Recognizing 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

Abstract: In this paper, we consider the recognition problem on two classes of perfectly orderable graphs, namely, 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 $O(\min\{nm\alpha(n),\, nm + n^2 \log n\})$ time whether a given graph G on n vertices and m edges contains a house or a hole; this leads to an $O(\min\{nm\alpha(n),\, nm + n^2 \log n\})$-time and $O(n + m)$-space algorithm for recognizing HHD-free graphs. We also show that determining whether the complement $\overline{G}$ of the graph G contains a house or a hole can be efficiently resolved in $O(nm)$ time using $O(n^2)$ space, which in turn leads to an $O(nm)$-time and $O(n^2)$-space algorithm for recognizing WPO-graphs. The previously fastest algorithms for recognizing 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 [3] defined the class of graphs that admit a perfect order and called them *perfectly orderable* graphs. The perfectly orderable graphs are perfect; thus, many interesting problems in graph theory, which are NP-complete in general graphs, have polynomial-time solutions in graphs that admit a perfect order [2, 6]; unfortunately, it is NP-complete to decide whether a graph admits a perfect order [14]. 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 [2, 6]); note however that not all subclasses of perfectly orderable graphs admit polynomial-time recognition [9].

In this paper, we consider two classes of perfectly orderable graphs, namely, the HHD-free and the Welsh-Powell opposition graphs. A graph is *HHD-free* if it contains no hole (i.e., a chordless cycle on ≥ 5 vertices), no house, and no domino (D) as induced subgraphs (see Figure 1). In [10], Hoáng and

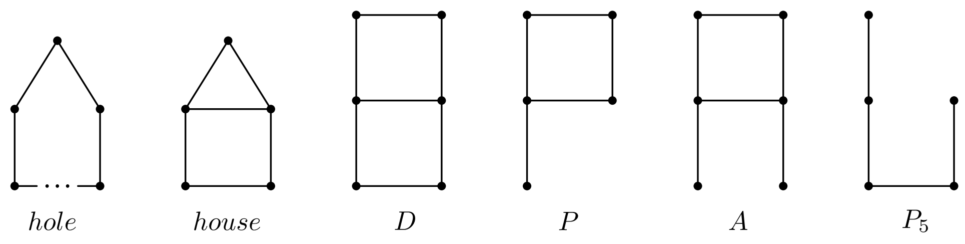

Figure 1: Some simple graphs.

Khouzam proved that the HHD-free graphs admit a perfect order, and thus are perfectly orderable. It is important to note that the HHD-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 [6]. A superclass of HHD-free graphs, which also properly generalizes the class of triangulated graphs, is the class of HH-free graphs; a graph is HH-free if it contains no hole and no house as induced subgraphs (see Figure 1). Chvátal conjectured and later Hayward [8] proved that the complement $\overline{G}$ of an HH-free graph G is also 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 $\mathbf{bip}^*$ graphs (see [2]), and hence are perfect [17]. The complexity of recognizing opposition graphs is unknown. It is also open whether there is an opposition graph that is not perfectly orderable [2]. 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 $c \prec d$ [18]. 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 [22].

Hoàng and Khouzam [10], 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 [11] 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.* [5] 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 D, 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 [18]) and the use of modular decomposition [13] allowed Eschen *et al.* to present an $O(nm)$-time recognition algorithm for HHP-free graphs.

For the class of WPO-graphs, Olariu and Randall [19] 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.* [5] combined their $O(nm)$-time recognition algorithm for HHP-free graphs with the

$O(n^3)$-time recognition algorithm for HH-free graphs proposed in [11], and showed that WPO-graphs can be recognized in $O(n^3)$ time.

In this paper, we present efficient algorithms for recognizing HHD-free graphs 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 $O(\min\{nm\alpha(n), nm + n^2 \log n\})$-time and $O(n+m)$-space algorithm for determining whether a graph on n vertices and m edges is HH-free. This result, along with results by Jamison and Olariu [12] and by Hoáng and Khouzam [10], enables us to describe an algorithm for recognizing HHD-free graphs which runs in $O(\min\{nm\alpha(n), nm + n^2 \log n\})$ 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 contains a hole or a house in $O(nm)$ time using $O(n^2)$ space. In light of the characterization of WPO-graphs due to Olariu and Randall [19] 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 $O(nm)$-time recognition algorithm for HHP-free graphs of Eschen *et al.* [5], our result yields an $O(nm)$-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. The recognition algorithms for HHD-free graphs and WPO-graphs are described and analyzed in Sections 4 and 5, respectively. Section 6 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 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

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.      $X \leftarrow \{x \in N(u) \mid \sigma^{-1}(u) < \sigma^{-1}(x)\}$;
5.      if $X \neq \emptyset$
6.      then $w \leftarrow \sigma(min\{\sigma^{-1}(x) \mid x \in X\})$;
7.          concatenate $X - \{w\}$ to $A(w)$;
8.      if $A(u) - N(u) \neq \emptyset$ then return("false");

9.  return("true");
```

Figure 2: The perfect elimination ordering testing algorithm.

$x, y \in V(G)$. 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 [2, 6, 20]. The notion of a simplicial vertex was generalized by Jamison and Olariu [12] 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 [5]).

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 [6, 20]. 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. Since neither the graph G nor the ordering σ changes during the execution of the Algorithm PEO, we can without error replace Step 6 by

$$w \leftarrow Next\_Neighbor_{G,\sigma}[u];$$

where $Next\_Neighbor_{G,\sigma}[\,]$ is an array whose values have been precomputed in accordance with the assignment in Step 6 of the Algorithm PEO shown in Figure 2.

3 Recognizing HH-free graphs

The most important ingredient (and the bottleneck too) of the HHD-free graph recognition algorithm of Hoàng and Sritharan [11] 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[1] (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. ∎

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.

($\Longrightarrow$) 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 $\sigma^{-1}(y) < \sigma^{-1}(z)$ (Figure 4(a), where the dotted edge indicates a potential edge of G).

[1]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.

Algorithm Not-in-HHB(G, v)

```
1.  Compute a DegMN-ordering $\sigma = (y_1, y_2, \ldots, y_k)$ of the non-neighbors of $v$
    in the graph $G$;
    compute the array $Next\_Neighbor_{G'_v,\sigma}[\,]$ for the non-neighbors of $v$;
    for each non-neighbor $u$ of $v$ do
        $A(u) \leftarrow \emptyset$;

2.  for $i \leftarrow 1$ to $k$ do
3.      $u \leftarrow \sigma(i)$;
4.      $X \leftarrow N(u) \cap N(v)$;          {note: $\forall x \in X,\ \sigma^{-1}(u) < \sigma^{-1}(x)$}
5.      if $X \neq \emptyset$
6.      then $w \leftarrow Next\_Neighbor_{G'_v,\sigma}[u]$;
7.            if $w \in M(v)$ then concatenate $X$ to $A(w)$;       {note: $w \notin X$}
8.      if $A(u) - N(u) \neq \emptyset$ then return("false");

9.  return("true");
```

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

Then, we show that $Next\_Neighbor_{G'_v,\sigma}[y] = z$. Suppose for contradiction that $Next\_Neighbor_{G'_v,\sigma}[y] = w \neq z$. 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, $n(y,v) \leq n(w,v)$, 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 $n(w,v) \leq n(z,v)$ (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. The above can be used to yield 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. That is, we can show the following result.

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 (i.e, when $i = \sigma^{-1}(z)$). This happens only if at Step 8 of the current iteration of the for-loop there exists a vertex

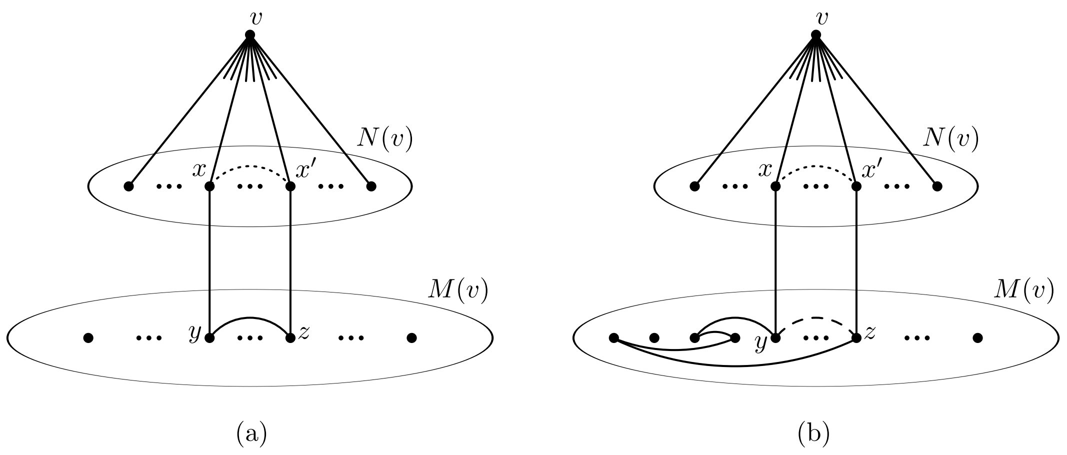

Figure 4:

$x \in A(z) - N(z)$. Then, from Lemmata 3.1 and 3.2, we have that $z \in M(v)$ and $A(z) - N(z) \subseteq N(v)$, 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, $xy \in E(G)$, $\sigma^{-1}(y) < \sigma^{-1}(z)$ which implies that $y \in M(v)$, and z is the $Next\_Neighbor_{G'_v,\sigma}[y]$ which implies that $yz \in E(G'_v)$. 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 $x'y \notin E(G)$ (Figure 4(a), where the dotted edge indicates a potential edge of G). For the vertices y, z (which are adjacent in G'_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) $yz \notin E(G)$. The edge yz was added during the construction of the graph G'_v which was produced so that each vertex $y_i \in M(v)$ becomes simplicial in the graph $G[\{y_i, y_{i+1}, \ldots, y_k\}]$, $1 \le i \le k$. It follows that G has an induced path on at least three vertices connecting y and z all of whose vertices are in $M(v)$ and precede the vertices y and z in σ (see also Lemma 2 of [11]). Let $a_1 a_2 \cdots a_q$ be a shortest such path where $a_1 = y$ and $a_q = z$; then this path is chordless (see Figure 4(b), where the dashed edge indicates an edge in $E(G'_v) - E(G)$). Since all the vertices $a_2, \ldots, a_{q-1}$ precede y in σ and since the Algorithm Not-in-HHB has not stopped before processing z, then $N(a_i) \cap N(v) \subseteq N(y) \cap N(v)$, $1 < i < q$. This implies that $x' \notin N(a_i)$ for all $i = 2, 3, \ldots, q-1$. 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 $a_1 = y$ and $xy \in E(G)$, and $a_q = z$ 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.

($\Longleftarrow$) 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 $x'z \in E(G)$), whereas if v belongs to a hole on more than 5 vertices or is the top of a building, let $v, x, y, a_1, \ldots, a_p, z, x'$ be the vertices inducing the hole or the building ($x, x' \in N(v)$, $y, a_1, \ldots, a_p, z \in M(v)$, $xy \in E(G)$, and $x'z \in E(G)$). Moreover, suppose without loss of generality that in either case $\sigma^{-1}(y) < \sigma^{-1}(z)$.

The definition of the graph G'_v (i.e., in G'_v, every vertex $y_i \in M(v)$ is simplicial in the subgraph induced by $\{y_i, y_{i+1}, \ldots, y_k\}$) implies that even if y and z are not adjacent in G, they are

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");
```

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

adjacent in G'_v. Then, there exists a sequence $y{=}b_1, b_2, \ldots, b_q{=}z$ of vertices in $M(v)$ such that $b_i = Next\_Neighbor_{G'_v,\sigma}[b_{i-1}]$ for $2 \leq i \leq q$ (note that $b_i = Next\_Neighbor_{G'_v,\sigma}[b_{i-1}]$ implies that $\sigma^{-1}(b_{i-1}) < \sigma^{-1}(b_i)$ and $b_{i-1}b_i \in E(G'_v)$). The existence of such a sequence is shown as follows: if z is the next-neighbor of y in G'_v, then the sequence is precisely y, z; otherwise, if b_2 is the next-neighbor of y, then $yb_2 \in E(G'_v)$ which from the fact that $yz \in E(G'_v)$ and from the definition of G'_v implies that $b_2 z \in E(G'_v)$; next, we repeat the above argument for b_2 and so on so forth; since there is a finite number of vertices between y and z in σ, eventually, we will find a vertex b_{q-1} such that z is b_{q-1}'s next-neighbor in G'_v. 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 $b_{r+1} = Next\_Neighbor_{G'_v,\sigma}[b_r]$, 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. ∎

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.

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

In order to avoid computing the graph G'_v, we take advantage of the following property of the chordal completion of a graph:

Lemma 3.3. *Let G be a graph, let $(v_1, v_2, \ldots, v_k)$ be an ordering of its vertices, and let G' be the graph resulting from G after the addition of 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 $r < \hat{r}$.

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 $v_tv_{\hat{r}}, v_tv_j \in E(G')$. Since $v_tv_{\hat{r}} \in E(G')$ and $t < \hat{r}$, then by the inductive hypothesis, there exists a vertex v_p, where $p \leq t < \hat{r}$, such that $v_pv_{\hat{r}}$ 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_tv_j of G', there exists a vertex v_q, where $q \leq t < j$, such that v_qv_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 $v_pv_{\hat{r}}$ in G and the fact that $p \leq t < \hat{r}$ imply that $v_p, v_{\hat{r}}$ belong to the same connected component of $G[\{v_1, v_2, \ldots, v_{\hat{r}}\}]$; moreover, since v_pv_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 $v_rv_j \in E(G')$, 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_iv_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 $v_rv_j \in E(G')$.

If $v_rv_j \in E(G)$, then clearly $v_rv_j \in E(G')$. Suppose now that $v_rv_j \notin E(G)$. 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 $v_rv_{p_1}v_{p_2}\cdots v_{p_\ell}v_i$ 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_iv_j is an edge of G, then $v_rv_{p_1}v_{p_2}\cdots v_{p_\ell}v_iv_j$ 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 $v_{p_0}v_{p_{\ell+2}} = v_rv_j$ is a path (= edge) in G', as desired. ∎

We note that the above lemma implies Lemma 2 of [11] as a corollary. Lemma 3.3 implies that for the computation of the value $Next\_Neighbor_{G'_v,\sigma}[y_r]$, where $\sigma = (y_1, y_2, \ldots, y_k)$, it suffices to find the leftmost (w.r.t. σ) vertex among $y_{r+1}, y_{r+2}, \ldots, y_k$ which is adjacent in G to a vertex in the connected component of $G[\{y_1, y_2, \ldots, y_r\}]$ to which y_r belongs. This can be efficiently done by processing the vertices in the order they appear in σ. In detail, the algorithm to compute the contents of the array $Next\_Neighbor_{G'_v,\sigma}[\,]$ is presented in Figure 6.

It is important to observe that, at the completion of the processing of vertex y_j, the sets of vertices maintained by the algorithm are in a bijection with the connected components of $G[\{y_1, y_2, \ldots, y_j\}]$; while processing y_j, we consider the edges y_iy_j where $i < j$, and we union the set containing y_j (which has vertex y_j as its rightmost vertex with respect to σ) to another set iff y_j is adjacent to a vertex in that set. The correctness of the algorithm is established in the following lemma.

Lemma 3.4. *The Algorithm Compute-Next_Neighbor correctly computes the values of $Next\_Neighbor_{G'_v,\sigma}[y_i]$ for all the vertices $y_i \in M(v)$ (i.e., all the vertices that are not adjacent to v in G).*

Algorithm Compute-Next_Neighbor(G, σ, v)

```
1.  {let $y_1, y_2, \ldots, y_k$ be the non-neighbors of $v$ in the order they appear in $\sigma$}
    make a set containing the vertex $y_1$;

2.  for $j = 2, 3, \ldots, k$ do
3.      make a set containing the vertex $y_j$;
4.      for each edge $y_i y_j$ of $G$, where $i < j$, do
5.          $y_r \leftarrow$ the rightmost (w.r.t. $\sigma$) vertex in the set to which $y_i$ belongs;
6.          if $y_r \neq y_j$
7.          then    {$y_i$ and $y_j$ belong to different sets}
8.                $Next\_Neighbor_{G'_v,\sigma}[y_r] \leftarrow y_j$;
9.                union the sets to which $y_i$ and $y_j$ belong;
```

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

Proof: First, we show that, for a vertex $y_r \in M(v)$, the Algorithm Compute-Next_Neighbor assigns a value to the entry $Next\_Neighbor_{G'_v,\sigma}[y_r]$ if and only if the vertex y_r is adjacent in the graph G'_v to a vertex in $\{y_{r+1}, y_{r+2}, \ldots, y_k\}$.

The entry $Next\_Neighbor_{G'_v,\sigma}[y_r]$ is assigned a value if, during the processing of a vertex y_j, where $j > r$, there exists an edge in G connecting y_j to a vertex y_i, and y_r is the rightmost (w.r.t. σ) vertex among the vertices in the set containing y_i. Since the vertices y_i and y_r belong to the same set, which corresponds to the same connected component of the subgraph $G[\{y_1, y_2, \ldots, y_{j-1}\}]$, and since y_r is the rightmost vertex in the set, then y_i, y_r belong to the same connected component of $G[\{y_1, y_2, \ldots, y_r\}]$. Due to this and due to the fact that $y_r y_j \in E(G)$, where $r < j$, Lemma 3.3 implies that the chordal completion of the subgraph $G[M(v)]$ induced by the non-neighbors of v in G, and thus the graph G'_v, contains the edge $y_r y_j$.

Suppose now that a vertex y_i is adjacent in G'_v to a vertex in $\{y_{i+1}, y_{i+2}, \ldots, y_k\}$; we show that the Algorithm Compute-Next_Neighbor assigns a value to the entry $Next\_Neighbor_{G'_v,\sigma}[y_i]$. Let y_s be the leftmost (w.r.t. σ) vertex among the vertices in $\{y_{i+1}, y_{i+2}, \ldots, y_k\}$ which are adjacent in G to a vertex in the connected component of $G[\{y_1, y_2, \ldots, y_i\}]$ to which y_i belongs. Then, right before processing y_s, the rightmost (w.r.t. σ) vertex in the set to which y_i belongs is precisely y_i. Suppose for contradiction that this is not the case and let $y_{s'}$ be the rightmost vertex, where $s' > i$. Then, there exists a simple path in G from y_i to $y_{s'}$; let it be $y_{p_0} y_{p_1} y_{p_2} \cdots y_{p_h}$, where $y_{p_0} = y_i$ and $y_{p_h} = y_{s'}$. 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, $y_{p_{\hat{t}}} y_{p_{\hat{t}-1}} \in E(G)$ and $y_{p_{\hat{t}-1}}, y_i$ belong to the same connected component of $G[\{y_1, y_2, \ldots, y_i\}]$ (since $y_{p_0} y_{p_1} \cdots y_{p_{\hat{t}}}$ is a path in G and $p_t \leq i$ for all $t = 0, 1, \ldots, \hat{t}-1$). However, this comes to a contradiction with the definition of y_s, because $i < p_{\hat{t}} < s$. Therefore, right before processing vertex y_s, y_i indeed is the rightmost (w.r.t. σ) vertex in the set to which it belongs, and thus the entry $Next\_Neighbor_{G'_v,\sigma}[y_i]$ will be assigned a value (it will be set equal to y_s during the processing of vertex y_s).

The proof will be complete if we show that, whenever the Algorithm Compute-Next_Neighbor assigns a value to an entry $Next\_Neighbor_{G'_v,\sigma}[y_r]$, it assigns the correct value. Observe that whenever the algorithm executes an assignment $Next\_Neighbor_{G'_v,\sigma}[y_r] \leftarrow y_j$, then $r < j$ and there exists an edge $y_i y_j$ in G such that $i < j$ and the vertices y_i, y_j belong to the same connected component of the subgraph $G[\{y_1, y_2, \ldots, y_r\}]$. Then, Lemma 3.3 implies that the chordal completion of the subgraph $G[M(v)]$ induced by the non-neighbors $M(v)$ of v, and thus the graph G'_v as well, contains the edge $y_r y_j$. We need to show that y_r is not adjacent in the chordal completion of $G[M(v)]$ to any vertex y_t, where $r < t < j$. Suppose for contradiction that there existed such a vertex y_t. Then, by Lemma 3.3, there exists a vertex y_s such that $s < r$, $y_s y_t \in E(G)$, and y_s, y_r belong to the same connected component of $G[\{y_1, y_2, \ldots, y_r\}]$. But then, while processing the vertex y_t, y_t will be included in the set containing y_r, in contradiction to the fact that, while processing y_j (which is done after processing y_t since $t < j$), the rightmost (w.r.t. σ) vertex in the set containing y_r is precisely y_r; recall that $r < t$. Therefore, all the assignments to the entries of the array $Next\_Neighbor_{G'_v,\sigma}[\,]$ are correct. ∎

3.2 Time and Space Complexity

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. 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 $Next\_Neighbor_{G'_v,\sigma}[\,]$. So, let us now turn to the time and space complexity of the Algorithm Compute-Next_Neighbor(G, σ, v). 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), then the rest of the execution of the Algorithm Compute-Next_Neighbor takes $O(n + m)$ time. The sets are maintained by our algorithm in a fashion amenable for Union-Find operations, where additionally the representative of each set also contains a field storing the rightmost (w.r.t. σ) vertex in the set. Then,

- making a set which contains a single vertex y_i requires building the set and setting the rightmost (w.r.t. σ) vertex in the set to y_i;
- finding the rightmost (w.r.t. σ) vertex in the set to which a vertex y_j belongs requires performing a Find operation to locate the representative of the set from which the rightmost vertex is obtained in constant time per Find operation;
- 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 y_j, where y_j is the rightmost vertex in any of the sets, then the rightmost vertex of the resulting set is y_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 $y_1, y_2, \ldots, y_k$, 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 connecting two non-neighbors of v. Hence, if we use disjoint-set forests to maintain the sets, the time to execute the above operations is $O(m\alpha(k))$ [4], where $\alpha(\,)$ is a very slowly growing function; if instead we use the linked-list representation, then the time is $O(m + k \log k)$ [4]. 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 $Next\_Neighbor_{G'_v,\sigma}[\,]$ for the k non-neighbors of the vertex v takes a total of $O(n + \min\{m\alpha(k), m + k \log k\})$ time and $O(k)$ space. Therefore, we have:

Theorem 3.2. *Let G be a graph on n vertices and m edges. Determining whether a vertex v of G belongs to a hole or is the top of a house or a building can be done in $O(n + \min\{m\alpha(k), m + k \log k\})$ time and $O(n + m)$ space, where k is the number of non-neighbors of v in G.*

Then, since the Algorithm Recognize-HH-free consists of applying the Algorithm Non-in-HHB on every vertex of the input graph, we obtain the following corollary:

Corollary 3.1. *Determining whether a graph G on n vertices and m edges contains a hole or a house (i.e., is not HH-free) can be done in $O(\min\{nm\alpha(n), nm + n^2 \log n\})$ 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 vertex u which is not adjacent to 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 is $Next\_Neighbor_{G'_v,\sigma}[y]$)[2]. Then, we can obtain a hole or a house of G by doing the following:

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 $n(y, v) \leq n(u, v)$ (note that y precedes u in σ) 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 vertex u and the vertices preceding u in σ, and we apply BFS on it starting at y until u is reached; 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, it is not difficult to see that all three steps of the certificate computation take linear time and space, if we assume that we have an adjacency-list representation of the input graph and that we use two auxiliary arrays to mark the neighbors of v and of y. 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 $O(n + m)$ space.*

[2] 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.

4 Recognition of HHD-free Graphs

Our HHD-free graph recognition algorithm is motivated by the corresponding algorithm of Hoàng and Sritharan [11], which in turn is motivated by the work of Hoàng and Khouzam [10] and relies on the following characterization of HHD-free graphs proved by Jamison and Olariu:

Theorem 4.1. (Jamison and Olariu [12]) *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.

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 below.

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").
```

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).

4.1 Time and Space Complexity

According to Corollary 3.1, Step 1 takes $O(\min\{nm\alpha(n), nm + n^2 \log n\})$ time and $O(n+m)$ space. Step 2 takes $O(n+m)$ time and space [6, 20]. 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 $G[\{v_1, \ldots, v_n\}] = G$ and that $G[\{v_{i+1}, \ldots, v_n\}]$ can be obtained from $G[\{v_i, \ldots, v_n\}]$ 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 $G[\{v_{i+1}, \ldots, v_n\}]$ from $G[\{v_i, \ldots, v_n\}]$ takes time proportional to the degree of v_i in $G[\{v_i, \ldots, v_n\}]$ 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 [11]):

Theorem 4.2. (Hoàng and Khouzam [10]) *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 $G[\{v_i, \ldots, v_n\}]$, computing its co-components, and testing whether a vertex set is a module in $G[\{v_i, \ldots, v_n\}]$ can all be done in time and space linear in the size of $G[\{v_i, \ldots, v_n\}]$, Step 3 takes a total of $O\Big(\sum_i \Big(n+m+deg(v_i)\Big)\Big) = O(nm)$ 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, it can be determined whether G is an HHD-free graph in $O(\min\{nm\alpha(n), nm + n^2 \log n\})$ time and $O(n+m)$ space.*

4.2 Providing a Certificate

If the Algorithm Recognize-HH-free answers that the input graph G is not HH-free in Step 1 (in which case the Algorithm Recognize-HHD-free answers that the graph is not HHD-free), then the graph G contains a hole or a house and a certificate can be obtained by the augmented algorithm described in Section 3.3 in time and space linear in the size of G (Theorem 3.3). The Algorithm Recognize-HHD-free may also answer that G is not HHD-free in Step 3 as well, if an induced domino is detected in G. In order to locate such a domino, we replace Step 3 of the Algorithm Recognize-HHD-free by the following steps:

```
3. for $i = n-5, n-6, \ldots, 1$ do
        if $v_i$ is not semi-simplicial in $G[\{v_i, v_{i+1}, \ldots, v_n\}]$
        then
        3.1 find a $P_4$ $xv_iyz$ in $G[\{v_i, v_{i+1}, \ldots, v_n\}]$;
        3.2 construct three auxiliary arrays of size $n$ each, which mark the neighbors of $x$, $z$, and
            $v_i$, respectively, in $G[\{v_i, v_{i+1}, \ldots, v_n\}]$;
        3.3 for each vertex $a \in \{v_{i+1}, v_{i+2}, \ldots, v_n\}$ do
                if $a$ is adjacent to $x$ and non-adjacent to $v_i$
                then        {$a$ differs from $v_i, x, y, z$, and $az \notin E(G)$}
                    for each neighbor $b$ of $a$ in $G[\{v_i, v_{i+1}, \ldots, v_n\}]$ do
                        if $b$ is adjacent to $z$ and non-adjacent to $v_i$
                        then break;      {$b$ differs from $v_i, x, y, z, a$, and $bx, by \notin E(G)$}
        3.4 print(Domino: $v_i, x, y, z, a, b$);
            return("the graph is not HHD-free").
```

We note that in the augmented algorithm the vertices are processed in reverse order compared to what they used to. This has the advantage that the vertex v_i, which is first found not to be semi-simplicial, is guaranteed to belong to all the induced dominoes in the graph $G[\{v_i, v_{i+1}, \ldots, v_n\}]$; in fact, we show that any P_4 of $G[\{v_i, v_{i+1}, \ldots, v_n\}]$ with v_i as a midpoint participates in a domino with v_i being a corner vertex (Lemma 4.2). Additionally, since after Step 1 of Algorithm Recognize-HHD-free, the input graph G and all induced subgraphs are HH-free, if vertex a is adjacent to x and is not adjacent to v_i then a is not adjacent to z (no matter whether a is adjacent to y or not); for the same reason, if b is adjacent to a, z and is not adjacent to v_i, then b is not adjacent to x or y, and a is adjacent to y. Thus, the vertices v_i, x, y, z, a, b induce a domino in G. The above discussion and Lemma 4.2 establish the correctness of the augmented algorithm. In order to prove Lemma 4.2, we take advantage of the following well known property of a LexBFS ordering [6]

P1: Let $(v_1, v_2, \ldots, v_n)$ be an ordering of the vertices of a graph G obtained by running LexBFS on G starting at vertex v_n. Then, for any vertices v_i, v_j, v_k such that $i < j < k$ and $v_k \in N(v_i) - N(v_j)$, there exists a vertex v_t such that $t > k$ and $v_t \in N(v_j) - N(v_i)$.

as well as another property which we establish in Lemma 4.1.

Lemma 4.1. *Let G be an HH-free graph and let $(v_1, v_2, \ldots, v_n)$ be an ordering of the vertices of G obtained by running LexBFS on G starting at vertex v_n. If for three vertices v_p, v_q, v_r of G it holds that $p < q < r$ and $v_p v_q \in E(G)$, $v_p v_r \in E(G)$, and $v_q v_r \notin E(G)$, and if v_s is the rightmost vertex in the ordering which belongs to $N(v_q) - N(v_p)$, then $v_r v_s \in E(G)$.*

Proof: We note that v_s is well defined: according to Property P1, the facts $p < q < r$ and $v_q v_r \notin E(G)$ imply that there exists a vertex v_t such that $t > r$ and $v_t \in N(v_q) - N(v_p)$. Suppose for contradiction that $v_r v_s \notin E(G)$. To simplify the notation, let $v_{i_0} = v_p$, $v_{i_1} = v_q$, $v_{i_2} = v_r$, and $v_{i_3} = v_s$; clearly, $i_0 < i_1 < i_2 < i_3$, and

$$v_{i_1} v_{i_0} \in E(G) \tag{1}$$
$$v_{i_2} v_{i_0} \in E(G) \qquad v_{i_2} v_{i_1} \notin E(G) \tag{2}$$
$$v_{i_3} v_{i_0} \notin E(G) \qquad v_{i_3} v_{i_1} \in E(G) \qquad v_{i_3} v_{i_2} \notin E(G). \tag{3}$$

Let us consider the following iterative process:

```
1. $j \leftarrow 3$;
2. while $v_{i_j} v_{i_{j-1}} \notin E(G)$ do
3.     $A \leftarrow N(v_{i_{j-1}}) - N(v_{i_{j-2}})$;
4.     $v_{i_{j+1}} \leftarrow$ rightmost vertex (w.r.t. the LexBFS ordering) among the elements of $A$;
5.     $j \leftarrow j + 1$;
```

It is important to note that every time the condition of the while-loop is checked, we have that v_{i_j} is well defined, and $i_{j-2} < i_{j-1} < i_j$: initially, we have $v_{i_j} = v_{i_3}$, which, as we explained above, is well defined, and $i_1 < i_2 < i_3$; furthermore, if $i_{j-2} < i_{j-1} < i_j$, then if the while-loop condition is found true, Property P1 applied on $v_{i_{j-2}}, v_{i_{j-1}}$, and v_{i_j} implies that the set A contains at least one element v_t with $t > i_j$, and thus $v_{i_{j+1}}$ is well defined and $i_j < i_{j+1}$, i.e., $i_{j-1} < i_j < i_{j+1}$. Since there are only finitely many vertices to the right of v_{i_3} in the LexBFS ordering, eventually a vertex v_{i_ℓ} will be found such that $v_{i_\ell} v_{i_{\ell-1}} \in E(G)$ and $i_\ell > i_3$. The above iterative process and the definition of the set A imply:

$$v_{i_j} v_{i_{j-1}} \notin E(G) \qquad \forall j = 3, 4, \ldots, \ell - 1; \tag{4}$$
$$v_{i_\ell} v_{i_{\ell-1}} \in E(G); \tag{5}$$
$$v_{i_j} v_{i_{j-2}} \in E(G) \qquad \forall j = 4, 5, \ldots, \ell; \tag{6}$$
$$v_{i_j} v_{i_{j-3}} \notin E(G) \qquad \forall j = 4, 5, \ldots, \ell. \tag{7}$$

Next, we show that, for each $k = 3, 4, \ldots, \ell$, vertex v_{i_k} is not adjacent to any of $v_{i_0}, v_{i_1}, \ldots, v_{i_{k-3}}$. Consider any k in the above range; we show by induction on $j = k-3, k-2, \ldots, 0$ that $v_{i_k} v_{i_j} \notin E(G)$. For the basis case, we have $j = k-3$; then, $v_{i_k} v_{i_{k-3}} \notin E(G)$ due to Eq. (3) if $k = 3$ and due to Eq. (7) if $k \geq 4$. For the inductive hypothesis, we assume that $v_{i_k} v_{i_j} \notin E(G)$ for some j such that $1 \leq j \leq k-3$; for the inductive step, we need to show that $v_{i_k} v_{i_{j-1}} \notin E(G)$. If $v_{i_k} v_{i_{j-1}} \in E(G)$, then Property P1 applies on the vertices $v_{i_{j-1}}, v_{i_j}, v_{i_k}$ (note that $i_{j-1} < i_j < i_k$ and that $v_{i_j} v_{i_k} \notin E(G)$) and implies that

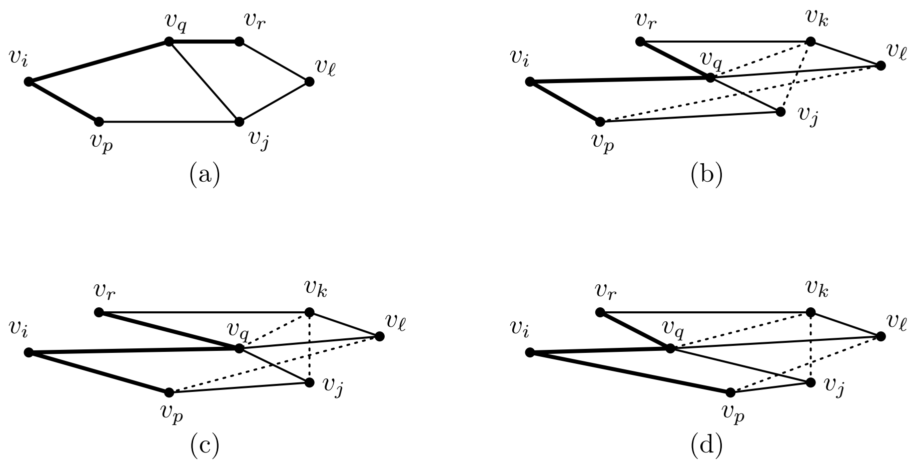

Figure 7: Cases 1–4 of Lemma 4.2.

there exists a vertex $v_t \in N(v_{i_j}) - N(v_{i_{j-1}})$ where $t > i_k$; however, since $i_{j+2} < i_k < t$, the existence of v_t contradicts the definition of $v_{i_{j+2}}$ as the rightmost vertex with respect to the LexBFS ordering which belongs to $N(v_{i_j}) - N(v_{i_{j-1}})$ (see line 4 of the iterative process given above). Thus, for all k, j such that $3 \leq k \leq \ell$ and $0 \leq j \leq k-3$, $v_{i_k} v_{i_j} \notin E(G)$. Taking also into account Eqs (1)-(7), we conclude that the vertices $v_{i_0}, v_{i_1}, \ldots, v_{i_\ell}$, where $\ell \geq 4$, induce a hole in the graph $G[\{v_i, v_{i+1}, \ldots, v_n\}]$, in contradiction to the fact that G contains no holes. Therefore, $v_r v_s \in E(G)$. ∎

Lemma 4.2. *Let G be an HH-free graph and let $(v_1, v_2, \ldots, v_n)$ be an ordering of the vertices of G obtained by running LexBFS on G starting at vertex v_n. Suppose further that, for each $j = i+1, i+2, \ldots, n$, vertex v_j is semi-simplicial in the graph $G[\{v_j, v_{j+1}, \ldots, v_n\}]$ whereas vertex v_i is* not *semi-simplicial in the graph $G_i = G[\{v_i, v_{i+1}, \ldots, v_n\}]$. Then, for every P_4 $v_p v_i v_q v_r$ in G_i, the graph G_i (and hence G as well) contains a domino as shown in Figure 8.*

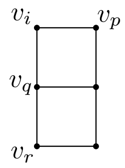

Figure 8:

Proof: Since the graph G_i contains the P_4 $v_p v_i v_q v_r$, clearly $i < p, q, r \leq n$. We consider the following cases based on the relative order of the vertices v_p, v_q, v_r in the LexBFS ordering: case 1 covers the case $q < r$, whereas cases 2, 3, and 4 (which are similar to each other) cover all the possibilities if $q > r$.

1. $q < r$: Lemma 4.1 applies on v_i, v_p, v_q: if $p < q$, it implies that the rightmost vertex in the LexBFS ordering that is adjacent to v_p and non-adjacent to v_i is also adjacent to v_q (see Figure 7(a)); if $q < p$, it implies that the rightmost vertex in the LexBFS ordering which is adjacent to v_q and non-adjacent to v_i is also adjacent to v_p. In either case, let that rightmost vertex be v_j; clearly, v_j differs from any of v_p, v_q, v_r. Then, $v_j v_r \notin E(G)$, otherwise the vertices v_i, v_p, v_q, v_r, v_j would induce a house in G. Lemma 4.1 then applies on v_q, v_r, v_j (no matter whether $j < r$ or $r < j$) and implies that there exists a vertex, say, v_ℓ, to the right of v_j and v_r in the LexBFS ordering such that $v_\ell v_j \in E(G)$, $v_\ell v_r \in E(G)$, and $v_\ell v_q \notin E(G)$; see Figure 7(a). Then, $v_\ell v_p \notin E(G)$, otherwise the vertices $v_i, v_p, v_q, v_r, v_\ell$ would induce a house or a C_5 in G. Furthermore, $v_\ell v_i \notin E(G)$, otherwise Property P1 would apply on the vertices v_i, v_p, v_ℓ (or v_i, v_q, v_ℓ if $q < p$) and would contradict the fact that v_j is the rightmost vertex that is adjacent to v_p (v_q, respectively) and non-adjacent to v_i. Then, the vertices $v_i, v_p, v_q, v_r, v_j, v_\ell$ induce a domino as shown in Figure 8.

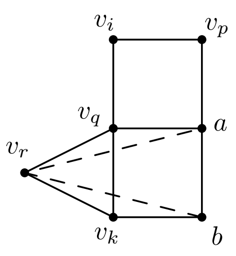

Figure 9:

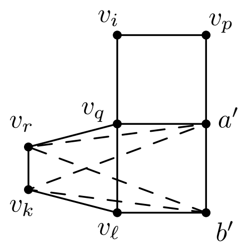

Figure 10:

2. *$p < r < q$:* Lemma 4.1 applies on v_i, v_p, v_q and it implies that the rightmost vertex, say, v_j, in the LexBFS ordering that is adjacent to v_p and non-adjacent to v_i is also adjacent to v_q (Figure 7(b)). Then, $v_j v_r \notin E(G)$, otherwise the vertices v_i, v_p, v_q, v_r, v_j would induce a house in G. Property P1 on v_p, v_r, v_j implies that there exists a vertex v_k such that $k > j$, $v_k v_r \in E(G)$, and $v_k v_p \notin E(G)$. Then, $v_k v_i \notin E(G)$, otherwise Property P1 would apply on v_i, v_p, v_k implying that there would exist a vertex v_t such that $t > k$ and $v_t \in N(v_p) - N(v_i)$, in contradiction to the definition of v_j since $j < k < t$. We distinguish the following two cases:

   (a) $v_k v_q \in E(G)$: Then, G_i contains the P_4 $v_p v_i v_q v_k$, where $i < p < q < k$; Case 1 above applies and implies that this P_4 along with two vertices, say, a and b, participate in a domino (see Figure 9). The vertex v_r is adjacent to both v_q and v_k; thus, it differs from both a and b. Additionally, v_r has to be adjacent to at least one of a, b, otherwise a house would be induced; however, it cannot be adjacent to a otherwise v_i, v_p, v_q, v_r, a would induce a house. Thus, v_r is adjacent to b and the vertices v_i, v_p, v_q, v_r, a, b induce a domino as shown in Figure 8.

   (b) $v_k v_q \notin E(G)$: Then, if $v_k v_j \in E(G)$, the vertices $v_i, v_p, v_q, v_r, v_j, v_k$ induce a domino as shown in Figure 8. Suppose that $v_k v_j \notin E(G)$; see Figure 7(b). Lemma 4.1 applies on v_r, v_q, v_k and implies that the rightmost vertex, say, v_ℓ, in the LexBFS ordering that is adjacent to v_q and non-adjacent to v_r is also adjacent to v_k. Then, $v_\ell v_i \notin E(G)$, otherwise the vertices $v_i, v_q, v_r, v_k, v_\ell$ would induce a house in G. If $v_\ell v_p \in E(G)$, then the vertices $v_i, v_p, v_q, v_r, v_k, v_\ell$ induce a domino as shown in Figure 8. Suppose now that $v_\ell v_p \notin E(G)$. Then, G_i contains the P_4 $v_p v_i v_q v_\ell$, where $i < p < q < \ell$; Case 1 above applies implying that this P_4 along with two vertices, say, a' and b', participate in a domino (see Figure 10); since $v_r v_p \notin E(G)$ and $v_r v_q \in E(G)$, v_r differs from either of a', b', whereas v_k may coincide with b'. If $v_k = b'$ then because $v_r a' \notin E(G)$ (otherwise the vertices v_i, v_p, v_q, v_r, a' would induce a house in G), the vertices $v_i, v_p, v_q, v_r, a', v_k$ induce a domino as shown in Figure 8. Let us consider now that $v_k \neq b'$; recall that v_k is adjacent to v_r, v_ℓ and non-adjacent to v_q. Since the vertices $v_q, v_r, v_k, v_\ell, a', b'$ cannot form a domino (that would have been a domino in G_{i+1}), at least one of v_r, v_k is adjacent to at least one of a', b'. Clearly, $v_r a' \notin E(G)$, otherwise G would contain an induced house. Then, if $v_r b' \in E(G)$, the vertices $v_i, v_p, v_q, v_r, a', b'$ induce a domino as shown in Figure 8. If $v_r b' \notin E(G)$, then $v_k a' \in E(G)$; otherwise, $v_k b' \in E(G)$ (recall that at least one of v_r, v_k is adjacent to at least one of a', b') and the vertices v_q, v_r, v_k, a', b' would induce a hole. Then, the vertices $v_i, v_p, v_q, v_r, a', v_k$ induce a domino as shown in Figure 8.

3. *$r < p < q$:* Lemma 4.1 applies on v_i, v_p, v_q and it implies that the rightmost vertex, say, v_j, in the LexBFS ordering that is adjacent to v_p and non-adjacent to v_i is also adjacent to v_q (see

Figure 7(c)). Then, $v_j v_r \notin E(G)$, otherwise the vertices v_i, v_p, v_q, v_r, v_j would induce a house in G. Property P1 on v_i, v_r, v_p implies that there exists a vertex v_k such that $k > p$, $v_k v_r \in E(G)$, and $v_k v_i \notin E(G)$; note that k may be smaller than q, between q and j, or even larger than j. Then, $v_k v_p \notin E(G)$, otherwise the vertices v_i, v_p, v_q, v_r, v_k would induce a house or a C_5 depending on whether v_k is adjacent to v_q or not. We distinguish the following two cases:

(a) $v_k v_q \in E(G)$: Then, G_i contains the P_4 $v_p v_i v_q v_k$; Case 1 or Case 2 above applies (depending on whether $k > q$ or not) and implies that this P_4 along with two vertices, say, a and b, participate in a domino (see Figure 9). From now on, this case is identical to Case 2(a).

(b) $v_k v_q \notin E(G)$: Then, if $v_k v_j \in E(G)$, the vertices $v_i, v_p, v_q, v_r, v_j, v_k$ induce a domino as shown in Figure 8. Suppose that $v_k v_j \notin E(G)$. Lemma 4.1 applies on v_r, v_q, v_k: if $q < k$, it implies that the rightmost vertex in the LexBFS ordering that is adjacent to v_q and non-adjacent to v_r is also adjacent to v_k; if $q > k$, it implies that the rightmost vertex in the LexBFS ordering that is adjacent to v_k and non-adjacent to v_r is also adjacent to v_q. In either case, this rightmost vertex, say, v_ℓ, is adjacent to both v_q and v_k, and non-adjacent to v_r. From now on, this case is identical to Case 2(b).

4. *$r < q < p$:* Lemma 4.1 applies on v_i, v_q, v_p and it implies that the rightmost vertex, say, v_j, in the LexBFS ordering that is adjacent to v_q and non-adjacent to v_i is also adjacent to v_p (see Figure 7(d)). Then, $v_j v_r \notin E(G)$, otherwise the vertices v_i, v_p, v_q, v_r, v_j would induce a house in G. Property P1 on v_i, v_r, v_p implies that there exists a vertex v_k such that $k > p$, $v_k v_r \in E(G)$, and $v_k v_i \notin E(G)$. Then, $v_k v_p \notin E(G)$, otherwise the vertices v_i, v_p, v_q, v_r, v_k would induce a house or a C_5 depending on whether v_k is adjacent to v_q or not. We distinguish the following two cases:

(a) $v_k v_q \in E(G)$: Then, G_i contains the P_4 $v_p v_i v_q v_k$, where $i < q < p < k$; Case 1 above applies and implies that this P_4 along with two vertices, say, a and b, participate in a domino (see Figure 9). Again, from now on, this case is identical to Case 2(a).

(b) $v_k v_q \notin E(G)$: Then, if $v_k v_j \in E(G)$, the vertices $v_i, v_p, v_q, v_r, v_j, v_k$ induce a domino as shown in Figure 8. Suppose that $v_k v_j \notin E(G)$. Lemma 4.1 applies on v_r, v_q, v_k and implies that the rightmost vertex, say, v_ℓ, in the LexBFS ordering that is adjacent to v_q and non-adjacent to v_r is also adjacent to v_k. This case too, from now on, proceeds as Case 2(b).

In all cases, we obtained a domino as shown in Figure 8. ∎

Time and Space Complexity. Let n and m be the number of vertices and edges of the input graph G. Since we process the vertices v_i in the order they are visited by LexBFS starting with vertex v_{n-5}, we need a representation of the subgraph $G[\{v_{n-5}, v_{n-4}, \ldots, v_n\}]$ and we need to be able to construct efficiently a representation of the subgraph $G[\{v_i, v_{i+1}, \ldots, v_n\}]$ from a similar representation of $G[\{v_{i+1}, v_{i+2}, \ldots, v_n\}]$. The former can be easily done in $O(n+m)$ time and space from a copy of the adjacency-list representation of the graph G from which we remove all unnecessary records and lists. The latter can be done as follows: we traverse the adjacency list of vertex v_i for the graph G; we select among v_i's neighbors (in G) those that belong to $\{v_{i+1}, v_{i+2}, \ldots, v_n\}$ and we construct from them the adjacency list of v_i for $G[\{v_i, v_{i+1}, \ldots, v_n\}]$; for each neighbor u of v_i included in the adjacency list of v_i, we add a record storing v_i in the adjacency list of u. Thus, obtaining $G[\{v_i, v_{i+1}, \ldots, v_n\}]$ from $G[\{v_{i+1}, v_{i+2}, \ldots, v_n\}]$ takes time proportional to the degree $deg(v_i)$ of vertex v_i in G; as a result, the construction of the subgraphs for all the vertices v_i takes $O(n+m)$ time and space. Then, checking whether a vertex v_i is semi-simplicial in $G[\{v_i, v_{i+1}, \ldots, v_n\}]$ is done as described in Section 4.1 in

$O(n+m)$ time and space by taking advantage of Theorem 4.2. Thus, the processing of all the vertices for semi-simpliciality takes a total of $O(nm)$ time and $O(n+m)$ space, just as it was the case in the original version of Algorithm Recognize-HHD-free.

Let us now focus on the time and space complexity of Steps 3.1-3.3 which are executed exactly once when and if a vertex v_i is found not to be semi-simplicial in the graph $G[\{v_i, v_{i+1}, \ldots, v_n\}]$. Step 3.1 is executed as follows: the way to test the semi-simpliciality of the vertex v_i in the graph $G[\{v_i, v_{i+1}, \ldots, v_n\}]$ (which is described in Section 4.1) implies that, because v_i has been found not to be semi-simplicial, we have found a vertex z and a co-component C of the subgraph of $G[\{v_i, v_{i+1}, \ldots, v_n\}]$ induced by the neighbors of v_i such that z is adjacent to some but not all the vertices of C; we partition the set C into a set A of neighbors and a set B of non-neighbors of z (clearly, $A \neq \emptyset$ and $B \neq \emptyset$); then, for each vertex in A, in turn, we mark its neighbors in an auxiliary array M of size n and check whether there exists a vertex in B which is not marked in M; as soon as such a vertex is found, we stop; since the set $A \cup B$ forms a co-component of $G[N(v_i) \cap \{v_{i+1}, \ldots, v_n\}]$, we are sure to find vertices $x \in B$ and $y \in A$ such that $xy \notin E(G)$, and the four vertices x, y, z, v_i induce a P_4 xv_iyz in $G[\{v_i, v_{i+1}, \ldots, v_n\}]$. The above description implies that for any vertex $x' \in A$ processed before x, it holds that $B \subseteq N(x')$ and thus the processing of x' takes $O(deg(x'))$ time; in turn, the processing of x takes $O(deg(x)+1)$ time. Thus, Step 3.1 can be completed in $O(n+m)$ time and space. Step 3.2 takes $O(n)$ time: we need to initialize the arrays and to traverse the adjacency lists of vertices x, z, and v_i in $G[\{v_i, v_{i+1}, \ldots, v_n\}]$. Finally, Step 3.3 can be executed in $O\big(\sum_{j=i+1}^{n}(deg(v_j)+1)\big) = O(n+m)$ time and $O(1)$ space; due to the auxiliary arrays, adjacency testing can be done in $O(1)$ time per test. Thus, we have the following theorem.

Theorem 4.4. *The Algorithm Recognize-HHD-free can be easily augmented to provide a certificate whenever it decides that the input graph G is not HHD-free; if G has n vertices and m edges, the certificate computation takes $O(n+m)$ time and space.*

5 Recognition of WPO-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 [19].

Theorem 5.1. (Olariu and Randall [19]) *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.* [5] described an $O(nm)$-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 [10]. We next show that we can detect whether the complement $\overline{G}$ of G contains a hole or a house in $O(nm)$ time. Combining these two algorithms, we get an $O(nm)$-time algorithm for recognizing WPO-graphs.

Let G be a graph and let v be an arbitrary vertex of G. We construct the graph $\widehat{G}_v$ from G as follows:

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

Note that in $\overline{G}$ the neighbors of v are the vertices in $M(v)$, i.e., the non-neighbors of v in G, and the non-neighbors are the vertices in $N(v)$. Thus, the graph $\widehat{G}_v$ is precisely $\overline{G}$ with any edges between vertices in $M(v)$ removed. Then, it is not difficult to see that the following result holds.

Lemma 5.1. *The vertex v belongs to a hole or is the top of a house or a building in $\overline{G}$ if and only if v belongs to a hole in $\widehat{G}_v$.*

Because in $\widehat{G}_v$ there are no edges between vertices adjacent to v, the vertex v cannot be the top of a house or a building. Thus, we can run the Algorithm Not-in-HHB($\widehat{G}_v, v$) and the vertex v belongs to a hole in $\widehat{G}_v$ if and only if the algorithm returns "false." Assuming that the graph G has n vertices and m edges, 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 the vertex v in G; then, the construction of $\widehat{G}_v$ takes $O(m + n\,deg(v))$ time and $O(n\,deg(v))$ space, and the execution of Not-in-HHB($\widehat{G}_v, v$) runs in $O(n+\min\{n\,deg(v)\,\alpha(n), n\,deg(v)+deg(v)\log deg(v)\}) = O(n + n\,deg(v) + deg(v)\log deg(v)) = O(n\,deg(v))$ 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.

Therefore, in light of Lemma 5.1, we have the following result.

Theorem 5.2. *Let G be an undirected graph on n vertices and m edges. Then, it can be determined whether the complement $\overline{G}$ is an HH-free graph in $O(nm)$ time and $O(n^2)$ space.*

From Theorem 5.2 and the result of Eschen *et al.* [5] (i.e., HHP-free graphs can be recognized in $O(nm)$ time and $O(n + m)$ space), we obtain the following theorem.

Theorem 5.3. *Let G be an undirected graph on n vertices and m edges. Then, it can be determined whether G is a WPO-graph in $O(nm)$ time and $O(n^2)$ space.*

6 Concluding Remarks

We have presented recognition algorithms for the classes of HHD-free graphs and WPO-graphs running in $O(\min\{nm\alpha(n), nm + n^2 \log n\})$ and $O(nm)$ time, respectively, where n is the number of vertices and m is the number of edges of the input graph. Our proposed algorithms are simple, use simple data structures, and require $O(n + m)$ and $O(n^2)$ space, respectively. Moreover, we show how our HH-free and HHD-free graph recognition algorithms can be augmented to yield a certificate (a hole, a house, or a domino) whenever they decide that the input graph is not HH-free or HHD-free; in either case, the certificate computation takes $O(n + m)$ time and space.

Figure 11 shows a diagram of class inclusions for a number of classes of perfectly orderable graphs and the currently best time complexities to recognize members of these classes. For definitions of the classes shown, see [2, 6]. In the diagram, there exists an arc from a class $\mathcal{A}$ to a class $\mathcal{B}$ if and only if $\mathcal{B}$ is a proper subset of $\mathcal{A}$. Hence, if any two classes are not connected by an arc, then each of these classes contains graphs not belonging to the other class (there are such sample graphs for each pair of non-linked classes). Figure 11 shows also the depicted classes of graphs partitioned based on the time complexities of the currently fastest recognition algorithms: see [2] and [1, 5, 7, 15, 16, 21].

We leave as an open problem the designing of $O(nm)$-time algorithms for recognizing HH-free and HHD-free graphs; note that an $O(nm)$-time algorithm for recognizing HH-free graphs directly implies an $O(nm)$-time recognition algorithm for HHD-free graphs. Additionally, in light of the $O(nm)$-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.

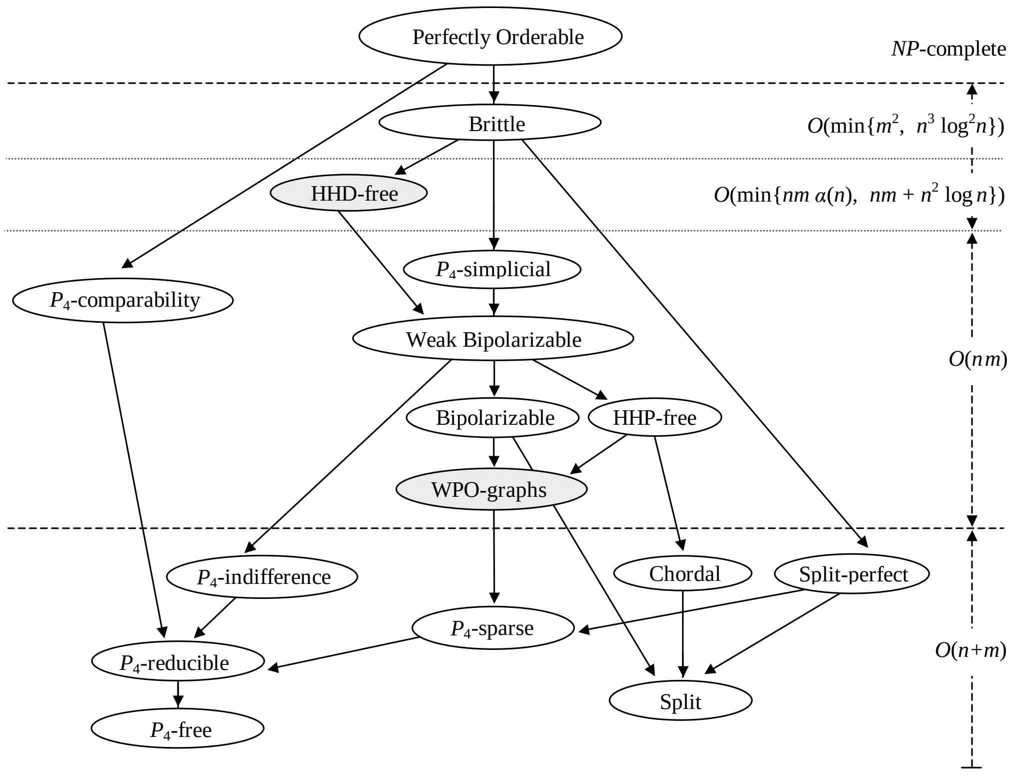

Figure 11: Class inclusions and recognition time complexities.

References

[1] A. Brandstädt and V.B. Le, Split-perfect graphs: Characterizations and algorithmic use, *SIAM J. Discrete Math.* **17**, 341–360, 2004.

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

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

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

[5] 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.

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

[7] M. Habib, C. Paul, and L. Viennot, Linear time recognition of P_4-indifference graphs, *Discrete Math. and Theoret. Comput. Sci.* **4**, 173–178, 2001.

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

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

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

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

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

[13] 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.

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

[15] S.D. Nikolopoulos and L. Palios, Recognizing Bipolarizable and P_4-simplicial graphs, *Proc. 29th Workshop on Graph Theoretic Concepts in Computer Science (WG'03)*, LNCS 2880, 358–369, 2003.

[16] S.D. Nikolopoulos and L. Palios, Algorithms for P_4-comparability graph recognition and acyclic P_4-transitive orientation, to appear in *Algorithmica*, 2004.

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

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

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

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

[21] A.A. Schäffer, Recognizing brittle graphs: remarks on a paper of Hoàng and Khouzam, *Discrete Appl. Math.* **31**, 29–35, 1991.

[22] 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.