On The Monotonicity of Process Number

Nicolas Nisse, Ronan Soares

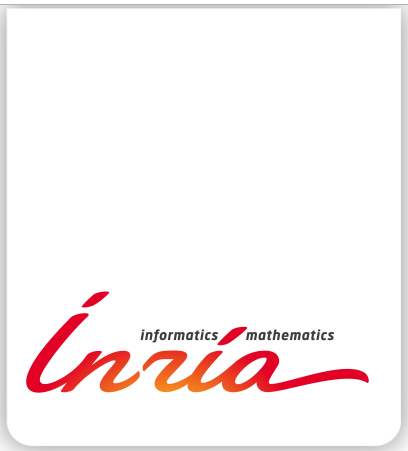
RESEARCH REPORT

N° 8132

October 2012

Project-Teams MASCOTTE

ISSN 0249-6399 ISRN INRIA/RR--8132--FR+ENG



On The Monotonicity of Process Number

Nicolas Nisse, Ronan Soares

Project-Teams MASCOTTE

Research Report n° 8132 — October 2012 — 14 pages

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 tree- and 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 such 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, arbitrary fast, it moves in the opposite direction of the arcs of a digraph, but only as long as it has 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.

Key-words: Graph Searching, Process Number, Monotonicity

RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

La Monotonie du Process Number

Résumé : Les jeux de capture impliquent une équipe d'agents qui doivent capturer un fugitif se déplaçant dans un graphe. Ces jeux ont été très étudiés pour leur interprétation en termes de décompositions de graphes (*tree-decomposition, path-decomposition*). Dans le but de définir de "bonnes" décompositions pour les graphes orientés, des jeux similaires ont été définis et étudiés dans les graphes orientés. Dans ce travail, nous considérons un jeu qui a été défini dans le contexte du routage dans les réseaux WDM. Dans ce jeu, le *processing game*, le fugitif est invisible, arbitrairement rapide et se déplace dans le sens opposé des arcs. De plus, le fugitif est capturé dès qu'il lui est impossible d'accéder à une composante fortement connexe sans agents. Nous prouvons que ce jeu est monotone. Cela permet de montrer l'équivalence du processing game et d'une nouvelle décomposition de graphes orientés.

Mots-clés : Recherche dans les Graphes, Monotonie, Process Number

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 path-decomposition or tree-decomposition in the case of undirected graphs (e.g., see [12]). Because graph searching games are equivalent to path- and tree-decompositions in undirected graphs, several attempts have been done to define such games in directed graphs [2, 3, 14].

1.1 Graph Searching and monotonicity

In graph searching games, a team of *searchers* aims to capture a fugitive that stands at the vertices of a graph G (see [10] for a survey). The fugitive can move arbitrary fast along the paths of G as long as it does not meet any searcher. A node $v \in V(G)$ is *clear* if all paths from v to the node occupied by the fugitive contain a node occupied by a searcher. In particular, a node occupied by a searcher is clear. A vertex that is not clear is said *contaminated*. Given a graph G with all nodes initially contaminated, a *strategy* for the searchers is a sequence of the following two possible actions: (R_1) place a searcher at a node of G, or (R_2) remove a searcher from a node. A strategy is *winning* if it allows to capture the fugitive whatever it does or, equivalently, if all nodes 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 and 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.

There are many variants of this problem arising due to different properties, or behaviors, given to the fugitive or to the searchers. For instance, if the fugitive is visible, the corresponding search number of a graph equals its *tree-width* plus one [20]. On the other hand, if the fugitive is invisible, the search number is equal to the *path-width* plus one [16]. 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 node is clear it never becomes contaminated anymore. Equivalently, in the case of an invisible fugitive, a searcher cannot be removed from a node if it has a neighbor that is neither occupied nor has been occupied before, i.e., once a node 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 [18] (invisible) and [20] (visible). A more simpler proof in the invisible case has been proposed by Bienstock and Seymour [4], and a unified proof for both visible and the invisible case can be found in [19]. 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 [25] neither when the fugitive is visible [11].

1.2 Graph searching in directed graphs

In [15], 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 [1]. [3] proposed a variant where the visible fugitive can move along directed paths without searchers and [17] defined a variant where the invisible fugitive can move along directed paths without searchers only when a searcher is about to land at the node that the fugitive is currently occupying. Both these variants, respectively related to DAG-width and Kelly-width, are not monotone [17].

In some other cases, considering an invisible fugitive, more optimistic 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 [4] to show that, in this variant, the monotonicity cannot increases the number of searchers by more than one [2]. Hunter then completed this proof to show the monotonicity of this variant [13]. Other variants that generalize the *edge-graph searching* (e.g., see [10]) 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) [23, 24].

1.3 Process Number

Surprisingly, a variant of graph searching in directed graphs has been defined in the context of routing reconfiguration in WDM networks [7]. 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 an directed graph N. The set of connections is given by $C \subseteq V(N) \times V(N)$. An initial routing of these connections, I, is given by a set 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 disjoint arc-wise. The final routing, F, is represented in the same manner as the initial routing, i.e. a set of arc-wise disjoint paths joining each pair $(a, b) \in C$.

Let P_a^i be the path joining two vertices of a connection a in its initial routing and P_a^f be the path joining two vertices of a connection a in its final routing. The objective of the routing reconfiguration problem is change the routing of the connections from the initial routing, I, to the final routing, F, while minimizing some criteria. In order to do this the following operations are allowed on the current routing of the connections on the network, at the beginning, this is the initial routing. One first operation is interrupting a connection, a, that is we remove the path $P_a^i \in I$ joining the two nodes of a from the current routing on the network. The second operation is to re-establish an interrupted connection a in its final routing, that is we add the path $P_a^f \in F$ to the current routing on the network. The last operation allowed is to switch the routing of a connection, a, from its initial path $P_a^i \in I$ to its final path $P_a^f \in F$ on the current routing of the network. These operations are allowed as long as there is no conflict in the routing, that is all paths in the current routing are disjoint arc-wise. We remark that it is always possible to change from the initial routing to the final routing by interrupting every connection then re-establishing these connections in their final routing.

There are several criteria that can be used to measure how "good" are different the sequences of operations used to change the initial routing of the network to the final routing. For example, the total number of interruptions is studied in [5] and the maximum number of simultaneous interruptions during the re-routing is studied in [7].

In the *processing game*, a team of searchers aims at *processing* all nodes of a digraph. A node is said *safe* if all its out-neighbors are either occupied or already processed. Given a digraph

$D = (V, A)$ where initially all nodes are unoccupied and not processed, a *monotone process strategy* is a sequence $(s_1, \ldots, s_n)$ of steps that results in processing all nodes of D, where each step s_i is one of the following three moves.

M_1: place a searcher at node $v \in V$;

M_2: process a safe *unoccupied* node $v \in V$;

M_3: 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 $\text{monpr}(D)$.

The relationship between the processing game and the routing reconfiguration problem is given by the dependency digraph. The dependency digraph $D = (V, A)$ of an instance of the routing reconfiguration problem has one vertex for each connection in the routing reconfiguration instance and there is an arc $e = (u, v) \in A$ if the connection given by u in its final routing shares an arc in the network with the connection given by v in its initial routing. Hence, a strategy for the process game gives a strategy for the routing reconfiguration problem. An important result is that for any directed graph D there is an instance of the reconfiguration problem such that D is its dependency digraph [5].

In [7], it is shown that, for any digraph D, $vs(D) \leq \text{monpr}(D) \leq vs(D) + 1$ where $vs(D)$ is the *vertex separation* of D. Moreover, to compute the monotone process number is NP-complete in general but it is polynomial in the class of graphs D with $\text{monpr}(D) \leq 2$ [8] and in the class of trees [9]. The relationship between the monotone process number of a digraph and its minimum feedback vertex set has been studied in [5]. The process number of digraphs has been mainly studied for its applications in the rerouting problem in WDM networks [6,21,22]. 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 node as the fugitive.

It is important to notice that the processing game when played on a symmetric digraph is not equivalent to the invisible graph searching games. Given a symmetric directed graph G the following inequation is true: $\text{s}(\bar{G}) - 1 \leq \text{monpr}(G) \leq s(\bar{G})$, where $\bar{G}$ denotes the underlying graph of G and $\text{s}(\bar{G})$ denotes the minimum number of searchers necessary to capture an invisible fugitive in $\bar{G}$. Moreover, this inequality is tight. Let K_n be the complete directed graph with n vertices, then $\text{s}(\bar{K_n}) = \text{monpr}(K_n) + 1$ and, for every directed graph G let G' be the directed graph obtained from G by adding a loop to every vertex of G, then $\text{s}(\bar{G'}) = \text{monpr}(G')$.

Let G be any directed graph, $\text{cn}(G)$ 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, for any directed graph G, $\text{cn}(G) - 1 \leq \text{monpr}(G) \leq \text{cn}(G)$, again the same examples above are sufficient to show that this inequality is tight in both sides.

In this work, we consider the more general variant of non necessarily monotone processing game. That is, we allow a processed node 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 nodes of D, where each step s_i consists of a move M_1 or M_2 or

M_3': 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.

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

The minimum number of searchers used by a process strategy of D is the *process number*, denoted by $\mathrm{pr}(D)$. In [9], 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 $\mathrm{pr}(D) = \mathrm{pr}(\overleftarrow{D})$ for any digraph $D = (V, A)$, where $\overleftarrow{D} = (V, \overleftarrow{A})$ and $\overleftarrow{A} = \{(a, b) : (b, a) \in A\}$.

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 [20] and adapted for directed graphs in [2, 23, 24]. More precisely, we first define the notion of mixed processing game and show its monotonicity thanks to an intermediate result dealing with crusades. Then, from any mixed process strategy we construct a process strategy with the same number of agents in a way that monotonicity is preserved.

2.1 Preliminaries

Throughout this section, we use the following notations. Let $D = (V, A)$ be a digraph. For any $v \in V$, let $N^-(v)$ denote the set of in-neighbors of v. The *border* of a set $X \subseteq A$, denoted by $\delta(X)$, is the set of vertices that are the head of an arc in X and the tail of an arc in $A \setminus X$. 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, $v \in \delta(X) \cap \delta(Y)$. □

Let $D = (V, A)$ be a digraph whose no arcs are initially processed. A *mixed process strategy* of D is a sequence $(s_1, \ldots, s_n)$ with the following actions that results in processing all arcs in A.

R_1 **(Place):** place a searcher at an unoccupied node $v \in V$;

R_2 **(Remove):** remove a searcher from node $v \in V$; 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.

R_3 **(Head):** process an arc $(u, v) \in A$ if $v \in V$ is *occupied*;

R_4 **(Slide):** slide the searcher at u along $(u, v) \in A$ if u is occupied, v is not occupied and all arcs $e \neq (u, v)$ with tail u are already processed, this process the arc (u, v);

R_5 **(Extend):** 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 fewest 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 edges 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 $X_0 = \emptyset$, $X_n = E$, 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 $i \leq n$.

Note that the notion of mixed strategy and crusade are different from the ones defined in [2].

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 the 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 is processed if s_i corresponds to R_3, R_4 or R_5), 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 $1 \leq i \leq n$. It 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 R_1 (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 R_2 (Remove) at a vertex v, 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 R_3 (Head), 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 R_4 (Slide) at an edge $e = (u, v)$, 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 R_5 (Extend), 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 $X_{i-1} \subset X_i$:

- 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, $|X_i \setminus X_{i-1}| = 1$.
- 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 $|\delta(X_{i-1} \cup X_i)| \geq |\delta(X_i)|$.
- 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 previous item, $|\delta(X_{i-1} \cap X_i)| \leq |\delta(X_{i-1})|$. 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 using at most k searchers. 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 $S = \emptyset$, the two above properties 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 actions that satisfies the two above properties 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 the step $s_{j_{i-1}+1}$ to be R_3 (Head) at e_i, i.e., the arc e_i is processed.
  - If moreover $v \notin \delta(X_i)$ then we define the step $s_{j_{i-1}+2}$ to be R_2 at 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 the step s_k to be R_2 at 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 the step $s_{j_{i-1}+1}$ to be R_1 at v, i.e., a searcher is placed at v, and the step $s_{j_i} = s_{j_{i-1}+2}$ is defined as R_3 (Head) at 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 the step $s_{j_{i-1}+1} = s_{j_i}$ to be R_4 through 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 the step $s_{j_{i-1}+1}$ to be R_5 (extend) at 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 R_2 at 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 that 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 the step $s_{j_{i-1}+1}$ to be R_1 at v, i.e., a searcher is placed at v, and $s_{j_i} = s_{j_{i-1}+2}$ to be R_3 (Head) at 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 defined the step $s_{j_i} = s_{j_{i-1}+1}$ as R_5 (Extend) at e_i, i.e., e_i is processed. The induction hypothesis holds.

Therefore, $S = (s_1, \ldots, s_{j_n})$ satisfies the two properties, 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 $\mathrm{mpr}(\vec{D})$ be the digraph obtained from any digraph $D = (V, A)$ by adding a copy of every arc of D.

Theorem 5. *For any digraph $D = (V, A)$, $\mathrm{monpr}(D) \leq \mathrm{mpr}(\vec{D}) \leq \mathrm{pr}(D)$.*

Proof. We first show that $\mathrm{mpr}(\vec{D}) \leq \mathrm{pr}(D)$.

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 $\vec{D}$. 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 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 $P_i \subseteq M_{j_i}$. Hence, since $P_n = V$, 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 M_1), then let us define the step $m_{j_{i-1}+1}$ to be R_1 at vertex v, i.e., a searcher is placed at v. Then, let $\{e_1, \cdots, e_r\}$ be the set of arcs with

head v. For any $\ell \in [2, r+1]$, let us define the step $m_{j_{i-1}+\ell}$ to be R_3 (Head) at e_ℓ. 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 $j_{i-1} < \ell \leq j_i$, $O^m_\ell = O^p_i = O^p_{i-1} \cup \{v\}$.

- If s_i consists in processing an unoccupied vertex v (move M_2), 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 $v \notin O^p_{i-1} = O^m_{j_{i-1}}$ then v is also unoccupied at step j_{i-1} of S^m.

  Hence, let $\{e_1, \cdots, e_r\}$ be the set of arcs with head v. For any $1 \leq \ell \leq r$, let us define $m_{j_{i-1}+\ell}$ as R_5 (Extend) at 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 the step $m_{j_{i-1}+1} = m_{j_i}$ to be R_2 at 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 agents and passing through e that reaches an unprocessed arc in $\vec{\vec{D}}$. 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.

Now, let us show that $\text{monpr}(D) \leq \text{mpr}(\vec{\vec{D}})$. By Lemmas 2, 3 and 4, there exists a monotone mixed process strategy using $\text{mpr}(\vec{\vec{D}})$ searchers in $\vec{\vec{D}}$. Let $S^m = (m_1, \ldots, m_n)$ be such a strategy.

We first notice that if there is a step m_i ($1 \leq i \leq n$) that applies a rule of type R_4 (Slide) through an arc $e_1 = (u, v)$, then the second arc $e_2 = (u, v)$ must be processed and there must be no searcher at v. Hence it is possible to replace the step m_i by the following: first remove the agent from u, without recontaminating any arc (since otherwise e_2 would have been recontaminated before), place the agent at v and apply R_3 (Head) operation at e_1. Therefore, we may assume that S^m never applies moves of type R_4.

Another remark is that, if the 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 the step m_{i+1} applies the rule R_2 to 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 of u at a step further, we can instead use the extend rule R_5. Finally, by previous remark, this searcher is never used to apply rule R_4.

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 $\text{mpr}(\vec{\vec{D}})$ 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^p_i, resp., O^m_i, 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 $O^m_{i-1} = O^p_{i-1}$, 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 node v). We define s_i depending on m_i:

- Assume first that m_i consists in placing a searcher at vertex v (R_1). Then, let s_i consist in placing a searcher at v (M_1). The claim holds, since $M_i \subseteq M_i$ and $O^p_i = O^p_{i-1} \cup \{v\} = O^m_{i-1} \cup \{v\} = O^m_i$.

- If m_i consists in removing a searcher from a vertex v (R_2) then, since S^m is monotone, recontamination does not happen. That is, there are no unoccupied directed path from a process arc to an unprocessed one. Note that v is occupied since $O^m_{i-1} = O^p_{i-1}$. In that case, let s_i consists in processing v and removing the searcher at v (M_3), 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, $O^p_i = O^p_{i-1} \setminus \{v\} = O^m_{i-1} \setminus \{v\} = O^m_i$.

- If m_i consists in processing an arc $e = (u, v) \in A(D)$ (R_3 or R_5). Then, if e is the only unprocessed arc with tail u before m_i then

  - If u is occupied by the remark above, 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 $O^m_i = O^p_i$.
  - Else, let s_i consists in processing u (applying M_2). 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 $A(\vec{D}) \setminus A(D)$), then s_i consists in doing nothing and the properties hold.

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

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

Corollary 1. *recontamination does not help to process a digraph, i.e., 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 $P = ((W_1, X_1), \cdots, (W_t, X_t))$ such that:

- for any $1 \leq i \leq t$, $W_i \subseteq V$ and $X_i \subseteq V$;
- $(X_1, \cdots, X_t)$ is a partition of $V \setminus \bigcup_{i=1}^{t} W_i$;
- $\forall i \leq j \leq k$, $W_i \cap W_k \subseteq W_j$;
- X_i induces a Directed Acyclic Graph (DAG), for any $1 \leq i \leq t$;
- $\forall (u, v) \in A$, $\exists j \leq i$ 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 first result, shows that reversing the arcs of a digraph does not change its process width. Let $\overleftarrow{D}$ be the digraph obtained by reversing the sense of the arcs of a digraph $D = (V, A)$.

Lemma 6. *For any digraph D,* $\mathrm{prw}(D) = \mathrm{prw}(\overleftarrow{D})$.

Proof. Let $P = ((W_1, X_1), \ldots, (W_t, X_t))$ be a process decomposition for D with width w. Let $\overleftarrow{P} = ((W_t, X_t), \ldots, (W_1, X_1))$. Clearly, the first three properties of process decomposition hold, and the width of $\overleftarrow{P}$ is w. It remains to show that $\forall (u,v) \in A(\overleftarrow{D})$, $\exists i \leq j$ such that $u \in W_i \cup X_i$ and $v \in W_j \cup X_j$. To do that, consider an edge $(u,v) \in A(\overleftarrow{D})$, 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 $\overleftarrow{P}$ is a process decomposition of $\overleftarrow{D}$. □

Theorem 7. *For any digraph D,* $\mathrm{pr}(D) = \mathrm{prw}(D)$.

Proof. By Theorem 1, $\mathrm{pr}(D) = \mathrm{monpr}(D)$. Hence, we show that $\mathrm{monpr}(D) = \mathrm{prw}(D)$.

To show that $\mathrm{prw}(D) \geq \mathrm{monpr}(D)$ let $P = ((W_1, X_1), \ldots, (W_t, X_t))$ be a process decomposition of D with 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 valid since all 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 $\mathrm{monpr}(D) \leq \mathrm{prw}(D)$.

Now, let us show that $\mathrm{monpr}(D) \geq \mathrm{prw}(D)$. Let $S = (s_1, \cdots, s_t)$ 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 with width at most k. For any $1 \leq i \leq t$, let (W_i, X_i) be defined as follows. Let $(W_0, X_0) = (\emptyset, \emptyset)$:

- M_1: if s_i consists in placing a searcher at vertex v, then $W_i = W_{i-1} \cup \{v\}$ and $X_i = \emptyset$;
- M_2: if s_i consists in processing an unoccupied vertex v, then $W_i = W_{i-1}$ and $X_i = \{v\}$;
- M_3: 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 $(X_1, \cdots, X_t)$ 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 $v \in W_j \cup X_j$. For purpose of 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, $\max_{i \leq t} |W_i| \leq k$. □

Remark 1. Note that, by the proof of Theorem 7, for any digraph D, there is an optimal process decomposition $((W_1, X_1), \ldots, (W_t, X_t))$ of D such that X_i has size at most one for any $i \leq t$.

Corollary 2. *Given a digraph $D = (V, A)$ and $\overleftarrow{D}$, the graph obtained from D by reversing all the arcs, then* $\mathrm{monpr}(D) = \mathrm{monpr}(\overleftarrow{D}) = \mathrm{pr}(D) = \mathrm{pr}(\overleftarrow{D})$.

4 Conclusion

We have shown that the same number of agents are needed even if we allow recontamination in the process number game. We defined a decomposition that is equivalent to the process number game. It is still not known whether or not there is a dual structure for this decomposition, in the same manner as a bramble is a dual structure for the tree-decomposition.

References

[1] Isolde Adler. Directed tree-width examples. *J. Comb. Theory Ser. B*, 97:718–725, September 2007.

[2] János Barát. Directed path-width and monotonicity in digraph searching. *Graphs and Combinatorics*, 22:161–172, 2006.

[3] Dietmar Berwanger, Anuj Dawar, Paul Hunter, Stephan Kreutzer, and Jan Obdrzálek. The dag-width of directed graphs. *J. Comb. Theory, Ser. B*, 102(4):900–923, 2012.

[4] D. Bienstock and Paul Seymour. Monotonicity in graph searching. *J. Algorithms*, 12(2):239–245, April 1991.

[5] Nathann Cohen, David Coudert, Dorian Mazauric, Napoleão Nepomuceno, and Nicolas Nisse. Tradeoffs in process strategy games with application in the wdm reconfiguration problem. *Theor. Comput. Sci.*, 412(35):4675–4687, 2011.

[6] D. Coudert, F. Huc, D. Mazauric, N. Nisse, and 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, February 2009.

[7] D. Coudert, S. Perennes, Q.-C. Pham, and J.-S. Sereni. Rerouting requests in wdm networks. In *AlgoTel'05*, pages 17–20, mai 2005.

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

[9] David Coudert, Florian Huc, and Dorian Mazauric. A distributed algorithm for computing the node search number in trees. *Algorithmica*, 63(1-2):158–190, 2012.

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

[11] Pierre Fraigniaud and Nicolas Nisse. Monotony properties of connected visible graph searching. *Information and Computation*, 206(12):1383 – 1393, 2008.

[12] Robert Ganian, Petr Hlinený, Joachim Kneis, Daniel Meister, Jan Obdrzálek, Peter Rossmanith, and Somnath Sikdar. Are there any good digraph width measures? In *5th International Symposium on Parameterized and Exact Computation (IPEC)*, volume 6478 of *Lecture Notes in Computer Science*, pages 135–146. Springer, 2010.

[13] Paul Hunter. Losing the +1: Directed path-width games are monotone, 2006. unpublished result, `http://www.cs.ox.ac.uk/people/paul.hunter/papers/losing.pdf`.

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

[15] Thor Johnson, Neil Robertson, Paul D. Seymour, and Robin Thomas. Directed tree-width. *J. Comb. Theory, Ser. B*, 82(1):138–154, 2001.

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

[17] Stephan Kreutzer and Sebastian Ordyniak. Digraph decompositions and monotonicity in digraph searching. *CoRR*, abs/0802.2228, 2008.

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

[19] Frédéric Mazoit and Nicolas Nisse. Monotonicity of non-deterministic graph searching. *Theor. Comput. Sci.*, 399(3):169–178, 2008.

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

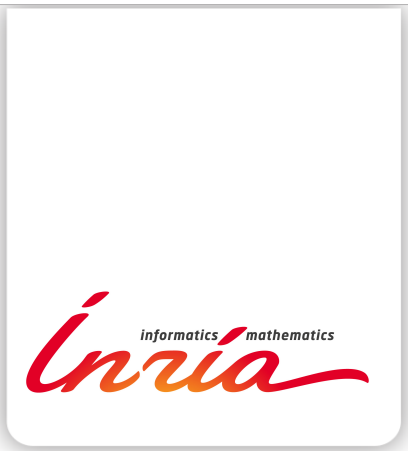
[21] F. Solano and M. Pióro. A mixed-integer programing formulation for the lightpath reconfiguration problem. In *VIII Workshop on G/MPLS Networks (WGN8)*, 2009.

[22] Fernando Solano. Analyzing two conflicting objectives of the wdm lightpath reconfiguration problem. In *Proceedings of the Global Communications Conference (GLOBECOM)*, pages 1–7. IEEE, 2009.

[23] Boting Yang and Yi Cao. Digraph strong searching: Monotonicity and complexity. In *AAIM*, pages 37–46, 2007.

[24] Boting Yang and Yi Cao. On the monotonicity of weak searching. In *COCOON*, pages 52–61, 2008.

[25] Boting Yang, Danny Dyer, and Brian Alspach. Sweeping graphs with large clique number. *Discrete Mathematics*, 309(18):5770 – 5780, 2009. <ce:title>Combinatorics 2006, A Meeting in Celebration of Pavol Hells 60th Birthday (May 1 to5, 2006)</ce:title>.



RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399