REPORTS IN INFORMATICS

ISSN 0333-3590

Nondeterministic Graph Searching: From Pathwidth to Treewidth

Fedor V. Fomin, Pierre Fraigniaud and Nicolas Nisse

REPORT NO 300 May 2005

Department of Informatics
UNIVERSITY OF BERGEN
Bergen, Norway

This report has URL `http://www.ii.uib.no/publikasjoner/texrap/ps/2005-300.ps`

Reports in Informatics from Department of Informatics, University of Bergen, Norway, is available at `http://www.ii.uib.no/publikasjoner/texrap/`.

Requests for paper copies of this report can be sent to:

Department of Informatics, University of Bergen, Høyteknologisenteret,
P.O. Box 7800, N-5020 Bergen, Norway

Nondeterministic Graph Searching: From Pathwidth to Treewidth

Fedor V. Fomin*
Department of Informatics, University of Bergen,
PO Box 7800, 5020 Bergen, Norway. *fomin@ii.uib.no*

Pierre Fraigniaud†
CNRS, Lab. de Recherche en Informatique, Université Paris-Sud,
91405 Orsay, France. *pierre@lri.fr*

Nicolas Nisse‡
Lab. de Recherche en Informatique, Université Paris-Sud,
91405 Orsay, France. *nisse @lri.fr*

25th May 2005

Abstract

We introduce nondeterministic graph searching with a controlled amount of nondeterminism and show how this new tool can be used in algorithm design and combinatorial analysis applying to both pathwidth and treewidth. We prove equivalence between this game-theoretic approach and graph decompositions called q-*branched* tree decompositions, which can be interpreted as a parameterized version of tree decompositions. Path decomposition and (standard) tree decomposition are two extreme cases of q-branched tree decompositions. The equivalence between nondeterministic graph searching and q-branched tree decomposition enables us to design an exact (exponential time) algorithm computing q-branched treewidth for all $q \geq 0$, which is thus valid for both treewidth and pathwidth. This algorithm performs as fast as the best known exact algorithm for pathwidth. Conversely, this equivalence also enables us to design a lower bound on the amount of nondeterminism required to search a graph with the minimum number of searchers.

Keywords: treewidth, pathwidth, graph searching.

1 Introduction

Treewidth and pathwidth are among the most key parameters in graph algorithms, also playing important roles in structural graph theory. Both parameter serve as the important tools in Robertson and Seymour's Graph Minors project [17]. Many intractable problems can be solved in polynomial time when the input is restricted to graphs of bounded treewidth. (See Bodlaender's survey [4] for a comprehensive overview.) Treewidth also plays a crucial role in Downey & Fellows parameterized complexity theory (Chapter 6 in [9]). Moreover, treewidth is the basic ingredient for

*Additional support by Norges forskningsråd projects 162731/V00 and 160778/V30.
†Additional supports from the INRIA Project "Grand Large", and from the Project PairAPair of the ACI "Masse de Données".
‡Additional supports from the Project Fragile of the ACI "Sécurité & Informatique".

many applications in artificial intelligence, databases and logical-circuit design. To mention just a few of these applications: Exact inference in Bayesian networks, reasoning with structured constraint-satisfaction problems, propositional satisfiability and first-order logic. See [1] for further references.

In this paper we introduce the new notion of q-branched treewidth which can be interpreted as a parameterized version of treewidth. Loosely speaking, a rooted tree decomposition is q-branched if every path from the root of the tree to a leaf contains at most q branching nodes (nodes with at least two children). This notion is a natural generalization of path and tree decompositions: For $q = \infty$, q-branched treewidth is equivalent to the treewidth, and, for $q = 0$, q-branched treewidth is equivalent to the pathwidth of a graph.

Both parameters, pathwidth and treewidth, have nice game-theoretic interpretations. (See a survey of Bienstock [3].) Pathwidth can be described as a search game where searchers, looking for a fugitive, are successively placed to and removed from vertices of the graph. (Kirousis and Papadimitriou [13] called this version of searching by node searching.) The purpose of searching is to capture the fugitive that is invisible and moves arbitrarily fast along paths in the graph. The fugitive is not allowed to run through the vertices currently occupied by searchers. So the fugitive is caught when a searcher is placed on the vertex it occupies, and it has no possibility to leave the vertex because all the neighbors are occupied (guarded) by searchers. The goal of search games is to find a search strategy that guarantees the fugitive's capture. The pathwidth of a graph G is equal to the minimum number of searchers needed for a successful search strategy on G, minus one.

Treewidth also can be described as a search game, where a team of searchers are trying to catch a *visible* fugitive. It was shown by Seymour and Thomas [18] that the minimum number of searchers required to catch the fugitive on a graph G in this game is equal to the treewidth of G plus one. (An alternative game-theoretic interpretation of treewidth in terms of searching was given by Dendris et al. [6] who restrict the ability of the (called *inert*) fugitive to move only when a searcher is placed at the vertex where the fugitive currently stands.)

Game theoretic interpretation of width parameters is interesting not only in its own. Very often it provides a deeper insight to the problem yielding new structural and algorithmic results. Good examples are proofs of min-max theorems on treewidth by Seymour and Thomas [18], the polynomial time algorithm computing branch-width of a planar graph in [19], the linear time algorithm on trees for computing cutwidth in [15], as well as the computation of the topological bandwidth in [14], and the vertex separation number in [7]. It is therefore natural to ask if there is a game theoretic interpretation of the q-branched treewidth.

Our results. To answer the question above, we introduce a new game model providing a unique approach to both search models of Kirousis-Papadimitriou, and Seymour-Thomas. In our search game the searchers can query an oracle which possesses information about the position of the fugitive. However the number of times the searchers can query the oracle is limited. This situation can be interpreted as using powerful but expensive intelligence service with limited resources. More formally, q-limited graph searching is a graph searching game in which the search program is allowed to perform nondeterministic search steps. In the same spirit as in the field of complexity theory addressing limited nondeterminism (cf., e.g., [11] for a survey), the number of nondeterministic steps of the search program is however limited. The parameter q limits the program to at most q nondeterministic steps.

We first show a formal equivalence between q-limited graph searching and q-branched treewidth. Precisely, we prove that a graph G has a q-branched treewidth $\leq k$ if and only if it can be searched with at most $k + 1$ searchers by a search strategy using at most q nondeterministic steps. Moreover, we establish a one-to-one correspondence between the q-branched tree decompositions of G of width $\leq k$ and the q-limited search strategies using $\leq k+1$ searchers.

Then we use q-limited graph searching to design an exact (exponential-time) algorithm computing the q-branched treewidth of a graph. The interest in exact and fast exponential-time algorithms solving hard problems dates back to the sixties and seventies [12, 20]. The last decade has led to much research in fast exponential-time algorithms. We refer to Woeginger's survey [21] for an overview. However despite of the importance of treewidth and pathwidth, and despite the fact that much progress on exponential-time solutions to other graph problems have been made, the only worst-case bound known so far for finding pathwidth is $2^n \cdot n^{O(1)}$. This can be obtained by adopting classical TSP dynamic programming approach [12]. For treewidth, the fastest known exponential algorithm is an $O(1.96^n)$ algorithm due to Fomin et al. [10]. In this paper we design an algorithm computing q-branched treewidth of a graph on n vertices in time $2^n \cdot n^{O(1)}$ for any $q \geq 0$.

Finally, the equivalence between q-limited graph searching and q-branched tree decomposition enables us to design a lower bound on the amount of nondeterminism required to search a graph with the minimum number of searchers. Precisely, we prove that, for any graph G of treewidth $\mathbf{tw}(G) = k$, the smallest $q \geq 0$ such that G can be searched by $k+1$ searchers using a q-limited search program is at least $\log_2(\mathbf{pw}(G)/\mathbf{tw}(G))$ where $\mathbf{pw}(G)$ is the pathwidth of G.

2 Formal Definitions

In this section, we formally define the two notions of q-branched treewidth and q-limited graph searching. Later on, these two notions will be shown to be equivalent.

Branched Treewidth. A *tree decomposition* of graph G is a pair $(T, \mathcal{X})$ where T is a tree of node set I, and $\mathcal{X} = \{X_i, i \in I\}$ is a collection of subsets of $V(G)$ satisfying the following three conditions:

1. $V(G) = \cup_{i \in I} X_i$;
2. For any edge e of G, there is a set $X_i \in \mathcal{X}$ containing both end-points of e;
3. For any triple i_1, i_2, i_3 of nodes of T, if i_2 is on the path from i_1 to i_3 in T, then $X_{i_1} \cap X_{i_3} \subseteq X_{i_2}$.

The *width* of a tree decomposition is defined as $\mathrm{width}(T, \mathcal{X}) = \max_{i \in I} |X_i| - 1$. A *rooted* tree decomposition of a graph G is a tree decomposition $(T, \mathcal{X})$ of G where T is rooted at some node $r \in I$. It is denoted by $(T, \mathcal{X}, r)$. A *branching node* of a rooted tree decomposition is a node with at least 2 children.

For any $q \geq 0$, a *q-branched tree decomposition* of a graph G is a rooted tree decomposition $(T, \mathcal{X}, r)$ of G such that every path in T from the root r to a leaf contains at most q branching nodes.

Thus a path decomposition rooted at one of its extremities is a 0-branched tree decomposition, and a (standard) tree decomposition is a ∞-branched tree decomposition.

For any graph G, the *q-branched treewidth* $\mathbf{tw}_q(G)$ of G is the minimum width of over all q-branched tree-decomposition of G.

Therefore, $\mathbf{pw}(G) = \mathbf{tw}_0(G)$ and $\mathbf{tw}(G) = \mathbf{tw}_\infty(G)$. Figure 1 displays the "spectral width" of a graph G, i.e., the graph of the function $f_G : \mathbf{N} \to \mathbf{N}$ such that $f_G(q) = \mathbf{tw}_q(G)$. In this figure, $\tau(G) = \min\{q \geq 0 \mid \mathbf{tw}_q(G) = \mathbf{tw}(G)\}$ and $\pi(G) = \max\{q \leq \tau(G) \mid \mathbf{tw}_q(G) = \mathbf{pw}(G)\}$.

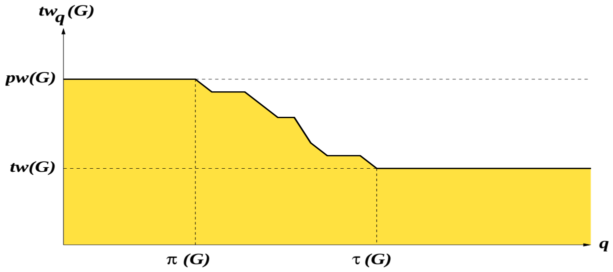

Figure 1: Spectral width of graph G

2.1 Graph Searching

Search games are games between a *fugitive* and *searchers* in a graph. The fugitive and the searchers occupy vertices of the graphs. The goal of the fugitive is to escape from the searchers. It is caught when a searcher is placed on the vertex currently occupied by the fugitive. The fugitive permanently knows where the searchers are, and moves arbitrarily fast, but it cannot meet a searcher without being caught. The searchers do not know the position of the fugitive.

More formally, a *search program* is a (deterministic) program that takes as input a graph G and an integer $k \geq 1$, and returns an ordered sequence of *search steps*. This sequence is called the *search strategy* for G. Each step is an operation that consists in either "placing a searcher at $v \in V(G)$" or "removing a searcher from $v \in V(G)$". After a searcher s has been placed at v, and before s is removed from v, vertex v is said to be *occupied* by searcher s. When a vertex has been occupied by a searcher, it becomes *clear*. Vertices that have not been cleared yet are called *contaminated*. The search program is correct if it satisfies the following constraints:

1. no more than k searcher are simultaneously occupying vertices of G;
2. a step "place a searcher at v" occurs at most once, for every vertex v;
3. when a searcher is removed from a vertex v, for any path P between v and contaminated vertices, there is a searcher occupying a vertex of P.

A *fugitive program* in a graph G is a deterministic automaton F whose states are all possible triples (S, X, v) where $S \subset V(G)$, $X \subseteq S$, and $v \in V(G) \setminus S$. If the automaton is in state (S, X, v), then the fugitive is currently occupying vertex v, the searchers are occupying vertices in X, and S is the set of clear vertices. Given a state (S, X, v) of the automaton, the transition function of the fugitive program returns a new state (S, X, v') where v and v' are constrained to be in the same connected component of $G \setminus S$. Then the fugitive moves in G from vertex v to vertex v'. The initial state of the fugitive program is the state $(\emptyset, \emptyset, v_0)$ for some $v_0 \in V(G)$.

A *search game* is then a game between the fugitive program and the search program. A *configuration* of the game is a tripple (S, X, v) where S is the set of clear vertices, X is the set of vertices occupied by searchers, and v is the position of the fugitive. From constraint 3 of the search program, we always have $\delta(S) \subseteq X$, where $\delta(S)$ denotes the set of vertices in S that have a neighbor in $G \setminus S$. Initially, the fugitive is placed in v_0, where $(\emptyset, \emptyset, v_0)$ is the initial state of the fugitive program. I.e., the initial configuration of the game is $(\emptyset, \emptyset, v_0)$. Then the search program and the fugitive program play alternatively. Each step of the search program transforms the current configuration (S, X, v) of the game into a configuration $(S \cup \{u\}, X \cup \{u\}, v)$ (in case of a step "place a searcher at u"), or into a configuration $(S, X \setminus \{u\}, v)$ (in case of a step "remove a searcher from u").

The search program *wins* the game if the game reaches a configuration in which $v \in X$. Otherwise the fugitive wins. If the search program wins, then the fugitive is said to be caught. Note that the fugitive wins if the search program cannot carry on without violating one of its three constraints.

The search program that places a different searcher on every vertex of the graph wins against any fugitive. It however requires n searchers in n-node graphs.

Definition. *The* search number *of G, denoted by $\mathbf{s}(G)$, is the minimum number of searchers required by a search program to win against any fugitive in G.*

2.2 Nondeterministic Graph Searching

A *nondeterministic* search program is a search program that can do nondeterministic steps. Each nondeterministic step consists in a *query operation*. Given the set $S \subset V(G)$ of clear vertices, a query returns a connected component C of $G \setminus S$, and all vertices in $G \setminus C$ are cleared. The choice of C is nondeterministic. Alternatively, it can be viewed as given by an oracle answering on a query by letting the searchers know in which component is the fugitive. A nondeterministic search program is thus a nondeterministic program that takes as input a graph G and an integer $k \geq 1$, and returns an ordered sequence of search steps, each of them being one of the following three operations:

- place a searcher at $v \in V(G)$;
- remove a searcher from $v \in V(G)$;
- query the oracle.

Of course, the program must satisfy the same three constraints as any (deterministic) search program.

A nondeterministic search program *wins* the game against a fugitive F if there exists an execution of the program which results in clearing the node currently occupied by the fugitive.

We are interested in the tradeoff between the number of searchers used by a search program and the number of query steps performed by the program. For any $q \geq 0$, a *q-limited* nondeterministic search program is a nondeterministic search program that performs at most q query steps. Therefore, a q-limited nondeterministic search program wins the game against a fugitive F if it can catch the fugitive by querying at most q times an oracle that returns the connected component where is currently hidden the fugitive.

Definition. *The* q-limited nondeterministic search number $\mathbf{s}_q(G)$ *of a graph G, or simply the* q-limited search number *of G, is the minimum number of searchers required by a q-limited nondeterministic search program to win against any fugitive in G.*

Therefore, the 0-limited nondeterministic search number of a graph is its search number, i.e., $\mathbf{s}_0(G) = \mathbf{s}(G)$. We will prove in the next section that the ∞-limited nondeterministic search number $\mathbf{s}_\infty(G)$ of a graph G is equal to its visible-search number.

3 Branched Treewidth vs. Limited Graph Searching

In this section, we show that the q-branched treewidth and the q-limited search number are actually the same, up to 1. This equality will be later shown to be useful for the design of algorithms and for the computation of combinatorial bounds.

Theorem 1. *For any $q \geq 0$, for any graph G, $\mathbf{tw}_q(G) = \mathbf{s}_q(G) - 1$.*

Proof. Let $(T, \mathcal{X}, r)$ be a q-branched decomposition of width k. For a node i of T let $d(i)$ be the set of descendants of i in T. We define the search program of $k+1$ searchers querying the oracle at most q times as follows. Initially the searchers are placed on the vertices of X_r. Suppose that, at some step of searching, for some node i of T the searchers are on vertices X_i and the set of contaminated vertices is $\cup_{j \in d(i)} X_j \setminus X_i$. Note that if i is a leaf, G is cleared. Let i be a non-leaf node of T. Depending on the number of children of i we choose different strategy for the searchers.

Case A. i has only one child l. We remove first the searchers from $X_i \setminus X_l$ and then place searchers to X_l. Since the cardinality of X_i and X_l is at most $k+1$ we use at most $k+1$ searchers. By properties of tree decompositions, for every contaminated vertex $v \in \cup_{j \in d(i)} X_j \setminus X_i$ and every cleared vertex u, every (u, v)-path contains a vertex from $X_i \cap X_l$. Thus after removing the searchers from $X_i \setminus X_l$ no recontamination occurs and we arrive at the situation when the searchers are at X_l and the set of contaminated vertices is $\cup_{j \in d(l)} X_j \setminus X_l$.

Case B. i has more than one child. In this case the searchers query the oracle. Let C be the connected component of $G[\cup_{j \in d(i)} X_j \setminus X_i]$ returned by the oracle. Then there is a unique child l of i such that $C \cap X_l \neq \emptyset$. We remove the searchers from $X_i \setminus X_l$ and then place searchers to X_l. Again, in this case we arrive at the situation when the searchers are at X_l and the set of contaminated vertices is $\cup_{j \in d(l)} X_j \setminus X_l$.

Eventually, the searchers reach the situation when they are placed on the vertices X_i where i is leaf of T and thus the whole graph is cleared. The number of searchers used is at most $\max_{j \in V(T)} |X_j| \leq k+1$. Since for every leaf i, the path from r to i contains at most q branches, the case B occurs at most q times, thus the searchers query the oracle at most q times. Hence $\mathbf{s}_q(G) \leq \mathbf{tw}_q(G) + 1$.

We prove $\mathbf{tw}_q(G) \leq \mathbf{s}_q(G) - 1$ by proving a slightly stronger claim.

Claim 2. Suppose that there is a search program of $k+1$ searchers on G with at most q queries and such that, initially, searchers are placed on vertices $X \subseteq V(G)$. Then there is a q-branched tree decomposition $(T, \mathcal{X}, r)$ with $X_r = X$ and of width $\leq k$.

To prove the claim we proceed by induction on q. For $q = 0$, the required path decomposition $P = (X_0, X_1, \ldots, X_m)$ is constructed by taking $X_0 = X$, and, for $i \geq 1$, X_i to be the vertex set occupied by searchers after the ith step. To check that P is the path decomposition we observe that every vertex should be at some step occupied by a searcher and thus is contained in some node of P. Every pair of adjacent vertices $\{u, v\}$ is contained in some node of P because otherwise fugitive can avoid capture by choosing u or v at every step of searching. The third property of tree decompositions follows from the constraints 2 and 3 of graph searching.

Let $q \geq 1$ and suppose that for all $q' < q$, the claim is correct. Consider a winning search program of $k+1$ searchers with at most q queries to the oracle. Suppose that the first time the searchers query the oracle occurs at step $t \geq 0$. Let X be the set of vertices occupied by the searchers and S be the cleared vertices at this step. Let $G_1, G_2, \ldots, G_p$ be the subgraphs of G obtained from the connected components of $G \setminus S$ by adding X. Each of these subgraphs is searchable by $k+1$ searchers with at most $q-1$ queries with the search starting from X. By induction assumption, for each $1 \leq i \leq p$, there is a rooted tree decomposition $(T^{(i)}, \mathcal{Y}^{(i)}, r_i)$ of G_i with at most $q-1$ branches and with the root r_i of $T^{(i)}$ satisfies $Y_{r_i} = X$.

We construct a tree decomposition $(T, \mathcal{Y}, r)$ of G as follows. Let $X_1, \ldots, X_t$ be the vertices occupied by the searchers at the first t steps of searching. In particular

$X_t = X$. We construct the path decomposition $(X_1, \ldots, X_t)$ rooted at X_1. Then we add the tree decompositions $(T^{(i)}, \mathcal{Y}^{(i)}, r_i)$, $1 \leq i \leq p$, and identify every r_i to the node t of the path decomposition. The resulting tree decomposition is a q-branched tree decomposition of width $\leq k$. □

4 Exact Exponential Algorithm

For any $q \geq 0$, the decision problem that takes as input a graph G and an integer $k \geq 1$, and returns whether or not $\mathbf{tw}_q(G) \leq k$, is NP-complete. Indeed, it is known [2] that the problem of deciding whether $\mathbf{tw}(G) \leq k$ is NP-complete, even when restricted to the co-bipartite graphs, i.e., the complements of bipartite graphs. Since, for any co-bipartite graph G, $\mathbf{tw}(G) = \mathbf{pw}(G)$, the NP-completeness of deciding $\mathbf{tw}_q(G) \leq k$ follows from the fact that $\mathbf{tw}(G) \leq \mathbf{tw}_q(G) \leq \mathbf{pw}(G)$ for any $q \geq 0$. It is known [5] that $\mathbf{tw}$ can be approximated up to multiplicative factor $O(\log \mathbf{tw})$, in polynomial time. (This bound has been recently improved to $O(\sqrt{\log \mathbf{tw}})$, cf. [8].) However, for pathwidth, no approximation algorithm is known (except by combining the ones for treewidth with the fact that $\mathbf{pw}(G) \leq O(\log n) \cdot \mathbf{tw}(G)$ for any n-node graph G). On the other hand, as mentioned in the introduction, several exact (exponential) algorithms have been designed for treewidth, and for pathwidth as well. In this section, we show that one can design an exact algorithm that applies to q-branched treewidth, for all $q \geq 0$. This algorithm uses the correspondence between q-branched treewidth and q-limited search number.

Theorem 3. *There exists an algorithm that, for any n-node graph G, computes $\mathbf{tw}_q(G)$ and an optimal q-branched tree decomposition of G, in time $O(2^n n \log n)$.*

Proof. Based on Theorem 1, we design an algorithm that computes $\mathbf{s}_q(G)$, and an optimal q-limited search strategy for G. This startegy can be then transformed into a q-branched tree decomposition using the arguments in the proof of Theorem 1. Let G be a graph, and fix $k \geq 1$. We define the configuration digraph H as follows.

$$V(H) = \{S \subseteq V(G) \text{ s.t. } |\delta(S)| \leq k\}.$$

A set S of clear vertices for which $|\delta(S)| > k$ are unreachable by a search program using k searchers, and thus it is not included in $V(H)$. The nodes in H are called H-configurations, to avoid confusion with the configurations of the search game. The edge-set of H has two types of directed edges: *place* edges, and *query* edges. A place edge, or simply p-edge, is an edge (S, S') where $|\delta(S)| < k$ and $S' = S \cup \{v\}$, $v \notin S$. Clearly, a p-edge corresponds to the placement of a searcher at node v. A query edge, or simply q-edge, is an edge (S, S') where $S' = G \setminus C$ and C is a connected component of $G \setminus S$. It is assumed that there is a q-edge $(S, G \setminus C)$ only if $G \setminus S$ has at least two connected components (i.e., there is no self-loop in H). Thus, a q-edge $(S, G \setminus C)$ corresponds to a query to the oracle that returns C. The objective of our algorithm is to find a path in the configuration digraph H from $S = \emptyset$ to $S = V(G)$ that can be put in correspondence with a search strategy.

For the purpose of finding such a path, we label every node of $V(H)$ by a non-negative integer. The labeling starts from the H-configuration $V(G)$ and goes backwards. The H-configuration $V(G)$ is labeled 0. All the other H-configurations are labeled ∞. All the H-configurations without any outgoing edge are declared *finished*. (In particular $V(G)$ is finished.) All the other H-configurations are declared *pending*. We proceed as long as there is at least one pending H-configuration S satisfying one of the two following conditions:

Case 1. S has an outgoing p-edge e connecting to an H-configuration S' that is finished. (Informally, this case occurs if the labeling has not yet considered the game configuration $(S, \delta(S), v)$, $v \notin S$, from which the next search step is: place a searcher at $S' \setminus S$.)

Case 2. S has all its outgoing q-edges $e_1, \dots, e_d$ connecting to H-configurations $S'_1, \dots, S'_d$ that are finished. (Informally, this case occurs if the labeling has not yet considered the game configuration $(S, \delta(S), v)$, $v \notin S$, from which the next search step is: query the oracle.)

In case 1, we update the label of S by:

$$\text{label}(S) = \min\{\text{label}(S),\ \text{label}(S')\}$$

and the edge e is removed from H. In case 2, we update the label of S by:

$$\text{label}(S) = \min\{\text{label}(S),\ 1 + \max_{1 \le i \le d} \text{label}(S'_i)\}$$

and all the edges $e_1, \dots, e_d$ are removed from H. In both cases, if the pending H-configuration S has no more outgoing edges because of the edge(s) removal, then S is declared finished.

Claim 4. The labeling process terminates.

To prove the claim, notice that H is a directed acyclic graph because every edge goes from an H-configuration S to another H-configuration S' with $|S'| > |S|$ (recall that we did not allowed self-loops in H). Removing edges from H preserves this property. Therefore every node becomes eventually finished and thus the labeling process terminates.

Claim 5. The H-configuration $\emptyset$ is labeled $q < \infty$ if and only if q is the smallest number of queries required to clear G using $\le k$ searchers. The H-configuration $\emptyset$ is labeled ∞ if one cannot clear G using $\le k$ searchers, independently of the number of queries to the oracle.

We prove that claim by proving a slighly more general result: for any H-configuration $S \neq V(G)$, S is finished and labeled $\text{label}(S) \le q \neq \infty$ if and only if one can clear G starting from S with $\le k$ searchers and performing $\le q$ queries. By starting from S, it is meant that we assume an initial configuration of the search game in which S is clear, $|\delta(S)|$ searchers are placed at the vertices of $\delta(S)$, and the fugitive is at some vertex of $G \setminus S$. We proceed by induction on $q = \text{label}(S)$.

If $q = 0$, then there is a path P in H from S to $V(G)$ using only p-edges. Let $(S', S'') \in P$, with $S'' = S' \cup \{v\}$. The portion of the search strategy corresponding to that edge consists in removing one by one all searchers occupying vertices $\notin \delta(S')$, and placing a searcher at v. Hence one can catch the fugitive without performing queries by starting from S and following the edges of P until one reaches the configuration $V(G)$. Conversely, if one can clear G starting from S with $\le k$ searchers and performing no queries, then there is a path in H from S to $V(G)$ composed on only p-edges. These edges are defined by placement steps in the search strategy.

Assume now that the result holds for q, and consider S such that $\text{label}(S) = q + 1$. We define a *good* edge as a p-edge (S', S'') such that $\text{label}(S) = \text{label}(S') = \text{label}(S'')$. From S, start traveling in H by using good edges only, until one reaches a configuration S^* with

$$\text{label}(S) = \text{label}(S^*) = 1 + \max_{i=1,\dots,d} \text{label}(S^*_i)$$

where the edges (S^*, S^*_i), $i = 1, \dots, d$, are all the q-edges out-going from S^*. This configuration S^* exists because (1) a good edge (S', S'') satisfies $|S''| > |S'|$, and (2) $\text{label}(S') = \text{label}(S'') = \text{label}(S) < \infty$. Therefore, if a configuration S^* as specified above would not be met, then, by (1) an H-configuration with out-degree 0 would eventually be reached, and by (2) this H-configuration could only be $V(G)$ since otherwise its label would be ∞. Since $\text{label}(S) = q+1 > 0$, by induction this would contradict the fact that there is no path from S to $V(G)$ composed of p-edges only. So S^* is well defined.

For all $i = 1, \dots, d$, $\text{label}(S_i^*) \leq q$. Therefore, by induction hypothesis, one can clear G starting from any S_i^* using $\leq k$ searchers, and quering $\leq q$ times the oracle. The search strategy from S starts by performing place and remove steps according to the path in H from S to S^*. The search then queries the oracle at S^*, and gets into one of the configurations S_i^*. The rest of the search follows from the induction hypothesis.

Conversely, assume that one can clear G starting from S with $\leq k$ searchers and performing $q+1$ queries. Consider a corresponding search strategy in G, and assume that the first query to the oracle occurs at step t. The $t-1$ first steps can be put in correspondence with a path P in H starting at S, and that contains good edges only. Let (S^*, X^*, v^*) be the configuration of the game after step $t-1$. P connects S with the H-configuration S^*. The query at step t corresponds to the outgoing q-edges (S^*, S_i^*), $i = 1, \dots, d$, of S^*. From each of the S_i^*s, the search proceeds with at most q queries. Hence, by induction, $\text{label}(S_i^*) \leq q$. Therefore, $\text{label}(S^*) \leq q+1$. Since P is a path of good edges, we get $\text{label}(S) = \text{label}(S^*) \leq q+1$.

For each k, the running time of the labeling procedure is linear in the number of edges of H, which is $O(2^n n)$. Thus by binary search we can find the search number and $\mathbf{tw}_q(G)$ in $O(2^n n \log n)$.

□

5 Bounding the Nondeterminism

In this section, we compute a lower bound on the number of nondeterministic steps a search program must perform in a graph G in order to clear the graph with the minimum possible number of searchers, i.e., $\mathbf{tw}(G) + 1$ searchers.

Theorem 6. *For any $q \geq 1$, for any graph G, $\mathbf{tw}_{q-1}(G) \leq 2\,\mathbf{tw}_q(G)$.*

Proof. We first need a technical result. Let G be a graph and let $(T, \mathcal{X}, r)$ be a tree decomposition of G rooted at r. For any $i \in V(T)$, the subtree of T rooted at i is denoted by T_i. The subgraph of G induced by the vertices in the nodes of T_i is denoted by $G[T_i]$. Let $(T', \mathcal{X}', i)$ be a tree decomposition of $G[T_i]$ such that $X_i = X'_i$. Let T_{new} be the tree obtained from T and T' by removing T_i from T, and replacing it by T' rooted at i. Let $\mathcal{X}_{new} = \{Y_j,\ j \in V(T_{new})\}$ be the familly of subsets of $V(G)$ such that $Y_j = X'_j$ if $j \in V(T')$ and $Y_j = X_j$ if $j \in V(T_{new}) \setminus V(T')$. Clearly, we have:

Claim 7. *$(T_{new}, \mathcal{X}_{new})$ is a tree decomposition of G. Moreover, if $\text{width}(T, \mathcal{X}) = k$ and $\text{width}(T', \mathcal{X}') = k'$ then $\text{width}(T_{new}, \mathcal{X}_{new}) \leq \max\{k, k'\}$.*

Let $(T, \mathcal{X}, r)$ be an optimal q-branched tree decomposition of G rooted at r, i.e., $(T, \mathcal{X})$ is of width $\mathbf{tw}_q(G)$. We define the *lowest-branching* nodes of T as those nodes $i \in V(T)$ such that (1) i has at least two children in T, and (2) for any $j \in V(T_i) \setminus \{i\}$, j has at most one child. Let Π be the set of paths from r to a leaf in T containing exactly q branching nodes. Let J be the set of the lowest-branching nodes that belong to some path in Π. Let $i \in J$. T_i consists in the node i and $d \geq 2$ paths $P_1, \dots, P_d$ pending at i. We define the path T' obtained by concatenating the P_is, i.e., $T' = (i, P_1, \dots, P_d)$. Any node j of T' is a node of T, hence we can define $X'_j = X_i \cup X_j$. Let $\mathcal{X}' = \{X'_j \mid j \in V(T')\}$. $(T', \mathcal{X}')$ is a path decomposition of $G[T_i]$ of width $\leq 2\,\mathbf{tw}_q(G)$. Let $T^{(i)}_{new}$ be the tree obtained from T and T' by removing

T_i from T, and replacing it by T' rooted at i. Let $\mathcal{X}_{new}^{(i)} = \{Y_j,\ j \in V(T_{new}^{(i)})\}$ be the family of subsets of $V(G)$ such that $Y_j = X'_j$ if $j \in V(T')$ and $Y_j = X_j$ if $j \in V(T_{new}^{(i)}) \setminus V(T')$. By the claim above, $(T_{new}^{(i)}, \mathcal{X}_{new}^{(i)})$ is a tree decomposition of G of width $\leq 2\,\mathbf{tw}_q(G)$.

We repeat this process for every node $i \in J$. Since for any two nodes $i \neq i'$ in J, $T_i \cap T_{i'} = \emptyset$, every application of the replacement process is independent from the previous ones. Therefore, we eventually obtain a rooted tree decomposition $(T^*, \mathcal{X}^*, r)$ of G of width $\leq 2\,\mathbf{tw}_q(G)$. By construction, any path from r to a leaf of T^* contains at most $q-1$ branching nodes. Therefore $\mathbf{tw}_{q-1}(G) \leq 2\,\mathbf{tw}_q(G)$. □

Note that the bound of Theorem 6 is tight, as witnessed by the graphs consisting in a complete binary tree (all non-leaf vertices, including the root, are of degree 3) of depth q, in which every vertex u is replaced by a complete graph K_u of k vertices, and every edge $\{u,v\}$ is replaced by a perfect matching between the two complete graphs K_u and K_v. Theorem 6 has important corollaries:

Corollary 8. *For any graph G, the smallest $q \geq 0$ such that $\mathbf{tw}_q(G) = \mathbf{tw}(G)$ satisfies $\tau(G) \geq \log_2(\mathbf{pw}(G)/\mathbf{tw}(G))$.*

Remark. There exist n-node graphs G such that $\mathbf{tw}(G) = \mathbf{pw}(G) + 1$ and the greatest $q \geq 0$ such that $\mathbf{tw}_q(G) = \mathbf{pw}(G)$ satisfies $\pi(G) \geq \Omega(n)$.

Rephrasing Corollary 8, we get:

Corollary 9. *For any graph G, the smallest number of nondeterministic steps of a nondeterministic search program that clears G with $\mathbf{tw}(G) + 1$ searchers is at least $\log_2(\mathbf{pw}(G)/\mathbf{tw}(G))$.*

6 Conclusion

In this paper, we introduced a nondeterministic graph searching game, with a controlled amount of nondeterminism. The objective of this concept was to unify pathwidth and treewidth, at least as far as the design of algorithms, and the computation of combinatorial bounds in concerned. We believe that this is a promising field of investigations, as illustrated by the design of an exact algorithm for q-branched treewidth, valid for any $q \geq 0$. Still, a lot of work has to be done before being able to design common tools for both pathwidth and treewidth.

In particular, it would be particularly interesting to design a polynomial-time algorithm computing the q-limited search number (or equivalently the q-branched treewidth) of trees. As far as algorithm design is concerned, it would also be quite interesting to design an $O(c^n)$-time exact algorithm for the q-branched treewidth of arbitrary graphs, with $c < 2$. Such an algorithm is known [10] in the case of treewidth (i.e., $q = \infty$), but not for pathwidth (i.e., $q = 0$). Last but not least, it is known that, for node-search (i.e., 0-limited graph searching) and visible-search (i.e., ∞-limited graph searching), removing constraint 3 of the search program does not enable to decrease the number of searchers. It would be important to know whether this is true for any $q \geq 0$, i.e., whether or not "recontamination helps" for q-limited graph searching, for any $q \geq 0$.

References

[1] E. Amir, *Efficient approximation for triangulation of minimum treewidth*, in Uncertainty in Artificial Intelligence: Proceedings of the Seventeenth Conference (UAI-2001), San Francisco, CA, 2001, Morgan Kaufmann Publishers, pp. 7–15.

[2] S. Arnborg, D. G. Corneil, and A. Proskurowski, *Complexity of finding embeddings in a k-tree*, SIAM J. Algebraic Discrete Methods, 8 (1987), pp. 277–284.

[3] D. Bienstock, *Graph searching, path-width, tree-width and related problems (a survey)*, DIMACS Ser. in Discrete Mathematics and Theoretical Computer Science, 5 (1991), pp. 33–49.

[4] H. L. Bodlaender, *A partial k-arboretum of graphs with bounded treewidth*, Theoret. Comput. Sci., 209 (1998) , pp. 1–45.

[5] V. Bouchitté, D. Kratsch, H. Müller, and I. Todinca. *On treewidth approximations.* Discrete Applied Mathematics 136(2-3), pages 183-196, 2004.

[6] N. D. Dendris, L. M. Kirousis, and D. M. Thilikos, *Fugitive-search games on graphs and related parameters*, Theor. Comp. Sc., 172 (1997), pp. 233–254.

[7] J. A. Ellis, I. H. Sudborough, and J. Turner, *The vertex separation and search number of a graph*, Information and Computation, 113 (1994), pp. 50–79.

[8] U. Feige, M. Hajiaghayi, J. Lee . *Improved approximation algorithms for minimum-weight vertex separators.* In 37th ACM Symposium on Theory of Computing (STOC 2005).

[9] R. G. Downey and M. R. Fellows, *Parameterized Complexity*, Springer-Verlag, New York, 1999.

[10] F. V. Fomin, D. Kratsch, and I. Todinca, *Exact algorithms for treewidth and minimum fill-in,* Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP 2004), LNCS vol. 3142, Springer, Berlin, 2004, pp. 568–580.

[11] J. Goldsmith, M. Levy, and M. Munhenk, *Limited Nondeterminism.* SIGACT News, Introduction to Complexity Theory Column 13, June 1996.

[12] M. Held and R. Karp, *A dynamic programming approach to sequencing problems*, J. Soc. Indust. Appl. Math., 10 (1962), pp. 196–210.

[13] L. M. Kirousis and C. H. Papadimitriou, *Searching and pebbling*, Theor. Comp. Sc., 47 (1986), pp. 205–218.

[14] F. S. Makedon, C. H. Papadimitriou, and I. H. Sudborough, *Topological bandwidth*, SIAM J. Alg. Disc. Meth., 6 (1985), pp. 418–444.

[15] F. S. Makedon and I. H. Sudborough, *On minimizing width in linear layouts*, Disc. Appl. Math., 23 (1989), pp. 243–265.

[16] N. Robertson and P. D. Seymour, *Graph minors. II. Algorithmic aspects of tree-width*, J. Algorithms, 7 (1986), pp. 309–322.

[17] N. Robertson and P. D. Seymour, *Graph minors. X. Obstructions to tree-decomposition*, J. Combin. Theory Ser. B, 52 (1991), pp. 153–190.

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

[19] P. Seymour and R. Thomas, *Call routing and the ratcatcher*, Combinatorica, 14 (1994), pp. 217–241.

[20] R.E. Tarjan and A.E. Trojanowski, *Finding a maximum independent set*, SIAM J. Computing, 6 (1977), pp. 537–546.

[21] G. J. Woeginger, *Exact algorithms for NP-hard problems: a survey*, in Combinatorial Optimization: "Eureka, you shrink", LNCS vol. 2570, Springer, Berlin, 2003, pp. 185–207.