Algorithmica (2007) 47: 119–138
DOI: 10.1007/s00453-006-1225-y

Algorithmica

Detecting Holes and Antiholes in Graphs

Stavros D. Nikolopoulos[1] and Leonidas Palios[1]

Abstract. In this paper we study the problems of detecting holes and antiholes in general undirected graphs, and we present algorithms for these problems. For an input graph G on n vertices and m edges, our algorithms run in $O(n + m^2)$ time and require $O(nm)$ space; we thus provide a solution to the open problem posed by Hayward et al. in [17] asking for an $O(n^4)$-time algorithm for finding holes in arbitrary graphs. The key element of the algorithms is the use of the depth-first-search traversal on appropriate auxiliary graphs in which moving between any two adjacent vertices is equivalent to walking along a P_4 (i.e., a chordless path on four vertices) of the input graph or on its complement, respectively. The approach can be generalized so that for a fixed constant $k \geq 5$ we obtain an $O(n^{k-1})$-time algorithm for the detection of a hole (antihole resp.) on at least k vertices. Additionally, we describe a different approach which allows us to detect antiholes in graphs that do not contain chordless cycles on five vertices in $O(n + m^2)$ time requiring $O(n + m)$ space. Again, for a fixed constant $k \geq 6$, the approach can be extended to yield $O(n^{k-2})$-time and $O(n^2)$-space algorithms for detecting holes (antiholes resp.) on at least k vertices in graphs which do not contain holes (antiholes resp.) on $k - 1$ vertices. Our algorithms are simple and can be easily used in practice. Finally, we also show how our detection algorithms can be augmented so that they return a hole or an antihole whenever such a structure is detected in the input graph; the augmentation takes $O(n + m)$ time and space.

Key Words. Holes, Antiholes, Weakly chordal graphs, Co-connectivity.

1. Introduction. We consider finite undirected graphs with no loops or multiple edges. Let G be such a graph and let $v_0, v_1, \ldots, v_{k-1}$ be a sequence of k distinct vertices such that there is an edge from v_i to $v_{(i+1) \bmod k}$ (for all $i = 0, 1, \ldots, k-1$), and no other edge between any two of these vertices; we say that this is a *chordless cycle* on k vertices. A *hole* is a chordless cycle on five or more vertices; an *antihole* is the complement of a hole.

Holes and antiholes have been extensively studied in many different contexts in algorithmic graph theory. They constitute forbidden induced subgraphs for many different classes of graphs, both perfect (see [4] and [14]) and not perfect. In fact, there is a structural characterization of a hierarchy of graph classes with no holes on k vertices in which each class excludes antiholes on i vertices, $5 \leq i \leq k + 1$; at one end of the hierarchy is the class of graphs with no holes and at the other end is the class of graphs with no holes and no antiholes [13] (the graphs with no holes and no antiholes form the class of weakly chordal graphs, also known as weakly triangulated graphs [15]). Thus, the fast detection of holes or antiholes helps in the efficient recognition of all these classes of graphs.

Moreover, the problem of detecting holes or antiholes has been the focus of considerable attention lately in light of the recent proof by Chudnovsky et al. [8] of the perfect

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

Received December 19, 2002; revised March 23, 2004, and May 25, 2005. Communicated by H. Gabow.
Online publication January 24, 2007.

graph conjecture [1], which states that a graph is perfect if and only if it contains no holes and no antiholes on an odd number of vertices. Thus, efficient algorithms for detecting induced holes or antiholes on an odd number of vertices will imply fast recognition of perfect graphs; the currently fastest algorithm for recognizing perfect graphs runs in $O(n^9)$ time [7].

Several algorithms for detecting holes and antiholes in graphs have been proposed in the literature. The definition of holes and antiholes implies that such algorithms can be applied without error on the biconnected components of the input graph and of its complement, respectively, instead of the entire graph. Although this approach may lead to the fast detection of holes and antiholes in graphs with small biconnected components, it does not yield any gain in the asymptotic sense.

The problem of determining whether a given graph on n vertices and m edges contains a chordless cycle on k or more vertices, for some fixed value of $k \geq 4$, is solved in $O(n^k)$ time [16]. Spinrad [21] reduced the time complexity of the problem to $O(n^{k-3}M)$, where $M \simeq n^{2.376}$ is the time required to multiply two $n \times n$ matrices. Note that the problem of determining whether a graph contains a chordless cycle on four or more vertices can be solved in $O(n+m)$ time [14], [20], [23] (it is the well-known chordal graph recognition problem).

The algorithms of [16] and [21] can be used for the recognition of weakly chordal graphs in $O(n^5)$ and $O(n^{4.376})$ time, respectively. Further progress on the weakly chordal graph recognition problem includes the $O(n^4)$-time algorithm of Spinrad and Sritharan [22], and the $O(m^2)$-time algorithms of Hayward et al. [17] and of Berry et al. [2]. It is interesting to note that the algorithm of [17] produces a hole or an antihole certificate whenever the input graph is not weakly chordal. In the same paper the authors posed as an open problem the designing of an $O(n^4)$-time algorithm to find a hole in an arbitrary graph.

In this paper we study the above-mentioned open problem and we present two algorithms, one for the detection of holes and another for the detection of antiholes in arbitrary graphs. Both algorithms run in $O(n+m^2)$ time and require $O(nm)$ space, and rely on the detection of a cycle in the input graph or in its complement, respectively, satisfying certain conditions. The existence of such a cycle is checked by means of running depth-first-search (DFS) on appropriate auxiliary (directed) graphs; in fact, in order to achieve the stated space complexity, we run DFS on these graphs *implicitly*. The approach can be generalized and yields algorithms for detecting holes and antiholes on at least k vertices; these algorithms take $O(n\,m^{p-1})$ time if $k = 2p \geq 6$, and $O(n+m^p)$ time if $k = 2p+1 \geq 5$, thus improving over the time complexity of the best currently known algorithm for the problem [21].

We also describe another algorithm for the detection of antiholes in graphs that do not contain chordless cycles on five vertices: the algorithm processes each edge of the input graph in order to determine whether the endpoints of the edge participate in an antihole (on at least six vertices), and relies on the computation of the co-connected components of subgraphs of the input graph; it runs in $O(n+m^2)$ time and takes $O(n+m)$ space. In addition to providing a different way of approaching the problem, this result has the potential to yield a linear-space antihole detection algorithm provided that C_5s can be detected in $O(n+m^2)$ time and $O(n+m)$ space. The same approach yields an $O(n^2m)$-time and $O(n^2)$-space algorithm for detecting holes in graphs that

do not contain chordless cycles on five vertices. As with the previous approach, this can also be extended to the problem of detecting antiholes on at least k vertices for any constant $k \geq 6$: for an undirected graph G on n vertices and m edges which does not contain antiholes on $k-1$ vertices, it can be determined whether G contains an antihole on at least k vertices in $O(n+m^{p-1})$ time if $k = 2p$ or in $O(n\,m^{p-1})$ time if $k = 2p+1$ using $O(n+m)$ space; for holes, the respective time complexities are $O(n^2\,m^{p-2})$ time if $k = 2p$ and $O(n\,m^{p-1})$ time if $k = 2p+1$, and the space complexity is $O(n^2)$.

Additionally, we describe how to augment our three main detection algorithms so that they return a hole or an antihole in the case where such a structure exists in the input graph; the augmented hole (antihole, respectively) detection algorithm produces a hole (an antihole, respectively) in $O(n+m)$ additional time and $O(n+m)$ space, thus, providing the most natural evidence that the input graph indeed contains a hole or an antihole. Finally, we note that, as a by-product, our hole and antihole detection algorithms can be used for recognizing weakly chordal graphs leading to a solution that matches the best currently known time complexity for this problem [2], [17].

The paper is structured as follows. In Section 2 we establish the notation and the terminology we use throughout the paper. In Section 3 we describe the hole detection algorithm, establish its correctness, and analyze its complexity. An antihole detection algorithm for general graphs is given in Section 4, while an antihole detection algorithm for graphs that do not contain chordless cycles on five vertices is given in Section 5. Section 6 concludes the paper with a summary of our results and some open problems.

2. Preliminaries. Let G be a finite undirected graph with no loops or multiple edges. We denote by $V(G)$ and $E(G)$ the vertex set and edge set of G. The subgraph of a graph G induced by a subset S of vertices of G is denoted by $G[S]$.

A *path* in G is a sequence of vertices $v_0v_1\cdots v_k$ such that $v_iv_{i+1} \in E(G)$ for $i = 0, 1, \ldots, k-1$; we say that this is a path from v_0 to v_k and that its *length* is k. A path is called *simple* if none of its vertices occurs more than once; it is called *trivial* if its length is equal to 0. A simple path $v_0v_1\cdots v_k$ is *chordless* if $v_iv_j \notin E(G)$ for any two non-consecutive vertices v_i, v_j in the path. Throughout the paper the chordless path on k vertices is denoted by P_k. In particular, a chordless path on three vertices is denoted by P_3 and a chordless path on four vertices is denoted by P_4. A sequence of vertices $v_0v_1\cdots v_{k-1}$ forms a *cycle* (resp. *simple cycle*) iff $v_0v_{k-1} \in E(G)$ and $v_0v_1\cdots v_{k-1}$ is a path (resp. simple path) in G; its length is equal to k. A simple cycle $v_0v_1\cdots v_{k-1}$ is said to be *chordless* if no edge v_iv_j exists in $E(G)$ such that $|i-j| \neq 1 \bmod k$. The chordless cycle on k vertices is denoted by C_k; in particular, C_5 is the chordless cycle on five vertices.

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 $N[x] := N(x) \cup \{x\}$. The neighborhood of a subset S of vertices is defined as $N(S) := \left(\bigcup_{x\in S} N(x)\right) - S$ and its closed neighborhood as $N[S] := N(S) \cup S$. The notion of the neighborhood can be extended to edges: for an edge $e = xy$, the *neighborhood* (*closed neighborhood*) of e is the vertex set $N(\{x, y\})$ (resp. $N[\{x, y\}]$) and is denoted by $N(e)$ (resp. $N[e]$). For an

edge $e = xy$, we define the following three sets:

$$\begin{aligned} A(e; x) &= N(x) - N[y], \\ A(e; y) &= N(y) - N[x], \\ A(e) &= N(x) \cap N(y); \end{aligned}$$

clearly, these sets form a partition of the neighborhood $N(e)$ of the edge e.

We close this section by describing the co-connectivity problem which plays a crucial role in the antihole detection algorithm for graphs that do not contain a C_5, which we propose in this paper. The co-connectivity problem on a graph G is that of finding the connected components of the complement $\overline{G}$; the connected components of $\overline{G}$ are called *co-connected components* (or *co-components*) of G. The co-components of a graph G on n vertices and m edges can be computed in $O(n + m)$ time and space [5], [12], [18].

3. Detecting Holes. The hole detection algorithm relies on the result stated in the following lemma.

LEMMA 3.1. *An undirected graph G contains a hole if and only if G contains a cycle $u_0u_1 \cdots u_k$, where $k \geq 4$, such that the paths $u_iu_{i+1}u_{i+2}u_{i+3}$ for each $i = 0, 1, \ldots, k-3$, and the path $u_{k-2}u_{k-1}u_ku_0$ are P_4s of G.*

PROOF. ($\Longrightarrow$) Suppose that G contains a hole; then the vertices of the hole induce a cycle meeting the conditions of the lemma.

($\Longleftarrow$) Suppose now that G contains a cycle as described in the lemma; let $v_0v_1 \cdots v_h$ be the shortest such cycle. Then this cycle is a hole:

(a) since the cycle meets the conditions of the lemma, then $h \geq 4$, which implies that the cycle is of length at least equal to 5;
(b) the cycle is chordless. Suppose for contradiction that there existed chords. With each chord v_iv_j, we associate its "length," which is defined as $length(v_iv_j) = |j - i|$; let $v_\ell v_r$, where $\ell < r$, be the chord with minimum length. Note that $r \geq \ell + 4$; this follows from the fact that $r \geq 4$ (because $v_0v_1v_2v_3$ is a P_4) and the fact that $v_{r-3}v_{r-2}v_{r-1}v_r$ is a P_4. Then $v_{r-2}v_{r-1}v_rv_\ell$ is a P_4 in G because it is a path in G, $v_{r-2}v_r \notin E(G)$ (recall that $v_{r-3}v_{r-2}v_{r-1}v_r$ is a P_4), and $v_\ell v_{r-2} \notin E(G)$ and $v_\ell v_{r-1} \notin E(G)$ for otherwise these would be chords whose *length*-value would be smaller than that of the chord $v_\ell v_r$, in contradiction to the minimality of $length(v_\ell v_r)$. Additionally, $v_iv_{i+1}v_{i+2}v_{i+3}$ is a P_4 for all $i = \ell, \ell + 1, \ldots, r - 3$. Thus, the cycle $v_\ell v_{\ell+1} \cdots v_r$ would meet the conditions of the lemma; as it would be shorter than the cycle $v_0v_1 \cdots v_h$, this would contradict the fact that the latter cycle is the shortest such cycle. Hence, the cycle $v_0v_1 \cdots v_h$ is chordless.

Therefore, G contains a hole. □

Lemma 3.1 is the basis of our algorithm for the detection of holes. Suppose that we are interested in detecting whether a graph G contains a hole. We consider an auxiliary

directed graph G' which is defined as follows:

$$V(G') = \{v_{abc} \mid abc \text{ is a } P_3 \text{ in the graph } G\},$$
$$E(G') = \{(v_{abc}, v_{bcd}) \mid abcd \text{ is a } P_4 \text{ in the graph } G\}.$$

Note that if abc is a P_3 of G, the graph G' contains the vertices v_{abc} and v_{cba}; similarly, if $abcd$ is a P_4 of G, the graph G' contains the edges (v_{abc}, v_{bcd}) and (v_{dcb}, v_{cba}). The definition of G' implies that moving from vertex to vertex in G' is equivalent to proceeding along P_4s of G. Thus, the condition of Lemma 3.1 on G can be checked by applying DFS on G' [10]; more specifically, it is not difficult to see that the following holds:

LEMMA 3.2. *Let G be a graph, let G' be the associated auxiliary graph defined above, and suppose that we run DFS on G'. If during the DFS, the DFS-path is $v_{u_0u_1u_2}v_{u_1u_2u_3}\cdots v_{u_{k-2}u_{k-1}u_k}$, where $u_i \neq u_j$ for all $0 \leq i < j < k$, and $u_k = u_\ell$ for some ℓ such that $0 \leq \ell < k$, then the vertices $u_\ell, u_{\ell+1}, \ldots, u_{k-1}$ form a cycle in G satisfying the conditions of Lemma* 3.1. *Moreover, if G contains a hole, then during the DFS on G' we will find a sequence of vertices in G' whose associated P_3s in G form a cycle as in Lemma* 3.1.

If the graph G has n vertices and m edges, the graph G' has $O(nm)$ vertices and $O(m^2)$ edges; note that a P_4 $abcd$ is uniquely characterized by the ordered pair $((a, b), (c, d))$ where (a, b) and (c, d) are ordered pairs of adjacent vertices in G. Thus, constructing G', so that we can run DFS on it, clearly requires $O(m^2)$ space. To reduce the space required, we do not construct G'; instead, we run DFS on it implicitly: in order to search the graph G exhaustively, we start from each P_3 of G; in the general step, we try to extend a P_3 abc into P_4s of the form $abcd$, then, for each such P_4, we proceed by extending the P_3 bcd into P_4s of the form $bcde$, and so on; in the above cases, the *active-path* is abc, it becomes $abcd$, then $abcde$, and so on (when we backtrack, the corresponding vertices are removed from the end of the current active-path); if ever we proceed to a P_3 xyz such that z appears again in the current active-path, then the graph G contains a cycle as in Lemma 3.1 and consequently a hole. Note that the current active path contains precisely the vertices of the P_3s associated with the vertices in the current DFS-path on G', where the common vertices of G in adjoining P_3s are recorded exactly once.

During the DFS on the graph G', vertices (corresponding to P_3s of G) are marked so that they are not "visited" again. In order to simulate this, we use an auxiliary array *visited_P3*$[(a, b), c]$, where $a, b, c \in V(G)$ and a, b are adjacent in G; for each edge ab of G, the array has entries *visited_P3*$[(a, b), c]$ as well as *visited_P3*$[(b, a), c]$ for every $c \in V(G)$, and hence its size is $2m \cdot n$. The entry *visited_P3*$[(a, b), c]$ is equal to 1 iff the vertices a, b, c induce a P_3 abc of G which has already been visited during the DFS, otherwise it is equal to 0 (note that if abc is a P_3 of G, the array contains the entries *visited_P3*$[(a, b), c]$ and *visited_P3*$[(c, b), a]$). Additionally, in order to be able to test whether a vertex belongs to the current active-path, we use another auxiliary array *in_path*[ ] of size n; for a vertex v, *in_path*$[v]$ is equal to 1 if v belongs to the current active-path, and is 0 otherwise. Below, we give a detailed description of the algorithm when applied on a connected input graph G; the case of a disconnected input graph is

discussed after the analysis of the algorithm. The algorithm assumes that G is given in adjacency-list representation, from which it computes the adjacency matrix of G so that adjacency tests can be answered in constant time.

HOLE-DETECTION ALGORITHM

Input: a connected undirected graph G.
Output: a message whether G contains a hole or not.

1. Initialize the entries of the arrays *visited_P3*[ ] and *in_path*[ ] to 0; compute the adjacency matrix A[ ] of G;
2. `for` each vertex u of G `do`
   2.1 $in\_path[u] \leftarrow 1$;
   2.2 `for` each edge vw of G `do`
   `if` u is adjacent to v and non-adjacent to w `and` $visited\_P3[(u, v), w] = 0$
   `then` $in\_path[v] \leftarrow 1$;
   $process(u, v, w)$;
   $in\_path[v] \leftarrow 0$;
   `if` u is non-adjacent to v and adjacent to w `and` $visited\_P3[(u, w), v] = 0$
   `then` $in\_path[w] \leftarrow 1$;
   $process(u, w, v)$;
   $in\_path[w] \leftarrow 0$;
   2.3 $in\_path[u] \leftarrow 0$;
3. Print that G does not contain a hole.

where the procedure *process*( ) is as follows:

process(a, b, c)

1. $in\_path[c] \leftarrow 1$;
2. $visited\_P3[(a, b), c] \leftarrow 1$;
   $visited\_P3[(c, b), a] \leftarrow 1$;
3. `for` each vertex d adjacent to c in G `do`
   3.1 `if` d is adjacent neither to a nor to b in G
   `then` {*abcd is a* P_4 *of* G}
   3.2 `if` $in\_path[d] = 1$
   `then` print that G has a hole; exit;
   `else if` $visited\_P3[(b, c), d] = 0$
   `then` $process(b, c, d)$;
4. $in\_path[c] \leftarrow 0$;

It is important to observe that the description of the procedure *process*( ) guarantees that from a P_3 *abc* we proceed to a P_3 *bcd* only if *abcd* is a P_4 of the input graph G. Additionally, following the general description of the DFS, the procedure sets the corresponding entries of the array *visited_P3*[ ] shortly after it begins, thus preventing a second call to the procedure on the same P_3. Thus, the procedure *process*( ) is called exactly once for each P_3 of G.

The correctness of the algorithm follows from Lemmas 3.1 and 3.2, and from the correctness of the implicit execution of DFS on the graph G'.

Time and space complexity. Let n and m be the number of vertices and edges of the input graph G, respectively. Since G is connected, $n = O(m)$. We note that in order to be able to access the endpoints of an edge in constant time, we assume that the record of each edge uv of G contains pointers to u and v; similarly, in order to be able to access an edge in constant time given its endpoints, we assume that each entry of the adjacency matrix of G corresponding to two vertices u and v that are adjacent in G also stores a pointer to the edge uv.

Before analyzing the time complexity of each step of the algorithm, we turn to the procedure *process*( ). We note that the procedure is called exactly once for each P_3 of G, i.e., $O(nm)$ times, and that, if we ignore the time taken by the recursive calls, a call *process*(a, b, c) takes $O(|N(c)| + 1)$ time by using the adjacency list of the vertex c to retrieve c's neighbors, and by using the adjacency matrix $A[\,]$ to answer adjacency tests in constant time. Therefore, the time taken by all the calls to the procedure *process*( ) is $O(m^2)$, since each quadruple of vertices a, b, c, d where abc is a P_3 and d is adjacent to c is uniquely characterized by the ordered pair $((a, b), (c, d))$ where (a, b) and (c, d) are ordered pairs of adjacent vertices in G.

Step 1 of the main body of the algorithm clearly takes $O(nm)$ time. If the time taken by the calls to the procedure *process*( ) is ignored, Step 2 takes $O(nm)$ time; again, the adjacencies are checked in constant time by means of the adjacency matrix $A[\,]$ of G. Step 3 takes constant time. Thus, the time complexity of the algorithm for a connected graph on n vertices and m edges is $O(m^2)$. The space needed is $O(nm)$: $O(n)$ and $O(nm)$ for the arrays *in_path*[ ] and *visited_P*3[ ], respectively, and $O(n^2)$ for the matrix $A[\,]$ and the adjacency-list representation of the input graph.

Summarizing, we have the following result.

LEMMA 3.3. *Let G be a connected undirected graph on n vertices and m edges. Then the proposed algorithm determines whether G contains a hole in $O(m^2)$ time and $O(nm)$ space.*

The case of a disconnected input graph. If the input graph G is disconnected, we work on each of its connected components; let n_i and m_i denote the number of vertices and edges of the ith connected component, respectively. The computation of the connected components takes $O(n+m)$ time [10], while processing each of them takes $O(m_i^2)$ time. Since $\sum_i m_i = m$, we have that $O(n+m^2)$ time suffices for detecting holes in any graph on n vertices and m edges. In this case the space needed is $O(\sum_i (n_i m_i)) = O(nm)$. Therefore, we obtain the following theorem.

THEOREM 3.1. *Let G be an undirected graph on n vertices and m edges. Then there is an algorithm to determine whether G contains a hole using $O(n+m^2)$ time and $O(nm)$ space.*

3.1. *Providing a Certificate.* The hole-detection algorithm can be easily augmented so that it provides a certificate whenever it decides that the input graph G contains a hole.

To facilitate the computation of the certificate, we slightly modify the definition of the array *in_path*[ ]: in particular, for vertices participating in the active-path, the entries of the array *in_path*[ ] store the position of the corresponding vertex in the path, i.e., if v is the ith vertex in the active-path, *in_path*[v] is equal to i (if v does not participate in the active-path, *in_path*[v] remains 0); this may necessitate replacing the call *process*(a, b, c) by *process*(a, b, c, i), whenever c is the ith vertex in the path.

Now, suppose that the algorithm concludes that G contains a hole; then the condition in Step 3.2 of the procedure *process*( ) during the execution of a call, say, *process*(a, b, c, k), is found true for some vertex d. If d is located in the jth position of the current active-path, the vertices located in positions $j, j+1, \ldots, k$ form a cycle satisfying the conditions in the statement of Lemma 3.1. Then if this cycle is chordless, it is a hole; otherwise, in accordance with the second part of the proof of Lemma 3.1, it suffices to find a chord of the cycle with the minimum length-value.

Therefore, in order to isolate a hole, we execute the following steps right before stopping in Step 3.2 of *process*(a, b, c, k):

(i) the length-value is initialized to $k - j$ corresponding to the edge cd which closes the cycle (note that the vertices c and d are in positions k and j of the active-path, respectively);
(ii) we consider all edges incident on the vertices of the cycle, we find those which have both endpoints on the cycle and are not edges of the cycle, and among these we find an edge (if any) exhibiting the minimum length-value;
(iii) if the edge exhibiting the computed minimum length-value is incident on the vertices in positions $i_{\min}$ and $i_{\max}$ of the active-path, where $j \leq i_{\min} < i_{\max} \leq k$, then the vertices in the positions $i_{\min}, i_{\min}+1, \ldots, i_{\max}$ induce a hole in G.

The correctness of the computation follows from the proof of Lemma 3.1. Moreover, it is not difficult to see that the certificate computation takes $O(n+m)$ time: the edges incident on a vertex can be accessed in constant time per such edge using the adjacency-list representation of the input graph, and finding the position of a vertex in the active-path is done in constant time by using the updated array *in_path*[ ]. The additional space required for this computation is $O(1)$. Thus, we have:

THEOREM 3.2. *Let G be an undirected graph on n vertices and m edges. The hole-detection algorithm presented in this section can be augmented so that it outputs a hole as a certificate whenever G contains one. The certificate computation takes $O(n+m)$ time and $O(1)$ space.*

3.2. *Detecting Holes on at Least k Vertices.* The above approach which yields the hole-detection algorithm can be generalized to yield algorithms for the detection of holes on at least k vertices, where $k \geq 5$. For an input graph G, we consider the following family of directed graphs $G^{(\ell)}$:

$$V(G^{(\ell)}) = \{v_{u_1u_2\cdots u_{\ell-1}} \mid u_1u_2\cdots u_{\ell-1} \text{ is a } P_{\ell-1} \text{ in the graph } G\},$$
$$E(G^{(\ell)}) = \{(v_{u_1u_2\cdots u_{\ell-1}}, v_{u_2u_3\cdots u_\ell}) \mid u_1u_2\cdots u_\ell \text{ is a } P_\ell \text{ in the graph } G\}.$$

Clearly, $G = G^{(2)}$ and $G' = G^{(4)}$ where G' is the auxiliary graph defined in Section 3. Then, in the same fashion in which the execution of DFS on $G' = G^{(4)}$ enables us to solve the problem of detecting a hole (on at least five vertices), the execution of DFS on $G^{(k-1)}$ enables us to solve the problem of detecting whether an undirected graph contains a hole on at least k vertices, for any constant $k \geq 5$. This follows from the following extension of Lemma 3.1:

COROLLARY 3.1. *Let $k \geq 5$ be a constant. An undirected graph G contains a hole on at least k vertices if and only if G contains a cycle $u_0u_1 \cdots u_t$, where $t \geq k-1$, such that every path $u_iu_{i+1} \cdots u_{i+k-2}$ for each $i = 0, 1, \ldots, t-k+2$, and the path $u_{t-k+3}u_{t-k+4} \cdots u_tu_0$ are all P_{k-1}s of G.*

Then Lemma 3.2 and Corollary 3.1 yield the following corollary:

COROLLARY 3.2. *Let G be a graph, let $k \geq 5$ be a constant, let $G^{(k-1)}$ be the associated auxiliary graph defined above, and suppose that we run DFS on $G^{(k-1)}$. Then, if during the DFS, the DFS-path is $v_{u_0 \cdots u_{k-3}} v_{u_1 \cdots u_{k-2}} \cdots v_{u_{r-k+3} \cdots u_r}$, where $u_i \neq u_j$ for all $0 \leq i < j < r$, and $u_r = u_p$ for some p such that $0 \leq p < r$, then the vertices $u_p, u_{p+1}, \ldots, u_{r-1}$ form a cycle in G satisfying the conditions of Corollary* 3.1. *Moreover, if G contains a hole on at least k vertices, then during the DFS on $G^{(k-1)}$ we will find a sequence of vertices in $G^{(k-1)}$ whose associated P_{k-2}s in G form a cycle as in Lemma* 3.1.

In this case as well, we do not construct the graph $G^{(k-1)}$. Instead, we implicitly run DFS on it: we start from each P_{k-2} of G that we have not encountered so far; in the general step, we try to extend a P_{k-2} $u_0u_1 \cdots u_{k-3}$ into P_{k-1}s of the form $u_0u_1 \cdots u_{k-3}u_{k-2}$, then, for each such P_{k-1}, we proceed by extending the P_{k-2} $u_1u_2 \cdots u_{k-2}$ into P_{k-1}s, and so on. Since there are $O(m^a)$ chordless paths on $2a$ vertices and $O(nm^a)$ on $2a+1$ vertices, and it takes $O(k)$ time to determine whether a vertex extends a P_{k-1} into a P_k, we have:

COROLLARY 3.3. *Let G be an undirected graph on n vertices and m edges, and let $k \geq 5$ be a constant. Then, by implicitly running DFS on the auxiliary graph $G^{(k-1)}$, we can determine whether G contains a hole on at least k vertices in $O(n\,m^{p-1})$ time if $k = 2p$, and in $O(n + m^p)$ time if $k = 2p+1$.*

Corollary 3.3 implies that the detection of holes on at least k vertices takes $O(n^{k-1})$ time resulting in an improved time complexity on this problem [16], [21]. The drawback is that the space needed is $O(m^{p-1})$ if $k = 2p$, and $O(n\,m^{p-1})$ if $k = 2p+1$.

4. Detecting Antiholes. Since an antihole is the complement of a hole, one can use the algorithm of the previous section on the complement of a graph in order to determine whether it contains an antihole. Such an approach may however require $\Theta(n^4)$ time, where n is the number of vertices of the graph, since the complement may have as many as $\Theta(n^2)$ edges. Below, we present an algorithm for the detection of antiholes which applies the approach described in Section 3 on the complement of the input graph G without however computing the complement explicitly and which takes $O(n+m^2)$ time

and $O(nm)$ space when G has n vertices and m edges. As in Section 3, the algorithm uses an array *visited_P3*$[(a,b),c]$, where $a,b,c \in V(G)$ and $ab \in E(G)$, and thus is of size $2m \cdot n$; *visited_P3*$[(a,b),c]$ is equal to 1 iff acb is a P_3 of $\overline{G}$ which has already been visited during the DFS, and is 0 otherwise. The input graph G is assumed to be connected; if G is disconnected, then we apply the algorithm on each of G's connected components; it is important to observe that the subgraph induced by the vertices of an antihole is connected.

ANTIHOLE-DETECTION ALGORITHM

Input: a connected undirected graph G.
Output: a message whether G contains an antihole or not.

1. Initialize the entries of the arrays *visited_P3*[ ] and *in_path*[ ] to 0; compute the adjacency matrix of G;
2. `for` each vertex u of G `do`
   2.1 *in_path*$[u] \leftarrow 1$;
   2.2 `for` each edge vw of G `do`
   `if` u is adjacent neither to v nor to w `and`
   visited_P3$[(v,w),u] = 0$
   `then` *in_path*$[v] \leftarrow 1$;
   process(v,u,w);
   in_path$[v] \leftarrow 0$;
   2.3 *in_path*$[u] \leftarrow 0$;
3. Print that G does not contain an antihole.

where the procedure *process*( ) is as follows:

process(a,b,c)

1. *in_path*$[c] \leftarrow 1$;
2. *visited_P3*$[(a,c),b] \leftarrow 1$;
   visited_P3$[(c,a),b] \leftarrow 1$;
3. `for` each vertex d adjacent to b in G `do`
   3.1 `if` d is adjacent to a and non-adjacent to c in G
   `then` {*abcd is a* P_4 *of* $\overline{G}$}
   3.2 `if` *in_path*$[d] = 1$
   `then` print that G has an antihole; exit;
   `else if` *visited_P3*$[(b,d),c] = 0$
   `then` *process*(b,c,d);
4. *in_path*$[c] \leftarrow 0$;

Note that for a call *process*(a,b,c), a and c are adjacent in G, while b is adjacent to neither a nor c. So, if there exists a vertex d such that d is adjacent to a and b and not adjacent to c, then the vertices a,b,c,d induce the P_4 $abcd$ in $\overline{G}$.

The correctness of the algorithm is established as in the case of the hole-detection algorithm of the previous section.

Time and space complexity. Similarly to the case of the hole-detection algorithm, we obtain the results stated in Lemma 4.1 and Theorem 4.1 below; observe that if we ignore

the time taken by the recursive calls, the execution of the call *process*(a, b, c) takes $O(|N(b)|+1)$ time, and that each quadruple of vertices a, b, c, d where abc is a P_3 of $\overline{G}$ and d is adjacent to b in G is uniquely characterized by the ordered pair $((a, c), (b, d))$, where (a, c) and (b, d) are ordered pairs of adjacent vertices in G.

LEMMA 4.1. *Let G be a connected undirected graph on n vertices and m edges. Then the proposed algorithm determines whether G contains an antihole using $O(m^2)$ time and $O(nm)$ space.*

THEOREM 4.1. *Let G be an undirected graph on n vertices and m edges. Then there is an algorithm to determine whether G contains an antihole using $O(n + m^2)$ time and $O(nm)$ space.*

4.1. *Providing a Certificate.* Similarly to the hole-detection algorithm, the above algorithm can be easily augmented so that it provides a certificate whenever it decides that the input graph G contains an antihole. In this case as well, we use the modified array *in_path*[ ], which is described in Section 3.1. If the algorithm concludes that G contains an antihole, then the condition in Step 3.2 of the procedure *process*( ) during the execution of a call, say, *process*(a, b, c, k), is found true for some vertex d; suppose that d is located in the jth position of the current active-path. Then the vertices located in positions $j, j+1, \ldots, k$ of the path form a cycle in $\overline{G}$ satisfying the conditions in the statement of Lemma 3.1.

Since an antihole is the complement of a hole, the observations in Section 3.1 imply that, in order to isolate an antihole, we need to find a pair of vertices of the cycle, say at positions i and i', such that the vertices are not adjacent in G, are not consecutive in the cycle, and the length-value $|i - i'|$ is minimized. To do that efficiently, we use an auxiliary array *pathvertex*$[1 \,..\, k]$ whose entries correspond to the vertices in the active-path, and we work as follows:

(i) The length-value is initialized to $k - j$ corresponding to the pair of vertices c, d (which are located in positions k and j of the active-path, respectively), and the entries of the array *pathvertex*[ ] are initialized to 0.

(ii) For each vertex v of G such that v is in position i of the active-path, where $j \leq i \leq k-4$, we do the following: we set to 1 the entries of the array *pathvertex*[ ] corresponding to the neighbors of v; next, we find the leftmost entry equal to 0 (if any) in the subarray *pathvertex*$[i + 4 \,..\, k]$; if this is the entry t corresponding to vertex u, and if the difference $t - i$ is less than the current length-value, we update the length-value to $t - i$ and the associated pair of vertices to v, u; finally, we reset to 0 the entries of the array *pathvertex*[ ] corresponding to the neighbors of v, so that all the entries of the array are again equal to 0.

(iii) If the two vertices associated with the computed minimum length-value are located in positions $i_{\min}$ and $i_{\max}$ of the active-path, where $j \leq i_{\min} < i_{\max} \leq k$, then the vertices in the positions $i_{\min}, i_{\min}+1, \ldots, i_{\max}$ induce an antihole in G.

It is important to observe that the size of the array *pathvertex*[ ] is $O(n)$, since the vertices in the active-path are distinct. The correctness of the computation again follows from the proof of Lemma 3.1. Its time complexity is $O(n+m)$: the initialization assignments take

$O(n)$ time while the processing of each vertex takes time proportional to the number of its neighbors, thanks to the adjacency-list representation of the graph and the array *in_path*[ ]. The additional space required is $O(n)$. Therefore, the following theorem holds:

THEOREM 4.2. *Let G be an undirected graph on n vertices and m edges. The antihole-detection algorithm presented in this section can be augmented so that it outputs an antihole as a certificate whenever G contains one. The certificate computation takes $O(n+m)$ time and $O(n)$ space.*

4.2. *Detecting Antiholes on at Least k Vertices.* The approach used in the algorithm in this section can be generalized to yield an algorithm for detecting antiholes on at least $k \geq 5$ vertices; see also Section 3.2. Since the number of $\overline{P}_\ell$s of a graph on n vertices and m edges is $O(m^\alpha)$ if $\ell = 2\alpha$ and $O(n\,m^\alpha)$ if $\ell = 2\alpha + 1$, as with Corollary 3.3, we have the following result:

COROLLARY 4.1. *Let G be an undirected graph on n vertices and m edges, and let $k \geq 5$ be a constant. Then there is an algorithm to determine whether G contains an antihole on at least k vertices using $O(n\,m^{p-1})$ time if $k = 2p$, and $O(n+m^p)$ time if $k = 2p+1$.*

5. Detecting Antiholes in Graphs that Do Not Contain a C_5. In this section we use a different approach to solve the problem of detecting antiholes (and holes), which yields efficient algorithms for graphs that do not contain C_5s. In particular, for each edge $e = xy$ of the input graph G, we consider the partition of the vertices of G into the sets $\{x, y\}$, $A(e; x)$, $A(e; y)$, $A(e)$, and $V(G) - N[e]$, and we try to determine whether paths $ayuxb$ in the complement $\overline{G}$, where $a \in A(e; x)$, $u \in V(G) - N[e]$, and $b \in A(e; y)$, can be extended to form an antihole in G. This approach helps us detect antiholes in general graphs if we take advantage of the following property:

LEMMA 5.1. *Let G be an undirected graph. Then G contains an antihole if and only if there exists an edge $e = xy$ of G and a vertex $u \in V(G) - N[e]$ such that in the complement of the subgraph of G induced by $N(e) \cap N(u)$ there exists a path from a vertex in $A(e; x)$ to a vertex in $A(e; y)$.*

PROOF. ($\Longrightarrow$) Suppose that G contains an antihole and let this be the complement of the hole $v_0 v_1 \cdots v_k$, where $k \geq 4$. Then the vertices v_1 and v_k are adjacent in G; let e be the edge of G connecting them. Then $v_0 \in V(G) - N[e]$, $v_2 \in A(e; v_k)$, $v_{k-1} \in A(e; v_1)$, and $\{v_2, \ldots, v_{k-1}\} \subseteq N(e) \cap N(v_0)$; these and the fact that the sequence $v_{k-1}, \ldots, v_2$ induces a path in $\overline{G}$ imply that the conditions of the lemma hold for the edge $v_1 v_k$ of G and the vertex v_0.

($\Longleftarrow$) Suppose now that there exists an edge $e = xy$ of G and a vertex $u \in V(G) - N[e]$ such that in the complement of the subgraph $G[N(e) \cap N(u)]$ there exists a path from a vertex in $A(e; x)$ to a vertex in $A(e; y)$. Let $p_0 p_1 \cdots p_k$ be a shortest such path; that is, $p_0 \in A(e; x) \cap N(u)$, $p_k \in A(e; y) \cap N(u)$, $p_i \in A(e) \cap N(u)$ for all $i = 1, 2, \ldots, k-1$, and $p_i p_j \in E(G)$ for $0 \leq i < j - 1 \leq k - 1$. Then the

subgraph $G[\{x, u, y, p_0, p_1, \ldots, p_k\}]$ of G is the complement of a chordless cycle of length at least 5; in other words, G contains an antihole. □

Lemma 5.1 readily implies an antihole-detection algorithm which for a graph G on n vertices and m edges runs in $\Theta(nm^2)$ time in the worst case: for the appropriate pairs of an edge $e = xy$ and a vertex u, we compute the co-components of the subgraph $G[N(e) \cap N(u)]$ and check whether any of them contains vertices from both $A(e; x)$ and $A(e; y)$; as there may be as many as $\Theta(nm)$ such pairs, and for each one of them, $\Theta(n + m)$ time may be needed and suffices for the above-mentioned computations, the overall time complexity is $\Theta(nm^2)$. An improved algorithm can be obtained for graphs not containing C_5s, for which the following fact holds.

FACT 5.1. *Let G be an undirected graph which does not contain a C_5, and let $e = xy$ be an edge of G. Then for every pair of vertices a and b such that $a \in A(e; x)$, $b \in A(e; y)$, and $(N(a) \cap N(b)) - N[e] \neq \emptyset$, it holds that a and b are adjacent in G.*

PROOF. Let w be any vertex in $(N(a) \cap N(b)) - N[e]$; clearly, $wa, wb \in E(G)$, and $wx, wy \notin E(G)$. If the vertices a and b were not adjacent in G, then the subgraph of G induced by a, x, y, b, and w would be a C_5, a contradiction. □

In light of Fact 5.1, for any edge $e = xy$ and any vertex $u \in V(G) - N[e]$ of a graph G that does not contain a C_5, any path from a vertex in $A(e; x) \cap N(u)$ to a vertex in $A(e; y) \cap N(u)$ in the complement of the subgraph $G[N(e) \cap N(u)]$ is of length at least 2. Moreover, such a path contains a vertex in $A(e) \cap N(u)$; otherwise, the vertices of the path would belong to $A(e; x) \cup A(e; y)$, which implies that there exist vertices $a' \in A(e; x)$ and $b' \in A(e; y)$ such that a', b' are consecutive in the path, in contradiction to Fact 5.1. Then, instead of computing such a path we need only determine if there exist vertices $a \in A(e; x) \cap N(u)$, $b \in A(e; y) \cap N(u)$, and $v \in A(e) \cap N(u)$ such that v, a belong to the same co-component of the subgraph $G[N(x) \cap N(u)]$, and v, b belong to the same co-component of the subgraph $G[N(y) \cap N(u)]$; see Figure 1: C and D denote the co-components of the subgraphs $G[N(x) \cap N(u)]$ and $G[N(y) \cap N(u)]$ containing v and a, and v and b, respectively, and the dotted paths indicate chordless paths in the complements of these subgraphs. We show next that the existence of such vertices v, a, and b is equivalent to the existence of an antihole in G.

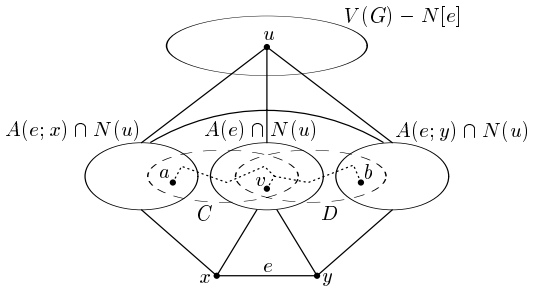

Fig. 1

LEMMA 5.2. *Let G be an undirected graph which does not contain a C_5. Then G contains an antihole if and only if there exists an edge $e = xy$ of G and vertices $u \in V(G) - N[e]$, $a \in A(e; x) \cap N(u)$, $b \in A(e; y) \cap N(u)$, and $v \in A(e) \cap N(u)$ such that a, v belong to the same co-component of the subgraph $G[N(x) \cap N(u)]$, and b, v belong to the same co-component of the subgraph $G[N(y) \cap N(u)]$.*

PROOF. ($\Longrightarrow$) Suppose that G contains an antihole. Since G does not contain a C_5, the antihole is of length at least 6; let it be the complement of the hole $v_0v_1 \cdots v_k$ ($k \geq 5$). The vertices v_1 and v_k are adjacent in G; if e is the edge of G connecting them, then $v_0 \in V(G) - N[e]$, $v_2 \in A(e; v_k) \cap N(v_0)$, $v_{k-1} \in A(e; v_1) \cap N(v_0)$, and $\{v_3, \ldots, v_{k-2}\} \subseteq A(e) \cap N(v_0)$. Since $k \geq 5$, it is easy to see that the conditions of the lemma hold for v_1, v_k, v_0, v_{k-1}, v_2, v_3 in place of x, y, u, a, b, v, respectively.

($\Longleftarrow$) Suppose now that there exists an edge $e = xy$ of G and vertices u, a, b, v as described in the statement of the lemma. Then in the complement of the subgraph of G induced by $N(e) \cap N(u)$ there exists a path from vertex $a \in A(e; x)$ to vertex $b \in A(e; y)$. Then, by Lemma 5.1, G contains an antihole. □

We give below a detailed description of the antihole-detection algorithm for a connected input graph G; for disconnected input graphs, we apply the algorithm on each of their connected components; recall that the subgraph induced by the vertices of an antihole is connected.

ANTIHOLE-DETECTION ALGORITHM FOR GRAPHS THAT DO NOT CONTAIN A C_5

Input: a connected undirected graph G that does not contain a C_5.
Output: a message whether G contains an antihole or not.

1. `for` each vertex u of G `do`
   - 1.1 `for` each vertex w not adjacent to u in G `do`
     - 1.1.1 compute the set $N_{u,w} = N(u) \cap N(w)$;
     - 1.1.2 compute the co-components of $G[N_{u,w}]$;
     - 1.1.3 store the set $N_{u,w}$ as a list of vertex records, ordered by vertex index, where each vertex $z \in N_{u,w}$ is associated with the representative $cc(N_{u,w}; z)$ of the co-component to which it belongs;
   - 1.2 `for` each edge $e = xy$ of G such that $x, y \notin N[u]$ (i.e., $u \in V(G) - N[e]$) `do`
     - 1.2.1 {*mark the co-components of $G[N_{u,x}]$ containing a vertex in $A(e; x)$*}
       `for` each vertex $w \in N_{u,x}$ `do`
       $mark1[w] \leftarrow 0$;
       `for` each vertex $w \in N_{u,x} - N_{u,y}$ `do`
       $mark1[cc(N_{u,x}; w)] \leftarrow 1$; {*mark the representative*}
     - 1.2.2 {*mark the co-components of $G[N_{u,y}]$ containing a vertex in $A(e; y)$*}
       `for` each vertex $w \in N_{u,y}$ `do`
       $mark2[w] \leftarrow 0$;

for each vertex $w \in N_{u,y} - N_{u,x}$ do
$mark2[cc(N_{u,y}; w)] \leftarrow 1$; {*mark the representative*}
1.2.3 for each vertex $v \in N_{u,x} \cap N_{u,y}$ do
if $mark1[cc(N_{u,x}; v)] = 1$ and $mark2[cc(N_{u,y}; v)] = 1$
then print that G contains an antihole; exit;
2. Print that G does not contain an antihole.

The correctness of the algorithm follows from Lemma 5.2: for a vertex u of G and an edge $e = xy$ such that $x, y \notin N[u]$, we have that $A(e; x) \cap N(u) = N_{u,x} - N_{u,y}$, $A(e; y) \cap N(u) = N_{u,y} - N_{u,x}$, and $A(e) \cap N(u) = N_{u,x} \cap N_{u,y}$; moreover, the condition "if $mark1[cc(N_{u,x}; v)] = 1$ and $mark2[cc(N_{u,y}; v)] = 1$" and the fact that the vertex v belongs to $N_{u,x} \cap N_{u,y}$ imply that in the complement of $G[N_{u,x}]$ there exists a path from v to a vertex in $A(e; x)$ and that in the complement of $G[N_{u,y}]$ there exists a path from v to a vertex in $A(e; y)$.

Time and space complexity. Let n and m be the number of vertices and edges of the input graph G; since G is connected, $n = O(m)$. Step 1.1.1 can be completed in $O(n)$ time, while the construction of $G[N_{u,w}]$ and the computation of its co-components can be done in $O(|N_{u,w}|^2)$ time [5], [12], [18]. Since $|N_{u,w}| \leq \min\{|N(u)|, |N(w)|\}$, we have that $|N_{u,w}|^2 \leq |N(u)| \cdot |N(w)|$; thus, for a vertex u of G, Step 1.1.2 takes $O(n) + \sum_w O(|N(u)| \cdot |N(w)|) = O(m\,|N(u)|)$ time.[2] The construction of the list storing $N_{u,w}$ in Step 1.1.3 can be done in $O(|N(u)|)$ time by traversing the adjacency list of u, by collecting those vertices that belong to $N_{u,w}$, and by updating the co-component representative information; the ordering by vertex index comes for free, had we sorted the adjacency lists of the vertices in G by vertex index, something which can be achieved in $O(n + m)$ time using radix sorting during a preprocessing phase.

The sorting of the lists representing the sets $N_{u,w}$ implies that determining which vertices belong to $N_{u,x} - N_{u,y}$, $N_{u,y} - N_{u,x}$, and $N_{u,x} \cap N_{u,y}$ can be achieved by simply traversing the lists for $N_{u,x}$ and $N_{u,y}$ in lockstep fashion. Then each execution of Step 1.2 takes $O(|N(u)|)$ time, since $N_{u,x}, N_{u,y} \subseteq N(u)$. Thus, for a vertex u of G, Step 1.2 takes $\sum_e O(|N(u)|) = O(m\,|N(u)|)$ time. Step 2 takes constant time. In total, the entire execution of the algorithm on G takes $O(\sum_u O(n + m + m\,|N(u)|)) = O(m^2)$ time.

We now turn to the space complexity of the algorithm. The adjacency-list representation of the input graph requires $O(n + m)$ space. For an iteration of the for loop in Step 1, we need: $O(n)$ space to store $N_{u,w}$ and the re-indexing arrays, and $O(n + m)$ space to store $G[N_{u,w}]$, both reusable in each iteration of the for loop in Step 1.1; $\sum_w O(1 + |N_{u,w}|) = O(n + m)$ space to store the list representations of the sets $N_{u,w}$ for all $w \in V(G) - N[u]$; $O(n)$ space for the arrays $mark1[\,]$ and $mark2[\,]$, reusable in each iteration of the for loop in Step 1.2. This space can be reused from iteration to iteration of the for loop in Step 1, so that the space complexity of the algorithm is $O(n + m)$.

In summary, our antihole-detection algorithm runs in $O(m^2)$ time using $O(n + m)$ space when applied on a connected undirected graph on n vertices and m edges. If

[2] Note that working on the subgraph $G[N_{u,w}]$ requires re-indexing of vertices, i.e., mapping the indices $1, 2, \ldots, n$ of vertices in G to the indices $1, 2, \ldots, |N_{u,w}|$ of vertices in $G[N_{u,w}]$ and vice versa; this can be done by using two arrays of $O(n)$ total space which take $O(n)$ time to initialize and constant time to answer each re-indexing request.

the input graph is disconnected, then we apply the algorithm on each of its connected components. Since the connected components of a graph can be computed in time and space linear in the size of the graph [10] and since these components are pairwise vertex- and edge-disjoint, we obtain the following result.

THEOREM 5.1. *Let G be an undirected graph on n vertices and m edges which does not contain a C_5. Then there is an algorithm to determine whether G contains an antihole using $O(n + m^2)$ time and $O(n + m)$ space.*

We note that the algorithm is applicable to graphs for which we know in advance that they do not contain a C_5. If the algorithm is applied on a general graph, then it exhibits the following behavior: if it answers that an antihole exists, then indeed the input graph contains an antihole (which however may be a C_5); if it answers that it does not contain an antihole, then the input graph contains no antihole on at least six vertices but it may contain a C_5.

5.1. *Providing a Certificate.* Like the previous algorithms, this algorithm too can be augmented so that it provides a certificate whenever it decides that the input graph G contains an antihole. In particular, if G contains an antihole, then whenever the algorithm finds that, for a vertex u and an edge $e = xy$ of G such that $x, y \notin N[u]$, there exists a vertex $v \in N_{u,x} \cap N_{u,y}$ for which $mark1[cc(N_{u,x}; v)] = 1$ and $mark2[cc(N_{u,y}; v)] = 1$, it executes the following in Step 1.2.3 before terminating:

(i) computes the subgraph $G[N(e) \cap N(u)]$;
(ii) uses a dummy vertex s and makes it adjacent to all the vertices of the subgraph except for those in $N_{u,x} - N_{u,y}$;
(iii) runs BFS on the *complement* of the resulting graph starting at s until a vertex, say, b, in $N_{u,y} - N_{u,x}$ is encountered;

It is not difficult to see that if the path on tree edges from s to b in the BFS-tree of Step (iii) is $sv_1v_2 \cdots v_kb$, then the vertices $x, u, y, v_1, \ldots, v_k, b$ induce an antihole in G of length at least 6 (since G does not contain a C_5, then $k \geq 2$ in accordance with Fact 5.1).

The computation of the adjacency-list representation of the subgraph $G[N(e) \cap N(u)]$ can be done in $O(n + m)$ time and space by using a copy of the adjacency-list representation of G and by removing from it all unnecessary lists and vertex records. The addition of the dummy vertex s can be done in $O(n)$ time and space. Executing BFS on the complement of the resulting graph can be done in time and space linear in the size of the graph, i.e., in $O(n + m)$ time and space (see [5], [12], and [18]). Finally, the path on tree edges needed to complete the antihole can be easily obtained in time linear in its length if the BFS-tree is represented by means of parent pointers. Therefore, we have:

THEOREM 5.2. *Let G be an undirected graph on n vertices and m edges which does not contain a C_5. The antihole-detection algorithm presented in this section can be augmented so that it provides a certificate that G contains an antihole, whenever it decides so of G. The certificate computation takes $O(n + m)$ time and space.*

REMARK 5.1 (Detecting Holes in Graphs Not Containing C_5s). Since an antihole is the complement of a hole and the complement of a C_5 is also a C_5, one can detect whether a graph G without a C_5 contains a hole by applying the above algorithm on its complement $\overline{G}$; this results into an $O(n^4)$-time and $O(n^2)$-space algorithm. The time complexity can be improved if the operation of the algorithm on $\overline{G}$ is interpreted in terms of G so that $\overline{G}$ is not constructed explicitly.

In general terms, the algorithm processes all the P_3s of G; for each such P_3 xuy, it tries to determine whether there exists a vertex w that is not adjacent to u, x, or y, and there exists a path from w in $G[M_{u,x}]$ to a vertex adjacent to y and a path from w in $G[M_{u,y}]$ to a vertex adjacent to x, where $M_{u,x}$ (resp. $M_{u,y}$) is the set of vertices which are adjacent neither to u nor to x (resp. neither to u nor to y), i.e., $M_{u,x} = (V(G) - N[u]) \cap (V(G) - N[x])$ and $M_{u,y} = (V(G) - N[u]) \cap (V(G) - N[y])$. Note that such a vertex w exists iff there exists a chordless path $v_1 v_2 \cdots v_k$, where $k \geq 3$, such that $v_1 \in N(y) - (N(u) \cup N(x))$, $v_k \in N(x) - (N(u) \cup N(y))$, and $v_i \notin N(u) \cup N(x) \cup N(y)$ for all $i = 2, 3, \ldots, k-1$; this is equivalent to the vertices $x, u, y, v_1, \ldots, v_k$ inducing a hole of length at least 6.

The algorithm is given in detail below. It is a variant of the antihole algorithm presented earlier in this section, where all the "adjacencies" have been replaced by "non-adjacencies" and vice versa.

HOLE-DETECTION ALGORITHM FOR GRAPHS NOT CONTAINING A C_5

Input: an undirected graph G which does not contain a C_5.
Output: a message whether G contains a hole or not.

1. `for` each vertex u of G `do`
   - 1.1 `for` each vertex v adjacent to u in G `do`
     - 1.1.1 compute the set $M_{u,v} = (V(G) - N[u]) \cap (V(G) - N[v])$;
     - 1.1.2 compute the connected components of $G[M_{u,v}]$;
     - 1.1.3 store the set $M_{u,v}$ as a list of vertex records, ordered by vertex index, where each vertex $w \in M_{u,v}$ is associated with the representative $comp(M_{u,v}; w)$ of the component to which it belongs;
   - 1.2 `for` each vertex x adjacent to u in G `do`
     - `for` each vertex y adjacent to u and not adjacent to x in G `do`
       - {*mark the conn.components of* $G[M_{u,x}]$ *containing a neighbor of* y *in* G}
       - `for` each vertex $w \in M_{u,x}$ `do`
         - $mark1[w] \leftarrow 0$;
       - `for` each vertex $w \in M_{u,x} - M_{u,y}$ `do`
         - `if` w is adjacent to y in G
         - `then` $mark1[comp(M_{u,x}; w)] \leftarrow 1$;
           {*mark the representative*}
       - {*mark the conn.components of* $G[M_{u,y}]$ *containing a neighbor of* x *in* G}
       - `for` each vertex $w \in M_{u,y}$ `do`
         - $mark2[w] \leftarrow 0$;

```
        for each vertex w ∈ M_{u,y} − M_{u,x} do
            if w is adjacent to x in G
            then mark2[comp(M_{u,y}; w)] ← 1;
                                    {mark the representative}
        for each vertex v ∈ M_{u,x} ∩ M_{u,y} do
            if mark1[comp(M_{u,x}; v)] = 1 and
               mark2[comp(M_{u,y}; v)] = 1
            then print that G contains a hole; exit;
```

2. Print that G does not contain a hole.

It is not difficult to see that the algorithm runs in $O(n^2m)$ time and requires $O(n^2)$ space.

This result indicates that the same approach results in a hole-detection algorithm which in the worst case proves asymptotically more time- and space-consuming than the corresponding antihole-detection algorithm. This seems to be due to the fact that checking whether a graph contains an antihole of length k requires that certain $\Theta(k^2)$ edges exist and that certain k edges are missing, whereas in the case of a hole of length k, one needs to verify that k edges exist and $\Theta(k^2)$ edges are missing; in the former case, the cost of checking the non-existence of the k edges can be paid for by the $\Theta(k^2)$ existing edges, something which does not hold in the latter case.

REMARK 5.2 (Extending the Approach to Detecting Antiholes/Holes on at Least k Vertices). The approach described in this section can be extended to yield an algorithm for the detection of antiholes on at least k vertices in graphs not containing antiholes on $k-1$ vertices. In this case the vertex x is "expanded" into a $\overline{P}_{k-5}$. Then, for each such $\overline{P}_{k-5}\ x_1x_2\cdots x_{k-5}$ and for each vertex y adjacent to all the vertices x_i, we work on the sets

$$A(x_1\cdots x_{k-5}, y; x) = \left(\bigcap_{1\le i\le k-5} N(x_i)\right) - N[y],$$

$$A(x_1\cdots x_{k-5}, y; y) = \left(\bigcap_{1\le i\le k-6} N(x_i)\right) \cap N(y) - N[x_{k-5}],$$

$$A(x_1\cdots x_{k-5}, y) = \left(\bigcap_{1\le i\le k-5} N(x_i)\right) \cap N(y)$$

(instead of $A(e; x)$, $A(e; y)$, and $A(e)$, respectively) and for vertices u in $(\bigcap_{i=2}^{k-5} N(x_i)) - (N(x_1)\cup N(y))$, so that the vertices $y, u, x_1, \ldots, x_{k-5}$ induce a $\overline{P}_{k-3}$ in the input graph G. Since the number of different choices of a $\overline{P}_{k-5}\ x_1x_2\cdots x_{k-5}$ and a vertex y adjacent to all the vertices x_i is $O(m^a)$ if $k-4=2a$ and $O(nm^a)$ if $k-4=2a+1$, then in a fashion similar to the one we used to prove Theorem 5.1, we can show the following corollary:

COROLLARY 5.1. *Let G be an undirected graph on n vertices and m edges which does not contain antiholes on $k-1$ vertices. Then there is an algorithm to determine whether*

G contains an antihole on at least k vertices using $O(n + m^{p-1})$ *time if* $k = 2p$, *and* $O(n\,m^{p-1})$ *time if* $k = 2p + 1$; *the algorithm requires* $O(n + m)$ *space.*

Similarly, for the case of holes, we "expand" vertex x into a P_{k-5} and we consider vertices u in $(N(x_1) \cap N(y)) - (\bigcup_{i=2}^{k-5} N(x_i))$ so that the vertices $y, u, x_1, \ldots, x_{k-5}$ induce a P_{k-3} in G; then, in light of Remark 5.1, we also have:

COROLLARY 5.2. *Let G be an undirected graph on n vertices and m edges which does not contain holes on* $k - 1$ *vertices. Then there is an algorithm to determine whether G contains a hole on at least k vertices using* $O(n^2\,m^{p-2})$ *time if* $k = 2p$, *and* $O(n\,m^{p-1})$ *time if* $k = 2p + 1$; *the algorithm requires* $O(n^2)$ *space.*

In both the antihole and the hole case, the time complexity is $O(n^{k-2})$.

6. Concluding Remarks. We have presented algorithms for detecting holes and antiholes in undirected graphs. For an input graph on n vertices and m edges, both algorithms run in $O(n + m^2)$ time and require $O(nm)$ space. The algorithms can be augmented so that they return a hole or an antihole, whenever such a structure exists in the graph, in $O(n + m)$ additional time and space. We have also described an antihole detection algorithm for graphs not containing a C_5 which runs in $O(n + m^2)$ time and requires only $O(n + m)$ space.

The obvious open problem is to design algorithms for finding a hole and/or an antihole in general graphs with improved time and/or space complexity; note that all the P_3s participating in P_4s of a graph on n vertices and m edges can be computed in $O(nm)$ time [19]. It is worth mentioning that $o(n + m^2)$-time algorithms for both problems would imply an improvement on the currently best algorithms for recognizing weakly chordal graphs [2], [17].

We also pose as an open problem the construction of $O(n + m^2)$-time algorithms for detecting whether a graph contains a C_5; the existence of a C_5 can be easily determined in $O(nm^2)$ time. None of our algorithms seems to be modifiable to handle this special case while maintaining the $O(n + m^2)$ time complexity. As we mentioned, due to our antihole-detection algorithm for graphs that do not contain a C_5, an $O(n + m^2)$-time and $O(n + m)$-space algorithm for detecting a C_5 would imply an antihole-detection algorithm of the same time and space complexity.

Finally, in light of the "strong perfect graph theorem" [8], it would be very interesting to come up with efficient algorithms for the detection of odd-length holes and/or odd-length antiholes in general graphs. Currently, the problem of detecting odd-length holes in general graphs is open; an algorithm has been proposed for "cleaned" graphs only which runs in $O(n^{10})$ time [11]. Regarding the difficulty of this problem, it is worth mentioning that the problem of determining whether a particular vertex participates in an odd-length hole is NP-complete [3]. On the other hand, algorithms are available for the detection of even-length holes [6], [9]; the fastest among them is claimed to run in $O(n^{15})$ time. Finally, if one is interested in detecting whether a graph contains an odd-length hole *or* an odd-length antihole, then there exists an $O(n^9)$-time algorithm for it [7].

Acknowledgments. The authors thank the anonymous referees for their suggestions which improved the presentation of the paper and resulted in the generalization of the results to holes/antiholes on at least k vertices; in particular, they are thankful to one of them for his/her encouragement to emphasize the connection between the algorithm in Section 3 and the implicit execution of DFS on the auxiliary graph G'. The second author also thanks R. Sritharan for helpful and interesting discussions.

References

[1] C. Berge, Färbung von Graphen deren sämtliche bzw. deren ungerade Kreise starr sind, *Wiss. Z. Martin-Luther-Univ. Halle-Wittenberg Mathe.-Natur.* 114–115, 1961.

[2] A. Berry, J.-P. Bordat, and P. Heggernes, Recognizing weakly triangulated graphs by edge separability, *Nordic J. Comput.* **7**, 164–177, 2000.

[3] D. Bienstock, On the complexity of testing for odd holes and induced odd paths, *Discrete Math.* **90**, 85–92, 1991.

[4] A. Brandstädt, V.B. Le, and J.P. Spinrad, *Graph Classes: A Survey*, SIAM Monographs on Discrete Mathematics and Applications, SIAM, Philadelphia, PA, 1999.

[5] K.W. Chong, S.D. Nikolopoulos, and L. Palios, An optimal parallel co-connectivity algorithm, *Theory Comput. Systems* **37**, 527–546, 2004.

[6] M. Chudnovsky, K. Kawarabayashi, and P. Seymour, Detecting even holes, *J. Graph Theory* **48**, 85–111, 2005.

[7] M. Chudnovsky, G. Cornuéjols, X. Liu, P. Seymour, and K. Vušković, Recognizing Berge graphs, *Combinatorica* **25**, 143–187, 2005.

[8] M. Chudnovsky, N. Robertson, P. Seymour, and R. Thomas, The strong perfect graph theorem, *Ann. of Math.* **164**, 51–229, 2006.

[9] M. Conforti, G. Cornuéjols, A. Kapoor, and K. Vušković, Even-hole-free graphs. Part II: Recognition algorithm, *J. Graph Theory* **40**, 238–266, 2002.

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

[11] G. Cornuéjols, X. Liu, and K. Vušković, A polynomial algorithm for recognizing perfect graphs, *Proc. 44th IEEE Symp. on Foundations of Computer Science* (*FOCS* 2003), pp. 20–27, 2003.

[12] E. Dahlhaus, J. Gustedt, and R.M. McConnell, Partially complemented representations of digraphs, *Discrete Math. Theoret. Comput. Sci.* **5**, 147–168, 2002.

[13] E. Eschen and R. Sritharan, Characterization of some graphs classes with no long holes, *J. Combin. Theory Ser. B* **65**, 156–162, 1995.

[14] M.C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.

[15] R.B. Hayward, Weakly triangulated graphs, *J. Combin. Theory Ser. B* **39**, 200–208, 1985.

[16] R.B. Hayward, Two classes of perfect graphs, Ph.D. Thesis, School of Computer Science, McGill University, 1987.

[17] R.B. Hayward, J. Spinrad, and R. Sritharan, Weakly chordal graph algorithms via handles, *Proc. 11th ACM–SIAM Symp. on Discrete Algorithms* (*SODA* 2000), pp. 42–49, 2000.

[18] H. Ito and M. Yokoyama, Linear time algorithms for graph search and connectivity determination on complement graphs, *Inform. Process. Lett.* **66**, 209–213, 1998.

[19] S.D. Nikolopoulos and L. Palios, Algorithms for P_4-comparability graph recognition and acyclic P_4-transitive orientation, *Algorithmica* **39**, 95–126, 2004.

[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] J.P. Spinrad, Finding large holes, *Inform. Process. Lett.* **39**, 227–229, 1991.

[22] J.P. Spinrad and R. Sritharan, Algorithms for weakly triangulated graphs, *Discrete Appl. Math.* **59**, 181–191, 1995.

[23] R.E. Tarjan and M. Yannakakis, Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs, *SIAM J. Comput.* **13**, 566–579, 1984.