Recognizing HHD-free and Welsh-Powell Opposition Graphs*

Stavros D. Nikolopoulos and Leonidas Palios

Department of Computer Science, University of Ioannina
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; this in turn leads to an $O(nm)$-time and $O(n^2)$-space algorithm for recognizing WPO-graphs. The previously best algorithms for recognizing HHD-free and WPO-graphs required $O(n^3)$ time and $O(n^2)$ space.

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 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 [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].

* Research partially funded by the European Union and the Hellenic Ministry of Education through EPEAEK II.

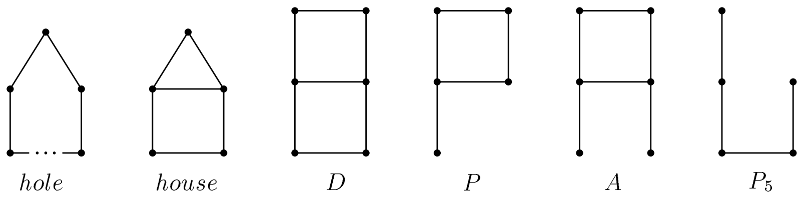

Fig. 1. Some simple graphs.

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 [8], Hoáng and 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 [5]. A subclass 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 [6] 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 **bip*** graphs (see [1]), and hence are perfect graphs [14]. 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 $c \prec d$ [15]. 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 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 [15]) and the use of modular decomposition [11] 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 [16] 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 $O(nm)$-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 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 [10], and by Hoáng and Khouzam [8] enable 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 [16] 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.* [4], our result yields an $O(nm)$-time and $O(n^2)$-space algorithm for recognizing WPO-graphs.

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 $2 \le |B| < |V(G)|$ 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

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

Fig. 2. The perfect elimination ordering testing algorithm.

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 $v \in M(x)$, we use $n(v,x)$ to denote the number of vertices in the set $N(v) \cap N(x)$, i.e., the set of common neighbors of v and x. The *degree* of a vertex x in a graph G, denoted $deg(x)$, is the number of edges incident on x; thus, $deg(x) = |N(x)|$.

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 $(v_1, v_2, \ldots, v_n)$ such that each vertex v_i is simplicial in the graph G_i induced by the vertex set $\{v_i, \ldots, v_n\}$, $1 \leq i \leq n$; 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, 17]. 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 $(v_1, v_2, \ldots, v_n)$ such that each vertex v_i is semi-simplicial in the graph $G_i = G[\{v_i, \ldots, v_n\}]$, $1 \leq i \leq n$. 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, 17]. 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

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

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.

Note: Due to lack of space we have omitted the proofs of several Lemmata and Theorems of this paper; all the proofs can be found in [13].

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[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 $n(y,v) \le n(y',v)$; recall that, $n(y,v)$ is the number of common neighbors of y and v, or, equivalently, the degree of the vertex $y \in M(v)$ in the graph induced by the set $N(v) \cup \{y\}$. 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 $A(y_i) - N(y_i) \subseteq N(v)$.*

[1] A building is a graph on vertices $v_1, v_2, \ldots, v_p$, where $p \ge 6$, and edges $v_1 v_p$, $v_2 v_p$, and $v_i v_{i+1}$ 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 $X \neq \emptyset$ then
6.          $w \leftarrow Next\_Neighbor_{G'_v,\sigma}[u]$;
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";
```

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

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 $z \in M(v)$.*

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.*

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*

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 $y_r \neq y_j$        {$y_i$ and $y_j$ belong to different sets}
7.          then
8.              $Next\_Neighbor_{G'_v,\sigma}[y_r] \leftarrow y_j$;
9.              union the sets to which $y_i$ and $y_j$ belong;
```

Fig. 4. Algorithm for computing the contents of the array $Next\_Neighbor_{G'_v,\sigma}[\,]$.

$v_i, v_{i+1}, \ldots, v_k$. 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 $v_1, v_2, \ldots, v_i, \ldots, v_r$.

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 $Next\_Neighbor_{G'_v,\sigma}[y_r]$, 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 4.

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 $y_i y_j$ 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).*

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, σ, 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 $O(m\alpha(k))$ [3], where $\alpha(\ )$ is a very slowly growing function; if instead we use the linked-list representation, then the time is $O(m + k \log k)$ [3]. 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.*

Applying the Algorithm Non-in-HHB on every vertex of a graph and observing that a building contains a hole, 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.*

4 Recognition of 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.

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 Rec-HHD-free

Input: an undirected graph G on n vertices and m edges.

Output: "true," if G is an HHD-free graph; otherwise, "false."

```
1. if the graph $G$ is not HH-free
   then return("false");
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("false");
4. return("true").
```

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 [5, 17]. 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 [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 $G[\{v_i, \ldots, v_n\}]$, 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.*

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 [16].

Theorem 5.1. (Olariu and Randall [16]) *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 $O(nm)$-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 [8]. 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:

$$\begin{aligned} \circ\ V(\widehat{G}_v) &= V(G) \\ \circ\ E(\widehat{G}_v) &= \{\, vy \mid y \in M(v) \,\} \\ &\quad \cup\ \{\, xy \mid x \in N(v),\ y \in M(v),\ \text{and } xy \notin E(G) \,\} \\ &\quad \cup\ \{\, xx' \mid x, x' \in N(v) \text{ and } xx' \notin E(G) \,\} \end{aligned}$$

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 $O(n\,deg(v) + deg^2(v)) = O(n\,deg(v))$ 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 $O(n + n\,deg(v) + deg(v) \log deg(v)) = O(n\,deg(v))$ 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.* [4] (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, our HH-free and HHD-free graph recognition algorithms can be easily 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 [13].

We leave as an open problem the designing of $O(nm)$-time algorithms for recognizing HHD-free graphs. 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.

References

1. A. Brandstadt, 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.D. Nikolopoulos and L. Palios, Recognizing HHD-free and Welsh-Powell opposition graphs, Technical Report TR-16-04, Dept. of Computer Science, University of Ioannina, 2004.
14. S. Olariu, All variations on perfectly orderable graphs, *J. Combin. Theory* Ser. B **45**, 150–159, 1988.
15. S. Olariu, Weak bipolarizable graphs, *Discrete Math.* **74**, 159–171, 1989.
16. S. Olariu and J. Randall, Welsh-Powell opposition graphs, *Inform. Process. Lett.* **31**, 43–46, 1989.
17. D.J. Rose, R.E. Tarjan, and G.S. Lueker, Algorithmic aspects of vertex elimination on graphs, *SIAM J. Comput.* **5**, 266–283, 1976.
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.