Computing and updating the process number in trees

David Coudert, Florian Huc, Dorian Mazauric

Computing and updating the process number in trees*

David Coudert, Florian Huc, Dorian Mazauric
Mascotte, INRIA, I3S, CNRS, University of Nice Sophia, France
{firstname.lastname@sophia.inria.fr}

Abstract

The process number is the minimum number of requests that have to be simultaneously disturbed during a routing reconfiguration phase of a connection oriented network. From a graph theory point of view, it is similar to the node search number, and thus to the pathwidth, however they are not always equal. In general determining these parameters is NP-complete.

We present a distributed algorithm to compute these parameters and the edge search number, in trees. It can be executed in an asynchronous environment, requires n steps, an overall computation time of $O(n \log n)$, and n messages of size $\log_3 n + 2$. Then, we propose a distributed algorithm to update these parameters on each component of a forest after addition or deletion of any tree 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 new connected component.

Keywords: pathwidth, process number, distributed algorithm.

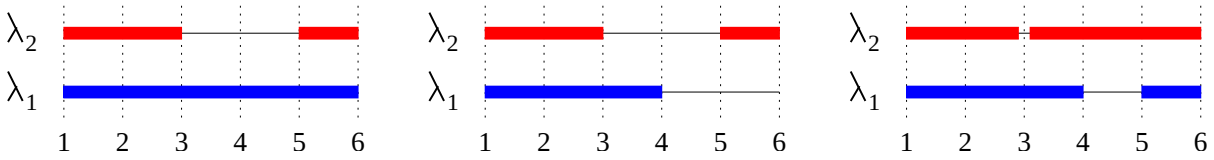
1 Introduction

Treewidth and pathwidth have been introduced by Robertson and Seymour [RS83] as part of the graph minor project. Those parameters are very important since many problems can be solved in polynomial time for graphs with bounded treewidth or pathwidth. 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 [EST94, Sch90, Sko03], but so far no distributed algorithm exists.

The algorithmic counter part of the notion of pathwidth (denoted pw) is the cops and robber game [KP86, FT08, DPS02]. 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 node search number (denoted ns). Other graph invariants closely related to the notion of pathwidth have been proposed such as the process number [CPPS05, CS07] (denoted pn) and the edge search number [MHG+88] (denoted es). Their determination is in general NP-complete [KP86].

In this paper, we describe in Sec. 2 the motivation of the problem from a network reconfiguration problem point of view. In Sec. 3, we propose a fully distributed algorithm to compute the process number of trees, which can be executed in an asynchronous environment. 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.

*This work was partially funded by the European projects ist fet Aeolus and COST 293 Graal, ARC CARMA, ANR JC OSERA, CRC CORSO and Région PACA.



(a) Initial routing of (1,6), (1,3) and (5,6)

(b) Removal of (1,6) and addition of (1,4)

(c) Solution with (1,3), (1,4), (3,6) and (5,6)

Figure 1: Starting from the routing of Fig. 1(a), the removal of request (1,6) and addition of request (1,4) gives the routing of Fig. 1(b). Request (3,6) can not be added in Fig. 1(b), although the routing of Fig. 1(c) is possible.

2 Motivation and Modeling

The process number of a (di)graph has been introduced to model a routing reconfiguration problem in connection oriented networks such as WDM, MPLS or wireless backbone networks [CPPS05, CS07]. In such networks, and starting from an optimal routing of a set requests, the routing of a new connection request can be done greedily using available resources (e.g. capacity, wavelengths) thus avoiding to reroute existing connections. Some resources might also be released after the termination of some requests. In fine, such traffic variations may lead to a poor usage of resources with eventual rejection of new connections. For example, in Fig. 1 where the network is a 6 nodes path with two wavelengths, a new connection request from 3 to 6 would be rejected in Fig. 1(b) although the routing of Fig. 1(c) is possible. To optimize the number of granted requests, the routing has to be reconfigured regularly.

In this context, routing reconfiguration problem consists in going from a routing, R_1, to another, R_2, by switching requests one by one from the original to the destination route. This yield to a scheduling problem. Indeed, resources assigned to request r in R_2 might be used by some request r' in R_1 might, thus request r' has to be rerouted before r. We represent these constraints by a digraph $D = (V, A)$ in which each node corresponds to a request, and there is an arc from vertex u to vertex v if v must be rerouted before u. When the digraph D is acyclic, the scheduling is straightforward, but in general, it contains cycles. To break them, some requests have to be temporarily interrupted, thus removing incident arcs in D, and so, the optimization problem is to find a scheduling minimizing the number of requests simultaneously interrupted. When the digraph is symmetric, the problem can be solved on the underlying undirected graph G, and we will restrict our study to this case in the following.

As for the pathwidth, our problem can be expressed as a cops and robber game. An interruption is represented by placing an agent on the corresponding node in G, a node is said *processed* when the corresponding request has been rerouted, and we call a *process strategy* a series of the three following actions allowing to reroute all requests with respect to the constraints represented by the graph.

(1) put an agent on a node (*interrupt a connection*).

(2) remove an agent from a node if all its neighbors are either processed or occupied by an agent (*release a connection to its final route when destination resources are available*). The node is now processed (*connection has been rerouted*).

(3) process a node if all its neighbors are occupied by an agent.

A *p-process strategy* is a strategy which process the graph using p agents and the *process number*, pn(G), is the smallest p such that a p-process strategy exists. For example, a star has process

number 1, a path of more than 4 nodes has process number 2, a cycle of size 5 or more has process number 3, and a $n \times n$ grid, $n \geq 3$, has process number $n + 1$. Moreover, it has been proved in [CPPS05, CS07] that $\text{pw}(G) \leq \text{pn(G)} \leq \text{pw(G)} + 1$, where $\text{pw}(G)$ is the *pathwidth* of G [RS83], and that determining the process number is in general NP-complete.

The node search number [KP86], $\text{ns}(G)$, can be defined similarly except that we only use rules (1) and (2). It was proved by Ellis *et al.* [EST94] that $\text{ns}(G) = \text{pw}(G) + 1$, and by Kinnersley [Kin92] that $\text{pw}(G) = \text{vs}(G)$, where $\text{vs}(G)$ is the *vertex separation* of G. Those results show that vertex separation, node search number and pathwidth are equivalent, but so far it is not known when equivalence also holds with the process number.

3 Distributed Algorithms

We propose an algorithm, `algoHD`, to compute the process number of a tree with an overall of $O(n \log n)$ operations. The principle of `algoHD` is to perform a hierarchical decomposition of the tree. Each node u of degree $d(u)$ collects a compact view of the subtree rooted at each of its sons ($d(u) - 1$ neighbors), computes a compact view of the subtree it forms and sends it to its father (last neighbor), thus constructing a hierarchical decomposition. The algorithm is initialized at the leaves, and the node receiving messages from all its neighbors (the root) concludes on the process number of the tree. Notice that our algorithm is fully distributed and that it can be executed in an asynchronous environment assuming that each node knows its neighbors.

The message sent by a node v to its father v_0 describes the structure of the *subtree T_v rooted at v*, that is the connected component of T minus the edge vv_0 containing v. More precisely, the message describes a decomposition of T_v into a set of smaller disjoint trees, each of them being indexed by its root. See [CHM08] for more details.

Lemma 1 *Given a n-nodes tree T, `algoHD` computes* pn(T) *in n steps and overall $O(n \log n)$ operations, sending n messages each of size* $\log_3 n + 2$.

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 it 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 2 *Given two trees $T_i = (V_i, E_i)$ rooted at $r_i \in V_i$, $i = 1, 2$, its hierarchical decompositions, qnd $r'_i \in V_i$, we can compute the hierarchical decomposition of $T = (V_1 \cup V_2, E_1 \cup E_2 \cap (r'_1, r'_2))$, and so compute its process number in $O(D)$ steps of time complexity $O(\log n)$ each, using $O(D)$ messages of size* $\log n + 3$ *(D is the diameter of T).*

The best and worst cases of the incremental algorithm (`IncHD`) are:

- 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 an overall of $O(n^2 \log n)$ operations.
- Best case: edges are inserted in the order induced by `algoHD` (inverse order of a breadth first search). `IncHD` needs an overall of $O(n \log n)$ operations.

4 Conclusion

In this paper we have proposed a distributed algorithm to compute the process number of a tree, as well as the node and edge search numbers, changing only the values of the initial cases of our algorithm. Then, we have proposed a dynamic algorithm to update these invariants after addition or deletion of any tree edge. Finally we have adapted the algorithm to compute the process number of a tree if its size is unknown and we have characterized the trees for which the process number (resp. edge search number) equals the pathwidth [CHM08]. A challenging task is to characterize other classes of graphs where equality holds or to prove it is NP-hard to decide it in the general case.

References

[CHM08] D. Coudert, F. Huc, and D. Mazauric. A distributed algorithm for computing and updating the process number of a forest. Research Report 6560, INRIA, 06 2008.

[CPPS05] 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.

[CS07] D. Coudert and J-S. Sereni. Characterization of graphs and digraphs with small process number. Research Report 6285, INRIA, September 2007.

[DPS02] J. Díaz, J. Petit, and M. Serna. A survey on graph layout problems. *ACM Computing Surveys*, 34(3):313–356, 2002.

[EST94] 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.

[FT08] F. Fomin and D. Thilikos. An annotated bibliography on guaranteed graph searching. *Theor. Comput. Sci.*, 399(3):236–245, 2008.

[Kin92] N. G. Kinnersley. The vertex separation number of a graph equals its pathwidth. *Information Processing Letters*, 42(6):345–350, 1992.

[KP86] M. Kirousis and C.H. Papadimitriou. Searching and pebbling. *Theoretical Computer Science*, 47(2):205–218, 1986.

[MHG+88] 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.

[RS83] N. Robertson and P. D. Seymour. Graph minors. I. Excluding a forest. *J. Combin. Theory Ser. B*, 35(1):39–61, 1983.

[Sch90] 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.

[Sko03] K. Skodinis. Construction of linear tree-layouts which are optimal with respect to vertex separation in linear time. *Journal of Algorithms*, 47(1):40–59, 2003.