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
MASCOTTE, INRIA – I3S(CNRS/UNSA)
Sophia Antipolis, France.
david.coudert@sophia.inria.fr

Jean-Sébastien Sereni
Institute for Theoretical Computer Science (ITI)
and Department of Applied Mathematics (KAM)
Charles University
Prague, Czech Republic.
sereni@kam.mff.cuni.cz

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.

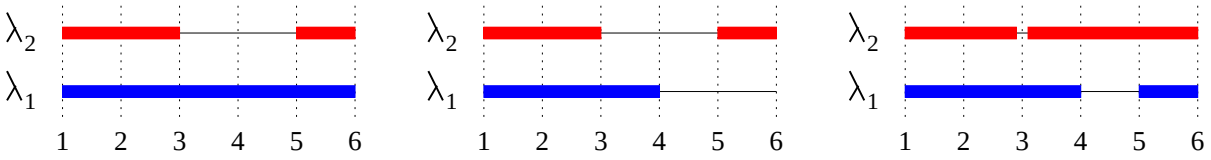
Keywords: rerouting, process number, vertex separation, pathwidth.

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 a length 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 (Fig. 1(b)). In the depicted situation, a new request (3,6) is 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.

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



(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.

The problem is modeled 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 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).

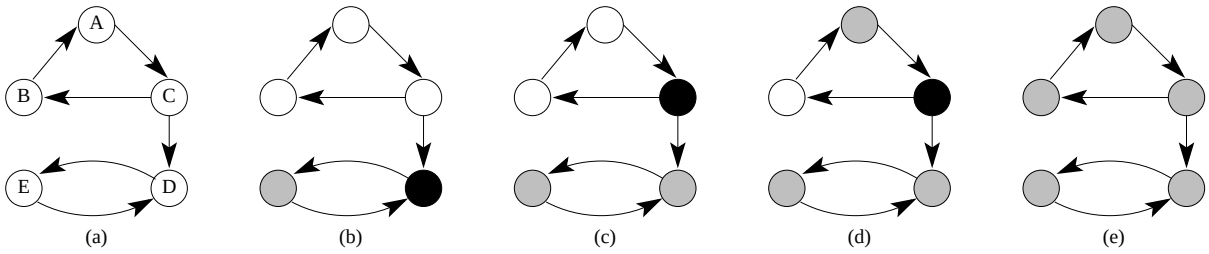


Figure 2: Processing of a graph: processed vertices are in grey and vertices in TMU are in black. In (a), every vertex has at least one outneighbor in the initial state, so we must put a vertex in TMU. In (b), the vertex D has been put in TMU, which allows to process the vertex E. To reach the state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

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 [7] proved that the vertex separation of any undirected graph equals its pathwidth. 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.* [3], 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. 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 $M \cup S$ 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 $M \cup S$ in TMU uses at most, and possibly, vs $+1$ TMUs. Hence, $p(D) \leq \mathrm{vs} + 1$. □

As determining the vertex separation of an arbitrary graph is APX [5], the preceding result shows that the process number problem also is. Further study of the links between the process number, the vertex separation and also the search number has been performed recently [3, 11].

We focus on the problem of recognition and characterization of digraphs and graphs with small process numbers.

2 Graphs

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

Theorem 2. *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 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 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.

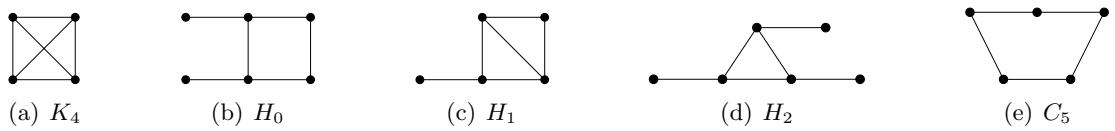
We focus on graphs with small process number. The first interesting case is when p is two, since only independent sets can be 0-processed and only the stars have process number exactly one. We note here that Bodlaender proved that every minor-closed class of graphs that does not contain all planar graphs has a linear time recognition algorithm [1]. 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 [9].

2.1 Graphs with process number two

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

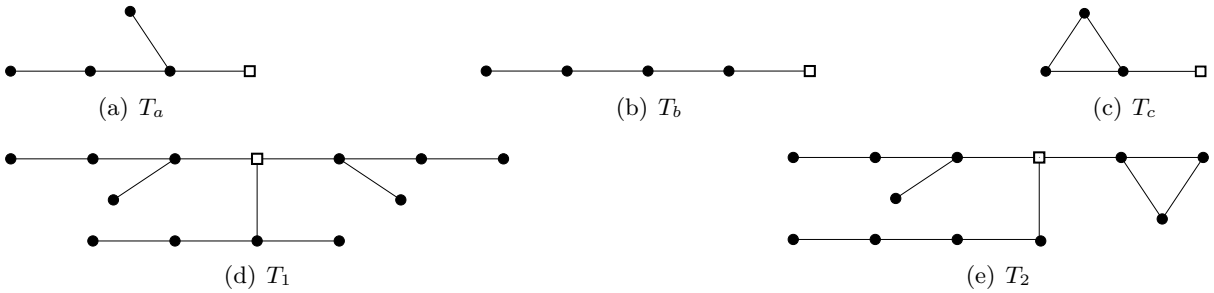
Lemma 3. *Let K be one of the graphs of Fig. 3. Every graph with a K-minor has process number at least three.*

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



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

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



(a) T_a (b) T_b (c) T_c

(d) T_1 (e) T_2

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.

Theorem 5. *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 Figs. 3 and 4;*

(c) *G consists of vertices $a_1, a_2, \ldots a_r$, $r \geq 1$, 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 3 and 4. Let us show now that (b) implies (c).

We choose a longest path $P := x_1 x_2 \ldots x_s$ of G that maximizes the number of connected components of $G - P$ that are stars. We assume that $s \geq 6$ for otherwise either G is a star and the implication is trivial, or $s \in \{4, 5\}$ and the implication can be directly checked.

Since G does not contain a cycle of length at least five, no vertex x_i is adjacent to x_{i+j} for $j \geq 4$. Moreover, as G does not contain the graph H_0 of Fig. 3, the vertex x_i is not adjacent to x_{i+3} if $i \in \{2, 3, \ldots, s-4\}$. We set $X' := \{x_2, x_3, \ldots, x_{s-1}\}$.

For $i \in \{2, \ldots, s-3\}$, if the vertex x_i is adjacent to x_{i+2}, then every vertex $v \in V$ adjacent to both x_i and x_{i+2} has degree two in G, for otherwise G would contain the graph H_2 of Fig. 3. In particular, x_{i+1} has degree two in G. We remove from X' all such vertices, and consider them as vertices b_i^j. Thus, we obtain a subset X of X', and a set B of vertices b_i^j of degree two in G for $i \in \{1, 2, \ldots, k\}$. Notice that this set is uniquely defined, because G contains no H_1. It only remains to show that the set X fulfills the desired conditions, which is trivially the case

if $X \cup B$ covers G. We can write $X = \{v_2, v_3, \ldots, v_s\}$ with $v_2 = x_2$, $v_s = x_{k-1}$ and the ordering of the vertices v_i is the same as the one of the vertices x_i.

Let C be a connected component of $G' := G - (X \cup B)$, and let v_i be a neighbor of C in X with i as small as possible. Note that both x_1 and x_k are isolated vertices in G', since P is a longest path of G. We now consider several cases regarding C.

The component C is comprised of x_1 or x_k By symmetry, we may assume that $V(C) = \{x_1\}$. By the previous remarks, $N(x_1) \subseteq \{x_2, x_3, x_4\}$. Note that C fulfills the desired conditions if $N(x_1) = \{x_2\}$. If $N(x_1) = \{x_2, x_3\}$ then the vertex x_1 can be added to the set B. If $N(x_1) = \{x_2, x_3, x_4\}$, then G contains H_1, a contradiction. Finally, if $N(x_1) = \{x_2, x_4\}$ then x_3 has degree two in G (otherwise G would contain H_0). Thus, x_3 belongs to B, and we can also add v_1 to B. Therefore C fulfills the desired condition.

The component C is an isolated vertex $v \notin \{x_1, x_k\}$ Then, v is not adjacent to v_{i+k} for $|k| \geq 3$, otherwise G would contain C_5. If $vv_{i+1} \in E$, then v has degree two in G since G has no H_0, no H_1 and no H_2. (Observe that, as $v_i = x_j$ for some j, the vertex v_{i+1} is either x_{j+1} or x_{j+2}.) So we can add v to the set B.

The component C contains at least two vertices First, observe that C is attached to X, otherwise G would contain H_0 or H_2. Suppose that C contains a path of length four $uvwz$. Then, as $P = x_1 \ldots x_s$ is a longest path of G, we infer by Lemma 4 that either $v_i = x_4$ or $v_i = x_{s-3}$. Hence, $s \geq 7$. Moreover, v_i is adjacent to exactly one of v and w and none of u and z. By symmetry, we assume that $vv_i \in E$ and that $v_i = x_4$. Note also that C is comprised precisely of the vertices u, v, w and z.

Observe that none of x_1, x_2 and x_3 has a neighbor x_j with $j \in \{5, 6, \ldots, s-1\}$, by Lemma 3. So, since P has been chosen so as to maximize the number of connected components of $G - P$ that are stars, we infer that there is a connected component C_1 of $G - P$ linked to $\{x_1, x_2, x_3\}$. Otherwise, the path $zwvx_4x_5 \ldots x_s$ would be a longest path that would contradict the maximality property of P. Therefore, by Lemma 4, we deduce that $s = 7$. Recall that none of x_5, x_6 and x_7 has a neighbor x_j with $j < 3$. Again by the maximality condition fulfilled by the longest path P, we deduce that there is a connected component C_2 of $G - P$ linked to $\{x_5, x_6, x_7\}$. Note that $C_1 \neq C_2$ by Lemma 3. But this contradicts Lemma 4, the vertex x_4 being the vertex $\square$.

Consequently, the component C is either a star or a triangle $T := uvw$. In the latter case, we deduce that $v_i = x_4$ or $v_i = x_{s-3}$ by Lemma 4, and that $s \geq 7$. By symmetry, assume that $v_i = x_4$. The vertex x_4 is linked to exactly one vertex of C, say v. We obtain a contradiction similarly as before by considering the path $uwvx_4x_5 \ldots x_s$ and using Lemma 4.

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 $i < r$, proceed all subgraphs G_i^l (this uses a second TMU, which is freed at the end), (3) put a_{i+1} in TMU, (4) proceed all vertices b_i^j, (5) proceed 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.

Corollary 6. *Given two graphs H and H' that can be 2-processed and their corresponding paths $a_1, a_2, \ldots, a_r$ for H and $a'_1, a'_2, \ldots, a'_s$ 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 5. □

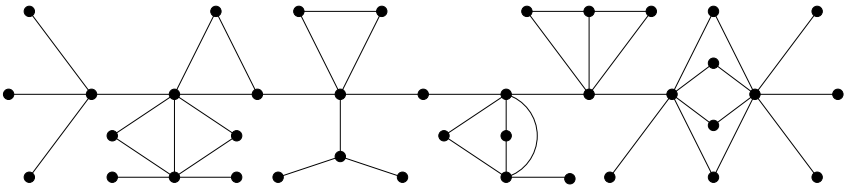


Figure 5: A typical graph with process number two.

A linear time and space complexity algorithm for deciding if a graph can be 2-processed can be found in Appendix A. The idea of the algorithm is as follows. First, notice that we can decide if a graph is a star in time $\mathcal{O}(|N(u)| + |N(v)|)$, 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 5, 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 6, 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. Finally, we can state the following proposition.

Proposition 7. *Given a graph G, we can check in linear time if $p(G) \leq 2$.*

3 Digraphs

In this section we characterize the classes of directed graphs with process number at most two.

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, 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 exist 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* [6]. This can be checked in linear time [12, 10]. From this follows that we can characterize digraphs that can be 1-processed.

Lemma 8. *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.

The interested reader in Appendix C 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 [12] and Rosen [10], which better fits our setting.

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 consists of putting a in TMU. Note that a digraph can be 2-processed if and only if it is a $(2,a)$-digraph for some vertex a. First, we show how to determine whether D is a $(2,a)$-digraph.

Lemma 9. *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 $p(D-a)\leq 2$. So the charaterization 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' 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, there are no arcs from $H+a$ to $H'-a'$, and so condition (ii) is fulfilled.

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

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

Algorithm 1 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 neighborhood of its predecessors, $N^-(a)$
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**

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 in 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 10. *Let D be a digraph and v a vertex of D with exactly one outneighbor u. Let D' be obtained by contracting arc vu into 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 11. *Given a strongly-connected digraph D and a vertex $a \in V(D)$, Algorithm 1 decides in time $O(n(n+m))$ if (D,a) is a $(2,a)$-digraph.*

Proof. Let us prove that Algorithm 1 is correct. We assume that each time a vertex is processed, the neighborhood 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 is a digraph with process number 2. We consider the partition (H, H') of $D-a$ and the subset $Y \subseteq V(H)$ given by Lemma 9. We set $Y' := V(H) \setminus Y$. If $p(H) = 0$, then we may assume that $Y = V(H)$, and hence $Y' = \emptyset$.

Algorithm 2 Function Contract-rooted

Require: a connected digraph D, a vertex a and the set V_D^1 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_D^1 \ {a} is not empty do
2:   Let u be any vertex of V_D^1 − {a}, which we remove from V_D^1
3:   if N^+(u) > 0 then
4:     Let v be the outneighbor of u
5:     for all w ∈ N^−(u) do
6:       N^+(w) ← N^+(w) \ {u} ∪ {v}
7:       if |N^+(w)| = 1 then
8:         V_D^1 ← V_D^1 ∪ {w}
9:       end if
10:    end for
11:  end if
12: end while
```

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 2 will contract Y^r into a', starting from the leaves.

Now, Algorithm 1 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 9, and in the latter case, it would mean that H' is not a $(2, a')$-digraph by Lemma 10, a contradiction. Therefore, Algorithm 1 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 10, 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 10.

The computation time of Algorithm 1 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 16 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 2 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 1 is in $O(n(n+m))$. □

We note that Algorithm 2 can me modified to decide if a strongly connected digraph D can be 1-processed [6], 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 11.

Proposition 12. *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).

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 15, Algorithm 3), as well as a characterization in terms of excluded minors and a structural description (Theorem 5). For digraphs with process number two, we found a characterization that allows to recognize (and process) them in time $\mathcal{O}(n^2(n+m))$. Finally, we linked the process number to the connectivity, by determining the graphs with process number equal to their connectivity (Theorem 2).

As for the excluded minor characterization, we are currently studying [4] graphs with process number 3. It may be the last case achievable, since we have so far a list of 185 266 forbidden minors. It is interesting to note that for the pathwidth, such a characterization has been found up to pathwidth three [8] — 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] H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

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

[3] 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.

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

[5] 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.

[6] 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.

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

[8] 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. Efficient algorithms and partial k-trees.

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

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

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

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

A Recognition of graphs with process number at most two

We show in Proposition 15 that deciding whether a graph can be 2-processed can be done in linear time. To this end, we first note in Proposition 13 that we can decide very efficiently if a graph can be 1-processed, and in Proposition 14 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 $u.\textsc{active}$ set to FALSE if the vertex is in TMU or if it has been processed, and an integer — or a pointer — $u.\textsc{tag}$, 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 5) is linear in the size of G.

Proposition 13. *Given a graph G and a vertex u, we can decide in time $O(\deg(u)+\deg(v)-1)$ 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 $\deg(u) > 1$, then every neighbor of u has degree one. This can be checked in time $O(\deg(u))$;
- if $\deg(u) == 1$, then the unique neighbor v of u cannot have neighbors of degree larger than one, which can be checked in time $O(\deg(v) - 1)$.

So, overall, the time complexity is $O(\deg(u) + \deg(v) - 1)$. □

Proposition 14. *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 2, 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 15. *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 that can be 2-processed and the vertex a_1 of condition (c) of Theorem 5, then we can process it in linear time. To this end, let us analyze the algorithm described in the proof of Theorem 5.

- Put vertex a_i on TMU: we just have to set the Boolean variable $a_i.\textsc{active}$ 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 13 we can decide whether a neighbor u of a_i belongs to a star or not in time $O(\deg(u) + \deg(v) + 1)$, 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 $O(\deg(u)+\deg(v)+1)$, 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 proceed 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, if any left. 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 can be 2-processed if and only if this algorithm processes the whole graph. Note that the algorithm fails if more than one vertices are candidates to be the vertex a_{i+1}.

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

(2) According to the previous step and Corollary 6, given a graph G and a vertex a_i, we can check in linear time if G can be 2-processed or not. For that we proceed all subgraphs of G attached to a_i that can be 1-processed. Now $G - \{a_i\}$ must contains 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, add them a_i plus the required edges, and apply step (1) on H from a_i. If step (1) succeeds and if the vertices of H' have not yet been processed then apply step (1) on H' from a_i to know if G can be 2-processed.

If some vertices of H' were processed during the processing of H, then we can conclude that G contains a cycle of length at least 5 and so that it cannot 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 5. 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 $\deg(u) > 2$, 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 at distance one and two from u, respectively. Now, 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 5.

- 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. Suppose that 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. Thus, in the subgraph of G induced by removing the vertices a_i, it belongs to a component that cannot be 1-processed, a contradiction. The same 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. □

B Algorithm to 2-process graphs

A precise description of an algorithm to recognize graphs with process number 2 (and obtain a 2-strategy, if any) is given by Algorithms 3, 4, 5, 6 and 7.

Algorithm 3 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.

a.ACTIVE $\leftarrow$ FALSE
$CC_2 \leftarrow$ FALSE {CC_2 indicates if a connected component that cannot be 1-processed has already been found.}
for all $v \in N(a)$ such that v.ACTIVE and v.TAG $\neq a$ **do**
  if Is-Star(G, v, a, a) **then**
    Process-Star(G, v, a)
  else if not CC_2 **then**
    $CC_2 \leftarrow$ TRUE
  else
    return FAILED
  end if
end for
if not CC_2 **then**
  return SUCCEED
end if
$FC \leftarrow$ FALSE {FC indicates if we have found a candidate for the next vertex to be visited, a_{i+1}}
for all $v \in N(a)$ such that v.ACTIVE **do**
  if not FC **then**
    if $\deg(v) = 2$ **then**
      $a' \leftarrow N(v) - \{a\}$
    else
      $a' \leftarrow v$
    end if
    $FC \leftarrow$ TRUE
  else if ($\deg(v) = 2$ and $N(v) - \{a\} \neq \{a'\}$) or ($\deg(v) > 2$ and $v \neq a'$) **then**
    return FAILED
  end if
  v.ACTIVE $\leftarrow$ FALSE
end for
return Test-2-process-from(G, a')

Algorithm 4 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 ← u
2: if deg(u) − χ(a, u) = 1 then
3:    c ← N(u) − {a}
4: end if
5: c.TAG ← a
6: bool ← TRUE
7: for all v ∈ N(c) − {a} do
8:    if deg(v) − χ(a, v) > 1 then
9:       bool ← FALSE
10:   end if
11:   v.TAG ← t
12: end for
13: return bool
```

Algorithm 5 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 ← u
2: if deg(u) − χ(a, u) = 1 then
3:    c ← N(u) − {a}
4: end if
5: for all v ∈ N(c) − {a} do
6:    v.ACTIVE ← FALSE
7: end for
```

Algorithm 6 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.}

Let u be a vertex of G
for all $v \in V(G)$ **do**
  if $\deg(u) < \deg(v)$ **then**
    $u \leftarrow v$
  end if
end for
{Then we count the number of vertices of degree ≥ 3 at distance one and two.}
if $\deg(u) \geq 3$ **then**
  $k_1 \leftarrow 0$, $k_2 \leftarrow 0$
  for all $v \in N(u)$ **do**
    if $\deg(v) \geq 3$ **then**
      $x_1 \leftarrow v$
      $k_1 \leftarrow k_1 + 1$
    end if
  end for
  for all $v \in N(N(u)) - \{u\}$ **do**
    if $\deg(v) \geq 3$ **then**
      $x_2 \leftarrow v$
      $k_2 \leftarrow k_2 + 1$
    end if
  end for
  {Finally we decide whether u, x_1 or x_2 is of type a_i.}
  if $k_1 = 1$ and Is-Star(G, u, x_1) **then**
    $u \leftarrow x_1$
  else if $k_1 = 0$ and $k_2 = 1$ and Is-Star(G, u, x_2) **then**
    $u \leftarrow x_2$
  end if
end if
return u

Algorithm 7 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 χ 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*

C Recognition of digraphs with process number one

We give in Proposition 16 an alternative to the algorithms of Shamir [12] and Rosen [10], which better fits our setting. It decides in linear time and space complexity if a digraph can be 1-processed. It follows that we can compute the minimum feedback vertex set of 1-process digraphs in linear time.

Proposition 16. *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 $\mathcal{O}(n+m)$ using a topological sort [2] 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 $\mathcal{O}(n+m)$.

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.

Observe that if there exists a directed path $x_i, y_1, y_2, \dots, y_\ell, x_j$ with $\{y_1, y_2, \dots, y_\ell\} \cap V(C) = \emptyset$, then none of the vertices $x_{i+1}, x_{i+2}, \dots, x_{j-1}$ can be the sought vertex v. (We allow here ℓ to be 0, in which case it means that there is an arc from x_i to x_j.) Furthermore, if there exist two directed paths $P_1 := x_iy_1\dots y_\ell x_k$ and $P_2 := x_jz_1\dots z_{\ell'}x_k$ both internally disjoint from C (but P_1 and P_2 need not be disjoint) such that $j < k < i$, then none of the vertices $x_j, x_{j-1}, \dots, x_1, x_k, x_{k-1}, \dots, x_{i+1}$ can be the vertex v.

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 $\mathcal{L}[l] := i$ for each l from $j-1$ down to $i+1$. 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 $\mathcal{L}[l] := i$ 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 in $O(n+m)$.

When all the DFS are performed, then it remains to choose a valid vertex x_i in $\mathcal{L}$ and to check if $D - \{x_i\}$ is a DAG. If $\mathcal{L}$ contains no valid vertices, then we can conclude that D can not be 1-processed and that $D - C$ contains a cycle.

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 DFS. So this part take time $O(m)$. Also, the total cost due to $\mathcal{L}$ is $O(n+m)$. Finally, we can check if $D - \{x_i\}$ 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 DFS 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. □