On the Monotonicity of Process Number

Nicolas Nisse, Ronan Pardo Soares

On the Monotonicity of Process Number

Nicolas Nisse[a,b], Ronan Pardo Soares[a,b,c,*]

[a] *Inria, France*
[b] *Univ. Nice Sophia Antipolis, CNRS, I3S, UMR 7271, 06900 Sophia Antipolis, France*
[c] *ParGO - Univ. Federal do Ceará, Brazil*

Abstract

Graph searching games involve a team of searchers that aims at capturing a fugitive in a graph. These games have been widely studied for their relationships with the tree- and the path-decomposition of graphs. In order to define decompositions for directed graphs, similar games have been proposed in directed graphs. In this paper, we consider a game that has been defined and studied in the context of routing reconfiguration problems in WDM networks. Namely, in the *processing game*, the fugitive is invisible, arbitrarily fast, it moves in the opposite direction of the arcs of a digraph, but only as long as it can access to a strongly connected component free of searchers. We prove that the processing game is monotone which leads to its equivalence with a new digraph decomposition.

Keywords: Graph Searching, Process Number, Monotonicity

1. Introduction

During the last few years, an important research effort has been done in order to design digraph decompositions that are as powerful as the path-decomposition or the tree-decomposition in the case of undirected graphs (e.g., see [1]). Because graph searching games are equivalent to path- and tree-decompositions in undirected graphs, several proposals have been done to define such games in directed graphs [2, 3, 4]. In this paper, we study the *processing game* and show its equivalence with a new digraph decomposition.

1.1. Graph Searching and monotonicity

In graph searching games, a team of *searchers* aims at capturing a fugitive that stands at the vertices of a connected graph G (see [5] for a survey). Initially, no searchers are occupying the vertices of G and any vertex may host the fugitive. The vertices are initially said *contaminated*. The fugitive can move by

*Corresponding author. Phone: +558594846666, Fax: +558594846666
Email addresses: nicolas.nisse@inria.fr (Nicolas Nisse), ronan.soares@gmail.com (Ronan Pardo Soares)

following the paths of G while it does not meet any searcher. The fugitive is arbitrarily fast in the sense that its moves are instantaneous whatever be the length of the paths it follows. A *strategy* for the searchers is a sequence of the following two possible actions: $(Place(v))$ place a searcher at a vertex v of G, or $(Remove(v))$ remove a searcher from a vertex v. When a searcher occupies a vertex, this vertex becomes *clear*. However, a clear vertex v is *recontaminated* if it is not occupied and there is a path without searchers from v to a contaminated vertex. In other words, a vertex $v \in V(G)$ remains clear if all paths from v to a contaminated vertex contain a vertex occupied by a searcher. In particular, a vertex occupied by a searcher is clear.

A strategy is *winning* if it allows to capture the fugitive whatever it does or, equivalently, if all vertices are eventually clear. That is, in a winning strategy, a searcher eventually occupies the same vertex as the fugitive and the fugitive cannot move anymore (i.e., all the neighbors of its current position are occupied by searchers). The number of searchers used by a strategy is the maximum number of occupied vertices throughout all steps of the strategy. Any graph G has a trivial winning strategy that consists in placing a searcher at every vertex of G. Such a strategy uses n searchers where n it the number of vertices of G. A natural question is then to design a winning strategy using the lowest number of searchers. The *search number* of a graph G is the smallest integer $k \geq 1$ such that there is a winning strategy using k searchers in G. As an example, consider the cycle with $n \geq 3$ vertices $\{v_1, \cdots, v_n\}$. A possible strategy is as follows: first three searchers are placed at v_1, v_2 and v_3. Then, for $i = 2$ to $n-2$, remove the searcher at v_i and place it at v_{i+2}. This strategy is winning and uses 3 searchers and it is easy to see that there are no winning strategy using at most two searchers in the cycle. Hence, the search number of any cycle with at least three vertices is 3.

There are many variants of this game arising due to different properties, or behaviors, given to the fugitive or to the searchers. For instance, the fugitive may be visible, i.e., its location is known at any moment of the game and the strategy may take advantage of this knowledge, or it may be invisible, i.e., the fugitive may be at any contaminated vertex. If the fugitive is visible, the corresponding search number of a graph equals its *tree-width* plus one [6]. On the other hand, if the fugitive is invisible, the search number is equal to the *path-width* plus one [7]. The relationship between graph decompositions and search strategies mainly relies on the *monotonicity* property of these variants of graph searching. A strategy is said *monotone* if the area reachable by the fugitive is never increasing, i.e., once a vertex is clear it never becomes contaminated anymore. Equivalently, in the case of an invisible fugitive, a searcher cannot be removed from a vertex if it has a neighbor that is neither occupied nor has been occupied before, i.e., once a vertex has been occupied, the fugitive must not be able to reach it anymore. A variant of graph searching is said *monotone* if "recontamination does not help", i.e., for any graph with search number k, there is a winning monotone strategy using k searchers. That is, the number of searchers necessary to capture a fugitive considering only monotone strategies is not bigger than without this consideration.

The visible and invisible variants of graph searching were proven to be monotone in [8] (invisible) and [6] (visible). A simpler proof in the invisible case has been proposed by Bienstock and Seymour [9], and a unified proof for both visible and the invisible case can be found in [10]. Note that there are graph searching variants that are not monotone in undirected graphs, i.e., imposing the monotonicity of strategies may increase the number of searchers required to capture the fugitive. In *connected graph searching*, the area that cannot be reached by the fugitive is restricted to be connected along all stages of the strategy. The connected graph searching variant has been proved to be not monotone when the fugitive is invisible [11] neither when the fugitive is visible [12].

1.2. Graph searching in directed graphs

In [13], Johnson *et al.* defined the first variant of graph searching in directed graphs, related to directed tree-width. This variant, where the visible fugitive can move along directed cycles that are free of searchers, is however not monotone [14]. In [4], a variant is proposed where the visible fugitive can move along directed paths without searchers and [15] defined a variant where the invisible fugitive can move along directed paths without searchers only when a searcher is about to land at the vertex that the fugitive is currently occupying. Both these variants, respectively related to DAG-width and Kelly-width, are not monotone [15].

In some other cases, considering an invisible fugitive, more positive results have been provided. Barát defined the *directed path-width* related to a graph searching variant where the invisible fugitive is constrained to follow the direction of the arcs, i.e., it can move along directed paths free of searchers [2]. Barát adapted the framework of Bienstock and Seymour [9] to show that, in this variant, the monotonicity cannot increase the number of searchers by more than one [2]. Hunter then completed this proof to show the monotonicity of this variant [16]. Other variants that generalize the *edge-graph searching* (see [5]) to directed graphs have been defined. Yang and Cao proved the monotonicity of strong and weak graph searching variants where the searchers can moreover slide along arcs either in both directions (strong) or in the direction of arcs (weak) [17, 18].

1.3. Process Number

Surprisingly, a variant of graph searching in directed graphs has been defined in the context of routing reconfiguration in Wavelength-Division Multiplexing (WDM) networks [19]. WDM networks are currently becoming more flexible, offering new on-demand services for provisioning new light-paths, but also allowing better management of maintenance operations and equipment failures. A building block for flexibility and reliability is the possibility to *reconfigure* the routing, that is to compute new optical paths for some connection requests and then to switch the traffic from former to new optical paths. Such process may however affect the quality of service by inducing potential traffic disruptions. Thus, the routing reconfiguration process must be carefully optimized (see, e.g., [20, 21, 22, 23]).

An instance of the routing reconfiguration problem is defined by a network, a set of connections, an initial routing and a final routing. The networks is represented by a directed graph N. The set of connections is given by $C \subseteq V(N) \times V(N)$. An initial routing of these connections is given by a set I of directed paths in N joining each pair $(a, b) \in C$, with the restriction that two different paths do not share an arc of N, that is, these paths are arc-wise disjoint. Similarly, the final routing F is a set of arc-wise disjoint paths joining each pair $(a, b) \in C$. The *routing reconfiguration problem* consists in sequentially reroute the connections from the initial routing to the final one and minimizing the traffic disruption.

There are two ways to reroute a connection $r \in C$ from its initial route $P \in I$ to its final one $Q \in F$. Either connection r is first interrupted and then it is established on its final route Q when all resources (i.e., all arcs) of Q are available. Or the final route of r is first established (again, all resources of Q must be available) and then the initial route of r is turned off. The first way is referred to as *Break-before-Make* and the latter one as *Make-before-Break*. Clearly, routing reconfiguration is always possible: first, all connections are interrupted and then, sequentially, the final route of each connection is established. Proceeding this way, all connections are rerouted in a Break-before-Make manner and thus all connections are interrupted. The problem is to find a better rerouting strategy, i.e., a strategy with less traffic disruption.

In [21], Jose and Somani studied the problem of minimizing the total number of interruptions. In [19], Coudert *et al.* considered the problem of minimizing the maximum number of connections that are simultaneously interrupted. For this purpose, they introduced a new game on digraphs, namely the *processing game*, which is similar to graph searching games.

Following the terminology of [19], the vertices of a digraph may be either *processed* or *unprocessed*. We say that a vertex is *processed* if it was unprocessed and becomes processed. In the *processing game*, a team of searchers aims at *processing* all vertices of a digraph. A vertex is said *safe* if all its out-neighbors are either occupied by a searcher or already processed. Given a digraph $D = (V, A)$ where initially all vertices are unoccupied and unprocessed, a *monotone process strategy* is a sequence $(s_1, \ldots, s_n)$ of steps that results in processing all vertices of D, where each step s_i is one of the following three moves.

$Place(v)$**:** place a searcher at vertex $v \in V$;

$Process(v)$**:** process a safe *unoccupied* vertex $v \in V$;

$Remove_{monot}(v)$**:** process a safe *occupied* vertex $v \in V$ and remove the searcher from it.

The minimum number of searchers used by a monotone process strategy of D is the *monotone process number*, denoted by $\mathrm{monpr}(D)$.

The routing reconfiguration problem is actually equivalent to the processing game in some particular digraph [19, 24]. Given an instance of the routing reconfiguration problem, the *dependency digraph* $D = (V, A)$ is defined as follows.

The digraph D has one vertex per connection (hence, we may identify the vertices of D and the connections of the instance of the rerouting problem). For any $a, b \in V$, there is an arc from a to b if the final route of connection a uses arcs that belong to the initial route of connection b. Note that, if $(a, b) \in A$, then connection a cannot be rerouted while the initial route of b is still established. There is a one-to-one mapping between the rerouting strategies and the monotone process strategies for D. Indeed, placing a searcher at some vertex of D corresponds to the interruption of the corresponding connection and processing a vertex corresponds to the establishment of the final route of the connection. Processing an unoccupied vertex corresponds to reroute the connection without interruption, i.e., in a Make-before-Break manner, and processing an occupied vertex corresponds to the Break-before-Make manner. Note that only safe vertices may be processed. This comes from the fact that, if a vertex v is safe, it means that all routes that use resources required by the final route of v have been turned off (all out-neighbors of v are either occupied or processed) and then the final route of v can be established. Finally, the number of searchers used during the process strategy is exactly the maximum number of connections that are simultaneously interrupted.

Besides its interest for the routing reconfiguration problem, the process number has a close relationship with other graph parameters. Recall that the vertex separation is a digraph parameter that coincides with the pathwidth in undirected graphs [25]. For any digraph D, $vs(D) \leq \text{monpr}(D) \leq vs(D) + 1$ where $vs(D)$ is the *vertex separation* of D [19]. Moreover, to compute the monotone process number is NP-hard in general but it is polynomial in the class of graphs D with $\text{monpr}(D) \leq 2$ [26] and in the class of trees [27]. The relationship between the monotone process number of a digraph and its minimum feedback vertex set has been studied in [24]. The process number of digraphs has been mainly studied for its applications in the rerouting problem in WDM networks [28, 29, 30]. Note also that, in undirected graphs (seen as symmetric digraphs[1]), the monotone processing game is equivalent to the monotone graph searching game where the invisible fugitive is captured if all the neighbors of its position are occupied, i.e., it is not anymore required that a searcher occupies the same vertex as the fugitive. More generally, in the processing game, a team of searchers aims at capturing an invisible and arbitrarily fast fugitive in a digraph, where the fugitive is constrained to move in the opposite direction of the arcs of a digraph and it is captured as soon as the set of vertices it can access induces an acyclic digraph.

It is important to notice that the processing game when played on a symmetric digraph is not equivalent to the graph searching games with an invisible fugitive. Given a symmetric directed graph D the following inequality is true: $\text{s}(\bar{D}) - 1 \leq \text{monpr}(D) \leq s(\bar{D})$, where $\bar{D}$ denotes the underlying graph of D and $\text{s}(\bar{D})$ denotes the minimum number of searchers necessary to capture an invisi-

[1] A digraph $D = (V, A)$ is *symmetric* if, for any $(a, b) \in A$, then $(b, a) \in A$.

ble fugitive in $\bar{D}$. Moreover, each of these bounds may be reached as shown by the following examples. Let K_n be the complete directed graph with n vertices, then $\mathrm{s}(\bar{K_n}) = \mathrm{monpr}(K_n) + 1$ and, for every directed graph D let D' be the directed graph obtained from D by adding a loop to every vertex of D, then $\mathrm{s}(\bar{D'}) = \mathrm{monpr}(D')$.

Let D be any directed graph, $\mathrm{cn}(D)$ denotes the number of searchers necessary to capture an invisible fugitive that can only move in the direction of the arcs as defined in [2]. Then, $\mathrm{cn}(D) - 1 \leq \mathrm{monpr}(D) \leq \mathrm{cn}(D)$ for any directed graph D. The same examples as above show that both bounds may be reached.

In this work, we consider the more general variant of processing game that may be not monotone. That is, we allow a processed vertex to become unprocessed. More precisely, a *process strategy* for a digraph D is a sequence $(s_1, \ldots, s_n)$ of steps that results in processing all vertices of D, where each step s_i consists of a move $Place$ or $Process$ or

Remove(v)**:** process an *occupied* vertex $v \in V$ and remove the searcher from it. If v was not safe then recontamination occurs: successively, all processed vertices (including v) that have an unoccupied and unprocessed out-neighbor become unprocessed.

The minimum number of searchers used by a process strategy of D is the *process number*, denoted by $\mathrm{pr}(D)$. In [27], it is proved that $\mathrm{pr}(D) = \mathrm{monpr}(D)$ for any symmetric digraph D. In this work, we prove that the result holds for any digraph. Moreover, our monotonicity result allows us to prove that $\mathrm{pr}(D) = \mathrm{pr}(D^T)$ for any digraph $D = (V, A)$, where $D^T = (V, A^T)$ and the set $A^T = \{(a, b) \ : \ (b, a) \in A\}$.

2. Recontamination does not help to process a digraph

In this section, we present the result that the process number is monotone, i.e. $\mathrm{monpr}(D) = \mathrm{pr}(D)$ for any directed graph D. For this purpose, we use the techniques introduced in [6] and adapted for directed graphs in [2, 17, 18]. More precisely, we first define the notion of mixed processing game and show its monotonicity thanks to an intermediate result dealing with crusades (see definition below). Then, from any mixed process strategy we construct a process strategy with the same number of searchers in a way that monotonicity is preserved.

2.1. Preliminaries

Throughout this section, we use the following notation. Let $D = (V, A)$ be a digraph. For any $v \in V$, let $N^-(v)$ denote the set of in-neighbors of v. For any set $X \subseteq A$, let $\delta(X)$ denote the set of vertices that are the head of an arc in X and the tail of an arc in $A \setminus X$. The *border* of a set $X \subseteq A$ is $|\delta(X)|$. For any $X \subseteq A$, X^c denotes $A \setminus X$. First, we show that the border function δ is submodular.

Lemma 1. *For any digraph D and any $X, Y \subseteq A(D)$:*

$$|\delta(X \cap Y)| + |\delta(X \cup Y)| \leq |\delta(X)| + |\delta(Y)|.$$

Proof. We show that every vertex counted in the left side of the equation is counted at least the same amount of times in the right side of the equation. Let $v \in \delta(X \cup Y) \cup \delta(X \cap Y)$.

If $v \in \delta(X \cap Y)$, let $e_1 = (u, v) \in X \cap Y$ and $e_2 = (v, w) \in X^c \cup Y^c$. Therefore, either $(v, w) \in X^c$ and $v \in \delta(X)$, or $(v, w) \in Y^c$ and $v \in \delta(Y)$. If $v \in \delta(X \cup Y)$, let $e_1 = (u, v) \in X \cup Y$ and $e_2 = (v, w) \in X^c \cap Y^c$. Therefore, either $(u, v) \in X$ and $v \in \delta(X)$, or $(u, v) \in Y$ and $v \in \delta(Y)$. Finally, let us assume that $v \in \delta(X \cup Y) \cap \delta(X \cap Y)$. Because $v \in \delta(X \cap Y)$, there exists an edge $e_1 = (u, v) \in X \cap Y$ and because $v \in \delta(X \cup Y)$, there exists an edge $e_2 = (v, w) \in X^c \cap Y^c$. Hence, $v \in \delta(X) \cap \delta(Y)$. □

Process strategies as defined in Section 1.3 aim at *processing* sequentially the vertices of a digraph. Following the techniques of [6], we first need to define strategies that process the arcs of a digraph. In what follows, an arc can be processed in three ways (rules $Head$, $Slide$ or $Extend$ described below).

Let $D = (V, A)$ be a digraph whose arcs are initially unprocessed. A *mixed process strategy* for D is a sequence $(s_1, \ldots, s_n)$ with the following rules that results in processing all arcs in A.

$Place(v)$**:** place a searcher at an unoccupied vertex $v \in V$;

$Remove(v)$**:** remove a searcher from vertex $v \in V$; if there were unprocessed arcs with tail v and v is now unoccupied, then *recontamination* occurs. That is, successively, any processed arc $(u, w) \in A$ such that w is unoccupied and there is an unprocessed arc (w, z) becomes unprocessed.

$Head(u, v)$**:** process an arc $(u, v) \in A$ if $v \in V$ is *occupied*;

$Slide(u, v)$**:** slide the searcher at u along $(u, v) \in A$ and process the arc (u, v). This rule can be applied if u is occupied, v is not occupied and all arcs $e \neq (u, v)$ with tail u are already processed;

$Extend(u, v)$**:** process an arc $(u, v) \in A$ if all arcs with tail v are already processed.

The number of searchers used by a mixed process strategy is the maximum number of occupied vertices over all steps of the strategy. The *mixed process number*, denoted by $\mathrm{mpr}(D)$, is the lowest number of searchers used by such a strategy. Moreover, in the mixed process number game, we say that a vertex, v, is processed if all arcs with tail v are processed. A mixed process strategy is *monotone* if no recontamination occurs, i.e., once an arc has been processed, it must remain processed until the end of the strategy.

Next, we recall the definition of crusades used in [2] and give these crusades an appropriate border function to work with the mixed process number game.

A *crusade* in $D = (V, A)$ is a sequence $(X_0, X_1, \ldots, X_n)$ of subsets of A such that $X_0 = \emptyset$, $X_n = A$, and $|X_i \setminus X_{i-1}| \leq 1$, for $1 \leq i \leq n$. The crusade has *border* k if $|\delta(X_i)| \leq k$ for $0 \leq i \leq n$.

A crusade is *progressive* if $X_0 \subset X_1 \subset \ldots \subset X_n$. Hence, in a progressive crusade $(X_0, X_1, \ldots, X_n)$, $|X_i \setminus X_{i-1}| = 1$ for all $1 \leq i \leq n$.

The notion of mixed strategies and crusades are not new, they were already used in [6, 2] to show that some pursuit-evasion games are monotone. In this work we use a the notions of mixed strategy and crusade that closely resemble the ones in [2]. Our definitions of mixed strategy and of crusade follow the same set of rules as the ones in [2], however our border function differs. Our border function δ is similar to the one in [2], but the direction of arcs are reversed. More precisely, in [2], the border function of a set X is the set of vertices that are a tail of an arc in X and the head of an arc in $A \setminus X$.

2.2. Monotonicity

Lemma 2. *Let D be a digraph. If $\mathrm{mpr}(D) \leq k$, then D admits a crusade with border k.*

Proof. Let $S = (s_1, \ldots, s_n)$ be a mixed process strategy of $D = (V, A)$ that uses at most k searchers. For any $0 < i \leq n$, let A_i be the set of processed arcs and Z_i be the set of occupied vertices after step s_i. Moreover, let $A_0 = Z_0 = \emptyset$.

By definition of a mixed process strategy, at most one arc is processed in each step s_i (one arc a is processed if s_i corresponds to $Head(a)$, $Slide(a)$ or $Extend(a)$), hence $|A_i \setminus A_{i-1}| \leq 1$ for any $1 \leq i \leq n$. After the last step s_n of S, all the arcs of the graph must be processed, hence $A_n = A$. This proves that $C = (A_0, \ldots, A_n)$ is a crusade.

It remains to show that $\delta(A_i) \leq k$ for every $0 \leq i \leq n$. To do so, we prove by induction that $\delta(A_i) \subseteq Z_i$ for any $1 \leq i \leq n$. This is clearly true for $i = 0$. Assume that $\delta(A_{i-1}) \subseteq Z_{i-1}$ for some i, $0 \leq i < n$. We prove that $\delta(A_i) \subseteq Z_i$:

- If s_i is $Place$, then $A_i = A_{i-1}$ and thus $\delta(A_i) = \delta(A_{i-1}) \subseteq Z_{i-1} \subseteq Z_i$.
- If s_i is $Remove$, let u be a vertex of $\delta(A_i)$, hence there is an arc $e_1 = (w_1, u) \in A_i$ and an arc $e_2 = (u, w_2) \in A \setminus A_i$, therefore $u \in Z_i$, otherwise e_1 would also become unprocessed in step i making $u \notin \delta(X_i)$, hence $\delta(A_i) \subseteq Z_i$.
- If s_i is $Head(u, v)$, then $A_i = A_{i-1} \cup \{(u, v)\}$ and $\delta(A_i) \setminus \delta(A_{i-1}) \subseteq \{v\}$. Since v must be occupied, we have $v \in Z_i = Z_{i-1}$, by induction $\delta(A_{i-1}) \subseteq Z_{i-1}$, and therefore $\delta(A_i) \subseteq Z_i$.
- If s_i is $Slide(u, v)$, then $A_i = A_{i-1} \cup \{(u, v)\}$ and $Z_i = (Z_{i-1} \setminus \{u\}) \cup \{v\}$. Since all arcs with tail u are processed after this step, $u \notin \delta(A_i)$. Moreover, $\delta(A_i) \setminus \delta(A_{i-1}) \subseteq \{v\}$. Hence $\delta(A_i) \subseteq Z_i$.
- If s_i is $Extend(u, v)$, then $A_i = A_{i-1} \cup \{(u, v)\}$ and $Z_i = Z_{i-1}$. Since all arcs with tail v must be already processed, $\delta(A_i) = \delta(A_{i-1}) \subseteq Z_{i-1} = Z_i$.

□

Lemma 3. *If there is a crusade of $D = (V, A)$ with border k, then there is a progressive crusade with border k.*

Proof. Let $C = (X_0, \ldots, X_n)$ be a crusade of D with border k such that $\sum_{i=0}^{n} |\delta(X_i)|$ is minimum, and subject to this, $\sum_{i=0}^{n} |X_i|$ is minimum. We show that C is progressive. Let $0 < i \leq n$, we show that $X_{i-1} \subset X_i$:

1. Assume first that $|X_i \setminus X_{i-1}| = 0$, then $X_i \subseteq X_{i-1}$.
   Hence, $(X_0, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n)$ is a crusade with border k, contradicting the minimality of $\sum_{i=0}^{n} |X_i|$. Thus, $|X_i \setminus X_{i-1}| = 1$.
2. Then assume that $|\delta(X_{i-1} \cup X_i)| < |\delta(X_i)|$.
   Hence $(X_0, \ldots, X_{i-1}, X_{i-1} \cup X_i, X_{i+1}, \ldots, X_n)$ is a crusade with at most k searchers, contradicting the minimality of $\sum_{i=0}^{n} |\delta(X_i)|$.
   Therefore $|\delta(X_{i-1} \cup X_i)| \geq |\delta(X_i)|$.
3. By Lemma 1, $|\delta(X_{i-1} \cap X_i)| + |\delta(X_{i-1} \cup X_i)| \leq |\delta(X_{i-1})| + |\delta(X_i)|$.
   Hence, by item (2), $|\delta(X_{i-1} \cap X_i)| \leq |\delta(X_{i-1})|$.
   Therefore, $(X_0, \ldots, X_{i-2}, X_{i-1} \cap X_i, X_i, \ldots, X_n)$ is a crusade with at most k searchers.
   From the minimality of $\sum_{i=0}^{n} |X_i|$ we have that $|X_{i-1} \cap X_i| \geq |X_{i-1}|$, hence $X_{i-1} \subseteq X_i$.

□

Lemma 4. *If there is a progressive crusade of $D = (V, A)$ with border k, then there is a* monotone *mixed process strategy using at most k searchers.*

Proof. Let $C = (X_0, \ldots, X_n)$ be a progressive crusade of D with border k. We build a a monotone mixed process strategy $S = (s_1, \ldots, s_{n'})$ of D with the following properties. For any $0 < i \leq n'$, let A_i be the set of processed arcs and let Z_i be the set of occupied vertices after step s_i. Let $A_0 = Z_0 = \emptyset$. There are $0 = j_0 < j_1 < j_2 < \cdots < j_n = n'$ such that:

1. for any $0 \leq i \leq n$, $A_{j_i} = X_i$;
2. for any $0 < i \leq n$ and for any $j_{i-1} < \ell < j_i$, $Z_\ell \subseteq \delta(X_i)$ or $Z_\ell \subseteq \delta(X_{i-1})$, and $Z_{j_i} = \delta(X_i)$.

Starting with $S = \emptyset$, Properties (1) and (2) hold for $i = 0$. Let $0 < i \leq n$ and let us assume that $(s_1, \ldots, s_{j_{i-1}})$ is a sequence of rules that satisfies Properties (1) and (2) for any $0 \leq j < i$. We will build the next steps of the strategy until s_{j_i}. Let $X_i \setminus X_{i-1} = \{e_i\}$, where $e_i = (u, v)$. Note that $\delta(X_i) \setminus \delta(X_{i-1}) \subseteq \{v\}$ and $\delta(X_{i-1}) \setminus \delta(X_i) \subseteq \{u, v\}$. We have several cases to consider:

- let us first assume that $v \in \delta(X_{i-1})$. Hence, $v \in Z_{j_{i-1}}$ and there is a searcher at v after step $s_{j_{i-1}}$. We define step $s_{j_{i-1}+1}$ to be $Head(e_i)$, i.e., the arc e_i is processed.
  - If moreover $v \notin \delta(X_i)$ then we define step $s_{j_{i-1}+2}$ to be $Remove(v)$, i.e., we remove the searcher at v. Because $v \notin \delta(X_i)$ and (u, v) is processed, there are no unprocessed arcs with tail v and therefore, no recontamination occurs.

Let $k = j_{i-1} + 3$ if $v \notin \delta(X_i)$ and $k = j_{i-1} + 2$ otherwise.

- Finally, if $u \in \delta(X_{i-1}) \setminus \delta(X_i)$, then we define step s_k to be $Remove(u)$, i.e., we remove the searcher at u. Because $u \in \delta(X_{i-1})$, there is an arc with head u that was processed after step $s_{j_{i-1}}$. Because $u \notin \delta(X_i)$ and (u, v) is processed, there are now no unprocessed arcs with tail u and therefore, no recontamination occurs.

Hence, $j_{i-1} + 1 \leq j_i \leq j_{i-1} + 3$. Clearly, for any $j_{i-1} + 1 \leq \ell \leq j_{i-1} + 3$, $Z_\ell \subseteq \delta(X_{i-1})$ and $\delta(X_i) = Z_{j_i}$ in all cases. Moreover, in all cases, no recontamination occurs and then $A_{j_i} = X_i$.

- Now, let us assume that $v \notin \delta(X_{i-1})$. By induction, there was no searcher at v after step $s_{j_{i-1}}$.
  - First, let us consider the case when $u \in \delta(X_{i-1})$:
    - if $v \in \delta(X_i)$ and $u \in \delta(X_i)$: Let us define step $s_{j_{i-1}+1}$ to be $Place(v)$, i.e., a searcher is placed at v, and step $s_{j_i} = s_{j_{i-1}+2}$ is defined as $Head(e_i)$, i.e., the edge e_i is processed. Clearly, $A_{j_i} = A_{j_{i-1}} \cup \{e_i\}$ and $Z_{j_{i-1}+1} = Z_{j_i} = Z_{j_{i-1}} \cup \{v\} = \delta(X_i)$.
    - if $v \in \delta(X_i)$ and $u \notin \delta(X_i)$: in that case, the only arc in $A \setminus X_{i-1}$ which has u as tail is e_i, otherwise $u \in \delta(X_i)$. Therefore we define step $s_{j_{i-1}+1} = s_{j_i}$ to be $Slide(e_i)$, i.e., the searcher at u slides to v processing e_i. Note that no recontamination occurs and $A_{j_i} = A_{j_{i-1}} \cup \{e_i\} = X_{i-1} \cup \{e_i\} = X_i$. The induction hypothesis holds since $Z_{j_i} = (Z_{j_{i-1}} \setminus \{u\}) \cup \{v\} = (\delta(X_{i-1}) \setminus \{u\}) \cup \{v\} = \delta(X_i)$.
    - if $v \notin \delta(X_i)$ then there are no arcs with tail v that are in $A \setminus X_{i-1}$. Hence, we can define step $s_{j_{i-1}+1}$ to be $Extend(e_i)$, i.e., e_i is processed.
      If moreover $u \notin \delta(X_i)$, let $s_{j_i} = s_{j_{i-1}+2}$ be defined as $Remove(u)$, i.e., the searcher at u is removed. Because $u \in \delta(X_{i-1}) \setminus \delta(X_i)$ and (u, v) is now processed, there are no unprocessed arcs with tail u and therefore, no recontamination occurs.
      Hence, $j_{i-1} + 1 \leq j_i \leq j_{i-1} + 2$ and the induction hypothesis holds in both cases.
  - Finally, consider the case when $u \notin \delta(X_{i-1})$. Note that, in this case, since $u \notin \delta(X_{i-1})$ and u is a tail of $e_i \in X_i$, then $u \notin \delta(X_i)$.
    - if $v \in \delta(X_i)$ then we define step $s_{j_{i-1}+1}$ to be $Place(v)$, i.e., a searcher is placed at v, and $s_{j_i} = s_{j_{i-1}+2}$ to be $Head(e_i)$, i.e., e_i is processed. The induction hypothesis holds.
    - if $v \notin \delta(X_i)$, since $e_i \in X_i$, then there are no arcs with tail v that are in $A \setminus X_{i-1}$. Hence we can define step $s_{j_i} = s_{j_{i-1}+1}$ as $Extend(e_i)$, i.e., e_i is processed. The induction hypothesis holds.

Therefore, $S = (s_1, \ldots, s_{j_n})$ satisfies Properties (1) and (2), and S is a monotone mixed process strategy using at most k searchers in D, since, for all $1 \leq i \leq j_n$, we have that $|Z_i| \leq k$, for all $1 \leq i < j_n$, $A_i \subseteq A_{i+1}$, and $A_{j_n} = X_n = A$. □

In what follows, let $\mathrm{mpr}(\ddot{D})$ be the digraph obtained from any digraph D by duplicating every arc of D.

Lemma 5. *For any digraph D, $\mathrm{mpr}(\ddot{D}) \leq \mathrm{pr}(D)$.*

Proof. Let $S^p = (s_1, \ldots, s_n)$ be a process strategy for D using k searchers. We define a mixed process strategy $S^m = (m_1, \ldots, m_j)$ using at most k searchers for $\ddot{D}$. Let P_i be the set of processed vertices at step $i \leq n$ in S^p and let M_j be the set of vertices u such that, at step m_j in S^m, all arcs with u as tail are processed. Also, let O_i^p (resp., O_i^m) be the set of vertices occupied by a searcher at step s_i in S^p (resp., at step m_i in S^m).

For any $0 < i \leq n$, we build a phase, i.e., a sequence of moves, of S^m according to s_i. That is, depending on the type of rule applied in s_i, we add a sequence of moves $m_{j_{i-1}+1}, m_{j_{i-1}+1}, \ldots, m_{j_i}$ in S^m such that $P_i \subseteq M_{j_i}$. As $P_n = V$, at the last step all arcs are processed. To do this, assume that $m_1, \ldots, m_{j_{i-1}}$ are already defined based on $(s_1, \ldots, s_{i-1})$ and that $P_{i-1} \subseteq M_{j_{i-1}}$. Moreover, assume that $O_{i-1}^p = O_{j_{i-1}}^m$. We define $m_{j_{i-1}+1}, m_{j_{i-1}+1}, \ldots, m_{j_i}$ depending on which rule is applied in s_i:

- If s_i is a place operation at vertex v (move $Place(v)$), then let us define step $m_{j_{i-1}+1}$ to be $Place(v)$, i.e., a searcher is placed at v. Then, let $\{e_1, \ldots, e_r\}$ be the set of arcs with head v. For any $\ell \in [2, r+1]$, let us define step $m_{j_{i-1}+\ell}$ to be $Head(e_\ell)$. That is, all arcs with head v are sequentially processed.

  Hence, $j_i = j_{i-1} + r + 1$. The claim holds since $P_i = P_{i-1} \subseteq M_{j_{i-1}} \subseteq M_{j_i}$, and moreover, for any $j_{i-1} < \ell \leq j_i$, $O_\ell^m = O_i^p = O_{i-1}^p \cup \{v\}$.

- If s_i consists in processing an unoccupied vertex v (move $Process(v)$), then after step s_{i-1} in S^p, all vertices that are in the out-neighborhood of v are already processed. Hence, by the construction of S^m, after step $s_{j_{i-1}}$ in S^m, all arcs with tail v are already processed. Moreover, because $v \notin O_{i-1}^p = O_{j_{i-1}}^m$ then v is also unoccupied at step j_{i-1} of S^m.

  Hence, let $\{e_1, \ldots, e_r\}$ be the set of arcs with head v. For any $1 \leq \ell \leq r$, let us define $m_{j_{i-1}+\ell}$ as $Extend(e_i)$. That is, all arcs with head v are sequentially processed.

  In that case, $j_i = j_{i-1} + r$. The claim holds, since, in particular, $v \in M_{j_{i-1}}$.

- Now consider the case when s_i consists in processing an occupied vertex v and removing the searcher at v (move M_3'). Let us define step $m_{j_{i-1}+1} = m_{j_i}$ to be $Remove(v)$, i.e., the searcher at v is removed. In the case of recontamination in S^m, all vertices, v, in $v \in M_{j_{i-1}} \setminus M_{j_i}$ are tail of some arc e such that there is a path from v avoiding searchers and passing through e that reaches an unprocessed arc in $\ddot{D}$. Therefore, v also becomes unprocessed in S^p, i.e. $v \in P_{i-1} \setminus P_i$. Hence, the claim holds.

Therefore, S^m is a mixed process strategy for D using at most k searchers. □

We now prove a technical proposition that will be used in the proof of Lemma 7.

Proposition 6. *For any digraph D, there is a monotone mixed process strategy $S = (m_1, \ldots, m_n)$ using $\mathrm{mpr}(\ddot{D})$ searchers in $\ddot{D}$ and such that*

1. *S never applies moves of type Slide, and*
2. *if step m_i consists in processing an arc (u, v) such that u is occupied and all arcs with u as tail are already processed, then step m_{i+1} applies the rule $Remove(u)$.*

Proof. By Lemmas 2, 3 and 4, there exists a monotone mixed process strategy using $\mathrm{mpr}(\ddot{D})$ searchers in $\ddot{D}$. Let $S = (m_1, \ldots, m_n)$ be such a strategy.

1. If there is a step m_i $(1 \leq i \leq n)$ that applies the rule $Slide(e_1)$ for $e_1 = (u, v)$, then the second arc $e_2 = (u, v)$ must be processed and there must be no searcher at v. Then, step m_i is replaced by the following: first remove the searcher from u without recontaminating any arc, place the searcher at v and apply $Head(e_1)$. This must be possible for otherwise e_2 would have been recontaminated before. Therefore, we may assume that S^m never applies moves of type *Slide*.
2. If step m_i consists in processing an arc (u, v) such that u is occupied and all arcs with u as tail are already processed, then we may assume that step m_{i+1} applies the rule $Remove(u)$, i.e., the searcher at u is removed (and no recontamination occurs). Indeed, after step m_i, the searcher at u is not used to preserve from recontamination because all its outgoing arcs are processed and because the strategy is monotone. Moreover, if this searcher was used to process one in-coming arc a of u at a later step, we can instead use the rule $Extend(a)$. Finally, by 1., this searcher is never used to apply rule *Slide*.

□

Lemma 7. *For any digraph D,* $\mathrm{monpr}(D) \leq \mathrm{mpr}(\ddot{D})$.

Proof. Let $S^m = (m_1, \ldots, m_n)$ be a monotone mixed process strategy using $\mathrm{mpr}(\ddot{D})$ searchers in $\ddot{D}$ and such that

1. S never applies moves of type *Slide*, and
2. if step m_i consists in processing an arc (u, v) such that u is occupied and all arcs with u as tail are already processed, then step m_{i+1} applies the rule $Remove(u)$.

Such a strategy exists by Proposition 6.

Let M_i be the set of *unoccupied* vertices u such that all arcs with tail u are already processed after step m_i.

We now define a monotone process strategy $S^p = (s_1, \ldots, s_n)$ for D that uses at most $\mathrm{mpr}(\ddot{D})$ searchers. Let P_i be the set of processed vertices at step

$i \leq n$ in S^p and let M_i be the set of *unoccupied* vertices u such that all arcs with tail u are already processed after step m_i in S^m. Also, let O_i^p (resp., O_i^m) be the set of vertices occupied by a searcher at step s_i in S^p (resp., at step m_i in S^m). Assume that $(s_1, \ldots, s_{j_{i-1}})$ is already defined such that $O_{i-1}^m = O_{i-1}^p$, and $M_{i-1} \subseteq P_{i-1}$ or ($M_{i-1} \subseteq P_{i-1} \cup \{v\}$ and m_i consists in removing a searcher from some vertex v). We define s_i depending on m_i:

- Assume first that m_i consists in placing a searcher at vertex v ($Place(v)$). Then, let s_i consist in placing a searcher at v ($Place(v)$). The claim holds, since $M_i \subseteq M_i$ and $O_i^p = O_{i-1}^p \cup \{v\} = O_{i-1}^m \cup \{v\} = O_i^m$.

- If m_i consists in removing a searcher from a vertex v ($Remove(v)$) then, since S^m is monotone, recontamination does not happen, that is, there are no unoccupied directed path from a processed arc to an unprocessed one. Note that v is occupied since $O_{i-1}^m = O_{i-1}^p$. In this case, let s_i consist in processing v and removing the searcher at v ($Remove_{monot}(v)$). This is possible since all out-neighbors of v are either occupied or processed in S^m.

  The claim holds since $P_{j_i} = P_{j_{i-1}} \cup \{v\}$ and $M_i = M_{i-1} \cup \{v\}$, and moreover, $O_i^p = O_{i-1}^p \setminus \{v\} = O_{i-1}^m \setminus \{v\} = O_i^m$.

- If m_i consists in processing an arc $e = (u, v) \in A(D)$ ($Head(u, v)$ or $Extend(u, v)$). Then, if e is the only unprocessed arc with tail u before m_i then

  - If u is occupied by Property (2) of S^m, the next step m_{i+1} consists in removing the searcher at u. In that case, s_i consists in doing nothing and we have $M_i \subseteq P_i \cup \{u\}$ and $O_i^m = O_i^p$.
  - Else, let s_i consists in processing u (applying $Process(u)$). Again, the properties hold.

  If m_i consists in processing an arc $(u, v) \in A(D)$ that is not the last unprocessed out-going arc of u (in particular, we may assume it is the case for all arcs in $A(\ddot{D}) \setminus A(D)$), then s_i consists in doing nothing and the properties hold.

Therefore, S^p is a monotone process strategy for D using at most $\mathrm{mpr}(\ddot{D})$ searchers. □

From Lemmas 5 and 7, we get the following theorem.

Theorem 8. *For any digraph $D = (V, A)$,* $\mathrm{monpr}(D) \leq \mathrm{mpr}(\ddot{D}) \leq \mathrm{pr}(D)$.

Since, for any digraph D, $\mathrm{pr}(D) \leq \mathrm{monpr}(D)$, we obtain the next corollary:

Corollary 9. *For any digraph D,* $\mathrm{pr}(D) = \mathrm{monpr}(D)$.

3. Process Decomposition

In this section we define a digraph decomposition that is equivalent to (monotone) process strategies. This allows us to prove that the process number is invariant when reversing all arcs of a digraph. Let $D = (V, A)$ be a digraph.

A *process decomposition* of D is a sequence of pairs $P = ((W_1, X_1), (W_2, X_2), (W_3, X_3), \ldots, (W_t, X_t))$ such that:

1. for any $1 \leq i \leq t$, $W_i \subseteq V$ and $X_i \subseteq V$;
2. $\{X_1, \ldots, X_t\}$ is a partition of $V \setminus \bigcup_{i=1}^{t} W_i$;
3. $\forall i \leq j \leq k$, $W_i \cap W_k \subseteq W_j$;
4. X_i induces a Directed Acyclic Graph (DAG), for any $1 \leq i \leq t$;
5. $\forall (u, v) \in A$, $\exists j \leq i$ such that $v \in W_j \cup X_j$ and $u \in W_i \cup X_i$.

The width of a process decomposition is given by $\max_{1 \leq i \leq n} |W_i|$, and the *process-width*, denoted by $\mathrm{prw}(D)$, of a digraph D is given by the minimum width over all process decompositions of D. A scheme of a process decomposition can be found in Figure 1.

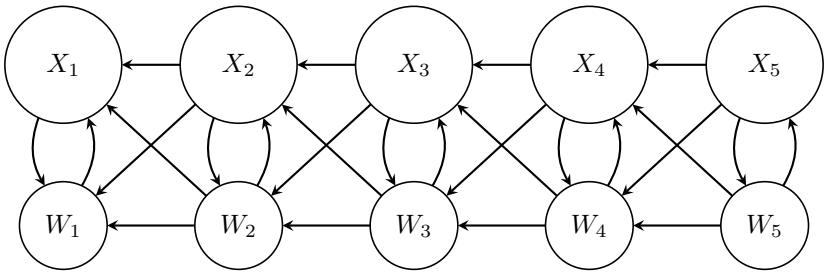

Figure 1: Scheme of a Process Decomposition. The sets X_i are disjoint from any other set in the decomposition and with each inducing a DAG in the original graph.

We show that a process decomposition can be used to design a monotone process strategy (Lemma 11). Intuitively, in order to design a monotone process strategy from a process decomposition we do the following. First we place searchers in all vertices of W_1. Then, we can proceed to process sequentially, in the oposite of a topological order, all vertices in X_1. Such a topological order exists since X_1 induces a DAG. Then, we can remove the searchers and process all vertices in $W_1 \setminus W_2$. We then repeat these same steps for each W_i.

Lemma 10 shows that reversing the arcs of a digraph does not change its process width. Then, we show the equivalence between process decompositions and monotone process strategies.

Let D^T be the digraph obtained by reversing the direction of the arcs of a digraph $D = (V, A)$.

Lemma 10. *For any digraph D,* $\mathrm{prw}(D) = \mathrm{prw}(D^T)$.

Proof. Let $P = ((W_1, X_1), \ldots, (W_t, X_t))$ be a process decomposition for D with width w. Let $P^T = ((W_t, X_t), \ldots, (W_1, X_1))$. Clearly, the first three properties

of process decomposition hold, and the width of P^T is w. It remains to show that $\forall (u,v) \in A(D^T)$, $\exists i \leq j$ such that $u \in W_i \cup X_i$ and $v \in W_j \cup X_j$. To do that, consider an edge $(u,v) \in A(D^T)$, since P is a process decomposition of D and $v\vec{u} \in A(D)$, we have that for some $j \leq i$, $v \in W_j \cup X_j$ and $u \in W_i \cup X_i$, therefore P^T is a process decomposition of D^T. □

Lemma 11. *For any digraph D,* $\mathrm{prw}(D) \geq \mathrm{monpr}(D)$.

Proof. Let $P = ((W_1, X_1), \ldots, (W_t, X_t))$ be a process decomposition of D of width w. We construct a monotone process strategy of D using at most w searchers. For any $1 \leq i \leq t$, we define the sequence of moves (Phase i) from (W_i, X_i), such that, after this sequence, the vertices of $W_i \cap W_{i+1}$ are occupied by searchers and the vertices in $\bigcup_{j=1}^{i}(W_j \cup X_j) \setminus W_{i+1}$ have been processed.

At phase $i+1$, we first place searchers at the vertices of $W_{i+1} \setminus W_i$. Then, in the inverse of a topological ordering of the DAG induced by X_{i+1}, vertices of X_{i+1} are processed. This is possible because, for any vertex v in X_{i+1}, any out-neighbor u of v is in $\bigcup_{j=1}^{i+1} X_j \cup W_j$ and so u is either already processed or occupied. Finally, searchers are removed from the vertices in $W_{i+1} \setminus W_{i+2}$ and these vertices are processed. Again, this is possible since every out-neighbor of a vertex in $W_{i+1} \setminus W_{i+2}$ belongs to $\bigcup_{j=1}^{i+1} X_j \cup W_j$ (by the last property of the decomposition).

Clearly, such a strategy is monotone and uses at most w searchers, hence $\mathrm{monpr}(D) \leq \mathrm{prw}(D)$. □

Lemma 12. *For any digraph D,* $\mathrm{monpr}(D) \geq \mathrm{prw}(D)$.

Proof. Let $S = (s_1, \ldots, s_t)$ be a monotone process strategy of D using k searchers. We remark that a searcher is removed from a vertex v, this vertex is also processed during the same step. We construct a process decomposition of D of width at most k. For any $1 \leq i \leq t$, let (W_i, X_i) be defined as follows. Let $(W_0, X_0) = (\emptyset, \emptyset)$:

- *Place*(v): if s_i consists in placing a searcher at vertex v, then $W_i = W_{i-1} \cup \{v\}$ and $X_i = \emptyset$;
- *Process*(v): if s_i consists in processing an unoccupied vertex v, then $W_i = W_{i-1}$ and $X_i = \{v\}$;
- *Remove*$_{monot}(v)$: if s_i consists in processing an occupied vertex v removing the searcher at v, then $W_i = W_{i-1} \setminus \{v\}$ and $X_i = \emptyset$.

It is easy to see that $\{X_1, \ldots, X_t\}$ is a partition of $V \setminus \bigcup_{i=1}^{t} W_i$ since all vertices are either occupied or processed (only once) without being occupied. Moreover, any X_i being reduced to at most a singleton induces a DAG. By the rules of the monotone process strategy, any vertex is occupied at most once (i.e., there are no two steps of S that consist in placing a searcher at the same vertex), and so $\forall i \leq j \leq k$, $W_i \cap W_k \subseteq W_j$.

Finally, let $(u, v) \in A$ and let i be the greatest integer such that $u \in W_i \cup X_i$ and let j be the smallest integer such that $v \in W_j \cup X_j$. By contradiction, assume that $j > i$. Then, u is processed at step s_i while its out-neighbor v is neither processed nor occupied at step i, since $j > i$, a contradiction.

Clearly, $\max_{i \leq t} |W_i| \leq k$. □

By Theorem 9, $\mathrm{pr}(D) = \mathrm{monpr}(D)$. Moreover, Lemmas 11 and 12 show that $\mathrm{monpr}(D) = \mathrm{prw}(D)$. Hence,

Theorem 13. *For any digraph D, $\mathrm{pr}(D) = \mathrm{prw}(D)$.*

Remark 1. *Note that, by the proof of Theorem 13, for any digraph D, there is an optimal process decomposition $((W_1, X_1), \ldots, (W_t, X_t))$ of D in which X_i has size at most one for any $1 \leq i \leq t$.*

Corollary 14. *Given a digraph $D = (V, A)$ and D^T, the graph obtained from D by reversing all the arcs, then $\mathrm{monpr}(D) = \mathrm{monpr}(D^T) = \mathrm{pr}(D) = \mathrm{pr}(D^T)$.*

4. Conclusion

Both tree-decompositions and path-decompositions have the notion of a dual structure, brambles and blockages respectively. For instance, the tree-width of an undirected graph G equals $k - 1$ if and only if G has no bramble greater[2] than k [6]. The monotonicity of a game plays an important role in the relationship between the width of a decomposition and its dual. Hence, it would be interesting to use our monotonicity result to define a dual for the process number.

On the other hand, the visible variant of the processing game appears to be an interesting candidate for providing a tree-decomposition for digraphs since the ability of a visible fugitive in the processing game is "between" the ones in the games in [13] and [4] (both having distinct advantages).

[2]Measured by the size of its hitting set.

[1] R. Ganian, P. Hlinený, J. Kneis, D. Meister, J. Obdrzálek, P. Rossmanith, S. Sikdar, Are there any good digraph width measures?, in: 5th Int. Symp. on Parameterized and Exact Computation, Vol. 6478 of LNCS, Springer, 2010, pp. 135–146.

[2] J. Barát, Directed path-width and monotonicity in digraph searching, Graphs and Combinatorics 22 (2006) 161–172.

[3] P. Hunter, S. Kreutzer, Digraph measures: Kelly decompositions, games, and orderings, Theor. Comput. Sci. 399 (3) (2008) 206–219.

[4] D. Berwanger, A. Dawar, P. Hunter, S. Kreutzer, J. Obdrzálek, The dag-width of directed graphs, JCTB 102 (4) (2012) 900–923.

[5] F. Fomin, D. Thilikos, An annotated bibliography on guaranteed graph searching, Theo. Comp. Sci. 399 (3) (2008) 236–245.

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

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

[8] A. S. LaPaugh, Recontamination does not help to search a graph, J. ACM 40 (2) (1993) 224–245.

[9] D. Bienstock, P. Seymour, Monotonicity in graph searching, J. Algorithms 12 (2) (1991) 239–245.

[10] F. Mazoit, N. Nisse, Monotonicity of non-deterministic graph searching, Theor. Comput. Sci. 399 (3) (2008) 169–178.

[11] B. Yang, D. Dyer, B. Alspach, Sweeping graphs with large clique number, Discrete Mathematics 309 (18) (2009) 5770 – 5780.

[12] P. Fraigniaud, N. Nisse, Monotony properties of connected visible graph searching, Information and Computation 206 (12) (2008) 1383 – 1393.

[13] T. Johnson, N. Robertson, P. D. Seymour, R. Thomas, Directed tree-width, J. Comb. Theory, Ser. B 82 (1) (2001) 138–154.

[14] I. Adler, Directed tree-width examples, JCTB 97 (2007) 718–725.

[15] S. Kreutzer, S. Ordyniak, Digraph decompositions and monotonicity in digraph searching, CoRR abs/0802.2228.

[16] P. Hunter, Losing the +1: Directed path-width games are monotone (2006).

[17] B. Yang, Y. Cao, Digraph strong searching: Monotonicity and complexity, in: AAIM, 2007, pp. 37–46.

[18] B. Yang, Y. Cao, On the monotonicity of weak searching, in: COCOON, 2008, pp. 52–61.

[19] D. Coudert, S. Perennes, Q.-C. Pham, J.-S. Sereni, Rerouting requests in wdm networks, in: AlgoTel'05, 2005, pp. 17–20.

[20] B. Mukherjee, WDM-based local lightwave networks-P. II: Multi-hop systems, IEEE Network 6 (4) (1992) 20–32.

[21] N. Jose, A. Somani, Connection rerouting/network reconfiguration, in: Design of Reliable Communication Networks, IEEE, 2003.

[22] X. Chu, T. Bu, X.-Y. Li, A study of lightpath rerouting schemes in wavelength-routed WDM networks, in: IEEE Intl. Conference on Communications (ICC), IEEE, Hong-Kong, 2007, pp. 2400–2405.

[23] F. Solano, M. Pióro, Lightpath reconfiguration in WDM networks, IEEE/OSA J. Opt. Commun. Netw. 2 (12) (2010) 1010–1021.

[24] N. Cohen, D. Coudert, D. Mazauric, N. Nepomuceno, N. Nisse, Tradeoffs in process strategy games with application in the wdm reconfiguration problem, Theor. Comput. Sci. 412 (35) (2011) 4675–4687.

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

[26] D. Coudert, J.-S. Sereni, Characterization of graphs and digraphs with small process number, Discrete Applied Mathematics 159 (11) (2011) 1094–1109.

[27] D. Coudert, F. Huc, D. Mazauric, A distributed algorithm for computing the node search number in trees, Algorithmica 63 (1-2) (2012) 158–190.

[28] D. Coudert, F. Huc, D. Mazauric, N. Nisse, J.-S. Sereni, Routing reconfiguration/process number: Coping wih two classes of services, in: 13th Conference on Optical Network Design and Modeling (ONDM), Lecture Notes in Computer Science, Braunschweig, Germany, 2009.

[29] F. Solano, Analyzing two conflicting objectives of the wdm lightpath reconfiguration problem, in: Proceedings of the Global Communications Conference (GLOBECOM), IEEE, 2009, pp. 1–7.

[30] F. Solano, M. Pióro, A mixed-integer programing formulation for the lightpath reconfiguration problem, in: VIII Workshop on G/MPLS Networks (WGN8), 2009.