Recognition of some perfectly orderable graph classes

Elaine M. Eschen\textsuperscript{a,}\textsuperscript{*}, Julie L. Johnson\textsuperscript{b,1}, Jeremy P. Spinrad\textsuperscript{b,1}, R. Sritharan\textsuperscript{c,2}

\textsuperscript{a}Lane Department of Computer Science and Electrical Engineering, West Virginia University, P.O. Box 6109, Morgantown, WV 26506, USA
\textsuperscript{b}Computer Science Department, Vanderbilt University, Nashville, TN 37235, USA
\textsuperscript{c}Computer Science Department, The University of Dayton, Dayton, OH 45469-2160, USA

Received 19 June 2001; received in revised form 3 April 2002; accepted 3 June 2002

Abstract

This paper presents new algorithms for recognizing several classes of perfectly orderable graphs. Bipolarizable and $P_4$-simplicial graphs are recognized in $O(n^{3.76})$ time, improving the previous bounds of $O(n^4)$ and $O(n^5)$, respectively. Brittle and semi-simplicial graphs are recognized in $O(n^3)$ time using a randomized algorithm, and $O(n^3 \log^2 n)$ time if a deterministic algorithm is required. The best previous time bound for recognizing these classes of graphs is $O(m^2)$. Welsh–Powell opposition graphs are recognized in $O(n^3)$ time, improving the previous bound of $O(n^4)$. HHP-free graphs and maxibrittle graphs are recognized in $O(mn)$ and $O(n^{3.76})$ time, respectively.

© 2003 Elsevier Science B.V. All rights reserved.

Keywords: Perfectly orderable graph; Brittle graph; Recognition algorithm

1. Introduction

Chvátal [2] introduced the class of perfectly orderable graphs, for which there is an ordering of the vertices (called a perfect ordering) such that for all induced subgraphs, a greedy coloring that follows the ordering uses the minimum number of colors.
While a characterization of the class of perfectly orderable graphs via forbidden minimal induced subgraphs remains elusive, Middendorf and Pfeifer [27] have shown that recognition of perfectly orderable graphs is NP-complete. This has motivated researchers to study subclasses of perfectly orderable graphs such as brittle graphs [3,19], HHD-free graphs [19,23], and several others [4,5,15,18,21,22,32]. This paper improves the time complexity of recognizing several perfectly orderable graph classes that were introduced because they can be recognized in polynomial time and easily shown to have perfect orderings.

The graphs we consider are simple. For a graph $G = (V,E)$, $n$ is the cardinality of $V$ and $m$ is the cardinality of $E$. For $x \in V$, the (open) neighborhood of $x$, denoted $N(x)$, is the set of vertices that are adjacent to $x$. The closed neighborhood of $x$ is the set $N(x) \cup \{x\}$. We use $M(x)$ to denote the set $V - N(x) - \{x\}$. The complement of a graph $G$ is denoted by $\bar{G}$.

$P_k$ denotes a chordless path on $k$ vertices. We list the vertices of a $P_k$ in the natural order within square brackets. In a $P_4$ $[a,b,c,d]$, we refer to the middle vertices $b$ and $c$ as vertices that are end-$P_4$, and the end vertices $a$ and $d$ as vertices that are end-$P_4$. If a vertex $x$ is not in the middle of any $P_4$ in a graph, we say $x$ is not mid-$P_4$. Similarly, if a vertex $x$ is not at the end of any $P_4$ in a graph, we say $x$ is not end-$P_4$. Note if $[a,b,c,d]$ is a $P_4$ in graph $G$, then $[c,a,d,b]$ is a $P_4$ in the complement of $G$. Thus, a vertex is mid-$P_4$ in a graph $G$ if and only if it is end-$P_4$ in $\bar{G}$. We will use this fact frequently.

A graph is chordal if it does not contain an induced cycle on four or more vertices. A vertex of a graph is simplicial if its neighborhood induces a clique (or equivalently, if it is not the middle vertex of any $P_4$). A vertex of a graph that is simplicial in the complement of the graph is said to be co-simplicial. For an ordering $(v_1,v_2,\ldots,v_n)$ of the vertices of a graph $G$, let $G_i$ be the subgraph of $G$ induced by $\{v_i,\ldots,v_n\}$, for $1 \leq i \leq n$. A graph $G$ has a perfect (or simplicial) 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 $G_i$. It is well known that a graph has a perfect elimination ordering if and only if it is chordal [10,11,35].

A superclass of the chordal graphs known as good graphs [20] (also known as quasi-triangulated graphs [13]) is defined by successive deletion of simplicial or co-simplicial vertices. A graph is good if and only if its vertices can be linearly ordered $(v_1,v_2,\ldots,v_n)$ such that each vertex $v_i$ is either simplicial or co-simplicial in $G_i$.

Jamison and Olariu [25] generalized the notion of a simplicial vertex. A vertex is semi-simplicial in a graph $G$ if it is not mid-$P_4$ in $G$. A graph $G$ has a semi-perfect (or semi-simplicial) 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 $G_i$. A graph is semi-simplicial [17] if and only if it has a semi-perfect elimination ordering. This characterization yields an $O(m^2)$ time recognition algorithm [17].

The class of brittle graphs was defined by Chvátal [3] as follows. A vertex $x$ is soft in a graph $G$ if $x$ is either not mid-$P_4$ or not end-$P_4$ in $G$. A graph is brittle if and only if its vertices can be linearly ordered $(v_1,v_2,\ldots,v_n)$ such that each vertex $v_i$ is soft in $G_i$. A vertex of a graph that is semi-simplicial in the complement of the graph is said to be co-semi-simplicial. Since a vertex that is not end-$P_4$ in $G$ is not mid-$P_4$,
in \( \tilde{G} \), brittle graphs can be viewed as a natural generalization of good graphs. A graph is brittle if and only if its vertices can be linearly ordered \((v_1, v_2, \ldots, v_n)\) such that each vertex \(v_i\) is either semi-simplicial or co-semi-simplicial in \(G_i\).

A graph is weakly chordal if it does not contain an induced cycle or the complement of an induced cycle on five or more vertices. It is not hard to see that every brittle graph is weakly chordal. This can be refined somewhat; brittle graphs are a subclass of the domination graphs [9], a subclass of weakly chordal graphs in which every induced subgraph has a vertex whose neighborhood is contained in the closed neighborhood of another vertex. A generalization of brittle graphs called quasi-brittle graphs was studied in [32], and a subclass of brittle graphs called \(P_4\)-laden was studied in [12].

A number of characterizations of brittle graphs are given in [19]; it is noted that one of these characterizations gives an \(O(n^3 m)\) time recognition algorithm. Schäffer [37] dealt specifically with the recognition problem for brittle graphs, and gave an \(O(m^2)\) time recognition algorithm derived from the definition.

In this paper we present two algorithms for recognizing brittle graphs by direct application of the definition, i.e. repeated search for soft vertices. The algorithms are conceptually simple, but each relies on a complex algorithm for a well-known problem as a subroutine. First, we present an algorithm that requires \(O(n)\) adjacency matrix multiplications as its bottleneck step. Since the current best time bound for matrix multiplication is \(O(n^{2.376})\) [6], this yields an \(O(n^{3.376})\) time bound, which is better than Schäffer’s bound for dense graphs. We then present an algorithm that uses modular (or substitution) decomposition. When used together with known algorithms for modular decomposition and on-line maintenance of spanning trees, this approach yields an \(O(n^3 \log^2 n)\) time deterministic or \(O(n^3)\) time randomized recognition algorithm for brittle graphs.

The algorithm presented here for recognizing brittle graphs that uses matrix multiplication maintains, for each vertex \(x\), a count of the number of \(P_4\)s containing \(x\) as a midpoint, and thus, can also be used to recognize semi-simplicial graphs in \(O(n^{3.376})\) time. Our algorithm for recognizing brittle graphs that is based on modular decomposition can also be used to recognize semi-simplicial graphs; improving the time bound to \(O(n^3 \log^2 n)\) for a deterministic algorithm and \(O(n^3)\) for a randomized algorithm.

An obstruction in a graph with a vertex ordering \(<\) is a \(P_4\) \([a, b, c, d]\) such that \(a < b\) and \(d < c\). It is easy to see that if \(<\) is a perfect order on a graph \(G\), then \(G\) contains no obstruction (since the chromatic number of a \(P_4\) is two, but a greedy coloring of an obstruction uses three colors). Chvátal [2] proved that an ordering of the vertices of a graph is perfect if and only if it contains no obstruction. Using this characterization, it is easy to see that brittle graphs are perfectly orderable. An ordered list \(L\) of vertices can be formed by adding soft vertices as they are eliminated to two sublists of \(L\). If we place the vertices that are not end-\(P_4\) on a sublist growing inwards from the front of \(L\) and the vertices that are not mid-\(P_4\) on a sublist growing inwards from the end of \(L\), we always obtain a perfect ordering. To see this, suppose the resulting vertex order has an obstruction. It is easy to argue that no vertex of the obstruction could have been the first among the vertices of the obstruction to be eliminated.

An ordering \(<\) of the vertices of an undirected graph \(G\) corresponds to an acyclic orientation of \(G\) such that \(x \rightarrow y\) if and only if \(xy\) is an edge of \(G\) and \(x < y\). Also,
given an acyclic directed graph $H$, an ordering $< \, \text{of} \, \text{the} \, \text{vertices} \, \text{of} \, \text{the} \, \text{underlying} \, \text{undirected} \, \text{graph} \, G \, \text{can} \, \text{be} \, \text{constructed} \, \text{such} \, \text{that} \, H \, \text{is} \, \text{the} \, \text{corresponding} \, \text{orientation} \, \text{of} \, G$. Thus, Chvátal’s characterization of perfectly orderable graphs can be restated as: a graph is perfectly orderable if and only if it admits an acyclic orientation that does not contain any $P_4 \, \{a,b,c,d\}$ with edges $ab$ and $cd$ oriented $a \rightarrow b$ and $c \leftarrow d$. This led to a number of definitions of graph classes, based on the types of orientations of $P_4$s that are allowed.

The class of bipolarizable graphs [16] (also called Raspail graphs in [21]) is defined by requiring an acyclic orientation of edges such that for every $P_4 \, \{a,b,c,d\}$, edges $ab$ and $cd$ are oriented $a \leftarrow b$ and $c \rightarrow d$, i.e. the end edges of the $P_4$ point to the end vertices. A characterization of bipolarizable graphs in terms of an infinite family of forbidden subgraphs is given in [16]. This class can be recognized in $O(n^3)$ time by examining all $P_4$s in $G$, assigning necessary orientations to edges, and testing for acyclicity. The algorithm we present for recognizing brittle graphs that uses matrix multiplication finds all end edges of $P_4$s, and thus, can also be used to recognize bipolarizable graphs in $O(n^3 \cdot 376)$ time.

Another graph class based on allowable orientations of $P_4$s are the $P_4$-simplicial graphs [21]. This class requires an acyclic orientation of edges such that every $P_4 \, \{a,b,c,d\}$ has one of two orientations: either the end edges of the $P_4$ point to the end vertices, or all edges of the $P_4$ are oriented in the same direction. Note that the vertex ordering corresponding to such an edge orientation is simplicial (perfect) when restricted to any $P_4$ in the graph. $P_4$-simplicial graphs were shown in [21] to be a special subclass of brittle graphs called strongly brittle graphs. A graph $G$ is strongly brittle if and only if its vertices can be linearly ordered $(v_1, v_2, \ldots, v_n)$ such that each vertex $v_i$ is not the midpoint of a $P_4 \, \{a,b,v_i,c\}$ in $G$, where $b$ and $c$ are in $G_i$. This characterization directly yields an $O(n^5)$ recognition algorithm for $P_4$-simplicial graphs. We use this characterization, matrix multiplication, and the techniques presented here for brittle graphs to recognize $P_4$-simplicial graphs in $O(n^3 \cdot 376)$ time.

Other algorithms presented in this paper make use of known characterizations for the graph classes via forbidden minimal induced subgraphs, as well as modular decomposition, matrix multiplication, and some properties of the recognition algorithm for (house, hole)-free (HH-free) graphs given in [23]. Some of the specific graphs referred to in this paper are presented in Fig. 1. A hole in a graph is an induced cycle on five or more vertices. A house is the complement of a $P_5$.

A graph is HHP-free if it contains no hole and no induced house or “P”. We show that HHP-free graphs can be recognized in $O(mn)$ time, and use this as a subroutine to solve other problems. A Welsh–Powell ordering (WP-ordering) for a graph is a nonincreasing ordering of the vertices according to their degrees [39]. A graph is an opposition graph if its vertices can be ordered so that there is no $P_4 \, \{a,b,c,d\}$ with $a \leftarrow b$ and $c \rightarrow d$. Or equivalently, a graph is an opposition graph if it admits an acyclic orientation of edges that does not contain any $P_4 \, \{a,b,c,d\}$ with edges $ab$ and $cd$ oriented $a \rightarrow b$ and $c \leftarrow d$ [30]. A graph is a Welsh–Powell opposition graph (WPO) if the graph is an opposition graph for every WP-ordering of the graph. Olariu and Randall [33] characterized WPO graphs and showed that they and their complements are brittle. They also gave an $O(n^4)$ algorithm to recognize WPO graphs;
we present an $O(n^3)$ algorithm for the problem. We note that the time complexity of recognizing opposition graphs is currently an open problem. It is also unknown if there is an opposition graph that is not perfectly orderable [1].

Preissmann et al. [34] introduced the class of maxibrittle graphs; a graph is maxibrittle if in every induced subgraph of the graph, every vertex of maximum degree is not end-$P_4$, and every vertex of minimum degree is not mid-$P_4$. It follows from the definition that every maxibrittle graph is brittle. We present an $O(n^{3.376})$ time algorithm to recognize maxibrittle graphs.

The remaining sections of this paper provide the details of these recognition algorithms. In Section 2, we describe recognition algorithms that rely on $O(n)$ matrix multiplication operations. These algorithms recognize brittle, semi-simplicial, $P_4$-simplicial and bipolarizable graphs. Section 3 is a brief description of modular decomposition, which is used in other of our recognition algorithms. In Section 4, we give recognition algorithms for semi-simplicial and brittle graphs that are based on a characterization of these classes first introduced by Hoàng and Khouzam. These algorithms are faster than those described in Section 2, and we explain why both sets of algorithms are useful. In Sections 5 and 6 we give recognition algorithms for HHP-free and Welsh–Powell opposition graphs. In Section 7, we describe an algorithm for recognizing maxibrittle graphs. In the conclusions section we discuss some open problems.

2. Recognition via matrix multiplication

The algorithms described in this section recognize brittle, semi-simplicial, $P_4$-simplicial and bipolarizable graphs using $O(n)$ matrix multiplications. Although this approach yields a slower time bound for recognizing brittle and semi-simplicial graphs than the algorithm that is presented later, we include it for the following reasons. First, matrix multiplication is a standard operation and its use gives a conceptually simple algorithm with time bound $o(n^4)$, which is better than the previous best bound for dense graphs [37]. Second, the algorithm actually does more than recognize brittle graphs; it allows us to maintain a count of the number of $P_4$s involving each vertex as vertices are deleted. Since many graph classes are defined on the basis of vertices being in
“few” $P_4$s (see [1] for a summary), this algorithm may be useful for recognizing other classes as well.

2.1. Brittle graphs

Recall that a graph is brittle if all its vertices can be eliminated by successive deletion of soft vertices; a vertex is soft if it is either not mid-$P_4$ or not end-$P_4$. To recognize brittle graphs, we construct such a vertex elimination ordering for the given graph, if possible. We maintain for each vertex $x$ a count of the number of $P_4$s containing $x$ as an endpoint, and the number of $P_4$s containing $x$ as a midpoint. Clearly, we can test whether vertices are soft given these counts. Note that the choice of soft vertex to delete is arbitrary since the deletion of a vertex cannot cause another vertex to become the midpoint or endpoint of a $P_4$.

We will describe two phases of the computation: an initial phase, calculating the $P_4$ counts in the original graph, and an update phase, calculating the changes necessary when a soft vertex is deleted. Since vertices that are midpoints of $P_4$s are endpoints of $P_4$s in the complement and vice versa, procedures that maintain counts are run on the complement graph to obtain counts for a vertex in the opposite role.

To compute the number of $P_4$s having vertex $x$ as an endpoint, we create a graph $H_x(G)$ from $G$ as follows. Vertices of $H_x(G)$ consist of one vertex $y_0$ for each neighbor $y$ of $x$ in $G$, and two vertices $w_1$ and $w_2$ for each nonneighbor $w$ of $x$ in $G$. Add edge $y_0w_1$ if and only if $y$ and $w$ are adjacent in $G$. Let $w$ and $z$ be nonneighbors of $x$ in $G$. Add edges $w_1z_2$ and $z_1w_2$ if and only if $w$ and $z$ are adjacent in $G$. A $P_3[1, 2, 3]$ in $H_x(G)$ corresponds to a $P_4$ in $G$ with endpoint $x$ if and only if $y$ and $z$ are not adjacent in $G$.

It is well known that if $A_G$ is the adjacency matrix of a graph $G$, $A_G^2[u, w]$ is the number of paths of length 2 from vertex $u$ to vertex $w$. Thus, to compute the number of $P_4$s in $G$ with $x$ as an endpoint, we first compute the square of $A_{H_x(G)}$, where $A_{H_x(G)}$ is the adjacency matrix for $H_x(G)$. We then sum over all neighbors $y$ of $x$ the values $A_{H_x(G)}^2[y_0, z_2]$ such that $y$ and $z$ are not adjacent in $G$.

The $P_4$s in $G$ with $x$ as midpoint can be counted by constructing $H_x(\tilde{G})$ from $\tilde{G}$ and counting the number of $P_4$s that have $x$ as an endpoint by the same procedure as above. We thus can perform the initial phase of counting, for all $x$, the number of $P_4$s with $x$ as an endpoint and the number of $P_4$s with $x$ as a midpoint by performing $O(n)$ multiplications of adjacency matrices of graphs with $O(n)$ vertices.

We now describe three procedures that update the $P_4$ counts of other vertices when vertex $v$ is removed.

If $v$ is to be deleted because it is not mid-$P_4$, we find all $P_4$s where $v$ is an endpoint in order to lower the midpoint and endpoint counts of the other vertices in these $P_4$s. Consider the $P_4[v, a, b, c]$. Procedure 1 updates the endpoint count of vertex $c$, Procedure 2 updates the midpoint count of vertex $a$, and Procedure 3 updates the midpoint count of vertex $b$.

If $v$ is to be deleted because it is not end-$P_4$, we find all $P_4$s where $v$ is a midpoint in order to lower the midpoint and endpoint counts of the other vertices in these $P_4$s. A
A graph contains the $P_4[b,v,c,a]$ if and only if its complement contains the $P_4[v,a,b,c]$. Thus, working in the complement, Procedure 1 updates the midpoint count of vertex $c$. Procedure 2 updates the endpoint count of vertex $a$, and Procedure 3 updates the endpoint count of vertex $b$.

Assume that $v= v_i$ is to be eliminated because $v$ is not mid-$P_4$. (If $v=v_i$ is removed because $v$ is not end-$P_4$, we perform the same procedures on $G_i$.) Construct the graph $H_v(G_i)$ from $G_i$ as was done in the initial phase, and compute $A^2_{H_v(G_i)}$.

**Procedure 1.** For each nonneighbor $z$ of $v$ in $G_i$, find the number of $P_4$s with $v$ and $z$ as endpoints. This is simply the sum over all nonneighbors $y$ of $z$ of the values $A^2_{H_v(G_i)}[y_0,z_2]$. This is subtracted from the current count of $P_4$s with $z$ as endpoint.

**Procedure 2.** For each neighbor $y$ of $v$ in $G_i$, find the number of $P_4$s with $v$ as endpoint and $y$ as a midpoint. This is the sum over all nonneighbors $z$ of $y$ of the values $A^2_{H_v(G_i)}[y_0,z_2]$. 

**Procedure 3.** For each nonneighbor $z$ of $v$ in $G_i$, find the number of $P_4$s with $v$ as endpoint and $z$ as midpoint. This can be done by making a small modification of the graph $H_v(G_i)$. Let $H'_v(G_i)$ be the graph with the same vertex set as $H_v(G_i)$ and edges defined as follows. Add edge $y_0w_1$ if and only if $y$ and $w$ are not adjacent in $G_i$, and add edges $w_1z_2$ and $z_1w_2$ if and only if $w$ and $z$ are adjacent in $G_i$. There is a $P_4[v,y,z,w]$ in $G$ if and only if there is a $P_3[y_0,w_1,z_2]$ in $H'_v(G_i)$ such that $y$ and $z$ are adjacent in $G_i$. Therefore, we can count the number of $P_4$s with $v$ as endpoint and a nonneighbor $z$ of $v$ as midpoint by summing the values $A^2_{H'_v(G_i)}[y_0,z_2]$ over all vertices $y$ that are adjacent to both $v$ and $z$.

Thus, we can perform all updates caused by deletion of a vertex $v$ in $O(n^2)$ time, plus the time needed to square the adjacency matrix of a graph with $O(n)$ vertices. Repeating this until all vertices have been eliminated and adding the work done in the initial phase, this gives a total time bound of $O(n^{3.376})$ for recognizing brittle graphs.

### 2.2. Semi-simplicial graphs

The algorithm of Section 2.1 can also be used to recognize semi-simplicial graphs in the same time bound. Recall that a graph is semi-simplicial if all its vertices can be eliminated by successive deletion of semi-simplicial vertices; a vertex is semi-simplicial if it is not mid-$P_4$. To recognize semi-simplicial graphs, we construct such a vertex elimination ordering for the given graph, if possible. We need to maintain for each vertex $x$ a count of the number of $P_4$s containing $x$ as a midpoint. We use an initial phase and an update phase as described above. First, we calculate the number of $P_4$s containing $x$ as a midpoint in the original graph $G$ for each vertex $x$ in $G$. Then we continue to update these counts whenever a vertex is deleted. This yields a time bound of $O(n^{3.376})$ for recognizing semi-simplicial graphs.
2.3. $P_4$-simplicial graphs

To recognize $P_4$-simplicial graphs we use their characterization as strongly brittle graphs due to Hoàn and Reed [21]. Recall that a graph $G$ is strongly brittle (equivalently, $P_4$-simplicial) if and only if its vertices can be linearly ordered $(v_1, v_2, \ldots, v_n)$ such that each vertex $v_i$ is not the midpoint of a $P_4$ $[b, v_i, c, a]$ in $G$, where $b$ and $c$ are in $G$. We create an algorithm similar to that for brittle graphs, successively deleting vertices when they are no longer mid-$P_4$ in the sense given in the definition of the class. The difference being that in the $P_4$-simplicial case a vertex is considered the midpoint of a $P_4$ whether or not its nonneighbor in the $P_4$ is in $G$. We will use Procedures 1 and 2 of the brittle graph recognition algorithm with modifications to recognize $P_4$-simplicial graphs.

Suppose vertex $v = v_i$ is deleted because it is not the midpoint of any $P_4$ in the sense given in the definition of the class. There are two cases to consider where the deletion of $v$ affects the midpoint counts of other vertices: (i) there is a $P_4$ $[v, a, b, c]$ in $G$ where $a$ and $b$ are in $G_i$, and $c$ may or may not be in $G_i$, and (ii) there is a $P_4$ $[b, v, c, a]$ in $G$ where $a$ and $c$ are in $G_i$, and $b$ is not in $G_i$.

For each vertex $x$, let $H_x(G)$ and $H_{x}(G)$ be the graphs constructed for the initial counting procedure as described in the brittle graph recognition algorithm.

Consider a $P_4$ $[v, a, b, c]$ where $a$ and $b$ are in $G_i$. Deleting vertex $v$ will lower the midpoint count of vertex $a$, but not that of $b$. In the recognition algorithm for brittle graphs we run Procedure 2 on $G_i$ (constructing $H_i(G_i)$) to update the midpoint count of vertices such as $a$ when $v$ is deleted because it is not mid-$P_4$. To find the number of $P_4$s that have $v$ as an endpoint, middle vertices $a$ and $b$ in $G_i$, and $c$ as an endpoint (whether $c$ is in $G_i$ or not), we alter Procedure 2 in the following way.

Let $H''_v(G)$ be the subgraph of $H_i(G)$ induced by the following vertices. Include in $H''_v(G)$ all vertices $y_0$ such that $y$ is in $G_i$. These are neighbors of $v$, like vertex $a$ in our $P_4_i$. Include in $H''_v(G)$ all vertices $z_1$ such that $z$ is in $G_i$. These correspond to nonneighbors of $v$ that have not yet been deleted, like $b$ in our $P_4_i$. Include in $H''_v(G)$ all the $z_2$ vertices of $H_i(G)$. These are vertices like $c$ in our $P_4_i$ that may or may not be in $G_i$. Any $P_3$ $[y_0, w_1, z_2]$ in $H''_v(G)$, where $y$ is not adjacent to $z$ in $G_i$, corresponds to the $P_4$ $[v, y, w, z]$ in $G$, where $y$ and $w$ are in $G_i$ and $z$ may or may not be in $G_i$. The sum over all nonneighbors $z$ of $y$ of the values $A'_{H''(G)}[y_0, z_2]$ is the number of such $P_4$s. Subtract this from the midpoint count of $y$.

Now consider a $P_4$ $[b, v, c, a]$ where $a$ and $c$ are in $G_i$, and $b$ is not in $G_i$. We now focus on the midpoint count of $c$ when vertex $v$ is deleted. Note that the deletion of $b$ did not change the midpoint count of $c$. Recall that a graph contains the $P_4$ $[b, v, c, a]$ if and only if $[v, a, b, c]$ is a $P_4$ in the complement of the graph. In the recognition algorithm for brittle graphs we run Procedure 1 on $G_i$ (constructing $H_i(G_i)$) to update the midpoint count of vertex $c$ when $v$ is deleted because it is not end-$P_4$. Procedure 1 with modifications serves our needs here. We alter Procedure 1 as follows.

Let $H''_c(G)$ be the subgraph of $H_i(G)$ induced by the following vertices. Include in $H''_c(G)$ all vertices $y_0$ such that $y$ is in $G_i$. These are the neighbors of $v$ in $G$ that have not yet been deleted, like $a$ in our $P_4_i$. Include in $H''_c(G)$ all the $z_1$ vertices of $H_i(G)$ such that $z$ is not in $G_i$. These are the nonneighbors of $v$ in $G$ that have...
been deleted, like \( b \) in our \( P_4 \) in \( \tilde{G} \). Include in \( H''(\tilde{G}) \) all vertices \( z_2 \) such that \( z \) is in \( G \). Any \( P_4 \) \([y_0,w_1,z_2]\) in \( H''(\tilde{G}) \) where \( y \) and \( z \) are not adjacent in \( \tilde{G} \) corresponds to the \( P_4 \) \([v,y,w,z]\) in \( \tilde{G} \) and the \( P_4 \) \([w,v,z,y]\) in \( G \). The sum of over all nonneighbors \( z \) of \( y \) of the values \( A^2_{H''(\tilde{G})}[y_0,z_2] \) is the count of \( P_4 \)s where \( z \) is a midpoint and a neighbor of \( v \). Subtract this number from the midpoint count of \( z \).

We have demonstrated that we can find the initial midpoint counts for all vertices in a given graph \( G \), and then update the counts in such a way as to recognize \( P_4 \)-simplicial graphs in \( O(n^3) \) time.

2.4. Bipolarizable graphs

The brittle graph recognition algorithm can be modified to recognize bipolarizable graphs in the same time bound. An edge \( xy \) is part of some \( P_4 \) \([x,y,w,z]\) (and thus, must be oriented from \( y \) to \( x \)) if and only if there is some nonneighbor \( z \) of \( x \) and \( y \) such that \( A^2_{H''(\tilde{G})}[y_0,z_2] \) is nonzero. Thus, all edges on the ends of \( P_4 \)s can be identified and oriented using \( O(n) \) multiplications of adjacency matrices of graphs with \( O(n) \) vertices. Note that for all these edges the orientation is forced by the definition of bipolarizable graphs, and that if all edges are oriented without contradictions (i.e. no edge is forced to be oriented both from \( x \) to \( y \) and from \( y \) to \( x \)), then every \( P_4 \) has a valid orientation no matter how the remaining edges are oriented.

After orienting all outer edges of \( P_4 \)s without contradictions, we check that the subgraph \( \tilde{G} \) of \( G \) with the same vertex set as \( G \) and edge set equal to the oriented edges is acyclic. If \( \tilde{G} \) has a cycle, then the input graph is not bipolarizable. Now to complete the acyclic orientation of \( G \), we take any topological sort of \( \tilde{G} \) and direct all remaining edges of \( G \) from lower to higher index with respect to this ordering. This algorithm gives a time bound of \( O(n^{3.376}) \) for recognizing bipolarizable graphs.

3. Modular decomposition

A module in a graph is a subset \( M \) of vertices that is “indistinguishable” from outside \( M \); i.e., each \( v \in V - M \) is either adjacent to every vertex of \( M \), or nonadjacent to every vertex of \( M \). We provide here a brief description of the modular decomposition of a graph. For a summary of many of the facets of modular decomposition, see [28,29].

If \( G \) is disconnected, a “parallel” decomposition step is performed, dividing \( G \) into connected components. If \( \tilde{G} \) is disconnected, a “series” decomposition step is performed, dividing \( \tilde{G} \) into complement-connected components. If \( G \) and \( \tilde{G} \) are connected, a “prime” decomposition step is performed, decomposing \( \tilde{G} \) into maximal proper submodules. In each case, all subgraphs with more than one vertex are decomposed recursively. The modular decomposition of \( G \) can be represented by a tree called the modular decomposition tree; internal nodes are labeled parallel, series, or prime, depending on the decomposition step that took place, and the decomposition tree for each subgraph found at a decomposition step is made a child of the node created for that step.
It is well known that each prime module has a unique decomposition into maximal proper submodules, making the decomposition tree for a graph unique up to isomorphism. It is also known that $M$ is a module of $G$ if and only if $M$ is either equal to the set of all leaf descendants of a prime node in the decomposition tree, or $M$ is the union of leaf descendants of some set of children of a parallel or series node in the decomposition tree [29]. Note that $M$ is a module of $G$ if and only if $M$ is a module of $\tilde{G}$. Recently, a number of algorithms [7,8,26] have been designed that find the modular decomposition tree of a graph in linear time.

4. Recognition via the Hoàng and Khouzam characterization

This section presents an algorithm for recognizing semi-simplicial and brittle graphs. This algorithm is based on the relationship between soft vertices and modules, which was noted by Hoàng and Khouzam in [19]. They essentially prove the following theorem.

**Theorem 1** (Hoàng and Khouzam [19]). A vertex of a graph $G$ is not in the middle of any $P_4$ if and only if every complement-connected component of the graph induced by $N(v)$ is a module in $G$.

Technically, Hoàng and Khouzam prove only the more difficult part of the theorem, the “only if” direction. However, it is easy to observe that if every complement-connected component of the graph induced by $N(v)$ is a module, then $v$ cannot be the middle of a $P_4$, since the vertex at distance 2 from $v$ in such a $P_4$ would distinguish two nonadjacent neighbors of $v$, which are necessarily in the same complement-connected component.

We can use the theorem above, together with existing algorithms, to improve the time complexity of both brittle graph and semi-simplicial graph recognition. There are a number of online algorithms that maintain information about connected components of a graph as edges are deleted; the current best algorithm [24] takes $O(\log^2 n)$ time per edge deleted.

To recognize semi-simplicial graphs, we repeatedly look for a vertex $x$ that is not mid-$P_4$, and remove it. The recognition algorithm runs as follows. We first compute the modular decomposition tree of the graph $G$, which has $O(n)$ nodes. Then, for each vertex $v$ in $G$, we find and label every complement-connected component of the graph induced by $N(v)$.

We must now determine whether every complement-connected component of the subgraph induced by $N(v)$ is a module of $G$. Given the component numbers, this can be done in $O(n)$ time per vertex, using the fact that modules of a graph are equal to either all leaf descendants of a prime node or the union of the leaf descendants of some set of children of a series or parallel node in the decomposition tree. We begin a postorder traversal of the decomposition tree. Let $i$ be the next internal node encountered during this postorder traversal. If all the children of $i$ have the same component number, mark $i$ with this component number. If all the vertices of this
component are descendants of $i$, then this component forms a module of $G$ and we can unmark $i$. If $i$ is a prime node that has both a child marked with a component number and at least one child that is not marked with this component number, we can conclude that this complement-connected component in the subgraph induced by $N(v)$ is not a module and thus, $v$ is the midpoint of at least one $P_4$. If $i$ is a series or parallel node that has some children marked with component numbers, and not all numbers are the same or there are unmarked children, then these components form modules of $G$ if and only if all vertices of these components are descendants of the marked children of $i$. It is not difficult to test for these conditions in time proportional to the number of children of $i$. If we reach the root in this postorder traversal, we can conclude that $v$ is not mid-$P_4$ in $G$ and we can remove it. We can test whether vertex $v$ is not mid-$P_4$ given the component numbers in $O(n^2)$ time and thus, we can find and remove a semi-simplicial vertex in $O(n^2)$ time. Overall, the total time traversing the modular decomposition trees is $O(n^3)$.

For each vertex $v$, we must maintain information regarding the complement-connected components of the graph induced by $N(v)$ while vertices are being deleted. We use one of two highly nontrivial algorithms to maintain this information involving connectivity as vertices are deleted. The problem of maintaining spanning trees efficiently as edges are deleted, rather than simply added, was an important open problem in dynamic graph algorithms; the first important breakthrough is contained in [14]. We can use a deterministic algorithm that runs in $O(\log^2 n)$ time per edge deletion [24], or a (Las Vegas) randomized algorithm with time complexity $O(n^2)$ for performing all vertex deletions [38]; both allow us to produce the current connected components in $O(n)$ time. For the deterministic algorithm, when a vertex $x$ is deleted, we update the connectivity information as we delete all edges incident on $x$; this takes $O(n^2 \log^2 n)$ time, since $O(n)$ edges are deleted from $O(n)$ subgraphs. Thus, the total time spent by the algorithm in maintaining information regarding connected components is $O(n^3 \log^2 n)$. Using the randomized algorithm, we spend $O(n^2)$ time maintaining connectivity information for each vertex $v$, giving an overall time bound of $O(n^3)$. Finally, we must recompute the modular decomposition tree of $G$ each time a vertex is deleted. This takes $O(mn)$ time overall, since each iteration takes linear time. This gives an $O(n^3 \log^2 n)$ time deterministic algorithm or an $O(n^3)$ time randomized algorithm for recognizing semi-simplicial graphs.

To recognize brittle graphs, we repeatedly look for a soft vertex $x$, and remove it. To determine whether a vertex $v$ is soft, we check whether every complement-connected component of the graph induced by $N(v)$ is a module in $G$ (in which case $v$ is not mid-$P_4$), and whether every connected component of the graph induced by $M(v)$ is a module of $G$ (in which case $v$ is not end-$P_4$). This is done by first finding and labeling every complement-connected component of the graph induced by $N(v)$. We then find and label every connected component of the graph induced by $M(v)$. We then perform the postorder traversal of the modular decomposition tree as described in the previous algorithm to determine if these components are modules in $G$. Each time a soft vertex is removed, we recompute the modular decomposition of the graph and update connectivity information. In the recognition of brittle graphs, we must maintain information regarding the connected components of the graph induced by
$M(v)$ as well as the complement-connected components of the graph induced by $N(v)$. This recognition algorithm for brittle graphs has a running time of $O(n^3 \log^2 n)$ for the deterministic case and $O(n^3)$ for the randomized case. A bottleneck step for both algorithms given above is finding connected components in the appropriate subgraphs as vertices are deleted, which may be a problem of independent interest.

5. HHP-free graphs

A graph is HHDA-free if it contains no hole and no induced house, domino, or “A”. Since every HHP-free graph is HHDA-free, it then follows that graph $G$ is HHP-free if and only if $G$ is HHDA-free and it has no “P”. Our approach, therefore, is to first test if the given graph is HHDA-free and then test for the presence of a “P”. We use the following characterization of HHDA-free graphs due to Olariu [31]. A module is nontrivial if it is nonempty and neither a singleton nor the entire vertex set of the graph.

**Theorem 2** (Olariu [31]). Graph $G$ is HHDA-free if and only if every induced subgraph of $G$ either is chordal or contains a nontrivial module.

A characteristic graph of a module $M$ in the modular decomposition of a graph $G$ is a graph induced by a subset consisting of a single vertex from each maximal submodule of $M$. Theorem 2 ensures that if $G$ is HHDA-free, then the characteristic graph of every prime module in the modular decomposition of $G$ is chordal. The characteristic graphs of series and parallel modules are clearly chordal.

If $H$ is an induced subgraph of $G$ that does not contain a nontrivial module, then the proper intersection of $H$ with any module of $G$ can contain at most one vertex. Hence, $H$ is isomorphic to the characteristic graph of some prime module in the modular decomposition of $G$. As none of hole, house, domino, and “A” have a nontrivial module, if $G$ is not HHDA-free, some characteristic graph of a prime module in the modular decomposition will fail to be chordal. Thus, Theorem 2 implies a linear time recognition algorithm for HHDA-free graphs [31]: perform modular decomposition and then check if the characteristic graph at each prime node of the decomposition tree is chordal.

We need the following lemma.

**Lemma 3.** HHDA-free graph $G$ has a “P” if and only if some characteristic graph in the modular decomposition of $G$ has a $P_4 \{a,b,c,d\}$ such that either the leaf descendants of vertex $b$ (or those of vertex $c$) induce a subgraph in $G$ that has at least two nonadjacent vertices.

**Proof.** We argue only the nontrivial direction. Suppose $G$ contains a “P”. As “P” is not chordal, but every characteristic graph in the modular decomposition of $G$ is chordal, it must be the case that the 4-cycle in the “P” is decomposed. But, each of the two $P_4$s in the “P” are not decomposable and the result follows. □
Thus, we have the following algorithm to recognize HHP-free graphs. First construct the modular decomposition tree $T$ of the given graph $G$ in linear time [7,8,26]. Perform a postorder traversal of $T$ to mark those nodes whose leaf descendents induce a subgraph in $G$ with at least two nonadjacent vertices. Then, for each characteristic graph $H$ of $T$, check if $H$ is chordal. If $H$ is not chordal, $G$ is not HHP-free. Otherwise, check if $H$ has any marked vertex $v$ that is mid-$P_4$ in $H$. We can use Theorem 1 to check in linear time whether a vertex is mid-$P_4$ or not. Each edge in a characteristic graph of $T$ corresponds to an edge in $G$, and an edge in $G$ corresponds to at most one edge among the edges in characteristic graphs of $T$. The number of nodes in a modular decomposition tree is $O(n)$, and each node in $T$ corresponds to exactly one node in one characteristic graph. Thus, the sum of the sizes of all characteristic graphs in $T$ is linear in the size of $G$, and hence the algorithm runs in $O(mn)$ time.

6. Welsh–Powell opposition graphs

We use the following characterization of WPO-graphs due to Randall and Olariu [33].

Theorem 4 (Olariu and Randall [33]). $G$ is a WPO-graph if and only if $G$ contains no induced $C_5$, $P_5$, house, or “P”.

It then follows that $G$ is a WPO-graph if and only if $G$ is HHP-free and $\tilde{G}$ is HH-free. As HH-free graphs and HHP-free graphs can be recognized in $O(n^3)$ time [23] and $O(mn)$ time, respectively, it follows that WPO-graphs can be recognized in $O(n^3)$ time.

7. Maxibrittle graphs

Our algorithm is based on the following characterization of maxibrittle graphs due to Preissmann et al. [34].

Theorem 5 (Preissmann et al. [34]). $G$ is a maxibrittle graph if and only if $G$ contains no induced $C_5$, $P_5$, house, fish, or complement of a fish.

It then follows that $G$ is a maxibrittle graph if and only if both $G$ and $\tilde{G}$ are HH-free and contain no fishes. Our approach to recognition is to first check that both $G$ and $\tilde{G}$ are HH-free. We then check if the HH-free graph $G$ has a fish or not; we then repeat the same in $\tilde{G}$. In order to determine whether an HH-free graph $G$ contains a fish, we locate a convenient vertex $x$ in $G$ and determine in $O(n^{2.376})$ time whether $x$ belongs to any fish in $G$; if it does, then $G$ is not a maxibrittle graph. Otherwise, we delete vertex $x$ from $G$ and similarly determine if the HH-free graph $G - x$ has a fish. As HH-free graphs can be recognized in $O(n^3)$ time [23], this approach yields an algorithm with
the time complexity of $O(n^{3.376})$. The convenient vertex that we find is the one that is not mid-$P_4$ in the graph.

In reference to the graph fish in Fig. 1, we refer to the vertex labeled $m$ as the mouth of fish; we refer to the vertex labeled $t$ as the tail of fish. As any vertex that is not mid-$P_4$ in a fish can only be either the mouth or the tail, we simply check for this. Note that every maxibrittle graph (and its complement) contains no hole and no induced house or domino, and hence is HHD-free. Therefore, the following theorem due to Jamison and Olariu [25] guarantees that if $G$ is maxibrittle, there is a vertex that is not mid-$P_4$ in $G$ and that such a vertex can be found in linear time.

**Theorem 6** (Jamison and Olariu [25]). A graph $G$ is HHD-free if and only if every ordering of the vertices of $G$ produced by LexBFS is a semi-perfect elimination ordering.

Here LexBFS refers to the well-known lexicographic breadth first search algorithm, which runs in linear time [36].

We have the following algorithm:

**Algorithm.** maxibrittle

**Input:** Graph $G$

**Output:** true if $G$ is maxibrittle and false otherwise.

```
{  1. if $G$ is not HH-free or $\bar{G}$ is not HH-free then
     return (false);
   2. for $H = G$ and then $\bar{G}$ do
       Order vertices of $H$ as $v_1 v_2 \ldots v_n$ using LexBFS;
       for $i = 1$ to $n$ do
           if $v_i$ is mid-$P_4$ in $H$ then
               return (false);
           if fish-mouth($H$, $v_i$) or fish-tail ($H$, $v_i$) then
               return (false);
           else
               $H = H - v_i$;
       }
   3. return (true);
}
```

We now explain how given an HH-free graph $G$ and a vertex $x$ that is not mid-$P_4$ in $G$, one can determine in $O(n^{3.376})$ time whether $v$ is mouth or tail of a fish in $G$. We need some properties of HH-free graphs first. The first of the lemmas appears in [23]. The second is a variation of a lemma from [23]; we provide its proof here.
We introduce some notation that will be used in the rest of the paper. For vertices 
\( x \) and \( y \) in a graph \( G \), we say \( x \) sees \( y \) to mean \( x \) is adjacent to \( y \) in \( G \) and \( x \)
misses \( y \) to mean \( x \) is not adjacent to \( y \) in \( G \). For a subset \( S \) of vertices and a vertex
\( x \) we say \( x \) sees \( S \) to mean \( x \) sees all the vertices in \( S \). A subset \( S \) of vertices in
a graph is nontrivial if \( S \) has at least two vertices and \( S \) is not the set of all the
vertices.

Let \( G \) be a graph and \( x \) be a vertex of \( G \). For a vertex \( y \in M(x) \), let \( n(x, y) \) refer
to the number of common neighbors, \(|N(x)\cap N(y)|\), of \( x \) and \( y \) in \( G \). Let \( \mathcal{R} = y_1 y_2 \ldots y_k \)
be a nondecreasing ordering of vertices of \( M(x) \) according to \( n(x, y_i) \). For vertices
\( u \in M(x) \) and \( v \in M(x) \), we say \( u < v \) in \( \mathcal{R} \) if \( u \) comes before \( v \) in \( \mathcal{R} \); also, we say \( v \)
dominates \( u \) if \( (N(x) \cap N(u)) \subseteq (N(x) \cap N(v)) \).

**Lemma 7** (Hoàng and Sritharan [23]). Let \( G \) be an HH-free graph and \( x \) be a vertex
in \( G \). Suppose vertices \( v, w \in M(x) \) are such that \( v < w \) in \( \mathcal{R} \). Further suppose that \( v \)
sees \( w \) in \( G \). Then \( w \) dominates \( v \).

**Lemma 8.** Let \( G \) be an HH-free graph and \( x \) be a vertex of \( G \). Suppose vertices
\( u, v, w \in M(x) \) are such that \( u < v < w \) in \( \mathcal{R} \). Further suppose that \( u \) sees both \( v \) and
\( w \), and \( v \) misses \( w \). Then \( w \) dominates \( v \).

**Proof.** By way of contradiction, suppose \( w \) does not dominate \( v \). Then \( v \) sees a vertex
\( a \in N(x) \) that \( w \) misses. Now, as \( n(x, w) \geq n(x, v) \), \( w \) must see some vertex \( b \in N(x) \)
that \( v \) misses. Then, by Lemma 7, \( u \) sees \( a \). Now, suppose \( u \) sees \( a \). Then the vertices
\( \{x, b, w, u, a\} \) induce either a \( C_5 \) or a house, which is a contradiction. Therefore, suppose
that \( u \) misses \( a \). If \( a \) sees \( b \), then the vertices \( \{b, w, u, v, a\} \) induce a \( C_5 \), which is also a
contradiction. On the other hand, if \( a \) misses \( b \), then the vertices \( \{x, b, w, u, v, a\} \) induce
a \( C_6 \), another contradiction. \( \square \)

**Lemma 9.** Let \( G \) be an HH-free graph and \( x \) be a vertex of \( G \) that is not mid-P_4.
Then \( x \) is tail of a fish in \( G \) if and only if there exist vertices \( u, v, w \in M(x) \) such that
\( u < v < w \) in \( \mathcal{R} \), \( w \) misses some vertex in \( N(x) \), \( u \) sees both \( v \) and \( w \), \( v \) misses
\( w \), and \( n(x, v) > n(x, u) \).

**Proof.** Suppose \( x \) is tail of some fish in \( G \). It is then easy to verify (via Lemmas 7
and 8) that the three vertices of the fish that are nonadjacent to \( x \) satisfy the conditions
for \( u, v, w \) in the lemma. Suppose vertices \( u, v, w \) that satisfy the conditions of the
lemma exist. By Lemma 7, \( v \) and \( w \) dominate \( u \). By Lemma 8, \( w \) dominates \( v \). Now,
as \( n(x, v) > n(x, u) \), there exists a vertex \( a \in N(x) \) that \( v \) sees but \( u \) misses. Since
\( w \) dominates \( v \), \( v \) sees \( a \) also. Let \( b \in N(x) \) be the vertex that \( w \) misses. Since \( w \)
dominates \( u \) and \( v \), both \( u \) and \( v \) miss \( b \). Finally, as \( x \) is not mid-P_4, \( b \) must see \( a \)
(or else \( \{b, x, a, w\} \) is a \( P_4 \) in which \( x \) is a midpoint). Then the vertices \( \{x, a, b, u, v, w\} \)
induce a fish in \( G \) in which \( x \) is the tail. \( \square \)

Our basic approach in algorithm fish-tail(G,x) is based on Lemma 9 and is the
following: for each pair of vertices \( v, w \in M(x) \) such that \( v \) misses \( w \) and \( w \) misses
some vertex of $N(x)$, we look for a vertex $u \in M(x)$ such that $u,v,w$ meet the conditions of Lemma 9.

Algorithm. fish-tail

Input: HH-free graph $G$ and vertex $x$ that is not mid-$P_4$.

Output: true if $x$ is tail of a fish in $G$ and false otherwise.

{  
1. Construct directed graph $H_1 = (V_1, E_1)$ where $V_1 = M(x)$, and $(v,u) \in E_1$ if and only if $u$ sees $v$ in $G$, $u < v$ in $\mathcal{R}$, and $n(x,u) < n(x,v)$.
2. Let $A$ be the adjacency matrix for $H_1$. Compute $B = A \times A^T$.
3. for each pair $v,w \in M(x)$ do
   if $v$ misses $w$ and $w$ misses some vertex of $N(x)$ and $B[v,w] > 0$
   then return (true);
4. return (false);
}

Lemma 10. Algorithm fish-tail is correct and runs in $O(n^{2.376})$ time.

Proof. Note that $B[v,w] > 0$ if and only if vertices $v$ and $w$ have a common out-neighbor $u$ in the graph $H_1$. As any out-neighbor of a vertex in $H_1$ comes before it in $\mathcal{R}$, it follows that $B[v,w] > 0$ if and only if there is a vertex $u \in M(x)$ such that $u,v,w$ satisfy conditions of Lemma 9. Since the running time of the algorithm is dominated by the step that computes matrix $B$, the time complexity is $O(n^{2.376})$. \qed

We need some more notation. Suppose $G$ is a graph and $x$ is a vertex of $G$ that is not mid-$P_4$. For a vertex $y \in M(x)$, we use $b(y)$ to refer to the number of non-trivial connected components of the subgraph induced by $N(x)$ in $\hat{G}$ that $y$ sees in $G$. Recall, by Theorem 1, that each such nontrivial component is a nontrivial module of $G$.

Lemma 11. Let $G$ be an HH-free graph and $x$ be a vertex of $G$ that is not mid-$P_4$. Then $x$ is mouth of a fish in $G$ if and only if there exist vertices $u,v,w \in M(x)$ such that $u < v < w$ in $\mathcal{R}$, $u$ sees both $v$ and $w$, $v$ sees $w$, and $b(w) > b(v)$.

Proof. Suppose $x$ is mouth of some fish in $G$. It is then easy to verify (via Lemmas 7, 8 and Theorem 1) that the three vertices of the fish that are nonadjacent to $x$ satisfy the conditions for $u,v,w$ in the lemma. Suppose vertices $u,v,w$ that satisfy the conditions of the lemma exist. By Lemma 7, $w$ dominates $v$ and $v$ dominates $u$. Now, as $w$ dominates $v$ and as $b(w) > b(v)$, it must be the case that $w$ sees some nontrivial component $S$ of the subgraph induced by $N(x)$ in $\hat{G}$ that $v$ misses. As $v$ dominates $u$, $u$ then misses $S$ also. Let $a$ and $b$ be two nonadjacent vertices of $N(x)$ that belong to $S$; as $S$ is a nontrivial component, $a$ and $b$ exist. Now, the vertices $\{x,a,b,u,v,w\}$ induce a fish whose mouth is $x$. \qed
Our basic approach in algorithm fish-mouth($G$, $x$) is based on Lemma 11 and is the following: for each pair of vertices $u, v \in M(x)$ such that $u$ sees $v$, we look for a vertex $w \in M(x)$ such that $u, v, w$ meet the conditions of Lemma 11.

Algorithm. fish-mouth

Input: HH-free graph $G$ and vertex $x$ that is not mid-$P_4$.

Output: true if $x$ is mouth of a fish in $G$ and false otherwise.

{ 
1. Construct directed graph $H_2 = (V_2, E_2)$ where $V_2 = M(x)$, 
   and $(u, v) \in E_2$ if and only if $u$ sees $v$ in $G$, $u < v$ in $\mathcal{R}$, and $b(u) < b(v)$.
2. Let $A$ be the adjacency matrix for $H_2$. Compute $C = A \times A^T$.
3. for each pair $u, v \in M(x)$ do
   if $u$ sees $v$ and $C[u, v] > 0$ then
     return (true);
4. return (false);
}

Lemma 12. Algorithm fish-mouth is correct and runs in $O(n^{2.376})$ time.

Proof. Note that $C[u, v] > 0$ if and only if vertices $u$ and $v$ have a common out-neighbor $w$ in the graph $H_2$. Let us assume, without loss of generality, that $u < v$ in $\mathcal{R}$. Since any out-neighbor of a vertex in $H_2$ comes after it in $\mathcal{R}$, it follows from the construction of $H_2$ that $C[u, v] > 0$ if and only if there is a vertex $w \in M(x)$ such that $u < v < w$ in $\mathcal{R}$, $b(w) > b(v)$, and $b(w) > b(u)$. We just have to make the argument that $w$ sees some nontrivial component $S$ of the subgraph induced by $N(x)$ in $\tilde{G}$ that $u$ and $v$ both miss. A nontrivial component $S$ that $w$ sees but $v$ misses exists as $b(w) > b(v)$ and $w$ dominates $v$. Now, as $v$ dominates $u$, $u$ will miss $S$ also. Therefore, $u, v, w$ satisfy conditions of Lemma 11. Since the running time of the algorithm is dominated by the step that computes matrix $C$, the time complexity is $O(n^{2.376})$. \hfill \Box

We therefore have the following theorem:

Theorem 13. Maxibrittle graphs can be recognized in $O(n^{3.376})$ time.

8. Conclusions

Bipolarizable and $P_4$-simplicial graphs can be recognized in $O(n^{3.376})$ time. Brittle graphs and semi-simplicial graphs can be recognized deterministically in $O(n^3 \log^2 n)$ time or by a randomized Las Vegas algorithm in $O(n^3)$ time. An elimination sequence useful in some optimization problems can also be calculated in this amount of time. While these time bounds improve previous known bounds, the algorithms work by orienting all forced edges and by successively looking for soft vertices in a general
graph, completely ignoring special properties of brittle, semi-simplicial and bipolarizable graphs such as that there are no long holes in the graph or its complement. Thus, it is to be expected that the time bounds for recognizing all of these graph classes can be improved considerably. Welsh–Powell opposition graphs can be recognized in $O(n^3)$ time and maxibrittle graphs can be recognized in $O(n^{3.376})$ time. Again, these algorithms make minimal use of structural properties of the graphs. Thus, it is expected that better recognition algorithms for these classes are also possible.

Acknowledgements

We would like to thank Monika Hertzenberg and Mikkel Thorup for their assistance with this paper.

References

