Exact and Approximate Algorithms for Computing a Second Hamiltonian Cycle

Argyrios Deligkas
Department of Computer Science, Royal Holloway University of London, Egham, UK
argyrios.deligkas@rhul.ac.uk

George B. Mertzios
Department of Computer Science, Durham University, UK
george.mertzios@durham.ac.uk

Paul G. Spirakis
Department of Computer Science, University of Liverpool, UK
Computer Engineering & Informatics Department, University of Patras, Greece
p.spirakis@liverpool.ac.uk

Viktor Zamaraev
Department of Computer Science, University of Liverpool, Liverpool, UK
viktor.zamaraev@liverpool.ac.uk

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 and William Tutte proved in 1946, using a non-constructive parity argument, that such a second Hamiltonian cycle always exists. Our main result is a deterministic algorithm which computes the second Hamiltonian cycle in $O(n \cdot 2^{0.299862744n}) = O(1.23103^n)$ time and in linear space, thus improving the state of the art running time of $O^*(2^{0.3n}) = O(1.2312^n)$ for solving this problem (among deterministic algorithms running in polynomial space). Whenever the input graph G does not contain any induced cycle C_6 on 6 vertices, the running time becomes $O(n \cdot 2^{0.2971925n}) = O(1.22876^n)$. 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 (not necessarily cubic) 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.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Mathematics of computing → Graph algorithms; Mathematics of computing → Graph theory

Keywords and phrases Hamiltonian cycle, cubic graph, exact algorithm, approximation algorithm.

Digital Object Identifier 10.4230/LIPIcs.MFCS.2020.21

Related Version A full version of the paper is available at `https://arxiv.org/abs/2004.06036`.

Funding *George B. Mertzios*: Supported by the EPSRC grant EP/P020372/1.
Paul G. Spirakis: Supported by the NeST initiative of the EEE/CS School of the University of Liverpool and by the EPSRC grant EP/P02002X/1.

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 [14] 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 [16]. 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 [2,15]. One of the next major improvements came decades later by Eppstein [11] 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 [11]. After that, there has been a series of improvements on the running time for TSP and the Hamiltonian cycle problem in degree-three graphs. In this direction there are two different lines of research, one for algorithms using polynomial space and one for algorithms using exponential space. With respect to algorithms using polynomial space, the most recent results are an $O(1.2553^n)$-time algorithm by Liśkiewicz and Schuster [18] and an $O^*(2^{0.3n}) = O(1.2312^n)$-time algorithm by Xiao and Nagamochi [23], where $O^*(\cdot)$ suppresses polynomial factors. For bounded-degree graphs, it is known by Björklund et al. [5] that TSP can be solved in $O^*((2-\varepsilon)^n)$ time, where $\varepsilon > 0$ only depends on the maximum degree of the input graph. Furthermore, for general graphs there exists a Monte Carlo algorithm for computing a Hamiltonian cycle with running time $O^*(1.657^n)$, given by Björklund [3]. By allowing exponential space, the running time for solving TSP on degree-three graphs can be improved further to $O^*(1.2186^n)$ [6], while a Hamiltonian cycle can also be detected in $O^*(1.1583^n)$ time using a Monte Carlo algorithm [8]. In our paper we focus on algorithms running in polynomial space.

On the other hand, using a non-constructive parity argument, Cedric Smith and William Tutte [21] 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 [19].

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. Similarly, any algorithm $\mathcal{A}'$ which computes the parity of the number of Hamiltonian cycles in a given graph can be also used as a subroutine to solve Smith. Such an algorithm $\mathcal{A}'$, which runs in time $O(1.619^n)$ and in polynomial space, was given by Björklund and Husfeldt [4] for directed graphs, but the result carries over to undirected graphs as well.

Thomason [20] 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 [19]. 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 [7] that on a specific family of cubic graphs (which is a variation of the family introduced by Krawczyk [17]) the lollipop algorithm runs in time at least 2^{cn}, for some constant c. Thus, the state of the art running time (using polynomial space) 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 $O^*(2^{0.3n})$ [23]. 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 a 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 all vertices have degree three except two vertices which have degree four [10, 12]. 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 $n - o(n)$ [13].

Our contribution. In this paper we do the first attempt to understand the internal structure of the lollipop algorithm of Thomason [20]. Our main result in this direction embarks from the following trivial observation, which is not specific to Thomason's algorithm or to cubic graphs.

▶ **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 3 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 among deterministic algorithms running in polynomial space (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 $O^*(2^{0.3n})$, 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 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 eventually achieve is $O(n \cdot 2^{0.299862744n}) = O(1.23103^n)$, while our algorithms runs in linear space. In the particular case where the input graph G contains no induced cycle C_6 on 6 vertices, the running time becomes $O(n \cdot 2^{0.2971925n}) = O(1.22876^n)$.

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 14). On the other hand, we complement the results of [13] 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 15).

Due to space constraints, the missing proofs can be found in the full version of the paper [9].

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 $P = (v_1, v_2, \ldots, v_k)$ (resp. a cycle $C = (v_1, v_2, \ldots, v_k, v_1)$) 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.

▶ **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'.*

Now we define the auxiliary notion of an *X-certificate* which is a pair of chords forming the shape of an "X" in a given Hamiltonian cycle. If an X-certificate exists then a second Hamiltonian cycle can be immediately computed.

▶ **Definition 2.** *Let $G = (V, E)$ be a Smith graph with n vertices and let $C_0 = (v_1, v_2, \ldots, v_n)$ be a given Hamiltonian cycle of G. Let $i, k \in \{1, 2, \ldots, n\}$, where $k \notin \{i-1, i, i+1\}$ (here we consider all indices modulo n), such that $v_i v_k, v_{i+1}v_{k+1} \in E$. Then the pair $\{v_i v_k, v_{i+1}v_{k+1}\}$ of chords is an X-*certificate *of G.*

▶ **Observation 2.** *Let G be a Smith graph with n vertices, let $C_0 = (v_1, v_2, \ldots, v_n)$ be a Hamiltonian cycle of G, and let the pair $\{v_i v_k, v_{i+1}v_{k+1}\}$ of chords be an X-*certificate *of G, where $i < k$. Then $C_1 = (v_1, v_2, \ldots, v_i, v_k, v_{k-1}, \ldots, v_{i+1}, v_{k+1}, v_{k+2}, \ldots, v_n)$ is a second Hamiltonian cycle of 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 [20] 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 $C_0 = (v_1, v_2, \ldots, v_n, v_1)$; 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 $v_1 v_n$ from the cycle C_0, thus obtaining the Hamiltonian path $P_1 = (v_1, v_2, \ldots, v_n)$. We color this removed edge $v_1 v_n$ *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 $P_2 = (v_1, v_2, \ldots, v_k, v_n, \ldots, v_{k+1})$. 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 can be formally given by four coloring rules, which are intuitively described 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).

As we prove in our main technical contribution in this section (see Theorem 3), the coloring of the edges proceeds such that the following main invariant is maintained:

▶ **Main Invariant.** *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.*

▶ **Theorem 3.** *The Main Invariant 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.*

The next corollary follows by the proof of Theorem 3, and will allow us to reduce the asymptotic running time of our algorithm in Section 4 by a factor of n.

▶ **Corollary 4.** *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.*

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 (among algorithms running in polynomial space). Our algorithm is inspired by the structural property of Theorem 3. 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, we first present some necessary definitions and notation. Let G be a Smith graph and $C_0 = (v_1, v_2, \ldots, v_n)$ 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_iv_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_iv_{i+1}, where we consider all indices modulo n. The algorithm 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_1v_t *red*, where $v_t \in \{v_2, v_n\}$, the chord $v_tv_t^*$ *blue*, and the two C_0-edges adjacent to the edge v_tv_1 *black*. That is, if $v_t = v_2$ (resp. if $v_t = v_n$) then the edges v_1v_n and v_2v_3 (resp. v_1v_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_1v_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 $v_1v_1^*$ 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_xv_y be the last (blue) edge of D. For each of the two C_0-edges $v_{y-1}v_y$ and v_yv_{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_xv_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_yv_{y+1}, v_yv_{y-1} are eligible, we create two new instances I_1 and I_2, where I_1 (resp. I_2) has the edge v_yv_{y+1} (resp. v_yv_{y-1}) colored red and the edge v_yv_{y-1} (resp. v_yv_{y+1}) colored black.
- **Blue-Force:** Whenever a chord v_xv_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_yv_{y+1}, v_yv_{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_iv_{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 $v_iv_i^*$ (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 $v_iv_i^*$ is colored yellow by the operation Black-Force (i.e. once both C_0-edges $v_{i-1}v_i, v_iv_{i+1}$ become black); furthermore let $v_k = v_i^*$. If at least one of the C_0-edges $v_{k-1}v_k, v_kv_{k+1}$ has been previously colored red, we announce "contradiction". Otherwise, for each of the C_0-edges $v_{k-1}v_k, v_kv_{k+1}$, 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 $v_{i-1}v_i, v_iv_{i+1}, v_{k-1}v_k, v_kv_{k+1}$ that are incident to the chord $v_iv_i^*$ are colored black.)

The main idea of the algorithm is as follows. If both edges v_yv_{y+1}, v_yv_{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_yv_{y+1} (resp. v_yv_{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 four 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 all colored edges in the two different instances I_1, I_2 are identical, apart from the colors of these four 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 four edges. Furthermore, as it turns out, every such a quadruple 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.

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 [23]. 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 $O^*\left(2^{\frac{3}{2}\cdot\frac{1}{5}n}\right) = O^*\left(2^{0.3n}\right)$ 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 $O\left(2^{(0.3-\varepsilon)n}\right)$ 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. Now we define the operation Ambivalent-Flip, which appropriately changes at the end of the algorithm the already chosen colors of an ambivalent quadruple. 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 6 and 7), 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 the algorithm, 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 5** (forcing path and cycle). *Let G be a Smith graph. At an arbitrary iteration of the algorithm, a path $P = (v_{i_1}, v_{i_2}, \ldots, v_{i_k})$ of G is a* forcing path starting at vertex v_{i_1} *if:*

- *each of its edges $v_{i_1}v_{i_2}, \ldots, v_{i_{k-1}}v_{i_k}$ is yet uncolored and*
- *each of its first $k-1$ vertices $v_{i_1}, \ldots, v_{i_{k-1}}$ 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 $C = (v_{i_1}, v_{i_2}, \ldots, v_{i_k}, v_{i_1})$ of G is a forcing cycle *if:*

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

In the next lemma (Lemma 6) we prove the correctness of our algorithm, and after that we prove our crucial technical Lemma 7 which specifies how the current instance is transformed in one iteration of the 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 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, while all other vertices in $V - W(I)$ are referred to as *unbiased* vertices.

▶ **Lemma 6.** *Let $G = (V, E)$ be a Smith graph and C_0 be a Hamiltonian cycle of G. Then, the algorithm correctly computes a second Hamiltonian cycle C_1 of G on the input $I = (G, C_0, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$.*

▶ **Lemma 7.** *Let $I = (G, C_0, Q, Red, Blue, Black, Yellow)$ be the instance at some iteration of the algorithm, 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 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* $|W(I_2)| \leq |W(I)| - 5$, $|U(I_2)| \leq |U(I)| - 9$.

We are now ready to use the results of our technical Lemma 7 to derive an upper bound for the running time of the algorithm.

▶ **Theorem 8.** *Let G be a Smith graph on n vertices with a given Hamiltonian cycle C_0. Then the algorithm runs in $O(n \cdot 2^{0.299862744n}) = O(1.23103^n)$ time and in linear space. If G does not contain any induced cycle C_6 on 6 vertices, then the running time becomes $O(n \cdot 2^{0.2971925n}) = O(1.22876^n)$.*

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 9** ([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 10** ([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 $C_0 = (v_1, v_2, \ldots, v_n, v_1)$. 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 $\mathrm{dist}(xy, ab) = \min\{d(x, a) + d(y, b), d(b, x) + d(a, y)\}$;
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 $\mathrm{dist}(xy, ab) = d(x, a) + d(b, y)$.

In the proof of our main result of this section (see Theorem 13) 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 Figure 1 for an illustration).

▶ **Lemma 11.** *[[22], 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 12.** *Let $G = (V, E)$ be an n-vertex graph with a Hamiltonian cycle C_0.*

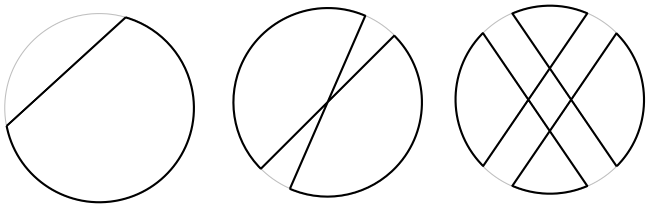
(1) If G has a chord of length ℓ, then G contains a cycle $C' \neq C_0$ of length at least $n - \ell + 1$.

(2) If G has two crossing chords e_1, e_2 and $\mathrm{dist}(e_1, e_2) = d$, 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. $\mathrm{dist}(e_1, e_2) = d_1$ *and* $\mathrm{dist}(f_1, f_2) = d_2$,
then G contains a cycle $C' \neq C_0$ of length at least $n - d_1 - d_2 + 4$.



(a) A short chord. **(b)** A pair of crossing chords. **(c)** Crossing pairs of parallel chords.

Figure 1 An illustration of Lemma 12.

5.2 Long cycles in Hamiltonian graphs

▶ **Theorem 13.** *Let $G = (V, E)$ be an n-vertex Hamiltonian graph of minimum degree $\delta \geq 3$. Let $C_0 = (v_1, v_2, \ldots, v_n, v_1)$ 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 12 (1).

Let $q = \alpha\sqrt{n}$. We arbitrarily partition the vertices[1] of the Hamiltonian cycle C_0 into r consecutive intervals $B_0, B_1, \ldots, B_{r-1}$, 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 $i \in \{0, 1, \ldots, r-1\}$. It is a routine task to check that such a partition exists.

For every $i \in \{0, 1, \ldots, r-1\}$ 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 $i \in \{0, 1, \ldots, r-1\}$, 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 $i, j \in \{0, 1, \ldots, r-1\}$ 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 $j \in \{0, 1, \ldots, r-1\} \setminus \{i-1, i, i+1\}$. Indeed, by Lemma 11, graph H_i has a matching M_i of size at least

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

[1] More formally, we partition the interval $[1, n]$ into the consecutive intervals $B_0, B_1, \ldots, B_{r-1}$, which immediately implies a partition of the vertices of the Hamiltonian cycle C_0.

and therefore, by the pigeonhole principle, there exists $j \in \{0, 1, \ldots, r-1\} \setminus \{i-1, i, i+1\}$ 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 $\sigma : \{0, 1, \ldots, r-1\} \to \{0, 1, \ldots, r-1\}$ 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 $\text{dist}(f_{i,1}, f_{i,2}) \leq 2\big(\lfloor q \rfloor + 2\alpha^2 - 1\big) \leq 2\alpha(\sqrt{n} + 2\alpha) - 2$, 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 $(x_0, x_1, \ldots, x_{r-1})$ and the chord set being $\{x_i x_{\sigma(i)} : i = 0, 1, \ldots, r-1\}$. 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 12 (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 12 (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 $B_1, \ldots, B_{r-1}$ 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 $j \in \{0, 1, \ldots, r-1\} \setminus \{i-1, i, i+1\}$. 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 $i \in \{0, 1, \ldots, r-1\}$ 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 13, and they provide immediate extensions of the results of [1] and [13], respectively.

▶ **Corollary 14.** *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 15.** *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 Cristina Bazgan, Miklos Santha, and Zsolt Tuza. On the approximation of finding a(nother) hamiltonian cycle in cubic hamiltonian graphs. *Journal of Algorithms*, 31(1):249–268, 1999.

2 Richard Bellman. Dynamic programming treatment of the Travelling Salesman Problem. *Journal of the ACM*, 9(1):61–63, 1962.

3 Andreas Bjorklund. Determinant sums for undirected hamiltonicity. *SIAM Journal on Computing*, 43(1):280–299, 2014.

4 Andreas Björklund and Thore Husfeldt. The parity of directed Hamiltonian cycles. In *Profeecings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 727–735, 2013.

5 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. The traveling salesman problem in bounded degree graphs. *ACM Transactions on Algorithms*, 8(2):18:1–18:13, 2012.

6 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Information and Computation*, 243:86–111, 2015.

7 Kathie 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.

8 Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast hamiltonicity checking via bases of perfect matchings. In *Proceedings of the 45th ACM Symposium on Theory of Computing Conference (STOC)*, pages 301–310, 2013.

9 Argyrios Deligkas, George B. Mertzios, Paul G. Spirakis, and Viktor Zamaraev. Exact and approximate algorithms for computing a second hamiltonian cycle. *CoRR*, 2020. Available online at `https://arxiv.org/abs/2004.06036`.

10 R.C. Entringer and Henda Swart. Spanning cycles of nearly cubic graphs. *Journal of Combinatorial Theory, Series B*, 29(3):303–309, 1980.

11 David Eppstein. The Traveling Salesman Problem for cubic graphs. *Journal of Graph Algorithms and Applications*, 11(1):61–81, 2007.

12 Herbert 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.

13 António Girão, Teeradej Kittipassorn, and Bhargav Narayanan. Long cycles in Hamiltonian graphs. *Israel Journal of Mathematics*, 229(1):269–285, 2019.

14 Ronald J. Gould. Recent advances on the Hamiltonian problem: Survey III. *Graphs and Combinatorics*, 30(1):1–46, 2014.

15 Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210, 1962.

16 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer, 1972.

17 Adam Krawczyk. The complexity of finding a second Hamiltonian cycle in cubic graphs. *Journal of Computer and System Sciences*, 58(3):641–647, 2001.

18 Maciej Liśkiewicz and Martin R Schuster. A new upper bound for the Traveling Salesman Problem in cubic graphs. *Journal of Discrete Algorithms*, 27:1–20, 2014.

19 Christos 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.

20 Andrew G. Thomason. Hamiltonian cycles and uniquely edge colourable graphs. *Advances in Graph Theory*, 3:259–268, 1978.

21 William T. Tutte. On Hamiltonian circuits. *Journal of the London Mathematical Society*, 1(2):98–101, 1946.

22 Douglas West. *Introduction to graph theory*. Prentice hall Upper Saddle River, 2 edition, 2001.

23 Mingyu Xiao and Hiroshi Nagamochi. An exact algorithm for TSP in degree-3 graphs via circuit procedure and amortization on connectivity structure. *Algorithmica*, 74(2):713–741, 2016.