European Journal of Combinatorics 31 (2010) 1372–1384

Contents lists available at ScienceDirect

European Journal of Combinatorics

journal homepage: www.elsevier.com/locate/ejc

Minimal identifying codes in trees and planar graphs with large girth

David Auger

Télécom ParisTech, 46 rue Barrault, 75634 Paris Cedex 13, France

ARTICLE INFO

Article history:
Received 7 May 2009
Accepted 12 October 2009
Available online 19 January 2010

ABSTRACT

Let G be a finite undirected graph with vertex set $V(G)$. If $v \in V(G)$, let $N[v]$ denote the closed neighbourhood of v, i.e. v itself and all its adjacent vertices in G. An identifying code in G is a subset $\mathcal{C}$ of $V(G)$ such that the sets $N[v] \cap \mathcal{C}$ are nonempty and pairwise distinct for each vertex $v \in V(G)$. We consider the problem of finding the minimum size of an identifying code in a given graph, which is known to be *NP*-hard. We give a linear algorithm that solves it in the class of trees, but show that the problem remains *NP*-hard in the class of planar graphs with arbitrarily large girth.

1. Introduction

By a *graph* we mean a finite, undirected graph, without loops and multiple edges. If G is a graph, we denote respectively by $V(G)$ and $E(G)$ the sets of vertices and edges of G. An edge $\{x, y\} \in E(G)$ with $x, y \in V(G)$ will be simply denoted by xy. We refer the reader to [3] for basic notions such as *adjacent* vertices, *paths*, *cycles* and *connectivity*. Let us just recall that a *tree* is a connected graph with no cycles, and that the *girth* of a graph G is the smallest possible length of a cycle in G.

The *closed neighbourhood* of a vertex $v \in V(G)$ is the set $N_G[v]$ (or simply $N[v]$ when there is no ambiguity) containing v and all its adjacent vertices in G. We recall that the *degree* of a vertex $v \in V(G)$ is the number $d(v)$ of vertices $w \in V(G)$ such that $vw \in E(G)$. The *maximum degree* of G is the maximum possible degree of a vertex in G. Finally, a graph is *planar* if it can be drawn in the plane in such a way that its edges do not cross. For precise definitions and additional background about graphs, we refer the reader once again to [3]. Readers will also need basic knowledge of notions of algorithmic complexity such as polynomial reduction and *NP*-completeness; for these notions we refer them to [7].

E-mail address: auger@telecom-paristech.fr.

0195-6698/$ – see front matter
doi:10.1016/j.ejc.2009.11.012

In this paper, what we call a *code* is simply a set of vertices $\mathcal{C} \subseteq V(G)$, and we refer to its elements as *codewords*. A codeword c in a code $\mathcal{C}$ is said to *cover* a vertex v if $v \in N[c]$; we say that v is covered by $\mathcal{C}$ if it is covered by at least one codeword. If v, w are distinct vertices in $V(G)$, we say that a codeword $c \in \mathcal{C}$ *separates* v and w, or v from w, if c covers exactly one of these two vertices.

An *identifying code* in G is a code $\mathcal{C} \subseteq V(G)$ such that all the sets

$$N[v] \cap \mathcal{C}$$

for $v \in V(G)$ are nonempty and pairwise distinct.

Equivalently, $\mathcal{C}$ is an identifying code if every vertex v of G is *identified* by $\mathcal{C}$, i.e. if v is covered by $\mathcal{C}$ and separated from every other vertex of G by at least one codeword.

The notion of an identifying code was introduced in [9] with the original motivation of achieving fault diagnosis for multiprocessor systems. The general idea is the following: assume that vertices in the graph are processors, and that a codeword is a processor equipped with a sensor, with the ability to detect a faulty processor if it is in its closed neighbourhood. Then if there is at most one faulty processor and if every codeword sends us one bit of information, relating to whether it detects a faulty processor or not, the fact that $\mathcal{C}$ is an identifying code enables us to find out, from the $|\mathcal{C}|$ bits of information received, whether there is a fault in the graph, and also to deduce its position if there is one.

It was proved in [5] that the problem of finding the minimum size of an identifying code in a given graph is *NP*-hard. In the following sections, we first provide a linear algorithm which computes the minimum size of an identifying code in a given tree, and then exhibit some classes of graphs, in a certain sense as close as desired to the class of trees, where the problem of finding the minimum size of an identifying code remains *NP*-hard. For combinatorial results concerning minimal identifying codes in trees, see [1] and [2]. One can also find results for identifying codes in some graphs with high girth in [10].

Algorithms for similar coding problems in trees are known: see [4] for an algorithm computing a minimal identifying code in an *oriented* tree, and [6] and [12] for algorithms computing minimal locating–dominating codes.

For a comprehensive bibliography concerning identifying codes and locating–dominating codes, see [11].

2. An algorithm for minimum identifying codes in trees

In this section we prove the following result:

Theorem 1. *There exists a linear algorithm which computes the minimum size of an identifying code in a given tree.*

2.1. Almost identifying codes

For the algorithm mentioned in Theorem 1 we will need the following notion: if G is a graph and $A \subseteq V(G)$, we say that a subset $\mathcal{C}$ of $V(G)$ is an *A-almost identifying code* of G if the sets

$$\mathcal{C} \cap N[v]$$

are all nonempty and pairwise distinct for all v in $V(G) \setminus A$. Thus an $\emptyset$-almost identifying code is just an identifying code.

In an A-almost identifying code, the vertices in A can be chosen as codewords and thus used to identify vertices in $V(G) \setminus A$, but do not need to be identified. If $v \in V(G)$, we write *v-almost identifying* for $\{v\}$-almost identifying.

Consider a graph G, a vertex $v \in V(G)$ and a v-almost identifying code $\mathcal{C}$. We say that:

- $\mathcal{C}$ satisfies the ID property (for *identifying*) if $\mathcal{C}$ is an identifying code in G;
- $\mathcal{C}$ satisfies the CO property (for *code*) if $v \in \mathcal{C}$;
- $\mathcal{C}$ satisfies the ADJ property (for *adjacent*) if v is adjacent to a codeword;

- $\mathcal{C}$ satisfies the FN property (for *favoured neighbour*) if there exists a neighbour w of v such that $N[w] \cap \mathcal{C} = \{v\}$; in this case we say that w is the favoured neighbour of v, in the sense that v is the only codeword covering w; since $\mathcal{C}$ is v-almost identifying, v admits at most one favoured neighbour;
- $\mathcal{C}$ satisfies $\overline{\text{ID}}$, $\overline{\text{CO}}$, $\overline{\text{ADJ}}$ or $\overline{\text{FN}}$ respectively if $\mathcal{C}$ does not satisfy properties ID, CO, ADJ or FN.

There exist dependence relationships between these properties; for instance the reader will easily check that:

- if $\mathcal{C}$ satisfies FN, then it satisfies CO;
- if $\mathcal{C}$ satisfies ID, then it satisfies CO or ADJ;
- if $\mathcal{C}$ satisfies ID, CO and FN, then it must satisfy ADJ.

2.2. Main and auxiliary functions

Let a tree T be given and let $v \in V(T)$. An identifying code of T can be viewed as a v-almost identifying code in T satisfying property ID; we denote by

$$f_{\text{ID}}(v, T)$$

the minimum size of such a code. More generally, if Π_i denotes a possible property of a v-almost identifying code for $1 \leq i \leq k$ (like ID, $\overline{\text{CO}}$, etc.), we denote by

$$f_{\Pi_1,\ldots,\Pi_k}(v, T)$$

the minimum size of a v-almost identifying code in T satisfying all the properties Π_i for $1 \leq i \leq k$; if such a code does not exist, the function takes the value $+\infty$.

We need to consider 17 functions; the first 10 we call *main functions* and the latter 7 *auxiliary functions*. Table 1 gives the list of the main functions, whereas auxiliary functions are given in Table 2; this table also gives simple formulas showing how auxiliary functions can be computed from main functions.

2.3. The algorithm AIC

The algorithm mentioned in Theorem 1 consists in choosing (randomly) a vertex v_1 in a given graph T and computing the values of the 17 main and auxiliary functions on (v_1, T), and then computing $f_{\text{ID}}(v_1, T)$ by

$$f_{\text{ID}}(v, T) = \min \begin{cases} f_{\text{ID},\text{CO},\text{ADJ},\text{FN}}(v_1, T), \\ f_{\text{ID},\text{CO},\text{ADJ},\overline{\text{FN}}}(v_1, T), \\ f_{\text{ID},\text{CO},\overline{\text{ADJ}}}(v_1, T), \\ f_{\text{ID},\overline{\text{CO}},\text{ADJ}}(v_1, T). \end{cases}$$

Remember that this value is the minimum size of an identifying code in T. In order to do this we define an algorithm AIC (for *almost identifying code*) which returns the values of the 17 main and auxiliary functions for a given pair (v_1, T), where T is a tree and v_1 a vertex of T.

Algorithm AIC recursively computes the values of the 17 functions in smaller and smaller trees. It uses the following facts:

- First, if T consists of a single vertex v_1, then the values of the 17 functions are easy to compute; these values are given in Table 3 (used in line 2 of the algorithm).
- Second, one can compute the values of the 7 auxiliary functions on (v_1, T) from the values of the 10 main functions on (v_1, T), using the formulas given in Table 2 (used in line 8 of the algorithm).
- Finally, if T consists of at least two vertices, then the vertex v_1 has at least one neighbour v_2 in T; this is the central step of the algorithm where we use recursion. If we remove the edge v_1v_2 from T, we obtain two trees T_1 and T_2 respectively containing v_1 and v_2 (see Fig. 1). We claim that the values of the 10 main functions on (v_1, T) can be computed from the values of the 17 main and auxiliary functions on (v_1, T_1) and (v_2, T_2). The formulas showing how this can be done are given in Table 4. This is used in line 7 of the algorithm.

Algorithm 1 AIC

Input: a tree T and a vertex v_1 of T.
Output: the list ℓ of the values of the 17 main and auxiliary functions on (v_1, T).

```
if v1 has degree 0 in T then
    initalize ℓ (Table 3);
else
    let v2 be a neighbour of v1 in T;
    let T1 and T2 be the trees respectively containing v1 and v2 as vertices, obtained from T by
    deletion of the edge v1v2;
    let l1 =AIC(v1, T1) and l2 =AIC(v2, T2);
    compute the 10 main functions on (v1, T) from l1 and l2 (Table 4);
    compute the 7 auxiliary functions on (v1, T) from the main functions on (v1, T) (Table 2);
end if
return the list ℓ of the values the 17 main and auxiliary functions on (v, T).
```

Table 1
List of main functions.

Number	Function
1	$f_{\mathrm{ID},\mathrm{CO},\mathrm{ADJ},\mathrm{FN}}$
2	$f_{\mathrm{ID},\mathrm{CO},\mathrm{ADJ},\overline{\mathrm{FN}}}$
3	$f_{\mathrm{ID},\mathrm{CO},\overline{\mathrm{ADJ}}}$
4	$f_{\mathrm{ID},\overline{\mathrm{CO}},\mathrm{ADJ}}$
5	$f_{\mathrm{CO},\mathrm{ADJ},\mathrm{FN}}$
6	$f_{\mathrm{CO},\mathrm{ADJ},\overline{\mathrm{FN}}}$
7	$f_{\mathrm{CO},\overline{\mathrm{ADJ}},\mathrm{FN}}$
8	$f_{\mathrm{CO},\overline{\mathrm{ADJ}},\overline{\mathrm{FN}}}$
9	$f_{\overline{\mathrm{CO}},\mathrm{ADJ}}$
10	$f_{\overline{\mathrm{CO}},\overline{\mathrm{ADJ}}}$

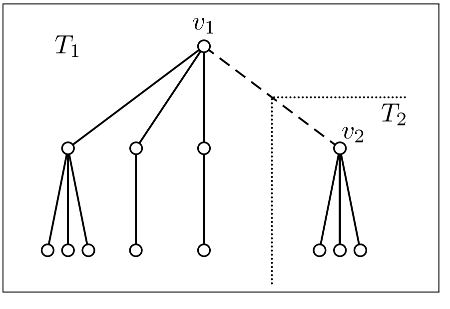

Fig. 1. The trees T_1 and T_2 resulting from the deletion of the edge v_1v_2.

The first two facts are easy to check; only the last one needs a proof. Since proving all cases in Table 4 could be long and tedious, we give hereafter a detailed proof of the first formula as a corollary to Lemma 2; the proofs of all other cases are similar. However, we give in the Appendix an exhaustive list of figures that can be used to check all cases.

Lemma 2. *Let T be a tree and v_1v_2 be an edge of T. Let T_1 and T_2 be the trees, respectively containing v_1 and v_2 as vertices, obtained from T by deletion of the edge v_1v_2. Let $\mathcal{C}$ be a code in T and $\mathcal{C}_i = \mathcal{C} \cap V(T_i)$ for $i \in \{1, 2\}$. Then $\mathcal{C}$ is a v_1-almost identifying code in T with properties* ID, CO, ADJ, FN *if and only if $\mathcal{C}_1$ is a v_1-almost identifying code in T_1 and $\mathcal{C}_2$ is a v_2-almost identifying code in T_2, and one of the following assertions is satisfied:*

(i) *$\mathcal{C}_1$ satisfies* ID, CO, ADJ, $\overline{\mathrm{FN}}$ *and $\mathcal{C}_2$ satisfies* $\overline{\mathrm{CO}}$, $\overline{\mathrm{ADJ}}$*;*
(ii) *$\mathcal{C}_1$ satisfies* CO, ADJ, FN *and $\mathcal{C}_2$ satisfies* CO, $\overline{\mathrm{ADJ}}$*;*

Table 2
List of auxiliary functions.

Number	Function
11	$f_{\text{CO},\overline{\text{ADJ}}}(v,T)=\min\begin{cases}f_{\text{CO},\overline{\text{ADJ}},\text{FN}}(v,T),\\ f_{\text{CO},\overline{\text{ADJ}},\overline{\text{FN}}}(v,T)\end{cases}$
12	$f_{\text{CO},\text{ADJ}}(v,T)=\min\begin{cases}f_{\text{CO},\text{ADJ},\text{FN}}(v,T),\\ f_{\text{CO},\text{ADJ},\overline{\text{FN}}}(v,T)\end{cases}$
13	$f_{\text{CO},\text{FN}}(v,T)=\min\begin{cases}f_{\text{CO},\text{ADJ},\text{FN}}(v,T),\\ f_{\text{CO},\overline{\text{ADJ}},\text{FN}}(v,T)\end{cases}$
14	$f_{\text{CO},\overline{\text{FN}}}(v,T)=\min\begin{cases}f_{\text{CO},\text{ADJ},\overline{\text{FN}}}(v,T),\\ f_{\text{CO},\overline{\text{ADJ}},\overline{\text{FN}}}(v,T)\end{cases}$
15	$f_{\overline{\text{CO}}}(v,T)=\min\begin{cases}f_{\overline{\text{CO}},\text{ADJ}}(v,T),\\ f_{\overline{\text{CO}},\overline{\text{ADJ}}}(v,T)\end{cases}$
16	$f_{\text{CO}}(v,T)=\min\begin{cases}f_{\text{CO},\text{ADJ},\text{FN}}(v,T),\\ f_{\text{CO},\text{ADJ},\overline{\text{FN}}}(v,T),\\ f_{\text{CO},\overline{\text{ADJ}},\text{FN}}(v,T),\\ f_{\text{CO},\overline{\text{ADJ}},\overline{\text{FN}}}(v,T)\end{cases}$
17	$f_{\text{ID},\text{CO}}(v,T)=\min\begin{cases}f_{\text{ID},\text{CO},\text{ADJ},\text{FN}}(v,T),\\ f_{\text{ID},\text{CO},\text{ADJ},\overline{\text{FN}}}(v,T),\\ f_{\text{ID},\text{CO},\overline{\text{ADJ}}}(v,T)\end{cases}$

Table 3
Initialization of the 17 functions on a single vertex.

Number	Function	Value	Number	Function	Value
1	$f_{\text{ID},\text{CO},\text{ADJ},\text{FN}}$	$+\infty$	10	$f_{\overline{\text{CO}},\overline{\text{ADJ}}}$	0
2	$f_{\text{ID},\text{CO},\text{ADJ},\overline{\text{FN}}}$	$+\infty$	11	$f_{\text{CO},\overline{\text{ADJ}}}$	1
3	$f_{\text{ID},\text{CO},\overline{\text{ADJ}}}$	1	12	$f_{\text{CO},\text{ADJ}}$	$+\infty$
4	$f_{\text{ID},\overline{\text{CO}},\text{ADJ}}$	$+\infty$	13	$f_{\text{CO},\text{FN}}$	$+\infty$
5	$f_{\text{CO},\text{ADJ},\text{FN}}$	$+\infty$	14	$f_{\text{CO},\overline{\text{FN}}}$	1
6	$f_{\text{CO},\text{ADJ},\overline{\text{FN}}}$	$+\infty$	15	$f_{\overline{\text{CO}}}$	0
7	$f_{\text{CO},\overline{\text{ADJ}},\text{FN}}$	$+\infty$	16	f_{CO}	1
8	$f_{\text{CO},\overline{\text{ADJ}},\overline{\text{FN}}}$	1	17	$f_{\text{ID},\text{CO}}$	1
9	$f_{\overline{\text{CO}},\text{ADJ}}$	$+\infty$			

(iii) $\mathcal{C}_1$ *satisfies* ID, CO, ADJ, FN *and* $\mathcal{C}_2$ *satisfies* $\overline{\text{CO}}$, ADJ*;*
(iv) $\mathcal{C}_1$ *satisfies* CO, FN *and* $\mathcal{C}_2$ *satisfies* CO, ADJ.

Proof. This case is depicted in Fig. A.1 in the Appendix.

Suppose first that $\mathcal{C}$ is a v_1-almost identifying code in T with properties ID, CO, ADJ, FN.

Observe that when going from T to T_1, only v_1 in T_1 loses a neighbour in the operation and so in T_1, for $v \neq v_1$, we have $N_{T_1}[v] \cap \mathcal{C}_1 = N_T[v] \cap \mathcal{C}$; thus $\mathcal{C}_1$ is a v_1-almost identifying code in T_1. A similar argument shows that $\mathcal{C}_2$ is a v_2-almost identifying code in T_2.

Next consider the code $\mathcal{C}_2$; elementary logic implies that one of the four possibilities ($\overline{\text{CO}}$ and $\overline{\text{ADJ}}$), (CO and $\overline{\text{ADJ}}$), ($\overline{\text{CO}}$ and ADJ) and (CO and ADJ) must happen. In all cases, since $\mathcal{C}$ satisfies CO and v_1 is a vertex of T_1, the code $\mathcal{C}_1$ will satisfy CO.

If $\mathcal{C}_2$ satisfies $\overline{\text{CO}}$, $\overline{\text{ADJ}}$ (top left square in Fig. A.1), since $v_2 \notin \mathcal{C}_2$ and $\mathcal{C}$ satisfies ADJ, the code $\mathcal{C}_1$ must satisfy ADJ. Since $v_2 \notin \mathcal{C}$, it cannot contribute to identifying v_1 in T and so $\mathcal{C}_1$ must satisfy ID. Finally, since $\mathcal{C}$ is v_1-almost identifying and v_2 is a favoured neighbour of v_1, no favoured neighbour for v_1 is allowed in T_1: thus $\mathcal{C}_1$ satisfies ID, CO, ADJ and $\overline{\text{FN}}$.

If $\mathcal{C}_2$ satisfies CO, $\overline{\text{ADJ}}$ (top right square in Fig. A.1), since $\mathcal{C}$ satisfies ID, the vertices v_1 and v_2 must be separated by a codeword of $\mathcal{C}$, which is either a neighbour of v_1, distinct from v_2, or a neighbour of v_2, distinct from v_1; but since $\mathcal{C}_2$ satisfies $\overline{\text{ADJ}}$ the second possibility cannot happen and so $\mathcal{C}_1$ satisfies ADJ. Next, since $\mathcal{C}$ satisfies FN and $\mathcal{C}_2$ satisfies CO, the favoured neighbour of v_1 is not v_2, and so $\mathcal{C}_1$ must satisfy FN. Thus $\mathcal{C}_1$ satisfies CO, ADJ and FN.

Table 4
Recurrence formulas for main functions.

1	$f_{\mathrm{ID,CO,ADJ,FN}}(v_1,T) = \min \begin{cases} f_{\mathrm{ID,CO,ADJ},\overline{\mathrm{FN}}}(v_1,T_1) + f_{\overline{\mathrm{CO}},\overline{\mathrm{ADJ}}}(v_2,T_2), \\ f_{\mathrm{CO,ADJ,FN}}(v_1,T_1) + f_{\mathrm{CO},\overline{\mathrm{ADJ}}}(v_2,T_2), \\ f_{\mathrm{ID,CO,ADJ,FN}}(v_1,T_1) + f_{\overline{\mathrm{CO}},\mathrm{ADJ}}(v_2,T_2), \\ f_{\mathrm{CO,FN}}(v_1,T_1) + f_{\mathrm{CO,ADJ}}(v_2,T_2) \end{cases}$
2	$f_{\mathrm{ID,CO,ADJ},\overline{\mathrm{FN}}}(v_1,T) = \min \begin{cases} f_{\mathrm{CO,ADJ},\overline{\mathrm{FN}}}(v_1,T_1) + f_{\mathrm{CO},\overline{\mathrm{ADJ}}}(v_2,T_2), \\ f_{\mathrm{ID,CO,ADJ},\overline{\mathrm{FN}}}(v_1,T_1) + f_{\overline{\mathrm{CO}},\mathrm{ADJ}}(v_2,T_2), \\ f_{\mathrm{CO},\overline{\mathrm{FN}}}(v_1,T_1) + f_{\mathrm{CO,ADJ}}(v_2,T_2) \end{cases}$
3	$f_{\mathrm{ID,CO},\overline{\mathrm{ADJ}}}(v_1,T) = f_{\mathrm{ID,CO},\overline{\mathrm{ADJ}}}(v_1,T_1) + f_{\overline{\mathrm{CO}},\mathrm{ADJ}}(v_2,T_2)$
4	$f_{\mathrm{ID},\overline{\mathrm{CO}},\mathrm{ADJ}}(v_1,T) = \min \begin{cases} f_{\overline{\mathrm{CO}},\mathrm{ADJ}}(v_1,T_1) + f_{\mathrm{ID,CO},\overline{\mathrm{ADJ}}}(v_2,T_2), \\ f_{\mathrm{ID},\overline{\mathrm{CO}},\mathrm{ADJ}}(v_1,T_1) + f_{\mathrm{ID},\overline{\mathrm{CO}},\mathrm{ADJ}}(v_2,T_2), \\ f_{\overline{\mathrm{CO}},\mathrm{ADJ}}(v_1,T_1) + f_{\mathrm{ID,CO,ADJ,FN}}(v_2,T_2), \\ f_{\overline{\mathrm{CO}}}(v_1,T_1) + f_{\mathrm{ID,CO,ADJ},\overline{\mathrm{FN}}}(v_2,T_2) \end{cases}$
5	$f_{\mathrm{CO,ADJ,FN}}(v_1,T) = \min \begin{cases} f_{\mathrm{CO,ADJ},\overline{\mathrm{FN}}}(v_1,T_1) + f_{\overline{\mathrm{CO}},\overline{\mathrm{ADJ}}}(v_2,T_2), \\ f_{\mathrm{CO,ADJ,FN}}(v_1,T_1) + f_{\mathrm{CO},\overline{\mathrm{ADJ}}}(v_2,T_2), \\ f_{\mathrm{CO,ADJ,FN}}(v_1,T_1) + f_{\overline{\mathrm{CO}},\mathrm{ADJ}}(v_2,T_2), \\ f_{\mathrm{CO,FN}}(v_1,T_1) + f_{\mathrm{CO,ADJ}}(v_2,T_2) \end{cases}$
6	$f_{\mathrm{CO,ADJ},\overline{\mathrm{FN}}}(v_1,T) = \min \begin{cases} f_{\mathrm{CO},\overline{\mathrm{FN}}}(v_1,T_1) + f_{\mathrm{CO}}(v_2,T_2), \\ f_{\mathrm{CO,ADJ},\overline{\mathrm{FN}}}(v_1,T_1) + f_{\overline{\mathrm{CO}},\mathrm{ADJ}}(v_2,T_2) \end{cases}$
7	$f_{\mathrm{CO},\overline{\mathrm{ADJ}},\mathrm{FN}}(v_1,T) = \min \begin{cases} f_{\mathrm{CO},\overline{\mathrm{ADJ}},\overline{\mathrm{FN}}}(v_1,T_1) + f_{\overline{\mathrm{CO}},\overline{\mathrm{ADJ}}}(v_2,T_2), \\ f_{\mathrm{CO},\overline{\mathrm{ADJ}},\mathrm{FN}}(v_1,T_1) + f_{\overline{\mathrm{CO}},\mathrm{ADJ}}(v_2,T_2) \end{cases}$
8	$f_{\mathrm{CO},\overline{\mathrm{ADJ}},\overline{\mathrm{FN}}}(v_1,T) = f_{\mathrm{CO},\overline{\mathrm{ADJ}},\overline{\mathrm{FN}}}(v_1,T_1) + f_{\overline{\mathrm{CO}},\mathrm{ADJ}}(v_2,T_2)$
9	$f_{\overline{\mathrm{CO}},\mathrm{ADJ}}(v_1,T) = \min \begin{cases} f_{\overline{\mathrm{CO}}}(v_1,T_1) + f_{\mathrm{ID,CO}}(v_2,T_2), \\ f_{\overline{\mathrm{CO}},\mathrm{ADJ}}(v_1,T_1) + f_{\mathrm{ID},\overline{\mathrm{CO}},\mathrm{ADJ}}(v_2,T_2) \end{cases}$
10	$f_{\overline{\mathrm{CO}},\overline{\mathrm{ADJ}}}(v_1,T) = f_{\overline{\mathrm{CO}},\overline{\mathrm{ADJ}}}(v_1,T_1) + f_{\mathrm{ID},\overline{\mathrm{CO}},\mathrm{ADJ}}(v_2,T_2)$

If $\mathcal{C}_2$ satisfies $\overline{\mathrm{CO}}$, ADJ (bottom left square in Fig. A.1), since v_2 cannot contribute to the identification of v_1 by $\mathcal{C}$ in T, $\mathcal{C}_1$ must satisfy ID. Since $v_2 \notin \mathcal{C}$ and $\mathcal{C}$ satisfies ADJ, this must also be the case for $\mathcal{C}_1$. Finally, by ADJ for $\mathcal{C}_2$, the favoured neighbour of v_1 in T is not v_2 and so $\mathcal{C}_1$ satisfies FN. Thus $\mathcal{C}_1$ satisfies ID, CO, ADJ and FN.

Eventually, if $\mathcal{C}_2$ satisfies CO, ADJ (bottom right square in Fig. A.1), then once again the favoured neighbour of v_1 in T must be found in T_1 and $\mathcal{C}_1$ satisfies FN. Thus $\mathcal{C}_1$ satisfies CO and FN.

For the converse implications, let us consider the first case: suppose that $\mathcal{C}_1$ satisfies ID, CO, ADJ, $\overline{\mathrm{FN}}$ and $\mathcal{C}_2$ satisfies $\overline{\mathrm{CO}}$, $\overline{\mathrm{ADJ}}$, and let us define $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$, a code in T (see the top left square in Fig. A.1):

- if $v \in V(T_1)$ we have $N_T[v] \cap \mathcal{C} = N_{T_1}[v] \cap \mathcal{C}_1$, and so all vertices in $V(T_1)$ are covered by nonempty and distinct sets of codewords from $\mathcal{C}$ (including v_1, since $\mathcal{C}_1$ satisfies ID);
- a similar observation can be made for vertices in $V(T_2) \setminus \{v_2\}$;
- if $v \in V(T_1)$ and $v' \in V(T_2) \setminus \{v_2\}$, then $N_T[v] \cap N_T[v']$ is empty or equal to $\{v_2\}$, but since $v_2 \notin \mathcal{C}$ we conclude that vertices in $V(T_1)$ are separated from vertices in $V(T_2) \setminus \{v_2\}$: a codeword covering v cannot cover v';
- it remains to settle the score for v_2: we have $N_T[v_2] \cap \mathcal{C} = \{v_1\}$, so v_2 is covered by $\mathcal{C}$; it is separated from v_1, since $\mathcal{C}_1$ satisfies ADJ, and separated from all vertices in $V(T_1) \setminus \{v_1\}$ since these vertices cannot be covered only by v_1 ($\mathcal{C}_1$ satisfies $\overline{\mathrm{FN}}$); and finally, v_2 is separated from vertices in $V(T_2) \setminus \{v_2\}$ since these vertices cannot be covered by v_1.

We have proved that $\mathcal{C}$ is an identifying code, i.e. a v_1-almost identifying code satisfying ID. Moreover, $\mathcal{C}$ obviously satisfies CO, ADJ and FN, with v_2 as the favoured neighbour of v_1.

We skip the proofs for the three remaining cases which are very much similar to the previous one. $\square$

From this lemma we directly deduce the following equality:

Corollary 3. *With the notation of Lemma 2, we have the following equality:*

$$f_{\text{ID,CO,ADJ,FN}}(v_1, T) = \min \begin{cases} f_{\text{ID,CO,ADJ},\overline{\text{FN}}}(v_1, T_1) + f_{\overline{\text{CO}},\overline{\text{ADJ}}}(v_2, T_2), \\ f_{\text{CO,ADJ,FN}}(v_1, T_1) + f_{\text{CO},\overline{\text{ADJ}}}(v_2, T_2), \\ f_{\text{ID,CO,ADJ,FN}}(v_1, T_1) + f_{\overline{\text{CO}},\text{ADJ}}(v_2, T_2), \\ f_{\text{CO,FN}}(v_1, T_1) + f_{\text{CO,ADJ}}(v_2, T_2). \end{cases}$$

With the help of the Appendix, an interested reader can easily check all formulas given in Table 4, and conclude that the algorithm is valid. Before ending this part, let us notice that in the execution of the algorithm, when the function AIC (v_1, T) is called, if the degree of v is at least 1 then an edge is removed from T before computing AIC (v_1, T_1) and AIC (v_2, T_2), and thus the number of calls to the function AIC is at most the number of edges in T. Since each step can be executed in constant time, we conclude that the algorithm runs in linear time. Let us also note that this algorithm could be easily modified in order to output an identifying code of minimal size (it would be sufficient in each computation to keep track of the functions which give the minimal values), or else to compute the number of identifying codes with minimal size in T.

3. Planar graphs with large girth

We proved in the previous section that the problem of finding the minimum size of an identifying code can be solved in linear time in the class of trees. Since it is known that the problem is *NP*-hard in the general case (see [5]), we found it interesting to narrow the gap between these two extremes. Without loss of generality, we restrict ourselves to connected graphs. If $\mathcal{H}$ is a class of graphs, let us call Min ID-code in $\mathcal{H}$ the problem of deciding, for a given graph $G \in \mathcal{H}$ and an integer p, whether G admits an identifying code of size at most p or not.

Let $\mathcal{P}_k^4$ denote the class of connected planar graphs, with maximum degree at most 4, and girth at least k where $k \geq 3$. It should be noted that if k is large the elements of $\mathcal{P}_k^4$ are "nearly" trees, in the sense that

$$\bigcap_{k \geq 3} \mathcal{P}_k^4$$

is the class of trees with maximum degree 4.

We prove the following result:

Theorem 4. *For all $k \geq 3$, the problem* Min ID-code in $\mathcal{P}_k^4$ *is NP-complete.*

Let us start with a lemma.

Lemma 5. *Let $P = av_1v_2 \cdots v_{2k}b$ be a path on $2k + 2$ vertices, where $k \geq 1$. Then:*

- *the minimal size of an $\{a, b\}$-almost identifying code in P which contains neither a nor b is $k + 1$;*
- *the minimal size of an $\{a, b\}$-almost identifying code in P which contains exactly one of a and b is $k+1$;*
- *the minimal size of an $\{a, b\}$-almost identifying code in P which contains a and b is $k + 2$.*

Proof. Let $\mathcal{C}$ be an $\{a, b\}$-almost identifying code in P. For every i such that $1 \leq i \leq 2k - 1$, the vertices v_i and v_{i+1} must be separated by a codeword; let us consider this as a *task* that has to be fulfilled. The vertices v_1 and v_{2k} must be covered by $\mathcal{C}$, giving us two other tasks, for a total of $2k + 1$ tasks. Suppose now that v_i is a codeword: if $i \in \{3, \ldots, 2k - 2\}$, it covers neither v_1 nor v_{2k}, but it separates v_{i-1} from v_{i-2} and v_{i+1} from v_{i+2}; thus v_i fulfills exactly two tasks. If $i \in \{1, 2\}$, then v_i covers v_1 but only separates the vertices v_{i+1} and v_{i+2}, thus also fulfills two tasks, and a similar observation can be made if $i \in \{2k - 1, 2k\}$, and for the vertices a and b.

Therefore, since we have $2k + 1$ tasks and since a given codeword fulfills exactly two of them, we need at least $\left\lceil \frac{2k+1}{2} \right\rceil = k + 1$ codewords in $\mathcal{C}$. Now since $\mathcal{C} \cap \{v_1, v_2, \ldots, v_{2k}\}$ is a $\{v_1, v_{2k}\}$-almost identifying code in the path $P' = v_1v_2 \cdots v_{2k}$, the same observation leads to the conclusion that there are at least k codewords in $\{v_1, v_2, \ldots, v_{2k}\}$: so if at most one of the vertices a, b is a codeword, we have $|\mathcal{C}| \geq k + 1$, and if a and b are codewords we have $|\mathcal{C}| \geq k + 2$.

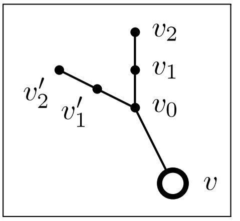

Fig. 2. The structure S_v linked to the vertex v of G in the proof of Theorem 4.

Conversely, it is easy to see that the codes

$$\mathcal{C}_1 = \{v_2, v_4, \ldots, v_{2k}\} \cup \{v_3\},$$
$$\mathcal{C}_2 = \{a, v_2, v_4, \ldots, v_{2k}\}$$

and

$$\mathcal{C}_3 = \{a, b\} \cup \{v_2, v_4, \ldots, v_{2k}\}$$

are $\{a, b\}$-almost identifying with the required conditions. □

Before proving Theorem 4, let us note that one can rapidly check whether a given code in a graph is identifying, and so the problem Min ID-code in $\mathcal{P}_k^4$ is in the class *NP* for all $k \geq 3$. In order to prove its *NP*-completeness it remains to polynomially reduce an *NP*-complete problem to Min ID-code in $\mathcal{P}_k^4$. We use the Min Vertex Cover in $\mathcal{P}^3$ problem.

Let $\mathcal{P}^3$ denote the class of planar graphs with maximum degree at most 3. We recall that a *vertex cover* in a graph G is a code $\mathcal{C} \subseteq V(G)$ such that for every edge $e = ab \in E(G)$, one has $a \in \mathcal{C}$ or $a \in \mathcal{C}$ (or both). The following problem was proved to be *NP*-complete in [8]:

Min Vertex Cover in $\mathcal{P}^3$:
- Instance: a planar graph $G \in \mathcal{P}^3$, and an integer p.
- Question: is there a vertex cover $\mathcal{C}$ of G with $|\mathcal{C}| \leq p$?

Thanks to this result we can now prove Theorem 4.

Proof of Theorem 4. Let $k \geq 3$. Let $G \in \mathcal{P}^3$ and $p \geq 0$ be an instance of Min Vertex Cover in $\mathcal{P}^3$; let n and m respectively denote the number of vertices and edges in G. We give a polynomial time construction of a graph $G' \in \mathcal{P}_k^4$ such that

$$G \text{ admits a vertex cover of size at most } p \text{ if and only if} \quad (1)$$
$$G' \text{ admits an identifying code of size at most } p + 3n + km.$$

This will settle the polynomial reduction and thus prove the theorem. The construction goes as follows: we keep the vertices of G but remove all edges. If two vertices a, b were adjacent in G, via the edge $e = ab$, we link them in G' by a path P_{ab} with $2k$ inner vertices. Finally, we link to every vertex v of G a structure S_v which is depicted in Fig. 2. It will be convenient, in this construction, to see $V(G)$ as a subset of $V(G')$. An example of a transformation for a simple graph is depicted in Fig. 3.

Obviously, the maximum degree of G' is the maximum degree of G plus 1 (because of the structures S_v) and G' is planar if G is. We can also note that since edges of G have been replaced by paths of length $2k+1$, the girth of G' is at least $3(2k+1) \geq k$. Thus $G' \in \mathcal{P}_k^4$ if $G \in \mathcal{P}^3$, and the construction is clearly polynomial when k is fixed.

We now prove (1). First, assume that $\mathcal{C}$ is a vertex cover of G with size at most p. Since $V(G)$ is a subset of $V(G')$ we can consider $\mathcal{C}$ as a code in G'. Let us start with $\mathcal{C}' := \mathcal{C}$ and add vertices to $\mathcal{C}'$ in order to build an identifying code of G':

- for $v \in V(G)$, we add to $\mathcal{C}'$ the vertices v_0, v_1 and v_1' in the corresponding structure S_v;
- for every edge ab of G, since $\mathcal{C}$ is a vertex cover of G we must have $a \in \mathcal{C}$ or $b \in \mathcal{C}$; let us denote by $av_1v_2 \cdots v_{2k}b$ the vertices of the path $aP_{ab}b$, and suppose for instance that $a \in \mathcal{C}$: then we add to $\mathcal{C}'$ the vertices $v_2, v_4, \ldots, v_{2k}$ of P_{ab}.

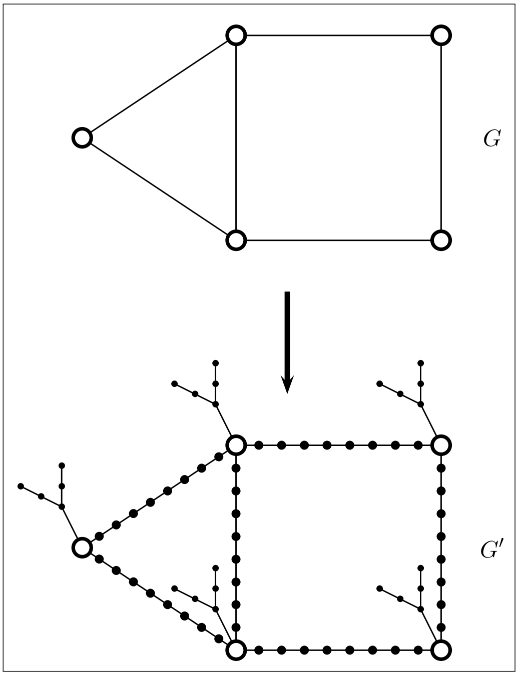

Fig. 3. An example of transformation for $k = 4$ in the proof of Theorem 4.

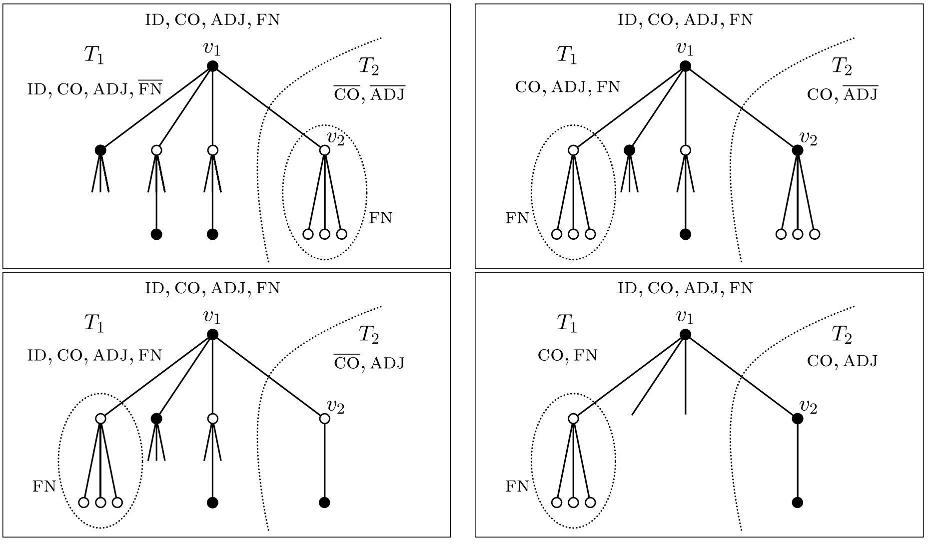

Fig. A.1. Computation of $f_{\text{ID,CO,ADJ,FN}}(v_1, T)$: four cases.

By doing so, we obtain a code $\mathcal{C}'$ with size

$$\left|\mathcal{C}'\right| = |\mathcal{C}| + 3n + km \leq p + 3n + km.$$

It remains to see that $\mathcal{C}'$ is an identifying code of G'. One can easily see that the structures S_v take care of covering and identifying themselves and the vertices of G; thus we just have to look at what

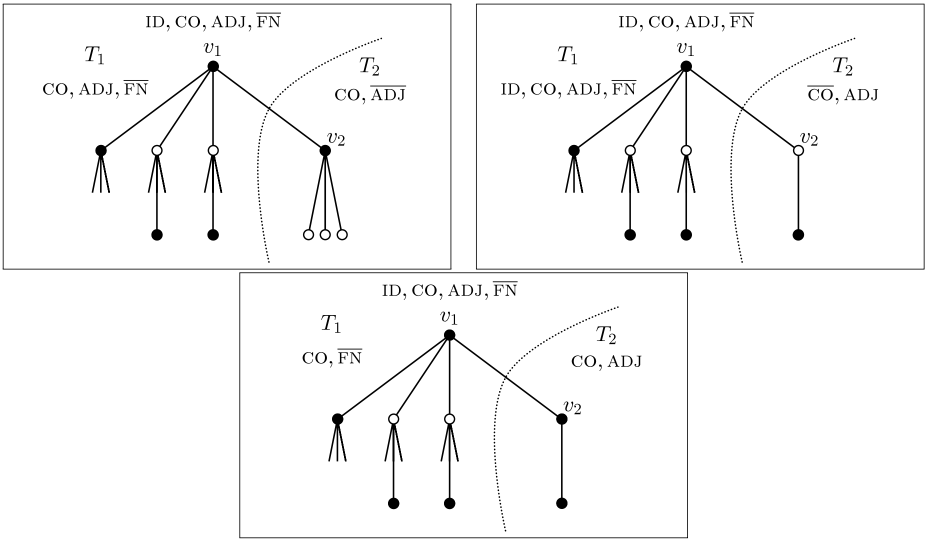

Fig. A.2. Computation of $f_{\text{ID,CO,ADJ},\overline{\text{FN}}}(v_1, T)$: three cases.

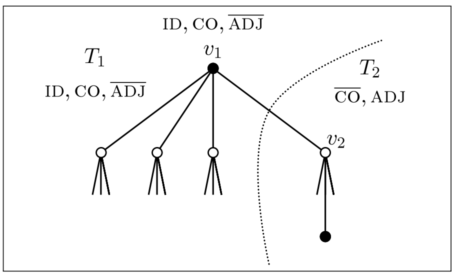

Fig. A.3. Computation of $f_{\text{ID,CO},\overline{\text{ADJ}}}(v_1, T)$: one case.

happens in the paths P_{ab} where the conclusion follows as in the proof Lemma 5. Note in particular that since $\mathcal{C}$ is a vertex cover of G, for every path P_{ab} at least one of the vertices a and b is a codeword.

Conversely, suppose that $\mathcal{C}'$ is an identifying code of G' with size at most $p + 3n + km$. Let us recall that the vertices of G' can be partitioned in the following way:

$$V(G') = V(G) \cup \bigcup_{v \in V(G)} V(S_v) \cup \bigcup_{ab \in E(G)} V(P_{ab}).$$

Then:

- for every $v \in V(G)$, consider the vertices of S_v: v_1 and v_2 must be separated, so we must have $v_0 \in \mathcal{C}'$, and v_2, as well as v_2', must be covered, so $v_1 \in \mathcal{C}'$ or $v_2 \in \mathcal{C}'$, and $v_1' \in \mathcal{C}'$ or $v_2' \in \mathcal{C}'$; all in all, there are at least three codewords of $\mathcal{C}'$ in each S_v;
- for every edge $ab \in E(G)$, by Lemma 5 the path P_{ab} must count at least $k + 1$ codewords if neither a nor b belongs to $\mathcal{C}'$, whereas it must count at least k codewords in the general case.

Let q be the number of *bad* edges of G for the vertex cover, i.e. edges $ab \in E(G)$ such that $a \notin \mathcal{C}'$ and $b \notin \mathcal{C}'$. Then we have

$$\left|\mathcal{C}' \cap V(G)\right| \leq \left|\mathcal{C}'\right| - 3n - (k+1)q - k(m-q)$$

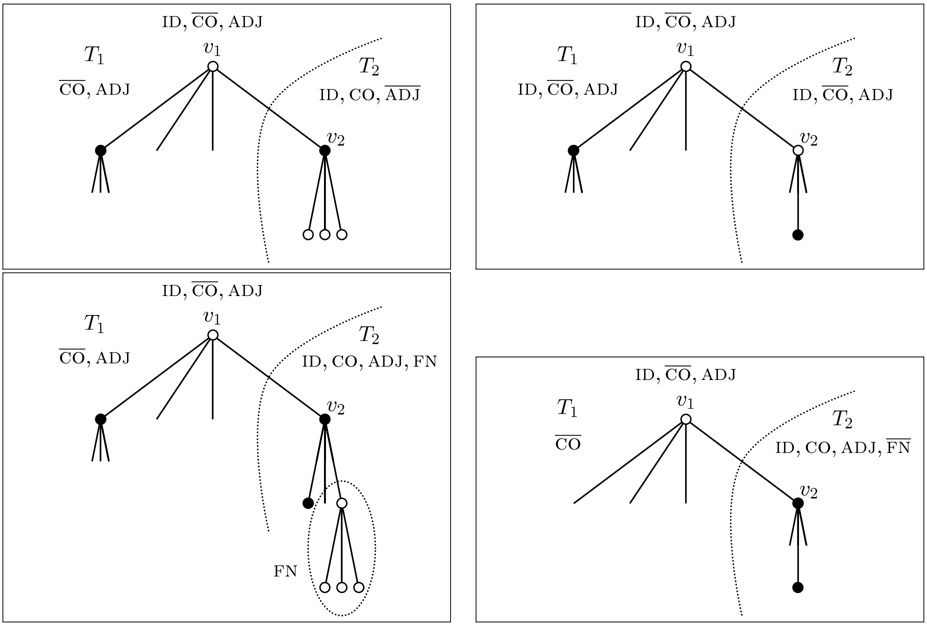

Fig. A.4. Computation of $f_{\mathrm{ID},\overline{\mathrm{CO}},\mathrm{ADJ}}(v_1, T)$: four cases.

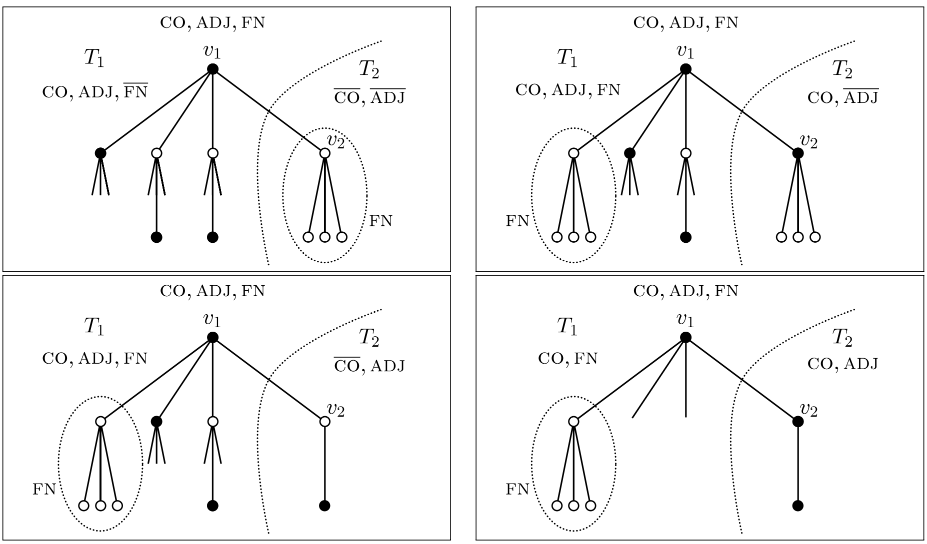

Fig. A.5. Computation of $f_{\mathrm{CO},\mathrm{ADJ},\mathrm{FN}}(v_1, T)$: four cases.

and so since $\left|\mathcal{C}'\right| \leq p + 3n + km$ it follows that

$$\left|\mathcal{C}' \cap V(G)\right| \leq p - q.$$

Thus $\mathcal{C}' \cap V(G)$ is a code in G which may not be a vertex cover; but if we add q vertices to $\mathcal{C}$ (one for every bad edge), we get a vertex cover of G with size at most p. $\square$

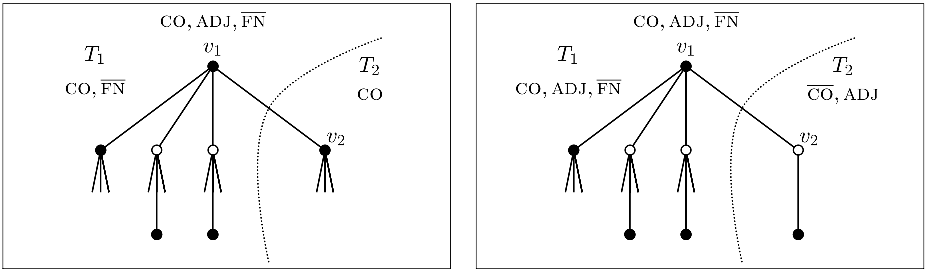

Fig. A.6. Computation of $f_{\mathrm{CO},\mathrm{ADJ},\overline{\mathrm{FN}}}(v_1, T)$: two cases.

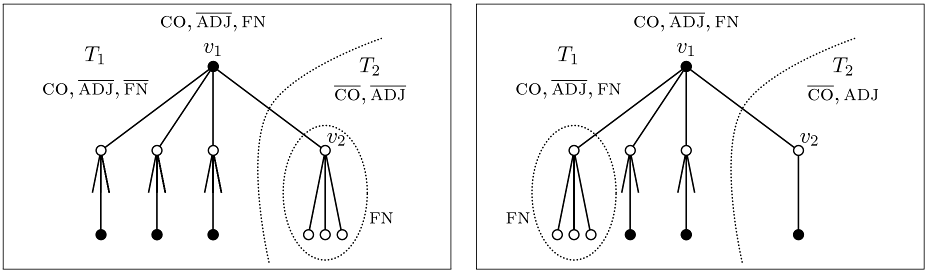

Fig. A.7. Computation of $f_{\mathrm{CO},\overline{\mathrm{ADJ}},\mathrm{FN}}(v_1, T)$: two cases.

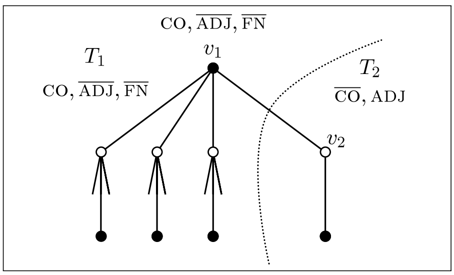

Fig. A.8. Computation of $f_{\mathrm{CO},\overline{\mathrm{ADJ}},\overline{\mathrm{FN}}}(v_1, T)$: one case.

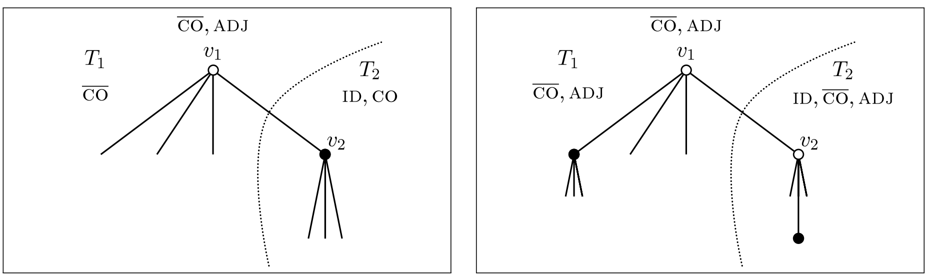

Fig. A.9. Computation of $f_{\overline{\mathrm{CO}},\mathrm{ADJ}}(v_1, T)$: two cases.

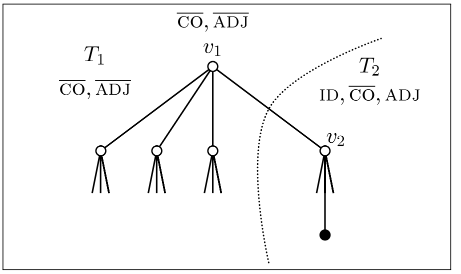

Fig. A.10. Computation of $f_{\overline{\mathrm{CO}},\overline{\mathrm{ADJ}}}(v_1, T)$: one case.

Appendix

In the following (Figs. A.1–A.10), codewords are in black, whereas white vertices are not codewords. An ellipse around some vertices with the label 'FN' means that one of these vertices is a favoured neighbour of v_1 or v_2.

References

[1] N. Bertrand, I. Charon, O. Hudry, A. Lobstein, 1-identifying codes on trees, Australasian Journal of Combinatorics 31 (2005) 21–35.
[2] M. Blidia, M. Chellali, F. Maffray, J. Moncel, A. Semri, Locating–domination and identifying codes in trees, Australasian Journal of Combinatorics 39 (2007) 219–232.
[3] J.A. Bondy, U.S.R. Murty, Graph Theory, Springer, 2008.
[4] I. Charon, S. Gravier, O. Hudry, A. Lobstein, M. Mollard, J. Moncel, A linear algorithm for minimum 1-identifying codes in oriented trees, Discrete Applied Mathematics 154 (2006) 1246–1253.
[5] I. Charon, O. Hudry, A. Lobstein, Minimizing the size of an identifying or locating–dominating code in a graph is NP-hard, Theoretical Computer Science 290 (2003) 2109–2120.
[6] C.J. Colbourn, P.J. Slater, L.K. Stewart, Locating dominating sets in series parallel networks, Congressus Numerantium 56 (1987) 135–162.
[7] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, San Francisco, 1979.
[8] M.R. Garey, D.S. Johnson, The rectilinear Steiner tree problem is *NP*-complete, SIAM Journal on Applied Mathematics 32 (4) (1977) 826–834.
[9] M.G. Karpovsky, K. Chakrabarty, L.B. Levitin, On a new class of codes for identifying vertices in graphs, IEEE Transactions on Information Theory 44 (1998) 599–611.
[10] T. Laihonen, On cages admitting identifying codes, Eurpoean Journal of Combinatorics 29 (2008) 737–741.
[11] A. Lobstein, Bibliography on identifying, locating–dominating and discriminating codes in graphs, http://www.infres.enst.fr/~lobstein/debutBIBidetlocdom.pdf.
[12] P.J. Slater, Domination and location in acyclic graphs, Networks 17 (1987) 55–64.