Discrete Mathematics and Theoretical Computer Science DMTCS vol. **8**, 2006, 65–82

Recognizing HH-free, HHD-free, and Welsh-Powell Opposition Graphs

Stavros D. Nikolopoulos and Leonidas Palios

Department of Computer Science, University of Ioannina, P.O.Box 1186, GR-45110 Ioannina, Greece
{stavros, palios}@cs.uoi.gr

received Oct 7, 2004, *revised Nov 11, 2005*, *accepted Jan 8, 2006*.

In this paper, we consider the recognition problem on three classes of perfect graphs, namely, the HH-free, the HHD-free, and the Welsh-Powell opposition graphs (or WPO-graphs). In particular, we prove properties of the chordal completion of a graph and show that a modified version of the classic linear-time algorithm for testing for a perfect elimination ordering can be efficiently used to determine in $O(n\ \min\{m\,\alpha(n,n),\ m + n\log n\})$ time whether a given graph G on n vertices and m edges contains a house or a hole; this implies an $O(n\ \min\{m\,\alpha(n,n),\ m + n\log n\})$-time and $O(n+m)$-space algorithm for recognizing HH-free graphs, and in turn leads to an HHD-free graph recognition algorithm exhibiting the same time and space complexity. We also show that determining whether the complement $\overline{G}$ of the graph G is HH-free can be efficiently resolved in $O(nm)$ time using $O(n^2)$ space, which leads to an $O(n\,m)$-time and $O(n^2)$-space algorithm for recognizing WPO-graphs. The previously best algorithms for recognizing HH-free, HHD-free, and WPO-graphs required $O(n^3)$ time and $O(n^2)$ space.

Keywords: HH-free graph, HHD-free graph, Welsh-Powell opposition graph, perfectly orderable graph, recognition.

1 Introduction

A linear order $\prec$ on the vertices of a graph G is *perfect* if the ordered graph $(G, \prec)$ contains no induced P_4 $abcd$ with $a \prec b$ and $d \prec c$ (such a P_4 is called an *obstruction*). In the early 1980s, Chvátal [2] defined the class of graphs that admit a perfect order and called them *perfectly orderable* graphs. The interest in perfectly orderable graphs comes from the fact that several problems in graph theory, which are NP-complete in general graphs, have polynomial-time solutions in graphs that admit a perfect order [1; 5]; unfortunately, it is NP-complete to decide whether a graph admits a perfect order [12]. Since the recognition of perfectly orderable graphs is NP-complete, we are interested in characterizing graphs which form polynomially recognizable subclasses of perfectly orderable graphs. Many such classes of graphs, with very interesting structural and algorithmic properties, have been defined so far and shown to admit polynomial-time recognitions (see [1; 5]); note however that not all subclasses of perfectly orderable graphs admit polynomial-time recognition [7].

In this paper, we consider the class of HH-free graphs and two classes of perfectly orderable graphs, namely, the HHD-free, and the Welsh-Powell opposition graphs; note that the complement of an HH-free graph is perfectly orderable (this was conjectured by Chvátal and proved by Hayward [6]). A graph is

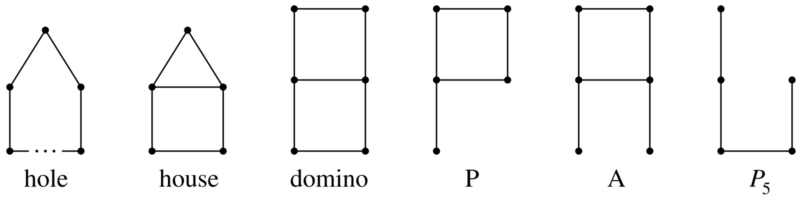

Fig. 1: Some simple graphs.

HH-free if it contains no hole (i.e., a chordless cycle on ≥ 5 vertices) and no house as induced subgraphs (see Figure 1). The HH-free graphs properly generalize the class of triangulated (or chordal) graphs, i.e., graphs with no induced chordless cycles of length greater than or equal to four [5]. A subclass of HH-free graphs, which also properly generalizes the class of triangulated graphs, is the class of HHD-free graphs; a graph is *HHD-free* if it contains no hole, no house, and no domino as induced subgraphs (see Figure 1). In [8], Hoàng and Khouzam proved that the HHD-free graphs admit a perfect order, and thus are perfectly orderable.

A graph is called an *Opposition graph* if it admits a linear order $\prec$ on its vertices such that there is no P_4 $abcd$ with $a \prec b$ and $c \prec d$. Opposition graphs belong to the class of **bip*** graphs (see [1]), and hence are perfect [13]. The complexity of recognizing opposition graphs is unknown. It is also open whether there is an opposition graph that is not perfectly orderable [1]. The class of opposition graphs contains several known classes of perfectly orderable graphs. For example, bipolarizable graphs are, by definition, opposition graphs; a graph is bipolarizable if it admits a linear order $\prec$ on its vertices such that every P_4 $abcd$ has $b \prec a$ and $c \prec d$ [14]. Another subclass of opposition graphs, which we study in this paper, are the Welsh-Powell opposition graphs. A graph is defined to be a *Welsh-Powell Opposition graph* (or WPO-graph for short), if it is an opposition graph for every Welsh-Powell ordering; a Welsh-Powell ordering for a graph is an ordering of its vertices in nondecreasing degree [18].

Hoàng and Khouzam [8], while studying the class of brittle graphs (a well-known class of perfectly orderable graphs which contains the HHD-free graphs), showed that HHD-free graphs can be recognized in $O(n^4)$ time, where n denotes the number of vertices of the input graph. An improved result was obtained by Hoàng and Sritharan [9] who presented an $O(n^3)$-time algorithm for recognizing HH-free graphs and showed that HHD-free graphs can be recognized in $O(n^3)$ time as well; one of the key ingredients in their algorithms is the reduction to the recognition of triangulated graphs. Recently, Eschen *et al.* [4] described recognition algorithms for several classes of perfectly orderable graphs, among which a recognition algorithm for HHP-free graphs; a graph is HHP-free if it contains no hole, no house, and no "P" as induced subgraphs (see Figure 1). Their algorithm is based on the property that every HHP-free graph is HHDA-free graph (a graph with no induced hole, house, domino, or "A"), and thus a graph G is HHP-free graph if and only if G is a HHDA-free and contains no "P" as an induced subgraph. The characterization of HHDA-free graphs due to Olariu (a graph G is HHDA-free if and only if every induced subgraph of G either is triangulated or contains a non-trivial module [14]) and the use of modular decomposition [11] allowed Eschen *et al.* to present an $O(n\,m)$-time recognition algorithm for HHP-free graphs.

For the class of WPO-graphs, Olariu and Randall [15] gave the following characterization: a graph G is WPO-graph if and only if G contains no induced C_5 (i.e., a hole on 5 vertices), house, P_5, or "P" (see Figure 1). It follows that G is a WPO-graph if and only if G is HHP-free and $\overline{G}$ is HH-free. Eschen *et al.* [4] combined their $O(n\,m)$-time recognition algorithm for HHP-free graphs with the $O(n^3)$-time recognition algorithm for HH-free graphs proposed in [9], and showed that WPO-graphs can be recognized in $O(n^3)$ time.

In this paper, we present efficient algorithms for recognizing HH-free, HHD-free, and WPO-graphs. We show that a variant of the classic linear-time algorithm for testing whether an ordering of the vertices of a graph is a perfect elimination ordering can be used to determine whether a vertex of a graph G belongs to a hole or is the top of a house or a building in G. We take advantage of properties characterizing the chordal completion of a graph and show how to efficiently compute for each vertex v the leftmost among v's neighbors in the chordal completion which are to the right of v, without explicitly computing the chordal completion. As a result, we obtain an $O(n \min\{m\,\alpha(n,n),\, m + n\log n\})$-time and $O(n+m)$-space algorithm for determining whether a graph on n vertices and m edges is HH-free; we also describe how the algorithm can be augmented to provide a certificate (an induced house or hole) whenever it decides that the input graph is not HH-free.

Our HH-free graph recognition algorithm, in conjunction with results by Jamison and Olariu [10] and by Hoàng and Khouzam [8], enables us to describe an algorithm for recognizing HHD-free graphs which also runs in $O(n \min\{m\,\alpha(n,n),\, m + n\log n\})$ time and requires $O(n+m)$ space. Additionally, for a graph G on n vertices and m edges, we show that we can detect whether the complement $\overline{G}$ of G is HH-free in $O(n\,m)$ time using $O(n^2)$ space. In light of the characterization of WPO-graphs due to Olariu and Randall [15] which implies that a graph G is a WPO-graph if and only if G is HHP-free and its complement $\overline{G}$ is HH-free, and the $O(n\,m)$-time recognition algorithm for HHP-free graphs of Eschen *et al.* [4], our result yields an $O(n\,m)$-time and $O(n^2)$-space algorithm for recognizing WPO-graphs.

The paper is structured as follows. In Section 2, we review the terminology that we use throughout the paper and we present well known results that are useful for our algorithms. In Section 3, we present the methodology and establish properties that enable us to efficiently determine whether a given graph is HH-free, describe the algorithm, and give its analysis and the certificate computation. The recognition algorithms for HHD-free graphs and WPO-graphs are described and analyzed in Section 4. Section 5 summarizes our results and presents some open problems.

2 Preliminaries

We consider finite undirected graphs with no loops or multiple edges. Let G be such a graph; then, $V(G)$ and $E(G)$ denote the set of vertices and of edges of G respectively. The subgraph of a graph G induced by a subset S of G's vertices is denoted by $G[S]$. A subset $B \subseteq V(G)$ of vertices is a *module* if $2 \leq |B| < |V(G)|$ and each vertex $x \in V(G) - B$ is adjacent to either all vertices or no vertex in B. The *neighborhood* $N(x)$ of a vertex $x \in V(G)$ is the set of all the vertices of G which are adjacent to x. The *closed neighborhood* of x is defined as $N[x] := N(x) \cup \{x\}$. We use $M(x)$ to denote the set $V(G) - N[x]$ of non-neighbors of x. Furthermore, for a vertex $y \in M(x)$, we use $n(x,y)$ to denote the number of vertices in the set $N(x) \cap N(y)$, i.e., the set of common neighbors of x and y, or equivalently, the degree of the vertex y in the graph induced by the set $N(x) \cup \{y\}$. The *degree* of a vertex x in a graph G, denoted $deg(x)$, is the number of edges incident on x; thus, $deg(x) = |N(x)|$.

A path $v_0 v_1 \cdots v_k$ of a graph G is called *simple* if none of its vertices occurs more than once; it is called a *cycle* (*simple cycle*) if $v_0 v_k \in E(G)$. A simple path (cycle) is *chordless* if $v_i v_j \notin E(G)$ for any two non-consecutive vertices v_i, v_j in the path (cycle). A chordless path (chordless cycle, respectively) on n vertices is commonly denoted by P_n (C_n, respectively). In particular, a chordless path on 4 vertices is denoted by P_4. If $abcd$ is a P_4 of a graph, then the vertices b and c are called *midpoints* and the vertices a

Algorithm PEO(G, σ)

1. **for** each vertex $u \in V(G)$ **do**
    $A(u) \leftarrow \emptyset$;
2. **for** $i \leftarrow 1$ to $n-1$ **do**
3. $u \leftarrow \sigma(i)$;
4. $X \leftarrow \{x \in N(u) \mid \sigma^{-1}(u) < \sigma^{-1}(x)\}$; $\{$*neighbors to the right of u (w.r.t. σ)*$\}$
5. **if** $X \neq \emptyset$
6. **then** $w \leftarrow \sigma(min\{\sigma^{-1}(x) \mid x \in X\})$; $\{$*leftmost (w.r.t. σ) among vertices in X*$\}$
7. concatenate $X - \{w\}$ to $A(w)$;
8. **if** $A(u) - N(u) \neq \emptyset$ **then return**("false");
9. **return**("true");

Fig. 2: The perfect elimination ordering testing algorithm.

and d *endpoints* of the P_4 $abcd$.

Let G be a graph and let x, y be a pair of vertices. If G contains a path from vertex x to vertex y, we say that x *is connected to* y. The graph G is *connected* if x is connected to y for every pair of vertices $x, y \in V(G)$. The *connected components* (or *components*) of G are the equivalence classes of the "is connected to" relation on the vertex set $V(G)$. The *co-connected components* (or *co-components*) of G are the connected components of the complement $\overline{G}$ of the graph G.

A graph G has a *perfect elimination ordering* if its vertices can be linearly ordered $(v_1, v_2, \ldots, v_n)$ so that each vertex v_i is simplicial in the graph $G_i = G[\{v_i, v_{i+1}, \ldots, v_n\}]$ induced by the vertices $v_i, v_{i+1}, \ldots, v_n$, for $1 \leq i \leq n$; a vertex of a graph is *simplicial* if its neighborhood induces a complete subgraph. It is well-known that a graph is triangulated if and only if it has a perfect elimination ordering [1; 5; 16]. The notion of a simplicial vertex was generalized by Jamison and Olariu [10] who defined the notion of a semi-simplicial vertex: a vertex of a graph G is *semi-simplicial* if it is not a midpoint of any P_4 of G. A graph G has a *semi-perfect elimination ordering* if its vertices can be linearly ordered $(v_1, v_2, \ldots, v_n)$ so that each vertex v_i is semi-simplicial in the graph G_i, for $1 \leq i \leq n$. A graph is a *semi-simplicial graph* if and only if it has a semi-perfect elimination ordering (see [4]).

Let $\sigma = (v_1, v_2, \ldots, v_n)$ be an ordering of the vertices of a graph G; $\sigma(i)$ is the i-th vertex in σ, i.e., $\sigma(i) = v_i$, while $\sigma^{-1}(v_i)$ denotes the position of vertex v_i in σ, i.e., $\sigma^{-1}(v_i) = i$, $1 \leq i \leq n$. In Figure 2, we include the classic algorithm PEO(G, σ) for testing whether the ordering σ is a perfect elimination ordering; if the graph G has n vertices and m edges, the algorithm runs in $O(n+m)$ time and requires $O(n+m)$ space [5; 16]. Note that, in Step 4 of the Algorithm PEO(G, σ), the set X is assigned the neighbors of the vertex u which have larger $\sigma^{-1}(\,)$-values; that is, $X = N(u) \cap \{\sigma(i+1), \ldots, \sigma(n)\}$; thus, in Step 6, the vertex w is the neighbor of u in G which is first met among the vertices to the right of u along the ordering σ.

3 Recognizing HH-free graphs

The most important ingredient (and the bottleneck too) of the HHD-free graph recognition algorithm of Hoàng and Sritharan [9] is an algorithm to determine whether a simplicial vertex v of a graph G is *high*, i.e., it is the top of a house or a building[(i)] (or belongs to a hole) in G, which involves the following steps:

- ▷ They compute an ordering of the set $M(v)$ of non-neighbors of v in G where, for two vertices $y, y' \in M(v)$, y precedes y' whenever $n(v,y) \leq n(v,y')$; recall that, for $y \in M(v)$, $n(v,y)$ is the number of common neighbors of v and y. As we will be using this ordering in the description of our approach, we call it a *DegMN-ordering* of $M(v)$.

- ▷ They perform chordal completion on $G[M(v)]$ with respect to a DegMN-ordering of $M(v)$.

- ▷ The vertex v is high if and only if the graph G'_v resulting from G after the chordal completion on $G[M(v)]$ is triangulated.

As we mentioned in the introduction, the algorithm of Hoàng and Sritharan runs in $O(n^3)$ time, where n is the number of vertices of the input graph. In order to be able to beat this, we need to avoid the chordal completion step. Indeed, we show how we can take advantage of the Algorithm PEO and of properties of the chordal completion in order to compute all necessary information without actually performing the chordal completion. In particular, we prove that the following results hold:

Lemma 3.1 *Let G be a graph, v a vertex of G, and $(y_1, y_2, \ldots, y_k)$ a DegMN-ordering of the non-neighbors $M(v)$ of v in G. Moreover, let G'_v be the graph resulting from G after the chordal completion on $G[M(v)]$ with respect to the DegMN-ordering $(y_1, y_2, \ldots, y_k)$ and let $\sigma = (y_1, y_2, \ldots, y_k, x_1, x_2, \ldots, x_{deg(v)}, v)$ where $x_1, x_2, \ldots, x_{deg(v)}$ is an arbitrary ordering of the neighbors of v in G. If Algorithm PEO(G'_v, σ) returns "false" while processing vertex $y_i \in M(v)$, then $A(y_i) - N(y_i) \subseteq N(v)$.*

Proof: Since the Algorithm PEO returns "false" while processing vertex $y_i \in M(v)$, then $A(y_i) - N(y_i) \neq \emptyset$. Suppose that there exists a vertex $y_j \in M(v)$ belonging to $A(y_i) - N(y_i)$. The vertex y_j was added to $A(y_i)$ at Step 7 of a prior iteration of the for-loop, say, while processing vertex y_ℓ. It follows that $\sigma^{-1}(y_\ell) < \sigma^{-1}(y_i) < \sigma^{-1}(y_j)$, and $y_i, y_j \in N(y_\ell)$. Since $y_j \notin N(y_i)$, we have that y_ℓ is not simplicial in $G'_v[\{y_\ell, y_{\ell+1}, \ldots, y_k\}]$; a contradiction to the definition of G'_v. □

Lemma 3.2 *Let G'_v and σ be as in the statement of Lemma 3.1. The vertex v belongs to a C_5 or is the top of a house in the graph G'_v if and only if Algorithm PEO(G'_v, σ) returns "false" while processing vertex z, where $z \in M(v)$.*

Proof: ($\Longleftarrow$) The Algorithm PEO(G'_v, σ) returns "false" while processing vertex z only if at Step 8 there exists a vertex $x \in A(z) - N(z)$. From Lemma 3.1 we have that $A(z) - N(z) \subseteq N(v)$; thus, $x \in N(v)$. The vertex x was added to $A(z)$ at Step 7 of a prior iteration of the for-loop, say, while processing vertex y; then, $\sigma^{-1}(y) < \sigma^{-1}(z) < \sigma^{-1}(x)$, and $yz \in E(G'_v)$ and $xy \in E(G'_v)$. Moreover, since $z \in M(v)$ and $x \notin N(z)$, we have that $y \in M(v)$ and $xz \notin E(G'_v)$. Since $\sigma^{-1}(y) < \sigma^{-1}(z)$, the definition of the DegMN-ordering implies that $n(v,y) \leq n(v,z)$; because $xy \in E(G'_v)$ and $xz \notin E(G'_v)$, there exists a vertex $x' \in N(v)$ such that $x'z \in E(G'_v)$ and $x'y \notin E(G'_v)$. But then, the vertices v, x, x', y, z induce either a C_5 or a house: if $xx' \notin E(G'_v)$ then v belongs to a C_5; otherwise, v is the top of a house.

(i) A building is a graph on vertices $v_1, v_2, \ldots, v_p$, where $p \geq 6$, and edges v_1v_p, v_2v_p, and v_iv_{i+1} for $i = 1, 2, \ldots, p-1$; the vertex v_1 is called the *top* of the building.

($\Longrightarrow$) Among the C_5s of G'_v to which v belongs and the houses of G'_v with v as the top vertex, consider the C_5 or house whose vertices, say, y and z, that belong to $M(v)$ are such that the quantity $|\sigma^{-1}(y)-\sigma^{-1}(z)|$ is minimized. Let x, x' be the remaining two vertices of the C_5 or house, where $x, x' \in N(v)$, $xy \in E(G'_v)$, $xz \notin E(G'_v)$, $x'z \in E(G'_v)$, and $x'y \notin E(G'_v)$, and suppose without loss of generality that $\sigma^{-1}(y) < \sigma^{-1}(z)$ (see Figure 4(a); the dotted edge indicates a potential edge of G).

Next, we show that z is the leftmost among the neighbors of y that are to the right of y (with respect to σ) in G'_v. Suppose for contradiction that the leftmost among these neighbors is $w \neq z$. Then, $\sigma^{-1}(y) < \sigma^{-1}(w) < \sigma^{-1}(z)$, and since $yw \in E(G'_v)$ and $yz \in E(G'_v)$, the definition of the graph G'_v implies that $wz \in E(G'_v)$. Additionally, if $xw \notin E(G'_v)$, then due to the ordering σ, $n(v, y) \leq n(v, w)$, which (because $xy \in E(G'_v)$) implies that there exists a vertex $p \in N(v)$ such that $pw \in E(G'_v)$ and $py \notin E(G'_v)$; but then, the vertices v, x, y, w, p induce a C_5 or a house to which v belongs and $|\sigma^{-1}(y)-\sigma^{-1}(w)| < |\sigma^{-1}(y)-\sigma^{-1}(z)|$, in contradiction to the minimality of the C_5 or house induced by v, x, y, z, x'. Thus, $xw \in E(G'_v)$. Then, if $x'w \notin E(G'_v)$, the vertices v, x, w, z, x' induce a C_5 or a house to which v belongs and $|\sigma^{-1}(w)-\sigma^{-1}(z)| < |\sigma^{-1}(y)-\sigma^{-1}(z)|$, in contradiction to the minimality of the C_5 or house induced by v, x, y, z, x'. If however $x'w \in E(G'_v)$ then, because $n(v, w) \leq n(v, z)$ (since $\sigma^{-1}(w) < \sigma^{-1}(z)$) and because $xw \in E(G'_v)$ whereas $xz \notin E(G'_v)$, there exists a vertex $q \in N(v)$ such that $qz \in E(G'_v)$ and $qw \notin E(G'_v)$; but then, the vertices v, x', w, z, q induce a C_5 or a house to which v belongs and $|\sigma^{-1}(w) - \sigma^{-1}(z)| < |\sigma^{-1}(y) - \sigma^{-1}(z)|$, in contradiction again to the minimality of the C_5 or house induced by v, x, y, z, x'. Therefore, $w = z$. Then, while processing vertex y, the Algorithm PEO includes vertex x in X in Step 4 and later in Step 7 adds x in $A(z)$; then, while processing vertex z, the Algorithm PEO detects that $A(z) - N(z) \neq \emptyset$ since $x \notin N(z)$, and returns "false." □

Lemma 3.1 implies that, while running Algorithm PEO(G'_v, σ), it suffices to collect in the set X (Step 4) only the common neighbors of u and v; in turn, Lemma 3.2 implies that it suffices to execute the for-loop of Steps 2-8 only for the non-neighbors of v.

Additionally, since neither the graph G nor the ordering σ changes during the execution of the Algorithm PEO, then, for each vertex u of G, we can precompute the corresponding vertex w (see Step 6). In fact, in light of Lemma 3.2, we can precompute an array $Next\_Neighbor_{G'[M(v)],\sigma_v}[\,]$, such that for each vertex $u \in M(v)$, the entry $Next\_Neighbor_{G'[M(v)],\sigma_v}[u]$ is equal to the leftmost among the neighbors of u that are to the right of u (with respect to a DegMN-ordering σ_v of the non-neighbors $M(v)$ of v in G) in the chordal completion $G'[M(v)]$ of the subgraph $G[M(v)]$. Note that a vertex $u \in M(v)$ need not have any neighbors among the vertices that follow it in σ_v; in such a case, the entry $Next\_Neighbor_{G'[M(v)],\sigma_v}[u]$ does not get assigned a value, yet, since the for-loop (Steps 2-8) of Algorithm PEO will only be executed for the non-neighbors of v because of Lemma 3.2, we do not need to update any set $A(\,)$.

Based on the above, we obtain the Algorithm Not-in-HHB, presented in Figure 3, which takes as input a graph G and a vertex v of G, and returns "true" if and only if the vertex v does not belong to a hole, and it is not the top of a house or a building in G. The correctness of Algorithm Not-in-HHB is established in the following theorem.

Theorem 3.1 *Algorithm Not-in-HHB(G, v) returns "false" if and only if the vertex v belongs to a hole or is the top of a house or a building in G.*

Proof: ($\Longrightarrow$) Suppose that the algorithm returns "false" while processing vertex $z \in M(v)$ (i.e, when $i = \sigma_v^{-1}(z)$). This happens only if at Step 8 of the current iteration of the for-loop there exists a vertex

Algorithm Not-in-HHB(G, v)

---

1. Compute a DegMN-ordering $\sigma_v = (y_1, y_2, \ldots, y_k)$ of the non-neighbors of v in the graph G;
   compute the array $Next\_Neighbor_{G'[M(v)],\sigma_v}[\,]$ for the chordal completion $G'[M(v)]$ of the subgraph $G[M(v)]$ with respect to the ordering σ_v;
   for each non-neighbor u of v **do**
       $A(u) \leftarrow \emptyset$;
2. **for** $i \leftarrow 1$ to k **do**
3. $u \leftarrow \sigma_v(i)$;
4. $X \leftarrow N(u) \cap N(v)$;   {*note:* $\forall x \in X$, *u precedes x in the ordering* σ_v}
5.   **if** $X \neq \emptyset$ **and** the entry $Next\_Neighbor_{G'[M(v)],\sigma_v}[u]$ has been assigned a value
6.   **then** $w \leftarrow Next\_Neighbor_{G'[M(v)],\sigma_v}[u]$;   {*note:* $w \in M(v)$}
7.   concatenate X to $A(w)$;   {*note:* $w \notin X$}
8.   **if** $A(u) - N(u) \neq \emptyset$ **then** **return**("false");
9. **return**("true");

Fig. 3: The algorithm for determining whether a vertex v belongs to a hole or is the top of a house or a building.

$x \in A(z) - N(z)$. Then, from Lemma 3.2, we have that $A(z) - N(z) \subseteq N(v)$, which implies that $x \in N(v)$ and $xz \notin E(G)$. The vertex x was added to $A(z)$ at Step 7 of a prior iteration, say, while processing vertex y; thus, $xy \in E(G)$ and $z = Next\_Neighbor_{G'[M(v)],\sigma_v}[y]$ which implies that $\sigma_v^{-1}(y) < \sigma_v^{-1}(z)$ (and thus $y \in M(v)$), and $yz \in E(G'[M(v)])$. As in the proof of Lemma 3.2, we can show that there exists a vertex x' of G such that $x' \in N(v)$, $x'z \in E(G)$, and $x'y \notin E(G)$ (see Figure 4(a), where the dotted edge indicates a potential edge of G). For the vertices y, z (which are adjacent in $G'[M(v)]$), we distinguish two cases:

(i) $yz \in E(G)$. Then, if $xx' \notin E(G)$, the vertex set $\{v, x, y, z, x'\}$ induces a hole (in fact, a C_5), otherwise it induces a house with vertex v at the top.

(ii) $yz \notin E(G)$. Then, because $z = Next\_Neighbor_{G'[M(v)],\sigma_v}[y]$ (i.e., the vertices y, z are adjacent in the chordal completion $G'[M(v)]$), Lemma 2 of [9] implies that the graph G has an induced path on at least three vertices connecting y and z all of whose vertices are in $M(v)$ and such that for each vertex w of the path other than y, z, it holds that $N(w) \cap N(v) \subseteq N(y) \cap N(v)$. Let $a_1 a_2 \cdots a_q$ be a chordless such path where $a_1 = y$, $a_q = z$, and $q \geq 3$; then, $a_i \in M(v)$ and $N(a_i) \cap N(v) \subseteq N(y) \cap N(v)$, for all $1 < i < q$ (see Figure 4(b), where the dashed edge indicates an edge in $E(G'[M(v)]) - E(G)$). Since $x' \notin N(y)$, this implies that $x' \notin N(a_i)$ for all $i = 1, 2, \ldots, q-1$. Let $p = \max\{i \mid x \in N(a_i)\}$; p is well defined since $x \in N(a_1)$, and $p < q$ since $x \notin N(a_q)$ —recall that $a_1 = y$, $a_q = z$, $xy \in E(G)$, and $xz \notin E(G)$. Then, the vertices $a_{p+1}, \ldots, a_{q-1}$ are not adjacent to x or x', so that if $xx' \notin E(G)$, the vertices $v, x, a_p, \ldots, a_{q-1}, z, x'$ induce a hole in G, whereas if $xx' \in E(G)$ they induce a house or a building with v at its top.

In both cases, the vertex v belongs to a hole or is the top of a house or a building in the graph G.

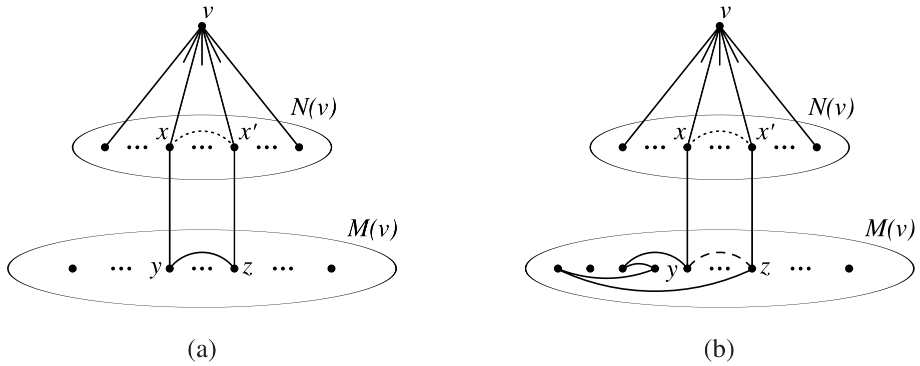

Fig. 4

($\Longleftarrow$) Suppose that the vertex v belongs to a hole or is the top of a house or a building in G. If v belongs to a C_5 or is the top of a house, let v, x, y, z, x' be the vertices inducing the C_5 or house ($x, x' \in N(v)$, $y, z \in M(v)$, $xy \in E(G)$, and $x'z \in E(G)$); if v belongs to a hole on more than 5 vertices or is the top of a building, we consider such a hole or building on the fewest vertices, and let them be $v, x, y, a_1, \ldots, a_p, z, x'$ ($x, x' \in N(v)$, $y, a_1, \ldots, a_p, z \in M(v)$, $xy \in E(G)$, and $x'z \in E(G)$). Moreover, suppose without loss of generality that in either case $\sigma_v^{-1}(y) < \sigma_v^{-1}(z)$. In case v belongs to a hole on ≥ 6 vertices or is the top of a building, we can show that all the a_is precede both y and z in the ordering σ_v: if any a_i appears between y and z, then the definition of the DegMN-ordering σ_v implies that there exists vertex $x'' \in N(v)$ such that $x'' \in N(a_i) - N(y)$ (recall that $xy \in E(G)$ and $xa_i \notin E(G)$); then the vertices $v, x, y, a_1, \ldots, a_i, x''$ induce a hole or a building on fewer vertices. Then, this property of the a_is and Lemma 3.3 imply that $yz \in E(G'[M(v)])$.

From the above, we conclude that, in any case, the vertices y, z are adjacent in $G'[M(v)]$; then, we show that there exists a sequence $y{=}b_1, b_2, \ldots, b_q{=}z$ of non-neighbors of v in G such that $b_i = Next\_Neighbor_{G'[M(v)],\sigma_v}[b_{i-1}]$ for $2 \leq i \leq q$ (note that $b_i = Next\_Neighbor_{G'[M(v)],\sigma_v}[b_{i-1}]$ implies that $\sigma_v^{-1}(b_{i-1}) < \sigma_v^{-1}(b_i)$ and $b_{i-1}b_i \in E(G'[M(v)])$). The existence of such a sequence is shown as follows: if $z = Next\_Neighbor_{G'[M(v)],\sigma_v}[y]$, then the sequence is precisely y, z; otherwise, if $b_2 = Next\_Neighbor_{G'[M(v)],\sigma_v}[y]$, then $yb_2 \in E(G'[M(v)])$ which from the fact that $yz \in E(G'[M(v)])$ and from the definition of $G'[M(v)]$ implies that $b_2z \in E(G'[M(v)])$; next, we repeat the above argument for b_2 in the place of y, and so on so forth; since there is a finite number of vertices between y and z in σ_v, eventually, we will find a vertex b_{q-1} such that $z = Next\_Neighbor_{G'[M(v)],\sigma_v}[b_{q-1}]$. Let $r = \max\{i \mid x \in N(b_i)\}$; r is well defined since $x \in N(b_1)$, and $r < q$ since $x \notin N(b_q)$ — recall that $b_1 = y$ and $xy \in E(G)$, and $b_q = z$ and $xz \notin E(G)$. Since $x \in N(b_r)$ and $b_{r+1} = Next\_Neighbor_{G'[M(v)],\sigma_v}[b_r]$, the processing of vertex b_r will result in the addition of vertex x in the set $A(b_{r+1})$; then, the Algorithm Not-in-HHB will return "false" when it processes vertex b_{r+1}, because $x \notin N(b_{r+1})$, if not earlier. $\Box$

In light of Theorem 3.1 and by observing that a building contains a hole, we obtain the very simple HH-free graph recognition algorithm that is given in Figure 5.

Algorithm Recognize-HH-free

---

1. **for** each vertex v of the input graph G **do**
   if Algorithm Not-in-HHB(G, v) returns "false"
   then return("the graph is not HH-free"); $\{G$ *contains a house or a hole*$\}$

2. **return**("the graph is HH-free");

Fig. 5: The algorithm for determining whether a graph G is HH-free.

3.1 Computation of the values of Next_Neighbor$_{G'[S],\sigma}$[ ].

Let G be a graph, S a subset of the vertex set of G, σ an ordering of the vertices in S, and let $G'[S]$ be the chordal completion of the subgraph $G[S]$ with respect to σ. For each vertex $x \in S$, we want to determine the leftmost among the neighbors of x that are to the right of x (w.r.t. σ) in $G'[S]$. In order to avoid computing the graph $G'[S]$, we take advantage of the following property of the chordal completion:

Lemma 3.3 *Let G be a graph, let $(v_1, v_2, \ldots, v_k)$ be an ordering of a vertex set $S \subseteq V(G)$, and let G' be the graph resulting from the subgraph $G[S]$ after we had applied chordal completion on it with respect to the ordering of the vertices in S (i.e., after having added edges so that, for all $i = 1, 2, \ldots, k$, vertex v_i is simplicial in the subgraph induced by the vertices $v_i, v_{i+1}, \ldots, v_k$). Then, the graph G' contains the edge $v_r v_j$, where $r < j$, if and only if there exists an edge $v_i v_j$ in G such that $i \leq r$ and the vertices v_i, v_r belong to the same connected component of the subgraph of G induced by the vertices $v_1, v_2, \ldots, v_i, \ldots, v_r$.*

Proof: ($\Longrightarrow$) Suppose that $v_r v_j \in E(G')$. We will show the following:

> *Proposition.* If $v_r v_j \in E(G')$, then there exists a vertex v_i, where $i \leq r$, such that $v_i v_j$ is an edge of G and the vertices v_i, v_r belong to the same connected component of the subgraph $G[\{v_1, v_2, \ldots, v_i, \ldots, v_r\}]$.

We use induction on r.

Basis: $r = 1$. Then, since $v_1 v_j \in E(G')$, it can only be that $v_1 v_j \in E(G)$ and the proposition holds with v_i being v_1.

Inductive hypothesis: Suppose that the proposition holds for all $r < \hat{r}$, where $\hat{r} \geq 2$.

Inductive step: We show that the proposition holds for $r = \hat{r}$. Hence, suppose that $v_{\hat{r}} v_j \in E(G')$. If $v_{\hat{r}} v_j$ is an edge of G, then the proposition clearly holds with v_i being $v_{\hat{r}}$. Otherwise, the edge $v_{\hat{r}} v_j$ must have been added while making a vertex v_t simplicial; then, $t < \hat{r} < j$ and $v_t v_{\hat{r}}, v_t v_j \in E(G')$. Since $v_t v_{\hat{r}} \in E(G')$ and $t < \hat{r}$, then by the inductive hypothesis, there exists a vertex v_p, where $p \leq t < \hat{r}$, such that $v_p v_{\hat{r}}$ is an edge of G and v_p, v_t belong to the same connected component of $G[\{v_1, v_2, \ldots, v_t\}]$. Similarly, by the inductive hypothesis for the edge $v_t v_j$ of G', there exists a vertex v_q, where $q \leq t < j$, such that $v_q v_j$ is an edge of G and v_q, v_t belong to the same connected component of $G[\{v_1, v_2, \ldots, v_t\}]$. Therefore, the vertices v_p, v_q belong to the same connected component of $G[\{v_1, v_2, \ldots, v_t\}]$. Then, the existence of the edge $v_p v_{\hat{r}}$ in G and the fact that $p \leq t < \hat{r}$ imply that $v_p, v_{\hat{r}}$ belong to the same

Algorithm Compute-Next_Neighbor(G, S, σ)

```
{let σ = (v_1, v_2, ..., v_k) be the given ordering of the vertices in S}
make a set containing the vertex v_1;
for j = 2, 3, ..., k do
      make a set containing the vertex v_j;
      for each edge v_i v_j of G, where i < j, do
            v_r ← the rightmost (w.r.t. σ) vertex in the set to which v_i belongs;
            if v_r ≠ v_j
            then      {v_i and v_j belong to different sets}
                  Next_Neighbor_{G'[S],σ}[v_r] ← v_j;
                  union the sets to which v_i and v_j belong;
```

Fig. 6: The algorithm for computing the contents of the array $Next\_Neighbor_{G'[S],\sigma}[\,]$.

connected component of $G[\{v_1, v_2, \ldots, v_{\hat{r}}\}]$; moreover, since $v_p v_j$ is an edge of G and $p \leq t < \hat{r} < j$, the proposition holds with v_i being v_p.

The induction then implies that the proposition holds for all edges $v_r v_j \in E(G')$, where $r < j$.

($\Longleftarrow$) Suppose now that for vertices v_r, v_j of G, where $r < j$, there exists a vertex v_i, where $i \leq r$, such that $v_i v_j$ is an edge of G and the vertices v_i, v_r belong to the same connected component of the subgraph $G[\{v_1, v_2, \ldots, v_r\}]$. We will show that $v_r v_j \in E(G')$.
If $v_r v_j \in E(G)$, then clearly $v_r v_j \in E(G')$. Suppose now that $v_r v_j \notin E(G)$. Since the vertices v_i, v_r belong to the same connected component of $G[\{v_1, v_2, \ldots, v_r\}]$, there exists a simple path $v_r v_{p_1} v_{p_2} \cdots v_{p_\ell} v_i$ from v_r to v_i in $G[\{v_1, v_2, \ldots, v_r\}]$; then, clearly, $p_t < r$ for all $t = 1, 2, \ldots, \ell$. Since $v_i v_j$ is an edge of G, then $v_r v_{p_1} v_{p_2} \cdots v_{p_\ell} v_i v_j$ is a simple path from v_r to v_j in G, and hence in G'. For ease of notation, let us set $v_{p_0} = v_r$, $v_{p_{\ell+1}} = v_i$, and $v_{p_{\ell+2}} = v_j$, so that the path from v_r to v_j becomes $v_{p_0} v_{p_1} v_{p_2} \cdots v_{p_{\ell+1}} v_{p_{\ell+2}}$. Let p_s be the minimum among $p_1, p_2, \ldots p_{\ell+1}$, i.e., v_{p_s} is the leftmost (with respect to σ) vertex among $v_{p_1}, v_{p_2}, \ldots v_{p_{\ell+1}}$; then, since for all $t = 1, 2, \ldots, \ell+1$ it holds that $p_t < r < j$, we have that $p_s < p_{s-1}$ and $p_s < p_{s+1}$. The definition of the graph G' implies that v_{p_s} is simplicial in $G'[\{v_{p_s}, v_{p_s+1}, \ldots, v_k\}]$; since v_{p_s} is adjacent in G' to $v_{p_{s-1}}$ and to $v_{p_{s+1}}$, G' contains the edge $v_{p_{s-1}} v_{p_{s+1}}$. Thus, $v_{p_0} v_{p_1} \cdots v_{p_{s-1}}$ $v_{p_{s+1}} \cdots v_{p_{\ell+2}}$ is a simple path in G' from v_r to v_j, i.e., we obtained a path in G' from v_r to v_j where the leftmost vertex v_{p_s} has been removed. The process of removing the leftmost (w.r.t. σ) vertex among the vertices of the path can be repeated over and over, and in each case we obtain a shorter simple path in G' from v_r to v_j. Since $p_t < r < j$ for all $t = 1, 2, \ldots, \ell+1$, eventually all the vertices $v_{p_1}, v_{p_2}, \ldots, v_{p_{\ell+1}}$ will be removed and we will have that $v_{p_0} v_{p_{\ell+2}} = v_r v_j$ is a path (= edge) in G', as desired. □

We note that the above lemma implies Lemma 2 of [9] as a corollary. Lemma 3.3 implies that for the computation of the value $Next\_Neighbor_{G'[S],\sigma}[v_r]$, where $\sigma = (v_1, v_2, \ldots, v_k)$, it suffices to find the leftmost (w.r.t. σ) vertex among $v_{r+1}, v_{r+2}, \ldots, v_k$ which is adjacent in G to a vertex in the connected component of $G[\{v_1, v_2, \ldots, v_r\}]$ to which v_r belongs. This can be efficiently done by processing the vertices $v_1, v_2, \ldots, v_k$ in order from v_1 to v_k. In detail, the algorithm to compute the contents of the array $Next\_Neighbor_{G'[S],\sigma}[\,]$ is presented in Figure 6; note that if a vertex $v_i \in S$ has no neighbors in $G'[\{v_i, \ldots, v_k\}]$, then the entry $Next\_Neighbor_{G'[S],\sigma}[v_i]$ is not assigned a value.

It is important to observe the following:

Observation 3.1 *At the completion of the processing of vertex v_j in Algorithm Compute-Next_Neighbor, the sets of vertices maintained by the algorithm are in a bijection with the connected components of $G[\{v_1, v_2, \ldots, v_j\}]$;*

The observation follows from the fact that while processing v_j, we consider the edges $v_i v_j$ where $i < j$, and we union the set containing v_j (which has vertex v_j as its rightmost vertex with respect to σ) to another set iff v_j is adjacent to a vertex in that set. The correctness of Algorithm Compute-Next_Neighbor is established in the following lemma.

Lemma 3.4 *The Algorithm Compute-Next_Neighbor correctly computes the values of the entries Next_Neighbor$_{G'[S],\sigma}[v_i]$ for all the vertices $v_i \in S$.*

Proof: First, we show that, for a vertex $v_r \in S$, the Algorithm Compute-Next_Neighbor assigns a value to the entry $Next\_Neighbor_{G'[S],\sigma}[v_r]$ if and only if the vertex v_r is adjacent in the graph $G'[S]$ to a vertex in $\{v_{r+1}, v_{r+2}, \ldots, v_k\}$.

The entry $Next\_Neighbor_{G'[S],\sigma}[v_r]$ is assigned a value if, during the processing of a vertex v_j, where $j > r$, there exists an edge in G connecting v_j to a vertex v_i, and v_r is the rightmost (w.r.t. σ) vertex among the vertices in the set containing v_i (Step 5 of Algorithm Compute-Next_Neighbor). Since the vertices v_i and v_r belong to the same set, which corresponds to the same connected component of the subgraph $G[\{v_1, v_2, \ldots, v_{j-1}\}]$ (Observation 3.1), and since v_r is the rightmost vertex in the set, then v_i, v_r belong to the same connected component of $G[\{v_1, v_2, \ldots, v_r\}]$. Due to this and due to the fact that $v_r v_j \in E(G)$, where $r < j$, Lemma 3.3 implies that the chordal completion $G'[S]$ of the subgraph $G[S]$ of G induced by the elements of S contains the edge $v_r v_j$.

Suppose now that a vertex v_i is adjacent in $G'[S]$ to a vertex in $\{v_{i+1}, v_{i+2}, \ldots, v_k\}$; we show that the Algorithm Compute-Next_Neighbor assigns a value to the entry $Next\_Neighbor_{G'[S],\sigma}[v_i]$. Let v_s be the leftmost (w.r.t. σ) vertex among the vertices in $\{v_{i+1}, v_{i+2}, \ldots, v_k\}$ which are adjacent in G to a vertex in the connected component of $G[\{v_1, v_2, \ldots, v_i\}]$ to which v_i belongs. We show that right before processing v_s, the rightmost (w.r.t. σ) vertex in the set to which v_i belongs is precisely v_i. Suppose for contradiction that this is not the case and let $v_{s'}$ be the rightmost vertex, where $s' > i$. Then, there exists a simple path in G from v_i to $v_{s'}$; let it be $v_{p_0} v_{p_1} v_{p_2} \cdots v_{p_h}$, where $v_{p_0} = v_i$ and $v_{p_h} = v_{s'}$. Clearly, $p_t < s'$ for all $t = 1, 2, \ldots, h-1$. Let $\hat{t} = \min\{\, t \in \{1, 2, \ldots, h\} \mid p_t > i \,\}$; $\hat{t}$ is well defined since $p_h = s' > i$. Then, $v_{p_{\hat{t}}} v_{p_{\hat{t}-1}} \in E(G)$ and since $v_{p_0} v_{p_1} \cdots v_{p_{\hat{t}-1}}$ is a path in G and $p_t \leq i$ for all $t = 0, 1, \ldots, \hat{t}-1$, the vertices $v_{p_{\hat{t}-1}}$ and v_i belong to the same connected component of $G[\{v_1, v_2, \ldots, v_i\}]$; however, these facts come to a contradiction with the definition of v_s, because $i < p_{\hat{t}} < s$. Therefore, right before processing vertex v_s, v_i indeed is the rightmost (w.r.t. σ) vertex in the set to which it belongs, and thus the entry $Next\_Neighbor_{G'[S],\sigma}[v_i]$ will be assigned a value (it will be set equal to v_s during the processing of vertex v_s).

The proof will be complete if we show that, whenever the Algorithm Compute-Next_Neighbor assigns a value to an entry $Next\_Neighbor_{G'[S],\sigma}[v_r]$, it assigns the correct value. Observe that whenever the algorithm executes an assignment $Next\_Neighbor_{G'[S],\sigma}[v_r] \leftarrow v_j$, then $r < j$ and there exists an edge $v_i v_j$ in G such that $i < j$ and the vertices v_i, v_j belong to the same connected component of the subgraph $G[\{v_1, v_2, \ldots, v_r\}]$. Then, Lemma 3.3 implies that the chordal completion $G'[S]$ of the subgraph $G[S]$ of G induced by the elements of S contains the edge $v_r v_j$. We need to show that v_r

is not adjacent in the chordal completion of $G[S]$ to any vertex v_t, where $r < t < j$. Suppose for contradiction that there existed such a vertex v_t. Then, by Lemma 3.3, there exists a vertex v_s such that $s < r$, $v_s v_t \in E(G)$, and v_s, v_r belong to the same connected component of $G[\{v_1, v_2, \ldots, v_r\}]$. But then, while processing the vertex v_t, v_t will be included in the set containing v_r, in contradiction to the fact that, while processing v_j (which is done after processing v_t since $t < j$), the rightmost (w.r.t. σ) vertex in the set containing v_r is precisely v_r; recall that $r < t$. Therefore, all the assignments to the entries of the array $Next\_Neighbor_{G'[S],\sigma}[\,]$ are correct. □

Time and Space Complexity. Let m be the number of edges of the graph G; recall also that $|S| = k$. If we ignore the operations to process sets (i.e., make a set, union sets, or find the rightmost (w.r.t. σ) vertex in a set) while running Algorithm Compute-Next_Neighbor(G, S, σ), then the rest of the execution of the algorithm takes $O(k + m)$ time. The sets are maintained by our algorithm in a fashion amenable for Union-Find operations, where additionally the representative of each set also carries a link to the node storing the rightmost (w.r.t. σ) vertex in the set. Then,

- making a set which contains a single vertex v_i requires building the set and setting the rightmost (w.r.t. σ) vertex in the set to v_i;
- finding the rightmost (w.r.t. σ) vertex in the set, say, A, to which a vertex v_j belongs, requires performing a Find operation to locate the representative of the set A, from which the rightmost vertex is obtained in constant time;
- unioning two sets requires constructing a single set out of the elements of the two sets, and updating the rightmost (w.r.t. σ) vertex information; since we always union a set with the set containing v_j, where v_j is the rightmost vertex in any of the sets, then the rightmost vertex of the resulting set is v_j, and this assignment can be done in constant time per union.

As the Algorithm Compute-Next_Neighbor creates one set for each one of the vertices $v_1, v_2, \ldots, v_k$, it executes k make-set operations; this also implies that the number of union operations is less than k. The number of times to find the rightmost (w.r.t. σ) vertex in a set is $O(m)$, since the algorithm executes one such operation for each edge of the subgraph $G[S]$. The sets may be represented as disjoint-set forests, in which case, the time to execute p make-set, find, and union operations involving q elements is $O(p\,\alpha(p, q))$ [17], where $\alpha(\,,\,)$ denotes the very slowly growing functional inverse of Ackerman's function; since Algorithm Compute-Next_Neighbor performs p operations, where $k \leq p = O(k + m)$, on k elements, and the function $\alpha(p, q)$ decreases as the ratio p/q increases, the use of disjoint-set forests implies a time complexity of $O(m\,\alpha(k, k))$. Alternatively, the sets may be represented by linked lists, and the time to execute p make-set, find, and union operations involving q elements is $O(p + q \log q)$ [3]; thus, in this case, the time complexity is $O(m + k \log k)$. In either case, the space required (in addition to the space needed to store the graph G) is $O(k)$. Thus, the computation of the values of the array $Next\_Neighbor_{G'[S],\sigma}[\,]$ for the k elements of the set S takes a total of $O(\min\{(k + m)\,\alpha(k, k),\ m + k \log k\})$ time and $O(k)$ space. Therefore, we have:

Lemma 3.5 *Let G be a graph on m edges, S a subset of vertices of G, and σ an ordering of the elements of S. Algorithm Compute-Next_Neighbor(G, S, σ) correctly computes the values of the entries of the array $Next\_Neighbor_{G'[S],\sigma}[\,]$ in $O(\min\{(k + m)\,\alpha(k, k),\ m + k \log k\})$ time and $O(k)$ space, where k is the cardinality of the set S.*

3.2 Complexity of the Algorithm Recognize-HH-free

Let us assume that the graph G has n vertices and m edges and that vertex v of G has k non-neighbors in G. It is not difficult to see that the execution of the Algorithm Not-in-HHB(G, v) for vertex v takes $O(n+m)$ time and space plus the time and space needed for the computation of the entries of the array $Next\_Neighbor_{G'[M(v)],\sigma_v}[\,]$. For the latter computation, we run Algorithm Compute-Next_Neighbor on $G, M(v)$, and σ_v, which takes $O(\min\{(k+m)\,\alpha(k,k),\ m+k\log k\}) = O(\min\{(n+m)\,\alpha(n,n),\ m+n\log n\})$ time and $O(k) = O(n)$ space (Lemma 3.5); note that the function $\alpha(i,i)$ increases as i increases (see definition in [17]). Therefore, Algorithm Not-in-HHB(G, v) takes $O(n+m+\min\{(n+m)\,\alpha(n,n),\ m+n\log n\}) = O(\min\{(n+m)\,\alpha(n,n),\ m+n\log n\})$ and $O(n+m)$ space. Hence, we have:

Theorem 3.2 *Let G be a graph on n vertices and m edges. Algorithm Not-in-HHB determines whether a vertex v of G belongs to a hole or is the top of a house or a building in $O(\min\{(n+m)\,\alpha(n,n),$ $m+n\log n\})$ time and $O(n+m)$ space.*

The Algorithm Recognize-HH-free consists of applying the Algorithm Non-in-HHB on every vertex of the input graph G. If G is connected, then $n = O(m)$ and processing all the vertices of G takes $O(\sum_v \min\{m\,\alpha(n,n),\ m+n\log n\})$ time and $O(n+m)$ space. If G is not connected, then we compute its connected components [3] and work on each of them separately (note that a hole and a house are both connected); clearly, the subgraph of G induced by each such component is connected and has $O(n)$ vertices and $O(m)$ edges. Then, the computation of the components and the processing of all the vertices of G takes $O(n+m+\sum_v \min\{m\,\alpha(n,n),\ m+n\log n\})$ total time and $O(n+m)$ space. Summarizing, we obtain the following corollary:

Corollary 3.1 *Algorithm Recognize-HH-free determines whether a graph G on n vertices and m edges contains a hole or a house (i.e., is not HH-free) in $O(n\ \min\{m\,\alpha(n,n),\ m+n\log n\})$ time and $O(n+m)$ space.*

3.3 Providing a Certificate

The Algorithm Recognize-HH-free can be made to provide a certificate (a hole or a house) whenever it decides that the input graph G is not HH-free. In particular, we augment the Algorithm Not-in-HHB(G, v) as follows:

- When processing vertex u, we add a reference to u to each element of the set X formed in Step 4, so that for each vertex w, each element of the set $A(w)$ carries a reference to the vertex during whose processing this element was added to $A(w)$ (see Step 7).

Our approach follows the proof of Theorem 3.1. The Algorithm Recognize-HH-free answers that the graph G is not HH-free when a call to Algorithm Not-in-HHB(G, v) for a vertex v returns "false," i.e., when in Step 8 the difference $A(u)-N(u)$ is non-empty for a non-neighbor u of v in G. Let $x \in N(v)$ be an element in $A(u)-N(u)$; the vertex x is associated with a vertex $y \in M(v)$ during whose processing vertex x was added to $A(u)$ (then, clearly, $u = Next\_Neighbor_{G'[M(v)],\sigma_v}[y]$).[(ii)]. Then, we can obtain a hole or a house of G by doing the following:

[(ii)] It must be noted that vertex x may have been added in $A(u)$ more than once by different vertices; yet, x and any of these vertices suffice for our purposes.

1. We traverse the neighbors of vertex u and find a vertex x' which is adjacent to v and not adjacent to y; such a vertex always exists since $n(v, y) \leq n(v, u)$ (note that y precedes u in the DegMN-ordering σ_v) and $x \in N(y) \cap N(v)$ whereas $x \notin N(u) \cap N(v)$.

2. We consider the subgraph of G induced by u, y and the vertices preceding y in σ_v, and we apply BFS on it starting at y until u is reached (note that u will be eventually reached, because $u = Next\_Neighbor_{G'[M(v)],\sigma_v}[y]$, and thus y and u belong to the same connected component of the subgraph of G induced by u, y and the vertices preceding u in σ_v; see Lemma 3.3); let $a_1 a_2 \cdots a_q$, where $a_1 = y$ and $a_q = u$, be the path in the BFS-tree from y to u, which is thus chordless. Moreover, $\{a_1, a_2, \ldots, a_{q-1}\} \cap N(x') = \emptyset$ (otherwise, the algorithm would have exited while processing a vertex preceding u; see proof of Theorem 3.1).

3. We compute $p = \max\{\, i \mid x \in N(a_i)\,\}$. Then, if $xx' \notin E(G)$, the vertices $v, x, a_p, \ldots, a_{q-1}, u, x'$ induce a hole in G; if $xx' \in E(G)$, the vertices v, x, a_p, u, x' induce a house in G if $p = q - 1$, whereas if $p \leq q - 2$, the vertices $x, a_p, \ldots, a_{q-1}, u, x'$ induce a hole.

The correctness of the computation follows from the proof of Theorem 3.1. Regarding the time and space complexity, we first note that the augmentation of the Algorithm Not-in-HHB does not asymptotically increase the time and space complexity of the Algorithm Recognize-HH-free. In turn, if we assume that we have an adjacency-list representation of the input graph G and that we use two auxiliary arrays (of size n) to mark the neighbors of v and of y, it is not difficult to see that all three steps of the certificate computation take $O(n + m)$ time and $O(n)$ space, where n, m are the numbers of vertices and edges of G, respectively. Therefore, we have:

Theorem 3.3 *The Algorithm Recognize-HH-free can be easily augmented to provide a certificate whenever it decides that the input graph G is not HH-free; if G has n vertices and m edges, the certificate computation takes $O(n + m)$ time and $O(n)$ space.*

4 Recognition of HHD-free and WPO-Graphs

In this section, we present two applications of the ideas and of the algorithm for recognizing HH-free graphs. In particular, we show how to use the HH-free graph recognition algorithm in order to recognize HHD-free graphs within the same time and space complexity, and how the ideas can be used to determine if the complement of a given graph is HH-free so that we can recognize WPO-graphs.

4.1 HHD-free Graphs

Our HHD-free graph recognition algorithm is motivated by the corresponding algorithm of Hoàng and Sritharan [9], which in turn is motivated by the work of Hoàng and Khouzam [8] and relies on the following characterization of HHD-free graphs proved by Jamison and Olariu:

Theorem 4.1 (Jamison and Olariu [10]) *The following two statements are equivalent:*

(i) The graph G is HHD-free;

(ii) For every induced subgraph H of the graph G, every ordering of vertices of H produced by LexBFS is a semi-perfect elimination.

Algorithm Recognize-HHD-free

---

1. **if** the input graph G is not HH-free
   then return("the graph is not HHD-free");

2. Run LexBFS on G starting at an arbitrary vertex w, and let $(v_1, v_2, \ldots, v_n)$ be the resulting ordering, where $v_n = w$.

3. **for** $i = 1, 2, \ldots, n-5$ **do**
   if v_i is not semi-simplicial in $G[\{v_i, v_{i+1}, \ldots, v_n\}]$
   then return("the graph is not HHD-free");

4. **return**("the graph is HHD-free").

Fig. 7: The algorithm for determining whether a graph G is HHD-free.

In fact, we could use the Algorithm Not-in-HHB(G, v) in Hoàng and Sritharan's HHD-free graph recognition algorithm in order to determine if vertex v is high, and we would achieve the improved time and space complexities stated in this paper. However, we can get the much simpler algorithm which we give in Figure 7.

Note that, after Step 1, we need only check whether the input graph G contains a domino; this is why, we only process the $n-5$ vertices $v_1, v_2, \ldots, v_{n-5}$ in Step 3. Additionally, it is important to observe that, for all $i = 1, 2, \ldots, n$, the ordering $(v_i, v_{i+1}, \ldots, v_n)$ is an ordering which can be produced by running LexBFS on the subgraph $G[\{v_i, v_{i+1}, \ldots, v_n\}]$ starting at vertex v_n. The correctness of the algorithm follows from Theorem 4.1 and the fact that if the currently processed vertex v_i in Step 3 is semi-simplicial then clearly it cannot participate in a domino (note that none of the vertices of a domino is semi-simplicial in any graph containing the domino as induced subgraph).

Time and Space Complexity. Let n and m be the number of vertices and edges of the input graph G. According to Corollary 3.1, Step 1 takes $O(n \min\{m\,\alpha(n,n),\ m + n\log n\})$ time and $O(n+m)$ space. Step 2 takes $O(n+m)$ time and space [5; 16]. The construction of the subgraphs $G[\{v_i, v_{i+1}, \ldots, v_n\}]$ in Step 3 can be done in a systematic fashion by observing that $G[\{v_1, v_2, \ldots, v_n\}] = G$ and that $G[\{v_{i+1}, v_{i+2}, \ldots, v_n\}]$ can be obtained from $G[\{v_i, v_{i+1}, \ldots, v_n\}]$ by removing vertex v_i and all its incident edges; if the graph G is stored using a (doubly-connected) adjacency-list representation with pointers for every edge ab connecting the record storing b in the adjacency list of a to the record storing a in the adjacency list of b and back, then obtaining $G[\{v_{i+1}, v_{i+2}, \ldots, v_n\}]$ from $G[\{v_i, v_{i+1}, \ldots, v_n\}]$ takes time proportional to the degree of v_i in $G[\{v_i, v_{i+1}, \ldots, v_n\}]$ and hence $O(deg(v_i))$ time, where $deg(v_i)$ denotes the degree of vertex v_i in G. Additionally, in order to check whether a vertex is semi-simplicial, we take advantage of the following result of Hoàng and Khouzam (which was also used in [9]):

Theorem 4.2 (Hoàng and Khouzam [8]) *Let G be a graph and x be a semi-simplicial vertex of G. If x is not simplicial, then each big co-component of the subgraph $G[N(x)]$ is a module of G.*

(A connected component or co-component of a graph is called *big* if it has at least two vertices; we also note that if a vertex x is simplicial then none of the co-components of the subgraph $G[N(x)]$ is big.) Since computing the subgraph induced by the neighbors of vertex v_i in $G[\{v_i, v_{i+1}, \ldots, v_n\}]$, computing its co-components, and testing whether a vertex set is a module in $G[\{v_i, v_{i+1}, \ldots, v_n\}]$ can all be done in time and space linear in the size of $G[\{v_i, v_{i+1}, \ldots, v_n\}]$, Step 3 takes a total of $O\Big(\sum_i \Big(n+m+deg(v_i)\Big)\Big) = O(n\,m)$ time and $O(n+m)$ space. Finally, Step 4 takes constant time. Therefore, we obtain the following theorem.

Theorem 4.3 *Let G be an undirected graph on n vertices and m edges. Then, Algorithm Recognize-HHD-free determines whether G is an HHD-free graph in $O(n\,\min\{m\,\alpha(n,n),\ m+n\log n\})$ time and $O(n+m)$ space.*

4.2 Welsh-Powell Opposition Graphs

Our algorithm for recognizing WPO-graphs relies on the fact that a graph G is a WPO-graph if and only if G is HHP-free and its complement $\overline{G}$ is HH-free, which follows from the following characterization due to Olariu and Randall [15]:

Theorem 4.4 (Olariu and Randall [15]) *A graph G is a WPO-graph if and only if G contains no induced C_5, P_5, house, or "P".*

Eschen *et al.* [4] described an $O(n\,m)$-time algorithm for recognizing whether a graph G on n vertices and m edges is HHP-free by using the modular decomposition tree of G and Theorem 4.2 due to Hoàng and Khouzam [8]. We next show that we can detect whether the complement $\overline{G}$ of G contains a hole or a house in $O(n\,m)$ time. Combining these two algorithms, we get an $O(n\,m)$-time algorithm for recognizing WPO-graphs.

The obvious approach of running Algorithm Recognize-HH-free on the complement $\overline{G}$ of G, in order to check whether $\overline{G}$ contains a hole or a house, proves to be expensive. Thus, instead, we process the vertices of G in turn; for each vertex v, we consider a (different) auxiliary graph $\widehat{G}_v$ as follows:

- $V(\widehat{G}_v) \;=\; V(G)$
- $E(\widehat{G}_v) \;=\; \{\, vy \mid y \in M(v)\,\}$
  $\cup\ \{\, xy \mid x \in N(v),\ y \in M(v),$ and $xy \notin E(G)\,\}$
  $\cup\ \{\, xx' \mid x, x' \in N(v)$ and $xx' \notin E(G)\,\}$

Note that in $\overline{G}$ the neighbors of v are the non-neighbors $M(v)$ of v in G, and its non-neighbors are the neighbors $N(v)$ of v in G. Thus, the graph $\widehat{G}_v$ is precisely $\overline{G}$ without any edges between neighbors of v. This implies that:

(i) v cannot be the top of a house or a building in $\widehat{G}_v$, and

(ii) if v is the top of a house or a building in $\overline{G}$, then it belongs to a hole in $\widehat{G}_v$.

Then, in order to check whether vertex v belongs to a hole or is the top of a house or a building in $\overline{G}$, it suffices to execute Algorithm Not-in-HHB($\widehat{G}_v, v$), and answer "true" if and only if Algorithm Not-in-HHB($\widehat{G}_v, v$) returns "false." Thus, we have the following result.

Lemma 4.1 *The vertex v belongs to a hole or is the top of a house or a building in $\overline{G}$ if and only if Algorithm Not-in-HHB($\widehat{G}_v, v$) returns "false."*

Lemma 4.1 and the complexity of Algorithm Not-in-HHB implies the following theorem.

Theorem 4.5 *Let G be an undirected graph on n vertices and m edges. Then, the algorithm that we described in this section determines whether the complement $\overline{G}$ is an HH-free graph in $O(n\,m)$ time and $O(n^2)$ space.*

Proof: It suffices to establish the theorem for a connected graph G. If G is not connected, then, because the complement of a house and the complement of a hole are both connected, we work with the connected components of G, which can be computed in $O(n+m)$ time and space [3]. Thus, suppose that G is connected; then, $n = O(m)$. In order to facilitate the construction of the auxiliary graphs, we compute and store the adjacency matrix of G, which takes $O(n^2)$ time and space. Then, for any vertex v of G, the graph $\widehat{G}_v$ has n vertices and $O(n + n\,deg(v) + deg^2(v)) = O(n\,deg(v))$ edges, where $deg(v)$ is the degree of v in G. We can easily obtain an adjacency-list representation of $\widehat{G}_v$ by computing the sets $N(v)$ of neighbors and $M(v)$ of non-neighbors of v in G, and by taking advantage of the adjacency matrix of G; the computation of the sets $N(v)$ and $M(v)$ takes $O(n+m)$ time and $O(n)$ space, while the construction of the adjacency lists of $\widehat{G}_v$ is completed in $O(m + n\,deg(v))$ additional time and $O(n\,deg(v))$ space for a total of $O(m + n\,deg(v))$ time and $O(n\,deg(v))$ space. The execution of Algorithm Not-in-HHB($\widehat{G}_v, v$) takes $O(\min\{(n + n\,deg(v))\,\alpha(n,n),\ n\,deg(v) + deg(v)\log deg(v)\}) = O(n\,deg(v) + deg(v)\log deg(v)) = O(n\,deg(v))$ time (Theorem 3.2; note that $k = deg(v)$). Thus, we can determine whether the vertex v belongs to a hole in $\widehat{G}_v$ in $O(m + n\,deg(v))$ time and $O(n\,deg(v))$ space. Then, the total time required, i.e., the time required for the computation of the adjacency matrix of G and the processing of all the vertices, is

$$O\big(n^2 + \sum_v (m + n\,deg(v))\big) = O\big(n\,m + n\sum_v deg(v)\big) = O(n\,m).$$

The space complexity is $O(n^2)$ since the space taken by each auxiliary graph $\widehat{G}_v$ can be reused after the processing of v. □

From Theorem 4.5 and the result of Eschen *et al.* [4] (i.e., HHP-free graphs can be recognized in $O(n\,m)$ time and $O(n+m)$ space), we obtain the following theorem.

Theorem 4.6 *Let G be an undirected graph on n vertices and m edges. Then, it can be determined whether G is a WPO-graph in $O(n\,m)$ time and $O(n^2)$ space.*

5 Concluding Remarks

We have presented recognition algorithms for the classes of HH-free and HHD-free graphs running in $O(n\,\min\{m\,\alpha(n,n),\ m + n\log n\})$ time, and for WPO-graphs running in $O(n\,m)$ time, where n is the number of vertices and m is the number of edges of the input graph. Our proposed algorithms are simple and require $O(n+m)$ and $O(n^2)$ space, respectively. Moreover, our HH-free graph recognition algorithm can be easily augmented to yield a certificate (a hole or a house) whenever it decides that the input graph is not HH-free.

We leave as an open problem the designing of $O(n\,m)$-time algorithms for recognizing HH-free and HHD-free graphs; note that an $O(n\,m)$-time algorithm for recognizing HH-free graphs directly implies an $O(n\,m)$-time recognition algorithm for HHD-free graphs. Additionally, in light of the $O(n\,m)$-time recognition of P_4-comparability, P_4-simplicial, bipolarizable, and WPO-graphs, it would be worth investigating whether the recognition of brittle and semi-simplicial graphs is inherently more difficult.

References

[1] A. Brandstädt, V.B. Le, and J.P. Spinrad, *Graph classes: A survey*, SIAM Monographs on Discrete Mathematics and Applications, 1999.

[2] V. Chvátal, Perfectly ordered graphs, *Annals of Discrete Math.* **21**, 63–65, 1984.

[3] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms* (2nd edition), MIT Press, Inc., 2001.

[4] E.M. Eschen, J.L. Johnson, J.P. Spinrad, and R. Sritharan, Recognition of some perfectly orderable graph classes, *Discrete Appl. Math.* **128**, 355–373, 2003.

[5] M.C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, Inc., 1980.

[6] R. Hayward, Meyniel weakly triangulated graphs I: co-perfect orderability, *Discrete Appl. Math.* **73**, 199–210, 1997.

[7] C.T. Hoàng, On the complexity of recognizing a class of perfectly orderable graphs, *Discrete Appl. Math.* **66**, 219–226, 1996.

[8] C.T. Hoàng and N. Khouzam, On brittle graphs, *J. Graph Theory* **12**, 391–404, 1988.

[9] C.T. Hoàng and R. Sritharan, Finding houses and holes in graphs, *Theoret. Comput. Sci.* **259**, 233–244, 2001.

[10] B. Jamison and S. Olariu, On the semi-perfect elimination, *Adv. Appl. Math.* **9**, 364–376, 1988.

[11] R.M. McConnell and J. Spinrad, Linear-time modular decomposition and efficient transitive orientation, *Proc. 5th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA'94)*, 536–545, 1994.

[12] M. Middendorf and F. Pfeiffer, On the complexity of recognizing perfectly orderable graphs, *Discrete Math.* **80**, 327–333, 1990.

[13] S. Olariu, All variations on perfectly orderable graphs, *J. Combin. Theory* Ser. B **45**, 150–159, 1988.

[14] S. Olariu, Weak bipolarizable graphs, *Discrete Math.* **74**, 159–171, 1989.

[15] S. Olariu and J. Randall, Welsh-Powell opposition graphs, *Inform. Process. Lett.* **31**, 43–46, 1989.

[16] D.J. Rose, R.E. Tarjan, and G.S. Lueker, Algorithmic aspects of vertex elimination on graphs, *SIAM J. Comput.* **5**, 266–283, 1976.

[17] R.E. Tarjan, *Data structures and network algorithms*, Society for Industrial and Applied Mathematics, 1983.

[18] D.J.A. Welsh and M.B. Powell, An upper bound on the chromatic number of a graph and its applications to timetabling problems, *Comput. J.* **10**, 85–87, 1967.