Connected Treewidth and Connected Graph Searching

Pierre Fraigniaud[1*] and Nicolas Nisse[2**]

[1] CNRS, Lab. de Recherche en Informatique, Université Paris-Sud, 91405 Orsay, France. *pierre@lri.fr*
[2] Lab. de Recherche en Informatique, Université Paris-Sud, 91405 Orsay, France. *nisse@lri.fr*

Abstract. We give a constructive proof of the equality between *treewidth* and *connected treewidth*. More precisely, we describe an $O(nk^3)$-time algorithm that, given any n-node width-k tree-decomposition of a connected graph G, returns a connected tree-decomposition of G of width $\leq k$. The equality between treewidth and connected treewidth finds applications in *graph searching* problems. First, using equality between treewidth and connected treewidth, we prove that the *connected* search number $\mathbf{cs}(G)$ of a connected graph G is at most $\log n + 1$ times larger than its search number. Second, using our constructive proof of equality between treewidth and connected treewidth, we design an $O(\log n \sqrt{\log OPT})$-approximation algorithm for connected search, running in time $O(t(n) + nk^3 \log^{3/2} k + m \log n)$ for n-node m-edge connected graphs of treewidth at most k, where $t(n)$ is the time-complexity of the fastest algorithm for approximating the treewidth, up to a factor $O(\sqrt{\log OPT})$.

1 Introduction

The *treewidth* of a graph is a central concept in the theory of Graph Minors developped by Robertson and Seymour. Roughly speaking, the treewidth, $\mathbf{tw}(G)$, of a graph G measures "how far" the graph G is from a tree. More formally, a *tree-decomposition* of graph G is a pair (T, X) where T is a tree, and $X = \{X_v, v \in V(T)\}$ is a collection of subsets of $V(G)$ satisfying the following three conditions:

- **C1:** $V(G) = \cup_{v \in V(T)} X_v$;
- **C2:** For any edge e of G, there is a set X_v such that both end-points of e are in X_v;
- **C3:** For any triple u, v, w of nodes in $V(T)$, if v is on the path from u to w in T, then $X_u \cap X_w \subseteq X_v$.

* 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".

Condition **C3** can be rephrased as: for any node x of G, $\{v \in V(T) \mid x \in X_v\}$ is a subtree of T. The sets X_v, $v \in V(T)$, are often called *bags*. The *width*, $\omega(T, X)$, of a tree-decomposition (T, X) is defined as $\max_{v \in V(T)} |X_v| - 1$, i.e., the width of (T, X) is roughly the maximum size of its bags. The treewidth $\mathbf{tw}(G)$ is defined as $\min \omega(T, X)$ where the minimum is taken over all tree-decompositions (T, X) of G. Hence the treewidth of any tree is 1, the treewidth of outerplanar graphs is ≤ 2, and the treewidth of an n-node complete graph is $n - 1$.

Treewidth is related to other types of graph-decompositions. In particular, Seymour and Thomas [15] introduced the concept of *carving*. For the sake of simplicity, we restrict ourself to edge-carving, i.e., *branch-decomposition* [14]. A branch-decomposition of a graph G is a pair (T, f) where T is a tree with all its internal nodes of degree 3, and f is a one-to-one mapping between the leaves of T and the edges of G. Given an edge e of T, removing e from T results in two trees $T_1^{(e)}$ and $T_2^{(e)}$. An *e-cut* of a branch-decomposition is defined as the pair $\{E_1^{(e)}, E_2^{(e)}\}$, where $E_i^{(e)} \subset E(G)$ is the set of leaves of $T_i^{(e)}$ for $i = 1, 2$. For any edge-set $E \subseteq E(G)$, let $\delta(E)$ denote the set of nodes of G with one incident edge in E and another in $E(G) \setminus E$. The *width* of a branch decomposition (T, f) is defined as $\omega(T, f) = \max_e |\delta(E_1^{(e)})|$ where the maximum is taken over all e-cuts in T. The *branchwidth*, $\mathbf{bw}(G)$, of G is then defined as $\min \omega(T, f)$ where the minimum is taken over all branch-decompositions (T, f) of G. It was proved in [14] that: $\mathbf{bw}(G) - 1 \leq \mathbf{tw}(G) \leq 3\,\mathbf{bw}(G)/2$.

Both tree-decomposition and branch-decomposition can be requested to be *connected*. An e-cut of a tree-decomposition (T, X) of a graph G is defined as the pair $\{X_1^{(e)}, X_2^{(e)}\}$, where $X_i^{(e)} \subseteq V(G)$ is the set of nodes of G in $\bigcup_{v \in V(T_i^{(e)})} X_v$ for $i = 1, 2$.

- A tree-decomposition is *connected* if, for any of its e-cuts, the two subgraphs of G, induced by $X_1^{(e)}$ and $X_2^{(e)}$ are connected. The connected treewidth, $\mathbf{ctw}(G)$, of a connected graph G, is defined as the minimum width of any connected tree-decomposition of G.
- A branch-decomposition is *connected* if, for any of its e-cut, the two subgraphs of G induced by $E_1^{(e)}$ and $E_2^{(e)}$ are connected. The connected branchwidth, $\mathbf{cbw}(G)$, of a connected graph G, is defined as the minimum width of any connected branch-decomposition of G.

A major result about branchwidth is that if a 2-edge-connected graph G has a branch-decomposition of width k, then it has a connected branch-decomposition of width $\leq k$ (see [15]). Therefore:

$$\text{For any 2-edge-connected graph } G,\ \mathbf{cbw}(G) = \mathbf{bw}(G). \tag{1}$$

The proof of (1) in [15] is non constructive, but it can be transformed into a constructive one (cf. [8]). The same result as (1) was proved for treewidth, by combining results in [9] and [11]. Indeed, on one hand, it was shown in [9] that a "clique tree" of a minimal triangulation H of a connected graph G is an optimal tree-decomposition of G (i.e., of width $\mathbf{tw}(G)$). On the other hand, [11] proved that the set Δ_H of the minimal separators of H is exactly the set of pairwise "parallel" minimal separators in G, and for any $S \in \Delta_H$, S induces the same connected components in H and G, which implies that the clique tree is in fact a connected tree-decomposition. As a consequence:

$$\text{For any connected graph } G,\ \mathbf{tw}(G) = \mathbf{ctw}(G). \tag{2}$$

Note that the equality $\mathbf{ctw} = \mathbf{tw}$ holds for any connected graph, whereas the equality $\mathbf{cbw} = \mathbf{bw}$ holds for 2-edge-connected graphs. The proof of (2) by combination of [9] and [11] is non constructive.

One of the contributions of this paper is the description of a constructive proof of (2). This result has an impact on the design of connected search strategies. Indeed, treewidth is related to several variants of the *graph searching* problem (see, e.g., [3, 6, 10, 12]). In graph searching, a fugitive is hidden in a graph G. A team of *searchers* is aiming at capturing this fugitive. These searchers can be placed at nodes, removed from nodes, and moved along the edges. The fugitive is assumed to be arbitrary fast, and permanently aware of the positions of the searchers. The graph searching problem asks for the design of search strategies using a minimum number of searchers. The search number of a graph G varies depending on the relative power of the fugitive and the searchers. If the searchers are permanently aware of the position of the fugitive, then the optimal size of the team is essentially $\mathbf{tw}(G)$ [3]. On the other hand, if the searchers are unaware of the position of the fugitive, then the optimal size of the team, $\mathbf{s}(G)$, is essentially the *pathwidth* $\mathbf{pw}(G)$ of G [3]. (A path-decomposition of G is a tree-decomposition (T, X) of G, where T is a path. The pathwidth is defined as $\min \omega(T, X)$ where the minimum is taken over all path-decompositions (T, X) of G.) For any graph G, we have [3]: $\mathbf{pw}(G) \leq \mathbf{s}(G) \leq \mathbf{pw}(G) + 2$. For instance, $\mathbf{s}(P_n) = 1$ for the n-node path P_n, $\mathbf{s}(C_n) = 2$ for the n-node cycle C_n, $\mathbf{s}(K_n) = n$ for the n-node clique K_n, and $\mathbf{s}(T) \leq \log_3(n-1) + 1$ for any n-node tree T [10].

It has been argued (cf., e.g., [1] and the references therein) that several practical applications (e.g., network security, speleological rescue [5], etc.) require the search strategy be connected, i.e., at any time of the search strategy, the portion of the searched graph is a connected subgraph. All searchers are initially placed at the same node, and clear the graph by moving along the edges from that initial node. In [1] is described a polynomial-time algorithm for computing the *connected* search number, $\mathbf{cs}$, of trees. There are n-node graphs G for which $\mathbf{cs}(G) > \mathbf{s}(G)$. (For instance, there are trees with connected search number $\lfloor \log_2 n \rfloor$). A major challenge regarding the connected search number is actually to bound the "cost of connectedness", that is to bound the ratio connected search number over search number. In [2], it is proved that the connected search number of a tree is at most twice its search number, and this bound is tight. Deriving bounds for arbitrary graphs is more complex, for at least two reasons. First, the set of graphs with connected search number at most k is not minor-closed, as opposed to the non-connected setting. Second, there are graphs for which no optimal connected search strategy is monotone [16], as opposed to the non-connected setting [4]. (Roughly, a search strategy is monotone if the fugitive cannot "recontaminate" a part of the graph that has been cleared). Nevertheless, using the concept of branchwidth, [8] shows that, for arbitrary connected m-edge graph G, the connected search number, $\mathbf{cs}(G)$, satisfies $\mathbf{cs}(G)/\mathbf{s}(G) \leq \lceil \log m \rceil + 1$. In [8] is also described an $O(t(n) + m^3)$-time $O(\log n \log OPT)$-approximation algorithm for connected search in arbitrary graphs, where $t(n)$ is the time complexity of the fastest algorithm for approximating the treewidth of an n-node graph, up to a factor $O(\log OPT)$.

Our results

1. We give a constructive proof of the equality between the treewidth and the connected treewidth of connected graphs. This proof is obtained via the design of a polynomial-time algorithm transforming an n-node tree-decomposition of width k into a connected tree-decomposition of width $\leq k$, in time $O(nk^3)$.
2. We prove that $\mathbf{cs}(G)/\mathbf{s}(G) \leq \log n + 1$ via the design of a connected search strategy based on a connected tree-decomposition of the graph.
3. We combine this design with our algorithm for connected tree-decomposition, resulting in an $O(t(n) + nk^3 \log^{3/2} k + m \log n)$-time $O(\log n \sqrt{\log OPT})$-approximation algorithm for connected search, where $t(n)$ is the time complexity of the fastest algorithm for approximating the treewidth of an n-node graph, up to a factor $O(\sqrt{\log OPT})$, and k is the treewidth of the graph.

The two latter results improve [8].

2 Subconnected Tree-Decomposition

Given a tree-decomposition (T, X) of a graph G, and $u \in V(T)$, we denote by (T, X, u) the tree-decomposition (T, X) rooted at node u. For $v \in V(T)$, we denote by T_v the subtree of (T, X, u) rooted at v. The subgraph of G induced by the nodes in the bags of T_v is denoted by $G[T_v]$.

Definition 1. *A rooted tree-decomposition (T, X, u) is* subconnected *at $v \in V(T)$ if, for any $w \in V(T_v)$, $G[T_w]$ is a connected graph. (T, X, u) is* subconnected *if it is subconnected at u.*

Note that, alternatively, one can define the subconnectedness of (T, X, u) in v as: (1) $G[T_v]$ is connected, and (2) for any child w of v in T_v, (T, X, u) is subconnected at w.

We now describe an elementary procedure, called `split`, whose iterative application transforms a tree-decomposition into a subconnected tree-decomposition. The procedure `split` takes as input (1) a rooted tree-decomposition (T, X, u) of a connected graph G, and (2) a node $v \in V(T_u)$, $v \neq u$, such that, for every child w of v in T_u, (T, X, u) is subconnected at w. `split` returns a rooted tree-decomposition (T', X', u) of G that is equal to (T, X, u), except at node v. Roughly speaking, node v in (T, X, u) is replaced by several nodes $v_1, \ldots, v_\ell$ in (T', X', u). The v_i's have the parent of v in (T, X, u) as parent in (T', X', u). The children of v in (T, X, u) are distributed among the v_i's. Procedure `split` satisfies that (T',X',u) is subconnected at every v_i, $i=1,\ldots,\ell$. Hereafter, we describe formally this procedure:

Procedure `split`: Let (T,X,u) be a rooted tree-decomposition of a connected graph G, of width k. Let $v \in V(T_u)$, $v \neq u$, such that:

- (T,X,u) is not subconnected at v;
- for every child w of v, (T,X,u) is subconnected at w.

Since G is connected, and since (T,X,u) is not subconnected at v but subconnected at every child of v, the subgraph of G induced by the nodes in the bag X_v is not connected. Let Y_i, $i=1,\ldots,r$ be the decomposition of X_v in connected components (i.e., the $G[Y_i]$'s are the connected components of $G[X_v]$). Let v' be the parent of v in T_u. Procedure `split` proceeds as follows:

Case 1: v is a leaf of T_u. `split` replaces v by r nodes $v_1,\ldots,v_r$, all connected to v', and every corresponding bag X_{v_i} is set to Y_i.

Case 2: v has s children $w_1,\ldots,w_s$, $s \geq 1$. Let Z_i, $i=1,\ldots,t$, be the connected components of $G[T_v]$. As we will prove later, there is a partition $\{I_i, i=1,\ldots,t\}$ of $\{1,\ldots,r\}$, and a partition $\{J_i, i=1,\ldots,t\}$ of $\{1,\ldots,s\}$ such that, for every $i=1,\ldots,t$:

$$Z_i = \Big(\cup_{j\in I_i} Y_j\Big) \cup \Big(\cup_{j\in J_i} G[T_{w_j}]\Big). \tag{3}$$

Procedure `split` replaces v by t nodes $v_1,\ldots,v_t$, all connected to v' (cf. Fig. 1). For every $i=1,\ldots,t$, node v_i is the parent of w_j for all $j \in J_i$, and the bag X_{v_i} corresponding to v_i is set to $\cup_{j\in I_i} Y_j$.

Using the same notations as in the description of Procedure `split`, we have:

Lemma 1. *Procedure `split` applied to node v returns a rooted tree-decomposition (T',X',u) of G, of width $\leq k$. The tree-decomposition (T',X',u) differs from (T,X,u) only at v, which is replaced by some nodes $v_1,\ldots,v_\ell$. (T',X',u) is subconnected at v_i, $i=1,\ldots,\ell$. Moreover, for every node z such that (T,X,u) is subconnected at z, (T',X',u) remains subconnected at z.*

Proof. The lemma clearly holds in Case 1. Hence we concentrate our attention to Case 2. In this case, $\ell = t$. Obviously, since the modification of T occurs at node v only, for every node z such that (T,X,u) is subconnected at z, (T',X',u) remains subconnected at z. Hence, we focus on the transformation of v into $v_1,\ldots,v_t$. First, let us show that Equation (3) holds. Let H be the bipartite graph whose one partition consists of r nodes $Y_1,\ldots,Y_r$, and the other partition consists of s nodes $w_1,\ldots,w_s$ (cf. Fig. 1). There is an edge between Y_i and w_j if and only if there is a node x of G that belongs to $Y_i \cap X_{w_j}$. By construction, there is a one-to-one correspondence between the connected components of $G[T_v]$ and the connected components of H. Equation (3) follows from this correspondence. From the construction of the v_i's, based on Equation (3), (T',X',u)

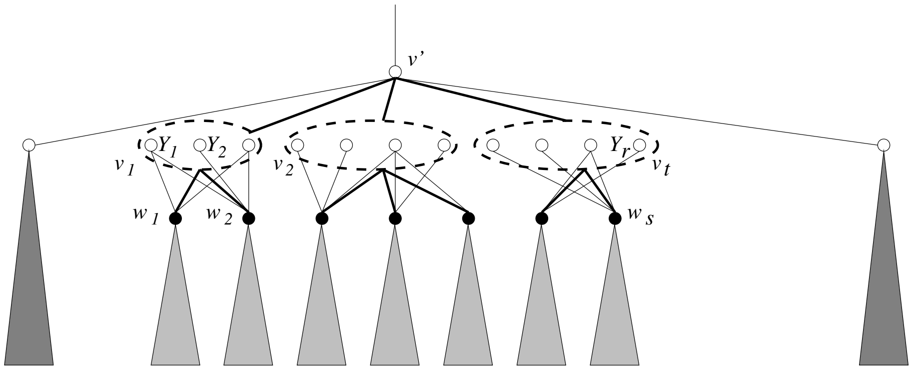

Fig. 1. The procedure `split` replaces node v by t nodes $v_1, \dots, v_t$. The Y_i's form a partition of X_v in connected components. The w_i's are the children of v. Node v' is the parent of v.

is subconnected at v_i, $i = 1, \dots, t$. Thus, it remains to prove that (T', X') is a tree-decomposition of G, of width $\leq k$.

Since $X'_{v_i} = \cup_{j \in I_i} Y_j$, and the I_i's form a partition of $\{1, \dots, r\}$, we get that $\cup_{i=1,\dots,r} X'_{v_i} = \cup_{j=1,\dots,r} Y_j = X_v$.

Thus every node of G appears in at least one bag, i.e., **C1** holds. Every edge of G appears in at least one bag too, because the Y_i's are the connected components of X_v, and thus there is no edge between nodes that belong to two different Y_i's. I.e., **C2** holds

Non surprisingly, Condition **C3** of tree-decomposition is the most tricky to check. Let x, y, z be three pairwise distinct nodes of T' with y on the path between x and z. Let us show that $X'_x \cap X'_z \subseteq X'_y$. Obviously, this claim holds if $v_i \notin \{x, y, z\}$ for all $i = 1, \dots, r$, because, in this case, the considered bags of T' are exactly those of T. The claim also holds if the path P from x to z contains two v_i's, because, in this case, $X'_x \cap X'_z = \emptyset$. Thus, in the following, we consider the case where the path P contains exactly one v_i. There are two subcases, depending on whether the node of P that belongs to $\{v_1, \dots, v_t\}$ is y, or one of the two end-points x or z.

Assume that $x = v_i$ for some $i \in \{1, \dots, r\}$, and that $v_j \notin \{y, z\}$ for all $j \neq i$. Then, $X'_x \cap X'_z \subseteq X_v \cap X'_z = X_v \cap X_z \subseteq X_y = X'_y$, and thus Condition **C3** holds.

Assume that $y = v_i$ for some $i \in \{1, \dots, r\}$, and that $v_j \notin \{x, z\}$ for all $j \neq i$. Assume, w.l.o.g., that $x \in V(T'_{v_i})$. Node z then belongs either to $V(T'_{v_i})$, or to the subtree of T containing v', obtained after removing the edge $\{v, v'\}$ from T. In both cases, $X'_x \cap X'_z = X_x \cap X_z \subseteq X_v$. Moreover, by construction of the bipartite graph H (cf. Fig. 1), $X'_x \cap X_v \subseteq \cup_{j \in I_i} Y_j$. Therefore, $X'_x \cap X'_z \subseteq \cup_{j \in I_i} Y_j = X'_{v_i}$, and thus Condition **C3** holds.

To complete the proof, observe that applying Procedure `split` can only decrease the width of the tree-decomposition since one bag is split into several smaller bags. Hence, the width of (T', X', u) is $\leq k$. $\square$

3 Connected Treewidth

In this section, we mainly prove the following result:

Theorem 1. *There exists a $O(nk^3)$-time algorithm that, given any n-node tree-decomposition of a connected graph G, of width k, returns a an $O(nk)$-node connected tree-decomposition of G, of width $\leq k$.*

Corollary 1. *For any connected graph G, $\mathbf{ctw}(G) = \mathbf{tw}(G)$.*

Proof of Theorem 1. We prove that Algorithm `make-it-connected` described on Figure 2 satisfies the statement of the theorem. This algorithm proceeds in two phases. Phase 1 proceeds upward the tree (rooted in an arbitrary node u). Phase 2 proceeds downward the tree. Let k be the width of the input tree-decomposition (T, X). Let us first prove the following:

Claim 1. At the end of Phase 1, the tree-decomposition (T, X, u) is subconnected, and its width is $\leq k$.

At every application of the while-loop in Phase 1, the selected node $v \in W$ satisfies the condition of application of Procedure `split`. The while-loop stops after $W = \emptyset$. Therefore, by Lemma 1, for every node $v \neq u$, (T, X, u) is subconnected at v. It remains to check that (T, X, u) is subconnected at u. The bags in T_u contain all nodes of G. Therefore, since G is connected, T_u is connected as well, and thus (T, X, u) is subconnected at u, i.e., (T, X, u) is subconnected. By Lemma 1, the application of Procedure `split` does not increase the width of the current tree-decomposition, therefore the width of the tree-decomposition resulting from Phase 1 has width $\leq k$.

The aim of Phase 2 is to transform the current subconnected tree-decomposition (T, X, u) into a connected tree-decomposition (possibly rooted at another node r).

Claim 2. At the end of Phase 2, the tree-decomposition (T, X, r) is connected, and its width is $\leq k$.

To prove that claim, we will prove the following invariant, satisfied after every application of the while-loop in Phase 2:

- **I1:** (T, X, r) is subconnected;
- **I2:** C is the set of edges e of T corresponding to non connected e-cuts.

These two statements are satisfied before the first application of the while-loop. Let $e = \{v, w\}$ be as specified in Algorithm `make-it-connected`. If $v \neq r$, let v' be the parent of v in T_r, possibly $r = v'$. (If $v = r$, then there is simply no need to define node v'.) Let S be the subtree of T containing v' obtained after removing $\{v, v'\}$ from T (cf. Fig. 3). From the choice of e, $\{v, v'\} \notin C$, and thus $G[S]$ is connected. Let $w_1, \ldots, w_s$ be the children of v in T_r, different from w, and let S_i be the subtree of T_r rooted at w_i, $i = 1, \ldots, s$. Since (T, X, r) is subconnected, $G[S_i]$ is connected for every i. Finally, let R be the subtree of T_r

```
Input: A tree-decomposition (T, X) of a connected graph G.
begin
   Pick any node u of T, and root T at u;
   /* Phase 1 */
   W ← {v ∈ V(T_u) | v ≠ u and (T, X, u) is not subconnected at v};
   while W ≠ ∅ do
      Let v ∈ W such that, for every child w of v, w ∉ W;
      (T, X, u) ← split(G, (T, X, u), v);
      W ← W \ {v};
   endwhile
   /* Phase 2 */
   r ← u; /* r is the root of T */
   C ← {e ∈ E(T) | the e-cut of T is not connected}
   while C ≠ ∅ do
      Let e = {v, w} ∈ C where v is the parent of w, and
            no edge on the path from r to v is in C;
      r ← w; /* the root is modified */
      (T, X, r) ← split(G, (T, X, r), v);
      C ← C \ {e};
      for all children w_j ≠ w of v such that {v, w_j} ∈ C do
         Let v_i be the new parent of w_j after application of split;
         Replace {v, w_j} by {v_i, w_j} in C;
      endfor
   endwhile
   return (T, X, r);
end.
```

Fig. 2. Algorithm `make-it-connected`.

rooted at w. Again, since (T,X,r) is subconnected, $G[R]$ is connected. Before application of Procedure `split` at v, (T,X) is re-rooted at w. Since $\{v,w\} \in C$, (T,X,w) is not subconnected at v. However, (T,X,w) is subconnected at every child $v', w_1, \ldots, w_s$ of v in (T,X,w). Therefore, we are in the condition of the application of Procedure `split`.

Invariant **I1** is satisfied. Indeed, after application of Procedure `split` at v in (T,X,w), (T,X,w) becomes subconnected at the new nodes $v_1, \ldots, v_\ell$. The only other nodes in which subconnectivity is questionable is along the path between r and v'. Since all edges of this path corresponds to a connected cut (from the choice of $\{v,w\}$), the subconnectivity remains satisfied along this path. ((T,X,w) is subconnected at w because G is connected.)

Now we prove that Invariant **I2** is also satisfied, that is all non-connected cuts in (T,X,w) are in C. Let us first consider the edge $f = \{v_i, v'\}$ introduced by Procedure `split` in (T,X,w). On one side of f, there is S, and on the other side there is $T \setminus S$. Both $G[S]$ and $G[T \setminus S]$ are connected because they correspond to the $\{v,v'\}$-cut, which is connected.

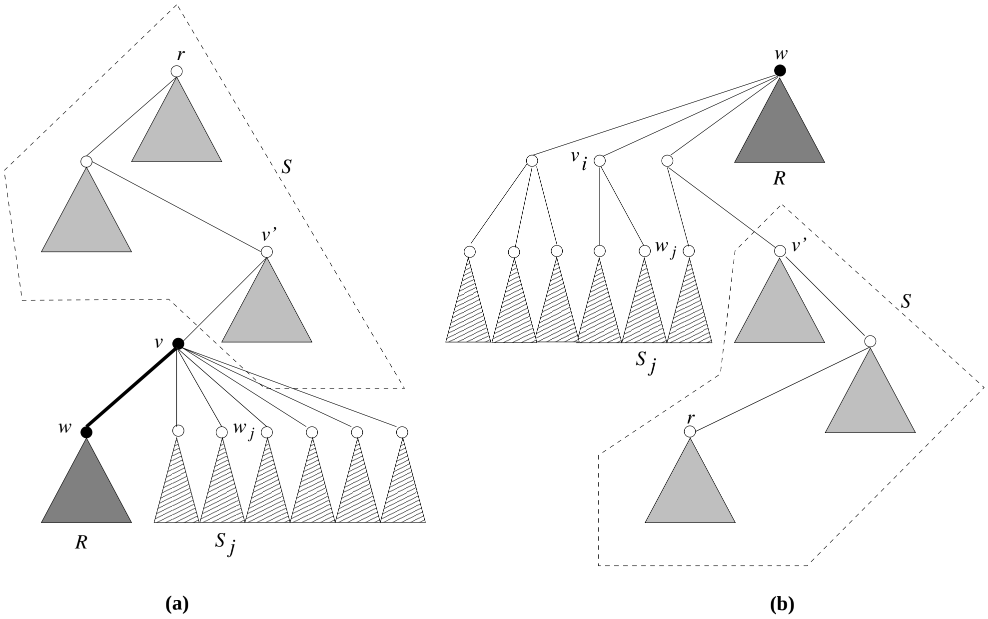

Fig. 3. In the second phase of Algorithm `make-it-connected`, the tree-decomposition is re-rooted at w before application of Procedure `split` at node v. (a) represents the tree rooted at r, and (b) represents the same tree, re-rooted at w, after application of Procedure `split` at v.

Next, we consider an edge $f = \{v_i, w\}$ introduced by Procedure `split` in (T, X, w). On one side of f, there is the subtree T_{v_i} of (T, X, w) rooted at v_i. Since Procedure `split` makes (T, X, w) subconnected at v_i, we get that $G[T_{v_i}]$ is connected. Let us show that $G[T \setminus T_{v_i}]$ is also connected. Let $j \neq i$. Since G is connected, let us consider an edge $\{x, y\}$ where $x \in G[T_{v_j}]$, and $y \notin G[T_{v_j}]$. By the property of tree-decomposition, there is a bag of $T \setminus T_{v_j}$ containing $\{x, y\}$, and thus $x \in X_w$. Therefore, for any $j \neq i$, $V(T_{v_j}) \cap R \neq \emptyset$. Since R is connected, we get that $G[T \setminus T_{v_i}]$ is connected. Therefore, the considered f-cut is connected.

Finally, we consider an edge $f = \{v_i, w_j\}$. On one side of f, there is S_j, and on the other side there is $T \setminus S_j$. $G[S_j]$ is connected because (T, X, r) is subconnected at w_j. $G[T \setminus S_j]$ is not necessarily connected. However, if it is not connected, then is was not connected before the application of the Procedure `split`. In other words, if the f-cut is not connected, then the f'-cut corresponding to $f' = \{v, w_j\}$ is not connected. Algorithm `make-it-connected` takes this into account by replacing $\{v, w_j\}$ by $\{v_i, w_j\}$ in C. This completes the proof that Invariant **I2** is satisfied.

We focus now on the time complexity of Algorithm `make-it-connected`. Before starting Phase 1, one can decompose all bags of the tree-decomposition (T, X) in connected components. This takes time $O(\sum_{v \in V(T)} |X_v|^2) \leq O(nk^2)$.

Let v be a node where Procedure `split` is applied during Phase 1. Assume that, when this occurs, v has d_v children in the current tree-decomposition. The corresponding bipartite graph H has thus one partition with d_v nodes $w_1, \ldots, w_{d_v}$. The other has r nodes $Y_1, \ldots, Y_r$. For each pair (Y_i, w_j), it takes time $O(|X_{w_j}| \cdot |Y_i|)$ to determine whether the edge $\{Y_i, w_j\}$ belongs to H, because one just needs to check whether or not $Y_i \cap X_{w_j} \neq \emptyset$. H can be constructed in time $O(\sum_{i,j} |X_{w_j}| \cdot |Y_i|) \leq O(\sum_j k \cdot |X_{w_j}|) \leq O(\sum_j k^2) \leq O(k^2 d_v)$. The connected components of H can be computed in parallel with its construction. Therefore, applying procedure `split` at node v takes a total time of $O(k^2 d_v)$. Now, observe that, after v has been split in ℓ nodes $v_1, \ldots, v_\ell$, we have $d_v = \sum_{i=1}^{\ell} d_{v_i}$. Therefore, summing up the costs of all applications of procedure `split` during Phase 1 results in time $O(k^2 \sum_{v \in V(T)} d_v) = O(|V(T)| k^2)$ where T is the tree-decomposition after Phase 1. Starting from an n-node tree-decomposition, Phase 1 produces a tree-decomposition with at most kn nodes. Therefore, Phase 1 takes time $O(nk^3)$. By similar arguments, one can show that Phase 2 takes time $O(nk^3)$ too, which completes the proof. □

4 Connected Graph Searching

In this section, we apply Theorem 1 to the design of an approximation algorithm for graph searching. Formally, a *search strategy* for a graph G is an ordered sequence of *search steps* where each step is an operation that consists in one of the following: (1) "placing a searcher at $u \in V(G)$", (2) "removing a searcher from $u \in V(G)$", or (3) "moving a searcher along an edge from one extremity to the other". Initially, all the edges of the graph are said to be "contaminated". When a searcher moves from u to and adjacent node v, the edge $\{u, v\}$ becomes *clear*. A clear edge e is preserved from recontamination if every path between e to a contaminated edge e' has a searcher occupying some of its nodes. A search strategy completes when all edges are clear. The search number $\mathbf{s}(G)$ of a graph G is a minimum number of searchers required to clear G. A search strategy is connected if, at every step, the set of clear edges induces a connected subgraph. The connected search number $\mathbf{cs}(G)$ of a connected graph G is a minimum number of searchers required to clear G by a connected search strategy.

Let $t(n)$ be the time-complexity of the fastest algorithm for approximating the treewidth of an n-node graph, up to a factor $O(\sqrt{\log OPT})$ (the best bound known so far is in [7]). From now on, all graphs are supposed to be simple (i.e., without loops and double edges).

Theorem 2. – *For any connected n-node graph G, $\frac{\mathbf{cs}(G)}{\mathbf{s}(G)} \leq \log n + 1$.*

– *There exists an $O(t(n) + nk^3 \log^{3/2} k + m \log n)$-time $O(\log n \sqrt{\log OPT})$-approximation algorithm for connected search in n-node m-edge graphs of treewidth k. More precisely, given any connected graph G, the algorithm returns a connected search strategy for G using at most $O(\mathbf{cs}(G) \log n \log \mathbf{cs}(G))$ searchers.*

Proof. We use the notion of *crusade*, introduced by Bienstock and Seymour [4]. Let G be a graph. For $E \subseteq E(G)$, let $\delta(E)$ be the set of vertices which are endpoints of an edge in E and an edge in $E(G) \setminus E$. A k-crusade in G is a sequence $(E_0, E_1, \cdots, E_r)$ of subsets of $E(G)$, such that $E_0 = \emptyset$, $E_r = E(G)$, $|E_i \setminus E_{i-1}| \leq 1$ for $1 \leq i \leq r$, and $|\delta(E_i)| \leq k$ for $0 \leq i \leq r$. If k denotes the smallest number for which there is a k-crusade in G, then $k \leq \mathbf{s}(G) \leq k+1$ (cf. [3]). A crusade $(E_0, E_1, \cdots, E_r)$ in a graph G is connected if E_i induces a connected subgraph of G for every $i = 1, \ldots, r$. It is easy to check that if k denotes the smallest number for which there is a connected k-crusade in G, then $\mathbf{cs}(G) \leq k+1$.

Claim 3. For any connected n-nodes graph G, and for any $e \in E(G)$, there exists a connected k-crusade $(E_0, E_1, \ldots, E_r)$ of G with $k \leq \mathbf{tw}(G) \log n$ and $E_1 = \{e\}$.

The proof of Claim 3 is by induction on n. If $n \in \{1, 2\}$, the result obviously holds. Let $n \geq 3$ and let us assume that, for any $n' < n$, Claim 3 holds. Let G be an n-node connected graph, and let $e \in E(G)$. Let (T, X) be a connected tree decomposition of G, of width $\mathbf{tw}(G)$. For a subtree T' of T, let us denote by $G[T']$ the subgraph of G induced by the nodes in the bags of T'. Theorem 2.5 in [13] specifies that, for any tree-decomposition:

- either there exists $u \in V(T)$ such that removing u from T results in subtrees $T_1, \cdots, T_q$ of T, with $|V(G[T_i])| \leq n/2$ for every $1 \leq i \leq q$,
- or there exists $\{u, u'\} \in E(T)$ such that removing $\{u, u'\}$ from T results in subtrees $T_1, \cdots, T_q$ of T, with $|V(G[T_i])| \leq n/2$ for every $1 \leq i \leq q$.

We consider the two cases ("node-centroid" and "edge-centroid") separatly.

Let us first consider the former case. Let $u \in V(T)$ such that for all $1 \leq i \leq q$, $|V(G[T_i])| \leq n/2$. By definition of connected tree decomposition, $G[T_i]$ is a connected subgraph of G. Assume, w.l.o.g., that there exists $v \in V(T_1)$ such that $e \in X_v$. By induction, let $(E_0^{(1)}, E_1^{(1)}, \ldots, E_{r_1}^{(1)})$ be a connected k-crusade of $G[T_1]$, with $k \leq \mathbf{tw}(G[T_1]) \log |G[T_1]|$, and $E_1^{(1)} = \{e\}$. We set $E_i = E_i^{(1)}$ for $i = 0, \ldots, r_1$. Since $\mathbf{tw}(G[T_1]) \leq \mathbf{tw}(G)$ and $|G[T_1]| \leq n/2$, we get $|\delta(E_i)| \leq \mathbf{tw}(G)(\log n - 1)$ for $i = 0, \ldots, r_1$. Since G is connected, there is $f \in E(G) \setminus E_{r_1}$ such that the subgraph induced by $E_{r_1} \cup \{f\}$ is connected.

- If $f \in X_u$, then let $E_{r_1+1} = E_{r_1} \cup \{f\}$. For computing $|\delta(E_{r_1+1})|$, consider any node $x \in \delta(E_{r_1+1})$. This node is incident to an edge in E_{r_1+1} and to an edge in $E(G) \setminus E_{r_1+1}$. Therefore $x \in X_u \cup (\cup_{v \in T_1} X_v)$ and $x \in \cup_{v \notin T_1} X_v$. As a consequence, by the third property of a tree-decomposition, $x \in X_u$ and thus $|\delta(E_{r_1+1})| \leq |X_u| \leq \mathbf{tw}(G)$.
- If $f \notin X_u$, then there exists $i \in \{2, \cdots, q\}$ and $v \in V(T_i)$ such that $f \in X_v$. Assume, w.l.o.g., that $i = 2$. Let $(E_0^{(2)}, E_1^{(2)}, \ldots, E_{r_2}^{(2)})$ be a connected k-crusade of $G[T_2]$, with $k \leq \mathbf{tw}(G[T_2]) \log |G[T_2]| \leq \mathbf{tw}(G)(\log n - 1)$ and $E_1^{(2)} = \{f\}$. For every $1 \leq i \leq r_2$, we set $E_{r_1+i} = E_{r_1} \cup E_i^{(2)}$. By construction, for any $1 \leq i \leq r_2$, $G[E_{r_1+i}]$ is a connected subgraph of G. Let $i \in \{1, \ldots, r_2\}$, and $x \in \delta(E_{r_1+i})$. We have $x \in \delta(E_i^{(2)}) \cup X_u$, and thus $|\delta(E_{r_1+i})| \leq \mathbf{tw}(G)(\log n - 1) + \mathbf{tw}(G) \leq \mathbf{tw}(G) \log n$.

We iterate the process until all edges of G are in the crusade, completing the analysis of the first case, i.e., where the "centroid" is a node. Due to lack of space, the second case, where the "centroid" is an edge, is omitted, but can be treated similarly as above. This completes the proof of Claim 3.

Item 1 of Theorem 2 is a direct consequence of Claim 3. Indeed, let k be the smallest integer such that there exists a connected k-crusade in G. We have seen that $\mathbf{cs}(G) \leq k+1$. Thus, by Claim 3, $\mathbf{cs}(G) \leq \mathbf{tw}(G)\log n + 1$. Since $\mathbf{tw}(G) \leq \mathbf{s}(G)$, we get $\mathbf{cs}(G) \leq \mathbf{s}(G)\log n + 1$ and thus $\mathbf{cs}(G)/\mathbf{s}(G) \leq \log n + 1$. Item 2 is obtained by combining claim 3 with the algorithm of Feige et al. [7]. □

References

1. L. Barrière, P. Flocchini, P. Fraigniaud, and N. Santoro. Capture of an intruder by mobile agents. In 14th ACM Symp. on Parallel Algorithms and Architectures (SPAA), pages 200-209, 2002.
2. L. Barrière, P. Fraigniaud, N. Santoro, and D. Thilikos. Connected and Internal Graph Searching. In 29th Workshop on Graph Theoretic Concepts in Computer Science (WG), Springer-Verlag, LNCS 2880, pages 34–45, 2003.
3. D. Bienstock. Graph searching, path-width, tree-width and related problems (a survey). DIMACS Series in Discrete Mathematics and Theoretical Computer Science 5, pages 33-49, 1991.
4. D. Bienstock and P. Seymour. Monotonicity in graph searching. Journal of Algorithms 12, pages 239-245, 1991.
5. R. Breisch. An intuitive approach to speleotopology. Southwestern Cavers VI(5), pages 72-78, 1967
6. J. A. Ellis, I.H. Sudborough, J.S. Turner. The Vertex Separation and Search Number of a Graph Information and computation 113, pages 50-79, 1994.
7. U. Feige, M. Hajiaghayi, and J. Lee. Improved approximation algorithms for minimum-weight vertex separators. In 37th ACM Symposium on Theory of Computing (STOC), 2005.
8. F. Fomin, P. Fraigniaud, D. Thilikos. The Price of Connectedness in Expansions. Technical Report LSI-04-28-R, UPC Barcelona, 2004.
9. M. C. Golumbic. Algorithmic graph theory and perfect graphs. Computer Science and Applied Mathematics, 1980.
10. N. Megiddo, S. Hakimi, M. Garey, D. Johnson and C. Papadimitriou. The complexity of searching a graph. Journal of the ACM 35(1), pages 18-44, 1988.
11. A. Parra and P. Scheffler. Characterizations and algorithmic applications of chordal graphe embeddings. Discrete Applied Mathematics 79, pages 171-188, 1997.
12. T. Parson. Pursuit-evasion in a graph. Theory and Applications of Graphs, Lecture Notes in Mathematics, Springer-Verlag, pages 426-441, 1976.
13. N. Robertson and P. D. Seymour. Graph minors II, Algorithmic Aspects of Tree-Width. Journal of Algorithms 7, pages 309-322, 1986.
14. N. Robertson and P. D. Seymour. Graph minors X, Obstructions to tree-decomposition. Journal Combin. Theory B, Vol. 52, pages 153-190, 1991.
15. P. Seymour and R. Thomas. Call routing and the rat-catcher. Combinatorica 14(2), pages 217–241, 1994.
16. B. Yang, D. Dyer, and B. Alspach. Sweeping Graphs with Large Clique Number. In 5th International Symposium on Algorithms and Computation (ISAAC), Springer, LNCS 3341, pages 908-920, 2004.