Exact Algorithms for Finding Longest Cycles in Claw-Free Graphs*

Hajo Broersma[1], Fedor V. Fomin[2], Pim van 't Hof[1], and Daniël Paulusma[1]

[1]School of Engineering and Computing Sciences, Durham University,
Science Laboratories, South Road, Durham DH1 3LE, England**.
{hajo.broersma,pim.vanthof,daniel.paulusma}@durham.ac.uk
[2]Department of Informatics, University of Bergen,
P.O. Box 7800, N-5020 Bergen, Norway.
fomin@ii.uib.no

Abstract. The Hamiltonian Cycle problem is the problem of deciding whether an n-vertex graph G has a cycle passing through all vertices of G. This problem is a classic NP-complete problem. Finding an exact algorithm that solves it in $\mathcal{O}^*(\alpha^n)$ time for some constant $\alpha < 2$ is a notorious open problem. The Longest Cycle problem, in which the task is to find a cycle of maximum length, is a natural generalization of the Hamiltonian Cycle problem. For a claw-free graph G, finding a longest cycle is equivalent to finding a closed trail (i.e., a connected even subgraph, possibly consisting of a single vertex) that dominates the largest number of edges of some associated graph H. Using this translation we obtain two exact algorithms that solve the Longest Cycle problem, and consequently the Hamiltonian Cycle problem, for claw-free graphs: one algorithm that uses $\mathcal{O}^*(1.6818^n)$ time and exponential space, and one algorithm that uses $\mathcal{O}^*(1.8878^n)$ time and polynomial space.

1 Introduction

We study exact algorithms for the Longest Cycle problem: given a graph G, find a cycle in G with the largest number of vertices. This problem generalizes the well-known NP-complete decision problem Hamiltonian Cycle (cf. [10]) that asks whether a graph G has a *hamiltonian cycle*, i.e., a cycle passing through all vertices of G.

The Hamiltonian Cycle problem can be seen as a special case of the well-known Traveling Salesman problem. The input of the latter problem is a complete graph together with an edge weighting. The goal is to find a hamiltonian cycle of minimum total weight. Held and Karp [14] present a classic dynamic programming algorithm that solves the Traveling Salesman problem in $\mathcal{O}^*(2^n)$ time and $\mathcal{O}^*(2^n)$ space for graphs on n vertices. The $\mathcal{O}^*$-notation

* An extended abstract of this paper has been presented at WG 2009 [3].
** This work has been supported by EPSRC (EP/D053633/1).

indicates that we suppress factors of polynomial order, and we use this notation throughout the paper. Polynomial-space algorithms for the Hamiltonian Cycle problem (which is a special case of the Traveling Salesman) were rediscovered several times [17, 16, 1]. It is a major and long outstanding open problem whether the Hamiltonian Cycle problem, and more generally, the Traveling Salesman problem, can be solved in $\mathcal{O}^*(\alpha^n)$ time for some constant $\alpha < 2$.

For some graph classes for which the Hamiltonian Cycle, and consequently the Traveling Salesman problem, remains NP-complete, faster exact algorithms have been designed. For planar graphs, and more generally, on graphs excluding some fixed graph as a minor, the Hamiltonian Cycle problem can be solved in $\mathcal{O}^*(c^{\sqrt{n}})$ for some constant c (cf. [6, 7, 24]). The Traveling Salesman problem can be solved in $\mathcal{O}^*(1.251^n)$ time for cubic graphs [15] and in $\mathcal{O}^*(1.890^n)$ time for graphs with maximum degree 4 [8]. Both algorithms use polynomial space. For graphs with maximum degree 4, an algorithm with time complexity $\mathcal{O}^*(1.733^n)$ is known [11], but this algorithm uses exponential space. More generally, Björklund et al. [2] present an algorithm that solves the Traveling Salesman problem in $\mathcal{O}^*((2-\epsilon)^n)$ for graphs with bounded degree, where $\epsilon > 0$ only depends on the maximum degree but not on the number of vertices. They show that this bound can be improved further for regular triangle-free graphs. These algorithms use exponential space. They also present a $\mathcal{O}^*((2-\epsilon)^n)$ time algorithm that uses polynomial space for bounded degree graphs in which the edges have bounded integer weights.

Our Results. We consider the class of claw-free graphs. This is a rich class containing, e.g., the class of line graphs and the class of complements of triangle-free graphs. It is also an intensively studied graph class, both within structural graph theory and within algorithmic graph theory; see [9] for a survey. The Hamiltonian Cycle problem is NP-complete for claw-free graphs; moreover, the problem remains NP-complete even on 3-connected cubic planar claw-free graphs [18]. This immediately implies that the Longest Cycle problem is NP-hard for claw-free graphs as well. In this work we show that on claw-free graphs the Longest Cycle problem can be solved significantly faster than in time $\mathcal{O}^*(2^n)$. We present two exact algorithms. Our first algorithm uses $\mathcal{O}^*(1.6818^n)$ time and exponential space, and our second algorithm uses $\mathcal{O}^*(1.8878^n)$ time and polynomial space. Our techniques are based on a (known) transformation of the problem into the problem of finding an optimum closed trail (i.e., a closed trail dominating the largest number of edges) of an associated graph, and a new study of structural properties of such trails. These techniques are different from the ones used in the already known algorithms, and are of independent interest.

Paper organization. Some basic definitions are presented in Section 2. Section 3 contains some structural results on closed trails, which will be used in the exact algorithms described in Section 4 and Section 5. In Section 4 we translate the problem of finding a longest cycle in a claw-free graph into the problem of finding an optimum closed trail of an associated triangle-free graph. Two exact

algorithms for finding such an optimum closed trail are presented in Section 5. Section 6 contains the conclusions and mentions some open problems.

2 Preliminaries

All graphs in this paper are finite, undirected and without multiple edges and loops. For notation and terminology not defined in this paper we refer to [5]. We denote the vertex set and edge set of a graph G by $V(G)$ and $E(G)$, respectively, and we assume throughout the paper that all graphs we consider have a nonempty vertex set. The *neighborhood* of a vertex v in G is denoted by $N_G(v) := \{w \in V(G) \mid vw \in E(G)\}$, and $d_G(v) = |N_G(v)|$ denotes the *degree* of v. The maximum degree among the vertices of a graph G is denoted by $\Delta(G)$. A *2-factor* of G is a spanning subgraph of G in which all vertices have degree 2. The subgraph of G induced by some nonempty subset $U \subseteq V(G)$ is denoted by $G[U]$. For any proper subset $S \subset V(G)$, we write $G - S$ to denote the graph $G[V(G) \setminus S]$, i.e., the graph obtained from G by removing all vertices of S. If $S = \{v\}$, we write $G - v$ instead of $G - \{v\}$. Similarly, for any set $S \subseteq E(G)$, the graph $G - S$ is the graph obtained from G by removing all edges of S.

A graph is called *triangle-free* if it does not contain a subgraph isomorphic to the cycle on three vertices. A graph is called *claw-free* if it has no induced subgraph isomorphic to the *claw*, i.e., the four-vertex star $K_{1,3} = (\{u, a, b, c\}, \{ua, ub, uc\})$. Let G be a claw-free graph. Then, for each vertex v of G, the set of neighbors of v in G induces a subgraph with at most two components. If this subgraph has two components, both of them must be cliques. If the subgraph induced by $N_G(v)$ is connected but not complete, we can perform an operation called *local completion of G at v* by adding edges joining all pairs of nonadjacent vertices in $N_G(v)$.

The *line graph* of a graph H with edges $e_1, \ldots, e_p$ is the graph $L(H)$ with vertices $u_1, \ldots, u_p$ such that there is an edge between any two vertices u_i and u_j if and only if e_i and e_j share one end vertex in H. Note that $L(K_3) = L(K_{1,3}) = K_3$; it is well-known that every connected line graph $F \neq K_3$ has a unique H with $F = L(H)$ (see e.g. [12]). We call H the *preimage graph* of F. For K_3 we let $K_{1,3}$ be its preimage graph.

A graph is called *even* if all its vertices have even degree. A graph is called a *closed trail* if it is a connected even graph. Note that an even graph, and a closed trail in particular, might consist of a single vertex. A closed trail that does not consist of a single vertex is called *nontrivial*; note that a nontrivial closed trail contains at least three vertices. Let T be a closed trail of a graph H. An edge $e \in E(H)$ is *dominated by T* if T contains at least one of the end vertices of e. In this context "dominated" means "edge-dominated", and this is the case whenever we speak of domination in this paper. Note that, by definition, every edge of a nontrivial closed trail T is dominated by T itself. For any closed trail T of H, we denote by $\beta(T)$ the number of edges of H dominated by T, i.e., $\beta(T) := |E(H) \setminus E(H - V(T))|$. If every edge of H is dominated by T, i.e., if $\beta(T) = |E(H)|$, then we say that T is a *dominating closed trail* of H. An

optimum nontrivial closed trail or ONCT of H is a nontrivial closed trail of H that dominates at least as many edges of H as any other nontrivial closed trail of H. A closed trail T of a graph H is called an *optimum closed trail* or OCT if $\beta(T) \geq \beta(T')$ for any closed trail T' of H. Note that every graph has an OCT, and that an OCT of H is either an ONCT of H, or a single vertex with degree $\Delta(H)$ in case $\Delta(H) \geq \beta(T)$ for any nontrivial closed trail T of H.

For any integer $k \geq 1$, a graph H is called *k-degenerate* if every subgraph of H (including H itself) has a vertex with degree at most k. We say that H is *k-ordered* if H allows a vertex ordering $v_1, \ldots, v_{|V(H)|}$ such that, for $1 \leq i \leq |V(H)|$, the graph $H[\{v_1, \ldots, v_i\}]$ is connected and v_i has at most k neighbors in $H[\{v_1 \ldots, v_i\}]$.

3 Closed Trails of Low Degeneracy and Ordering

In this section we study structural properties of closed trails. We will use such properties in the exact algorithms for the OCT problem presented in the next two sections. A cycle C of a connected graph H is called *removable* if the graph $H - E(C)$ is connected, and *non-separating* if $H - V(C)$ is connected. The following result is due to Thomassen and Toft [23].

Theorem 1 ([23]). *Every connected graph with minimum degree at least 3 has an induced non-separating cycle.*

Theorem 1 implies the following result.

Corollary 1. *Every connected graph with minimum degree at least 3 has an induced removable cycle.*

Proof. Let H be a connected graph with minimum degree at least 3. By Theorem 1, H has an induced non-separating cycle C. Since $H - V(C)$ is connected, all vertices of $V(H) \setminus V(C)$ belong to the same component of $H - E(C)$. Since H has minimum degree at least 3 and C is an induced cycle, every vertex of C has a neighbor in $V(H) \setminus V(C)$. Hence $H - E(C)$ is connected, so C is removable. □

Using Corollary 1, we can prove the following result.

Lemma 1. *Every closed trail contains a 2-degenerate spanning closed trail.*

Proof. Since a closed trail consisting of a single vertex is 2-degenerate, the lemma holds for such closed trails. We claim that every nontrivial closed trail contains a 2-degenerate spanning closed trail. Let H be a counterexample to this claim with $|E(H)|$ minimum, i.e., H is a nontrivial closed trail which does not contain a 2-degenerate spanning closed trail. In particular, H itself is not 2-degenerate. We repeatedly remove vertices from H with degree at most 2 in the current subgraph of H as long as possible. Let H' be the subgraph of H we obtain this way. Since H is not 2-degenerate, H' indeed exists. Let H_1 be a component of H'. Since H' has minimum degree at least 3, H_1 has a removable cycle C by Corollary 1.

Then C is also a removable cycle in H, since H is a connected supergraph of H_1. Hence the graph $H - E(C)$ is a spanning nontrivial closed trail of H. Since H is a counterexample, $H - E(C)$ is not 2-degenerate and $H - E(C)$ does not contain a 2-degenerate spanning closed trail. But then $H - E(C)$ is a counterexample to the claim that every nontrivial closed trail contains a 2-degenerate spanning closed trail, contradicting the minimality of H. □

The next lemma shows that the notions of degeneracy and ordering are closely related.

Lemma 2. *Every connected k-degenerate graph is $(k+1)$-ordered, for any $k \geq 1$.*

Proof. Let H be a connected k-degenerate graph, and suppose for contradiction that H is not $(k+1)$-ordered. We repeatedly remove vertices from H with degree at most $k+1$ in the current subgraph of H, until we cannot remove any vertex with degree at most $k+1$ without making the current subgraph disconnected. Let H' be the resulting (connected) subgraph of H. Since H is not $(k+1)$-ordered, H' indeed exists. Let U consist of all vertices with degree at most k in H'. By our procedure, every vertex of U is a cut vertex of H', and since H is k-degenerate, U is not empty. Hence H' contains at least one cut vertex. Let D be an end-block of H', i.e., a maximal 2-connected subgraph of H' containing exactly one cut vertex x of H'. By our procedure, every vertex of $D - x$ has degree at least $k+2$ in H', which means that every vertex of $D - x$ has degree at least $k+1$ in $D - x$. Since $D - x$ is a subgraph of H, this contradicts the k-degeneracy of H. □

It is well-known that a connected graph is 1-degenerate if and only if it is a tree. It is not hard to see that every tree, and therefore every connected 1-degenerate graph, is 1-ordered. This means that Lemma 2 can be slightly strengthened for $k = 1$. The following result shows that Lemma 2 is best possible for $k \geq 2$.

Proposition 1. *For any $k \geq 2$, there exists a connected k-degenerate graph that is not k-ordered.*

Proof. For any $k \geq 2$, let G_k be the graph constructed as follows. Start with the join of C_{k+2} and $\overline{K_{k-1}}$, i.e., the graph obtained from the disjoint union of a cycle of length $k+2$ and an independent set S on $k-1$ vertices by making every vertex of the cycle adjacent to every vertex of S. Let H be the graph obtained from this graph by removing one edge cw, where c is a vertex of the cycle and w is a vertex of S. Take k copies $H_1, \ldots, H_k$ of the graph H, and let $c_1, \ldots, c_k$ denote the copies of vertex c in $H_1, \ldots, H_k$, respectively. Finally, G_k is obtained by adding a vertex x and edges $xc_1, \ldots, xc_k$. As an example, the graph G_3 is depicted in Figure 1.

It is straightforward to verify that G_k is k-degenerate. We claim that G_k is not k-ordered. For contradiction, suppose G_k is k-ordered. By definition, G_k has an ordering $v_1, \ldots, v_{|V(G_k)|}$ of its vertices such that, for $1 \leq i \leq |V(G_k)|$, the graph $G_k[\{v_1, \ldots, v_i\}]$ is connected and v_i has at most k neighbors in $G_k[\{v_1 \ldots, v_i\}]$.

Since x is the only vertex of G_k with degree at most k in G_k, $x = v_{|V(G_k)|}$. But the fact that x is a cut vertex of G_k implies that the graph $G_k[\{v_1, \ldots, v_i\}]$ is not connected for $i = |V(G_k)| - 1$, yielding the desired contradiction. □

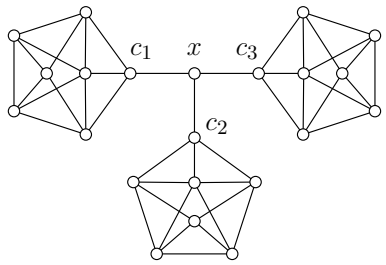

Fig. 1. The graph G_3, which is 3-degenerate but not 3-ordered.

Lemma 1 and Lemma 2 together imply the following result, which will be used in the exact algorithms described in the next two sections.

Corollary 2. *Every graph has a 2-degenerate 3-ordered optimum closed trail.*

Proof. Let T be an optimum closed trail of a graph H, and let $S \subseteq E(H)$ be the set of edges of H that are dominated by T. By Lemma 1, the graph T contains a 2-degenerate spanning closed trail T'. Since $V(T') = V(T)$, the set of edges of H dominated by T' is exactly the set S. Hence T' is an optimum closed trail of H. Since T' is 2-degenerate, T' is 3-ordered as a result of Lemma 2. □

4 Two Exact Algorithms for Finding a Longest Cycle

In this section we explain our two algorithms that solve the Longest Cycle problem for a claw-free graph G on n vertices. We assume from now on that G is connected, since we can treat the components of G separately in case G is disconnected. We also assume that G contains a longest cycle, i.e., that G is not a tree. Note that we can check in polynomial time whether G is a connected graph other than a tree.

For the first, third and fourth step below we do not have to develop any new theory or algorithms, but can rely on the beautiful existing machinery from the literature.

Step 1: restrict to the preimage graph H of the closure of G

We recursively repeat the local completion operation, as long as this is possible. This way we obtain the *closure* $cl(G)$ of G. Ryjáček [22] showed that the closure of G is uniquely determined, i.e., that the ordering in which one performs the

local completions does not matter. This means we can obtain $cl(G)$ in polynomial time. Ryjáček [22] also showed that the length of a longest cycle in G equals the length of a longest cycle in $cl(G)$. In particular, G is hamiltonian if and only if $cl(G)$ is hamiltonian. Furthermore he showed that for any claw-free graph G there is a unique (triangle-free) graph H such that $L(H) = cl(G)$. We can obtain the preimage graph of a line graph in polynomial time (see e.g. [21]). Hence we can compute the unique graph H with $L(H) = cl(G)$ in polynomial time.

Step 2: find an OCT of H

Harary and Nash-Williams [13] showed that a hamiltonian cycle in a line graph of any connected graph on at least three vertices corresponds to a dominating closed trail of the graph itself. By an easy variation on their arguments, many researchers have shown that a longest cycle in such a line graph corresponds to an optimum closed trail of the graph itself. This result, combined with the results from the previous step, implies that finding a longest cycle in G corresponds to finding an OCT of H. In Section 5 we present two exact algorithms for finding an OCT of a connected graph with n edges: one algorithm that uses $\mathcal{O}^*(1.6818^n)$ time and exponential space, and one algorithm that uses $\mathcal{O}^*(1.8878^n)$ time and polynomial space.

Step 3: translate the OCT of H back into a longest cycle in $cl(G)$

Let T be the OCT of H that we obtained in Step 2. We construct a longest cycle in $cl(G)$ by traversing T, picking up the edges (corresponding to vertices in $cl(G)$) one by one and inserting dominated edges as soon as an end vertex of a dominated edge is encountered. For traversing T we use the polynomial-time algorithm that finds a eulerian tour in a connected even graph (cf. [5]). We point out that in case T consists of a single vertex v, a longest cycle in $cl(G)$ is any cycle spanning the clique in $cl(G)$ that corresponds to all edges of H dominated by v.

Step 4: translate the longest cycle in $cl(G)$ into one in G

We can translate the longest cycle in $cl(G)$ obtained in Step 3 into a longest cycle in G in polynomial time by using the method described in [4]. There, two of the present authors show how to translate a 2-factor of $cl(G)$ into a 2-factor of G. It is straightforward to adapt this process and apply it to a single cycle D in $cl(G)$ such that we find, in polynomial time, a cycle C in G with the same length as D.

We mentioned that the first, third and fourth step above can be performed in polynomial time. We also mentioned that we will show in Section 5 that the second step can be executed in $\mathcal{O}^*(1.6818^n)$ time using exponential space, or in $\mathcal{O}^*(1.8878^n)$ time using polynomial space. Hence, we have found the following.

Theorem 2. *The* Longest Cycle *problem, and consequently the* Hamiltonian Cycle *problem, for a claw-free graph on n vertices can be solved in $\mathcal{O}^*(1.6818^n)$ time, using exponential space. It can also be solved in $\mathcal{O}^*(1.8878^n)$ time, using polynomial space.*

5 Two Exact Algorithms for Finding an OCT

In this section we present two exact algorithms for solving the following problem.

Optimum Closed Trail (OCT)
Instance: a connected graph H.
Task: find an optimum closed trail of H.

Both algorithms can be outlined as follows.

Algorithm solving the Optimum Closed Trail **problem**

Input : a connected graph H
Output : an optimum closed trail of H

Test whether or not H is a tree

If H is a tree, output a vertex v of H with degree $\Delta(H)$
If H is not a tree, find an optimum nontrivial closed trail T of H

Test whether or not $\beta(T) \geq \Delta(H)$

If $\beta(T) \geq \Delta(H)$, output T
If $\beta(T) < \Delta(H)$, output a vertex v of H with degree $\Delta(H)$

The difference between the two algorithms is the way in which they compute an ONCT of H in case H is not a tree. To find an ONCT of a connected graph H other than a tree, both algorithms start by branching on vertices of low degree by the same branching procedure, explained in Section 5.1. This way both algorithms obtain a set of subproblems. Each subproblem has the original graph H as input. However, for some subset of edges of H it is already decided whether they will be included in or excluded from the ONCT. Our first algorithm, described in Section 5.2, solves each of the subproblems using dynamic programming. Our second algorithm, described in Section 5.3, solves each of the subproblems by guessing the remaining edges of a possible ONCT.

5.1 Branching on Vertices of Low Degree

Let H be an instance of the OCT problem, and suppose H is not a tree. As can be seen in the outline of the algorithms at the start of Section 5, both algorithms find an ONCT of H. In order to find an ONCT of H, both algorithms start by assigning a so-called *parity label* $\ell(v) \in \{0, 1\}$ to each vertex v of H. Note that if T is an ONCT of H, then $d_T(v)$ is even for every $v \in V(H)$. After all, a vertex is either not in T (i.e., $d_T(v) = 0$), or a vertex has an even number of incident edges in T (since T is a nontrivial closed trail). Hence we initially set $\ell(v) := 0$ for every $v \in V(H)$.

Both algorithms now branch on vertices of degree at most d^*, thus creating a number of subproblems; more specifically, $d^* = 4$ for our first algorithm, and $d^* = 12$ for our second algorithm. The choice of these values of d^* is explained

in the next sections. During the branching process, the size of the graphs under consideration decreases, and we might change the ℓ-labels of certain vertices.

Suppose v is a vertex of degree $d \leq d^*$ in H. If $\ell(v) = 0$ (respectively $\ell(v) = 1$), then the algorithm branches into 2^{d-1} subproblems, each subproblem corresponding to a possible way of choosing an even (respectively odd) number $0 \leq p \leq d$ of edges incident with v that are guessed to be in the ONCT. We call the chosen edges *old trail edges*. For each choice W of old trail edges, we perform the following two operations:

1. set $\ell(w) := \ell(w) + 1 \pmod 2$ for every w with $vw \in W$;
2. delete v and all its d incident edges.

Repeat this procedure as long as the remaining graph contains a vertex of degree at most d^*. Let H' be the resulting graph. Then H' has minimum degree $d^* + 1$ and each vertex $u \in V(H')$ has some label $\ell(u) \in \{0, 1\}$. Let $E(H) = E(H') \cup R(H') \cup W(H')$, where $W(H')$ contains all old trail edges and $R(H')$ contains all other edges we removed from H. In the next stage, edges in $W(H')$ will be assumed to be in the ONCT we are looking for, whereas edges in $R(H')$ will be assumed not to be in the ONCT. If there exists a vertex $v \in V(H) \setminus V(H')$ incident with an odd number of old trail edges, then we discard the subproblem. The reason for this is the fact that we can never obtain a nontrivial closed trail in such a subproblem, since v will have odd degree in that trail and that is not possible. Otherwise, we keep the subproblem and call the tuple $(H', W(H'), \ell)$ a *stage-2 tuple*.

Lemma 3. *The branching phase creates $T(n_1) = \mathcal{O}^*(2^{\frac{d^*-1}{d^*}n_1})$ stage-2 tuples, where n_1 is the total number of edges deleted during this phase.*

Proof. Since for a vertex v of degree d we remove d edges and create 2^{d-1} subgraphs, we find $T(n_1) = 2^{d-1} \cdot T(n_1 - d)$, which yields $T(n_1) = \mathcal{O}^*(2^{\frac{d-1}{d}n_1})$. Since $d \leq d^*$, we end up with $\mathcal{O}^*(2^{\frac{d^*-1}{d^*}n_1})$ stage-2 tuples. □

We point out that the time complexity mentioned in Lemma 3 is $\mathcal{O}^*(1.6818^{n_1})$ if $d^* = 4$ and $\mathcal{O}^*(1.8878^{n_1})$ if $d^* = 12$.

5.2 An $\mathcal{O}^*(1.6818^n)$ Time Algorithm That Uses Exponential Space

In this section, we start by explaining how the first of our two algorithms for the OCT problem finds an ONCT of the input graph H in case H is not a tree. We then prove that our first algorithm for finding an OCT of a connected graph H is correct, and that it runs in $\mathcal{O}^*(1.6818^n)$ time.

Let H be an input of the OCT problem other than a tree. In case H has vertices of degree at most 4, we apply the branching procedure described in Section 5.1. Suppose that during the branching process n_1 edges were deleted (possibly $n_1 = 0$). Then, by Lemma 3, $\mathcal{O}^*(1.6818^{n_1})$ stage-2 tuples $(H', W(H'), \ell)$ have been created. Each of these stage-2 tuples will be processed using the dynamic programming procedure described below.

Let $(H', W(H'), \ell)$ be a stage-2 tuple. If $W(H')$ forms a dominating closed trail of H, i.e., if every edge of H has at least one end vertex in common with an edge in $W(H')$, then we have found an optimum closed trail and the algorithm outputs this trail. If this is not the case, then we enter the dynamic programming phase. In this phase, we consider each $u \in V(H')$. We define two labelings $\ell^* : \{u\} \to \{0,1\}$ with $\ell^*(u) := \ell(u)$ and $\bar{\ell} : \{u\} \to \{0,1\}$ with $\bar{\ell}(u) := \ell(u) + 1 \pmod 2$. We say that $(\{u\}, \ell^*)$ is an *option* if u is incident with at least one old trail edge. Otherwise $(\{u\}, \ell^*)$ is not an option. Furthermore, for every $u \in V(H')$, $(\{u\}, \bar{\ell})$ is not an option.

Suppose we know for all sets $S \subseteq V(H')$ of size at most k and all labelings $\ell' : S \to \{0,1\}$ whether (S, ℓ') is an option or not. Then for each set $S \subseteq V(H')$ of size k, for each vertex $v \in V(H') \setminus S$, and for each $\{0,1\}$-labeling ℓ' of $S \cup \{v\}$, we do as follows. Let p be the number of old trail edges incident with v. We consider every possible way of choosing $0 \leq q \leq 3$ edges incident with v and a vertex in S. The chosen edges will be referred to as *new trail edges*. For each choice N of new trail edges, we set $\ell'(x) := \ell'(x) + 1 \pmod 2$ for every $x \in S$ with $vx \in N$. We then perform the following three tests.

(1) Check if (S, ℓ') is an option.
(2) Check if $p + q$ is even if $\ell'(v) = 0$ and odd if $\ell'(v) = 1$.
(3) If $q = 0$, check if there is a path from v to S in H only using old trail edges.

Only if the answers to tests (1), (2) and (3) are all affirmative, we say that $(S \cup \{v\}, \ell')$ is an option. If so, we also check whether

(4) for each old trail edge e there is a path, consisting of only old trail edges, connecting e to a vertex in $S \cup \{v\}$;
(5) each vertex x in $S \cup \{v\}$ has label $\ell'(x) = 0$ and each vertex $y \in V(H') \setminus (S \cup \{v\})$ incident with an old trail edge has label $\ell(y) = 0$;

If the answers to tests (4) and (5) are both affirmative, the algorithm has detected a nontrivial closed trail T of H (as we prove in Theorem 3 below). We may assume that the algorithm also finds T, since that only requires some extra "bookkeeping" during the dynamic programming phase; we omitted the details for clarity of presentation. The algorithm then checks how many edges of H are dominated by T by computing $\beta(T)$. If $\beta(T) = |E(H)|$, then T is a dominating closed trail of H. Since every dominating closed trail is an optimum closed trail, the algorithm outputs T. Otherwise the algorithm stores T, unless it has already found a nontrivial closed trail T' with $\beta(T') \geq \beta(T)$ before, in which case T is discarded. If $k < |V(H')|$, the algorithm repeats the above procedure for all sets $S \subseteq V(H')$ of size $k+1$, all vertices $v \in V(H') \setminus S$ and all $\{0,1\}$-labelings ℓ' of $S \cup \{v\}$. If $k = |V(H')|$, then the algorithm terminates.

We now show that our first algorithm for finding an OCT is correct.

Theorem 3 (Correctness). *When run on a connected graph H, the algorithm returns an optimum closed trail of H.*

Proof. As shown in the outline at the beginning of Section 5, the algorithm starts by checking if the input graph H is a tree. If H is a tree, then every closed trail of H consists of a single vertex. In particular, an optimum closed trail of H consists of a single vertex v of degree $\Delta(H)$. Hence the algorithm correctly outputs v in this case. If H is not a tree, then H contains a nontrivial closed trail; in particular, H contains an ONCT. We show below that the algorithm in fact finds such an ONCT T of H by executing the branching and dynamic programming procedures described in Section 5.1 and Section 5.2, respectively. Since an OCT of H might consist of a single vertex even if H is not a tree, the ONCT T is not necessarily an OCT of H. Hence the algorithm checks if a vertex v of maximum degree in H dominates more edges of H than T does. If so, then v is an OCT of H, and the algorithm correctly outputs v. Otherwise T is both an ONCT and an OCT of H, so the algorithm correctly outputs T.

It remains to show that, in case H is not a tree, the algorithm finds an ONCT T of H by executing the branching and dynamic programming procedures described in Section 5.1 and Section 5.2, respectively. Note that H has an optimum nontrivial closed trail by our assumption that H is not a tree.

We first show that if the algorithm computes $\beta(T)$ for a subgraph T of H, then T is a nontrivial closed trail of H. Only if the algorithm has found a stage-2 tuple $(H', W(H'), \ell)$ with some option (S, ℓ) for which the answers to tests (4) and (5) are both affirmative, it computes $\beta(T)$ for a subgraph T of H consisting of all old trail edges in $W(H')$ plus all new trail edges that have been added between the vertices of S. The dynamic programming, together with tests (3) and (4), ensures that T is connected. Tests (1), (2) and (5) together with the definition of a stage-2 tuple ensure that T is even. Hence, every subgraph T of H for which the value of $\beta(T)$ is computed is a nontrivial closed trail of H. Note that the algorithm does not compute $\beta(T)$ for *each* nontrivial closed trail T of H, but only for those that can be "built up" satisfying certain connectivity conditions throughout the dynamic programming phase.

It remains to show that the algorithm always finds an *optimum* nontrivial closed trail T of H. Due to Corollary 2 we may assume that T is 3-ordered. We show that our algorithm stores T, unless it has already stored another optimum nontrivial closed trail of H before it finds T. Let V' consist of all vertices that are not removed in the branching procedure, so $V' := V(H')$ for the graph H' in every stage-2 tuple. Let T' be the subgraph of T with $V(T') = V(T) \cap V'$. Then there exists a stage-2 tuple $(H', W(H'), \ell)$ such that $W(H')$ is exactly the set of edges of T that are incident with at least one vertex in $V(T) \setminus V'$, and such that $\ell(v) = 0$ if $v \in V' \setminus V(T')$, and $\ell(v) = 0$ (respectively $\ell(v) = 1$) if $v \in V(T')$ and v is incident with an even (respectively odd) number of edges in $W(H')$. Since our algorithm considers all possible stage-2 tuples, it will detect tuple $(H', W(H'), \ell)$. As T is 3-ordered, each component of T' is 3-ordered. This means that our dynamic programming procedure, based on the number of ways a vertex can be made adjacent to a set S with at most three edges, will find a labeling ℓ' such that (T_i, ℓ') is an option for each component T_i of T. As these components are connected to each other via old trail edges, at some moment

(T', ℓ) will be formed. Then tests (1)-(5) will all be successful and the algorithm will compute $\beta(T)$ for the subgraph T. Since T is an optimum nontrivial closed trail of H, there is no other nontrivial closed trail of H that dominates more edges of H than T does. Hence the algorithm will store T, unless it has encountered a nontrivial closed trail T' of H with $\beta(T') = \beta(T)$ before it found T, in which case the algorithm has stored T'. □

Below we give the overall time complexity of our first algorithm for solving the OCT problem.

Theorem 4 (Running time). *The algorithm runs in $\mathcal{O}^*(1.6818^n)$ time on a connected graph with n edges.*

Proof. From the outline of the algorithm at the beginning of Section 5 it is clear that all steps not involving finding an ONCT can be performed in polynomial time. Hence it suffices to prove that the algorithm finds an ONCT of a connected graph H other than a tree in $\mathcal{O}^*(1.6818^n)$ time, where $n = |E(H)|$.

We first prove that the dynamic programming procedure presented at the beginning of Section 5.2 runs in $\mathcal{O}^*(3^p)$ time on any p-vertex graph. Let H' be a graph on p vertices. There are $\binom{p}{k}$ sets $S \subseteq V(H')$ of cardinality k, each of those sets has 2^k possible labelings ℓ', and there are $\binom{k}{0} + \binom{k}{1} + \binom{k}{2} + \binom{k}{3} = \mathcal{O}(k^3)$ ways to attach a new vertex v to a subset of cardinality k by using at most 3 edges. Each of the tests (1)-(5) can be done in polynomial time, and the same holds for computing $\beta(T)$ for a nontrivial closed trail T of H. Hence the time complexity of this procedure is

$$\mathcal{O}^*\Big(\sum_{k=1}^{p} \binom{p}{k} \cdot 2^k \cdot \mathcal{O}(k^3)\Big) = \mathcal{O}^*(3^p).$$

Let H be an instance of the OCT problem having n edges, and suppose H is not a tree. The algorithm repeatedly branches on vertices of degree at most $d^* = 4$. Let n_1 be the number of edges deleted during this branching phase. Then we obtain $\mathcal{O}^*(1.6818^{n_1})$ stage-2 tuples by Lemma 3. Let $(H', W(H'), \ell)$ be such a stage-2 tuple, where H' is a graph of minimum degree 5 having $n_2 := n - n_1$ edges and, say, p vertices. As shown above, the dynamic programming procedure uses $\mathcal{O}^*(3^p)$ time. Since the minimum degree in H' is 5, we obtain $n_2 \geq 5p/2$, or equivalently $p \leq 2n_2/5$. Hence we can process each stage-2 tuple in time $\mathcal{O}^*(3^{\frac{2n_2}{5}}) = \mathcal{O}^*(1.5519^{n_2})$. This means that the overall time complexity of our algorithm on a graph H having $n = n_1 + n_2$ edges is

$$\mathcal{O}^*(1.6818^{n_1} \cdot 1.5519^{n_2}) = \mathcal{O}^*(1.6818^n).$$

If we choose $d^* \neq 4$, then the above time complexity is no longer guaranteed. □

Theorem 3 and Theorem 4 immediately imply the following.

Corollary 3. *The OCT problem for a connected graph H with n edges can be solved in $\mathcal{O}^*(1.6818^n)$ time, using exponential space.*

5.3 An $\mathcal{O}^*(1.8878^n)$ Time Algorithm That Uses Polynomial Space

We describe our second algorithm for solving the OCT problem in the proof of the following theorem.

Theorem 5. *The OCT problem for a connected graph H with n edges can be solved in $\mathcal{O}^*(1.8878^n)$ time, using polynomial space.*

Proof. The second algorithm strongly resembles the first algorithm, described in Section 5.2. The only difference is the way in which the algorithm finds an ONCT of H in case H is not a tree. In order to prove correctness of our second algorithm for the OCT problem, it therefore suffices to prove correctness of the procedure of finding an ONCT of H described below.

Let H be a connected graph other than a tree. The algorithm executes the branching procedure described in Section 5.1, but this time we perform branching on vertices of degree at most $d^* = 12$. Suppose we delete n_1 edges during the branching process. By Lemma 3, this yields $\mathcal{O}^*(2^{11n_1/12}) = \mathcal{O}^*(1.8878^{n_1})$ stage-2 tuples $(H', W(H'), \ell)$, where each graph H' has p vertices of degree at least 13 and $n_2 = n - n_1$ edges. Note that $n_2 \geq 13p/2$, or equivalently $p \leq 2n_2/13$.

Since we assumed H not to be a tree, H has an ONCT T. By Theorem 2 we may assume that T is 2-degenerate. Let T' denote the (2-degenerate) subgraph of T that remains after the branching procedure. Note that T' is a subgraph of the graph H' of some stage-2 tuple $(H', W(H'), \ell)$. It is well-known that any 2-degenerate graph on p vertices has at most $2p$ (or, more precisely, at most $2p-3$) edges. For every stage-2 tuple $(H', W(H'), \ell)$, we check for every possible subset $S \subseteq E(H')$ of edges up to cardinality $2p$ whether S together with the old trail edges in $W(H')$ forms a nontrivial closed trail T of H. If so, then we compute the number $\beta(T)$ of edges of H dominated by T, which can be done in polynomial time. If $\beta(T) = |E(H)|$, then T is a dominating closed trail of H. Since every dominating closed trail is an optimum closed trail, the algorithm outputs T. Otherwise the algorithm stores T, unless it has already found a nontrivial closed trail T' with $\beta(T') \geq \beta(T)$ before, in which case T is discarded. Since we check all subsets $S \subseteq E(H')$ for every stage-2 tuple $(H', W(H'), \ell)$, we are guaranteed to find an optimum nontrivial closed trail of H. This proves that our second algorithm for the OCT problem is correct.

From the outline of the algorithm at the beginning of Section 5 it is clear that all steps not involving finding an ONCT can be performed in polynomial time. Since the above procedure for finding an ONCT evidently only uses polynomial space, it remains to determine the time complexity of this procedure. Using Stirling's approximation $n_2! \approx n_2^{n_2} e^{-n_2} \sqrt{2\pi n_2}$ and the fact that $p \leq 2n_2/13$, the total number of checks per stage-2 tuple can be estimated as follows:

$$\sum_{k=1}^{2p} \binom{n_2}{k} \leq 2p \binom{n_2}{2p} \leq 2p \binom{n_2}{\frac{4n_2}{13}} = \mathcal{O}^*\Big(\Big(\frac{1}{\alpha^\alpha (1-\alpha)^{1-\alpha}}\Big)^{n_2}\Big)\,,$$

where $\alpha = 4/13$, which constitutes $\mathcal{O}^*(1.8539^{n_2})$ checks. Since each of those checks can be performed in polynomial time and the number of stage-2 tuples

we have to process is $\mathcal{O}^*(1.8878^{n_1})$, the overall time complexity of our second algorithm is

$$\mathcal{O}^*(1.8878^{n_1} \cdot 1.8539^{n_2}) = \mathcal{O}^*(1.8878^n) \ .$$

If we choose $d^* \neq 12$, then this time complexity is no longer guaranteed. □

6 Conclusions

We presented the first exact algorithms breaking the barrier 2^n for the Longest Cycle problem on claw-free graphs. Our first algorithm uses $\mathcal{O}^*(1.6818^n)$ time and exponential space, and our second algorithm uses $\mathcal{O}^*(1.8878^n)$ time and polynomial space. A natural question is whether similar approaches can be used for other generalizations of the Hamiltonian Cycle problem. Since a hamiltonian cycle is a connected 2-factor, the related NP-hard problem of determining a 2-factor with the smallest number of components in a claw-free graph G seems an obvious candidate. It was shown in [4] that this problem is equivalent to finding a smallest set of edge-disjoint stars with at least three edges and nontrivial closed trails that together dominate all edges of the preimage graph H of the closure of G. However the approach in Section 5 for finding an OCT of H does not generalize in a straightforward way to finding such a smallest set of edge-disjoint stars with at least three edges and nontrivial closed trails. In fact, we do not believe a similar approach is possible because the counterpart of Section 5 would involve solving the following problem, which turns out to be NP-hard.

Decomposition in $\geq$ 3-Stars and Closed Trails (DEC)
Instance: a connected graph H.
Task: find a decomposition (partition) of $E(H)$ into stars with at least three edges and nontrivial closed trails.

It is not difficult to prove that the DEC problem is NP-hard by a reduction from the following decision problem, which is known to be NP-complete [19].

Decomposition in $\geq$ 3-Stars (DECOMP)
Instance: a connected graph H.
Question: can $E(H)$ be decomposed into stars with at least three edges?

Let G be an instance of the DECOMP problem. Replace each edge uv of G by the gadget illustrated in Figure 2, i.e., replace uv by a graph with vertex set $\{u, a, b, c, d, e, f, v\}$ and edge set $\{ua, ab, bc, cd, de, ef, fv, ae, bf\}$. Then, considering the edge cd, one readily checks that $E(G)$ has a decomposition into stars with at least three edges if and only if the newly constructed graph has a decomposition of its edge set into stars with at least three edges and nontrivial closed trails. This shows that the decision version of the DEC problem is NP-complete, and hence that the DEC problem is NP-hard. Note that the construction shows that the DEC problem remains NP-hard even when restricted to 2-degenerate graphs.

Since a hamiltonian cycle is a connected 2-regular spanning subgraph, another direction would be to consider the problem of finding a connected spanning

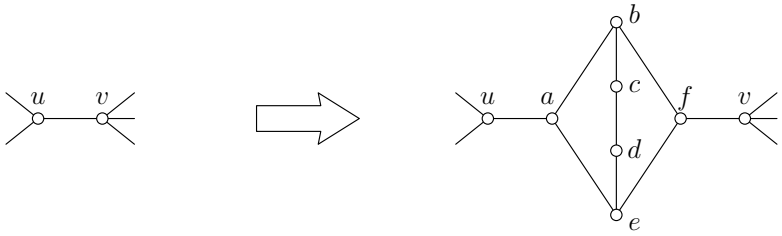

Fig. 2. The gadget for replacing the edges of G.

3-regular subgraph of a claw-free graph. We have no idea how to generalize our approach in order to solve this problem. We heavily relied on the closure technique and the relationship between longest cycles in a claw-free graph G and optimum closed trails in the preimage graph H of the closure of G. We think it is highly unlikely that there is a natural counterpart of our approach for finding connected 3-regular spanning subgraphs.

Another interesting open problem is whether we can solve the Traveling Salesman problem for claw-free graphs in $\mathcal{O}^*(\alpha^n)$ time for some constant $\alpha < 2$. This also requires some new ideas, as our current approach involving the relationship between longest cycles in a claw-free graph G and optimum closed trails in the preimage graph H of the closure of G does not suffice.

Can we find an $\mathcal{O}^*(\alpha^n)$ time algorithm that solves the Hamiltonian Cycle problem for some constant $\alpha < 2$ for the class of bipartite graphs, or equivalently (cf. [20]) for the class of split graphs, or a superclass of split graphs such as the class of P_5-free graphs? As the Hamiltonian Cycle problem is already NP-complete for chordal bipartite graphs [20], this question is interesting for that class as well. One might also try to design fast exact algorithms for the Hamiltonian Cycle problem restricted to superclasses of claw-free graphs such as $K_{1,4}$-free graphs.

References

1. E.T. Bax. Inclusion and exclusion algorithm for the Hamiltonian path problem. *Information Processing Letters 47*, 203–207, 1993.
2. A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. The travelling salesman problem in bounded degree graphs. In: *35th International Colloquium on Automata, Languages and Programming (ICALP 2008)*. Lecture Notes in Computer Science 5125, 198–209, 2008.
3. H.J. Broersma, F.V. Fomin, P. van 't Hof, and D. Paulusma. Fast exact algorithms for hamiltonicity in claw-free graphs. In: *35th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2009)*. Lecture Notes in Computer Science 5911, 44–53, 2009.
4. H.J. Broersma and D. Paulusma. Computing sharp 2-factors in claw-free graphs. In: *33th International Symposium on Mathematical Foundations of Computer Science (MFCS 2008)*. Lecture Notes in Computer Science 5162, 193–204, 2008.

5. R. Diestel. *Graph Theory, Second edition.* Graduate Texts in Mathematics 173, Springer, 2000.
6. F. Dorn, F. V. Fomin, and D. M. Thilikos, Catalan structures and dynamic programming on H-minor-free graphs. In: *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2008)*, ACM-SIAM, pp. 631-640.
7. F. Dorn, E. Penninkx, H. Bodlaender, and F. V. Fomin, Efficient exact algorithms on planar graphs: Exploiting sphere cut branch decompositions, *Proceedings of the 13th Annual European Symposium on Algorithms (ESA 2005)*, vol. 3669 of LNCS, Springer, 2005, pp. 95–106.
8. D. Eppstein. The traveling salesman problem for cubic graphs. *Journal of Graph Algorithms and Applications 11*, 61–81, 2007.
9. R. Faudree, E. Flandrin, and Z. Ryjáček. Claw-free graphs—a survey. *Discrete Mathematics 164*, 87–147, 1997.
10. M.R. Garey and D.S. Johnson. *Computers and Intractability.* W.H. Freeman and Co., New York, 1979.
11. H. Gebauer. On the number of hamilton cycles in bounded degree graphs. In: *4th Workshop on Analytic and Combinatorics (ANALCO 2008)*. SIAM, Philadelphia, 2008.
12. F. Harary, *Graph Theory.* Addison-Wesley, Reading MA, 1969.
13. F. Harary and C. St. J.A. Nash-Williams. On eulerian and hamiltonian graphs and line graphs. *Canadian Mathematical Bulletin 8*, 701–709, 1965.
14. M. Held and R.M. Karp. A dynamic programming approach to sequencing problems. *Journal of SIAM 10*, 196–210, 1962.
15. K. Iwama and T. Nakashima. An improved exact algorithm for cubic graph TSP. In: *13th Annual International Computing and Combinatorics Conference (COCOON 2007)*. Lecture Notes in Computer Science 4598, 108–117, 2007.
16. R.M. Karp. Dynamic programming meets the principle of inclusion and exclusion. *Operations Research Letters 1*, 49–51, 1982.
17. S. Kohn, A. Gottlieb, and M. Kohn, *A generating function approach to the traveling salesman problem.* In: *Proceedings of the ACM annual conference (ACM 1977)*, ACM Press, 1977, pp. 294–300.
18. M. Li, D.G. Corneil, and E. Mendelsohn. Pancyclicity and NP-completeness in planar graphs. *Discrete Applied Mathematics 98*, 219–225, 2000.
19. Z. Lonc. On the complexity of some edge-partition problems for graphs. *Discrete Applied Mathematics 70*, 177–183, 1996.
20. H. Müller. Hamiltonian circuits in chordal bipartite graphs. *Discrete Mathematics 156*, 291–298, 1996.
21. N.D. Roussopoulos. A $\max\{m, n\}$ algorithm for determining the graph H from its line graph G. *Information Processing Letters 2*, 108–112, 1973.
22. Z. Ryjáček. On a closure concept in claw-free graphs. *Journal of Combinatorial Theory, Series B 70*, 217–224, 1997.
23. C. Thomassen and B. Toft. Non-separating induced cycles in graphs. *Journal of Combinatorial Theory, Series B 31*, 199–224, 1981.
24. G.J. Woeginger. Open problems around exact algorithms. *Discrete Applied Mathematics 156*, 397–405, 2008.