Characterization of graphs and digraphs with small process number

David Coudert, Jean-Sébastien Sereni

Characterization of graphs and digraphs with small process number

David Coudert — Jean-Sébastien Sereni

N° 6285 — version 2

version initiale Septembre 2007 — version révisée Janvier 2008

Thème COM

ISSN 0249-6399 ISRN INRIA/RR--6285--FR+ENG

Characterization of graphs and digraphs with small process number

David Coudert[*], Jean-Sébastien Sereni[†]

Thème COM — Systèmes communicants
Projets Mascotte

Rapport de recherche n° 6285 — version 2 — version initiale Septembre 2007 — version révisée Janvier 2008 — 26 pages

Abstract: The process number of a digraph has been introduced as a tool to study rerouting issues in WDM networks. We consider the recognition and the characterization of (di)graphs with process number at most two.

Key-words: Rerouting, process number, vertex separation, pathwidth.

This work was partially funded by the ANR JC OSERA, and by European projects IST FET AEOLUS and COST 293 GRAAL.

[*] MASCOTTE, INRIA – I3S(CNRS/UNSA), Sophia Antipolis, France. david.coudert@sophia.inria.fr
[†] Institute for Theoretical Computer Science (ITI) and Department of Applied Mathematics (KAM) Charles University Prague, Czech Republic. sereni@kam.mff.cuni.cz

Charactérisation de graphes et digraphes ayant un petit process number

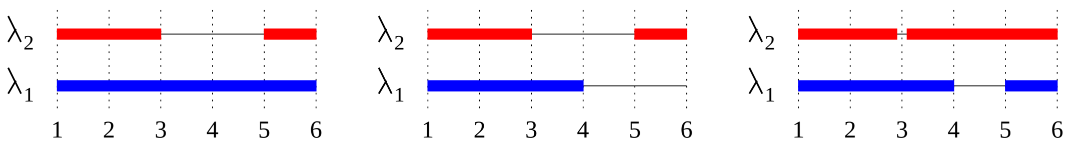
Résumé : Le *process number* d'un graphe orienté a été introduit pour étudier des problèmes de reroutage dans les réseaux WDM. Nous nous intéressons ici à la reconnaissance et à la caractérisation des graphes et graphes orientés ayant un process number au plus 2.

Mots-clés : Reroutage, process number, vertex separation, largeur de chemins

1 Introduction

In a Wavelength Division Multiplexing (WDM) network, each connection request — called *lightpath* in this context — is assigned a route in the network and a wavelength, under the constraint that two lightpaths sharing a link must have different wavelengths.

Usually, when connection requests are added or removed from a WDM network, the routing of older connections is not modified. Hence, it is likely that after some additions and removals the overall use of resources is far from optimal. So a new request may be rejected even if it could be added up to a whole rerouting of older requests. This is the case in the example of Fig. 1, where the WDM network consists of a path of order 6 with two wavelengths λ_1 and λ_2. Initially (Fig. 1(a)), request (1,6) is routed on λ_1, and requests (1,3) and (5,6) are routed on λ_2. Then request (1,6) is removed and a new request, (1,4), is routed on λ_1. Note that this is the only possibility to route this request without doing any rerouting (Fig. 1(b)). In the depicted situation, a new request (3,6) has to be rejected, although the routing of Fig. 1(c) would allow to satisfy all requests. Thus, operators have to reorganize regularly the routing of all requests so as to make better use of the resources. However, they usually want to also ensure a continuous service, i.e. once a request has been accepted and routed, it is not possible to stop its routing, even for a short time. So the service offered by the operator is never lost. We are interested in the problem of going from a routing to another without loss of services.



(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.

Given a WDM network, a set of connection requests I and two different routings for it in the network, R_1 and R_2, we want to switch from routing R_1 to routing R_2. For every request u, we let $R_i(u)$ be its routing in R_i, for $i \in \{1, 2\}$. Consider two requests u and v. If $R_2(u) \cap R_1(v) \neq \emptyset$, i.e. the routing of request u in R_2 uses resources already used by the routing of request v in R_1, then the request v has to be rerouted before we can reroute request u. A request might be switched to an intermediate route, that uses available resources. For instance, the operator may reserve a dedicated wavelength in the network for temporary routes. We assume that each request cannot be switched to more than one temporary route, that is the next routing of a request routed on a temporary route has to be its final routing. When a request that was previously switched to a temporary route reaches its final routing, then the freed resources can be used again, for another request. While independent switching of requests can be made simultaneously, we consider, for matter of exposition, that only one request is switched per unit of time.

We model the problem as follows: we construct a directed graph $D = (V, A)$, where each vertex u corresponds to one request, and there is an arc from vertex u to vertex v if and only if $R_2(u) \cap R_1(v) \neq \emptyset$. A vertex is said to be *processed* as soon as its corresponding request has been rerouted. We introduce the notion of Temporary Memory Unit (TMU): routing the request u on an intermediate route corresponds to putting the vertex u in a TMU. Therefore, a vertex can be processed if and only if all its outneighbors are either processed or in TMU. Note that a vertex without any outneighbor can be processed at any time. There are two basic operations: process a vertex according to the preceding rule; put a vertex in TMU. Fig. 2 shows the processing steps of a graph using one TMU.

Observe that once placed in a TMU, a vertex cannot recover its original state: it has to be processed. Nevertheless, it can occupy its TMU as long as desired. Processing a vertex which occupies a TMU frees the TMU, so that it can immediately be used by another vertex. The digraph is said to be *processed* when all its vertices have been processed. The problem is hence to find a suitable order to process all the vertices. If we do not want to use any TMU, then such a vertex ordering exists if and only if the digraph is acyclic; and in this case a processing order can be found in linear time. On the contrary, if we can use an arbitrary large number of TMUs, then we can first put all vertices in TMU and then process them in any order. We aim at minimizing the number of TMUs simultaneously in use. The *process number* $p(D)$ of a digraph D is the minimum number of TMUs for which there exits a process strategy for D. Notice that the process number is upper bounded by the *minimum forward vertex set number*, that is the smallest number of vertices which intersect all directed cycles. A process strategy that uses p (at most p, at least p, respectively) TMUs is called a p-process strategy (($\leq p$)-process strategy, ($\geq p$)-process strategy, respectively).

Observe that adding loops to a digraph D increases the process number by at most one, and it is straightforward to construct a loopless digraph D' such that $p(D) = p(D')$. Hence, unless stated, we consider in the sequel loopless digraphs. When D is symmetric, we work for convenience on the underlying undirected graph $G = (V, E)$.

An important invariant for digraphs and graphs is the notion of vertex separation. Let $D = (V, A)$ be a digraph and X a subset of its vertices. The *outneighborhood* of X in D is

$$N^+(X) := \{v \in V \setminus X : \text{ there exists } u \in X \text{ such that } (u, v) \in A\}.$$

A *layout* L of D is an ordering of the vertices, i.e. a one-to-one correspondence between V and $\{1, 2, \ldots, |V|\}$. The *vertex separation of* (D, L) is the maximum, over all indices $i \in \{1, 2, \ldots, |V|\}$, of the size of the outneighborhood of $\{L^{-1}(1), L^{-1}(2), \ldots, L^{-1}(i)\}$. The *vertex separation* $\mathrm{vs}(D)$ *of* D is the minimum, over all orderings L, of the vertex separation of (D, L). Note that this notion naturally extends to undirected graphs. In this case, Kinnersley [9] proved that the vertex separation of any undirected graph equals its pathwidth. The pathwidth is an important invariant of graphs which was introduced by Robertson and Seymour [11].

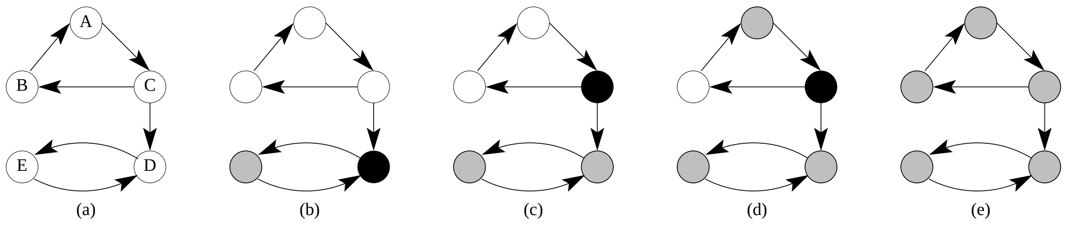


Figure 2: Processing of a graph: processed vertices are in grey and vertices in TMU are in black. In (a), every vertex has at least one outneighbor in the initial state, so we must put a vertex in TMU. In (b), the vertex D has been put in TMU, which allows to process the vertex E. To reach the state (c), put C in TMU, which allows to process D since all its outneighbors are either processed or in TMU.

The following result establishes a close link between the vertex separation and the process number of a digraph. It was first proved by Coudert *et al.* [4], but we recall the proof here for completeness.

Proposition 1. *For every digraph D,* $\mathrm{vs}(D) \leq p(D) \leq \mathrm{vs}(D) + 1$.

Proof. Consider a p-process strategy for D, and let L be the order in which the vertices are processed. Observe that if the strategy is stopped just after the i^{th} vertex has been processed, then any non-processed vertex having a processed inneighbor must be in TMU. As this is true for every $i \in \{1, 2, \ldots, |V| - 1\}$, this exactly means that the vertex separation of (D, L) is p, so $\mathrm{vs}(D) \leq p(D)$.
Let L be an ordering of the vertices of D, and let vs be the vertex separation of (D, L). We consider the process strategy for D that consists of processing the vertices in the increasing order induced by L. At any time, let P be the set of processed vertices and let M be the set of vertices in TMU. At each step, we ensure that $M := N_D^+(P)$.

The first vertex can be processed by putting its at most vs neighbors in TMU. Suppose that $i \geq 1$ vertices have been processed, and let v be the next vertex to be processed. If $v \notin M$, then as the vertex separation of (D, L) is vs we infer that $|M \cup (N^+(v) \setminus P)| \leq \mathrm{vs}$, so we can put all the outneighbors of v that are not in $M \cup P$ in TMU and process v. This uses at most vs TMUs simultaneously. If $v \in M$, then $|M \setminus \{v\} \cup (N^+(v) \setminus P)| \leq \mathrm{vs}$, so putting all the outneighbors of v not in $M \cup P$ in TMU uses at most, and possibly, $\mathrm{vs} + 1$ TMUs simultaneously. Hence, $p(D) \leq \mathrm{vs} + 1$. □

As determining the vertex separation of an arbitrary graph is APX [6], the preceding result shows that the process number problem also is.

The next proposition characterizes the optimal process-strategies for digraphs whose process number is different from their vertex separation.

Proposition 2. *For any digraph D, there exists a $p(D)$-strategy such that each vertex is in* TMU *before being processed if and only if $p(D) = \mathrm{vs}(D) + 1$.*

Proof. Suppose that the digraph D has a $p(D)$-process strategy such that each vertex is put in TMU before being processed. Let $v_1, v_2, \ldots, v_n$ be an enumeration of the vertices of D in the order in which they are processed. For each $i \in \{1, 2, \ldots, n\}$, we set $X_i := \{v_1, v_2, \ldots, v_i\}$. Stop the strategy just before the vertex v_i is processed. All the outneighbors of the vertices of X_i must be in TMU, and so is also v_i. Therefore, $|N^+(X_i)| \leq p(D) - 1$ for each $i \in \{1, 2, \ldots, n\}$, and hence $\mathrm{vs}(D) \leq p(D) - 1$. So $\mathrm{vs}(D) = p(D) - 1$ by Proposition 1.

Conversely, suppose that $p(D) = \mathrm{vs}(D) + 1$. Let H be the digraph obtained from D by adding a loop to each vertex that does not have one already. Thus, any strategy that processes H must put each vertex in TMU before processing it. Moreover, $\mathrm{vs}(H) = \mathrm{vs}(D)$ and $p(D) \leq p(H)$. Since $p(H) \leq \mathrm{vs}(H) + 1$ by Proposition 1, we infer that $p(H) = p(D)$. Therefore, any $p(D)$-strategy for H is a $p(D)$-strategy for D that put each vertex in TMU before processing it, as wanted. $\square$

The pathwidth of a graph is also its node-search number, and is closely related to other graph-searching invariants [1, 15]. Further study of the links between the process number, the vertex separation and also the search number has been performed recently [4, 14]. We refer the reader to the recent survey of Fomin and Thilikos about graph-searching [7].

We focus on the problem of recognition and characterization of digraphs and graphs with small process numbers. In Section 2, we first identify graphs whose connectivity equals their process number (Theorem 3). Then, we characterize graphs with process number at most two in terms of excluded minors and we also provide a structural description (Theorem 6). We moreover show how such graphs can be recognized and processed in linear time (Subsection 2.2). We turn to digraphs in Section **??**. We characterize digraphs with process number at most two (Lemma 14), and show how to recognize whether a graph D has process number at most two (and if yes how to process it) in time $O(n^2(n + m))$, where n is the number of vertices of D, and m its number of arcs (Proposition 17).

2 Graphs

We start by characterizing the graphs whose connectivity is equal to their process number.

Theorem 3. *A p-connected graph G can be p-processed if and only if there exists a neighborhood of size p whose deletion induces an independent set in G.*

Proof. Let G be a p-connected graph. If there is a set of p vertices of G whose deletion induces an independent set, then G has process number at most p (and hence exactly p since the minimum degree of G is at least p).

Conversely, let G be a p-connected graph with $p(G) = p$ and consider a p-strategy for G. Stop the strategy just before processing the first vertex v. By the p-connectivity, G has minimum degree p, so v has degree exactly p, and all its neighbors are in TMU. Let A be

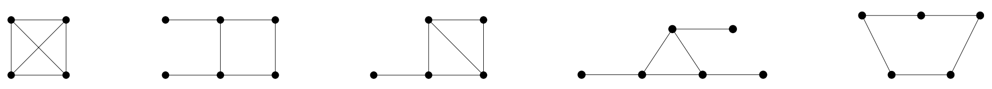

(a) K_4 (b) H_0 (c) H_1 (d) H_2 (e) C_5

Figure 3: Some minor-obstructions for 2-processed graphs.

the set of vertices whose neighborhood is included in $N(v)$ — and hence is *exactly* $N(v)$, by the p-connectivity. Without loss of generality, we can assume that the first steps of the strategy are to process all vertices of A. If all the vertices not in $N(v)$ have been processed, then the set $N(v)$ fulfills the desired condition.

Otherwise, there exists a vertex $w \notin A \cup N(v)$. Define z to be the next vertex to be processed. Since the strategy uses p TMUs and all the vertices of A have already been processed, $z \in N(v)$. Moreover, $N(z) \subset N(v) \cup A$. Thus, $N(A \cup \{z\}) \subseteq A \cup N(v)$. Consequently, $N(v) \setminus \{z\}$ is a set of $p-1$ vertices whose deletion disconnects w from $A \cup \{z\}$, a contradiction. □

The class of graphs with process number at most p is closed under minors. Indeed, assume that there exists a p-strategy for a given graph G. Let G' be the minor of G obtained by contracting the edge uv into a single vertex w. Without loss of generality, suppose that u is processed before v — hence v is in a TMU when u is processed. Apply the strategy to G'. The first step concerning the vertices u, v is to put one of them in TMU. Instead, put w in a TMU. The remaining of the strategy can then be applied, ignoring the processing of u, and processing w instead of v. Thus, G' also has process number at most p.

We focus on graphs with small process number. The first interesting case is when p is two, since only independent sets can be 0-processed and only the stars have process number exactly one. We note here that Bodlaender proved that every minor-closed class of graphs that does not contain all planar graphs has a linear time recognition algorithm [2]. This result follows from a linear time algorithm that determines whether a graph has treewidth, or pathwidth, at most k, and if so finds a tree decomposition, or a path decomposition, of width at most k, respectively. However, this algorithm is rather impracticable [12].

2.1 Graphs with process number two

In this section, we characterize graphs with process number at most two.

Lemma 4. *Let K be one of the graphs of Fig. 3. Every graph with a K-minor has process number at least three.*

Lemma 5. *Let K consist of three subgraphs chosen among T_a, T_b and T_c and merged at vertex* $\square$ *(see Fig. 4). Every graph with a K-minor has process number at least three.*

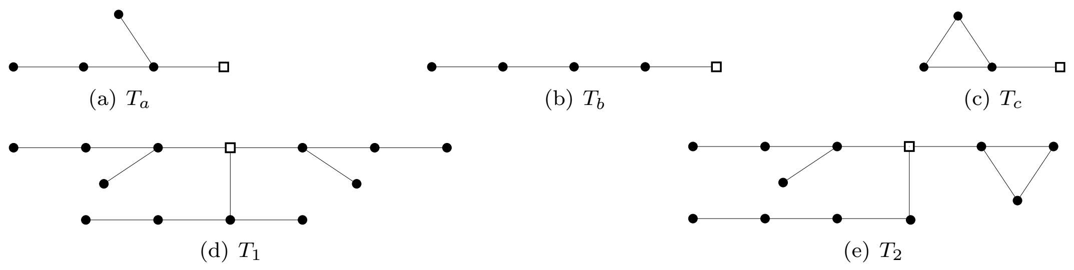


Figure 4: T_1 and T_2 are two of the 10 non-isomorphic minor-obstructions for 2-processed graphs obtained using 3 subgraphs chosen among T_a, T_b and T_c and merged at vertex $\square$.

Given a subgraph or a set of vertices X of a graph G, a subgraph H of $G - X$ is *attached* to a vertex x of X if x is the only vertex of X adjacent to a vertex of H in G. We can now give a complete characterization of graphs that can be 2-processed. Let M_1 be the collection of graphs depicted in Fig. 3 and let M_2 be the collection of graphs defined in Fig. 4.

Theorem 6. *For every connected graph $G = (V, E)$, the following assertions are equivalent.*

(a) *$p(G) \leq 2$;*

(b) *G does not contain any of the graphs of $M_1 \cup M_2$ as a minor;*

(c) *G consists of vertices $a_1, a_2, \dots a_r$, $r \geq 1$, such that each consecutive pair a_i, a_{i+1} is joined by an arbitrary number (at least one) of edges $\{a_i, a_{i+1}\}$ or paths $\left\{a_i, b_i^j, a_{i+1}\right\}$, $j \in \mathbb{N}$, along with an arbitrary number of subgraphs G_i^l attached to a_i, $l \in \mathbb{N}$, where each graph G_i^l is a star.*

Proof. The fact that (a) implies (b) follows from Lemmas 4 and 5. Let us show now that (b) implies (c).

We prove the assertion by induction on the number of vertices of G, the result being true if G has at most 3 vertices.

Suppose first that G is 2-connected. Thus, it has a cycle C of length 3 or 4, because G has no C_5-minor. If G has a 3-cycle C, then there exists a vertex d adjacent to at least 2 vertices of C, and exactly 2 since G has no K_4-minor. Let a and b be those two vertices, and let c be the third vertex of C. Since G contains no H_1, the vertices c and d have degree 2 in G. Therefore, G consists of the edge ab and some vertices of degree 2 adjacent to a and b, and hence G fulfills condition (c). Assume now that G has no 3-cycle, hence G has an induced 4-cycle C. If G is a 4-cycle, then the conclusion follows, so let assume that v is a vertex of G not in C. We can moreover assume that v has at least 2-neighbors in C. Since G has no C_5, the vertex v has exactly 2 neighbors in C, which are not adjacent. The vertex v cannot have degree more than 2 in G, for otherwise G would contain an H_0-minor.

Consequently, G consists of a 4-cycle $abcd$ and some vertices of degree 2 adjacent to the vertices a and c, and hence G satisfies condition (c).

We assume now that G has a cut-vertex v. Let $Z_1, Z_2, \ldots Z_n$ be the connected components of $G - v$, and for each index i let $D_i := Z_i + v$. If each Z_i is a star, then setting $r := 1$ and $a_1 := v$ shows that G satisfies (c). So we assume that Z_1 is not a star. We note that at most two components Z_i may not be stars since G has no graph of $M_1 \cup M_2$ as a minor.

We assume first that only Z_1 is not a star. By the induction hypothesis, D_1 fulfills condition (c), so we let $A' := \{a'_1, a'_2, \ldots, a'_s\}$ and B' be as stated in condition (c). Observe that we can moreover assume that each vertex a'_i with $1 < i < s$ is a cut-vertex of D_1, such that exactly two components of $D_1 - a_i$ are not stars. In particular, for $i \in \{1, s\}$ the vertex a'_i has a neighbor not in $A' \cup B'$. Moreover, we may assume that no vertex a'_i with $1 < i < s$ has degree 2, for otherwise we can consider them as vertices of B'. If $v \in A'$ then the graph G fulfills condition (c), the components $Z_2, \ldots, Z_n$ being just additional stars attached to v. If $v \in B'$, say $v = b_i^1$ for some index i, then a'_i or a'_{i+1} has degree 2 in G for otherwise G would contain a H_1 or H_2 as a minor. By symmetry, we assume that a'_{i+1} has degree 2, hence $i = s - 1$. Setting $A := A' \setminus \{a'_s\} \cup \{v\}$ and $B := B' \setminus \{v\} \cup \{a'_s\}$ shows that G fulfills condition (c). Finally, assume that v belongs to a star $S := G_i^1$. If v cannot be considered as the center of S, then by our assumption on A' and because G has no minor from M_2, we infer that $i \in \{1, s\}$, say $i = 1$. Moreover, note that va_1 is not an edge of G, for otherwise G would contain H_2. Let w be the center of S. Setting $A := A' \cup \{w, v\}$ yields the desired conclusion. If v is the center of S, then a similar argument (with $w = v$) applies if $i \in \{1, s\}$. So, suppose that $1 < i < s$. Then the subgraph induced by $\cup_{j>2} D_i$ is a star since G has no minor from M_2. Thus G has the desired structure.

It remains to deal with the case where another component Z_i, say Z_2, is not a star. Let $A'' := \{a''_1, \ldots, a''_t\}$ and B'' be as given by condition (c) applied to the graph D_2. We make the same assumption on A'' as on A', i.e. every vertex a''_i with $1 < i < t$ is a cut-vertex of D_2 and exactly two components of $D_2 - a_i$ are not stars. We also assume that such a vertex a''_i has degree more than 2. If $v \in A'$ then $v = a'_i$ with $i \in \{1, s\}$ by condition (b). Similarly, if $v \in A''$ then $v = a''_j$ with $j \in \{1, t\}$. In this case, setting $A := A' \cup A''$ yields the desired conclusion. So suppose that $v \notin A'$. Let a'_i be a vertex of A' closest to v. We infer that $i \in \{1, s\}$ by condition (b), say $i = 1$. Note that v cannot then belong to B'. So v belongs to a star G_1^l, and let w be the center of this star — if possible $w = v$. Similarly, either $v \in A''$ or v belongs to a star attached to a''_1 or a''_t. In the latter case, let u be the center of this star, letting u be w if possible. Setting $A := A' \cup \{w, v, u\} \cup A''$ shows that G fulfills condition (c). In the former case, we set $A := A' \cup \{w, v\}$. This concludes the proof of the assertion.

It remains to show that (c) implies (a). The graph G can be 2-processed as follows. (1) Put a_1 in TMU and set $i := 1$, (2) while $i < r$, process all subgraphs G_i^l (this uses a second TMU, which is freed at the end), (3) put a_{i+1} in TMU, (4) process all vertices b_i^j, (5) process the vertex a_i (which frees a TMU) and increment i. □

A typical example of a graph that can be 2-processed is given in Fig. 5.

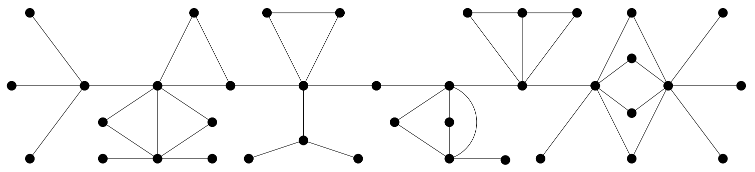


Figure 5: A typical graph with process number two.

Corollary 7. *Given two graphs H and H' that can be 2-processed and their corresponding paths $a_1, a_2, \dots, a_r$ for H and $a'_1, a'_2, \dots, a'_s$ for H', the graph G built from the union of H and H' and where vertices a_r and a'_1 are merged can be 2-processed.*

Proof. By the construction, G satisfies condition (c) of Theorem 6. □

2.2 An algorithm to recognize graphs with process number at most two

We present a linear time and space complexity algorithm for deciding whether a graph can be 2-processed. The idea of the algorithm is as follows. First, we note that we can decide whether a graph is a star in time $O(|N(u)| + |N(v)|)$, where u is any vertex of that graph and $v \in N(u)$, since one of them must be the center of the star. Then, if we are given the vertex a_1 of condition (c) of Theorem 6, we can process all attached stars in time linear in their size, then identify the vertex a_2, and so process the graph. Also, starting from vertex a_i and thanks to Corollary 7, we can identify in linear time the vertex a_1. Thus, the core of the algorithm is, starting from any vertex u, to identify in linear time a vertex a_i, which is done using a proper analysis of the size of the neighborhoods at distance one and two of u.

Before going into details, we need some more ground work. We show that deciding whether a graph can be 2-processed can be done in linear time. To this end, we first note in Proposition 8 that we can decide very efficiently if a graph can be 1-processed, and in Proposition 9 that we can decide in linear time if a 2-connected graph can be 2-processed.

From now on, we assume that a vertex u of G contains the list $N(u)$ of its neighbors, its degree, a Boolean variable u.ACTIVE set to FALSE if the vertex is in TMU or if it has been processed, and an integer — or a pointer — u.TAG, which is set to a if the vertex u is visited while processing vertex a. We also assume that we can access any vertex of G in constant time, and finally that $\chi(u, v)$ is a function which returns in constant time 1 if $v \in N(u)$ and 0 otherwise. More precisely, the function χ uses an array of size $|V(G)|$ initialized to 0. Neighbors of a are set to 1 at the beginning of the processing phase and set back to 0 at the end of the processing phase which can thus be done in time $O(|N(a)|)$. So, the overall cost due to the management of χ for all vertices of type a_i (see Theorem 6) is linear in the size of G.

Proposition 8. *Given a graph G and a vertex u, we can decide in time $O(|N(u)| + |N(v)|)$ if G can be 1-processed or not, where v is any neighbor of u.*

Proof. Since a star has at most one vertex of degree more than one, it is sufficient to check that:

- if $|N(u)| > 1$, then every neighbor of u has degree one. This can be checked in time $O(|N(u)|)$;
- if $|N(u)| = 1$, then the unique neighbor v of u cannot have neighbors of degree larger than one, which can be checked in time $O(|N(v)|)$.

So, overall, the time complexity is $O(|N(u)| + |N(v)|)$. □

Proposition 9. *Given a 2-connected graph G, we can decide in linear time if G can be 2-processed.*

Proof. Let $n \geq 3$ be the order of G. According to Theorem 3, we know that G should be either $K_{2,n-2}$ or $K_{2,n-2}$ plus an edge joining the two vertices of the bipartition of size two. This can be verified as follows: we choose three arbitrary vertices of G. One of them must have degree two and we call a and b its neighbors. Now it remains to check that the neighborhood of each vertex $v \in V \setminus \{a, b\}$ is exactly $\{a, b\}$. This procedure is linear in time. □

Proposition 10. *Given a graph G, we can check in linear time if $p(G) \leq 2$.*

Proof. The proof consists of three steps.

(1) First, we prove that if we are given a graph G and a vertex a, then we can decide in linear time if G can be 2-processed under the constraint that a is the vertex a_1 of condition (c) of Theorem 6. To this end, let us analyze the algorithm described in the proof of Theorem 6. We set $a_1 := a$, and we suppose that we are at step $i \geq 1$ of the loop.

- *Put the vertex a_i in* TMU*:* We just have to set the Boolean variable a_i.ACTIVE to FALSE.
- *Remove from G all subgraphs of kind G_i^l:* First, we have to determine which neighbors of a_i belong to stars and which are of type a_i or b_i^j. According to Proposition 8 we can decide whether a neighbor u of a_i belongs to a star or not in time $O(|N(u)| + |N(v)|)$, where v is a neighbor of u, if any. Note that we consider the degree of u and v minus $\chi(a_i, u)$ and $\chi(a_i, v)$, respectively. Simultaneously, we place a tag on all neighbors of u and v, to avoid double checking. If u belongs to a star, we process it in time $O(|N(u)| + |N(v)|)$, that is setting the Boolean variables ACTIVE to FALSE. So edges of stars will be visited twice during the processing of vertex a_i. We also visit all edges adjacent to vertex a_{i+1} once.

- *Determine the vertex a_{i+1} and process all vertices b_i^j:* To determine the next vertex a_{i+1}, we have to check that all remaining neighbors of degree one (vertices of type b_i^j, if any) have the same neighbor, which should also be the remaining neighbor of degree more than 1. (Note that there is such a neighbor for otherwise the previous step would have processed all the neighbors of a.) Then it remains to process vertices of type b_i^j. During this step, we visit all remaining neighbors of a_i once.

A graph G satisfies condition (c) of Theorem 6 with the vertex a being the vertex a_1 if and only if this algorithm processes the whole graph. Note that the algorithm fails if more than one vertex are candidate to be the vertex a_{i+1}.

Overall, each edge of G is visited twice and a constant number of operations are performed for each vertex. So we can process G in linear time.

(2) According to the previous step and Corollary 7, given a graph G and a vertex a_i, we can check in linear time if G can be 2-processed or not. Indeed, we process all subgraphs of G attached to a_i that can be 1-processed. Now $G - a_i$ must contain at most 2 connected components. If it has no connected components, then $p(G) \leq 2$; if it has only one connected component, then we apply step (1) on G from a_i; otherwise, let H and H' be these two components. We set $K := H + a_i$ and $K' := H' + a_i$. As observed in the proof of Theorem 6, the graph G has process number at most 2 if and only if K and K' both satisfy assertion (c) of Theorem 6 with a_i being considered as the vertex a_1 in the decomposition of K and in the decomposition of K'. Thus, we apply step (1) on K from a_i. If step (1) succeeds then we apply step (1) on H' from a_i to know whether G can be 2-processed.

(3) It remains to find a vertex a_i in G. We explain now a procedure that returns a vertex which can safely be considered as one of the vertices a_i, provided that G fulfills condition (c) of Theorem 6. To this end, choose the vertex u of maximum degree of G. If vertex u has degree two, then G is either a path or a cycle and step (2) will give a correct answer from u. If $|N(u)| > 2$, then u is either the center of a star or a vertex of type a_i. Notice that the center of a star is at distance at most two from one of the vertices a_i. So, let k_1 and k_2 be the number of vertices of degree at least three that are at distance one and two of u, respectively. Let x_1 and x_2 be any vertices of degree greater than 2 that are at distance one and two from u, respectively. Suppose that G can be 2-processed and consider the following cases.

- If $k_1 + k_2 = 0$, then the procedure can safely return u. Otherwise, the vertex u will be at distance at least three from a vertex of type a_i. Hence, the subgraph obtained by removing the path induced by the vertices a_i, and which contains u also contains a path of length at least four. Therefore, it cannot be 1-processed, which contradicts Theorem 6.

- If $k_1 = 1$, then either u or x_1 can safely be returned — and possibly both. So, it is sufficient to check if the subgraph containing x_1 in $G - \{u\}$ is a star. If it is true, then the procedure returns u and otherwise x_1.

- The case when $k_1 = 0$ and $k_2 = 1$ is similar to the previous one, with x_2 playing the role of x_1.
- If $k_1 \geq 2$ or $k_1 = 0$ and $k_2 \geq 2$, then u is returned. This is a safe choice: suppose that $p(G) \leq 2$ and yet u cannot be considered as one of the vertices a_i. When $k_1 \geq 2$, the vertex u has at least two neighbors of degree at least three, v and w. Suppose first that v and w are not adjacent. If both of them are vertices a_i, then u should be a vertex b_i^j, which is not the case since u has degree more than 2. If at most one of v and w is a vertex of type a_i, then the connected component of u in the subgraph of G induced by the deletion of the vertices a_i either is not attached to the vertices a_i, or cannot be 1-processed, a contradiction. Finally, if vw is an edge then G contains the graph H_1 or H_2 of Figure 3, which contradicts Theorem 6. An analogous argument holds when $k_1 = 0$ and $k_2 \geq 2$.

To sum-up, we can find in linear time a vertex of type a_i, and starting from that vertex, we can check in linear time if G can be 2-processed, which concludes the proof. □

A precise description of an algorithm to recognize graphs with process number 2 (and obtain a 2-strategy, if any) is given by Algorithms 1, 2, 3, 4 and 5.

Algorithm 1 Function Test-2-process-from

```
Require: a connected graph $G$ and a vertex $a$ with a stone on it.
Ensure: returns SUCCEED if the graph $G$ can be 2-processed with a first stone on $a$.
1: $a$.ACTIVE $\leftarrow$ FALSE
2: $CC_2 \leftarrow$ FALSE {$CC_2$ indicates if a connected component that cannot be 1-processed has
   already been found.}
3: for all $v \in N(a)$ such that $v$.ACTIVE and $v$.TAG $\neq a$ do
4:   if Is-Star$(G, v, a, a)$ then
5:     Process-Star$(G, v, a)$
6:   else if not $CC_2$ then
7:     $CC_2 \leftarrow$ TRUE
8:   else
9:     return FAILED
10:  end if
11: end for
12: if not $CC_2$ then
13:   return SUCCEED
14: end if
15: $FC \leftarrow$ FALSE {$FC$ indicates if we have found a candidate for the next vertex to be visited,
    $a_{i+1}$}
16: for all $v \in N(a)$ such that $v$.ACTIVE do
17:   if not $FC$ then
18:     if $|N(v)| = 2$ then
19:       $a' \leftarrow N(v) - \{a\}$
20:     else
21:       $a' \leftarrow v$
22:     end if
23:     $FC \leftarrow$ TRUE
24:   else if $(|N(v)| = 2$ and $N(v) - \{a\} \neq \{a'\})$ or $(|N(v)| > 2$ and $v \neq a')$ then
25:     return FAILED
26:   end if
27:   $v$.ACTIVE $\leftarrow$ FALSE
28: end for
29: return Test-2-process-from$(G, a')$
```

Algorithm 2 Function Is-Star

Require: a graph G, a vertex u that should belong to a star, a vertex a that should not be considered in the neighborhoods, and a tag t.

Ensure: returns TRUE if u belong to a star and FALSE otherwise. All visited vertices receive tag t.

```
1: $c \leftarrow u$
2: if $|N(u)| - \chi(a,u) = 1$ then
3:     $c \leftarrow N(u) - \{a\}$
4: end if
5: $c.\textsc{tag} \leftarrow a$
6: $bool \leftarrow \textsc{true}$
7: for all $v \in N(c) - \{a\}$ do
8:     if $|N(v)| - \chi(a,v) > 1$ then
9:         $bool \leftarrow \textsc{false}$
10:    end if
11:    $v.\textsc{tag} \leftarrow t$
12: end for
13: return $bool$
```

Algorithm 3 Procedure Process-Star

Require: a graph G, a vertex u that belongs to a star and a vertex a that should not be considered in the neighborhoods.

Ensure: Inactivate all vertices of the star attached to u except a.

```
1: $c \leftarrow u$
2: if $|N(u)| - \chi(a,u) = 1$ then
3:     $c \leftarrow N(u) - \{a\}$
4: end if
5: for all $v \in N(c) - \{a\}$ do
6:     $v.\textsc{active} \leftarrow \textsc{false}$
7: end for
```

Algorithm 4 Function First-Vertex

```
Require: a connected graph $G$.
Ensure: Return a vertex of type $a_i$.
    {We first choose the vertex of maximum degree of $G$.}
 1: Let $u$ be a vertex of $G$
 2: for all $v \in V(G)$ do
 3:   if $|N(u)| < |N(v)|$ then
 4:     $u \leftarrow v$
 5:   end if
 6: end for
    {Then we count the number of vertices of degree $\geq 3$ at distance one and two.}
 7: if $|N(u)| \geq 3$ then
 8:   $k_1 \leftarrow 0$, $k_2 \leftarrow 0$
 9:   for all $v \in N(u)$ do
10:     if $|N(v)| \geq 3$ then
11:       $x_1 \leftarrow v$
12:       $k_1 \leftarrow k_1 + 1$
13:     end if
14:   end for
15:   for all $v \in N(N(u)) - \{u\}$ do
16:     if $|N(v)| \geq 3$ then
17:       $x_2 \leftarrow v$
18:       $k_2 \leftarrow k_2 + 1$
19:     end if
20:   end for
      {Finally we decide whether $u, x_1$ or $x_2$ is of type $a_i$.}
21:   if $k_1 = 1$ and Is-Star$(G, u, x_1)$ then
22:     $u \leftarrow x_1$
23:   else if $k_1 = 0$ and $k_2 = 1$ and Is-Star$(G, u, x_2)$ then
24:     $u \leftarrow x_2$
25:   end if
26: end if
27: return $u$
```

Algorithm 5 Main procedure to 2-process graphs

Require: a graph G.

Ensure: returns TRUE if G can be 2-processed and FALSE otherwise.

```
1:  $a \leftarrow$ First-Vertex($G$)
2:  Initialize $\chi$ to 0 and set neighbors of $a$ to 1                $O(n)$
3:  $a$.ACTIVE $\leftarrow$ FALSE, $bool \leftarrow$ FALSE, TAG $\leftarrow 0$
4:  for all $v \in N(a)$ such that $v$.ACTIVE and $v$.TAG $\neq a$ do
5:    if Is-Star($G, v, a,$ TAG) then
6:      Process-Star($G, v, a$)
7:    else
8:      TAG $\leftarrow$ TAG $+ 1$
9:    end if
10: end for
11: if TAG $> 0$ and TAG $< 3$ then
12:   for all $v \in N(a)$ such that $v$.TAG $= 1$ do
13:     $v$.ACTIVE $\leftarrow$ FALSE
14:     $w \leftarrow v$
15:   end for
16:   $a$.ACTIVE $\leftarrow$ TRUE
17:   $bool \leftarrow$ Test-2-process-from($G, a$)
18:   if TAG $= 2$ and $bool$ and $w$.TAG $= 1$ then
19:     for all $v \in N(a)$ such that $v$.TAG $= 1$ do
20:       $v$.ACTIVE $\leftarrow$ TRUE
21:     end for
22:     $a$.ACTIVE $\leftarrow$ TRUE
23:     $bool \leftarrow$ Test-2-process-from($G, a$)
24:   end if
25: end if
26: return $bool$
```

3 Digraphs

In this section we characterize the classes of directed graphs with process number at most two. We first state a general remark.

Lemma 11. *For any digraph D, the process number of D is equal to the maximum of the process numbers of its strongly connected components.*

Proof. Let DAG-$\mathcal{C}$ be the acyclic digraph of the strongly connected components of D, i.e. each vertex of DAG-$\mathcal{C}$ corresponds to a strongly connected component of D, and there is an arc from a vertex u to a vertex v if and only if there is an arc between the corresponding strongly connected components in D. We can process each strongly connected component of D separately, in the order induced by DAG-$\mathcal{C}$. □

A digraph can be 0-processed if and only if it has no cycles, that is if it is a DAG. In particular a direct path can be 0-processed. Using a topological sort [3], one can check in linear time whether a digraph is acyclic.

3.1 Digraphs with process number 1

First of all, observe that a strongly connected digraph D can be 1-processed if there exists a vertex u such that $D - \{u\}$ is a DAG. In other words, a strongly connected digraph D can be 1-processed if it has a minimum feedback vertex set of size 1, that is if D is a *reducible flow graph* [8]. This can be checked in linear time [16, 13]. From this follows that we can characterize digraphs that can be 1-processed.

Lemma 12. *A digraph D can be 1-processed if and only if all its strongly connected components are reducible flow graphs.*

Note that a digraph D' obtained from a digraph D by contracting each strongly connected component S_i to a vertex s_i is a DAG. It follows that we can decide in linear time if a given digraph can be 1-processed.

Now follows a simple algorithm that decides in linear time and space complexity if a digraph can be 1-processed. This algorithm is an alternative to the algorithms of Shamir [16] and Rosen [13], which better fits our setting. In particular, we can compute the minimum feedback vertex set of 1-processed digraphs in linear time.

Proposition 13. *Given a digraph D with n vertices and m arcs, we can decide in time and space complexity $O(n + m)$ if D can be 1-processed.*

Proof. We assume in this proof that D is a strongly connected digraph. Otherwise, we identify each strongly connected components in time $O(n + m)$ using a topological sort [3] and apply the following algorithm on each of them without changing the overall complexity.

We use the observation that a strongly connected digraph D has process number 1 if and only if there is a vertex v such that $D - v$ is a DAG. The following shows how to determine whether such a vertex exists, and find one if any, in time $O(n + m)$.

Since D is strongly connected, it contains a directed cycle $C := x_1x_2 \dots x_kx_1$ and so the vertex v must be one of the vertices x_i. We maintain a list $\mathcal{L}$ of vertices that are candidates to be the vertex v. To this end, we define $\mathcal{L}$ to be an array of k integers, initialized to 0. A vertex x_i of C is *valid* if $\mathcal{L}[i]$ is 0.

Suppose that there exists a directed path $x_iy_1y_2 \dots y_\ell x_j$ with $V(C)$ disjoint from $\{y_1, y_2, \dots, y_\ell\}$. If $i < j$ then none of the vertices $x_{i+1}, x_{i+2}, \dots, x_{j-1}$ can be the sought vertex v. If $j < i$ then none of the vertices $x_{i+1}, \dots, x_k, x_1, \dots, x_{j-1}$ can be the sought vertex v. (We allow ℓ to be 0, in which case it means that there is an arc from x_i to x_j.)

For i from 1 to k, we run a Breadth-First Search (BFS) rooted at x_i and in which we do consider neither the outneighbors of the vertices of $V(C) \setminus \{x_i\}$, nor the outneighbors of the vertices already visited during a previous BFS. For each vertex, we record the step in which it is first visited, i.e. we record i if the vertex was first visited during the BFS rooted at x_i. Consider the step $i \in \{1, 2, \dots, k\}$.

If the BFS reaches a vertex x_j, then we set $\mathcal{L}[\ell] := i$ for each ℓ from $j-1$ down to $i+1$ (modulo k). Note that if $j = i$, then it means that only x_i remains valid, and so we can directly set $v := x_i$ and returns TRUE if and only if $D - x_i$ is a DAG.

If x_i is still valid and the BFS reaches a vertex already visited during, say, step $j < i$, then we set $\mathcal{L}[\ell] := i$ for ℓ from j down to 1 and from k down to $i+1$.

To cope with the complexity requirement, we make the vertices not valid in a backward way and stop when a vertex that has been removed previously is found. So doing, each cell of $\mathcal{L}$ is modified once, and we test in total at most $O(m)$ times if a vertex is still valid. So the cost due to maintaining the list of valid candidates during the algorithm is $O(n+m)$.

Observe that if a vertex $x_i \in V(C)$ is still valid once all the BFS are performed, then all the directed cycles of D that intersect C contain x_i. Thus, if LL is non-empty then it suffices to return TRUE if and only if $D - x_i$ is a DAG. If $\mathcal{L}$ contains no valid vertices, then we can conclude that for each $i \in \{1, 2, \dots, k\}$, the digraph $D - x_i$ contains a directed cycle, and consequently D cannot be 1-processed.

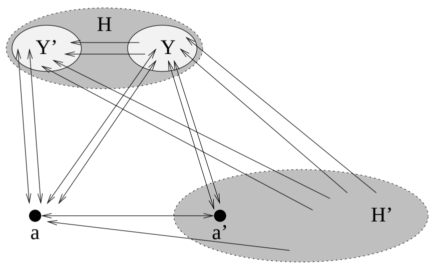
Overall this algorithm takes time $O(n+m)$. First, we can find a cycle in time $O(n)$, for example by choosing a starting vertex and moving to the first neighbor until we reach a vertex that has already been visited. Second, we visit each arc of D once during the BFS. So this part takes time $O(m)$. Also, the total cost due to $\mathcal{L}$ is $O(n+m)$. Finally, we can check whether a subgraph of D is a DAG in time $O(n+m)$ using a topological sort.

The space complexity is linear since except the size of the graph, a BFS needs only a stack, which can be implemented using an integer array of size n, and the list of candidates uses an array of size at most n. □

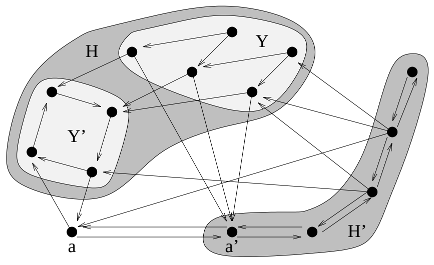
3.2 Digraphs with process number 2

Our aim in this subsection is to present a polynomial-time recognition algorithm for digraphs with process number 2.

Let D be a digraph and let a be one of its vertices. We say that D is a $(2, a)$-*digraph* if there exists an (≤ 2)-strategy to process D whose first step is to put a in TMU. Note



(a) General shape of a $(2,a)$-digraph



(b) Example of a $(2,a)$-digraph

Figure 6: General shape and example of $(2,a)$-digraphs.

that a digraph can be 2-processed if and only if it is a $(2,a)$-digraph for some vertex a (see Fig. 6(b)). First, we show how to determine whether D is a $(2,a)$-digraph.

Lemma 14. *Let D be a (weakly) connected digraph and let a be one of its vertices. The digraph D is a $(2,a)$-digraph if and only if the digraph $D-a$ can be partitioned into two subdigraphs H and H' satisfying the following conditions:*

(i) there exists a vertex a' of H' such that (H',a') is a $(2,a')$-digraph;

(ii) $N^+_D(H+a) \subseteq \{a'\}$

(iii) either

- *$p(H)=0$; or*
- *$p(H)=1$ and there exists a (possibly empty) set $Y \subset V(H)$ such that $N^-_D(a') \cap V(H) \subseteq Y$, $p(D[Y])=0$, and $(Y, V(H)\setminus Y)$ is a directed cut of H from Y to $V(H)\setminus Y$.*

Proof. If a has no outneighbors, then (D,a) is a $(2,a)$-digraph if and only if $p(D-a) \le 2$. So the characterization is valid with H being empty and H' being $D-a$. We assume now that a has at least one outneighbor in D.

Suppose first that there exist two subdigraphs H and H' as in the statement of the lemma. The following strategy shows that (D,a) is a $(2,a)$-digraph. Put the vertex a in TMU. If $p(H)=0$, then put a' in TMU, process a and then process H by condition (ii). If $p(H)=1$, then set $Y' := V(H)\setminus Y$ and process $D[Y']$. As $N^+_D(Y') \subseteq Y' \cup \{a\}$ by the definition of Y' and by condition (ii), we use at most one more TMU during this processing, and only a is left in TMU once $D[Y']$ is processed. Now, put a' in TMU and process the vertices of Y since $N^+_D(Y) \subset V(H) \cup \{a,a'\}$ by condition (ii) and $p(D[Y])=0$. Hence, in both cases, we have processed H, and a and a' are in TMUs. By condition (ii), we can now process a and then finish to process D since (H',a') is a $(2,a')$-digraph by condition (i).

Assume that (D, a) is a $(2, a)$-digraph, and consider a corresponding strategy to process it. Note that at most one outneighbor of a is processed after a, since the first step of the strategy consists of putting a in TMU. Stop the strategy just before a is processed. We let H be the subdigraph of D induced by all the vertices processed before a, and H' be the complement of H in $D - a$. By the definition, $|N_D^+(H + a) \cap V(H')| \leq 1$. If $|N_D^+(H + a) \cap V(H')| = 0$ then we define a' to be the first vertex of H' to be put in TMU by the strategy, and otherwise we define a' to be the unique outneighbor of $H + a$ in H'. In this case, the vertex a' must be already in TMU. As the strategy uses no more than two TMUs simultaneously, there are no arcs from $H + a$ to $H' - a'$, and so condition (ii) is fulfilled.

Recall that we consider the strategy only from the first step up to the last step before processing a. Observe that a is in TMU during all the steps considered, and if a' is put in TMU, then it stays in TMU until the last step considered. Consequently, no vertex of $X := N_D^-(a') \cap V(H)$ can be put in TMU, and hence $p(D[X]) = 0$. It also follows from this observation that $p(H) \leq 1$. If $p(H) = 1$, then let v be the first vertex of H to be put in TMU (hence $v \notin X$). Let Y' be the *outbranching of* v *in* H, that is

$$Y' := \{w \in V(H) : \text{ there exists a directed path from } v \text{ to } w \text{ in } H\}.$$

By the previous observation, we infer that $Y' \cap X = \emptyset$. Moreover, there are no arcs from Y' to $Y := V(H) \setminus Y'$. Thus, condition (iii) is fulfilled.

The remaining part of the strategy ensures that $H' + a$ is a $(2, a')$-digraph, which is more than required by condition (i). □

Before using the preceding characterization to derive a polynomial-time recognition algorithm, we state a useful lemma. Let D be a digraph and let v be a vertex of outdegree at most one of D. Let u be the unique outneighbor of v, if any. The *contraction of* v consists of removing v, linking every vertex of $N_D^-(v)$ to u, and removing any parallel arcs created (but not the loops that may appear).

Lemma 15. *Let D be a digraph and v a vertex of D with exactly one outneighbor u. Let D' be obtained by contracting the arc vu into the vertex u. Then $p(D) = p(D')$. Moreover, D is a $(2, a)$-digraph if and only if D' is a $(2, a')$ digraph where $a' = a$ if $a \neq v$, and $a' = u$ otherwise.*

Proof. Consider a p-process strategy for D'. Apply it to D with the extra step that v is processed as soon as u is processed or in TMU. This shows that $p(D) \leq p$. Conversely, consider a p-process strategy for D. We apply it to D', except that if a step puts v in TMU, we instead put u in TMU (if it is not already in TMU, or processed). This yields a p-strategy for D'. In particular, note that when v is processed then either u was already processed, or was put in TMU by the original strategy. Thus we do not use any extra TMU in the strategy for D'.

The 'moreover' part follows from above by a straightforward checking. □

Proposition 16. *Given a strongly-connected digraph D and a vertex $a \in V(D)$, Algorithm 6 decides in time $O(n(n + m))$ if (D, a) is a $(2, a)$-digraph.*

Algorithm 6 Function Is-$(2,a)$-digraph

```
Require: a strongly connected digraph $D$ and a vertex $a$
Ensure: returns SUCCEED if $D$ is a $(2,a)$-digraph.
 1: Put $a$ in TMU and remove it from the neighborhoods of its predecessors
 2: $\mathcal{C} \leftarrow$ set of strongly connected components of $D - a$
 3: Let DAG-$\mathcal{C}$ be the DAG of strongly connected components
 4: while it exists $C \in \mathcal{C}$ such that $C$ is a leaf of DAG-$\mathcal{C}$ and $p(C) \leq 1$ do
 5:    Process $C$, and so remove it from $D - a$, $\mathcal{C}$ and DAG-$\mathcal{C}$
 6: end while{Let $D_1$ be the remaining digraph}
 7: $D_2 \leftarrow$ Contract-rooted$(D_1, a)$
 8: if $V(D_2) = \{a\}$ then
 9:    return SUCCEED
10: else if $|N^+_{D_2}(a) - \{a\}| = 1$ then {we have $N^+_{D_2}(a) - \{a\} = \{a'\}$}
11:    return Is-$(2,a')$-digraph$(D_2 \setminus \{a\})$
12: else
13:    return FAILED
14: end if
```

Algorithm 7 Function Contract-rooted

```
Require: a connected digraph $D$, a vertex $a$ and the set $V^1_D$ of vertices of outdegree at most
    1 that is part of $D$.
Ensure: returns a reduced digraph, but vertex $a$ being unchanged.
 1: while $V^1_D \setminus \{a\}$ is not empty do
 2:    Let $u$ be any vertex of $V^1_D - \{a\}$, which we remove from $V^1_D$
 3:    if $N^+(u) > 0$ then
 4:       Let $v$ be the outneighbor of $u$
 5:       for all $w \in N^-(u)$ do
 6:          $N^+(w) \leftarrow N^+(w) \setminus \{u\} \cup \{v\}$
 7:          if $|N^+(w)| = 1$ then
 8:             $V^1_D \leftarrow V^1_D \cup \{w\}$
 9:          end if
10:       end for
11:    end if
12: end while
```

Proof. Let us prove that Algorithm 6 is correct. We assume that each time a vertex is processed, the neighborhoods of its predecessors are updated and so is the set V^1_D of vertices of out-degree at most 1.

Suppose first that (D,a) is a $(2,a)$-digraph. We consider the partition (H,H') of $D-a$ and the subset $Y \subseteq V(H)$ given by Lemma 14. We set $Y' := V(H) \setminus Y$. If $p(H) = 0$, then we may assume that $Y = V(H)$, and hence $Y' = \emptyset$.

Since $p(H) \leq 1$ and $N_D^+(Y') \subseteq Y'$ (arcs to a are removed by line 1), lines 2–5 will remove the whole digraph $D[Y']$, because (Y, Y') is a directed cut of H. Also, since $D[Y]$ is a DAG, every leaf vertex u without an arc to a' will be removed as $\{u\}$ is a strong component once $D[Y']$ is removed. Let $Y^r \subseteq Y$ be the remaining part of Y. The digraph $D[Y^r]$ is a DAG whose leaf vertices have for unique outneighbor a'. Thus, line 7 and so Algorithm 7 will contract Y^r into a', starting from the leaves.

Now, Algorithm 6 returns FAILED only if either the vertex a has more than one outneighbor in H', or it has an outneighbor b and $D_2 - a$ is not a $(2, b)$-digraph. The former case does not happen by Lemma 14, and in the latter case, it would mean that H' is not a $(2, a')$-digraph by Lemma 15, a contradiction. Therefore, Algorithm 6 returns SUCCEED, as desired.

Conversely, suppose now that the algorithm returns SUCCEED for a given digraph D, and let us prove that D is a $(2, a)$-digraph. We start by putting a in TMU. The algorithm starts by removing strongly-connected components that are leaves in DAG-$\mathcal{C}$, and have process number at most 1. We can safely process all these components using at most one TMU, which is freed at the end. Note that after these steps, the remaining digraph may not be strongly connected anymore, but the vertex a has outdegree at least one. Thanks to Lemma 15, we can ignore the contraction step of line 7. Then, as the algorithm returns SUCCEED, either only a remains, and we just process a to finish, or a has exactly one outneighbor called a', and the digraph $D_2 - a$ is a $(2, a')$-digraph. Thus, we can put a' in TMU, process a and then finish to process $D_2 - a$ using at most two TMUs. This shows that D is a $(2, a)$-digraph by Lemma 15.

The computation time of Algorithm 6 has two parts. The first part concerns the partition into strongly connected components (line 2) that takes time $O(n + m)$, the construction of DAG-$\mathcal{C}$ (line 3) in time $O(n)$, the application on each strongly-connected component of the algorithm of Proposition 13 for an overall cost in $O(n+m)$ including the update operations of line 5, and finally at most n recursive calls (line 11). Overall this part takes time $O(n(n+m))$.

The second part concerns Algorithm 7 and the maintenance of the corresponding data structures. Since the computation time of line 6 depends on the data structures chosen to store the digraph, we assume that the list of in- (respectively out-) neighbors is stored in an unsorted double linked list plus an array of size n recording for each neighbor its pointer in the list. Thus, we may add or remove a vertex of the in- (respectively out-) neighborhood of a vertex in constant time. Since a vertex may be contracted only once, and since in the worst case it has $O(n)$ predecessors, this part takes an overall time of $O(n^2)$.

Finally, the computation time of Algorithm 6 is in $O(n(n + m))$. □

We note that Algorithm 7 can me modified to decide if a strongly connected digraph D can be 1-processed, since it would then be contracted into a single vertex with a loop.

The process number of a digraph is at most p if and only if the process number of each of its strong components is at most p. Indeed, suppose that each strong component of a digraph D can be p-processed. The digraph D' of the strong components of D is acyclic. It suffices to p-process each strong component of D according to a topological order of D' to p-process D. Thus, we obtain the following result thanks to Proposition 16.

Proposition 17. *Given a digraph D we can decide in time $O(n^2(n+m))$ if it can be 2-processed.*

4 Conclusion

We modeled a rerouting problem in WDM networks using graph theory. To this end, we introduced a new (di)graph invariant, the process number, which turns out to be closely related to other well-studied invariants of (di)graphs. In particular, as Proposition 1 shows, it is a refinement of the vertex-separation (also called pathwidth in the case of undirected graphs). We also characterized the (optimal) process strategies of digraphs whose process number is different from their vertex separation (Proposition 2).

Our next goal was to characterize and recognize efficiently (di)graphs with small process number. In particular, we gave a linear time algorithm for recognition of graphs with process number at most two (Proposition 10, Algorithm 1), as well as a characterization in terms of excluded minors and a structural description (Theorem 6). For digraphs with process number two, we found a characterization that allows to recognize (and process) them in time $O(n^2(n+m))$. Finally, we linked the process number to the connectivity, by determining the graphs with process number equal to their connectivity (Theorem 3).

As for the excluded minor characterization, we are currently studying [5] graphs with process number 3. It may be the last case achievable, since we have so far a list of 185 266 forbidden minors, which are highly structured. It is interesting to note that for the pathwidth, such a characterization has been found up to pathwidth three [10] — for which there are 110 forbidden minors. On the other hand, the list for pathwidth 4 is not known, but it contains at least 122 millions forbidden minors and hence is probably out of reach. By Proposition 1, determining the excluded minors for graphs with process number 3 can be viewed as a scaling of this last problem, in the sense that this class contains graphs with pathwidth 3 and graphs with pathwidth 4.

References

[1] D. Bienstock, N. Robertson, P. Seymour, and R. Thomas. Quickly excluding a forest. *J. Combin. Theory Ser. B*, 52(2):274–283, 1991.

[2] H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

[3] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms.* The MIT Press, 1990.

[4] D. Coudert, S. Pérennes, Q.-C. Pham, and J.-S. Sereni. Rerouting requests in WDM networks. In *Septièmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications (AlgoTel'05)*, pages 17–20, Presqu'île de Giens, May 2005.

[5] D. Coudert and J.-S. Sereni. Obstruction set for graphs with process number three. In preparation.

[6] N. Deo, S. Krishnamoorthy, and M. A. Langston. Exact and approximate solutions for the gate matrix layout problem. *IEEE Transactions on Computer-Aided Design*, 6:79–84, 1987.

[7] F. V. Fomin and D. Thilikos. An annotated bibliography on guaranteed graph searching. *Theoretical Computer Science, Special Issue on Graph Searching*, 2008, to appear. . Available: http://www.ii.uib.no/ fomin/fedor/articles1/abgs.pdf.

[8] M.S. Hecht and J.D. Ullman. Characterizations of reducible flow graphs. *Journal of the Association for Computing Machinery*, 21(3):367–375, july 1974.

[9] N. G. Kinnersley. The vertex separation number of a graph equals its pathwidth. *Inform. Process. Lett.*, 42(6):345–350, 1992.

[10] N. G. Kinnersley and M. A. Langston. Obstruction set isolation for the gate matrix layout problem. *Discrete Appl. Math.*, 54(2-3):169–213, 1994.

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

[12] H. Röhrig. *Tree decomposition: A feasibility study.* Master's thesis, Max-Planck-Institut fur Informatik, Germany, 1998.

[13] B.K. Rosen. Robust linear algorithms for cutsets. *Journal of Algorithms*, 3(3):205–217, September 1982.

[14] J.-S. Sereni. *Colorations de graphes et applications.* PhD thesis, École doctorale STIC, Université de Nice-Sophia Antipolis, July 2006.

[15] P. D. Seymour and R. Thomas. Graph searching and a min-max theorem for tree-width. *J. Combin. Theory Ser. B*, 58(1):22–33, 1993.

[16] A. Shamir. A linear time algorithm for finding minimum cutsets in reducible graphs. *SIAM Journal on Computing*, 8(4):645–655, 1979.

Contents

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