

Exact and Approximate Algorithms for Computing a Second Hamiltonian Cycle*

Argyrios Deligkas[†] George B. Mertzios[‡] Paul G. Spirakis[§] Viktor Zamaraev[¶]

Abstract

In this paper we consider the following total functional problem: Given a cubic Hamiltonian graph G and a Hamiltonian cycle C_0 of G , how can we compute a second Hamiltonian cycle $C_1 \neq C_0$ of G ? Cedric Smith proved in 1946, using a non-constructive parity argument, that such a second Hamiltonian cycle always exists. Our main result is an algorithm which computes the second Hamiltonian cycle in time $O(n \cdot 2^{(0.3-\varepsilon)n})$ time, for some positive constant $\varepsilon > 0$, and in polynomial space, thus improving the state of the art running time for solving this problem. Our algorithm is based on a fundamental structural property of Thomason’s lollipop algorithm, which we prove here for the first time. In the direction of approximating the length of a second cycle in a Hamiltonian graph G with a given Hamiltonian cycle C_0 (where we may not have guarantees on the existence of a second Hamiltonian cycle), we provide a linear-time algorithm computing a second cycle with length at least $n - 4\alpha(\sqrt{n} + 2\alpha) + 8$, where $\alpha = \frac{\Delta-2}{\delta-2}$ and δ, Δ are the minimum and the maximum degree of the graph, respectively. This approximation result also improves the state of the art.

Keywords: Hamiltonian cycle, cubic graph, exact algorithm, approximation algorithm.

1 Introduction

Graph Hamiltonicity problems are among the most fundamental problems in theoretical computer science. Problems related to Hamiltonian paths and Hamiltonian cycles have attracted a tremendous amount of work over the years, see for example the recent survey of Gould [10] and the references therein. Deciding whether a given graph has a Hamiltonian cycle, i.e. a cycle that contains each vertex once, was among Karp’s 21 NP-hard problems [12]. On the other hand, there are several exponential-time algorithms for computing a Hamiltonian cycle or a solution to the Traveling Salesman Problem (TSP), which is a direct generalization of the Hamiltonian cycle problem. The first algorithms for the problem were based on dynamic programming and required $O(n^2 2^n)$ time [2, 11]. One of the next major improvements came decades later by Eppstein [7] who showed that a Hamiltonian cycle in a graph of degree at most three with n vertices can be computed in $O(2^{\frac{n}{3}}) \approx 1.26^n$ time and linear space; at the same time the algorithm can also compute an optimum solution for TSP on such graphs. The algorithm of Eppstein works by forcing specific edges of the graph which must be part of any generated cycle; a variation of this algorithm can also enumerate all Hamiltonian cycles in a graph of degree at most three in $O(2^{\frac{3n}{8}})$ time [7]. After that, there has been a series of improvements on the running time for TSP and the Hamiltonian cycle problem in degree-three graphs; the most recent results are an $O(1.2553^n)$ -time algorithm by Liśkiewicz and Schuster [14] and an $O^*(2^{\frac{3n}{10}}) \approx 1.2312^n$ -time algorithm by Xiao and Nagamochi [19], where $O^*(\cdot)$ suppresses polynomial factors. By allowing exponential space, the

*Supported by the NeST initiative of the EEE/CS School of the University of Liverpool and by the EPSRC grants EP/P020372/1 and EP/P02002X/1.

[†]Department of Computer Science, Royal Holloway University of London, UK. Email: argyrios.deligkas@rhul.ac.uk

[‡]Department of Computer Science, Durham University, UK. Email: george.mertzios@durham.ac.uk

[§]Department of Computer Science, University of Liverpool, UK, and Computer Engineering & Informatics Department, University of Patras, Greece. Email: p.spirakis@liverpool.ac.uk

[¶]Department of Computer Science, University of Liverpool, Liverpool, UK. Email: viktor.zamaraev@liverpool.ac.uk

running time for solving TSP on degree-three graphs can be improved further [4]. For general graphs there exists a randomized algorithm with running time $O(1.657^n)$, given by Bjorklund [3].

On the other hand, using a non-constructive parity argument, Cedric Smith and William Tutte [17] proved in 1946 that, for any fixed edge in a cubic (i.e. 3-regular) graph G , there exists an even (potentially zero) number of Hamiltonian cycles through this edge. Thus, the existence of a first Hamiltonian cycle guarantees the existence of a *second* one too, and this allows us to define the following total functional problem [15].

SMITH

Input: A cubic Hamiltonian graph G and a Hamiltonian cycle C_0 of G .

Task: Compute a second Hamiltonian cycle $C_1 \neq C_0$ of G .

It is easy to see that any algorithm \mathcal{A} for the Hamiltonian cycle (decision) problem on graphs with maximum degree three can be trivially adapted to solve SMITH as follows: for every edge e of the initial Hamiltonian cycle C_0 , run \mathcal{A} on $G \setminus e$, i.e. on the graph obtained by removing e from G . Then, as a second Hamiltonian cycle $C_1 \neq C_0$ always exists, at least one of these n calls of \mathcal{A} will return such a cycle C_1 . That is, SMITH can be solved in $n \cdot T(\mathcal{A})$ time, where $T(\mathcal{A})$ is the worst-case running time of \mathcal{A} on input graphs with n vertices.

Thomason [16] was the first one who provided an algorithm, known as the *lollipop algorithm*, for SMITH. This algorithm starts from the given Hamiltonian cycle C_0 of G and creates a sequence of distinct Hamiltonian paths where the last of these Hamiltonian paths trivially augments to a different Hamiltonian cycle of G . This algorithm was actually used by Papadimitriou to place SMITH within the complexity class PPA [15]. Although Thomason’s lollipop algorithm is well-known for decades, the internal structure of the algorithm’s execution on cubic Hamiltonian graphs remains so far mostly unclear and not well understood. In an attempt to construct worst-case instances for the lollipop algorithm, Cameron proved in 2001 [5] that on a specific family of cubic graphs (which is a variation of the family introduced by Krawczyk [13]) the lollipop algorithm runs in time at least 2^{cn} , for some constant c . Thus, the state of the art running time for computing a second Hamiltonian cycle in SMITH is to use the best known algorithm for the Hamiltonian cycle problem in cubic graphs which runs in $O^*(2^{\frac{3n}{10}})$ [19]. However, a tantalizing longstanding question is whether the knowledge of the first Hamiltonian cycle C_0 *strictly helps* to reduce the running time for computing a second Hamiltonian cycle C_1 . In this paper we provide evidence for the *affirmative* answer to this question.

A relaxation of SMITH is, given Hamiltonian cycle C_0 , to efficiently compute a second cycle (different than C_0) that is large enough. This relaxed problem becomes more meaningful for graphs with degrees larger than three, as it is well known that uniquely Hamiltonian graphs (i.e. graphs with a unique Hamiltonian cycle) exist, even when when all vertices have degree three except two vertices which have degree four [6, 8]. For cubic Hamiltonian graphs, Bazgan, Santha, and Tuza [1] showed that the knowledge of the first Hamiltonian cycle C_0 algorithmically strictly helps to approximate the length of a second cycle. In fact, if C_0 is not given along with the input, there is no polynomial-time constant-factor approximation algorithm for finding a long cycle in cubic graphs, unless $P=NP$. In contrast, if C_0 is given, then for every $\varepsilon > 0$ a cycle $C' \neq C_0$ of length at least $(1-\varepsilon)n$ can be found in $2^{O(1/\varepsilon^2)} \cdot n$ time, i.e. there is a linear-time PTAS for approximating the second Hamiltonian cycle [1]. The main ingredient in the proof of the latter result is an $O(n^{\frac{3}{2}} \log n)$ -time algorithm which, given G and C_0 , computes a cycle $C' \neq C_0$ of length at least $n - 4\sqrt{n}$ [1]. In wide contrast to cubic graphs, for graphs of minimum degree at least three, only existential proofs are known for a second large cycle. In particular, Girão, Kittipassorn, and Narayanan recently proved with a *non-constructive* argument that any n -vertex Hamiltonian graph with minimum degree at least 3 contains another cycle of length at least $n - o(n)$ [9].

Our contribution. In this paper we do the first attempt to understand the internal structure of the lollipop algorithm of Thomason [16]. Our main result in this direction embarks from the following trivial observation, which is not specific to Thomason’s algorithm.

Observation 1 *Let G be a cubic Hamiltonian graph and let C_0, C_1 be any two different Hamiltonian cycles of G . Then the symmetric difference $C_0 \Delta C_1$ of the edges of the two cycles is a 2-factor, i.e. a collection of cycles in G .*

Although Observation 1 determines that the symmetric difference of any two Hamiltonian cycles C_0 and C_1 is a collection of cycles in G , it does not rule out the possibility that $C_0 \Delta C_1$ contains more than one cycle. Our first technical contribution is that, for any given Hamiltonian cycle C_0 , there exists at least one other Hamiltonian cycle C_1 such that $C_0 \Delta C_1$ is *connected*, i.e. it contains exactly one cycle. More specifically, we prove that this holds for the particular Hamiltonian cycle C_1 that is computed by Thomason’s lollipop algorithm when starting from the cycle C_0 . For our proof we simulate the execution of the lollipop algorithm by simultaneously assigning to every edge one of four distinct colors in a specific way such that four coloring invariants are maintained. Using this coloring procedure, an alternating red-blue path is maintained during the execution of the algorithm, which becomes an alternating red-blue *cycle* at the end of the execution. As it turns out, this alternating cycle coincides with the symmetric difference $C_0 \Delta C_1$.

This fundamental structural property of the lollipop algorithm (see Theorem 2 in Section 3) has never been revealed so far, and it enables us to design a novel and more efficient algorithm for detecting a second Hamiltonian cycle of G . This improves the current state of the art in the computational complexity of SMITH (see Section 4). Instead of trying to generate the second Hamiltonian cycle C_1 directly from C_0 (as Thomason’s lollipop algorithm does), our new algorithm enumerates –almost– all alternating red-blue cycles, until it finds one alternating cycle D such that the symmetric difference $C_0 \Delta D$ is a Hamiltonian cycle of G (and not just a collection of cycles that collectively contain all vertices of G). During its execution, this algorithm iteratively has a choice between two different options for the next edge to be colored red, in which cases it branches to create two new instances. However, in order for the algorithm to achieve a strictly better worst-case running time than $O^*(2^{\frac{3n}{10}})$, it has to refrain from just always blindly branching to new instances. We are able to do this by identifying appropriate disjoint quadruples of edges, which we call *ambivalent quadruples*, and by *deferring* the choice for the colors of each of these quadruples until the very end. Then, at the last step of the algorithm we are able to choose their colors in nearly-linear time. That is, using the ambivalent quadruples we do not generate *all* possible alternating red-blue cycles but only a succinct representation of them. The running time of the algorithm that we achieve eventually is $O(n \cdot 2^{(0.3-\varepsilon)n})$ time for some strictly positive $\varepsilon > 0$. In the particular case where the input graph G contains no induced cycle C_6 on 6 vertices, the exponent becomes 0.2971925.

In the direction of approximating the length of a second cycle on graphs with minimum degree δ and maximum degree Δ , we provide in Section 5 a linear-time algorithm for computing a cycle $C' \neq C_0$ of length at least $n - 4a(\sqrt{n} + 2\alpha) + 8$, where $\alpha = \frac{\Delta-2}{\delta-2}$. On the one hand, this improves the results of [1] in two ways. First, it provides a direct generalization to arbitrary Hamiltonian graphs of degree at least 3. Second, our algorithm works in linear time in n for all constant-degree regular graphs; in particular it works in time $O(n)$ on cubic graphs (see Corollary 2). On the other hand, we complement the results of [9] as we provide a *constructive* proof for their result in case where the Δ and δ are $o(\sqrt{n})$ -factor away from each other. Formally, our algorithm constructs in linear time another cycle of length $n - o(n)$ whenever $\frac{\Delta}{\delta} = o(\sqrt{n})$ (see Corollary 3).

2 Preliminaries

Given a graph $G = (V, E)$, an edge between two vertices u and v is denoted by $uv \in E$, and in this case u and v are said to be *adjacent* in G . The *neighborhood* of a vertex $v \in V$ is the set $N(v) = \{u \in V : uv \in E\}$ of its adjacent vertices. A graph G is cubic if $|N(v)| = 3$ for every vertex $v \in V$. Given a path $P = (v_1, v_2, \dots, v_k)$ (resp. a cycle $C = (v_1, v_2, \dots, v_k, v_1)$) of G , the *length* of P (resp. C) is the number of its edges. Furthermore, $E(P)$ (resp. $E(C)$) denotes the set of edges of the path P (resp. of the cycle C). A path P (resp. cycle C) in G is a *Hamiltonian* path (resp. *Hamiltonian* cycle) if it contains each vertex of G exactly once. Every cubic Hamiltonian graph is referred to as a *Smith graph*. Given a Smith graph G and a Hamiltonian cycle C_0 of G , an edge of G which does not belong to C_0 is called a *chord* of C_0 , or simply a *chord*. The next theorem

allows us to assume without loss of generality that the input Smith graph G is triangle-free.

Lemma 1 *Let $G = (V, E)$ be a Smith graph with n vertices that contains at least one triangle, and let C_0 be a Hamiltonian cycle of G . Then either there exists a second Hamiltonian cycle C_1 of G that contains only two different edges than C_0 , or there exists a Smith graph G' with $n - 2$ vertices such that every Hamiltonian cycle in G corresponds bijectively to a Hamiltonian cycle in G' (or both).*

Proof. Let $C_0 = (v_1, v_2, \dots, v_n)$. As G contains at least one triangle, there must exist a vertex v_i such that $v_{i-1}v_{i+1} \in E$, i.e. the vertices v_{i-1}, v_i, v_{i+1} build a triangle. If $v_iv_{i+2} \in E$, then there exists a second Hamiltonian cycle C_1 of G , in which the edges $v_{i-1}v_i, v_iv_{i+1}, v_{i+1}v_{i+2}$ are replaced by the edges $v_{i-1}v_{i+1}, v_{i+1}v_i, v_iv_{i+2}$. Similarly, if $v_iv_{i-2} \in E$, then there exists a second Hamiltonian cycle C_1 of G , in which the edges $v_{i-2}v_{i-1}, v_{i-1}v_i, v_iv_{i+1}$ are replaced by the edges $v_{i-2}v_i, v_iv_{i-1}, v_{i-1}v_{i+1}$. In both cases, the second Hamiltonian cycle C_1 of G contains only two different edges than C_0 .

Now suppose that $v_iv_{i+2}, v_iv_{i-2} \in E$. We prove the statement of the lemma for the Smith graph G' that is obtained from G by removing vertices v_{i-1} and v_{i+1} and by connecting v_i to v_{i-2} and v_{i+2} . Let C be an arbitrary Hamiltonian cycle of G . If C contains the edges $v_{i-1}v_{i-2}$ and $v_{i-1}v_i$, then it is easy to see that C must also contain the edges v_iv_{i+1} and $v_{i+1}v_{i+2}$, and thus C corresponds to a Hamiltonian cycle C' of G' in a straightforward way. Otherwise, if C contains the edges $v_{i-1}v_{i-2}$ and $v_{i-1}v_{i+1}$, then it is easy to see that C must also contain the edges v_iv_{i+1} and $v_iv_i^*$, and thus in this case C also corresponds to a Hamiltonian cycle C' of G' in a straightforward way. The third case, where C contains the edges $v_{i-1}v_i$ and $v_{i-1}v_{i+1}$ is symmetric to the previous case. Thus every Hamiltonian cycle C of G corresponds to one Hamiltonian cycle C' of G' .

Conversely, let C' be a Hamiltonian cycle of G' . If C' contains the edges $v_{i-2}v_i$ and v_iv_{i+2} , then we can create a Hamiltonian cycle C of G by replacing these edges with the edges $v_{i-2}v_{i-1}, v_{i-1}v_i, v_iv_{i+1}, v_{i+1}v_{i+2}$. If C' contains the edges $v_{i-2}v_i$ and $v_iv_i^*$, then we can create a Hamiltonian cycle C of G by replacing these edges with the edges $v_{i-2}v_{i-1}, v_{i-1}v_{i+1}, v_iv_{i+1}, v_iv_i^*$. The third case where C' contains the edges v_iv_{i+2} and $v_iv_i^*$ is symmetric to the previous case. Thus every Hamiltonian cycle C' of G' corresponds to one Hamiltonian cycle C of G . ■

The next theorem now follows immediately by repeatedly applying Lemma 1.

Theorem 1 *Let $G = (V, E)$ be a Smith graph with n vertices that contains at least one triangle, and let C_0 be a Hamiltonian cycle of G . In linear time we can compute either a second Hamiltonian cycle C_1 of G or a triangle-free Smith graph G' with fewer vertices such that every Hamiltonian cycle in G bijectively corresponds to a Hamiltonian cycle in G' .*

3 A connected symmetric difference of the two Hamiltonian cycles

In this section we present the fundamental structural property of Thomason's lollipop algorithm that the symmetric difference of the two involved Hamiltonian cycles is connected. For the sake of presentation, in this section we simulate Thomason's lollipop algorithm [16] on an arbitrary given Smith graph G and, during this simulation, we assign colors to some of the edges of G . In particular, we assign to some edges of G one of the colors *red*, *blue*, *black*, and *yellow*. Note that the colors of the edges change in every step of the lollipop algorithm. Furthermore, every such (partial) edge-coloring of G uniquely determines one step of the lollipop algorithm on G that starts at a specific initial configuration.

Thomason's lollipop algorithm starts (at Step 0) with a Hamiltonian cycle $C_0 = (v_1, v_2, \dots, v_n, v_1)$; at this step we color all n edges of C_0 *black*, while all other edges are colored *yellow*. Any Step $i \geq 1$ of the lollipop algorithm is called *non-final* if the Hamiltonian path at this step does not correspond to a Hamiltonian cycle, i.e. v_1 is not connected in G to the last vertex of this Hamiltonian path.

Step 1 is derived from Step 0 by *removing* the edge v_1v_n from the cycle C_0 , thus obtaining the Hamiltonian path $P_1 = (v_1, v_2, \dots, v_n)$. We color this removed edge v_1v_n *red*. Let $N(v_n) = \{v_1, v_{n-1}, v_k\}$. At Step 2, the

lollipop algorithm continues by *adding* to the current Hamiltonian path P_1 the edge $v_n v_k$, thus obtaining a “lollipop” in which v_k keeps all its three incident edges, v_1 keeps only the incident edge $v_1 v_2$, and every other vertex keeps exactly two of its incident edges. Step 2 is completed by *removing* the edge $v_k v_{k+1}$ from P_1 , thus “breaking” the lollipop and obtaining the next Hamiltonian path $P_2 = (v_1, v_2, \dots, v_k, v_n, \dots, v_{k+1})$. It is important to note here that v_{k+1} is the vertex *immediately after* vertex v_k in the path P_{i-1} , where we consider that the path starts at v_1 . At Step 2 we color the newly added edge $v_n v_k$ *blue* and the removed edge $v_k v_{k+1}$ *red*, while the last vertex of the path P_2 is v_{k+1} . The algorithm continues towards Step 3 by adding to P_2 the third edge incident to v_{k+1} (i.e. the unique incident edge $v_{k+1} v_\ell$ different from the edges $v_k v_{k+1}$ and $v_{k+1} v_{k+2}$ that belonged to the previous path P_1) and by removing again the other incident edge of v_ℓ that “breaks” the lollipop. Similarly to Step 2, in Step 3 we color the newly added edge $v_{k+1} v_\ell$ *blue* and the newly removed incident edge of v_ℓ *red*.

As the lollipop algorithm progresses, the (partial) coloring of the edges of G continues, according to the following rules at Step $i \geq 1$. Recall that the Hamiltonian path at Step $i \geq 1$ is denoted by P_i . Furthermore, assume that during Step i , the path P_i is obtained by *adding* to P_{i-1} the edge $v_x v_y$ (where v_x is the last vertex of P_{i-1} , thus building a lollipop) and by subsequently *removing* from P_{i-1} the edge $v_y v_z$, thus breaking the constructed lollipop.

The description of the edge-coloring procedure that we apply at every step of the lollipop algorithm is formally given by the four coloring rules below. The intuitive description of these coloring rules is as follows. At every step, the black edges are those edges of the initial cycle C_0 which are still contained in the current Hamiltonian path, while the red edges are all the remaining edges of C_0 , i.e. those edges which do not belong to the current Hamiltonian path. The blue edges are those *chords* of C_0 that belong to the current Hamiltonian path. Finally, the yellow edges are all the remaining chords of C_0 , i.e. those chords that do not belong to the current Hamiltonian path. Initially we start with the cycle C_0 that contains n black edges and we remove one of them (the edge $v_1 v_n$) which becomes red. At every step of the algorithm we build the new *lollipop* when all three incident edges of some vertex v_y become either black or blue. This can happen either by adding a new (previously yellow) chord (thus coloring it blue) or by adding a new (previously colored red) C_0 -edge (thus coloring it black). Once we have build the new lollipop, we break it within the same step of the lollipop algorithm, either by removing a (previously colored black) C_0 -edge (thus coloring it red) or by removing a (previously colored blue) chord (thus coloring it yellow).

Coloring Rule 1 *If $v_x v_y \in C_0$ then we color $v_x v_y$ black (in this case $v_x v_y$ must be colored red at Step $i - 1$).*

Coloring Rule 2 *If $v_x v_y \notin C_0$ then we color $v_x v_y$ blue (in this case $v_x v_y$ must be yellow at Step $i - 1$).*

Coloring Rule 3 *If $v_y v_z \in C_0$ then we color $v_y v_z$ red (in this case $v_y v_z$ must be black at Step $i - 1$).*

Coloring Rule 4 *If $v_y v_z \notin C_0$ then we color $v_y v_z$ yellow (in this case $v_y v_z$ must be colored blue at Step $i - 1$).*

As we prove, the coloring of the edges proceeds such that the following invariants are maintained:

Invariant 1 *During every non-final Step $i \geq 1$, every C_0 -edge is colored either black or red.*

Invariant 2 *During every non-final Step $i \geq 1$, every non- C_0 -edge is either blue or yellow.*

Invariant 3 *At the end of every non-final Step $i \geq 1$, the set of all black and blue edges form a Hamiltonian path of G .*

Invariant 4 *When the lollipop is built during any non-final Step $i \geq 2$, the set of all red and blue edges form an alternating path of even length in G , starting at v_1 with a red edge. Furthermore, at the final step (i.e. when we build a second Hamiltonian cycle instead of a lollipop) the set of all red and blue edges form an alternating cycle D in G .*

The next observations follow immediately from the Coloring Rules 1-4 and by the proof of correctness of the Thomason’s lollipop algorithm [16].

Observation 2 *Invariants 1 and 2 are maintained at every non-final Step $i \geq 1$ of Thomason's lollipop algorithm.*

Observation 3 *At every non-final Step $i \geq 1$, the set of black and blue edges are exactly the edges of the Hamiltonian path at this step of Thomason's lollipop algorithm, and thus Invariant 3 is maintained.*

In the next theorem we prove the maintenance of Invariant 4, which is our main technical contribution in this section.

Theorem 2 *Invariant 4 is maintained at every (final or non-final) Step $i \geq 1$ of Thomason's lollipop algorithm. Thus, after the final step of the algorithm, the symmetric difference $C_0 \Delta C_1$ of C_0 with the produced Hamiltonian cycle C_1 is the alternating red-blue cycle D .*

Proof. The proof is done by induction on the number of steps of the lollipop algorithm. Invariant 4 is clearly true at Step 2 of the algorithm. In fact, when the lollipop is built during Step 2, there is only one red edge (i.e. the edge v_1v_n) and one blue edge that is incident to v_n , thus creating together a red-blue alternating path of length 2. In the induction hypothesis, assume that Invariant 4 is true until the non-final Step $i \geq 2$. Let v_x be the last vertex of the Hamiltonian path P_{i-1} and let v_xv_y be the newly added edge that creates the lollipop during Step i (recall that v_xv_y is either blue or black). Furthermore, let v_yv_z be the edge of the path P_{i-1} that is removed in order to break the lollipop, i.e. v_yv_z is either a previously black C_0 -edge that is colored red, or a previously blue chord that is yellow. Recall here that, by construction of the lollipop algorithm, v_z is the vertex *immediately after* vertex v_y in the path P_{i-1} , where the path starts at v_1 . Then v_z becomes the last vertex of the Hamiltonian path P_{i+1} at Step $i+1$. Finally, let v_zv_w be the newly added (blue or black) edge that creates the lollipop during Step $i+1$. Note that $v_w = v_1$ if and only if Step $i+1$ is the final step.

Case 1. v_xv_y is a newly added *blue* edge at Step i , i.e. a blue chord. Then, by the induction hypothesis, none of the other two incident edges of v_y is blue at Step i , and thus they are both black as they both belong to the path P_{i-1} . Therefore v_y is the last vertex of the red-blue alternating path at the time that the lollipop is built at Step i . Furthermore, the edge v_yv_z is colored *red* when the lollipop is broken at Step i , according to Coloring Rule 3. Now we distinguish the two cases for the color of the newly added edge v_zv_w during Step $i+1$.

Case 1a. v_zv_w is colored *blue* during Step $i+1$, i.e. v_zv_w is a yellow chord at Step i . First let $v_w \neq v_1$, i.e. Step $i+1$ is not a final step. Then, by the induction hypothesis, none of the other two incident edges of v_w is blue, and thus they are both black as they both belong to the path P_i . In this case, the alternating red-blue path at Step i (which ends at the vertex v_y with the blue edge v_xv_y) is extended by the red edge v_yv_z and the blue edge v_zv_w . Now let $v_w = v_1$, i.e. Step $i+1$ is the final step. Then, since until this step vertex v_1 is an endpoint of the red-blue alternating path, the addition of the blue edge $v_zv_w = v_zv_1$ at the final Step $i+1$, thus forming an alternating cycle (containing vertex v_1 and the initial red edge v_1v_n). This proves the induction step in Case 1a, see Figure 1(a).

Case 1b. v_zv_w is colored *black* during Step $i+1$, i.e. v_zv_w is a red C_0 -edge at Step i . Then, since v_z is the vertex *immediately after* vertex v_y in the path P_{i-1} , it follows that $v_z \neq v_1$. First suppose that $v_w = v_1$, i.e. that Step $i+1$ is the final step. Note that in this case $v_z = v_n$, since v_1v_n is the only red edge incident to v_1 until Step i . Furthermore, since v_yv_z is colored *red* when the lollipop is broken at Step i , it follows that in this case $v_y = v_{n-1}$. Note that, by the induction hypothesis, v_z is incident to a blue chord at Step i , since $v_z = v_n$ is the second vertex of the red-blue alternating path with even length. Thus, since v_yv_z becomes red at the end of Step i and v_zv_w becomes black at the final Step $i+1$, it follows that at Step $i+1$ the red and blue edges form an alternating cycle (containing the red edge $v_yv_z = v_{n-1}v_n$ and not containing vertex v_1), thus proving the induction step, see the final step in Figure 1(b).

Now suppose (within Case 1b) that $v_w \neq v_1$, i.e. that Step $i+1$ is non-final. Recall by the induction hypothesis that, until building the lollipop at Step i , the only red edge that is not incident to two blue edges is the first red edge, i.e. v_1v_n . Therefore, since $v_z, v_w \neq v_1$ and v_zv_w is red at Step i , it follows that both v_z

and v_w are incident to a blue chord at Step i . Thus, since at Step i the edge $v_y v_z$ changes color from black to red, while at Step $i + 1$ the edge $v_z v_w$ changes its color from red to black, it follows by the induction hypothesis that during Step $i + 1$ (when we break the lollipop) the red and blue edges form an alternating path in G , starting at v_1 with a red edge and ending at v_w with a blue edge. This proves the induction step.

Case 2. $v_x v_y$ is a newly added *black* edge at Step i , i.e. a C_0 -edge that was previously colored red. Note that $v_x, v_y \neq v_1$, since Step i is not the final step by the induction hypothesis. Denote the three neighbors of v_y in G by v_x, v_z , and v_q , where v_z is the vertex *immediately after* vertex v_y in the path P_{i-1} . Since v_x is the last vertex of P_{i-1} and $v_x v_y$ is red at the end of Step $i - 1$, it follows that one of the edges $v_y v_z$ and $v_y v_q$ is black and the other one is blue at the end of Step $i - 1$. Note that both edges $v_y v_z$ and $v_y v_q$ maintain their color after building the lollipop at Step i . Thus, since v_y has no red incident edge after building the lollipop at Step i , it follows by the induction hypothesis that vertex v_y is the last vertex of the red-blue alternating path.

Case 2a. $v_y v_z$ is *blue* (and $v_y v_q$ is black) at the beginning of Step i . Thus $v_y v_z$ becomes yellow when we break the lollipop at Step i . Furthermore, since $v_y v_z$ is a chord, the edge $v_z v_w$ is a C_0 -edge and it changes its color from red to black when we build the lollipop at Step $i + 1$. That is, the alternating red-blue path at Step i (which ends at v_z with the blue edge $v_y v_z$) shrinks by removing from it the blue edge $v_y v_z$ and the red edge $v_z v_w$.

Finally note that, if $v_w = v_1$, then we are left with no red and no blue edges after building the lollipop at Step $i + 1$, while Step $i + 1$ is the final step. However, the absence of red and blue edges at the final step implies that the obtained Hamiltonian cycle of the lollipop algorithm is the same as the initial Hamiltonian cycle C_0 , which is a contradiction to the correctness of the lollipop algorithm [16]. Therefore $v_w \neq v_1$, and thus Step $i + 1$ is a non-final step. This completes the proof of the induction step in Case 2a.

Case 2b. $v_y v_z$ is *black* (and $v_y v_q$ is blue) after building the lollipop at Step i , i.e. the alternating red-blue path ends at v_z with the blue edge $v_y v_q$. Then the edge $v_y v_z$ becomes *red* when we break the lollipop at the end of Step i . Furthermore recall that, at the time we build the lollipop at Step $i + 1$, the edge $v_z v_w$ is either a chord that becomes blue (from yellow) or a C_0 -edge that becomes black (from red).

First suppose that $v_z v_w$ is a chord that becomes *blue* (from yellow) when we build the lollipop at Step $i + 1$. If $v_w = v_1$ then Step $i + 1$ is the final step, and in this case the alternating red-blue path becomes an alternating cycle (containing vertex v_1 and the initial red edge $v_1 v_n$) when we build the lollipop at Step $i + 1$, which proves the induction step. Now let $v_w \neq v_1$, i.e. Step $i + 1$ is a non-final step. Then, since $v_z v_w$ is a yellow chord at Step i , it follows by the induction hypothesis that v_w is not incident to any red edge when we build the lollipop at Step $i + 1$. Thus, the alternating red-blue path at Step i (which ends at v_y with the blue edge $v_y v_q$) is augmented by adding to it the red edge $v_y v_z$ and the blue edge $v_z v_q$, which again proves the induction step.

Now suppose that $v_z v_w$ is a C_0 -edge that becomes *black* (from red) when we build the lollipop at Step $i + 1$. Let $v_w \neq v_1$ (i.e. Step $i + 1$ is non-final). Thus, since at Step i the edge $v_y v_z$ changes color from black to red, while at Step $i + 1$ the edge $v_z v_w$ changes its color from red to black, it follows by the induction hypothesis that during Step $i + 1$ (when we break the lollipop) the red and blue edges form an alternating path in G , starting at v_1 with a red edge and ending at v_w with a blue edge. This proves the induction step.

Finally let $v_w = v_1$, i.e. Step $i + 1$ is the final step. Then $v_z = v_n$, since $v_1 v_n$ is the only red edge incident to v_1 until Step i . Furthermore, since $v_y v_z$ is colored *red* when the lollipop is broken at Step i , it follows that in this case $v_y = v_{n-1}$. By the induction hypothesis, v_z is incident to a blue chord at Step i , since $v_z = v_n$ is the second vertex of the red-blue alternating path with even length. Thus, since $v_y v_z$ becomes red at the end of Step i and $v_z v_w = v_n v_1$ becomes black at the final Step $i + 1$, it follows that at Step $i + 1$ the red and blue edges form an alternating cycle (containing the red edge $v_y v_z = v_{n-1} v_n$ and not containing vertex v_1). This completes the proof of the induction step in Case 2b. ■

The next corollary follows by the proof of Theorem 2, and will allow us to reduce the asymptotic running time of our algorithm in Section 4 by a factor of n .

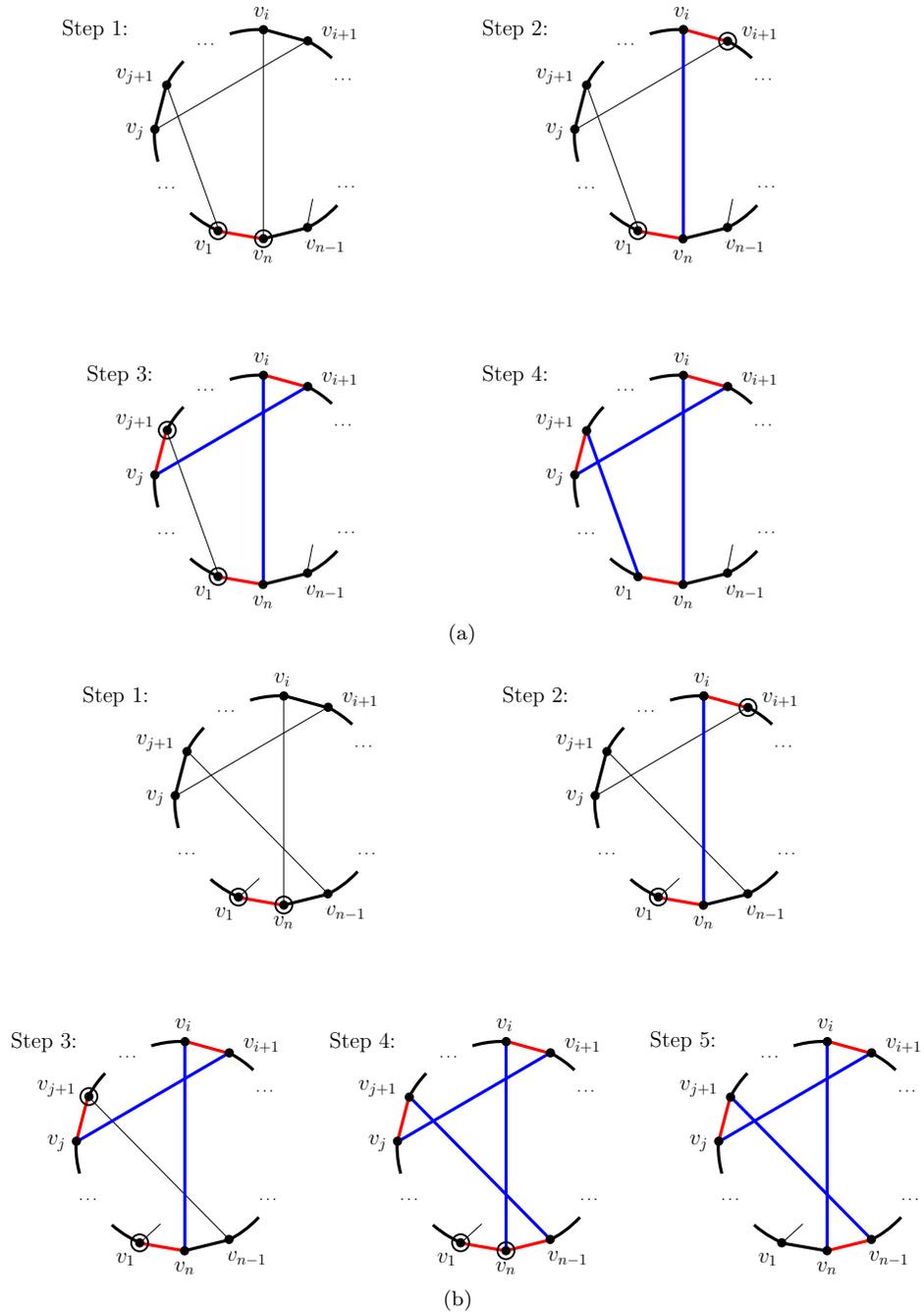


Figure 1: The edge-coloring during the execution of the lollipop algorithm in two example cubic Hamiltonian graphs. Every non-final Step $i \geq 2$ of the algorithm encompasses both building the new lollipop (with the blue edge) and breaking it (with the red edge), thus the illustrated red and blue edges always have more one red edge (i.e. the last red edge) than the alternating red-blue path of even length (see Theorem 2). At every non-final Step i of the algorithm, the endpoints of the corresponding Hamiltonian path P_i are illustrated by a circled vertex. In the example (a), vertex v_1 belongs to the alternating red-blue cycle, while in the example (b) vertex v_1 does not belong to it (see the final step in each example).

Corollary 1 *Let C_0 be a given Hamiltonian cycle of a Smith graph G . Let (v_i, v_j, v_k) be three consecutive vertices of C_0 . Then there exists a second Hamiltonian cycle C_1 of G such that (i) $C_0 \Delta C_1$ is a cycle in G and (ii) either the edge $v_i v_j$ or the edge $v_j v_k$ does not belong to C_1 .*

Proof. Part (i) of the corollary follows immediately by the statements of Theorem 2 and of Invariant 4. To prove part (ii) of the corollary, first note that, due to symmetry, we may denote without loss of generality $v_i = v_{n-1}$, $v_j = v_n$, and $v_k = v_1$.

Within the proof of Theorem 2, there are only two ways in which the alternating red-blue cycle can be built at the final step of Thomason’s lollipop algorithm. In the first way, the red-blue alternating cycle contains vertex v_1 and the red edge $v_1 v_n$ (this can happen only in Cases 1a and 2b). In the second way, the red-blue alternating cycle contains the red edge $v_{n-1} v_n$ but not vertex v_1 (this can happen only in Cases 1b and 2b). This completes the proof of the corollary. ■

4 The alternating cycles’ exploration algorithm

In this section we present our $O(n \cdot 2^{(0.3-\varepsilon)n})$ -time algorithm for SMITH, where $\varepsilon > 0$ is a strictly positive constant. This algorithm improves the state of the art, as it is asymptotically faster than all known algorithms for detecting a second Hamiltonian cycle in cubic graphs. Our algorithm is inspired from the structural property of Theorem 2. It starts from a designated vertex v_1 and constructs an alternating cycle D of red-blue edges (with respect to C_0 , in the terminology of Section 3) such that the symmetric difference $C_0 \Delta D$ is a Hamiltonian cycle C_1 of G . Equivalently, the algorithm constructs a second Hamiltonian cycle C_1 such that the symmetric difference $D = C_0 \Delta C_1$ is connected, i.e. one single cycle D of G in which every edge alternately belongs to C_0 and to C_1 , respectively.

Before we present and analyze our algorithm (Algorithm 1), we first present some necessary definitions and notation. Let G be a Smith graph and $C_0 = (v_1, v_2, \dots, v_n)$ be the initial Hamiltonian cycle of G . For every vertex v_i of G , we denote by v_i^* the unique vertex that is connected to v_i through a chord. That is, whenever $v_i v_j$ is a chord, we have that $v_j = v_i^*$ and $v_i = v_j^*$. Furthermore, every vertex v_i is incident to exactly two C_0 -edges $v_{i-1} v_i$ and $v_i v_{i+1}$, where we consider all indices modulo n . Algorithm 1 iteratively forces specific edges to be colored *red* (C_0 -edges not belonging to C_1), *black* (C_0 -edges belonging to C_1), *blue* (chords belonging to C_1), and *yellow* (chords not belonging to C_1). Initially, the algorithm starts by coloring the C_0 -edge $v_1 v_t$ *red*, where $v_t \in \{v_2, v_n\}$, the chord $v_t v_t^*$ *blue*, and the two C_0 -edges adjacent to the edge $v_t v_1$ *black*. That is, if $v_t = v_2$ (resp. if $v_t = v_n$) then the edges $v_1 v_n$ and $v_2 v_3$ (resp. $v_1 v_2$ and $v_{n-1} v_n$) are initially black. During its execution, the algorithm maintains an alternating red-blue path D of *even* length (starting with the red edge $v_1 v_t$ and ending with a blue edge), until D eventually becomes an alternating cycle. Note that D can only become a cycle when we color the chord $v_1 v_1^*$ blue. At every iteration the algorithm has (at most) two choices for the next red edge to be added to D , and thus it branches to (at most) two new instances of the problem, inheriting to both of them the choices of the forced (i.e. previously colored) edges made so far. At an arbitrary non-final step, let v_y be the last vertex of the alternating path D , and let $v_x v_y$ be the last (blue) edge of D . For each of the two C_0 -edges $v_{y-1} v_y$ and $v_y v_{y+1}$ that are incident to v_y , this edge is called *eligible* if it has not been forced (i.e. colored) at a previous iteration; otherwise it is called *non-eligible*. Here the term “eligible” stands for “eligible for branching”. We define the following operations; note that, once an edge has been assigned a color, it can *never* be forced to change its color.

- **Blue-Branch:** Whenever a chord $v_x v_y$ is colored blue (where v_y is the last vertex of the current red-blue alternating path D) and *both* C_0 -edges $v_y v_{y+1}, v_y v_{y-1}$ are eligible, we create two new instances I_1 and I_2 , where I_1 (resp. I_2) has the edge $v_y v_{y+1}$ (resp. $v_y v_{y-1}$) colored red and the edge $v_y v_{y-1}$ (resp. $v_y v_{y+1}$) colored black.
- **Blue-Force:** Whenever a chord $v_x v_y$ is colored blue (where v_y is the last vertex of the current red-blue alternating path D) and *exactly one* of the two C_0 -edges $v_y v_{y+1}, v_y v_{y-1}$ is eligible, we color this eligible C_0 -edge red.

- **Red-Force:** Assume that a C_0 -edge is colored red; note that this edge must be incident to a blue chord (i.e. its previous edge in the alternating path D). If its other incident chord is uncolored, we color it blue. Otherwise, if it has been previously colored yellow, we announce “contradiction”. Moreover, if this new red edge is incident to a C_0 -edge that is uncolored, we color this edge black.
- **Black-Force:** Assume that a C_0 -edge $v_i v_{i+1}$ is colored black, where this edge is adjacent to the (previously colored) black C_0 -edge $v_{i-1} v_i$ (resp. $v_{i+1} v_{i+2}$). If their commonly incident chord $v_i v_i^*$ (resp. $v_{i+1} v_{i+1}^*$) is so far uncolored, we color it yellow. Otherwise, if it has been previously colored blue, we announce “contradiction”.
- **Yellow-Force:** Assume that a chord $v_i v_i^*$ is colored yellow by the operation Black-Force (i.e. once both C_0 -edges $v_{i-1} v_i, v_i v_{i+1}$ become black); furthermore let $v_k = v_i^*$. If at least one of the C_0 -edges $v_{k-1} v_k, v_k v_{k+1}$ has been previously colored red, we announce “contradiction”. Otherwise, for each of the C_0 -edges $v_{k-1} v_k, v_k v_{k+1}$, if this edge is uncolored, we color it black. (Note that, if the Yellow-Force operation does not announce “contradiction”, at the end of the operation all four C_0 -edges $v_{i-1} v_i, v_i v_{i+1}, v_{k-1} v_k, v_k v_{k+1}$ that are incident to the chord $v_i v_i^*$ are colored black.)

The main idea of Algorithm 1 is as follows. In every non-final iteration we have that $D = Red \cup Blue$ is an alternating *path*, while in the final iteration D is an alternating *cycle*. Suppose that, during a non-final iteration, we extend D by adding a new blue chord $v_x v_y$ (where v_y is the last vertex of D). The cases where not both edges $v_y v_{y+1}, v_y v_{y-1}$ are eligible are covered by the following observation.

Observation 4 *The only case, in which at least one of the C_0 -edges $v_y v_{y+1}, v_y v_{y-1}$ is red, is when $v_y = v_1$. In this case, exactly one of the C_0 -edges $v_y v_{y+1}, v_y v_{y-1}$ is red and the other one is black (see the initialization lines 1-5 of the algorithm), and thus D becomes an alternating cycle and the next iteration is the final one. In all other cases, where none of the C_0 -edges $v_y v_{y+1}, v_y v_{y-1}$ is red, at least one of them is eligible; otherwise both $v_y v_{y+1}, v_y v_{y-1}$ are black, and thus their commonly incident chord $v_x v_y$ has been previously colored yellow (by the operation Black-Force), a contradiction. If only one of these two edges is eligible, the algorithm is forced to color this edge red (with the operation Blue-Force).*

If both edges $v_y v_{y+1}, v_y v_{y-1}$ are eligible, the algorithm *branches* (in most cases) to two new instances I_1 and I_2 , where I_1 (resp. I_2) has the eligible edge $v_y v_{y+1}$ (resp. $v_y v_{y-1}$) colored red. After the algorithm has branched to these two new instances I_1 and I_2 , it exhaustively applies the four forcing operations Blue-Force, Red-Force, Black-Force, and Yellow-Force, until none of them is applicable any more. The correctness of these forcing operations becomes straightforward by recalling our interpretation of the four colors, i.e. that the C_0 -edges belonging (resp. not belonging) to C_1 are colored *black* (resp. *red*), while the chords belonging (resp. not belonging) to C_1 are colored *blue* (resp. *yellow*).

In some cases, the exhaustive application of the forcing rules in the two new instances I_1, I_2 may only force very few edges, which results in a large running time of the algorithm before we reach a state where D becomes an alternating red-blue cycle. To circumvent this problem, we refrain from just always applying the operation Blue-Branch. Instead, in some cases we are able to *defer* the choice of the forced color of specific edges until the very end. More specifically, in some cases we are able to determine specific sets of 4 edges (each containing three C_0 -edges and one chord) which build a C_4 in G (i.e. a cycle of length 4) such that the all colored edges in the two different instances I_1, I_2 are identical, apart from the colors of these 4 edges. Therefore all forcing operations in the subsequent iterations of the algorithm are *identical* in both these instances I_1, I_2 , regardless of the specific colors of these 4 edges. Furthermore, as it turns out, every such a set of 4 of edges can receive forced colors in *exactly two* alternative ways. We call every such a set an *ambivalent quadruple* of edges. In these few cases, where an ambivalent quadruple occurs, we do not apply the operation Blue-Branch; instead we continue our forcing and branching operations in the subsequent iterations of the algorithm by only starting from one of these instances (instead of starting from both instances). Then, at the final step of the algorithm, i.e. when D becomes an alternating red-blue cycle, we are able to decide which of the two alternative edge colorings is correct for each ambivalent quadruple of edges (see the call to Procedure 2 in line 9 of the algorithm).

The above crucial trick of not always applying the operation Blue-Branch allows us to avoid generating *all* possible red-blue alternating cycles, thus obtaining an exponential speed-up of the algorithm and beating the state-of-the-art running time of $O^*(2^{0.3n})$ which is implied by the TSP-algorithm of [20]. For example, in one of the cases where an ambivalent quadruple occurs, if we would branch to two new instances we would only force 5 new edges. Thus, since G has $\frac{3}{2}n$ edges (as a cubic graph), forcing 5 edges at a time would imply the generation of at most $O^*\left(2^{\frac{3}{2} \cdot \frac{1}{5}n}\right) = O^*(2^{0.3n})$ instances in the worst case, each of them corresponding to a different red-blue alternating cycle. However, by deferring the exact coloring of all ambivalent quadruples until the end of the algorithm we bypass this problem: instead of generating *all possible* red-blue alternating cycles, we create a succinct representation of them by only generating $O(2^{(0.3-\varepsilon)n})$ alternating cycles (for some constant $\varepsilon > 0$), and then we determine from them the desired alternating cycle, i.e. the one which gives us a second Hamiltonian cycle as its symmetric difference with the given first Hamiltonian cycle C_0 (see the call to Procedure 2 in line 9 of the algorithm). Now we define operation Ambivalent-Flip, which appropriately changes at the end of the algorithm the already chosen colors of an ambivalent quadruple (see Procedure 2). Recall here that every ambivalent quadruple q contains exactly three C_0 -edges and one chord.

- **Ambivalent-Flip:** Let q be an ambivalent quadruple of (already colored) edges. For every C_0 -edge of q , if it has been colored red (resp. black), change its color to black (resp. red). Also, if the (unique) chord of q has been colored yellow (resp. blue), change its color to blue (resp. yellow).

Before we proceed with the proof of our main technical lemmas in this section (see Lemmas 2 and 3), we first need to define the notions of a *forcing path* and a *forcing cycle*. Intuitively, a forcing path consists of a sequence of edges of G such that, during the execution of Algorithm 1, once the first edge is forced to receive a specific color, every other edge of the path is also forced to receive some other specific color.

Definition 1 (forcing path and cycle) *Let G be a Smith graph. At an arbitrary iteration of Algorithm 1, a path $P = (v_{i_1}, v_{i_2}, \dots, v_{i_k})$ of G is a forcing path starting at vertex v_{i_1} if:*

- *each of its edges $v_{i_1}v_{i_2}, \dots, v_{i_{k-1}}v_{i_k}$ is yet uncolored and*
- *each of its first $k-1$ vertices $v_{i_1}, \dots, v_{i_{k-1}}$ is incident to exactly one already colored edge, while its last vertex v_{i_k} is incident to three yet uncolored edges.*

Similarly, a cycle $C = (v_{i_1}, v_{i_2}, \dots, v_{i_k}, v_{i_1})$ of G is a forcing cycle if:

- *each of its edges $v_{i_1}v_{i_2}, \dots, v_{i_{k-1}}v_{i_k}, v_{i_k}v_{i_1}$ is yet uncolored and*
- *each of its k vertices v_{i_1}, \dots, v_{i_k} is incident to exactly one already colored edge.*

Recall that, at every non-final iteration of the algorithm, there is exactly one *blue* edge v_xv_y such that its one endpoint v_x is incident to two other previously colored edges (one red and one black) and its other endpoint v_y is incident either to two uncolored edges or to one uncolored edge and one black edge. On the other hand, there might be several *black* edges v_iv_j such that v_i is incident to two other previously colored edges and v_j is incident to two uncolored edges. Furthermore, at the end of every iteration of the algorithm, every yellow and every red edge of G (apart from the first red edge of the alternating path D) is adjacent to four other colored edges. Thus the next observation follows easily.

Observation 5 *Let v_xv_y be the blue chord, where v_y is the last vertex of the red-blue alternating path D at some iteration of Algorithm 1. Furthermore let $P = (v_y, \dots, v_\ell)$ be a forcing path of G , starting at vertex v_y . Then every internal vertex of P is incident to one black C_0 -edge, as well as to one uncolored C_0 -edge and to one uncolored chord.*

In the next lemma we prove the correctness of Algorithm 1. Next we prove our crucial technical Lemma 3 which specifies how the current instance is transformed in one iteration of our algorithm. The input instance I of the algorithm consists of a Smith graph $G = (V, E)$, a Hamiltonian cycle C_0 of G , the set

Algorithm 1 ALTERNATING CYCLE DETECTION

Input: Instance $I = \{G, C_0, Q, Red, Blue, Black, Yellow\}$, where $G = (V, E)$ is a Smith graph, $C_0 = (v_1, v_2, \dots, v_n)$ is an initial Hamiltonian cycle of G , Q is a set of mutually disjoint quadruples of edges, and $Red, Blue, Black, Yellow$ are four disjoint edge-subsets of E such that $Red \cup Black \subseteq E(C_0)$ and $Blue \cup Yellow \subseteq E \setminus E(C_0)$.

Output: A second Hamiltonian cycle C_1 of G such that $D = C_0 \Delta C_1$ is connected.

```
1: if  $Q = Red = Blue = Black = Yellow = \emptyset$  then {initialization}
2:   Call the algorithm with the following parameters: {Look for an alternating cycle where  $v_1v_2$  is red}
3:    $Q \leftarrow \emptyset$ ;  $Red \leftarrow \{v_1v_2\}$ ;  $Blue \leftarrow \{v_2v_2^*\}$ ;  $Black \leftarrow \{v_2v_3, v_1v_n\}$ ;  $Yellow \leftarrow \emptyset$ 
4:   Call the algorithm with the following parameters: {Look for an alternating cycle where  $v_1v_n$  is red}
5:    $Q \leftarrow \emptyset$ ;  $Red \leftarrow \{v_1v_n\}$ ;  $Blue \leftarrow \{v_nv_n^*\}$ ;  $Black \leftarrow \{v_{n-1}v_n, v_1v_2\}$ ;  $Yellow \leftarrow \emptyset$ 
6: else {Main Iteration}
7:    $D \leftarrow Blue \cup Red$ 
8:   if  $D$  is a cycle then {final iteration of the algorithm}
9:     Call Procedure 2
10:  else { $D$  is a red-blue alternating path of even length}
11:    Let  $v_y$  be the last vertex of  $D$  (which ends with the blue chord  $v_xv_y$ )
12:    if  $v_yv_{y+1}, v_yv_{y-1} \notin Black$  then {both  $v_yv_{y+1}$  and  $v_yv_{y-1}$  are eligible  $C_0$ -edges}
13:      Let  $P^+$  and  $P^-$  be the forcing paths starting at  $v_y$  with the edge  $v_yv_{y+1}$  and  $v_yv_{y-1}$ , respectively
14:      if  $P^+ \cup P^-$  builds a  $C_4$  (i.e. a path with 4 edges) then
15:        Call Procedure 3
16:      else if  $P^+ \cup P^-$  builds a  $P_4$  (i.e. a path with 4 vertices) whose two endpoints are adjacent then
17:        Call Procedure 4
18:      else { $P^+ \cup P^-$  is neither a  $C_4$  nor a  $P_4$  whose two endpoints are adjacent}
19:        Apply the operation Blue-Branch, generating two new instances  $I_1, I_2$ 
20:        Apply exhaustively the operations Blue-Force, Red-Force, Black-Force, Yellow-Force to
        instances  $I_1, I_2$ , until no operation can be further applied
21:        for  $i \in \{1, 2\}$  do
22:          if no “contradiction” has been announced for  $I_i$  then call the algorithm on instance  $I_i$ 
23:        else {only one of  $v_yv_{y+1}, v_yv_{y-1}$  is an eligible  $C_0$ -edge}
24:          Apply exhaustively the operations Blue-Force, Red-Force, Black-Force, Yellow-Force to
          instance  $I$  until no operation can be further applied
25:          if no “contradiction” has been announced then
26:            Call the algorithm on the updated instance  $I$ 
```

Procedure 2 Check the alternating red-blue cycle D

```
1: for every ambivalent edge-quadruple  $q \in Q$  do
2:   if applying Ambivalent-Flip to the colors of  $q$  strictly reduces the number of connected components
   of  $C_0 \Delta D$  then
3:     Apply the operation Ambivalent-Flip to  $q$  and update  $D$  accordingly
4: if  $C_0 \Delta D$  is a Hamiltonian cycle of  $G$  then
5:   return Hamiltonian cycle  $C_1$  and alternating cycle  $D$ 
```

Q of all ambivalent quadruples, and four disjoint sets of forced (i.e. colored) edges $Red, Blue, Black, Yellow$. Initially the four sets of uncolored edges as well as the set Q are all empty. Given such an instance $I = (G, C_0, Q, Red, Blue, Black, Yellow)$, we denote by $U(I) = E \setminus \{Red \cup Blue \cup Black \cup Yellow\}$ be the set of all *unforced* (i.e. uncolored) edges in this instance. Furthermore we denote by $W(I)$ the set of vertices which are not incident to any edge of $Red \cup Black$ in I ; we refer to the vertices of $W(I)$ as *biased* vertices,

Procedure 3 Update of instance I when $P^+ \cup P^-$ builds a C_4

- 1: $Q \leftarrow Q \cup \{E(P^+ \cup E(P^-))\}$ {new ambivalent quadruple of edges}
 - 2: $Red \leftarrow Red \cup \{v_y v_{y+1}\}$; $Black \leftarrow Black \cup \{v_y v_{y-1}\}$
 - 3: Apply exhaustively the operations **Blue-Force**, **Red-Force**, **Black-Force**, **Yellow-Force** to instance I until no operation can be further applied
 - 4: **if** no “contradiction” has been announced **then**
 - 5: Call Algorithm 1 on the updated instance I
-

Procedure 4 Create instances I_1, I_2 when $P^+ \cup P^-$ builds a P_4 whose two endpoints are adjacent

- 1: $I_1 \leftarrow I$; $I_2 \leftarrow I$
 - 2: **if** P^+ contains only the edge $v_y v_{y+1}$ and P^- contains the two edges $v_y v_{y-1}, v_{y-1} v_{y+2}$ **then**
 - 3: Update I_1 such that:
 $Red \leftarrow Red \cup \{v_y v_{y+1}\}$; $Q \leftarrow Q \cup \{\{v_y v_{y+1}, v_y v_{y-1}, v_{y-1} v_{y+2}, v_{y+2} v_{y+1}\}\}$
 - 4: Update I_2 such that:
 $Red \leftarrow Red \cup \{v_y v_{y-1}\}$; $Black \leftarrow Black \cup \{v_{y+1} v_{y+2}\}$
 - 5: **else** $\{P^+$ contains the two edges $v_y v_{y+1}, v_{y+1} v_{y-2}$ and P^- contains only the edge $v_y v_{y-1}\}$
 - 6: Update I_2 such that:
 $Red \leftarrow Red \cup \{v_y v_{y-1}\}$; $Q \leftarrow Q \cup \{\{v_y v_{y-1}, v_y v_{y+1}, v_{y+1} v_{y-2}, v_{y-2} v_{y-1}\}\}$
 - 7: Update I_1 such that:
 $Red \leftarrow Red \cup \{v_y v_{y+1}\}$; $Black \leftarrow Black \cup \{v_{y-1} v_{y-2}\}$
 - 8: Apply exhaustively the operations **Blue-Force**, **Red-Force**, **Black-Force**, **Yellow-Force** to instances I_1, I_2 , until no operation can be further applied
 - 9: **for** $i \in \{1, 2\}$ **do**
 - 10: **if** no “contradiction” has been announced for I_i **then** call Algorithm 1 on instance I_i
-

while all other vertices in $V - W(I)$ are referred to as *unbiased* vertices. Finally, we refer to the set of ambivalent quadruples of instance I as $Q(I)$.

Lemma 2 *Let $G = (V, E)$ be a Smith graph and C_0 be a Hamiltonian cycle of G . Then, Algorithm 1 correctly computes a second Hamiltonian cycle C_1 of G on the input $I = (G, C_0, \emptyset, \emptyset, \emptyset, \emptyset)$.*

Proof. First recall the interpretation of the four colors: the C_0 -edges that belong (resp. do not belong) to the desired Hamiltonian cycle C_1 are colored *black* (resp. *red*), while the chords that belong (resp. do not belong) to C_1 are colored *blue* (resp. *yellow*). In the initialization phase, Algorithm 1 arbitrarily picks vertex v_1 and it generates two instances I_1, I_2 (see lines 1-5), where in I_1 (resp. I_2) the C_0 -edge $v_1 v_2$ is red, the chord $v_2 v_2^*$ is blue, and the C_0 -edges $v_2 v_3, v_1 v_n$ are black (resp. $v_1 v_n$ is red, $v_n v_n^*$ is blue, and $v_{n-1} v_n, v_1 v_2$ are black). Then the algorithm calls itself both on input I_1 and on input I_2 . This action of the algorithm is correct by Corollary 1, since the algorithm searches for the desired second Hamiltonian cycle C_1 such that either $v_1 v_2$ or $v_1 v_n$ is red (i.e. does not belong to C_1).

In each of these initial cases, the algorithm starts with one red C_0 -edge and one blue chord (i.e. with an alternating red-blue path D of length two), and it iteratively extends D by adding one red C_0 -edge and one blue chord to it, until either a “contradiction” is announced (by one of the forcing rules), or D becomes an alternating red-blue cycle by hitting the first red edge of D at vertex v_1 with the blue chord $v_1 v_1^*$. In every non-final iteration of the algorithm, i.e. when D is still an alternating path, the algorithm proceeds as follows. Suppose that D ends at vertex v_y with the blue chord $v_x v_y$. Due to Observation 4, at most one of the two incident C_0 -edges $v_y v_{y+1}, v_y v_{y-1}$ is red. If exactly one of them is red, then the other incident C_0 -edge is black, while D becomes an alternating cycle and the next iteration is the last one. Furthermore, it also follows by Observation 4 that, whenever none of the C_0 -edges $v_y v_{y+1}, v_y v_{y-1}$ is red, there is at least one eligible (i.e. uncolored) edge and at most one black edge among $v_y v_{y+1}, v_y v_{y-1}$.

If exactly one of these two edges is eligible and the other one is black (see lines 23-26), then the algorithm is forced to color this eligible edge red. Thus, in this case the algorithm correctly updates the current instance I by exhaustively applying the forcing operations Blue-Force, Red-Force, Black-Force, and Yellow-Force until none of them can be applied any more. Now suppose that both edges $v_y v_{y+1}, v_y v_{y-1}$ are eligible, and let P^+ (resp. P^-) be the forcing path starting at vertex v_y with the edge $v_y v_{y+1}$ (resp. $v_y v_{y-1}$). Furthermore suppose that none of the conditions of lines 14 and 16 are satisfied. Then the algorithm first branches into two new instances I_1, I_2 (by applying Blue-Force at vertex v_y) and it then exhaustively applies the four forcing rules (see lines 18-22). The correctness of these forcing operations becomes straightforward by recalling our interpretation of the four colors. At the final iteration of the algorithm (see the call to Procedure 2 in line 9 of the algorithm), if the set Q of all ambivalent quadruples is empty, then the algorithm just checks whether the symmetric difference between C_0 and the produced alternating red-blue cycle D is just one cycle. The correctness of this check follows by Corollary 1.

It remains to prove the correctness of the algorithm also in the case where one of the conditions of lines 14 and 16 is satisfied. First we analyze each of these two cases separately, as follows.

Case 1: line 14 is applied. The union $P^+ \cup P^-$ of the two forcing paths builds a C_4 (i.e. a cycle with 4 edges). Assume that each of the paths P^+, P^- has two edges, i.e. P^+ contains the C_0 -edge $v_y v_{y+1}$ and the chord $v_{y+1} v_{y+1}^*$, while P^- contains the C_0 -edge $v_y v_{y-1}$ and the chord $v_{y-1} v_{y-1}^*$. Then, since $P^+ \cup P^-$ is a C_4 , it follows that $v_{y+1}^* = v_{y-1}^*$, and thus v_{y+1}^* is incident to two different chords, which is a contradiction as G is a cubic graph. Therefore, one of the forcing paths P^+, P^- has length 1 and the other one has length 3. As these two cases are symmetric, assume without loss of generality that P^+ has length 1 (i.e. it only contains the C_0 -edge $v_y v_{y+1}$) and P^- has length 3. Since vertex v_{y+1} is the common endpoint of P^+ and P^- , it follows that P^- contains the C_0 -edge $v_y v_{y-1}$, the chord $v_{y-1} v_{y+2}$, and the C_0 -edge $v_{y+2} v_{y+1}$ (in this order). That is, the cycle $P^+ \cup P^-$ contains exactly three C_0 -edges and one chord. Furthermore note that the third edge incident to v_{y+1} (apart from v_y and v_{y+2}) is the chord $v_{y+1} v_{y+1}^*$.

Applying the operation Blue-Branch at this iteration would result in the creation of two new instances I_1, I_2 , in which the edges are colored as follows. In I_1 , the C_0 -edge $v_y v_{y+1}$ is colored red, the C_0 -edges $v_y v_{y-1}, v_{y+2} v_{y+1}$ are colored black, and the chord $v_{y-1} v_{y+2}$ is colored yellow. In I_2 , the C_0 -edge $v_y v_{y+1}$ is colored black, the C_0 -edges $v_y v_{y-1}, v_{y+2} v_{y+1}$ are colored red, and the chord $v_{y-1} v_{y+2}$ is colored blue. Note that, in *both* instances I_1, I_2 , these edge colorings force the chord $v_{y+1} v_{y+1}^*$ to be colored blue, and these are all edge colorings that can be forced so far.

That is, the only difference between the instances I_1, I_2 is the way the four edges of $P^+ \cup P^-$ are colored. Therefore, since each of the vertices of $P^+ \cup P^-$ is “saturated” in both I_1, I_2 (i.e. it has all its three incident edges colored), all forcing operations in the subsequent iterations of Algorithm 1 are *identical* in both I_1, I_2 . Using this fact, the algorithm marks the edges of $P^+ \cup P^-$ as an *ambivalent* quadruple of edges (see line 1 of Procedure 3). Furthermore, it colors these 4 edges according to I_1 only, i.e. without branching to both I_1, I_2 , and it calls itself on the updated instance I (see lines 2-5 of Procedure 3). Since all subsequent forcing operations would be identical in both instances I_1, I_2 , the algorithm continues either until a contradiction is concluded at a later iteration (in this case a contradiction would be concluded by both I_1, I_2) or until D becomes an alternating cycle.

Case 2: line 16 is applied. The union $P^+ \cup P^-$ of the two forcing paths builds a P_4 (i.e. a path with 4 vertices) whose two endpoints are adjacent. In this case, clearly one of the paths P^+, P^- contains one edge and the other one contains two edges. As these two cases are symmetric (see lines 2-4 and lines 5-7 of Procedure 4, respectively), it suffices to only analyze here the case that P^+ contains only the C_0 -edge $v_y v_{y+1}$ and P^- contains the C_0 -edge $v_y v_{y-1}$ and the chord $v_{y-1} v_{y+2}$. Note that, by the assumption of Case 2, the endpoints of the paths P^+, P^- are connected via the C_0 -edge $v_{y+1} v_{y+2}$. Furthermore, note that the third edge incident to v_{y+1} (apart from v_y and v_{y+2}) is the chord $v_{y+1} v_{y+1}^*$; similarly, the third edge incident to v_{y+2} (apart from v_{y-1} and v_{y+1}) is the C_0 -edge $v_{y+2} v_{y+3}$.

Applying the operation Blue-Branch at this iteration would result in the creation of two new instances I_1, I_2 , in which the edges are colored as follows. In I_1 , the C_0 -edge $v_y v_{y+1}$ is colored red, the C_0 -edges $v_y v_{y-1}, v_{y+1} v_{y+2}, v_{y+2} v_{y+3}$ are colored black, the chord $v_{y-1} v_{y+2}$ is colored yellow, and the chord $v_{y+1} v_{y+1}^*$ is colored blue. On the other hand, in I_2 the C_0 -edge $v_y v_{y+1}$ is colored black, the C_0 -edge $v_y v_{y-1}$ is colored

red, and the chord $v_{y-1}v_{y+2}$ is colored blue. Consider now applying again the operation Blue-Branch in the instance I_2 at vertex v_{y+2} (once the chord $v_{y-1}v_{y+2}$ has been colored blue in I_2). This would replace instance I_2 by two new instances I_2^1 and I_2^2 , in which the edges are colored as follows. In I_2^1 , the C_0 -edges $v_yv_{y+1}, v_{y+2}v_{y+3}$ are colored black, the C_0 -edges $v_yv_{y-1}, v_{y+1}v_{y+2}$ are colored red, and the chords $v_{y-1}v_{y+2}, v_{y+1}v_{y+1}^*$ are colored blue. Furthermore, in I_2^2 the C_0 -edges $v_yv_{y+1}, v_{y+1}v_{y+2}$ are colored black, the C_0 -edges $v_yv_{y-1}, v_{y+2}v_{y+3}$ are colored red, the chord $v_{y-1}v_{y+2}$ is colored blue, and the chord $v_{y+1}v_{y+1}^*$ is colored yellow.

Now note that both instances I_1 and I_2^1 are identical, apart from the colors of the four edges $v_yv_{y+1}, v_yv_{y-1}, v_{y-1}v_{y+2}, v_{y+1}v_{y+2}$. Therefore, since each of these four vertices $v_y, v_{y+1}, v_{y-1}, v_{y+1}$ is “saturated” in both I_1 and I_2^1 (i.e. it has all its three incident edges colored), all forcing operations in the subsequent iterations of Algorithm 1 are *identical* in both I_1, I_2^1 . Thus, instead of branching into the three instances I_1, I_2^1, I_2^2 , the algorithm only branches into the two instances I_1 and I_2^2 in Procedure 4, while it also marks the above four edges as an *ambivalent* quadruple of edges within I_1 (see line 3 of Procedure 4). Then, within the recursive call on instance I_1 , the algorithm continues either until a contradiction is concluded at a later iteration (in this case a contradiction would be concluded by both I_1, I_2^1) or until D becomes an alternating cycle.

Correctness for both Cases 1 and 2. Suppose that, at some iteration of Algorithm 1, D becomes an alternating cycle, and let Q be the set of ambivalent edge-quadruples that the algorithm has marked so far. Then the algorithm calls Procedure 2 (see line 8 of the algorithm). Recall by the above analysis of Cases 1 and 2 that, for every ambivalent edge-quadruple $q \in Q$, the algorithm had to choose between two alternative edge colorings of the four edges of q . Furthermore, until the execution of lines 1-3 of Procedure 2, these choices were made arbitrarily, as the choice between these two alternative colorings of q had no effect on the subsequent iterations of the algorithm. Now, performing the operation Ambivalent-Flip at an ambivalent edge-quadruple $q \in Q$, is equivalent to choosing the second alternative coloring of q .

Assume that, at the beginning of Procedure 2, the symmetric difference $C_0 \Delta D$ has k cycles. In lines 1-3 of the procedure, the algorithm attempts to sequentially perform the operation Ambivalent-Flip on all ambivalent quadruples $q \in Q$. By flipping the coloring of such a quadruple q , the number k of connected components of $C_0 \Delta D$ can either reduce to $k - 1$, or increase to $k + 1$, or stay unchanged at k . Note that, if it decreases to $k - 1$, then flipping the colors of q connects two different connected components (i.e. cycles) of $C_0 \Delta D$ into one. Similarly, if it increases to $k + 1$, then flipping the colors of q disconnects one cycle of $C_0 \Delta D$ into two different ones. Finally, if it stays unchanged at k , then flipping the colors of q simply replaces one cycle of $C_0 \Delta D$ with another cycle that visits the same vertices in a different order.

Now note that, performing the operation Ambivalent-Flip at one edge-quadruple q , does not change the color of an edge in any other quadruple $q' \in Q \setminus \{q\}$. Thus, at the end of the execution of lines 2-3 of Procedure 2, the algorithm can decide whether there exists a sequence of choices for the edge-colorings of the ambivalent quadruples in Q which derive the desired second Hamiltonian cycle C_1 . More specifically, if the symmetric difference $C_0 \Delta D$ is connected (see line 4 of Procedure 2, where now D is the updated red-blue cycle after exhaustively executing lines 2-3 of the procedure) then $C_0 \Delta D$ is the desired second Hamiltonian cycle. Otherwise, if $C_0 \Delta D$ still contains more than one cycle, it follows that no sequence of choices for the alternative edge-colorings of the quadruples in Q could derive a second Hamiltonian cycle. This completes the proof of the lemma. ■

Lemma 3 *Let $I = (G, C_0, Q, Red, Blue, Black, Yellow)$ be the instance at some iteration of Algorithm 1, where $G = (V, E)$ is a Smith graph, and let $D = Red \cup Blue$ be the current alternating red-blue path of even length. Then, within a constant number of iterations, either a “contradiction” is announced or the algorithm transforms the instance I in lines 12-26 either to a single instance I' , where $|U(I')| \leq |U(I)| - 2$, or to two instances I_1 and I_2 , where one of the following is satisfied:*

1. $|W(I_1)|, |W(I_2)| \leq |W(I)| - 2$ and $|U(I_1)|, |U(I_2)| \leq |U(I)| - 7$,
2. $|W(I_1)|, |W(I_2)| \leq |W(I)| - 2$ and $|U(I_1)|, |U(I_2)| \leq |U(I)| - 9$,
3. $|W(I_1)|, |W(I_2)| \leq |W(I)| - 4$ and $|U(I_1)|, |U(I_2)| \leq |U(I)| - 4$,

4. $|W(I_1)| \leq |W(I)| - 4$, $|U(I_1)| \leq |U(I)| - 4$, and $|W(I_2)| \leq |W(I)| - 4$, $|U(I_2)| \leq |U(I)| - 6$,
5. $|W(I_1)| \leq |W(I)| - 2$, $|U(I_1)| \leq |U(I)| - 9$, and $|W(I_2)| \leq |W(I)| - 4$, $|U(I_2)| \leq |U(I)| - 6$,
6. $|W(I_1)| \leq |W(I)| - 2$, $|U(I_1)| \leq |U(I)| - 5$, and $|W(I_2)| \leq |W(I)| - 4$, $|U(I_2)| \leq |U(I)| - 8$,
7. $|W(I_1)| \leq |W(I)| - 2$, $|U(I_1)| \leq |U(I)| - 3$, and $|W(I_2)| \leq |W(I)| - 6$, $|U(I_2)| \leq |U(I)| - 7$,
8. $|W(I_1)| \leq |W(I)| - 2$, $|U(I_1)| \leq |U(I)| - 3$, and
 $|W(I_2)| \leq |W(I)| - 4$, $|U(I_2)| \leq |U(I)| - 10$,
9. $|W(I_1)| \leq |W(I)| - 2$, $|U(I_1)| \leq |U(I)| - 3$, and $|W(I_2)| \leq |W(I)| - 5$, $|U(I_2)| \leq |U(I)| - 9$.

Proof. Let v_y be the last vertex of the alternating red-blue path D , and let $v_x v_y$ be its last blue chord. Throughout the proof we assume that no “contradiction” is announced at the current iteration. Recall that the set $W(I)$ contains all biased vertices of instance I , i.e. all vertices which are incident to an edge of $Red \cup Black$. Furthermore recall that the set $U(I)$ contains all unforced edges of instance I , i.e. all edges that are not contained in the set $Red \cup Blue \cup Black \cup Yellow$. Since D is an alternating red-blue path (and not a red-blue cycle) by the assumption of the lemma, note that at least one of the two C_0 -edges $v_y v_{y+1}, v_y v_{y-1}$ is eligible and at most one of them is already colored black, see Observation 4. Assume that one of these two edges is colored and the other one uncolored; note that the colored one can only be black. In this case the other edge is forced to be colored red by the operation Blue-Force. Furthermore, this triggers the operation Red-Force, which forces its uncolored incident chord blue. Thus, in this case the algorithm reduces the problem to a new single instance I' , which has at least two more edges colored, i.e. $|U(I')| \leq |U(I)| - 2$. This corresponds to the case (i) of the lemma.

For the remainder of the proof assume that both C_0 -edges $v_y v_{y+1}, v_y v_{y-1}$ are eligible. Then the operation Blue-Branch takes place and creates two instances I_1, I_2 (except the cases where line 15 or line 17 of Algorithm 1 is executed, which are dealt with separately in the proof below), where in I_1 the edge $v_y v_{y+1}$ is red and the edge $v_y v_{y-1}$ is black, and in I_2 the edge $v_y v_{y-1}$ is red and the edge $v_y v_{y+1}$ is black. Note that, in both cases, v_y becomes a new biased vertex at this iteration as it becomes incident to both a red edge and a black edge. It is not hard to see that the two C_0 -edges $v_y v_{y+1}, v_y v_{y-1}$ cannot participate together in a forcing cycle. Indeed, in such a forcing cycle C , one of the edge sequences $v_y v_{y+1}, v_{y+1} v_{y+1}^*, \dots$ and $v_y v_{y-1}, v_{y-1} v_{y-1}^*, \dots$ alternatively receives the colors red and blue, while the other one alternatively receives the colors black and yellow. Therefore, there exists exactly one forcing path P^- starting at vertex v_y with the edge $v_y v_{y-1}$, and exactly one forcing path P^+ starting at vertex v_y with the edge $v_y v_{y+1}$. The next observation follows easily from the fact that for every previously colored edge $v_i v_j$, at least one of its endpoints v_i, v_j is incident to *three* previously colored edges.

Observation 6 *The two forcing paths P^-, P^+ do not share any common internal vertex.*

Case 1. Both P^- and P^+ end with a C_0 -edge. That is, each of these forcing paths has an even number of *internal* vertices. Since the analysis for both new instances I_1, I_2 is symmetric, in most subcases of Case 1 (with the exception of Case 1(iv)) we only analyze instance I_1 , i.e. the case where $v_y v_{y+1}$ becomes red and $v_y v_{y-1}$ becomes black by the operation Blue-Branch. Let v_ℓ be the last vertex of P^+ , and assume without loss of generality that the last edge of P^+ is $v_{\ell-1} v_\ell$ (the other case, where the last edge of P^+ is $v_\ell v_{\ell+1}$ is exactly symmetric). Similarly, let v_q be the last vertex of P^- , and assume without loss of generality that the last edge of P^- is $v_{q-1} v_q$. That is, $v_{\ell-1}$ and v_{q-1} are the last *internal* vertices of P^+ and of P^- , respectively. Note that v_q becomes a new biased vertex after the forcing operations along P^- , as it becomes incident to the new black edge $v_{q-1} v_q$. Similarly v_ℓ becomes a new biased vertex after the forcing operations along P^+ , as it becomes incident to the new red edge $v_{\ell-1} v_\ell$ and to the new black edge $v_\ell v_{\ell+1}$. That is, v_y, v_q, v_ℓ become new biased vertices.

Case 1(i). $v_\ell = v_q$. Then, since P^+ and P^- share v_y as a common vertex, $P^+ \cup P^-$ cannot have just two edges, i.e. $P^+ \cup P^-$ has at least 4 edges. First assume that $P^+ \cup P^-$ has 4 edges, that is, $P^+ \cup P^-$ is a C_4 . Then the algorithm executes line 15 and calls Procedure 3. In this case, it only updates the current

instance I by forcing the colors of at least 5 edges, namely the 4 edges of $P^+ \cup P^-$ as well as the chord $v_\ell v_\ell^*$. Moreover, the updated instance has at least 2 new biased vertices v_y, v_ℓ . Furthermore the 4 edges of $P^+ \cup P^-$ are added as an ambivalent quadruple in the set Q (see line 1 of Procedure 3).

Now assume that $P^+ \cup P^-$ has exactly 6 edges. Then the algorithm executes lines 18-22 and creates two new instances I_1, I_2 , each of them having at least 2 new biased vertices v_y, v_ℓ and at least 7 new forced edges, namely at least 6 forced edges in $P^+ \cup P^-$ as well as the blue chord $v_\ell v_\ell^*$. Finally, assume that $P^+ \cup P^-$ has at least 8 edges. Then, similarly to the previous paragraph, we have at least 2 new biased vertices and 9 new forced edges.

Summarizing, in Case 1(i) we either have two instances I_1, I_2 , each having at least 2 new biased vertices and 7 forced edges (when $P^+ \cup P^-$ has exactly 6 edges), or at least 2 new biased vertices and 9 forced edges (when $P^+ \cup P^-$ has at least 8 edges), or we just have one updated instance I which has 2 new biased vertices and 5 forced edges.

Case 1(ii). $v_\ell = v_{q+1}$ (or equivalently, $v_q = v_{\ell+1}$). In this case, when edge $v_{\ell-1}v_\ell$ (i.e. the last edge of P^+) is colored red, the operation Red-Force is triggered which colors the C_0 -edge $v_\ell v_q$ black and the chord $v_\ell v_\ell^*$ blue. On the other hand, note that edge $v_{q-1}v_q$ (i.e. the last edge of P^-) is colored black. Thus, since both C_0 -edges $v_{q-1}v_q$ and $v_\ell v_q$ are black, the operation Black-Force is triggered at vertex v_q which colors the chord $v_q v_q^*$ yellow. Then, the operation Yellow-Force (which is triggered once $v_q v_q^*$ is colored yellow) colors at least one edge incident to v_q^* black. Thus, so far we have at the current iteration at least 6 new forced edges, namely at least one forced edge in each of P^+ and P^- , as well as the edges $v_\ell v_\ell^*, v_\ell v_q, v_q v_q^*$ and at least one C_0 -edge incident to v_q^* .

If v_q^* is incident to two uncolored C_0 -edges then v_q^* becomes a new biased vertex, i.e. we have four new biased vertices v_y, v_q, v_ℓ, v_q^* . If v_q^* is incident to two previously colored (i.e. black) C_0 -edges then the edges $v_q v_q^*$ and $v_\ell v_q$ have been colored at a previous iteration yellow and black, respectively, which is a contradiction as they received these colors at the current iteration. Finally, if v_q^* is incident to one previously colored (i.e. black) C_0 -edge and to one uncolored C_0 -edge, then a new forcing path starts at v_q with the chord $v_q v_q^*$. At the end of this forcing path, there must be at least one C_0 -edge incident to an unbiased vertex v_z , which becomes black after all Black-Force and Yellow-Force operations. Thus v_z becomes a new biased vertex, that is, we have four new biased vertices v_y, v_q, v_ℓ, v_z .

Summarizing, in Case 1(ii) we have two instances I_1, I_2 , each having at least 4 new biased vertices and 6 forced edges.

Case 1(iii). $v_{\ell+1} = v_{q+1}$. Similarly to Case 1(ii), when edge $v_{\ell-1}v_\ell$ is colored red, Red-Force is triggered which colors the C_0 -edge $v_\ell v_{\ell+1}$ black and the chord $v_\ell v_\ell^*$ blue. However, since v_q is the last vertex of P^- , it follows that the edge $v_q v_{q+1} = v_q v_{\ell+1}$ is uncolored, and thus $v_{\ell+1}$ becomes a new biased vertex. That is, we have 4 new biased vertices $v_y, v_q, v_\ell, v_{\ell+1}$. Assume that $P^+ \cup P^-$ has two edges. Then each of P^+ and P^- has only one edge, namely the C_0 -edges $v_y v_{y+1}$ and $v_y v_{y-1}$, respectively. Then, since the edges $v_\ell v_{\ell+1}$ and $v_q v_{\ell+1}$ are also C_0 -edges, it follows that there is a cycle of 4 C_0 -edges. This is a contradiction, as C_0 is a Hamiltonian cycle of a graph G with more than 4 vertices. Thus $P^+ \cup P^-$ has at least 4 edges, and thus we have at least 6 new forced edges, namely at least 4 forced edges in $P^+ \cup P^-$, as well as the edges $v_\ell v_\ell^*, v_\ell v_{\ell+1}$.

Summarizing, in Case 1(iii) we have two instances I_1, I_2 , each having at least 4 new biased vertices and 6 forced edges.

Case 1(iv). $\{v_\ell, v_{\ell+1}\} \cap \{v_q, v_{q+1}\} = \emptyset$. Similarly to Cases 1(ii) and 1(iii), the chord $v_\ell v_\ell^*$ is colored blue and the C_0 -edge $v_\ell v_{\ell+1}$ is colored black by the Red-Force operation triggered at the end of P^+ . If $v_{\ell+1}v_{\ell+2}$ is uncolored then vertex $v_{\ell+1}$ is a new biased vertex, that is, we have 4 new biased vertices $v_y, v_q, v_\ell, v_{\ell+1}$. Now assume that $v_{\ell+1}v_{\ell+2}$ is a previously colored (i.e. black) C_0 -edge. Then, at the end of P^+ we have a new forcing path starting at vertex v_ℓ with the edge $v_\ell v_{\ell+1}$. At the end of this forcing path, there must be at least one C_0 -edge incident to an unbiased vertex v_z , which becomes black after all Black-Force and Yellow-Force operations. Note that vertex v_q can possibly be one of the internal vertices of this new forcing path. Thus v_z becomes a new biased vertex, that is, we have 4 new biased vertices v_y, v_q, v_ℓ, v_z .

Suppose that *both* edges $v_{\ell+1}v_{\ell+2}$ and $v_{q+1}v_{q+2}$ are not previously colored black. Then, in both instances I_1, I_2 we have at least 4 new forced edges, namely at least one forced edge in each of P^+ and P^- , as well

as the edges $v_\ell v_\ell^*, v_\ell v_{\ell+1}$ for I_1 (resp. the edges $v_q v_q^*, v_q v_{q+1}$ for I_2). Note here this case is only possible to appear when, until the current iteration, vertex v_y is in the center of a path of at least 6 consecutive C_0 -edges that have not been colored yet, namely the edges $v_{y-3}v_{y-2}, v_{y-2}v_{y-1}, v_{y-1}v_y, v_y v_{y+1}, v_{y+1}v_{y+2}, v_{y+2}v_{y+3}$.

Now suppose that at least one of the edges $v_{\ell+1}v_{\ell+2}$ and $v_{q+1}v_{q+2}$ is a previously colored black edge, say this edge is $v_{\ell+1}v_{\ell+2}$ without loss of generality. Then, although in the instance I_2 we have again at least 4 new forced edges, in the instance I_1 we additionally have the chord $v_{\ell+1}v_{\ell+1}^*$ and at least one C_0 -edge incident to $v_{\ell+1}^*$ which are colored yellow and black, respectively. That is, I_1 has in this case at least 6 new forced edges.

Summarizing, in Case 1(iv) we have two instances I_1, I_2 , either each having at least 4 new biased vertices and 4 forced edges, or one instance having at least 4 new biased vertices and 4 forced edges and the other instance having at least 4 new biased vertices and 6 forced edges.

Case 2. Both P^- and P^+ end with a chord. That is, each of these forcing paths has an odd number of *internal* vertices. Since the analysis for both new instances I_1, I_2 is symmetric, here we only analyze instance I_1 , i.e. the case where $v_y v_{y+1}$ becomes red and $v_y v_{y-1}$ becomes black by the operation Blue-Branch. Let v_ℓ be the last vertex of P^+ and let the chord $v_\ell^* v_\ell$ be the last edge of P^+ . Similarly, let v_q be the last vertex of P^- and let the chord $v_q^* v_q$ be the last edge of P^- . That is, v_ℓ^* and v_q^* are the last *internal* vertices of P^+ and of P^- , respectively. Since P^+ and P^- share no common internal vertices by Observation 6, it follows that $v_\ell^* \neq v_q^*$, and thus also $v_\ell \neq v_q$ (as every vertex is incident to exactly one chord). Note that v_q becomes a new biased vertex after the forcing operations along P^- , as it becomes incident to the new black edges $v_q v_{q-1}$ and $v_q v_{q+1}$. That is, v_y and v_q become new biased vertices.

Case 2(i). $v_\ell \in \{v_{q-1}, v_{q+1}\}$, i.e. $v_\ell v_q$ is one of the two C_0 -edges that are incident to v_q . Assume without loss of generality that $v_\ell = v_{q-1}$, or equivalently $v_q = v_{\ell+1}$. Then the C_0 -edges incident to v_ℓ are $v_\ell v_q$ and $v_\ell v_{\ell-1}$. In this case the operation Yellow-Force which is triggered at the end of P^- (once the chord $v_q^* v_q$ is colored yellow) colors both C_0 -edges $v_\ell v_q$ and $v_q v_{q+1}$ black. Thus, since $v_\ell v_q$ becomes black and $v_\ell v_\ell^*$ becomes blue, the operation Blue-Force colors the C_0 -edge $v_\ell v_{\ell-1}$ red. Furthermore, the operation Red-Force colors the chord $v_{\ell-1} v_{\ell-1}^*$ blue. That is, we have at least 8 new forced edges, i.e. at least two forced edges in each of P^+, P^- , as well as the edges $v_\ell v_q, v_q v_{q+1}, v_\ell v_{\ell-1}, v_{\ell-1} v_{\ell-1}^*$.

Furthermore, v_ℓ becomes a new biased vertex as it becomes incident to a new black edge and a new red edge. If v_{q+1} is not incident to any previously colored black C_0 -edge, then v_{q+1} becomes a new biased vertex, that is, we have at least 4 new biased vertices $v_y, v_q, v_\ell, v_{q+1}$. Otherwise, if v_{q+1} is incident to a previously colored black C_0 -edge, then a new forcing path starts at v_{q+1} with the edge $v_{q+1} v_{q+1}^*$. At the end of this forcing path, there must be at least one C_0 -edge incident to an unbiased vertex v_z , which becomes black after all Black-Force and Yellow-Force operations. Thus v_z becomes a new biased vertex, that is, we have at least 4 new biased vertices v_y, v_q, v_ℓ, v_z .

Summarizing, in Case 2(i) we have two instances I_1, I_2 , each having at least 4 new biased vertices and 8 forced edges.

Case 2(ii). $v_q \in \{v_{\ell-1}, v_{\ell+1}\}$. This case is equivalent to Case 2(i).

Case 2(iii). $\{v_\ell, v_{\ell-1}, v_{\ell+1}\} \cap \{v_q, v_{q-1}, v_{q+1}\} = \emptyset$. Assume that both v_{q-1}, v_{q+1} are not incident to any previously colored black vertices. Then both v_{q-1}, v_{q+1} become new biased vertices, i.e. we have in total 4 new biased vertices $v_y, v_q, v_{q-1}, v_{q+1}$. Furthermore we have at least 6 new forced edges, namely at least two forced edges in each of P^+ and P^- , as well as the edges $v_q v_{q-1}, v_q v_{q+1}$.

Now assume that one of the vertices v_{q-1}, v_{q+1} (say v_{q-1}) is incident to a previously colored black vertex and the other one (say v_{q+1}) is not. Then, on the one hand, v_{q+1} becomes a new biased vertex. On the other hand, a new forcing path $P^\#$ starts at vertex v_q with the edge $v_q v_{q-1}$. At the end of this forcing path $P^\#$, there must be at least one C_0 -edge incident to an unbiased vertex v_z , which becomes black after all Black-Force and Yellow-Force operations. Note that vertex v_{q+1} can possibly be one of the internal vertices of this new forcing path $P^\#$. Thus v_z becomes a new biased vertex, that is, we have 4 new biased vertices v_y, v_q, v_{q+1}, v_z . Furthermore we have in this case at least 8 new forced edges, namely at least two forced edges in each of P^+ and P^- , the edges $v_q v_{q-1}, v_q v_{q+1}$, and at least two more new forced edges (at least one chord and one C_0 -edge) in the new forcing path $P^\#$.

Next assume that each of the vertices v_{q-1}, v_{q+1} is incident to a previously colored black vertex. Then,

at vertex v_q we have either two new forcing paths (one starting with the edge $v_q v_{q-1}$ and one starting with the edge $v_q v_{q+1}$) or one forcing cycle (containing the edges $v_q v_{q-1}$ and $v_q v_{q+1}$). If we have one forcing cycle, it must contain at least 5 new forced edges (as G is without loss of generality triangle-free by Theorem 1). This implies that we have in total at least two new biased vertices v_y, v_q and at least 9 new forced edges, namely at least two forced edges in each of P^+ and P^- , as well as at least 5 more forced edges in the forcing cycle at v_q .

Finally assume that we have two forcing paths $P^\#$ and P^* , starting with the edge $v_q v_{q-1}$ and with the edge $v_q v_{q+1}$, respectively. At the end of each of these forcing paths $P^\#$ and P^* , there must be at least one C_0 -edge incident to an unbiased vertex $v_{z^\#}$ and v_{z^*} , respectively, which becomes black after all Black-Force and Yellow-Force operations. Thus each of the vertices $v_{z^\#}$ and v_{z^*} becomes a new biased vertex. Assume that $v_{z^\#} \neq v_{z^*}$. Then we have 4 new biased vertices $v_y, v_q, v_{z^\#}, v_{z^*}$. Furthermore we have at least 10 new forced edges, namely at least two forced edges in each of P^+ and P^- , the edges $v_q v_{q-1}, v_q v_{q+1}$, and at least two more new forced edges (at least one C_0 -edge and one chord) in each of the new forcing paths $P^\#$ and P^* . Now assume that $v_{z^\#} = v_{z^*}$. Then there exists a new forcing path P^{**} starting at $v_{z^\#}$. At the end of this forcing path P^{**} there must be at least one C_0 -edge incident to an unbiased vertex v_z , which becomes black after all Black-Force and Yellow-Force operations. Thus v_z becomes a new biased vertex, that is, we have 4 new biased vertices $v_y, v_q, v_{z^\#}, v_z$. Furthermore, we have in this case at least 10 new forced edges, namely at least two forced edges in each of P^+ and P^- , at least 5 forced edges in $P^\# \cup P^*$, and at least one more forced edge in P^{**} .

Summarizing, in Case 2(iii) we have two instances I_1, I_2 , each of them having either at least 2 new biased vertices and 9 forced edges, or at least 4 new biased vertices and 6 forced edges.

Case 3. P^- ends with a chord and P^+ ends with a C_0 -edge. That is, P^- has an odd number and P^+ has an even number of *internal* vertices. Here the analysis for the two new instances I_1, I_2 is not symmetric, so we will analyze them separately.

Case 3(i). $P^+ \cup P^-$ builds a P_4 (i.e. a path with 4 vertices) whose two endpoints are adjacent. Then the algorithm executes line 17 and calls Procedure 4. As $P^+ \cup P^-$ builds a P_4 , either P^+ contains one edge and P^- contains two edges (see lines 2-4 of Procedure 4), or P^+ contains two edges and P^- contains one edge (see lines 5-7 of Procedure 4). As the analysis of both these cases is symmetric, we only consider here the first case, i.e. that P^+ contains one edge and P^- contains two edges. In this case the algorithm branches to two new instances I_1, I_2 , as follows. In I_1 , the C_0 -edge $v_y v_{y+1}$ becomes red, the C_0 -edges $v_y v_{y-1}, v_{y+1} v_{y+2}, v_{y+2} v_{y+3}$ become black, the chord $v_{y+1} v_{y+1}^*$ becomes blue, and the chord $v_{y-1} v_{y+2}$ becomes yellow. Furthermore the 4 edges $v_y v_{y+1}, v_y v_{y-1}, v_{y-1} v_{y+2}, v_{y+2} v_{y+1}$ are added as an ambivalent quadruple in the set Q (see line 3 of Procedure 4). In I_2 , the C_0 -edges $v_y v_{y-1}, v_{y+2} v_{y+3}$ become red, the C_0 -edges $v_y v_{y+1}, v_{y+1} v_{y+2}$ become black, the chords $v_{y-1} v_{y+2}, v_{y+3} v_{y+3}^*$ become blue, and the chord $v_{y+1} v_{y+1}^*$ becomes yellow. Furthermore, at least one more (previously uncolored) incident C_0 -edge of v_{y+1}^* becomes black by the Yellow-Force operation that is triggered once $v_{y+1} v_{y+1}^*$ becomes yellow. Thus, I_1 has at least 6 new forced edges and I_2 has at least 8 new forced edges.

In I_1 , if v_{y+3} is not incident to any previously colored black edge, it becomes a new biased vertex. That is, in this case we have in I_1 in total at least 4 new biased vertices $v_y, v_{y+1}, v_{y+2}, v_{y+3}$. Otherwise, if v_{y+3} is incident to a previously colored black edge, a new forcing path starts at v_{y+3} . At the end of this forcing path there must be at least one C_0 -edge incident to an unbiased vertex v_z , which becomes black after all Black-Force and Yellow-Force operations. Thus v_z becomes a new biased vertex, that is, we have 4 new biased vertices $v_y, v_{y+1}, v_{y+2}, v_z$.

In I_2 , if v_{y+1}^* is not incident to any previously colored black edge, it becomes a new biased vertex. That is, in this case we have in I_2 in total at least 4 new biased vertices $v_y, v_{y+1}, v_{y+2}, v_{y+1}^*$. Otherwise, if v_{y+1}^* is incident to a previously colored black edge, a new forcing path starts at v_{y+1}^* . At the end of this forcing path there must be at least one C_0 -edge incident to an unbiased vertex v_z , which becomes black after all Black-Force and Yellow-Force operations. Thus v_z becomes a new biased vertex, that is, we have 4 new biased vertices $v_y, v_{y+1}, v_{y+2}, v_z$.

Summarizing, in Case 3(i), I_1 has at least 4 new biased vertices and 6 new forced edges, while I_2 has at least 4 new biased vertices and 8 new forced edges.

Case 3(ii). $P^+ \cup P^-$ does not build a P_4 whose two endpoints are adjacent. Since P^- ends with a chord and P^+ ends with a C_0 -edge by the assumption of Case 3, it easily follows that $P^+ \cup P^-$ does also not build a C_4 . Therefore Algorithm 1 executes 18-22 and branches to two new instances I_1, I_2 , which we analyze separately below. Let v_ℓ be the last vertex of P^+ , and assume without loss of generality that the last edge of P^+ is $v_{\ell-1}v_\ell$ (the other case, where the last edge of P^+ is $v_\ell v_{\ell+1}$ is exactly symmetric). Furthermore, let v_q be the last vertex of P^- and let the chord $v_q^*v_q$ be the last edge of P^- . That is, $v_{\ell-1}$ and v_q^* are the last *internal* vertices of P^+ and of P^- , respectively.

Case 3(ii)(a). $v_\ell \in \{v_{q-1}, v_{q+1}\}$, i.e. $v_\ell v_q$ is one of the two C_0 -edges that are incident to v_q . Assume without loss of generality that $v_\ell = v_{q-1}$, or equivalently $v_q = v_{\ell+1}$. Then the C_0 -edges incident to v_ℓ are $v_\ell v_q$ and $v_\ell v_{\ell-1}$. Note that, if $P^+ \cup P^-$ has three edges, then $P^+ \cup P^-$ builds a P_4 whose two endpoints are adjacent, which is a contradiction to the assumption of Case 3(ii). Thus $P^+ \cup P^-$ has at least 5 edges.

In the instance I_1 , the operation Yellow-Force which is triggered at the end of P^- (once the chord $v_q^*v_q$ is colored yellow) colors both C_0 -edges $v_\ell v_q$ and $v_q v_{q+1}$ black. Furthermore, when the C_0 -edge $v_{\ell-1}v_\ell$ becomes red, the operation Red-Force colors the chord $v_\ell v_\ell^*$ blue. That is, we have at least 8 new forced edges, namely at least 5 forced edges in $P^+ \cup P^-$, as well as the edges $v_\ell v_\ell^*, v_\ell v_q, v_q v_{q+1}$. If $v_{q+1}v_{q+2}$ is not a black edge, then v_{q+1} becomes a new biased vertex, that is, we have the 4 biased vertices $v_y, v_\ell, v_q, v_{q+1}$. Otherwise, if $v_{q+1}v_{q+2}$ is a black edge, then a new forcing path starts at v_{q+1} with the chord $v_{q+1}v_{q+1}^*$. At the end of this forcing path, there must be at least one C_0 -edge incident to an unbiased vertex v_z , which becomes black after all Black-Force and Yellow-Force operations. Thus v_z becomes a new biased vertex, that is, we have 4 new biased vertices v_y, v_ℓ, v_q, v_z .

In the instance I_2 , recall that $v_y v_{y+1}$ becomes black and $v_y v_{y-1}$ becomes red by the operation Blue-Branch. Note that here v_ℓ becomes a new biased vertex after the forcing operations along P^+ , as it becomes incident to the new black edge $v_{\ell-1}v_\ell$. That is, we have at least the 2 new biased vertices v_y, v_ℓ . Furthermore we have at least 5 new forced edges, namely the edges $P^+ \cup P^-$.

Summarizing, in Case 3(ii)(a), I_1 has at least 4 new biased vertices and 8 forced edges, while I_2 has at least 2 new biased vertices and 5 forced edges.

Case 3(ii)(b). $v_\ell \notin \{v_{q-1}, v_{q+1}\}$. In the instance I_2 (i.e. where the C_0 -edges $v_y v_{y+1}$ and $v_y v_{y-1}$ become black and red, respectively) v_ℓ becomes a new biased vertex after the forcing operations along P^+ , as it becomes incident to the new black edge $v_{\ell-1}v_\ell$. That is, we have at least the 2 new biased vertices v_y, v_ℓ . Furthermore we have at least 3 new forced edges, namely at least one forced edge in P^+ and at least two forced edges in P^- . That is, in Case 3(ii)(b), I_2 has at least 2 new biased vertices and 3 forced edges.

Analysis of instance I_1 . Recall that here $v_y v_{y+1}$ becomes red and that $v_y v_{y-1}$ becomes black by the operation Blue-Branch. Note that v_q becomes a new biased vertex after the forcing operations along P^- , as it becomes incident to the new black edges $v_q v_{q-1}$ and $v_q v_{q+1}$. Moreover, note that v_ℓ becomes a new biased vertex after the forcing operations along P^+ , as it becomes incident to the new red edge $v_{\ell-1}v_\ell$ and to the new black edge $v_\ell v_{\ell+1}$. That is, v_y, v_ℓ, v_q become new biased vertices.

Case 3(ii)(b)(1). $v_\ell = v_q$. In this case, after the forcing operations along P^- , vertex v_ℓ is incident to the two black C_0 -edges $v_q v_{q-1}, v_q v_{q+1}$, and thus the edge $v_{\ell-1}v_\ell$ cannot be colored red in the forcing path P^+ , which is a contradiction.

Case 3(ii)(b)(2). $v_{\ell+1} \in \{v_{q-1}, v_{q+1}\}$. Then, since the case $v_{\ell+1} = v_{q+1}$ is equivalent to the case $v_\ell = v_q$ (which has been dealt with in Case 3(I_1)(i)), we assume that $v_{\ell+1} = v_{q-1}$. The Red-Force operation, which is triggered when the last edge $v_{\ell-1}v_\ell$ of P^+ becomes red, colors the C_0 -edge $v_\ell v_{\ell+1}$ black and the chord $v_\ell v_\ell^*$ blue. Furthermore the Yellow-Force operation, which is triggered when the last edge $v_q^*v_q$ of P^- becomes yellow, colors both C_0 -edges $v_q v_{q-1} = v_q v_{\ell+1}$ and $v_q v_{q+1}$ black. Thus vertex $v_{\ell+1}$ becomes a new biased vertex. That is, we have at least 4 new biased vertices $v_y, v_\ell, v_q, v_{\ell+1}$.

Assume that v_{q+1} is incident to a previously colored black edge. Then, once $v_q v_{q+1}$ is colored black, the operation Black-Force is triggered which colors the chord $v_{q+1}v_{q+1}^*$ yellow. On the other hand, once $v_\ell v_{\ell+1}$ and $v_q v_{\ell+1}$ are colored black, the Black-Force and Yellow-Force operations are triggered, and thus the chord $v_{\ell+1}v_{\ell+1}^*$ becomes yellow and at least one of the two C_0 -edges incident to $v_{\ell+1}^*$ becomes black. Thus we have at least 10 new forced edges, namely at least one forced edge in P^+ , at least two forced edges in P^- , the edges $v_\ell v_\ell^*, v_\ell v_{\ell+1}, v_q v_{\ell+1}, v_q v_{q+1}, v_{q+1}v_{q+1}^*, v_{\ell+1}v_{\ell+1}^*$, as well as at least one C_0 -edge incident to $v_{\ell+1}^*$.

Now assume that v_{q+1} is not incident to any previously colored black edge. Then, once $v_\ell v_{\ell+1}$ and $v_q v_{\ell+1}$ are colored black, the Black-Force operation is triggered, which colors the chord $v_{\ell+1} v_{\ell+1}^*$ yellow. Let $v_t = v_{\ell+1}^*$, i.e. the two C_0 -edges incident to $v_{\ell+1}^*$ are $v_{\ell+1}^* v_{t-1}$ and $v_{\ell+1}^* v_{t+1}$. Note that at least one of these edges $v_{\ell+1}^* v_{t-1}$ and $v_{\ell+1}^* v_{t+1}$ is uncolored, as otherwise $v_{\ell+1} v_{\ell+1}^*$ would have been colored yellow at a previous iteration, which is a contradiction. Furthermore note that both vertices v_{t-1} and v_{t+1} are different than v_q , since otherwise G would have a triangle on the vertices $v_{\ell+1}, v_{\ell+1}^*, v_q$, which is a contradiction by Theorem 1.

Suppose that $v_{q+1} \in \{v_{t-1}, v_{t+1}\}$, say without loss of generality that $v_{q+1} = v_{t-1}$. Then the edge $v_{\ell+1}^* v_{t-1} = v_{\ell+1}^* v_{q+1}$ is currently uncolored, as we assumed that v_{q+1} is not incident to any previously colored black edge. Once $v_{\ell+1} v_{\ell+1}^*$ is colored yellow, the operation Yellow-Force is triggered which colors the edge $v_{\ell+1}^* v_{q+1}$ black. Thus, as both $v_{\ell+1}^* v_{q+1}$ and $v_q v_{q+1}$ are colored black, the Black-Force operation is triggered which colors the chord $v_{q+1} v_{\ell+1}^*$ yellow. Thus we have again at least 10 new forced edges, namely at least one forced edge in P^+ , at least two forced edges in P^- , and the edges $v_\ell v_\ell^*, v_\ell v_{\ell+1}, v_q v_{\ell+1}, v_q v_{q+1}, v_{q+1} v_{\ell+1}^*, v_{\ell+1} v_{\ell+1}^*, v_{\ell+1}^* v_{q+1}$.

Finally suppose that $v_{q+1} \notin \{v_{t-1}, v_{t+1}\}$. If both edges $v_{\ell+1}^* v_{t-1}$ and $v_{\ell+1}^* v_{t+1}$ are uncolored, they both become black by the Yellow-Force operation that is triggered at $v_{\ell+1}^*$, once $v_{\ell+1} v_{\ell+1}^*$ becomes yellow. Thus in this case we have at least 10 new forced edges, namely at least one forced edge in P^+ , at least two forced edges in P^- , and the edges $v_\ell v_\ell^*, v_\ell v_{\ell+1}, v_q v_{\ell+1}, v_q v_{q+1}, v_{\ell+1} v_{\ell+1}^*, v_{\ell+1}^* v_{t-1}, v_{\ell+1}^* v_{t+1}$. Otherwise, let one of the edges $v_{\ell+1}^* v_{t-1}$ and $v_{\ell+1}^* v_{t+1}$ (say, the edge $v_{\ell+1}^* v_{t-1}$) be previously colored black. Then we have at least 9 new forced edges, namely at least one forced edge in P^+ , at least two forced edges in P^- , and the edges $v_\ell v_\ell^*, v_\ell v_{\ell+1}, v_q v_{\ell+1}, v_q v_{q+1}, v_{\ell+1} v_{\ell+1}^*, v_{\ell+1}^* v_{t+1}$. If the C_0 -edge $v_{t+1} v_{t+2}$ is uncolored, then v_{t+1} is a new biased vertex, and thus we have at least 5 new biased vertices $v_y, v_\ell, v_q, v_{\ell+1}, v_{t+1}$. Otherwise, if $v_{t+1} v_{t+2}$ is black, then a new forcing path starts at $v_{\ell+1}^*$ with the edge $v_{\ell+1}^* v_{t+1}$. At the end of this forcing path, there must be at least one C_0 -edge incident to an unbiased vertex v_z , which becomes black after all Black-Force and Yellow-Force operations. Thus v_z becomes a new biased vertex, that is, we have again at least 5 new biased vertices $v_y, v_\ell, v_q, v_{\ell+1}, v_z$.

Summarizing, in Case 3(ii)(b)(2), I_1 has either at least 4 new biased vertices and 10 new forced edges, or at least 5 new biased vertices and 9 new forced edges.

Case 3(ii)(b)(3). $v_{\ell+1} = v_q$. This case is equivalent to Case 3(ii)(a).

Case 3(ii)(b)(4). $\{v_\ell, v_{\ell+1}\} \cap \{v_q, v_{q-1}, v_{q+1}\} = \emptyset$. First suppose that none of the three vertices $v_{\ell+1}, v_{q-1}, v_{q+1}$ is incident to any previously colored black edge. Then all these three vertices become new biased vertices, and thus we have in total at least 6 new biased vertices (i.e. together with v_y, v_ℓ, v_q). Furthermore, in this case we have at least 7 new forced edges, namely at least one forced edge in P^+ , at least two forced edges in P^- , as well as the four edges $v_\ell v_\ell^*, v_\ell v_{\ell+1}, v_q v_{q-1}, v_q v_{q+1}$.

Suppose that exactly one of the three vertices $v_{\ell+1}, v_{q-1}, v_{q+1}$ is incident to a previously colored black edge; denote this vertex by v_t . Then, the two vertices in $\{v_{\ell+1}, v_{q-1}, v_{q+1}\} \setminus \{v_t\}$ are new biased vertices, i.e. we have in total at least 5 new biased vertices (together with v_y, v_ℓ, v_q). Furthermore, the Black-Force operation is triggered at v_t , which colors the chord $v_t v_t^*$ yellow. Moreover the Yellow-Force operation is forced at v_t^* , which colors at least one of the C_0 -edges incident to v_t^* black. Thus we have at least 9 new forced edges, namely at least one forced edge in P^+ , at least two forced edges in P^- , as well as the edges $v_\ell v_\ell^*, v_\ell v_{\ell+1}, v_q v_{q-1}, v_q v_{q+1}, v_t v_t^*$ and at least one of the C_0 -edges incident to v_t^* .

Now suppose that exactly two of the three vertices $v_{\ell+1}, v_{q-1}, v_{q+1}$ are incident to a previously colored black edge; denote these vertices by v_{t_1}, v_{t_2} . Then, the single vertex in $\{v_{\ell+1}, v_{q-1}, v_{q+1}\} \setminus \{v_{t_1}, v_{t_2}\}$ is a new biased vertex, i.e. we have in total at least 4 new biased vertices (together with v_y, v_ℓ, v_q). Furthermore, the Black-Force operation is triggered both at v_{t_1} and at v_{t_2} , which color the chords $v_{t_1} v_{t_1}^*$ and $v_{t_2} v_{t_2}^*$ yellow. Moreover, the Yellow-Force operations that are forced at $v_{t_1}^*$ and $v_{t_2}^*$ colors at least one of the C_0 -edges incident to $v_{t_1}^*$ or $v_{t_2}^*$ black (note that this C_0 -edge may be incident to both $v_{t_1}^*$ and $v_{t_2}^*$). Thus we have at least 10 new forced edges, namely at least one forced edge in P^+ , at least two forced edges in P^- , as well as the edges $v_\ell v_\ell^*, v_\ell v_{\ell+1}, v_q v_{q-1}, v_q v_{q+1}, v_{t_1} v_{t_1}^*, v_{t_2} v_{t_2}^*$ and at least one of the C_0 -edges incident to $v_{t_1}^*$ or $v_{t_2}^*$.

Finally suppose that all three vertices $v_{\ell+1}, v_{q-1}, v_{q+1}$ are incident to a previously colored black edge. Then the Black-Force operations that are triggered at these three vertices which color the

chords $v_{\ell+1}v_{\ell+1}^*$, $v_{q-1}v_{q-1}^*$, and $v_{q+1}v_{q+1}^*$ yellow. Thus we have at least 10 new forced edges, namely at least one forced edge in P^+ , at least two forced edges in P^- , as well as the edges $v_{\ell}v_{\ell}^*$, $v_{\ell}v_{\ell+1}$, v_qv_{q-1} , v_qv_{q+1} , $v_{\ell+1}v_{\ell+1}^*$, $v_{q-1}v_{q-1}^*$, $v_{q+1}v_{q+1}^*$. Furthermore, at the end of at least one of the new forcing paths starting at the vertices $v_{\ell+1}$, v_{q-1} , and v_{q+1} with the edges $v_{\ell+1}v_{\ell+1}^*$, $v_{q-1}v_{q-1}^*$, and $v_{q+1}v_{q+1}^*$, respectively, there must be at least one C_0 -edge incident to an unbiased vertex v_z , which becomes black after all Black-Force and Yellow-Force operations. Thus v_z becomes a new biased vertex, that is, we have again at least 4 new biased vertices v_y, v_{ℓ}, v_q, v_z .

Summarizing, in Case 3(ii)(b)(4), I_1 has either at least 6 new biased vertices and 7 new forced edges, or at least 5 new biased vertices and 9 new forced edges, or at least 4 new biased vertices and 10 new forced edges. ■

We are now ready to use the results of our technical Lemma 3 to derive an upper bound for the running time of Algorithm 1.

Theorem 3 *Let G be a Smith graph on n vertices with a given Hamiltonian cycle C_0 . If G does not contain any induced cycle C_6 on 6 vertices, then the running time of Algorithm 1 is $O(n \cdot 2^{0.2971925n})$. Otherwise the running time is $O(n \cdot 2^{(0.3-\varepsilon)n})$, for some constant $\varepsilon > 0$.*

Proof. We derive the running time of the algorithm from the inequalities given in the statement of Lemma 3. These inequalities upper bound the sizes of the sets $W(I_1), W(I_2)$ of unbiased vertices and of the sets $U(I_1), U(I_2)$ of unforced edges in the two instances I_1, I_2 which are obtained in the various cases where the algorithm branches. In fact, for upper-bounding the running time, we can ignore the cases where the algorithm just updates the current instance I (instead of branching to two new instances I_1, I_2), as in these cases the current instance shrinks in a constant number of steps by Lemma 3.

Given an instance I at some iteration of the algorithm, where $|W(I)| = x$ and $|U(I)| = y$, we denote by $f(x, y)$ the worst-case running time needed for the algorithm to compute all the desired alternating red-blue cycles $D = \text{Red} \cup \text{Blue}$ of I , (or to announce “contradiction” in the branches that such a cycle D does not exist). Recall that, with the help of the ambivalent quadruples, these alternating cycles D computed by the algorithm form a succinct encoding of all possible alternating red-blue cycles.

In any of the inequalities of Lemma 3, whenever the instance I is replaced by the instances I_1, I_2 and the values x, y in I are replaced by the values $x - k_1, y - \ell_1$ and by $x - k_2, y - \ell_2$ in I_1 and I_2 , respectively, then the running time $f(x, y)$ of the algorithm at I is replaced by the sum $f(x - k_1, y - \ell_1) + f(x - k_2, y - \ell_2)$ of the running times at I_1 and I_2 , respectively. That is, in the worst case, $f(x, y) \leq f(x - k_1, y - \ell_1) + f(x - k_2, y - \ell_2)$. However, in order to compute an upper bound for $f(x, y)$ from all the recurrences, we need to substitute “ \leq ” by “ \geq ” in each inequality. Thus we obtain the following system of recurrence inequalities immediately by the statement of Lemma 3.

1. $f(x, y) \geq 2f(x - 2, y - 7)$,
2. $f(x, y) \geq 2f(x - 4, y - 4)$,
3. $f(x, y) \geq 2f(x - 2, y - 9)$,
4. $f(x, y) \geq f(x - 4, y - 4) + f(x - 4, y - 6)$,
5. $f(x, y) \geq f(x - 2, y - 9) + f(x - 4, y - 6)$,
6. $f(x, y) \geq f(x - 2, y - 5) + f(x - 4, y - 8)$,
7. $f(x, y) \geq f(x - 2, y - 3) + f(x - 6, y - 7)$,
8. $f(x, y) \geq f(x - 2, y - 3) + f(x - 4, y - 10)$,
9. $f(x, y) \geq f(x - 2, y - 3) + f(x - 5, y - 9)$.

For the sake of presentation, we divide the analysis of the above recurrences into three parts. In Part A we give a naive upper bound on the solution of the above recurrences in worst case. In Part B we give an upper bound in the case where the input Smith graph G does not contain any induced cycle C_6 on 6 vertices. In Part C we present a more sophisticated analysis of the worst case (including C_6 's), thus obtaining an improved upper bound.

Part A: A naive upper bound of the worst case. To solve this system of recurrences, we set $f(x, y) = 2^{\alpha x + \beta y}$ and we compute the optimum values for α, β that satisfy all the above inequalities. As it can be easily verified, the values $\{\alpha = 0.15, \beta = 0.1\}$ satisfy all these inequalities together. Moreover it can be easily checked that, with these values of α, β , the first two relations become *equalities*, while all the other relations become *strict inequalities*. Since $x \leq n$ and $y \leq \frac{3}{2}n$, it follows that $\alpha x + \beta y \leq (\alpha + \frac{3}{2}\beta)n = 0.3 \cdot n$, and thus the running time of the algorithm in the worst case is upper-bounded by $2^{0.3 \cdot n}$.

Part B: The case of C_6 -free graphs. As it can be easily verified, the first inequality $f(x, y) \geq 2f(x-2, y-7)$ can only occur when the algorithm branches according to Case 1(i) in the proof of Lemma 3, and in particular when $P^+ \cup P^-$ is a cycle with *exactly* 6 edges. Now assume that the input Smith graph G does not contain any induced cycle of 6 vertices. Then the first of the above inequalities never occurs, and thus, in order to upper-bound the running time of the algorithm on C_6 -free graphs, it suffices to solve the above recurrence system only for the inequalities 2, 3, \dots , 9. To this end, we set again $f(x, y) = 2^{\alpha x + \beta y}$. As it can be easily verified, the values $\{\alpha = 0.155615, \beta = 0.094385\}$ satisfy all these inequalities together. Therefore, as $\alpha x + \beta y \leq (\alpha + \frac{3}{2}\beta)n = 0.2971925 \cdot n$, and thus the running time of the algorithm on C_6 -free graphs is upper-bounded by $2^{0.2971925 \cdot n}$.

Part C: A more careful analysis of the worst case. As it has been observed in Part A, by setting $f(x, y) = 2^{\alpha x + \beta y}$, the values $\{\alpha = 0.15, \beta = 0.1\}$ satisfy the first two relations as equalities and all the others as strict inequalities. Thus the worst case in Part A can occur when only one of the first two recurrence relations is realized at every iteration of the algorithm. Assume that the first (resp. second) recurrence is applied in total in k (resp. ℓ) branching iterations of the algorithm. Then, at the end of the algorithm we have $2^{k+\ell}$ produced instances. Recall by the analysis of Part B that the first inequality $f(x, y) \geq 2f(x-2, y-7)$ can only occur when the algorithm branches according to Case 1(i) in the proof of Lemma 3, where $P^+ \cup P^-$ is a cycle with *exactly* 6 edges. In this case, exactly four C_0 -edges and three chords are being forced (i.e. colored). In addition, these four newly forced C_0 -edges are either two pairs of consecutive C_0 -edges, or three consecutive C_0 -edges and one separate C_0 -edge (i.e. not consecutive with the other three ones). Furthermore, as it can be easily verified, the second inequality $f(x, y) \geq 2f(x-4, y-4)$ can only occur when the algorithm branches according to Case 1(iv) in the proof of Lemma 3. In this case, exactly three consecutive C_0 -edges and one chord are being forced (i.e. colored). Then, after the k and ℓ applications of the first and the second recurrence, respectively, $4k + 3\ell$ C_0 -edges and $3k + \ell$ chords have been forced. Since there are n C_0 -edges and $\frac{n}{2}$ chords in G , it follows that $4k + 3\ell \leq n$ and $3k + \ell \leq \frac{n}{2}$. From these inequalities it easily follows that the number $2^{k+\ell}$ of produced instances is maximized when $k = \frac{n}{10}$ and $\ell = \frac{n}{5}$, that is, when the algorithm branches $\frac{n}{10}$ (resp. $\frac{n}{5}$) times according to the first (resp. second) recursion rule. Only in this case the algorithm can produce in total $\Theta(2^{\frac{n}{10} + \frac{n}{5}}) = \Theta(2^{0.3n})$ instances.

We will prove that this worst case cannot be realized. To this end, first recall by Case 1(iv) in the proof of Lemma 3 that the second inequality (i.e. when we have exactly 4 new biased vertices and exactly 4 new forced edges) is only possible to appear at a vertex v_y when, until the current iteration, v_y is in the center of a path of at least 6 consecutive C_0 -edges that have not been colored yet. Then, in each of the two resulting new instances I_1, I_2 , exactly three new consecutive C_0 -edges are forced and v_y is an internal vertex of this path of the three C_0 -edges. For simplicity of the presentation, let us call these three C_0 -edges the *imperative path* of this application of the second recursion. Thus, in both I_1, I_2 , between any two imperative paths there is at least one edge that does not belong to any imperative path. Furthermore, in order to achieve $\frac{n}{10}$ applications of the first recursion and $\frac{n}{5}$ applications of the second recursion, at the end of the algorithm every two consecutive imperative paths have to be separated by either one, two, or three consecutive C_0 -edges which are forced by an application of the first reduction.

Now consider two consecutive imperative paths P_1, P_2 in one of the instances at the end of the algorithm,

and assume that P_1, P_2 are separated by the two consecutive C_0 -edges $v_{i-1}v_i, v_iv_{i+1}$; that is, P_1 ends at vertex v_{i-1} and P_2 ends at vertex v_{i+1} . Furthermore, assume without loss of generality that P_1 has been forced during the algorithm *before* P_2 has been forced. Then, since the edges $v_{i-1}v_i, v_iv_{i+1}$ are forced later (i.e. after the edges of P_1, P_2 have been forced) by an operation of the first recursion, these two edges have to belong to a cycle C_6 with 6 edges. In particular, this C_6 must contain the two chords $v_{i-1}v_{i+1}^*$ and $v_{i+1}v_{i-1}^*$, while v_{i-1}^* and v_{i+1}^* must have distance exactly two along C_0 . Let without loss of generality $v_{i-1}^* = v_{k-1}$ and $v_{i+1}^* = v_{k+1}$. Furthermore, due to our assumption above that the first (resp. second) recursion is applied in total $\frac{n}{10}$ (resp. $\frac{n}{5}$) times during the algorithm, it follows that the edges $v_{k-2}v_{k-1}$ and $v_{k+1}v_{k+2}$ belong to two other imperative paths P_3 and P_4 of the second recursion, respectively. Note here that, according to our assumptions above, $P_2 = (v_{i+1}, v_{i+2}, v_{i+3}, v_{i+4})$ and $P_4 = (v_{k+1}, v_{k+2}, v_{k+3}, v_{k+4})$, where $v_{i+1}^* = v_{k+1}$. Furthermore note that some of the paths P_1, P_2 may coincide with some of the paths P_3, P_4 .

Assume that the imperative path P_4 is forced *before* the imperative path P_2 is forced. Then, at the time when the algorithm branches to the second instance in the application of the second reduction at P_2 , the imperative path that replaces P_2 is either the path $(v_i, v_{i+1}, v_{i+2}, v_{i+3})$ or the path $(v_{i+2}, v_{i+3}, v_{i+4}, v_{i+5})$. If P_2 is replaced by the path $(v_i, v_{i+1}, v_{i+2}, v_{i+3})$, then the C_6 that corresponds to the application of the first reduction which forces the edge $v_{i-1}v_i$ must also contain the three consecutive C_0 -edges $v_{k-1}v_k, v_kv_{k+1}, v_{k+1}v_{k+2}$. This is a contradiction, as the edge $v_{k+1}v_{k+2}$ belongs to the imperative path P_4 which has been forced earlier during the algorithm. On the other hand, if P_2 is replaced by the path $(v_{i+2}, v_{i+3}, v_{i+4}, v_{i+5})$, then the C_6 that corresponds to the application of the first reduction which forces the edge $v_{i-1}v_i$ must contain the three consecutive C_0 -edges $v_{i-1}v_i, v_iv_{i+1}, v_{i+1}v_{i+2}$, as well as the C_0 -edge $v_{k-1}v_k$, but *not* the C_0 -edge v_kv_{k+1} . That is, in this case the edge v_kv_{k+1} belongs neither to an application of the first nor to an application of the second recurrence.

In a similar way one can also analyze the case where the imperative path P_4 is forced *after* the imperative path P_2 is forced, as well as the cases where between the imperative paths P_1, P_2 there are three C_0 -edges or just one C_0 -edge. In each of these cases, it follows that in some of the produced instances, the algorithm either forces a specific C_0 -edge in both an application of the first and the second recursion (which is a contradiction), or misses one C_0 -edge (i.e. it does not force it with an application of either the first or the second recursion). That is, when the algorithm only applies the first two reductions (which is the worst case as discussed above), after every at most four applications of the second reduction (i.e. the ones that create the imperative paths P_1, P_2, P_3, P_4), one C_0 -edge is not being forced in some of the produced instances, and thus the algorithm can never produce in total $\Theta(2^{0.3n})$ instances. That is, there exists a strictly positive $\varepsilon > 0$ such that the algorithm produces at most $O(2^{(0.3-\varepsilon)n})$ instances. This completes the analysis of Part C of the proof.

Finally, assume that we have already computed all the desired alternating red-blue *cycles* $D = Red \cup Blue$ of I , each of which, using the ambivalent quadruples, can potentially encode many other alternating red-blue cycles. It remains to compute the running time of each execution of Procedure 2. The symmetric difference $C_0 \Delta D$ can be computed in linear time. Assume that $C_0 \Delta D$ has $k \geq 2$ connected components. Then we create in linear $O(n)$ time a new auxiliary graph $H = (V_H, E_H)$, as follows. The vertex set V_H has one vertex for every connected component (i.e. cycle) of $C_0 \Delta D$, and thus $|V_H| \leq n$. Let $u_1, u_2 \in V_H$ be two different vertices of H , i.e. corresponding to two different connected components of $C_0 \Delta D$. Then, u_1 is adjacent to u_2 in E_H if and only if there exists an ambivalent quadruple which, if flipped, will connect the two corresponding connected components of $C_0 \Delta D$. Lines 1-3 of Procedure 2 can be implemented as follows. We run any linear-time (i.e. $O(n)$ -time) connectivity algorithm on H such as Breadth-First-Search. If H is not connected then no sequence of flips of the ambivalent quadruples can connect the components of $C_0 \Delta D$ into one Hamiltonian graph of G . Otherwise we obtain a spanning tree of H , and in this case the edges of the spanning tree indicate those ambivalent quadruples that need to be flipped in order to make $C_0 \Delta D$ a Hamiltonian graph of G . ■

5 Efficiently computing another long cycle in a Hamiltonian graph

In this section we prove our results on approximating the length of a second cycle on graphs with minimum degree $\delta \geq 3$ and maximum degree Δ . In [1], Bazgan, Santha, and Tuza considered the optimization problem of efficiently (i.e. in polynomial time) constructing a large second cycle different than the given Hamiltonian cycle C_0 in a given Hamiltonian graph G . In particular they proved the following results.

Theorem 4 ([1]) *Let G be an n -vertex cubic Hamiltonian graph and let C_0 be a Hamiltonian cycle of G . Given G and C_0 , for every $\varepsilon > 0$, a cycle $C' \neq C_0$ of length at least $(1-\varepsilon)n$ can be found in time $2^{O(1/\varepsilon^2)} \times n$.*

Theorem 5 ([1]) *Let G be an n -vertex cubic Hamiltonian graph and let C_0 be a Hamiltonian cycle of G . There is an algorithm which, given G and C_0 , computes a cycle $C' \neq C_0$ of length at least $n - 4\sqrt{n}$ in time $O(n^{3/2} \log n)$.*

5.1 Notation and preliminary results

Before we proceed to the main result of the section, we introduce some necessary notation and state preliminary results. Let $G = (V, E)$ be a graph with a designated Hamiltonian cycle $C_0 = (v_1, v_2, \dots, v_n, v_1)$. Two chords of C_0 are *independent* if they do not share an endpoint. The *length* of a chord $v_i v_j$, with $i < j$, is defined as $\min\{j - i, n + i - j\}$. We say that two vertices $u, v \in V$ are *chord-adjacent* if they are connected by a chord of G . Two independent chords e_1 and e_2 are called *crossing* if their endpoints appear in an alternating order around C_0 ; otherwise e_1 and e_2 are called *parallel*.

For $x, y \in V$, we denote by $d(x, y)$ the length of the path from x to y around C_0 . Note that, in general, $d(x, y) \neq d(y, x)$. We define the *distance* between two independent chords xy and ab as follows:

1. if xy and ab are crossing, such that a lies on the path from x to y around C_0 , then $\text{dist}(xy, ab) = \min\{d(x, a) + d(y, b), d(b, x) + d(a, y)\}$;
2. if xy and ab are parallel such that neither y nor b lie on the path from x to a around C_0 , then $\text{dist}(xy, ab) = d(x, a) + d(b, y)$.

In the proof of our main result of this section (see Theorem 6) we use the following two lemmas. The first one is a basic fact from graph theory and the second one is straightforward to check (see Fig. 2 for an illustration).

Lemma 4 [[18], Exercise 3.1.29] *Let $G = (V, E)$ be a bipartite graph of maximum degree Δ . Then G has a matching of size at least $\frac{|E|}{\Delta}$.*

Lemma 5 *Let $G = (V, E)$ be an n -vertex graph with a Hamiltonian cycle C_0 .*

- (1) *If G has a chord of length ℓ , then G contains a cycle $C' \neq C_0$ of length at least $n - \ell + 1$.*
- (2) *If G has two crossing chords e_1, e_2 and $\text{dist}(e_1, e_2) = d$, then G contains a cycle $C' \neq C_0$ of length at least $n - d + 2$.*
- (3) *If G has four pairwise independent chords e_1, e_2, f_1 , and f_2 such that*
 - (a) *e_1, e_2 are parallel and f_1, f_2 are parallel,*
 - (b) *e_i and f_j are crossing for every $i, j \in \{1, 2\}$,*
 - (c) *$\text{dist}(e_1, e_2) = d_1$ and $\text{dist}(f_1, f_2) = d_2$,*

then G contains a cycle $C' \neq C_0$ of length at least $n - d_1 - d_2 + 4$.

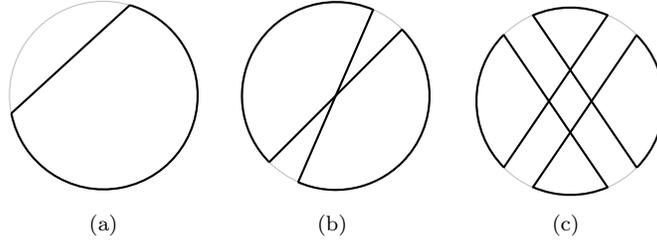


Figure 2: An illustration of Lemma 5. (a) A short chord. (b) A pair of crossing chords. (c) Crossing pairs of parallel chords.

5.2 Long cycles in Hamiltonian graphs

Theorem 6 *Let $G = (V, E)$ be an n -vertex Hamiltonian graph of minimum degree $\delta \geq 3$. Let $C_0 = (v_1, v_2, \dots, v_n, v_1)$ be a Hamiltonian cycle in G and let Δ denote the maximum degree of G . Then G has a cycle $C' \neq C_0$ of length at least $n - 4\alpha(\sqrt{n} + 2\alpha) + 8$, where $\alpha = \frac{\Delta - 2}{\delta - 2}$. Moreover, given C_0 , such a cycle C' can be computed in $O(m)$ time, where $m = |E|$.*

Proof. We start by showing the existence of the desired cycle C' . Without loss of generality we assume that $\alpha < \frac{\sqrt{n}}{2}$, as otherwise any cycle $C' \neq C_0$ in G satisfies the theorem. Furthermore, we assume that the length of every chord in G is at least $4\alpha(\sqrt{n} + 2\alpha) - 6$, as otherwise the existence of C' follows from Lemma 5 (1).

Let $q = \alpha\sqrt{n}$. We partition arbitrarily the vertices of the Hamiltonian cycle C_0 into r consecutive intervals B_0, B_1, \dots, B_{r-1} , such that $r \in \left\{ \left\lfloor \frac{\sqrt{n}}{\alpha} \right\rfloor, \left\lfloor \frac{\sqrt{n}}{\alpha} \right\rfloor + 1 \right\}$ and $\lfloor q \rfloor \leq |B_i| \leq \lfloor q \rfloor + 2\alpha^2$ for every $i \in \{0, 1, \dots, r-1\}$. It is a routine task to check that such a partition exists.

For every $i \in \{0, 1, \dots, r-1\}$ we denote by W_i the set of vertices that are chord-adjacent to a vertex in B_i , and by E_i we denote the set of chords that are incident to a vertex in B_i . Furthermore, we denote by H_i the graph with vertex set $B_i \cup W_i$ and edge set E_i . Since the length of every chord in G is at least $4\alpha(\sqrt{n} + 2\alpha) - 6$, observe that for every $i \in \{0, 1, \dots, r-1\}$, the set W_i is disjoint from $B_{i-1} \cup B_i \cup B_{i+1}$ (where the arithmetic operations with indices are modulo r). The latter, in particular, implies that H_i is a bipartite graph with color classes B_i and W_i .

Let $i, j \in \{0, 1, \dots, r-1\}$ be two distinct indices, we say that the intervals B_i and B_j are *matched* if there exist two independent chords such that each of them has one endpoint in B_i and the other endpoint in B_j . We claim that every interval B_i is matched to another interval B_j for some $j \in \{0, 1, \dots, r-1\} \setminus \{i-1, i, i+1\}$. Indeed, by Lemma 4, graph H_i has a matching M_i of size at least

$$\frac{\lfloor q \rfloor (\delta - 2)}{\Delta - 2} = \frac{\lfloor \alpha\sqrt{n} \rfloor}{\alpha} > \frac{\alpha\sqrt{n} - 1}{\alpha} \geq \sqrt{n} - 1 > \left\lfloor \frac{\sqrt{n}}{\alpha} \right\rfloor - 2 \geq r - 3,$$

and therefore, by the pigeonhole principle, there exists $j \in \{0, 1, \dots, r-1\} \setminus \{i-1, i, i+1\}$ such that at least two edges in M_i have their endpoints in B_j , meaning that B_i is matched to B_j .

Let $\sigma : \{0, 1, \dots, r-1\} \rightarrow \{0, 1, \dots, r-1\}$ be a function such that B_i is matched to $B_{\sigma(i)}$, and denote by $f_{i,1}$ and $f_{i,2}$ some fixed pair of independent chords between B_i and $B_{\sigma(i)}$. We observe that $\text{dist}(f_{i,1}, f_{i,2}) \leq 2(\lfloor q \rfloor + 2\alpha^2 - 1) \leq 2\alpha(\sqrt{n} + 2\alpha) - 2$, as the endpoints of $f_{i,1}$ and $f_{i,2}$ lie in the intervals B_i and $B_{\sigma(i)}$ each of length at most $\lfloor q \rfloor + 2\alpha^2$.

Let now R be an auxiliary graph with a Hamiltonian cycle $(x_0, x_1, \dots, x_{r-1})$ and the chord set being $\{x_i x_{\sigma(i)} : i = 0, 1, \dots, r-1\}$. Let $x_i x_j$ be a chord in R of the minimum length, where $j = \sigma(i)$. Without loss of generality, we assume that $i < j$ and $j - i \leq r + i - j$. Let x_k be a vertex of R such that $i < k < j$ and let $s = \sigma(k)$. Since $x_i x_j$ is of minimum length, the chords $x_i x_j$ and $x_k x_s$ are crossing, and hence each of $f_{i,1}$ and $f_{i,2}$ crosses both $f_{k,1}$ and $f_{k,2}$.

Finally, if $f_{i,1}, f_{i,2}$ or $f_{k,1}, f_{k,2}$ are crossing, then by Lemma 5 (2) there exists a cycle $C' \neq C_0$ of length at least $n - 2\alpha(\sqrt{n} + 2\alpha) + 4$. Otherwise, $f_{i,1}, f_{i,2}$ are parallel and $f_{k,1}, f_{k,2}$ are parallel, and hence by Lemma 5

(3) there exists a cycle $C' \neq C_0$ of length at least $n - 4\alpha(\sqrt{n} + 2\alpha) + 8$, which proves the first part of the theorem.

The above proof is constructive. We now explain at a high level how the proof can be turned into the desired algorithm. First, if $\alpha \geq \frac{\sqrt{n}}{2}$, then we output any cycle formed by a chord and the longer path of C_0 connecting the endpoints of the chord. Otherwise, we partition the vertices of C_0 into the intervals B_1, \dots, B_{r-1} and we assign to each vertex the index of its interval. Clearly, this can be done in $O(n)$ time. Next, we traverse the vertices of G along the cycle C_0 and for every vertex v of an interval B_i we check the chords incident to v . If we encounter a chord f of length less than $4\alpha(\sqrt{n} + 2\alpha) - 6$, then we output the cycle formed by f and the longer path of C_0 connecting the endpoints of f . Otherwise, for the interval B_i we keep the information of how many and which vertices of W_i belong to other intervals B_j for $j \in \{0, 1, \dots, r-1\} \setminus \{i-1, i, i+1\}$. When we find an interval B_j that has at least two elements from W_i , we set $\sigma(i)$ to j and proceed to the first vertex of the next interval B_{i+1} . By doing this, we also keep the information of the current shortest chord in the graph R (defined in the proof above). After finishing this procedure: (1) we have a function $\sigma(\cdot)$; (2) for every $i \in \{0, 1, \dots, r-1\}$ we know a pair $f_{i,1}, f_{i,2}$ of independent edges between B_i and $B_{\sigma(i)}$; and (3) we know k such that $x_k x_{\sigma(k)}$ is a minimum length chord in R . Clearly, this information is enough to identify the desired cycle in constant time. In total, we spent $O(n)$ time to compute the partition of the vertices into the intervals and we visited every chord at most twice, which implies the claimed $O(m)$ running time. ■

The next two corollaries are implied as immediate consequences of Theorem 6, and they provide immediate extensions of the results of [1] and [9], respectively.

Corollary 2 *Let $G = (V, E)$ be an n -vertex Hamiltonian δ -regular graph with $\delta \geq 3$, and let C_0 be a Hamiltonian cycle of G . Then G has a cycle $C' \neq C_0$ of length at least $n - 4\sqrt{n}$, which can be computed in $O(\delta n)$ time.*

Corollary 3 *Let $G = (V, E)$ be an n -vertex Hamiltonian graph of minimum degree $\delta \geq 3$. Let C_0 be a Hamiltonian cycle of G and let Δ denote the maximum degree of G . If $\frac{\Delta}{\delta} = o(\sqrt{n})$, then G has a cycle $C' \neq C_0$ of length at least $n - o(n)$, which can be computed in $O(m)$ time.*

References

- [1] C. Bazgan, M. Santha, and Z. Tuza. On the approximation of finding a(nother) hamiltonian cycle in cubic hamiltonian graphs. *Journal of Algorithms*, 31(1):249–268, 1999.
- [2] R. Bellman. Dynamic programming treatment of the Travelling Salesman Problem. *Journal of the ACM*, 9(1):61–63, 1962.
- [3] A. Bjorklund. Determinant sums for undirected hamiltonicity. *SIAM Journal on Computing*, 43(1):280–299, 2014.
- [4] H. L. Bodlaender, M. Cygan, S. Kratsch, and J. Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Information and Computation*, 243:86–111, 2015.
- [5] K. Cameron. Thomason’s algorithm for finding a second Hamiltonian circuit through a given edge in a cubic graph is exponential on Krawczyk’s graphs. *Discrete Mathematics*, 235:69–77, 2001.
- [6] R. Entringer and H. Swart. Spanning cycles of nearly cubic graphs. *Journal of Combinatorial Theory, Series B*, 29(3):303–309, 1980.
- [7] D. Eppstein. The Traveling Salesman Problem for cubic graphs. *Journal of Graph Algorithms and Applications*, 11(1):61–81, 2007.

- [8] H. Fleischner. Uniqueness of maximal dominating cycles in 3-regular graphs and of Hamiltonian cycles in 4-regular graphs. *Journal of Graph Theory*, 18(5):449–459, 1994.
- [9] A. Girão, T. Kittipassorn, and B. Narayanan. Long cycles in Hamiltonian graphs. *Israel Journal of Mathematics*, 229(1):269–285, 2019.
- [10] R. J. Gould. Recent advances on the Hamiltonian problem: Survey III. *Graphs and Combinatorics*, 30(1):1–46, 2014.
- [11] M. Held and R. M. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210, 1962.
- [12] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer, 1972.
- [13] A. Krawczyk. The complexity of finding a second Hamiltonian cycle in cubic graphs. *Journal of Computer and System Sciences*, 58(3):641–647, 2001.
- [14] M. Liśkiewicz and M. R. Schuster. A new upper bound for the Traveling Salesman Problem in cubic graphs. *Journal of Discrete Algorithms*, 27:1–20, 2014.
- [15] C. H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and system Sciences*, 48(3):498–532, 1994.
- [16] A. G. Thomason. Hamiltonian cycles and uniquely edge colourable graphs. *Advances in Graph Theory*, 3:259–268, 1978.
- [17] W. T. Tutte. On Hamiltonian circuits. *Journal of the London Mathematical Society*, 1(2):98–101, 1946.
- [18] D. West. *Introduction to graph theory*. Prentice hall Upper Saddle River, 2 edition, 2001.
- [19] M. Xiao and H. Nagamochi. An exact algorithm for TSP in degree-3 graphs via circuit procedure and amortization on connectivity structure. *Algorithmica*, 74(2):713–741, 2016.
- [20] M. Xiao and H. Nagamochi. An exact algorithm for TSP in degree-3 graphs via circuit procedure and amortization on connectivity structure. *Algorithmica*, 74(2):713–741, 2016.