partial order F it is also minimal in the partial order $F|_U$. Therefore, $SRC(F) \cap U \subseteq SRC(F|_U)$. If $SRC(F) \subseteq U$, every source of F is an element of U. Clearly, these are exactly the minimal elements of $F|_U$. □

We note that every forest of $\mathcal{F}(U)$ can be obtained by choosing a parent for every $u \in U$ from its neighbors. Therefore,

$$|\mathcal{F}(U)| \leq \prod_{u \in U} d_G(u) \leq \Delta(G)^{|U|}.$$

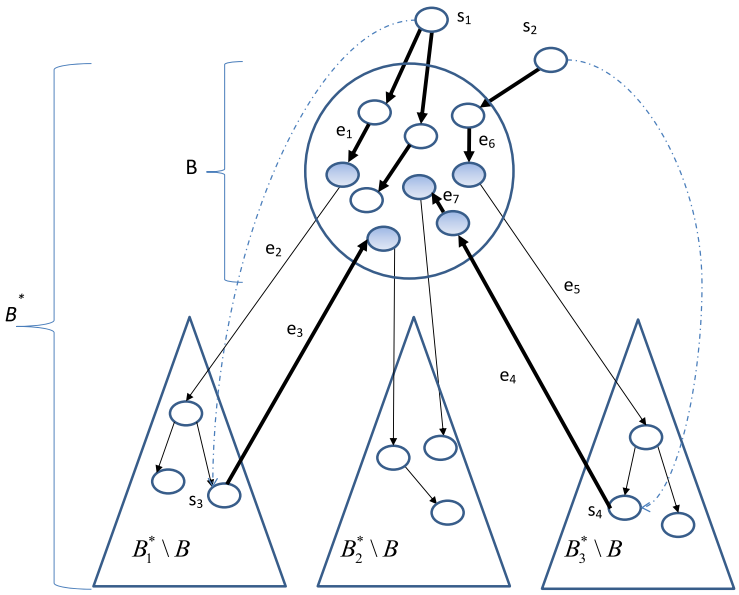


Fig. 4. An example for the decomposition of a forest $F^* \in \mathcal{F}(B^*)$. All vertices in the figure together with the solid arcs constitute the forest F^*. s_1 and s_2 are the sources of F^*. All vertices except these two are in B^* and they are partitioned into four sets, namely B, $B_1^* \setminus B$, $B_2^* \setminus B$, and $B_3^* \setminus B$. The dark vertices in B are those that are common with at least one of the bags B_1, B_2, B_3. We note that every vertex at distance 1 from $B^* \setminus B$ is dark. The forest F consists of all bold arcs, i.e. those that are entirely in B or enter B. F has four sources, namely s_1, s_2, s_3 and s_4. F_1^* consists of all arcs that are entirely in B_1^* or enter it. In the figure, those are all arcs within the leftmost triangle, together with e_1, e_2 and e_3. Note that e_3 is in F, F_1^* and F_2^*, while e_4 is also in a similar situation. Consider the sources of F, namely s_1, s_2, s_3 and s_4 as vertices of F^*: s_1 is the closest ancestor of s_3 and s_2 is the closest ancestor of s_4. This relation is drawn by the two dashed edges that constitute the forest f. Note that e_7 connects two dark vertices of B and it contributes (the same) cost to F and F_3^*. Lemma 2 takes such redundancies into account when computing the cost of F^*.

We now observe that every forest of $\mathcal{F}(F)$ can be obtained by choosing a parent for every $v \in SRC(F) \cap B^*$, from $SRC(F)$. Therefore,

$$|\mathcal{F}(F)| \leq (|SRC(F)| - 1)^{|SRC(F)|}.$$

The decomposition of a solution: In this section, our goal is to develop a dynamic programming algorithm that traverses the tree decomposition $\mathcal{T}$ in a bottom-up manner, and computes at every bag a set of optimal solutions satisfying certain criteria. We start with Lemma 1 that shows how a forest of $\mathcal{F}(B^*)$ is decomposed into sub forests, namely into a forest of $\mathcal{F}(B)$ and forests of $\mathcal{F}(B_i^*)$ for each child B_i of B. This lemma will also enable us to partition all forests of $\mathcal{F}(B^*)$ into equivalence classes. See Fig. 4 for an example. In Lemma 2, we develop a formula to decompose the cost of these forests in terms of the costs of its sub-forests. We use these results to calculate the cost of the best forest in each equivalence class. This is done in Lemma 3, which provides a recurrence formula for the costs of these best forests. The dynamic programming algorithm computes the best forest in each equivalence class via this recurrence formula.

Lemma 1. *Let B be a bag of $\mathcal{T}$ with children $B_1, \ldots, B_k$ in $\mathcal{T}$. $F^* \in \mathcal{F}(B^*)$ if and only if there exist unique forests $F, f, F_1^*, \ldots, F_k^*$ such that*

i) $F^* = F \cup \left(\cup_{i \in [k]} F_i^*\right)$
ii) $F \in \mathcal{F}(B)$
iii) $\forall i \in [k]$, $F_i^* \in \mathcal{F}(B_i^*)$
iv) $f \in \mathcal{F}(F)$
v) $\forall i \in [k]$, $F_i^* \sim (F, f)$.

Proof. ($\Rightarrow$) Let $F^* \in \mathcal{F}(B^*)$. We will show that the forests $F = F^*[B, in]$, $f = F^*|_{SRC(F)}$ and $F_i^* = F^*[B_i^*, in]$ for every $i \in [k]$ are the unique forests that satisfy conditions *i)* through *v)*.

i) As every arc of F^* is directed towards a vertex of B^* and $B^* = B \cup \left(\cup_i B_i^*\right)$, we conclude that $F^* = F \cup \left(\cup_{i \in [k]} F_i^*\right)$.

ii) $V(F) \subseteq N_G[B]$ by definition of F. To see that $B \subseteq V(F)$, let $v \in B \subseteq B^*$. By the definition of $\mathcal{F}(B^*)$, this implies that $v \in V(F^*) \setminus SRC(F^*)$. Let $e = in_{F^*}(v)$. Then, by the way F is built from F^*, $e \in E(F)$, implying that $v \in V(F)$. Therefore, $B \subseteq V(F) \subseteq N_G[B]$.

Let $v \in V(F) \setminus B$. By definition of F, $v \in SRC(F)$, i.e., $d_F^{(in)}(v) = 0$. Conversely, let $v \in SRC(F)$ and assume that $v \in B$. Then $v \in SRC(F^*)$, contradicting the definition of F^* because F^* cannot have sources in B^*. Therefore, $SRC(F) = V(F) \setminus B$.

Finally, F does not contain isolated vertices as implied by the definition of an inbound induced subgraph. We conclude that $F \in \mathcal{F}(B)$.

iii) It can be shown that $F_i^* \in \mathcal{F}(B_i^*)$ for every $i \in [k]$ by replacing B with B_i^* and F with F_i^* in the preceding arguments.

iv) Clearly, $V(f) = SRC(F)$ by the definition of f. It remains to show that $SRC(f) = SRC(F) \setminus B^*$. Consider $s \in SRC(F^*)$ and an arc $e = (s, v) \in E(F^*)$. By definition of F^*, we have $s \in N_G[B^*] \setminus B^*$ and $v \in B^*$. Then, Proposition 1 implies that $v \in B$. Therefore, by definition of F we have $e = (s, v) \in E(F)$. As F is a subgraph of F^*, we conclude that $s \in SRC(F)$. Combining with $s \notin B^*$, we get

$$SRC(F^*) \subseteq SRC(F) \setminus B^* \subseteq SRC(F). \tag{1}$$

Together with Proposition 2, this implies

$$SRC(f) = SRC(F^*). \tag{2}$$

Now let $s \in SRC(F) \setminus B^*$. The definition of F^* implies that $r \in SRC(F^*)$. Therefore, $SRC(F) \setminus B^* \subseteq SRC(F^*)$. Recalling (1) and (2) we get

$$SRC(f) = SRC(F^*) = SRC(F) \setminus B^*$$

as required.

v) Let $i \in [k]$ and let $F' = F \cup F_i^*$. F' is a subgraph of F^* and is therefore a forest. Moreover, $F'|_{SRC(F)}$ is a subgraph of $F^*|_{SRC(F)} = f$.

To show the uniqueness of F, assume $F' \in \mathcal{F}(B)$ satisfies the conditions stated for F. We have $V(F) \setminus SRC(F) = B = V(F') \setminus SRC(F')$. Therefore, v has an inbound arc $e \in E(F) \subseteq E(F^*)$ if and only if it has an inbound arc $e' \in E(F') \subseteq E(F^*)$. In this case, $e = e'$ since otherwise $d_{F^*}^{(in)}(v) \geq 2$. We conclude that $E(F) = E(F')$. Since none of F and F' contain isolated vertices (by definition of $\mathcal{F}(B)$), we have $F = F'$. The uniqueness of each one of the forests $F_1^*, \ldots, F_k^*$ is shown using the same argument.

We now show the uniqueness of f. Let $f' \in \mathcal{F}(F)$ be a forest satisfying the conditions stated for f. As $f, f' \in \mathcal{F}(F)$ we have $V(f') = SRC(F) = V(f)$ and $SRC(f') = SRC(F) \setminus B^* = SRC(f)$ implying $|E(F)| = |E(F')|$. Therefore, it is sufficient to show that $E(f) \subseteq E(f')$. Let $e \in E(f) = E(F^*|_{SRC(F)})$ with $e = (u, v)$. Then, u is the closest ancestor of v in F^* that is in $V(f)$. The path P from u to v starts with arcs from F and continues with arcs from $F_i^* \setminus F$ for some $i \in [k]$. Indeed, if P contains arcs from distinct forests F_i^*, F_j^*, it will constitute a path between B_i^* and B_j^* in $G \setminus B$, contradicting Proposition 1. Therefore, u is the closest ancestor from $V(f) (= V(f') = SRC(F))$ of v in $F \cup F_i^*$, i.e., e is an arc of $(F \cup F_i^*)|_{V(f')}$. As $F_i^* \sim (F, f')$ this implies that $e \in E(f')$.

($\Leftarrow$) Let $F, f, F_1^*, \ldots, F_k^*, F^*$ be forests satisfying conditions *i)* to *v)*. We have to show that $F^* \in \mathcal{F}(B^*)$.

We start by showing that F^* is a forest. Suppose that there is a vertex $v \in V(F^*)$ such that $d_{F^*}^{(in)}(v) \geq 2$. Let e_1, e_2 be two distinct arcs of $\delta_{F^*}^{(in)}(v)$. Clearly, these two arcs are part of two different forests from $F, F_1^*, \ldots, F_k^*$. Moreover, none of these arcs is an arc of F because $F_i^* \sim (F, f)$ for every $i \in [k]$. Let without loss of generality, e_1 (resp., e_2) be an arc of F_1^* (resp., F_2^*). Then, for $i \in \{1, 2\}$, $v \notin SRC(F_i^*)$ implying $v \in B_i^*$. Therefore, $v \in B_1^* \cap B_2^* \subseteq B$. Then, by definition of $\mathcal{F}(B)$, $v \in V(F) \setminus SRC(F)$, i.e., there is an arc $e \in V(F)$ entering v. As $e_1 \neq e_2$, without loss of generality $e \neq e_1$, contradicting the fact that the union of F and F_1 is a forest. Therefore, $d_{F^*}^{(in)}(v) \leq 1$ for every vertex v of F^*. To conclude that F^* is a forest, it remains to show that F^* does not contain a cycle.

Assume by contradiction that F^* contains a cycle C. Then C is a directed cycle because otherwise there is a vertex $v \in V(C)$ with $d_{F^*}^{(in)}(v) \geq d_C^{(in)}(v) = 2$. Clearly, $E(C) \setminus E(F) \neq \emptyset$, because otherwise F contains a directed cycle. $E(C) \setminus E(F)$ is the union of the sets $E(C) \cap \left(E(F_i^*) \setminus E(F)\right)$. Moreover, these sets are disjoint since $E(F_i^*) \cap E(F_j^*) \subseteq E(F)$. Furthermore, at least two of these sets are nonempty because otherwise C is contained in $F \cup F_i^*$ for some $i \in [k]$ constituting a contradiction to the fact that $F_i^* \sim (F, f)$. Consider two maximal non-trivial directed paths P, P' of C such that $E(P) \subseteq E(F_i^*) \setminus E(F)$ and $E(P') \subseteq E(F_i^*) \setminus E(F)$. Moreover, let P and P' be two such paths that are consecutive in C in the order P, P'. Let u, v (resp., $u'v'$) be the start and end vertices of P (resp., P'). These vertices appear in the order u, v, u', v' in C. Note that the following argument is valid also for the case $u' = v$ in which there is no path of F separating the paths P and P'. Clearly, $v \neq v'$ as P' is non-trivial. Moreover, $v, v' \in SRC(F)$ because their unique inbound arcs are not in F. The vertex v is an ancestor of v' in $F \cup F_i^*$. As $F_i^* \sim (F, f)$, this implies that (v, v) is an arc of f. Repeating the same argument for the end vertices of all such maximal directed paths we conclude that f contains a directed cycle, contradicting the fact that f is a forest.

We proceed with the proof that the forest $F^* \in \mathcal{F}(F^*)$. We have $V(F^*) = V(F) \cup \left(\cup_{i \in [k]} V(F_i^*)\right)$, $B \subseteq V(F) \subseteq N_G[B]$, $B_i^* \subseteq V(F_i^*) \subseteq N_G[B_i^*]$, and $B^* = B \cup \left(\cup_{i \in [k]} B_i^*\right)$. We conclude that $B^* \subseteq V(F^*) \subseteq N_G[B^*]$ as required.

Let $v \in B^* = B \cup \left(\cup_{i \in [k]} B_i^*\right)$. If $v \in B$, then it is not a source of F, and if $v \in B_i^*$ then it is not a source of F_i^*. Therefore, it is not a source of F^*. We conclude that $SRC(F^*) \subseteq V(F^*) \setminus B^*$. Now let $v \in V(F^*) \setminus B^*$. Then v is in at least one forest among the forests $F, F_1^*, \ldots, F_k^*$. If $v \in V(F)$ then as $v \notin B$, we conclude that $v \in SRC(F)$. If $v \in V(F_i^*)$ for some $i \in [k]$ then as $v \notin B_i^*$, we conclude that $v \in SRC(F_i^*)$. To summarize, the inbound degree of v is zero in all forests it participates. Therefore, $v \in SRC(F^*)$ concluding that $V(F^*) \setminus B^* \subseteq SRC(F^*)$. Combining with $SRC(F^*) \subseteq V(F^*) \setminus B^*$ we get $SRC(F^*) = V(F^*) \setminus B^*$, as required.

Finally, $\delta(F^*) > 0$ because $\delta(F) > 0$ and $\delta(F_i^*) > 0$ for every $i \in [k]$. □

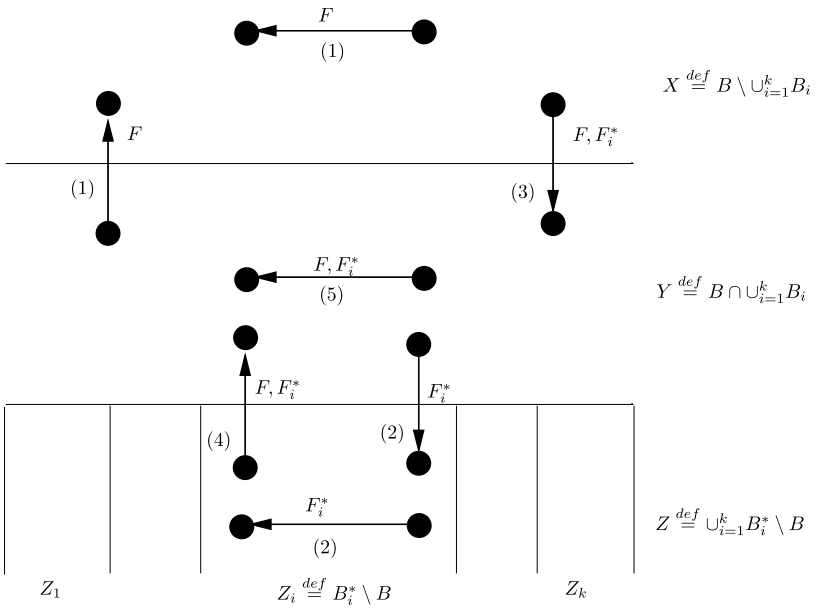


Fig. 5. The partition of B^* used in the proof and the categorization of the arcs $e \in E(F^*) \cap B^* \times B^*$ according to this partition. The names of the forests next to the arcs are the forests that contain the arc. The numbers in parentheses refer to the case number in the proof.

We define the partial cost of a forest F on a subset U of its vertices as

$$c_U(F) \stackrel{def}{=} \sum_{e \in E(F) \cap U \times U} c(prev(e), e)$$

and provide in the following lemma a formula for the cost of a forest in terms of its sub-forests.

Lemma 2. *Let B be a bag of $\mathcal{T}$ with children $B_1, \ldots, B_k$, and $F^* \in \mathcal{F}(B^*)$. Let $F, F_1^*, \ldots, F_k^*$ be forests whose existences are guaranteed by Lemma 1. Then*

$$c(F^*) = c(F) + \sum_{i=1}^{k} \left(c(F_i^*) - c_Y(F_i^*) \right)$$

where $Y = B \cap (\cup_{i \in [k]} B_i)$.

Proof. We will show that every arc of F^* contributes the same value to both sides of the equation. Let $e \in E(F^*)$ and $e = (u, v)$. We first note that for any sub-forest F' of F^* that contains the arc e, either u is a source of F' in which case the contribution of e to the cost of F' is zero, or $in_{F'}(e) = in_{F^*}(e)$, in which case the contributions of e to the costs of F' and F^* are equal.

We note that $v \in B^*$ because v is not a source of F^* and $F^* \in \mathcal{F}(B^*)$. If $u \notin B^*$, then u is a source of F^*. Therefore, for every sub-forest F' of F^*, either e is not part of F' or the cost of e in F' is zero, i.e., e contributes zero to both sides of the equality. In the rest of the proof we assume that $u, v \in B^*$. For this purpose, we partition B^* into the following $k+2$ sets of vertices: $X \stackrel{def}{=} B \setminus \cup_{i \in [k]} B_i$, $Y \stackrel{def}{=} B \cap \left(\cup_{i \in [k]} B_i \right)$, $Z_1 \stackrel{def}{=} B_1^* \setminus B, \ldots, Z_k \stackrel{def}{=} B_k^* \setminus B$. Let also $Z \stackrel{def}{=} \cup_{i \in [k]} Z_i$. It follows from the definitions that X, Y and Z are pairwise disjoint. Moreover, $B^* = X \cup Y \cup Z$ because $B^* = B \cup (\cup_{i \in [k]} B_i^*)$. We also note that since B is a vertex separator of G (Proposition 1), $Z_i \cap Z_j = \emptyset$ for any $i \neq j$. Therefore, $\{X, Y, Z_1, \ldots, Z_k\}$ is a partition of B^*. For the same reason, $e \notin X \times Z$ and $e \notin Z_i \times Z_j$ whenever $i \neq j$. Fig. 5 categorizes all the arcs $e \in E(F^*) \cap (B^* \times B^*)$ according to their endpoints.

We consider all possible cases of the endpoints of the arc $e = (u, v)$ of F^* that connects two vertices of $B^* = X \cup Y \cup Z$:

1. $v \in X$: In this case for every i, $e \notin E(F_i^*)$ and $e \in E(F)$. Moreover, as $v \in B$, v is a source of neither F nor F^*. Therefore, e contributes the same cost to both F and F^*.
2. $v \in Z$: In this case $e \in E(F_i^*)$ for exactly one $i \in [k]$, and $e \notin E(F)$. Moreover, as $v \in B_i^*$, v is not a source of F_i^*. Therefore, e contributes the same cost to both F_i^* and F^*. Finally, as $e \notin Y \times Y$, it does not contribute to $c_Y(F_i^*)$.
3. $v \in Y, u \in X$: In this case u is a source of F_i^* for every i. Therefore, e contributes zero to the cost of every F_i^*. Also, as in the previous case, $e \notin Y \times Y$, and it does not contribute to $c_Y(F_i^*)$. As $v \in B$, e contributes the same cost to both F and F^*.

4. $v \in Y, u \in Z$: In this case u is a source of F. Moreover, e is in exactly one tree F_i^*. Since $u \in B^*$, it is not a source of F_i^*. Therefore, it contributes the same cost to F_i^* and F^*. Finally, as $e \notin Y \times Y$, it does not contribute to $c_Y(F_i^*)$.
5. $v \in Y, u \in Y$: As $e \in Y \times Y$, for every $i \in [k]$, e contributes the same cost to F_i^* and $c_Y(F_i^*)$. Therefore, the contribution of e to the summation is zero. Finally, as $u \in B$, u is a source of neither F nor F^*, implying that it contributes the same cost to both F and F^*. □

For a forest $F \in \mathcal{F}(B)$ and $f \in \mathcal{F}(F)$, let $OPT(B, F, f)$ be the minimum cost of a forest $F^* \in \mathcal{F}(B^*)$ that is compatible with (F, f). Formally,

$$OPT(B, F, f) \stackrel{def}{=} \min_{F^* \in \mathcal{F}(B^*), F^* \sim (F,f)} c(F^*).$$

The following lemma uses Lemma 2 and provides a recurrence formula for $OPT(B, F, f)$.

Lemma 3. *Let B be a bag of $\mathcal{T}$ with children $B_1, \ldots, B_k$, $F \in \mathcal{F}(B)$, and $f \in \mathcal{F}(F)$. Then,*

$$OPT(B, F, f) = c(F) + \sum_{i=1}^{k} \min_{F_i \in \mathcal{F}(B_i)} \left(\min_{f_i \in \mathcal{F}(F_i), (F_i \cup f_i) \sim (F,f)} OPT(B_i, F_i, f_i) - c_Y(F_i) \right)$$

where Y is defined as in Lemma 2.

Proof. By definition and Lemmata 1 and 2, we have

$$\begin{aligned} OPT(B, F, f) &= \min_{F^* \in \mathcal{F}(B^*), F^* \sim (F,f)} c(F^*) \\ &= \min_{F_1^*, \ldots F_k^*, \forall i \in [k] (F_i^* \in \mathcal{F}(B_i^*), F_i^* \sim (F,f))} \left(c(F) + \sum_{i=1}^{k} \left(c(F_i^*) - c_Y(F_i^*) \right) \right). \\ &= c(F) + \sum_{i=1}^{k} \alpha_i \end{aligned}$$

where

$$\alpha_i = \min_{F_i^* \in \mathcal{F}(B_i^*), F_i^* \sim (F,f)} \left(c(F_i^*) - c_Y(F_i^*) \right).$$

By categorizing forests F_i^* according to the forests F_i and f_i in their decomposition and noting that $c_Y(F_i^*) = c_Y(F_i)$, we get

$$\alpha_i = \min_{F_i \in \mathcal{F}(B_i), f_i \in \mathcal{F}(F_i)} \left(\min_{F_i^* \in \mathcal{F}(B_i^*), F_i^* \sim (F,f), F_i^* \sim (F_i, f_i)} \left(c(F_i^*) - c_Y(F_i^*) \right) \right).$$

Claim. *Let $F_i, f_i, F_{i1}^*, \ldots, F_{ik'}^*$ be the forests of the unique decomposition of F_i^* as described by Lemma 1. Then $F_i^* \sim (F, f)$ if and only if $(F_i \cup f_i) \sim (F, f)$.*

Proof. As all graphs under consideration are subgraphs of forest F^*, they are forests.

($\Rightarrow$) Assume $F_i^* \sim (F, f)$. Then $(F_i^* \cup F)\big|_{V(f)} = (F_i^* \cup F)\big|_{SRC(F)}$ is a subgraph of f. Let $e = (u, v) \in E((F_i \cup f_i \cup F)|_{SRC(F)})$. Then there is a directed path P in $F_i \cup f_i \cup F$ from u to v where $u, v \in SRC(F)$. Consider an arc $e' = (y, z)$ of $E(f_i) \setminus E(F_i) \setminus E(F)$. As $f_i = F_i^*\big|_{SRC(F_i)}$, there is a path P' from y to z in F_i^*. By replacing every such arc of P with the corresponding path, we obtain a path from u to v that is entirely in $F \cup F_i^*$. Then $e = (u, v)$ is an arc of $(F_i^* \cup F)\big|_{SRC(F)}$ which is a subgraph of f. Therefore $e \in E(f)$. We conclude that $(F_i \cup f_i \cup F)|_{SRC(F)}$ is a subgraph of f, thus proving $(F_i \cup f_i) \sim (F, f)$.

($\Leftarrow$) Assume $(F_i \cup f_i) \sim (F, f)$. Then $(F_i \cup f_i \cup F)|_{V(f)} = (F_i \cup f_i \cup F)|_{SRC(F)}$ is a subgraph of f. Let $e = (u, v) \in E((F_i^*)\big|_{SRC(F)})$. Then there is a directed path P in F_i^* from u to v where $u, v \in SRC(F)$ and $x \notin SRC(F)$ for every internal vertex x of P (see Fig. 6). We will show that $F \cup F_i \cup f_i$ contains a path from u to v. Let x be any vertex of $V(P) \setminus \{u\} \cap V(F)$. Since x is not a source of $V(F)$ the arc entering x is in $E(F)$. Therefore, the path from u to x is in F. Let y be the last such vertex in P, i.e., the path from u to y is in F and none of the arcs of the sub-path P' of P from y to v are in F. Therefore, $E(P') \subseteq E(F_i^*) \setminus E(F)$. We will show that $F_i \cup f_i$ contains a path from u to v. We note that $y \in SRC(F_i^*) \subseteq SRC(F_i)$. Let $y = x_1, \ldots, x_k$ be the vertices of $V(P') \cap SRC(F_i)$. Since there is a path consisting of arcs of F_i^* from x_i to x_{i+1} for $i \in [k-1]$, (x_i, x_{i+1}) is an arc of f_i. Therefore, there is a path from y to x_k consisting of arcs of f_i. We will now show that the sub-path of P' from x_k to v consists of arcs of F_i. As $x_k \in SRC(F_i)$, this sub-path starts with a non-empty set of arcs from

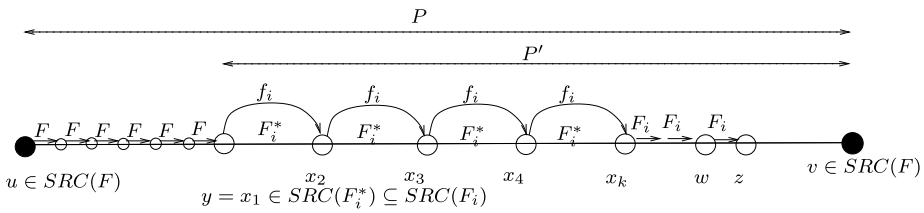


Fig. 6. Constructing a path of $F \cup F_i \cup f_i$ from a path P of $F \cup F_i^*$.

F_i. Let (w, z) be the last arc of F_i in this sub-path. If $z = v$, we are done. Otherwise, $z \in V(F_i) \setminus SRC(F_i) = B_i$. Moreover, $v \in SRC(F) \cap B_i^* \subseteq N_G(B) \cap B_i^*$ implying that $v \in B_i$. Therefore, $v \in V(F_i) \setminus SRC(F_i)$. Then, there is an inbound arc of v in F_i. As F_i^* is a forest, this arc is the last arc of P, contradicting the fact that $z \neq v$. We have thus shown that there is a path from u to v consisting of arcs from $F \cup f_i \cup F_i$. Therefore, (u, v) is an arc of $F_i \cup f_i \cup F|_{SRC(F)}$. Since $(F_i \cup f_i) \sim (F, f)$, (u, v) is an arc of f. Therefore, $(F_i^*)\big|_{SRC(F)}$ is a subgraph of f, i.e., $F_i^* \sim (F, f)$. □

We now proceed with the proof of the Lemma. Using the claim, the last equality can be rewritten as

$$\begin{aligned}
\alpha_i &= \min_{F_i \in \mathcal{F}(B_i),\, f_i \in \mathcal{F}(F_i),\, (F_i \cup f_i) \sim (F, f)} \left(\min_{F_i^* \in \mathcal{F}(B_i^*),\, F_i^* \sim (F_i, f_i)} c(F_i^*) - c_Y(F_i) \right) \\
&= \min_{F_i \in \mathcal{F}(B_i),\, f_i \in \mathcal{F}(F_i),\, (F_i \cup f_i) \sim (F, f)} (OPT(B_i, F_i, f_i) - c_Y(F_i)) \\
&= \min_{F_i \in \mathcal{F}(B_i)} \left(\min_{f_i \in \mathcal{F}(F_i),\, (F_i \cup f_i) \sim (F, f)} OPT(B_i, F_i, f_i) - c_Y(F_i) \right) \qquad \square
\end{aligned}$$

Algorithm 2 FindMinCCATreewidth(G, X, χ, r, c).

Initialization:
1: Compute a tree decomposition $\mathcal{T}$ of G of width $tw(G)$.
2: Direct $\mathcal{T}$ such that the unique bag B_r that contains r is the root.

Dynamic Programming:
3: **for all** bag $B \neq B_r$, in the order of postorder traversal of $\mathcal{T}$ **do**
4: $Y \leftarrow \emptyset$.
5: **for all** child B' of B in $\mathcal{T}$ **do**
6: $Y \leftarrow Y \cup (B \cap B')$.
7: **for all** $F \in \mathcal{F}(B)$, $f \in \mathcal{F}(F)$ **do**
8: $OPT(B, F, f) \leftarrow c(F)$.
9: **for all** child B' of B in $\mathcal{T}$ **do**
10: $OPT(B, F, f) += \min_{F' \in \mathcal{F}(B')} \left(\min_{f' \in \mathcal{F}(F'),\, (F' \cup f') \sim (F, f)} OPT(B', F', f') - c_Y(F') \right)$.

Find Optimum:
11: Let B_x be the child of B_r in $\mathcal{T}$.
12: Let f_r be the trivial forest with $V(f_r) = \{r\}$, $E(f_r) = \emptyset$.
13: **return** $\min_{F \in \mathcal{F}(B_x)} OPT(B_x, F, f_r)$.

The algorithm: Algorithm FindMinCCATreewidth first computes a tree decomposition $\mathcal{T}$ of G in Line 1. It then performs a bottom-up traversal of the tree via the loop at Line 3. At each bag of $\mathcal{T}$, the algorithm uses the result of Lemma 3 to compute the optimum for every equivalence class in the sub-loop at Line 7. Each equivalence class corresponds to a pair (F, f) where $F \in \mathcal{F}(B)$ and $f \in \mathcal{F}(F)$. The correctness of the algorithm follows directly from Lemma 3. We conclude with its time complexity as follows:

Theorem 2. FindMinCCATreewidth *solves* MinCCA *in time* $O^*((\Delta(G) \cdot tw(G))^{2(tw(G)+1)})$.

Proof. Without loss of generality we assume that $d_G(r) = 1$ and that there is exactly one bag B_r containing r in a tree decomposition $\mathcal{T}$ of G. Moreover, $d_{\mathcal{T}}(B_r) = 1$. Indeed, given a graph G with a tree decomposition $\mathcal{T}$ and $r \in V(G)$, we can build a graph G' such that $V(G') = V(G) \cup \{r'\}$, $E(G') = E(G) \cup \{e_r = rr'\}$. We color e_r using a new color whose reload cost is zero. A tree decomposition $\mathcal{T}'$ of G' can be obtained by adding the bag $B_{r'} = \{r, r'\}$ to $\mathcal{T}$ and making it adjacent to any bag containing r. The width of $\mathcal{T}'$ is equal to the width of $\mathcal{T}$, since any bag contains at least two vertices.

The correctness of algorithm FindMinCCATreewidth is an immediate consequence of Lemmata 1 through 3 and the above discussion. In the sequel, we calculate its time complexity.

The computation of a tree decomposition of G at Line 1 can be done in time linear in the size of G but exponential in $tw(G)$, i.e. $O^*(2^{O(k)})$ [6]. This is less than the time complexity that we want to prove. It is also easy to see that the dominant part of the rest of the computation is at Line 10. We now calculate the time complexity contributed by this line.

For a bag $B \in V(\mathcal{T})$ we have $|\mathcal{F}(B)| \leq \Delta(G)^{|B|} \leq \Delta(G)^{tw(G)+1}$. For any $F \in \mathcal{F}(B)$ we note that F has at most $|B|$ sources since every source has at least one child in B. Therefore, $|\mathcal{F}(F)| \leq (|SRC(F)| - 1)^{|SRC(F)|} \leq (|B| - 1)^{|B|} \leq tw(G)^{tw(G)+1}$. Thus, the number of pairs (F, f) such that $F \in \mathcal{F}(B)$, $f \in \mathcal{F}(F)$ is at most

$$\Delta(G)^{tw(G)+1} tw(G)^{tw(G)+1} = (\Delta(G) \cdot tw(G))^{tw(G)+1}.$$

For every edge (B, B') of $\mathcal{T}$, Line 10 is executed once per one such pair. Therefore, it is executed at most $|E(\mathcal{T})| \cdot (\Delta(G) \cdot tw(G))^{tw(G)+1}$ times.

We now bound the time complexity of one execution of Line 10. The dominant part of this line is the two loops calculating the minimum. This loop is executed as many times as the number of pairs F', f' such that $F' \in \mathcal{F}(B')$ and $f' \in \mathcal{F}(F')$, thus at most $(\Delta(G) \cdot tw(G))^{tw(G)+1}$ times. Testing the condition $(F' \cup f') \sim (F, f)$ is equivalent to test whether $F' \cup f' \cup F$ is a forest and $(F' \cup f' \cup F)\big|_{V(f)}$ is a subgraph of f. The number of arcs of $F' \cup f' \cup F$ is at most $\left|B'\right| + \left|B'\right| + |B| \leq 3(tw(G)+1)$ and every arc should be tested for being in $E(f)$, which can be done in time at most $|E(f)| \leq |B| \leq tw(G)+1$. Therefore, the condition can be tested in time $3(tw(G)+1)^2$. We conclude that the time complexity of one execution of Line 10 is $O((\Delta(G) \cdot tw(G))^{tw(G)+1} \cdot (tw(G)+1)^2)$. Multiplying by the number of times this line is executed and omitting the polynomial factor $|E(\mathcal{T})| \cdot (tw(G)+1)^2$, we get the claimed time complexity. □

Corollary 1. MinCCA *is in* FPT *when parameterized by* $tw(G) + \Delta(G)$.

Corollary 2. MinCCA *is in* XP *when parameterized by* $tw(G)$.

4. Conclusion

In this paper, we studied the MinCCA problem on bounded treewidth graphs. We first proposed a polynomial-time optimal algorithm for cactus graphs (i.e., graphs of treewidth at most 2), and then generalized this result to obtain an optimal polynomial-time algorithm for any graph of bounded treewidth. The running time of our algorithm implies that MinCCA is fixed parameter tractable when the parameter is $tw(G) + \Delta(G)$ where $tw(G)$ and $\Delta(G)$ are the treewidth and the maximum degree of the input graph G, respectively. The question of whether MinCCA parameterized by the treewidth alone is in FPT remains open. Our dynamic programming algorithm can be modified so that $\Delta(G)$ is replaced by the number $|X|$ of available of colors: Our algorithm extends forests by adding a parent to every source. For this purpose, we guess all possible parents, and the number of possibilities may be as much as $\Delta(G)$. We observe that the cost of the extension depends only on the colors of these edges. Therefore, instead of guessing parents, we can guess colors of the edge leading to the parent. This modification will make our algorithm an FPT with parameter $tw(G) + |X|$.

As an interesting direction for future work, we propose to study the solvability of MinCCA using *monadic second-order logic with edge set quantification (MSO2)*. Recall the fundamental result of Courcelle [8], stating that for each graph property that can be formulated in MSO2, there is a linear time algorithm that verifies if the property holds for a given graph G, if we have a bounded width tree decomposition of G. This result has been extended also to optimization problems (see, e.g., [4]). We note that an MSO2 formulation may be useful already for input graphs whose edges are two-colored, which are not known to be in FPT.

Another avenue for further research is to investigate MinCCA on graphs of bounded *cliquewidth*. Alternatively, it would be interesting to consider MinCCA on grids and planar graphs, which have unbounded cliquewidth (and hence unbounded treewidth). Finding polynomial-time algorithms, or deriving NP-hardness results for these special graph classes, would shed more light on the combinatorial properties of MinCCA.

References

[1] I. Akyildiz, W. Lee, M. Vuran, S. Mohanty, Next generation/dynamic spectrum access/cognitive radio wireless networks: a survey, Comput. Netw. 50 (13) (2006) 2127–2159.
[2] E. Amaldi, G. Galbiati, F. Maffioli, On minimum reload cost paths, tours, and flows, Networks 57 (3) (2011) 254–260.
[3] S. Arkoulis, E. Anifantis, V. Karyotis, S. Papavassiliou, On the optimal, fair and channel-aware cognitive radio network reconfiguration, Comput. Netw. 57 (8) (2013) 1739–1757.
[4] S. Arnborg, J. Lagergren, D. Seese, Easy problems for tree-decomposable graphs, J. Algorithms 12 (2) (1991) 308–340.
[5] S. Bayhan, F. Alagöz, Scheduling in centralized cognitive radio networks for energy efficiency, IEEE Trans. Veh. Technol. 62 (2) (2013) 582–595.
[6] H.L. Bodlaender, A linear-time algorithm for finding tree-decompositions of small treewidth, SIAM J. Comput. 25 (6) (1996) 1305–1317.
[7] A. Brandstädt, V.B. Lee, J.P. Spinrad, Graph Classes: A Survey, SIAM Monographs on Discrete Mathematics and Applications, vol. 3, 1999.
[8] B. Courcelle, Graph rewriting: an algebraic and logic approach, in: Handbook of Theoretical Computer Science, in: Formal Models and Sematics (B), vol. B, 1990, pp. 193–242.
[9] S. Even, Graph Algorithms, Computer Science Press, 1979.
[10] G. Galbiati, The complexity of a minimum reload cost diameter problem, Discrete Appl. Math. 156 (18) (2008) 3494–3497.
[11] G. Galbiati, S. Gualandi, F. Maffioli, On minimum changeover cost arborescences, in: Experimental Algorithms – 10th International Symposium, SEA, Kolimpari, Chania, Crete, Greece, May 2011, pp. 112–123.

[12] G. Galbiati, S. Gualandi, F. Maffioli, On minimum reload cost cycle cover, Discrete Appl. Math. 164 (2014) 112–120.
[13] I. Gamvros, L. Gouveia, S. Raghavan, Reload cost trees and network design, Networks 59 (4) (2012) 365–379.
[14] L. Gourvès, A. Lyra, C. Martinhon, J. Monnot, The minimum reload s–t path, trail and walk problems, Discrete Appl. Math. 158 (13) (July 2010) 1404–1417.
[15] D. Gözüpek, S. Buhari, F. Alagöz, A spectrum switching delay-aware scheduling algorithm for centralized cognitive radio networks, IEEE Trans. Mob. Comput. 12 (7) (2013) 1270–1280.
[16] D. Gözüpek, M. Shalom, A. Voloshin, S. Zaks, On the complexity of constructing minimum changeover cost arborescences, Theoret. Comput. Sci. 540–541 (2014) 40–52.
[17] F. Harary, R.Z. Norman, Dissimilarity characteristic theorems for graphs, Proc. Amer. Math. Soc. 11 (2) (1960) 332–334.
[18] J. Hopcroft, R. Tarjan, Efficient algorithms for graph manipulation, Commun. ACM 16 (6) (1973) 372–378.
[19] J. Kleinberg, E. Tardos, Algorithm Design, Addison–Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.
[20] B. Wang, K. Liu, Advances in cognitive radio networks: a survey, IEEE J. Sel. Top. Signal Process. 5 (1) (2011) 5–23.
[21] H. Wirth, J. Steffan, Reload cost problems: minimum diameter spanning tree, Discrete Appl. Math. 113 (1) (2001) 73–85.