



# A distributed algorithm for computing and updating the process number of a forest

David Coudert, Florian Huc, Dorian Mazaure

## ► To cite this version:

David Coudert, Florian Huc, Dorian Mazaure. A distributed algorithm for computing and updating the process number of a forest. [Research Report] RR-6560, 2008. inria-00288304v2

**HAL Id: inria-00288304**

**<https://inria.hal.science/inria-00288304v2>**

Submitted on 18 Jun 2008 (v2), last revised 27 Jun 2008 (v3)

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

# *A distributed algorithm for computing and updating the process number of a forest*

David Coudert — Florian Huc — Dorian Mazaauric

N° ????

June 2008

Thème COM



*apport  
de recherche*





## A distributed algorithm for computing and updating the process number of a forest

David Coudert\*, Florian Huc\*, Dorian Mazauric\*

Thème COM — Systèmes communicants  
Projet MASCOTTE

Rapport de recherche n° ??? — June 2008 — 16 pages

**Abstract:** In this paper, we present a distributed algorithm to compute various parameters of a tree such as the process number, the edge search number or the node search number and so the pathwidth. This algorithm requires  $n$  steps, an overall computation time of  $O(n \log n)$ , and  $n$  messages of size  $\log_3 n + 3$ . We then propose a distributed algorithm to update the process number (or the node search number, or the edge search number) of each component of a forest after adding or deleting an edge. This second algorithm requires  $O(D)$  steps, an overall computation time of  $O(D \log n)$ , and  $O(D)$  messages of size  $\log_3 n + 3$ , where  $D$  is the diameter of the modified connected component. Finally, we show how to extend our algorithms to trees and forests of unknown size using messages of less than  $2\alpha + 4 + \varepsilon$  bits, where  $\alpha$  is the parameter to be determined and  $\varepsilon = 1$  for updates algorithms.

**Key-words:** pathwidth, process number, search number, distributed algorithm.

MASCOTTE, INRIA, I3S(CNRS/UNSA), Sophia Antipolis, France.  
{firstname.lastname@sophia.inria.fr}

\* This work was partially funded by the European projects IST FET AEOLUS and COST 293 GRAAL, and done within the CRC CORSO with France Telecom R&D.

## Un algorithme distribué pour le calcul et la mise à jour du process number d'une forêt

**Résumé :** Dans cet article, nous présentons un algorithme distribué permettant de calculer divers paramètres d'un arbre tel le process number, la pathwidth et l'edge search number. Cet algorithme nécessite  $n$  étapes, a un temps d'exécution de  $O(n \log n)$  et génère  $n$  messages de taille  $\log_3 n + 3$ . Nous montrons ensuite comment il peut servir à mettre à jour le process number (ou la pathwidth ou l'edge search number) de chaque composante d'une forêt après l'ajout ou la suppression d'une arête. En fin on montre que cela peut être fait même si la taille de la forêt est inconnue.

**Mots-clés :** pathwidth, process number, search number, algorithme distribué

## 1 Introduction

Treewidth and pathwidth have been introduced by Robertson and Seymour [11] as part of the graph minor project. By definition, the treewidth of a tree is one, but its pathwidth might be up to  $\log n$ . A linear time centralized algorithms to compute the pathwidth of a tree has been proposed in [5, 12, 13], but so far no dynamic algorithm exists.

The algorithmic counter part of the notion of pathwidth is the node searching problem [8]. It consists in finding an invisible and fast fugitive in a graph using the smallest set of agents. The minimum number of agents needed gives the pathwidth. Other graph invariants closely related to the notion of pathwidth have been proposed such as the process number [2, 3] and the edge search number [9]. For this two invariants it is not known if they are strictly equivalent to the pathwidth or not.

In this paper, we propose a dynamic algorithm to compute those different parameters on trees and to update them in a forest after the addition or deletion of an edge. We also show that no distributed algorithm can always transmit a number of bits linear in  $n$  and give a characterisation of the trees whose process number and edge search number equals their pathwidth. To present our results, we concentrate on the process number.

As mentioned before the process number of a (di)graph has been introduced to model a routing reconfiguration problem in WDM or WiFi networks in [2, 3]. The graph represents a set of tasks that have to be realized. A *process strategy* is a serie of actions in order to realize all the tasks represented by the graph. It finishes when all the nodes of the graph are *processed*. In order to process the graph, the three actions we can do are:

- (1) put an agent on a node.
- (2) remove an agent from a node if all its neighbors are either processed or occupied by an agent. The node is now processed.
- (3) process a node if all its neighbors are occupied by an agent (the node is surrounded).

A *p-process strategy* is a strategy which process the graph using  $p$  agents. The *process number* of a graph  $G$ ,  $\text{pn}(G)$ , is the smallest  $p$  such that a  $p$ -process strategy exists. For example, a star has process number 1 (we place an agent on its center), a path of length at least 4 has process number 2, a cycle of size 5 or more has process number 3, and a  $n \times n$  grid has process number  $n + 1$ . Moreover, it has been proved in [2, 3] that  $\text{pw}(G) \leq \text{pn}(G) \leq \text{pw}(G) + 1$ , where  $\text{pw}(G)$  is the *pathwidth* of  $G$  [11].

The node search number [8],  $\text{ns}(G)$ , can be defined similarly except that we only use rules (1) and (2). It was proved by Ellis *et al.* [5] that  $\text{ns}(G) = \text{pw}(G) + 1$ , and by Kinnersley [7] that  $\text{pw}(G) = \text{vs}(G)$ , where  $\text{vs}(G)$  is the *vertex separation* of  $G$ . Those results show that the vertex separation, the node search number and the pathwidth are equivalent. Please refer to recent surveys [6, 4] for more information.

The following Theorem gives a construction which enforces each parameter to grow by 1, which implies that for any tree  $\text{ns}(T)$ ,  $\text{es}(T)$ ,  $\text{pw}(T)$ ,  $\text{vs}(T)$ , and  $\text{pn}(T)$  are less than  $\log_3(n)$ .

**Theorem 1 ([2] and [10])** *Let  $G_1, G_2$  and  $G_3$  be three connected graphs such that  $vs(G_i) = vs$ ,  $ns(G_i) = ns$  and  $pn(G_i) = p$ ,  $1 \leq i \leq 3$ . We construct the graph  $G$  by putting one copy of each of the  $G_i$ , and we add one node  $v$  that has exactly one neighbour in each of the  $G_i$ ,  $1 \leq i \leq 3$ . Then  $vs(G) = vs + 1$ ,  $ns(G) = ns + 1$  and  $pn(G) = p + 1$ .*

The algorithm we propose is based on the decomposition of a tree into subtrees forming a *hierarchical decomposition*. It is fully distributed, can be executed in an asynchronous environment and the construction of the hierarchical decomposition requires only a small amount of information.

It uses ideas similar to the ones used by Ellis *et al.* [5] to design an algorithm which computes the node search number in linear time. However their algorithm is centralized and the distributed version uses  $O(n \log n)$  operations and transmit a total of  $O(n \log n \log(\log n))$  bits. We improve the distributed version as our algorithm also requires  $O(n \log n)$  operations but transmit at most  $n(\log_3 n + 3)$  bits. We also prove that it is optimal in the sense that for any  $k \in \mathbb{N}$ , no dynamic algorithm, such that the vertex at which the edge addition/deletion is done, can only simultaneously sends one message to its neighbours, can always transmit less than  $\frac{k-1}{k}n(\log_3(n))$  bits. Furthermore, with a small increase in the amount of transmitted information, we extend our algorithm to a fully dynamic algorithm allowing to add and remove edges even if the total size of the tree is unknown.

Finally we explain how to adapt our algorithm to compute the node search number and the edge search number of a tree. It should also certainly be adapted to compute the mixed search number and other similar parameters.

This paper start with the presentation of the hierarchical decomposition of a tree in Section 2. Then in Section 3 we present an algorithm to compute the process number of a tree and analyze its complexity. In Section 4 we show how to update efficiently the process number of each component of a forest after the addition or the deletion of any tree edge, thus resulting in a dynamic algorithm. Section 5 concludes this paper with several improvements including extensions of our algorithm to trees of unknown size and to compute other parameters.

All along this paper, we assume that each node  $u$  knows the set of its neighbours which we note  $\Gamma(u)$ . However, the size of the tree is not needed as explained in Section 5.

## 2 Tools for the algorithm

The algorithm is initialized at the leaves. Each leaf sends a message to its only neighbor which becomes its *father*. Then, a node  $v$  which has received messages from all its neighbors but one process them and sends a message to its last neighbor, its *father*. We say that this node has been *visited*. Finally, the last node,  $w$ , receives a message from all its neighbours and computes the process number of  $T$ :  $pn(T)$ .  $w$  is called the *root* of  $T$ .

Notice that our algorithm is fully distributed, that it can be executed in an asynchronous environment (we assume that each node knows its neighbors) and that there are as many steps as nodes in the tree.

At each step, the goal of the message sent by a node  $v$  to its father  $v_0$  is to describe, in a synthetic way, the structure of the *subtree*  $T_v$  rooted at  $v$ , that is the connected component of  $T$  minus the edge  $vv_0$ ,  $(T - vv_0)$ , containing  $v$  (see Figure 1).

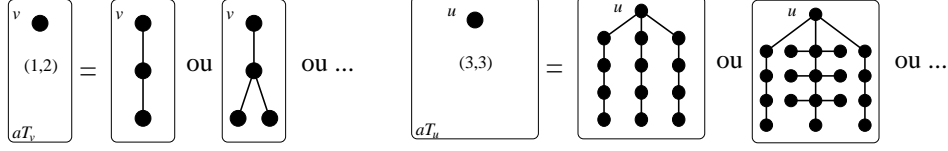


Figure 2: Example of trees whose associated vector are  $\text{vect}(v) = (1, 2)$  and  $\text{vect}(u) = (3, 3)$ .

In fact a message describes a decomposition of  $T_v$  into a set of smaller disjoint trees. The trees of this decomposition are indexed by their roots; we note  $R_v$  the set of roots of the trees of this decomposition. Through the algorithm, given a node  $w$ , a unique tree with root  $w$  will be computed, i.e. if in two different decompositions there is a tree rooted at  $w$ , it will be the same. We call a tree of a decomposition with root  $w$  an *associated-tree* and note it  $aT^w$ .

An associated-tree, and more generally any tree, can be of two types: *stable* or *unstable*. Intuitively, the process number of a stable tree will not be affected if we add a component of same process number whereas the process number of an unstable tree will increase in this case.

**Definition 1** Let  $T$  be a tree with root  $r$ .  $T$  is said *stable* if there is an optimal process strategy such that the last (or equivalently first) node to have an agent is  $r$  or if there is a  $(\leq 2)$ -process strategy finishing with  $r$ . Otherwise  $T$  is *unstable*. The node  $r$  is said *stable* or *unstable* accordingly to  $T$ .

**Remark** We consider a tree of process number one as stable (even if an optimal process strategy finishing at its root needs two agents) for technical reason.

From Definition 1, we give two values to describe if an associated-tree  $aT^w$  rooted at  $w$  is stable or unstable and to give its process number:  $\text{pn}$  its process number, and  $\text{pn}^+$  the minimum number of agents used in a process strategy such that the last (or first) node to have an agent is  $w$ . They together formed the vector associated to  $aT^w$ :  $\text{vect}(w) = (\text{pn}, \text{pn}^+)$ . By extension we associate  $\text{vect}(w)$  to  $w$ . Remark that they are unique for a given associated-tree but several associated-trees can have the same values, also they depend on the root of the associated-tree (see Figure 2). Remark also that to store this vector it is sufficient to store  $(\text{pn}, \text{pn}^+ - \text{pn})$ , which is an integer ( $\text{pn}$ ) and a bit since  $\text{pn} \leq \text{pn}^+ \leq \text{pn} + 1$ .

Back to our algorithm, each associated-tree  $aT^w$  of the decomposition of  $T_v$  will be described by its vector  $\text{vect}(w)$ , and the message sent by a node  $v$  to its father  $v_0$  contains the vector of all associated-trees of the decomposition. However if the decomposition does not verify some specific properties, this information is not sufficient to compute the process number of  $T_v$ . It is why we need the notion of hierarchical decomposition.

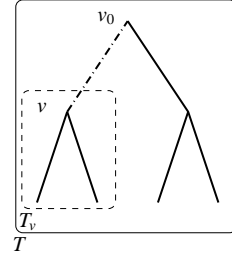


Figure 1: The subtree  $T_v$



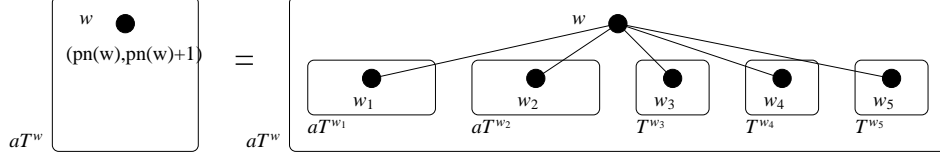


Figure 4: Structure of an unstable associated-tree  $aT^w$ .  $\text{vect}(w_1) = \text{vect}(w_2) = (\text{pn}(w), \text{pn}(w))$  and  $\forall i \in [3, 5], \text{pn}(T^{w_i}) < \text{pn}(w)$ .

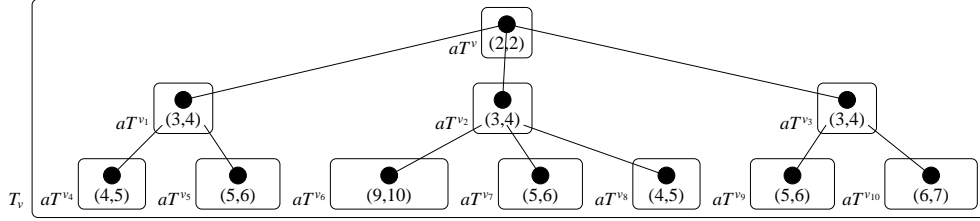


Figure 5: Example of a hierarchical decomposition of a tree  $T_v$  with process number 9.

## 2.1 Hierarchical decomposition

In a *hierarchical decomposition* of  $T_v$ , we impose that an associated-tree  $aT^w$  has a process number higher than the associated-tree  $aT^x$  containing the father of  $w$ , as illustrated in Figure 3. We also impose that a hierarchical decomposition has at most one stable associated-tree and if there is one it has to be minimal according to this order. Finally we impose that all unstable associated-trees satisfies Property 1. Figure 5 gives an example of a hierarchical decomposition of a tree with process number 9.

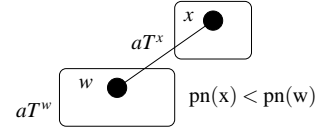


Figure 3:  $aT^x < aT^w$ .

**Property 1 (c.f. Figure 4)** Given a node  $w$ , its associated-tree  $aT^w$ , the subtree  $T_w$  rooted at  $w$ , and  $\Gamma(w) \cap T_w = \{w_1, \dots, w_k\}$ , if  $aT^w$ , and so  $w$ , is unstable it has the following structure:  $w$  has two neighbours  $w_1, w_2 \in \Gamma(w) \cap T_w$  which are stables and such that  $\text{pn}(w_1) = \text{pn}(w_2) = \text{pn}(w)$ . Furthermore  $aT^w$  is formed by its root  $w$ , the two stable associated-trees  $aT^{w_1}$  and  $aT^{w_2}$  and of  $l \leq k-2$  other subtrees  $T^{w_3}, \dots, T^{w_{l+2}}$  whose roots are visited neighbours and whose process number is at most  $\text{pn}(w) - 1$ . Notice that the subtrees  $T^{w_3}, \dots, T^{w_{l+2}}$  are not necessarily the associated-trees  $aT^{w_3}, \dots, aT^{w_{l+2}}$ .

To describe a given hierarchical decomposition, a node  $v$  stores a vector and a table encoding the shape of the associated-trees  $aT^v$ . We will see with Theorem 2 that it is sufficient to compute the process number of  $T_v$ . More precisely  $v$  stores:

- The vector of the stable associated-tree of the decomposition if there is one,  $(-1, -1)$  otherwise;

- A table  $t_v$  of length  $L(t_v) = \max_{w \in R_v}(\text{pn}(w))$  which in cell  $i$ , noted  $t_v[i]$ , contains the number of unstable associated-trees whose vector is  $(i, i+1)$  in the decomposition. (Remember that  $(1,2)$  is considered as stable, hence the first cell always contains 0).

For example in Figure 5,  $v$  and  $v_1$  store respectively:

$$HD(v) : t_v = \begin{bmatrix} 0 & 0 & 3 & 2 & 3 & 1 & 0 & 0 & 1 \end{bmatrix} \text{ and } (\text{pn}(v), \text{pn}^+(v)) = (2, 2)$$

$$HD(v_1) : t_{v_1} = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 \end{bmatrix} \text{ and } (\text{pn}(v_1), \text{pn}^+(v_1)) = (-1, -1)$$

**Lemma 1** Let  $T = (V, E)$  be a tree rooted at  $r$  and  $aT^w$ ,  $r \notin aT^w$ , an unstable associated-tree rooted at  $w \in V$  in a hierarchical decomposition. If  $\text{pn}(aT^w) = p$ ,  $\text{pn}(T) = p$  iff  $\text{pn}(T \setminus aT^w) \leq p - 1$ .

Furthermore if  $\text{pn}(T) = p$ ,  $T$  is unstable.

**Proof** If there is a tree  $aT^x$  in the hierarchical decomposition with  $\text{pn}(aT^x) > p$  then  $\text{pn}(T \setminus aT^w) > p$ . From now on we assume that for all  $aT^x$  of the hierarchical decomposition,  $\text{pn}(aT^x) \leq p$ . Using the properties of a hierarchical decomposition, it implies that  $w$  is the only node through which  $aT^w$  is connected to the rest of  $T$ .

By Property 1,  $aT^w$  is formed by its root  $w$ , two stable subtrees  $T^{w_1}$  and  $T^{w_2}$  with process number  $p$  and some other subtrees with process number less than  $p - 1$ .

If  $T \setminus aT^w$  has process number at least  $p$  then  $w$  is a node with three branches having process number at least  $p$ . Hence, by Theorem 1,  $T$  has process number at least  $p + 1$ .

Otherwise  $\text{pn}(T \setminus aT^w) < p$  and we describe a  $p$ -process strategy. We start by an optimal process strategy the stable associated-tree  $aT^{w_1}$ . It uses  $p$  agents and finishes with  $w_1$  occupied by an agent. Then we place an agent on  $w$  and process  $w_1$ . We continue with an optimal process strategy of  $T^w \setminus aT^{w_2}$ , it uses at most  $p - 1$  extra agents.

Now, since  $\text{pn}(T \setminus aT^w) < p$ , we continue with a  $(p - 1)$ -process strategy of  $T \setminus aT^w$ . We then place an agent on  $w_2$  and process  $w$ . It now only remains to process  $aT^{w_2}$  starting at  $w_2$  which can be done with  $p$  agents by assumption.

$T$  is clearly unstable since it contains an unstable subtree  $aT^w$  with same process number which does not contain the root of  $T$ .  $\square$

**Theorem 2** Given a rooted tree  $T$ , a table  $t$  and a vector  $\text{vect} = (\text{pn}, \text{pn}^+)$ , if there is a hierarchical decomposition of  $T$  described by  $(\text{vect}, t)$ , we can compute  $\text{pn}(T)$ . More precisely:

- $\text{pn}(T) = L(t) \Leftrightarrow \exists i \in [1..L(t)]$  such that  $t[i] = 0$  and  $\forall j \in [i + 1..L(t)] t[j] = 1$ . Furthermore  $T$  is unstable.
- If  $\text{pn}(T) \neq L(t)$  then  $\text{pn}(T) = \max\{\text{pn}, L(t) + 1\}$  and  $T$  is stable.

The Property a) means that if in the table  $t$  of a hierarchical decomposition there is a cell with a 0 followed only by cells full of 1, then the process number of a tree accepting such a hierarchical decomposition has process number  $L(t)$ .

**Proof of Theorem 2** First remark that the process number is at most  $L(t) + 1$ . By induction on  $L(t)$ .

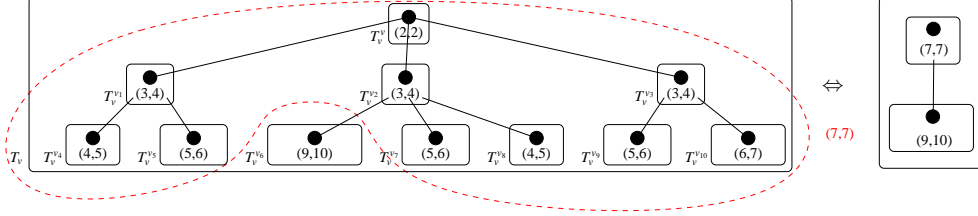


Figure 6: A simpler hierarchical decomposition of the example of Figure 5.

- If  $L(t) = 0$ ,  $T$  is a single node and  $\text{pn}(T) = 0$ . If  $L(t) = 1$ ,  $T$  is a stable tree with vector  $(1,1)$  or  $(1,2)$ . In both case  $\text{pn}(T) = 1$ . If  $L(t) = 2$  and  $t[2] = 0$ ,  $T$  is a stable tree with vector  $(2,2)$  and  $\text{pn}(T) = 2$ . If  $t[i] = 0$  for all  $i \leq L(t)$ ,  $T$  is a stable tree with vector  $(L(t), L(t))$  and  $\text{pn}(T) = L(t)$ .
- When  $L(t) \geq 2$  and  $t[L(t)] = 1$ . We call the associated-tree of the hierarchical decomposition having process number  $L(t)$   $aT^w$  and  $w$  its root. By Lemma 1,  $\text{pn}(T) = L(t) \Leftrightarrow \text{pn}(T \setminus aT^w) \leq L(t) - 1$ .
  - If  $\exists i \in [1..L(t)]$  with  $t[i] = 0$  and  $\forall j \in [i+1..L(t)]$   $t[j] = 1$ , we have  $\text{pn}(T \setminus aT^w) \leq L(t) - 1$ .
    - \* Indeed, either  $t[L(t) - 1] = 1$  and  $\text{pn}(T \setminus aT^w) = L(t) - 1$  by induction, so  $\text{pn}(T) = L(t)$ .
    - \* Or  $t[L(t) - 1] = 0$ . In this case either, we have a table with only 0 and we are at an initialisation case:  $\text{pn}(T \setminus aT^w) = L(t) - 1$  or we can delete this last cell, the length of the table is then  $L(t) - 2$  and we are sure that  $\text{pn}(T \setminus aT^w) \leq L(t) - 1$  by the very first remark of the proof. In both cases we have once again  $\text{pn}(T) = L(t)$ .
  - If in  $t$  there is a cell with a number bigger than one followed by cells full of one until the last cell, then, by induction,  $\text{pn}(T \setminus aT^w) = L(t)$  and hence  $\text{pn}(T) = L(t) + 1$ .
- When  $L(t) \geq 2$  and  $t[L(t)] \geq 2$ , we call one of the associated-tree of process number  $L(t)$   $aT^w$  and  $w$  its root.  $\text{pn}(T \setminus aT^w) \geq L(t)$ , hence, from Lemma 1  $\text{pn}(T) > L(t)$  which means  $\text{pn}(T) = L(t) + 1$  by the very first remark.

$T$  stable or unstable follows from Lemma 1 and the process strategy we described.  $\square$

## 2.2 Minimal hierarchical decomposition

In the example of Figure 5, Theorem 2 directly says it has process number 9. If we now consider this example minus the subtree of vector  $(9,10)$ , then Theorem 2 says it has process number 7 and furthermore that it is stable. Hence, we can get another hierarchical decomposition as shown on Figure 6.

In fact we can generalize this simplification. Given a table  $t$  and an index  $i \leq L(t)$ , we note  $t[1..i]$  the table composed of the  $i$  first cells of  $t$ . For a given hierarchical decomposition described by its vector and its table,  $HD = (vect, t)$ , we call  $HD_i = (vect, t[1..i])$  a *i-restricted hierarchical decomposition*. Notice that if  $HD$  is a hierarchical decomposition of a tree  $T$ , then  $HD_i$  is a hierarchical decomposition of the subtree composed of the associated-trees having process number at most  $i$ .

A last definition, if a tree accepts several hierarchical decompositions, we say they are *equivalent*.

We now describe the simplification of a given hierarchical decomposition  $HD = (vect, t)$  of a tree  $T$ . If there is  $i \leq L(t)$  such that a tree  $T_i$ , whose hierarchical decomposition is described by  $HD_i = (vect, t[1..i])$ , has process number  $i + 1$ , then  $HD$  is equivalent to a simpler hierarchical decomposition  $HD' = ((i + 1, i + 1), t')$ , where  $L(t') = L(t)$ ,  $t'[j] = 0$  for  $j \leq i + 1$ , and  $t'[j] = t[j]$  for  $j > i + 1$ . If no such  $i$  exist, the hierarchical decomposition can not be simplified.

We call a hierarchical decomposition we can not simplify a *minimal hierarchical decomposition*. Our algorithm will compute such decompositions for each subtree  $T_v$ ,  $v \in V$ . Furthermore we have:

**Lemma 2** *Let  $HD = ((pn, pn^+), t)$  be a minimal hierarchical decomposition. For all  $i \in [2..L(t)]$ , we have  $t[i] \in \{0, 1\}$ .*

### 3 Distributed algorithm for the process number

We can now describe precisely algorithm algoHD:

- The algorithm is initialized at the leaves. Each leaf sends the message  $((0, 0), [ ])$  (where  $[ ]$  represents a table of length 0) to its only neighbour which becomes its father.
- A node  $v$ , which has received messages from all its neighbours but one, computes the minimal hierarchical decomposition of  $T_v$  using Algorithm 1. Then it sends the message  $(pn(T_v), pn^+(T_v)), t_v)$  to its last neighbour, its father.
- The last node  $w$  receives a message from all its neighbours, it computes the minimal hierarchical decomposition of  $T_v = T$  and Theorem 2 gives the process number  $pn(T)$ .  $w$  is called the root of  $T$ .

**Remark** It may happen that two adjacent nodes  $v$  and  $w$  receive a message from all their neighbors. It is the case when node  $v$ , after sending its message to its last neighbor  $w$ , receives a message from  $w$ . In this case, both  $v$  and  $w$  are potential candidates to be the root of the tree. There are two possibilities to solve this problem. If each node has a unique identifier (e.g. MAC address) known by its neighbors, then the one of  $v$  and  $w$  with the largest identifier becomes the root, otherwise,  $u$  and  $w$  send each other a random bit, repeat in case of equality, and the 1 win.

**Lemma 3** *Given a tree  $T = (V, E)$ , with  $|V| = n$ , the time complexity of Algorithm 1 is  $O(\log n)$ .*

**Proof** All operations are linear in  $L(t_v)$ , and  $L(t_v) \leq pn(T) \leq \log_3 n$ . □

**Algorithm 1** Computation of the minimal hierarchical decomposition

**Require:**  $v_1, \dots, v_d$  the visited neighbours of  $v$ , and the corresponding minimal hierarchical decompositions  $HD(v_i) = ((pn(v_i), pn^+(v_i)), t_{v_i})$

**Require:**  $t_v^{int}$ , a table such that  $t_v^{int}[i] := t_{v_1}[i] + \dots + t_{v_{d-1}}[i]$ ,  $\forall i \in [2.. \max_{1 \leq j \leq d} L(t_{v_j})]$ .

**Require:**  $M_v := \{v_i; \forall j \in [1..d-1], pn(v_j) \leq pn(v_i)\}$  {all  $v_i$  such that  $pn(v_i)$  is maximum}

**Ensure:**  $vect(v)$  and  $t_v$   
 {computation }

1: Let  $(p_v, p_v^+)$  be the vector of the associated-tree of  $v$

2: **if**  $\forall v_i \in M_v, pn(v_i) < 2$  **then** {Initial cases}

3:  $(p_v, p_v^+) := \begin{cases} (0, 0) & \text{when } \forall v_i \in M_v, pn(v_i) = -1 \\ (1, 1) & \text{when } \forall v_i \in M_v, pn(v_i) = 0 \\ (1, 2) & \text{when } |M_v| = 1 \text{ and } vect(v_i) = (1, 1) \\ (2, 2) & \text{otherwise} \end{cases}$

4: **else** {general cases}

5: **if**  $|M_v| = 2$  **then** { $v$  is unstable}

6:  $(p_v, p_v^+) := (pn(v_i), pn(v_i) + 1)$ , where  $v_i \in M_v$

7: **else** { $v$  is stable}

8: **if**  $|M_v| > 2$  **then** {Theorem 1}

9:  $(p_v, p_v^+) := (pn(v_i) + 1, pn(v_i) + 1)$ , where  $v_i \in M_v$

10: **else**

11:  $(p_v, p_v^+) := (pn(v_i), pn(v_i))$ , where  $v_i \in M_v$

{computation of the table}

12:  $L(t_v) := \max \{L(t_v^{int}), p_v\}$

13:  $t_v := t_v^{int}$

14: **if**  $p_v < p_v^+$  and  $p_v > 1$  **then**

15:  $t_v[p_v] := t_v[p_v] + 1$

16:  $t_v[j] := 0, \forall j \in [2..p_v - 1]$

17:  $(p_v, p_v^+) := (-1, -1)$  {Here,  $(p_v, p_v^+)$  is stable}

18: Let  $k$  be such that  $t_v[k] > 1$  and  $t_v[i] \leq 1, \forall i \in [k+1..L(t_v)]$

19: Let  $k_1$  be such that  $t_v[k_1] = 0$  and  $t_v[i] = 1, \forall i \in [k..k_1 - 1]$

20: **if**  $t_v[p_v] = 0$  **then**

21:  $k_2 := p_v$

22: **else**

23: Let  $k_2$  be such that  $t_v[k_2] = 0$  and  $t_v[i] > 0, \forall i \in [p_v..k_2 - 1]$  {We assume that there exists a virtual cell  $t_v[L(t_v) + 1] = 0$ }

24: **if**  $k, k_1$  and  $k_2$  exist **then**

25:  $t_v[i] := 0, \forall i \in [2.. \max(k_1, k_2)] := 0$

26:  $vect(v) := (\max(k_1, k_2), \max(k_1, k_2))$

27: **else** {the hierarchical decomposition is already minimal}

28:  $vect(v) := (p_v, p_v^+)$

**Lemma 4** *Given a tree  $T = (V, E)$ , with  $|V| = n$ , algo HD computes  $\text{pn}(T)$  in  $n$  steps and overall  $O(n \log n)$  operations.*

**Proof** Each node  $v$  of degree  $d_v$  has to compute  $M_v$  (the set of neighbors  $v_i$  with maximum  $\text{pn}(v_i)$ ) which requires  $O(d_v)$  operations, and  $t_v^{\text{sum}}$  (the sum of all received tables) that is  $O(\sum_1^d L(t_v^{\text{sum}}))$  operations. Finally it applies Algorithm 1. As  $\sum_{v \in V} d_v = 2(n-1)$ , we have  $\sum_{v \in V} (d_v + \log n + \sum_1^d L(t_v^{\text{sum}})) = O(n \log n)$ .  $\square$

**Lemma 5** *Given a tree  $T = (V, E)$ , with  $|V| = n$ , algo HD sends  $n-1$  messages each of size  $\log_3 n + 2$ .*

**Proof** Node  $v$  sends its minimal hierarchical decomposition to its father, that is  $HD_v = (\text{vect}(v), t_v)$ , with  $\text{vect}(v) = (\text{pn}(v), \text{pn}^+(v))$ . From Theorem 1 we know that  $L(t_v) \leq \log_3 n$ , from Lemma 2,  $t_v$  contains only 0 and 1's, hence we need only  $\log_3 n$  bits to transmit  $t_v$ . Furthermore, if  $\text{pn}(v) \geq 1$ ,  $t_v[\text{pn}(v)] = 0$  and  $\forall i \leq \text{pn}(v), t_v[i] = 0$ . Hence we can add an artificial 1 to the cell of  $t_v$  with index  $\text{pn}(v)$  to indicate the value  $\text{pn}(v)$ .

To summarize, we transmit a table  $t$  and two bits  $ab$ .  $ab = 00$  means  $\text{vect}(v) = (-1, -1)$ ,  $ab = 01$  means  $\text{vect}(v) = (0, 0)$ ,  $10$  means  $\text{vect}(v) = (\text{pn}, \text{pn})$  and  $11$  means  $\text{vect}(v) = (\text{pn}, \text{pn} + 1)$ . When  $a = 1$ ,  $\text{pn}$  is the index of the first 1 in the transmitted table and  $t_v$  is the transmitted table minus this 1. When  $a = 0$ ,  $t_v$  is the transmitted table  $t$ . It is clear that in this coding, each message has size  $\log_3 n + 2$ .  $\square$

## 4 Dynamic and incremental algorithms

In this section, we propose a dynamic algorithm that allows to compute the process number of the tree resulting of the addition of an edge between two trees. It also allows to delete any edge. To do this efficiently, it uses one of the main advantage of the hierarchical decomposition: the possibility to change the root of the tree without additional information (Lemma 6). From that we design an incremental algorithm that computes the process number of a tree.

If we want to join two trees with an edge between their roots then it is easy to see that Algorithm 1 will do it. However if we do not join them through the root, a preprocessing to change the root of the trees needs to be done. In next Section we propose one. To apply this algorithm, each node needs to store the information received from each of its neighbors and a table which is the sum of the received tables:  $\forall v_i \in \Gamma(v) \cap T_v : \text{vect}_{v_i}, t_{v_i}$  and  $t_v^{\text{sum}}$ . Recall that  $t_v^{\text{sum}}$  is defined as  $t_v^{\text{sum}}[j] = \sum_{v_i \in \Gamma(v) \cap T_v} t_{v_i}[j]$  in the algorithm.

For a given tree  $T$ , we note  $D(T)$  or  $D$  if there is no ambiguity the *diameter* of  $T$ .

We describe now three functions we will use in the dynamic version of our algorithm.

### 4.1 Functions for updating the process number

**Lemma 6 (Change of the root)** *Given a tree  $T = (V, E)$  rooted at  $r_1 \in V$  of diameter  $D$ , and its hierarchical decomposition, we can choose a new root  $r_2 \in V$  and update accordingly the hierarchical decomposition in  $O(D)$  steps of time complexity  $O(\log n)$  each, using  $O(D)$  messages of size  $\log n + 3$ .*

**Proof** We describe an algorithm to change the root from  $r_1$  to  $r_2$ :

First,  $r_2$  sends a message to  $r_1$  through the unique path between  $r_1$  and  $r_2$ ,  $r_2 = u_0, u_1, u_2, \dots, u_k = r_1$ , to notify the change. Then,  $r_1$  computes its hierarchical decomposition, considering that  $u_{k-1}$  is its father. We assume that each node  $v$  stores the information received from its neighbours and  $t_v^{sum}$ .  $r_1$  applies Algorithm 1 using all vectors stored but  $vec_{u_{k-1}}$  and  $t_v^{sum} - t_{v_{k-1}}$ . Then it sends a message to  $u_{k-1}$ .

After,  $u_{k-1}$  computes its hierarchical decomposition, considering that  $u_{k-2}$  is its father, and sends a message to  $u_{k-2}$ . We repeat until  $r_2$  receives a message from  $u_1$ . Finally,  $r_2$  computes the process number of  $T$  and becomes the new root. We have a new hierarchical decomposition.

In this algorithm,  $u_i$  subtracts the table  $t_{u_{i-1}}$  from  $t_{u_i}^{sum}$ , and later adds  $t_{u_{i+1}}$ , computes  $M_{u_i}$  and finally applies Algorithm 1. Clearly, all computation requires  $O(\log n)$  operations. The messages need one more bit than in the previous algorithm to indicate whether a table has to be added or subtracted.  $\square$

**Lemma 7 (Addition of an edge)** *Given two trees  $T_{r_1} = (V_1, E_1)$  and  $T_{r_2} = (V_2, E_2)$  respectively rooted at  $r_1$  and  $r_2$ , we can add the edge  $(w_1, w_2)$ ,  $w_1 \in V_1$  and  $w_2 \in V_2$  and compute the process number of  $T = (V_1 \cup V_2, E_1 \cup E_2 \cup (w_1, w_2))$ , in at most  $D$  steps.*

**Proof** First we change the roots of  $T_{r_1}$  and  $T_{r_2}$  respectively to  $w_1$  and  $w_2$  using Lemma 6. Then,  $w_1$  and  $w_2$  decide of a root (see Remark 3) which finally computes the process number of  $T$ .  $\square$

**Lemma 8 (Deletion of an edge)** *Given a tree  $T = (V, E)$  rooted at  $r$  and an edge  $(w_1, w_2) \in E$ , after the deletion of edge  $(w_1, w_2)$ , we can compute the process number of the two disconnected trees in at most  $D$  steps.*

**Proof** W.l.o.g. we may assume that  $w_2$  is the father of  $w_1$ . Let  $T_{w_1}$  be the subtree rooted at  $w_1$  and  $T \setminus T_{w_1}$  the tree rooted at  $r$ . Remark that it includes  $w_2$ . The process number of  $T_{w_1}$  is deduced from the previously computed hierarchical decomposition. Now, to compute the process number of  $T \setminus T_{w_1}$ , we apply the change root algorithm and node  $w_2$  becomes the new root of  $T \setminus T_{w_1}$ .  $\square$

## 4.2 Incremental algorithm

From Lemma 7, we obtain an incremental algorithm (IncHD) that, starting from a forest of  $n$  disconnected vertices with hierarchical decomposition  $((0, 0, ) [ ])$ , add tree edges one by one in any order and updates the process number of each connected component. At the end, we obtain the process number of  $T$ .

This algorithm is difficult to analyze in average, but the best and worst cases are straightforward:

- Worst case:  $T$  consists of two subtrees of size  $n/3$  and process number  $\log_3(n/3)$  linked via a path of length  $n/3$ . Edges are inserted alternatively in each opposite subtrees. Thus IncHD requires  $O(n^2)$  steps and messages, and overall  $O(n^2 \log n)$  operations

- Best case: edges are inserted in the order induced by `algoHD` (inverse order of a breadth first search). `IncHD` needs  $O(n)$  messages and an overall of  $O(n \log n)$  operations.

Actually, the overall number of messages is  $O(nD)$  and the number of operations is  $O(nD \text{pn}(T))$ . They both strongly dependent on the order of insertion of the edges. Thus an interesting question is to determined the average number of messages and operations.

## 5 Improvements and extensions

**Reducing the amount of transmitted information** In our algorithms, it is possible to reduce the size of some messages and so the overall amount of information transmitted during the algorithm. For example, instead of transmitting  $\log n$  bits for  $t$ , we may transmit only  $L(t)$  bits plus the value  $L(t)$  on  $\log \log n$  bits. Overall we will exchange less than  $n(\text{pn}(T) + \log_2 \log_3 n + 2 + \epsilon)$  bits, where  $\epsilon = 1$  for the dynamic version of the algorithm (`IncHD`). Further improvements are possible with respect to the following lemma.

**Lemma 9** *Assuming that when an edge is added at vertex  $v$ ,  $v$  asks its neighbours information once and simultaneously, any dynamic algorithm satisfying this assumption induces a transmission of at least  $\frac{k-1}{k}n(\text{pn}(T) - 2)$  bits for any  $k \in \mathbb{N}$  and value of  $\text{pn}(T) \leq \log_3(n/k)$  in some trees  $T$ .*

**Proof** Suppose that we are given a dynamic algorithm such that when an edge is added at vertex  $v$ ,  $v$  asks its neighbours information once and simultaneously, and let  $k > 1$  be an integer. We consider a tree made of a path  $u-v$  of length  $\frac{k-1}{k}n$  with a tree  $T'$  at  $u$ . One of the messages received by  $v$  gives information about  $T'$ . If for all tree  $T'$  with process number  $p$ , the algorithm uses less than  $p - 2$  bits to encode this message, and since there is more than  $2^{p-2}$  hierarchical decompositions corresponding to a tree with process number  $p$ , there exists two trees  $T'_1$  and  $T'_2$  with different minimal hierarchical decompositions but which are encoded in the same way. We note  $T_1$  when  $T' = T'_1$  and  $T_2$  when  $T' = T'_2$ . Then, it exists a tree  $T''$  such that if we join it to (w.l.o.g)  $T_1$  at  $v$ , the process number of  $T_1$  increases by one whereas if we join  $T''$  to  $T_2$  at  $v$ , the process number of  $T_2$  does not increase.

Hence, there is a tree  $T'$  for which the algorithm encodes the information transmitted to  $v$  on at least  $p - 2$  bits. For this  $T'$  in our construction of  $T$ , the information received by  $v$  comes from  $u$  and hence it has transited through  $\frac{k-1}{k}n$  nodes. Therefore, the total of transmitted bits is at least  $\frac{k-1}{k}n(p - 2)$ .  $\square$

**Corollary 1** *Assuming that when an edge is added at vertex  $v$ ,  $v$  asks its neighbours information once and simultaneously, any dynamic algorithm induces a transmission of at least  $\frac{k-1}{k}n(\log_3 n)$  bits in some large enough trees, for any  $k \in \mathbb{N}$ .*

**Proof** Let  $k \in \mathbb{N}$ . By the previous Lemma for  $k + 1$ , there is a tree  $T$  with process number  $\log_3(n/(k + 1))$  which induces a transmission of at least  $\frac{k}{k+1}n(\log_3(n/(k + 1)) - 2)$  bits, and this larger than  $\frac{k-1}{k}n(\log_3 n)$  when  $\log n > k^2(\log_3(k + 1) + 2)$ .  $\square$



**Reducing the number of operations** It makes no doubt that the worst case complexity of `IncHD` and more specifically of Lemma 7 can be seriously improved. In particular, instead of changing the roots of both trees, we may change only  $r_1$  to  $w_1$ , then transmit information in the direction of  $r_2$ , and eventually stop the transmissions before  $r_2$  if the minimal hierarchical decomposition of some node remains unchanged.

It is also interesting to notice that using arguments similar to [5], we can get a centralized algorithm using a linear number of operations.

**Trees and forests of unknown size** If the size  $n$  of the tree is unknown, a node encodes each bit of the transmitted table  $t$  on 2 bits, that is 00 for 0 and 01 for 1. It allows to use 11 to code the end of the table and hence to know its length. Thus the receiver may decode the information without knowing  $n$ . In this coding the table requires  $2L(t) + 2$  bits and the transmission requires  $2L(t) + 4 + \epsilon$  bits, where  $\epsilon = 1$  for `IncHD` and 0 for `algoHD`. Remember that  $L(t) \leq \text{pn}(T)$ .

**Computing other parameters** Our algorithms can be adapted to compute the node search number or the pathwidth of any tree with the same time complexity and transmission of information. For that, it is sufficient to change the values of the initial cases (lines 1 and 1) in Algorithm 1.

For the node search number we would use the initial cases of the left of Figure 5. Notice that in this case we do not use the vector  $(1, 2)$ .

$$\begin{array}{lcl} \text{if } \forall v_i \in M_v, \text{pn}(v_i) < 2 \text{ then} & & \text{if } \forall v_i \in \Gamma(v), \text{pn}^+(v_i) < 2 \text{ then} \\ (p_v, p_v^+) & := & \\ \left\{ \begin{array}{ll} (1, 1) & \text{when } \forall v_i \in M_v, \text{pn}(v_i) = -1 \\ (2, 2) & \text{otherwise} \end{array} \right. & & (p_v, p_v^+) := \left\{ \begin{array}{ll} (0, 0) & \text{when } |M_v| = 0 \\ (1, 1) & \text{when } |M_v| = 1 \\ (1, 2) & \text{when } |M_v| = 2 \\ (2, 2) & \text{otherwise} \end{array} \right. \end{array}$$

Figure 7: Initial cases for node search number (left) and edge search number (right).

For the edge search number of a tree, we can prove that  $\text{ns}(T) - 1 \leq \text{es}(T) \leq \text{ns}(T)$ , whereas on a general graph we only have  $\text{ns}(T) - 1 \leq \text{es}(T) \leq \text{ns}(T) + 1$ . To adapt Algorithm 1 for the edge search number, we would use the initial cases of the right of Figure 5 plus the extra rule that all received vectors  $(1, 2)$  are interpreted as if they were vectors  $(2, 2)$ . Also, if all received vectors verifies  $\text{pn}^+(v_i) < 2$ ,  $M_v$  is the set of all received vectors different from  $(-1, -1)$ . Notice that it gives the first algorithm to compute the edge search number of trees.

Algorithm `algoHD` has been implemented for the process number, the node search number and the edge search number, as well as corresponding search strategies [1].

**About the difference of the parameters** Finally, the following lemma characterizes the trees for which the process number (resp. edge search number) equals the pathwidth.

**Lemma 10** *Given a tree  $T$ ,  $\text{pn}(T) = \text{pw}(T) + 1 = p + 1$  (resp.  $\text{pn}(T) = \text{es}(T) + 1 = p + 1$ ) iff there is a node  $v$  such that any components of  $T - \{v\}$  has pathwidth at most  $p$  and there is at least three components with process number (resp. edge search number)  $p$  of which at most two have pathwidth  $p$ .*

This lemma means that the difference between, e.g., the process number and the pathwidth comes from the difference on trees with smaller parameter and ultimately from trees with those parameters equal to 1 or 2.

To give such characterisations for more general classes of graphs remains a challenging problem.

## Acknowledgments

We would like to thanks Nicolas Nisse and Hervé Rivano for fruitfull discussions on this problem.

## References

- [1] <http://www-sop.inria.fr/members/Dorian.Mazauric/Capture/index.php.htm>.
- [2] D. Coudert, S. Perennes, Q.-C. Pham, and J.-S. Sereni. Rerouting requests in wdm networks. In *AlgoTel'05*, pages 17–20, Presqu'île de Giens, France, mai 2005.
- [3] D. Coudert and J.-S. Sereni. Characterization of graphs and digraphs with small process number. Research Report 6285, INRIA, September 2007.
- [4] J. Díaz, J. Petit, and M. Serna. A survey on graph layout problems. *ACM Computing Surveys*, 34(3):313–356, 2002.
- [5] J.A. Ellis, I.H. Sudborough, and J.S. Turner. The vertex separation and search number of a graph. *Information and Computation*, 113(1):50–79, 1994.
- [6] F. V. Fomin and D. Thilikos. An annotated bibliography on guaranteed graph searching. *Theoretical Computer Science, Special Issue on Graph Searching*, 2008, to appear.
- [7] N. G. Kinnersley. The vertex separation number of a graph equals its pathwidth. *Inform. Process. Lett.*, 42(6):345–350, 1992.
- [8] M. Kirousis and C.H. Papadimitriou. Searching and pebbling. *Theor. Comput. Sci.*, 47(2):205–218, 1986.
- [9] N. Megiddo, S. L. Hakimi, M. R. Garey, D. S. Johnson, and C. H. Papadimitriou. The complexity of searching a graph. *J. Assoc. Comput. Mach.*, 35(1):18–44, 1988.
- [10] T. D. Parsons. Pursuit-evasion in a graph. In *Theory and applications of graphs*, pages 426–441. Lecture Notes in Math., Vol. 642. Springer, Berlin, 1978.
- [11] N. Robertson and P. D. Seymour. Graph minors. I. Excluding a forest. *J. Combin. Theory Ser. B*, 35(1):39–61, 1983.
- [12] P. Scheffler. A linear algorithm for the pathwidth of trees. In R. Henn R. Bodendiek, editor, *Topics in Combinatorics and Graph Theory*, pages 613–620. Physica-Verlag Heidelberg, 1990.

- [13] K. Skodinis. Construction of linear tree-layouts which are optimal with respect to vertex separation in linear time. *J. Algorithms*, 47(1):40–59, 2003.



---

Unité de recherche INRIA Sophia Antipolis  
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes  
4, rue Jacques Monod - 91893 ORSAY Cedex (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique  
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

---

Éditeur  
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)  
<http://www.inria.fr>  
ISSN 0249-6399