Detecting an Odd Hole

MARIA CHUDNOVSKY, Princeton University
ALEX SCOTT, Mathematical Institute, University of Oxford, UK
PAUL SEYMOUR, Princeton University
SOPHIE SPIRKL, Rutgers University

We give a polynomial-time algorithm to test whether a graph contains an induced cycle with length more than three and odd.

CCS Concepts: • **Mathematics of computing → Graph algorithms**;

Additional Key Words and Phrases: Odd holes, perfect graphs, recognition algorithm

ACM Reference format:
Maria Chudnovsky, Alex Scott, Paul Seymour, and Sophie Spirkl. 2020. Detecting an Odd Hole. *J. ACM* 67, 1, Article 5 (January 2020), 12 pages.
https://doi.org/10.1145/3375720

1 INTRODUCTION

All graphs in this article are finite and have no loops or parallel edges. A *hole* of G is an induced subgraph of G that is a cycle of length at least four. In this article, we give an algorithm to test whether a graph G has an odd hole, with running time polynomial in $|G|$. ($|G|$ denotes the number of vertices of G.)

The study of holes, and particularly odd holes, grew from Claude Berge's "strong perfect graph conjecture" [1], that if a graph and its complement both have no odd holes, then its chromatic number equals its clique number. For many years, this was an open question, as was the question of finding a polynomial-time algorithm to test if a graph is perfect. ("Perfect" means every induced subgraph has chromatic number equal to clique number.) Both questions were settled at about the same time: in the early 2000's, two of us, with Robertson and Thomas [7], proved Berge's conjecture, and also, with Cornuéjols et al. [4], gave a polynomial-time algorithm to test if a graph has

This material is based upon work supported in part by the U. S. Army Research Office under Grant No. W911NF-16-1-0404 (Chudnovsky). This material is based upon work supported by the Air Force Office of Scientific Research (AFOSR) under Grant No. A9550-19-1-0187 (Seymour) and the National Science Foundation under Grant No. DMS-1763817 (Chudnovsky), Grant No. DMS-1800053 (Seymour), and Grant No. DMS-1802201 (Spirkl). Alex Scott is supported by a Leverhulme Trust Research Fellowship.
Authors' addresses: M. Chudnovsky, P. Seymour, and S. Spirkl, Dept of Mathematics, Princeton University, Fine Hall, Washington Rd, Princeton NJ 08544, USA; emails: {mchudnov, pds}@math.princeton.edu, sophie.spirkl@rutgers.edu; A. Scott's, Mathematical Institute, University of Oxford, Andrew Wiles Building, Woodstock Road, Oxford, OX2 6GG, UK; email: scott@maths.ox.ac.uk.

0004-5411/2020/01-ART5 $15.00
https://doi.org/10.1145/3375720

an odd hole or odd antihole, thereby testing perfection. (An *antihole* of G is an induced subgraph whose complement graph is a hole in the complement graph of G.)

Excluding both odd holes and odd antiholes is a combination that works well, and has "deep" structural consequences. Just excluding odd holes loses this advantage, and graphs with no odd holes seem to be much less well-structured than perfect graphs in some vague sense. It was only recently that two of us [16] proved that if a graph has no odd holes, then its chromatic number is bounded by a function of its clique number, resolving an old conjecture of Gyárfás [14]. For a long time, the question of testing for odd holes has remained open: while we could test for the presence of an odd hole or antihole in polynomial time, we were unable to separate the test for odd holes from the test for odd antiholes, and the complexity of testing for an odd hole remained open. Indeed, it seemed quite likely that testing for an odd hole was NP-complete; for instance, D. Bienstock [2, 3] showed that testing if a graph has an odd hole containing a given vertex is NP-complete. But, in fact, we can test for odd holes in polynomial time.

The main result of this article is the following:

Theorem 1.1. *There is an algorithm with the following specifications:*

Input: *A graph G.*
Output: *Determines whether G has an odd hole.*
Running time: $O(|G|^9)$.

Remarkably, the algorithm (or, rather, the proof of its correctness) is considerably simpler than the algorithm of Ref. [4], and so not only solves an open question, but gives a better way to test if a graph is perfect. Its running time is the same as the old algorithm for testing perfection. (A recent paper by Lai et al. [15] gives a modification of the algorithm of this article, which would bring the running time down to $O(|G|^8)$.)

Like the algorithm of Ref. [4] and several other algorithms that test for the presence of an induced subdivision of a fixed subgraph, the new algorithm has three stages. Say we are looking for an induced subgraph of "type T" in the input graph G.

- The algorithm first tests for certain "easily-detected configurations." These are configurations that can be efficiently detected and whose presence guarantees that G contains an induced subgraph of type T.
- The third step is an algorithm that tries to find a subgraph of type T directly; it would not be expected to work on a general input graph, but it would detect a subgraph of type T if there happens to be a particularly "nice" one in the input graph. For instance, in Ref. [4], we were looking for an odd hole, and the method was, try all triples of vertices and join them by three shortest paths, and see if they form an odd hole. This would normally not work, but it would work if for some shortest odd hole of the graph, there were no vertices in the remainder of the graph with more than three neighbours in this hole.
- The second step is to prepare the input for the third step; this step is called "cleaning" and is where the main challenges lie. In the cleaning step, the algorithm generates a "cleaning list," polynomially many subsets $X_1, \ldots, X_k$ of the vertex set of the input graph G, with the property that if G does in fact contain an induced subgraph of type T, then for some $i \in \{1, \ldots, k\}$, such a subgraph can be found in $G \setminus X_i$ using the method of step 3. This usually means that if G contains an induced subgraph of type T, then there is one, say H, such that some X_i contains all the vertices of $G \setminus V(H)$ that have many neighbours in H, and deleting X_i leaves H intact.

Cleaning was first used by Conforti and Rao [12] to recognize linear balanced matrices, and subsequently by Conforti et al. [11] to recognize balanced matrices, and by Conforti et al. [10]

to test for even holes, as well as in Ref. [4]. It then became a standard tool in induced subgraph detection algorithms [5, 6, 9]. This is the natural approach to try to test for an odd hole, and it seemed to have been explored thoroughly; but not thoroughly enough, as we shall see. We have found a novel method of cleaning that works remarkably well, and will have further applications [8, 13].

In both the old algorithm to check perfection, and the new algorithm of this article, we first test whether G contains a "pyramid" or "jewel" (easily-detected induced subgraphs that would imply the presence of an odd hole), and we may assume that it does not. Then, we try to search for an odd hole directly; to do so, we exploit the properties of an odd hole of minimum length, a so-called *shortest odd hole*. (These properties hold in graphs with no pyramid or jewel, but not in general graphs, which is why we test for pyramids and jewels first.)

Let C be a shortest odd hole. A vertex $v \in V(G)$ is *C-major* if there is no three-vertex path of C containing all the neighbours of v in $V(C)$ (and, consequently, $v \notin V(C)$); and C is *clean* (in G) if no vertices of G are C-major. If G has a shortest odd hole that is clean, then it is easy to detect that G has an odd hole, and this was done in theorem 4.2 of Ref. [4]. More exactly, there is a poly-time algorithm that either finds an odd hole, or deduces that no shortest odd hole of G is clean. Call this *procedure P*. The complicated part of the algorithm in Ref. [4] was generating a cleaning list, a list of polynomially-many subsets of $V(G)$, such that if G has an odd hole, then there is a shortest odd hole C and some X in the list such that $X \cap V(C) = \emptyset$, and every C-major vertex belongs to X. Given that, we take the list $X_1, \ldots, X_k$ say, and for each of the polynomially-many graphs $G \setminus X_i$, we run procedure P on it. If it ever finds an odd hole, then G has an odd hole and we are done. If not, then G has no odd hole and again we are done.

So, the key is generating the cleaning list. For this, Ref. [4] uses

THEOREM 1.2. *For every graph G not containing any "easily-detected configuration," if C is a shortest odd hole in G, and X is an anticonnected set of C-major vertices, then there is an edge uv of C such that u, v are both X-complete.*

(*Anticonnected* means connected in the complement, and *X-complete* means adjacent to every vertex in X.) But to arrange that 1.2 is true, it is necessary to expand the definition of "easily-detected configuration" to include some new configurations. It remains true that if one is present, then the graph has an odd hole or odd antihole and we can stop; if they are not present, then 1.2 is true. The problem is, if one of these new easily-detected configurations is present, it guarantees that G contains an odd hole or an odd antihole, but not necessarily an odd hole.

But there is a simpler way. Here is a rough sketch of a new procedure to clean a shortest odd hole C in a graph G with no pyramid or jewel. Let x be a C-major vertex such that there is a gap in C between two neighbours of x, as long as possible. (We can assume there is one.) Let the neighbours of x at the ends of this gap be d_1, d_2; thus, there is a path D of C between d_1, d_2 such that every C-major vertex either has a neighbour in its interior, or is adjacent to both ends. We can assume that the C-distance between d_1, d_2 is at least three. Also, we have a theorem that there is an edge f of C such that every C-major vertex nonadjacent to x is adjacent to one of the ends of f.

For the algorithm, what we do is we guess x, d_1, d_2, and f (more precisely, we enumerate all possibilities for them). Eventually, we will guess correctly. We also guess the two vertices neighbouring f in C, say c_1, c_4, where $f = c_2c_3$ and c_1, c_2, c_3, c_4 are in order in C. When we guess correctly, every C-major vertex either

—is adjacent to both d_1, d_2; or
—is different from c_1, c_2, c_3, c_4 and is adjacent to one of c_2, c_3; or
—is adjacent to x and has a neighbour in the interior of D.

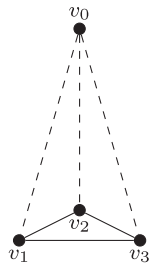

Fig. 1. A pyramid. Dashed lines represent paths of indeterminate length.

We can safely delete all common neighbours of d_1, d_2 except x; deleting these vertices will not remove any vertices of C. Also, we can safely delete all vertices different from c_1, c_2, c_3, c_4 that are adjacent to one of c_2, c_3. So now in the graph that remains after these deletions, say G', all C-major vertices different from x satisfy the third bullet above.

We do not know the path D, and so we cannot immediately identify the set of vertices satisfying the third bullet. (For this sketch, let us assume that D has length less than half that of C; if it is longer, there is a slight complication.) But we know (it is a theorem of Ref. [4]) that D is a shortest path between d_1, d_2 in the graph obtained from G' by deleting all C-major vertices; so, it is also a shortest path between d_1, d_2 in the graph G'' obtained from G' by deleting x and all its neighbours (except d_1, d_2). The algorithm computes G'', and then finds the union of the interiors of the vertex sets of all shortest paths between d_1, d_2 in G'', say F. It is another theorem of Ref. [4] that no vertex of $C \setminus V(D)$ has a neighbour in F; so it is safe to delete from G' all vertices of G' except d_1, d_2 that are not in F and have a neighbour in F. But then we have deleted all the C-major vertices, and now we just test for a clean shortest odd hole.

In an earlier version of this paper, we proved the result by a more complicated method that also seems to us novel and worth recording. It was necessary to first test for two more easily-detected configurations; but then, instead of constructing the set F above, the algorithm just guesses the component (F' say) of G'' that contains the interior of D, and deletes all neighbours of x that have neighbours in this component except d_1, d_2. This might delete some of the hole C, but we proved a theorem that enough of C remains that we can still use it in an algorithm to detect an odd hole. In particular, there is an odd path P of C of length at least three, with both ends adjacent to x such that the ends of P both have neighbours in F' and its internal vertices do not; and we can exploit this to detect the presence of an odd hole.

2 THE EASILY-DETECTED CONFIGURATIONS

Let $v_0 \in V(G)$, and for $i = 1, 2, 3$, let P_i be an induced path of G between v_0 and v_i such that

- P_1, P_2, P_3 are pairwise vertex-disjoint except for v_0;
- $v_1, v_2, v_3 \neq v_0$, and at least two of P_1, P_2, P_3 have length at least two;
- v_1, v_2, v_3 are pairwise adjacent; and
- for $1 \leq i < j \leq 3$, the only edge between $V(P_i) \setminus \{v_0\}$ and $V(P_j) \setminus \{v_0\}$ is the edge $v_i v_j$.

We call the subgraph induced on $V(P_1 \cup P_2 \cup P_3)$ a *pyramid*, with *apex* v_0 and *base* $\{v_1, v_2, v_3\}$ (See Figure 1). If G has a pyramid, then G has an odd hole (because two of the paths P_1, P_2, P_3 have the same length modulo two, and they induce an odd hole). It is shown in theorem 2.2 of Ref. [4] that:

Theorem 2.1. *There is an algorithm with the following specifications:*

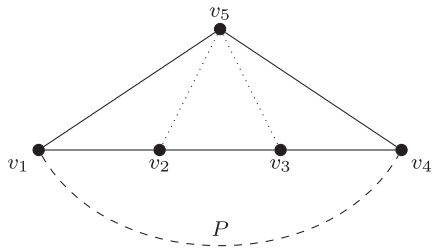

Fig. 2. A jewel. Dotted lines represent possible edges.

Input: *A graph G.*
Output: *Determines whether there is a pyramid in G.*
Running time: $O(|G|^9)$.

If $X \subseteq V(G)$, we denote the subgraph of G induced on X by $G[X]$. If X is a vertex or edge of G, or a set of vertices or a set of edges of G, we denote by $G \setminus X$ the graph obtained from G by deleting X. Thus, for instance, if b_1b_2 is an edge of a hole C, then $C \setminus \{b_1, b_2\}$ and $C \setminus b_1b_2$ are both paths, but one contains b_1, b_2 and the other does not. If P is a path, the *interior* of P is the set of vertices of the path P that are not ends of P.

We say that $G[V(P) \cup \{v_1, \ldots, v_5\}]$ is a *jewel* in G if $v_1, \ldots, v_5$ are distinct vertices, $v_1v_2, v_2v_3, v_3v_4, v_4v_5, v_5v_1$ are edges, v_1v_3, v_2v_4, v_1v_4 are nonedges, and P is a path of G between v_1, v_4 such that v_2, v_3, v_5 have no neighbours in the interior of P (See Figure 2). (We do not specify whether v_5 is adjacent to v_2, v_3, but if it is adjacent to one and not the other, then G also contains a pyramid.)

Again, if G contains a jewel, then it has an odd hole; and it is shown in theorem 3.1 of Ref. [4] that:

Theorem 2.2. *There is an algorithm with the following specifications:*

Input: *A graph G.*
Output: *Determines whether there is a jewel in G.*
Running time: $O(|G|^6)$.

It is proved in theorem 4.2 of Ref. [4] that:

Theorem 2.3. *There is an algorithm with the following specifications:*

Input: *A graph G containing no pyramid or jewel.*
Output: *Determines one of the following:*
(1) G contains an odd hole;
(2) there is no clean shortest odd hole in G.
Running time: $O(|G|^4)$.

Let us say a shortest odd hole C is *heavy-cleanable* if there is an edge uv of C such that every C-major vertex is adjacent to one of u, v. We deduce:

Theorem 2.4. *There is an algorithm with the following specifications:*

Input: *A graph G containing no pyramid or jewel.*
Output: *Determines one of the following:*
(1) G contains an odd hole;
(2) there is no heavy-cleanable shortest odd hole in G.
Running time: $O(|G|^8)$.

Proof. List all the four-vertex induced paths c_1-c_2-c_3-c_4 of G. For each one, let X be the set of all vertices of G different from $c_1, \ldots, c_4$ and adjacent to one of c_2, c_3. We test whether $G \setminus X$ has a clean shortest odd hole, by Theorem 2.3. If this never succeeds, output that G has no heavy-cleanable shortest odd hole.

To see correctness, note that if G has a heavy-cleanable shortest odd hole C, then C is clean in $G \setminus X$ for some X that we will test (assuming we have not already detected an odd hole); and when we do so, Theorem 2.3 will detect an odd hole. If G does not have a heavy-cleanable shortest odd hole, then two things might happen: either Theorem 2.3 detects an odd hole for some choice of X, or it never detects one. In either case, the output is correct. This proves Theorem 2.4. □

Let us say that a graph G is a *candidate* if it has no jewel or pyramid, and no heavy-cleanable shortest odd hole (and consequently no hole of length five). By combining the previous results, we deduce:

Theorem 2.5. *There is an algorithm with the following specifications:*

Input: *A graph G.*
Output: *Determines one of the following:*
(1) G contains an odd hole;
(2) G is a candidate.
Running time: $O(|G|^9)$.

In view of this, we just need to find a poly-time algorithm to test candidates for odd holes.

3 HEAVY EDGES

Let C be a graph that is a cycle, and let $A \subseteq V(C)$. An *A-gap* is a subgraph of C composed of a component X of $C \setminus A$, the vertices of A with neighbours in X, and the edges between A and X. (So if $|A| \geq 2$, the A-gaps, if they exist, are the paths of C of length ≥ 2, with both ends in A and no internal vertex in A.) The *length* of an A-gap is the number of edges in it (so if A consists just of two adjacent vertices, the A-gap has length $|E(C)| - 1$). We say that A is *normal* in C if every A-gap is even (and, consequently, if C has odd length, then $A \neq \emptyset$).

The following is proved in theorem 7.6 of Ref. [4]:

Theorem 3.1. *Let G be a graph containing no jewel or pyramid, let C be a shortest odd hole in G, and let X be a stable set of C-major vertices. Then the set of X-complete vertices in C is normal.*

Also we have:

Theorem 3.2. *If G is a graph containing no pyramid, and C is a shortest odd hole in G, then every C-major vertex has at least four neighbours in $V(C)$.*

Proof. Let v be C-major, and suppose it has at most three neighbours in $V(C)$. Let A be the set of neighbours of v in C. Every A-gap is even (since adding v gives a hole shorter than C), and since C is odd, some edge of C is not in a A-gap, that is, some two neighbours of v in $V(C)$ are adjacent. Since v is C-major, it has exactly three neighbours in $V(C)$, and they are not all three consecutive; but then $G[V(C) \cup \{v\}]$ is a pyramid, a contradiction. This proves 3.2. □

Theorem 3.3. *Let G be a graph containing no jewel or pyramid, let C be a shortest odd hole in G, and let x, y be nonadjacent C-major vertices. Then, every induced path between x, y with interior in $V(C)$ has even length.*

Proof. Let the vertices of C in order be $c_1, c_2, \ldots, c_n, c_1$. By 3.1 applied to $\{x, y\}$, some vertex of C is adjacent to both x, y, say c_n. Suppose that there is an odd induced path P between x, y

with interior in $V(C)$. Since x has only one neighbour in the interior of P, and x has at least four neighbours in $V(C)$, not all consecutive (because G contains no jewel), it follows that the interior of P has at most $n-4$ vertices, and so P has length at most $n-3$. But $c_n \notin V(P)$, and since adding c_n does not give an odd hole (because such an odd hole would be shorter than C), it follows that c_n has a neighbour in the interior of P. Thus, we may assume that the interior of P equals $\{c_1, c_2, \ldots, c_k\}$ for some even $k \geq 2$. If both x, y have a neighbour in the set $\{c_{k+2}, \ldots, c_{n-2}\}$, there is an induced path Q between x, y with interior in $\{c_{k+2}, \ldots, c_{n-2}\}$, and its union with one of x-c_n-y, P is an odd hole of length less than that of C, a contradiction. Thus, one of x, y has no neighbours in $\{c_{k+2}, \ldots, c_{n-2}\}$, say x. By 3.2, x has at least four neighbours in $V(C)$; so, it has exactly four and is adjacent to both c_{k+1}, c_{n-1}. Hence, the neighbours of x in $V(C)$ are c_{n-1}, c_n, c_{k+1} and exactly one of c_1, c_k. But k is even since P has odd length, so the path c_{k+1}-c_{k+2}-$\ldots$-c_{n-1} of C is odd; and since adding x does not make an odd hole shorter than C, it follows that c_{k+1}, c_{n-1} are adjacent, and so $k = n-3$. But then the four neighbours of x in C are consecutive, and so the subgraph induced on $V(C) \cup \{x\}$ is a jewel, a contradiction. This proves 3.3. □

If $X \subseteq V(G)$, we say an edge uv of G is *X-heavy* if $u, v \notin X$, and every vertex of X is adjacent to at least one of u, v. We need:

Theorem 3.4. *Let G be a graph containing no jewel or pyramid or 5-hole, and let C be a shortest odd hole in G. Let X be a set of C-major vertices, and let $x_0 \in X$ be nonadjacent to all other members of X. Then, there is an X-heavy edge in C.*

Proof. We proceed by induction on $|X|$. If X is stable, then, by 3.1, some vertex of C is X-complete (because the null set is not normal), and both edges of C incident with it are X-heavy, as required. We assume then that $x_1, x_2 \in X$ are adjacent. From the inductive hypothesis, some edge u_iv_i of C is $(X \setminus \{x_i\})$-heavy, for $i = 1, 2$; so we may assume that for $i = 1, 2$, x_i has no neighbour in u_iv_i (because, otherwise, u_iv_i is X-heavy). Consequently, u_1v_1 and u_2v_2 are distinct edges. Since x_0, x_1 both have neighbours in $\{u_2, v_2\}$, 3.3 implies that they have a common neighbour in $\{u_2, v_2\}$; so we may assume that x_0, x_1 are both adjacent to v_2, and, similarly, x_0, x_2 are both adjacent to v_1. Since the subgraph induced on $\{x_0, x_1, x_2, v_1, v_2\}$ is not a 5-hole, and $v_1 \neq v_2$, it follows that v_1, v_2 are adjacent. Since x_1 has no neighbour in $\{u_1, v_1\}$, it follows that $u_1 \neq v_2$, and, similarly, $u_2 \neq v_1$; so, u_1, v_1, v_2, u_2 are in order in C. We claim that v_1v_2 is X-heavy; for suppose not. Then there exists $x \in X$ nonadjacent to v_1, v_2; and so $x \neq x_0, x_1, x_2$. Since u_1v_1 is $(X \setminus \{x_1\})$-heavy, it follows that x is adjacent to u_1, and, similarly, to u_2; but then the subgraph induced on $\{x, u_1, v_1, v_2, u_2\}$ is a 5-hole, a contradiction. This proves 3.4. □

4 THE ODD HOLES ALGORITHM

We can now give the algorithm to detect an odd hole. We first present it in as simple a form as we can, but its running time will be $O(|G|^{12})$. Then, we show that with more care, we can bring the running time down to $O(|G|^9)$.

Let C be a hole and $x \in V(G) \setminus V(C)$. An *$x$-gap* is an induced path of C with length at least two, with both ends adjacent to x, and with its internal vertices nonadjacent to x. Thus, if P is an x-gap, then $G[V(P) \cup \{v\}]$ is a hole. We need the following theorem 4.1 of Ref. [4]:

Theorem 4.1. *Let G be a graph containing no jewel or pyramid, and let C be a clean shortest odd hole in G. Let $u, v \in V(C)$ be distinct and nonadjacent, and let L_1, L_2 be the two subpaths of C joining u, v, where $|E(L_1)| < |E(L_2)|$. Then:*

- *L_1 is a shortest path in G between u, v, and*
- *for every shortest path P in G between u, v, $P \cup L_2$ is a shortest odd hole in G.*

Here, then, is a preliminary version of the algorithm. We are given an input graph G. First, we apply the algorithm of Theorem 2.5, and we may assume it determines that G is a candidate.

Next, we enumerate all induced four-vertex paths c_1-c_2-c_3-c_4 of G, and all induced three-vertex paths d_1-x-d_2 of G. (They might overlap.) For each choice of c_1-c_2-c_3-c_4 and d_1-x-d_2, and each vertex d_3 of G (thus, we are checking all 8-tuples $(c_1, c_2, c_3, c_4, d_1, x, d_2, d_3)$), we do the following:

- Compute the set X_1 of all vertices adjacent to both d_1 and d_2 that are different from x, and compute the set X_2 of all vertices that are adjacent to one of c_2, c_3 and different from c_1, c_2, c_3, c_4. Check that $x \in X_2$, and none of $c_1, \ldots, c_4, d_1, d_2$ belongs to $X_1 \cup X_2$ (and if not, move on to the next 8-tuple). Let G' be the graph obtained from G by deleting $X_1 \cup X_2$. Compute the set Y of all vertices of G' that are different from and nonadjacent to x in G.
- If $d_3 \notin Y$, move on to the next 8-tuple. Otherwise, check that the distances in $G[Y \cup \{d_1, d_2\}]$ between d_1, d_3 and between d_2, d_3 are finite and equal (and if not, move on to the next 8-tuple).
- For each $y \in Y$, compute the distance in $G[Y \cup \{d_1, d_2\}]$ to d_1, to d_2, and to d_3. For $i = 1, 2$, let F_i be the set of all $y \in Y$ with the sum of the distances to d_i and to d_3 minimum, that is, the set of interiors of shortest paths in $G[Y \cup \{d_1, d_2\}]$ between d_3 and d_i. Let X_3 be the set of all vertices of G' different from d_1, d_2, d_3, x that are not in $F_1 \cup F_2$ and have a neighbour in $F_1 \cup F_2 \cup \{d_3\}$.
- Use the algorithm of 2.3 to determine either that $G \setminus (X_1 \cup X_2 \cup X_3 \cup \{x\})$ has an odd hole, or that it has no clean shortest odd hole. If it finds that there is an odd hole, we output this fact and stop. If after examining all choices of 8-tuple we have not found that there is an odd hole, we output that there is none and stop.

Let us see that this algorithm works correctly. (If x is a vertex of G, $N[x]$ denotes the set consisting of x and all its neighbours.) Certainly, if the input graph has no odd hole, then the output is correct; so we may assume that G is a candidate and C is a shortest odd hole in G. Since C is not heavy-cleanable, there is a C-major vertex x with an x-gap of length at least three; and so there is one, x, say, with an x-gap in C of maximum length at least three. Let this x-gap have ends d_1, d_2; so d_1, d_2 are adjacent to x, and there is a path D of C between d_1, d_2 such that x has no neighbour in its interior. Since x is C-major, the C-distance between d_1, d_2 is at least three (because the path of C joining d_1, d_2 different from D contains all the neighbours of v in $V(C)$). From the choice of x, every other C-major vertex is either adjacent to both d_1, d_2, or has a neighbour in the interior of D. Since C is a shortest odd hole, it follows that D has even length; let d_3 be its middle vertex.

By 3.4, there is an edge c_2c_3 of C such that all C-major vertices nonadjacent to x are adjacent to one of c_2, c_3. Let c_1-c_2-c_3-c_4 be a path of C. As the algorithm examines, in turn, each 8-tuple, it eventually will examine the 8-tuple $(c_1, c_2, c_3, c_4, d_1, x, d_2, d_3)$, and we will show that, in this case, the algorithm will determine that there is an odd hole.

Thus, suppose that the algorithm is now examining the "correct" 8-tuple. Let X_1, X_2, G', Y be as in the first bullet above. It follows that X_1, X_2 are disjoint from $V(C)$, and so C is a shortest odd hole in G'. Let $G'' = G' \setminus (N[x] \setminus V(C))$. Then $Y \subseteq V(G'')$ and C is a clean shortest odd hole in G''. Since d_3 is the middle vertex of D, the subpaths of D joining d_3 to d_1, d_2 both have length less than $|C|/2$; so, by the first statement of 4.1 applied to G'', these two subpaths are shortest paths joining their ends with interior in Y. Moreover, by the second statement of 4.1 applied to G'', for every choice of a shortest path L_i in $G'[Y \cup \{d_1, d_2\}]$ joining d_3, d_i (for $i = 1, 2$), no vertex of $L_i \setminus \{d_i\}$ belongs to or has a neighbour in $V(C) \setminus V(D)$; so, the set X_3 defined in the third bullet above contains no vertex in $V(C)$. But $X_1 \cup X_2 \cup X_3 \cup \{x\}$ contains every C-major vertex, and so C is a clean shortest odd hole in $G \setminus (X_1 \cup X_2 \cup X_3 \cup \{x\})$; hence, when we apply the algorithm of 2.3 to this subgraph, it will determine that it (and, hence, G) has an odd hole. This completes the proof of correctness.

For the running time: there are $|G|^8$ 8-tuples to check. For each one, the sequence of steps above takes time $O(|G|^4)$; so, the total running time is $O(|G|^{12})$.

Now, let us do it more carefully to reduce the running time. In order to explain the method, let us consider more closely a shortest odd hole C in a candidate G. As before, there is a C-major vertex x with an x-gap of length at least three; and so there is one, x, say, with an x-gap in C of maximum length at least three. Let this x-gap have ends d_1, d_2; so d_1, d_2 are adjacent to x, and there is a path D of C between d_1, d_2 such that x has no neighbour in its interior. By Theorem 3.4, there is an edge c_2c_3 of C such that both x and all C-major vertices nonadjacent to x are adjacent to one of c_2, c_3. Since x is adjacent to one of c_2, c_3, not both c_2, c_3 belong to the interior of D; and, since d_1, d_2 are nonadjacent, we may assume by exchanging d_1, d_2 or c_2, c_3 if necessary that c_2, c_3, d_1, x, d_2 are all distinct except that possibly $c_2 = d_1$. Now, there are six possibilities:

(1) $c_2 \neq d_1$ (and, hence, c_3 does not belong to the interior of D), and D has length less than $|C|/2$;
(2) $c_2 \neq d_1$ (and, hence, c_3 does not belong to the interior of D), and D has length more than $|C|/2$;
(3) $c_2 = d_1$, and c_3 does not belong to the interior of D, and D has length less than $|C|/2$;
(4) $c_2 = d_1$, and c_3 does not belong to the interior of D, and D has length more than $|C|/2$;
(5) c_3 belongs to the interior of D (and, hence, $c_2 = d_1$), and D has length less than $|C|/2$; and
(6) c_3 belongs to the interior of D (and, hence, $c_2 = d_1$), and D has length more than $|C|/2$.

Let us say C is of *type i* if it satisfies the the ith statement above, where $1 \leq i \leq 6$. (Thus, C may have more than one type.) To minimize running time, it seems best to design separate algorithms to test for the six types separately. We need the following lemma:

Theorem 4.2. *There is an algorithm with the following specifications:*

Input: *A graph G, and two disjoint subsets A, B of $V(G)$, and a vertex $h \notin A \cup B$ with no neighbour in $A \cup B$. Also, for each $v \in A \cup B$, an induced path R_v between v and h, containing no vertex in $A \cup B$ except v.*
Output: *Determines whether there exist $a \in A$ and $b \in B$ such that $R_a \cup R_b$ is an induced path.*
Running time: $O(|G|^3)$.

Proof. For each $a \in A$, compute the set S_a of all vertices of $G \setminus \{h\}$ that either belong to $V(R_a \setminus \{h\})$ or have a neighbour in this set. (This takes time $O(|G|^2)$ for each $a \in A$.) Now, for each $a \in A$ and each $b \in B$, test whether S_a is disjoint from $V(R_b \setminus \{h\})$. If so, $R_a \cup R_b$ is an induced path; otherwise, it is not. This proves 4.2. □

Now to handle the six types of shortest odd hole. We begin with:

Theorem 4.3. *There is an algorithm with the following specifications:*

Input: *A candidate G.*
Output: *Determines either that G has an odd hole, or that G has no shortest odd hole of type 1.*
Running time: $O(|G|^9)$.

Proof. List all 6-tuples $(c_2, c_3, d_1, x, d_2, d_3)$ of distinct vertices of G such that c_2c_3 is an edge, and d_1-x-d_2 is an induced path. Now, we test each such 6-tuple in turn, as follows. Let X_1 be the set of common neighbours of d_1, d_2 different from x, and let X_2 be the set of all vertices different from x, c_2, c_3, d_1, d_2 that are adjacent to one of c_2, c_3. Let C_1 be the set of all vertices of G different from c_2, c_3, x that are adjacent to c_2 and not to c_3, and let C_4 be the set that are adjacent to c_3 and not to c_2. Let G' be the graph obtained from G by deleting $X_1 \cup X_2 \cup (N[x] \setminus \{d_1, d_2\})$.

Find the distance in G' between d_1, d_2, say t. If t is infinite, move on to the next 6-tuple. If t is finite, for each $v \in V(G')$, compute the distance between v and d_i for $i = 1, 2$ (setting the distance to be infinite if there is no path), and let Y be the set of $v \in V(G)$ different from d_1, d_2 with the sum of these two distances equal to t. Let X_3 be the set of vertices of G different from x, d_1, d_2 that are not in Y and have a neighbour in Y.

Let $G'' = G \setminus (X_1 \cup X_2 \cup X_3 \cup \{x\})$. For each $v \in C_1 \cup C_4$, if there is a path of G between v and d_3 such that all its vertices except v belong to $V(G'')$, find such a path R_v of minimum length. Let C_1' be the set of $v \in C_1$ such that R_v exists and has even length, and let C_1'' be the set of $v \in C_1$ such that R_v exists and has odd length. Define C_4', C_4'' similarly. Apply 4.2 to test whether there exist $c_1 \in C_1'$ and $c_4 \in C_4'$ such that the paths R_{c_1}, R_{c_4} are both defined and have union an induced path between c_1, c_4 with interior in $V(G'')$. If so, output that G has an odd hole. Otherwise, apply 4.2 again to test whether there exist $c_1 \in C_1''$ and $c_4 \in C_4''$ such that the paths R_{c_1}, R_{c_4} are both defined and have union an induced path between c_1, c_4 with interior in $V(G'')$. If so, output that G has an odd hole. Otherwise, move on to the next 6-tuple. When all 6-tuples have been tested, if no odd hole is found, return that G has no shortest odd hole of type 1.

To see the correctness, suppose that C is a shortest odd hole of type 1 in G, and let c_2, c_3, d_1, d_2, x, D be as in the definition of type. Let d_3 be the vertex of C that is the middle vertex of the even path $C \setminus \{c_2, c_3\}$. When the algorithm tests the 6-tuple $(c_2, c_3, d_1, x, d_2, d_3)$, let X_1, X_2, C_1, C_4 be as in the description of the algorithm. From the choice of c_2, c_3, every C-major vertex nonadjacent to x belongs to X_2, and so no vertex in $V(G')$ is C-major in G. Let Z be the set of C-major vertices in G. Then, $V(G') \cap Z = \emptyset$. From the first assertion of 4.1, D is a shortest path between d_1, d_2 in $G \setminus Z$, of length t, say. Since C has type 1, and from the choice of D, it follows that $X_1 \cup X_2 \cup (N[x] \setminus \{d_1, d_2\})$ is disjoint from $V(D)$. Since $V(D) \subseteq V(G') \subseteq V(G) \setminus Z$, it follows that D is a shortest path between d_1, d_2 in G'. If L is a shortest path between d_1, d_2 in G', then L is also a shortest path between d_1, d_2 in $G \setminus Z$ and, thus, by the second assertion of 4.1, no vertex in $V(C) \setminus V(D)$ has a neighbour in $V(L) \setminus \{d_1, d_2\}$; so, $X_3 \cap V(C) = \emptyset$. Let $c_1 \in C_1$ and $c_4 \in C_4$ such that c_1-c_2-c_3-c_4 is a path of C. Thus, $C \setminus \{c_1, c_4\}$ is a subgraph of G''. Moreover, the paths R_{c_1}, R_{c_4} exist, and by the first assertion of 4.1, they both have length $(|V(C)| - 3)/2$; and, by the second assertion of 4.1, the union of R_{c_1}, R_{c_4} is an induced path between c_1, c_4. Then, adding c_2, c_3 gives an odd hole, as required. This proves 4.3. □

The algorithms for type 3 and type 5 are small modifications of this.

Theorem 4.4. *There is an algorithm with the following specifications:*

Input: *A candidate G.*
Output: *Determines either that G has an odd hole, or that G has no shortest odd hole of type 3.*
Running time: $O(|G|^9)$.

Proof. Enumerate all 7-tuples $(c_1, c_3, c_4, d_1, x, d_2, d_3)$ of distinct vertices such that c_1-d_1-c_3-c_4 is an induced path and d_1-x-d_2 is an induced path. Set $c_2 = d_1$. For each such 7-tuple, let X_1 be the set of common neighbours of d_1, d_2, and let X_2 be the set of vertices different from c_1, c_2, c_3, c_4 that are adjacent to one of c_2, c_3. Let G' be the graph obtained from G by deleting $X_1 \cup X_2 \cup (N[x] \setminus \{c_1, d_2\})$.

Find the distance in G' between c_1, d_2, say, t. If t is infinite, move on to the next 6-tuple. If t is finite, for each $v \in V(G')$, compute the distance between v, c_1 and between v, d_2, and let Y be the set of $v \in V(G)$ different from c_1, d_2 with the sum of these two distances equal to t. Let X_3 be the set of vertices of G different from x, c_1, d_2 that are not in Y and have a neighbour in Y.

Let $G'' = G \setminus (X_1 \cup X_2 \cup X_3 \cup \{x\})$. Find a shortest path between c_1, d_3 in G'' and a shortest path between c_4, d_3 in G'', and test whether their union is an odd induced path between c_1, c_4. If

so, output that G has an odd hole; otherwise, move on to the next 7-tuple. If no odd hole is found after testing all 7-tuples, output that G has no odd hole of type 3. The proof of correctness is like that for 4.3, and we omit it. This proves 4.4. □

Similarly we have:

THEOREM 4.5. *There is an algorithm with the following specifications:*

Input: *A candidate G.*
Output: *Determines either that G has an odd hole or that G has no shortest odd hole of type 5.*
Running time: $O(|G|^9)$.

THEOREM 4.6. *There is an algorithm with the following specifications:*

Input: *A candidate G.*
Output: *Determines either that G has an odd hole, or that G has no shortest odd hole of type 2.*
Running time: $O(|G|^9)$.

PROOF. Enumerate all 6-tuples $(c_2, c_3, d_1, x, d_2, d_3)$ of distinct vertices where c_2c_3 is an edge and d_1-x-d_2 is an induced path. We test each 6-tuple in turn as follows. Let X_1 be the set of common neighbours of d_1, d_2 different from x, and let X_2 be the set of all vertices different from x, c_2, c_3, d_1, d_2 that are adjacent to one of c_2, c_3. Let C_1 be the set of all vertices of G different from c_2, c_3, x that are adjacent to c_2 and not to c_3, d_3, and let C_4 be the set that are adjacent to c_3 and not to c_2, d_3. Let G' be the graph obtained from G by deleting $X_1 \cup X_2 \cup (N[x] \setminus \{d_1, d_2\})$. If $d_3 \notin V(G')$, move on to the next 6-tuple. Find the distance in G' between d_1, d_3, and between d_2, d_3, and check that they are finite and equal (to t, say); if not, move on to the next 6-tuple. For each $v \in V(G')$, find its distance in G' to d_1, d_2, and d_3; let Y_1 be the set of $v \neq d_1, d_2$ with the sum of its distances to d_1, d_3 equal to t, and let Y_2 be the set of $v \neq d_1, d_2$ with the sum of its distances to d_2, d_3 equal to t. (Thus, $d_3 \in Y_1, Y_2$.) Let X_3 be the set of vertices of G different from x, d_1, d_2 that are not in Y and have a neighbour in Y.

Let $G'' = G \setminus (X_1 \cup X_2 \cup X_3 \cup \{x\})$. For each $v \in C_1$, if there is a path of G between v and d_1 such that all its vertices except v belong to $V(G'')$, find such a path R_v of minimum length. For each $v \in C_4$, if there is a path of G between v and d_2 such that all its vertices except v belong to $V(G'')$, find such a path R_v of minimum length. Let C_1' be the set of $v \in C_1$ such that R_v exists and has even length, and let C_1'' be the set of $v \in C_1$ such that R_v exists and has odd length. Define C_4', C_4'' similarly. Apply 4.2 in the graph obtained from $G[C_1 \cup C_4 \cup V(G'')]$ by identifying d_1, d_2, to test whether there exist $c_1 \in C_1'$ and $c_4 \in C_4'$ such that the paths R_{c_1}, R_{c_4} are both defined, disjoint, and have no edges joining them. If so, output that G has an odd hole. Otherwise, apply 4.2 again to test whether there exist $c_1 \in C_1''$ and $c_4 \in C_4''$ such that the paths R_{c_1}, R_{c_4} are both defined and are disjoint and have no edges joining them. If so, output that G has an odd hole. Otherwise, move on to the next 6-tuple. When all 6-tuples have been tested, if no odd hole is found, return that G has no shortest odd hole of type 2.

To see the correctness, suppose that C is a shortest odd hole of type 2 in G, and let c_2, c_3, d_1, d_2, x, D be as in the definition of type. Let d_3 be the vertex of C that is the middle vertex of the even path D. When the algorithm tests the 6-tuple $(c_2, c_3, d_1, x, d_2, d_3)$, let X_1, X_2, C_1, C_4 be as in the description of the algorithm; then, it follows from 4.1 as before that the output is correct. This proves 4.6. □

Similar modifications handle the remaining two cases, and we omit them. In summary, we have:

Theorem 4.7. *There is an algorithm with the following specifications:*

Input: *A candidate G.*
Output: *Determines whether G has an odd hole.*
Running time: $O(|G|^9)$.

Proof. If G has an odd hole, then it has a shortest odd hole of one of the six types; by running the algorithms just described, we can detect it. □

Our main result, 1.1, follows immediately from this and 2.5. One final remark: we have an algorithm to determine whether G has an odd hole, but what about actually finding an odd hole? One could obviously do this with an extra factor of $|G|$ in the running time, just by deleting vertices and running 4.7 repeatedly to find a maximal subset of the vertex set whose deletion does not destroy all odd holes. But we can do better, and, in fact, it is easy to adapt the current algorithm to find an odd hole instead of just detecting the existence of one, with running time $O(|G|^9)$. We omit the details.

REFERENCES

[1] Claude Berge. 1961. Färbung von Graphen, deren sämtliche bzw. deren ungerade Kreise starr sind. *Wiss. Z.* 10 (1961), 114–11.
[2] Dan Bienstock. 1991. On the complexity of testing for odd holes and induced odd paths. *Discrete Math.* 90, 1 (1991), 85–92.
[3] Dan Bienstock. 1992. Corrigendum: On the complexity of testing for odd holes and induced odd paths. *Discrete Math.* 102, 1 (1992), 109.
[4] Maria Chudnovsky, Gérard Cornuéjols, Xinming Liu, Paul Seymour, and Kristina Vušković. 2005. Recognizing Berge graphs. *Combinatorica* 25, 2 (2005), 143–186.
[5] Maria Chudnovsky and Rohan Kapadia. 2008. Detecting a theta or a prism. *SIAM J. Discrete Math.* 22, 3 (2008), 1164–1186.
[6] Maria Chudnovsky, Ken-ichi Kawarabayashi, and Paul Seymour. 2005. Detecting even holes. *J. Graph Theory* 48, 2 (2005), 85–111.
[7] Maria Chudnovsky, Neil Robertson, Paul Seymour, and Robin Thomas. 2006. The strong perfect graph theorem. *Ann. Math.* 164 (2006), 51–229.
[8] Maria Chudnovsky, Alex Scott, and Paul Seymour. 2019. Testing for a long odd hole. *arXiv preprint arXiv:1904.12273* (2019).
[9] Maria Chudnovsky, Paul Seymour, and Nicolas Trotignon. 2013. Detecting an induced net subdivision. *J. Comb. Theory, Ser. B* 103, 5 (2013), 630–641.
[10] Michele Conforti, Gérard Cornuéjols, Ajai Kapoor, and Kristina Vušković. 2002. Even-hole-free graphs part II: Recognition algorithm. *J. Graph Theory* 40, 4 (2002), 238–266.
[11] Michele Conforti, Gérard Cornuéjols, and M. R. Rao. 1999. Decomposition of balanced matrices. *J. Comb. Theory, Ser. B* 77, 2 (1999), 292–406.
[12] Michele Conforti and M. R. Rao. 1993. Testing balancedness and perfection of linear matrices. *Math. Program.* 61, 1–3 (1993), 1–18.
[13] Linda Cook and Paul Seymour. 2019. Testing for a long even hole. *in preparation* (2019).
[14] András Gyárfás. 1987. Problems from the world surrounding perfect graphs. *Applicationes Mathematicae* 19, 3–4 (1987), 413–441.
[15] Kai-Yuan Lai, Hsueh-I Lu, and Mikkel Thorup. 2019. Three-in-a-tree in near linear time. *arXiv preprint arXiv:1909.07446* (2019).
[16] Alex Scott and Paul Seymour. 2016. Induced subgraphs of graphs with large chromatic number. I. Odd holes. *J. Comb. Theory, Ser. B* 121 (2016), 68–84.

Received May 2019; revised November 2019; accepted December 2019