# Fast Exact Algorithms for Hamiltonicity in Claw-Free Graphs

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

[1] Department of Computer Science, University of Durham,
Science Laboratories, South Road, Durham DH1 3LE, England*
{hajo.broersma,pim.vanthof,daniel.paulusma}@durham.ac.uk
[2] Institutt for informatikk, Universitet i Bergen,
Postboks 7803, 5020 Bergen, Norway
fomin@ii.uib.no

**Abstract.** The Hamiltonian Cycle problem asks if an $n$-vertex graph $G$ has a cycle passing through all vertices of $G$. This problem is a classic NP-complete problem. So far, finding an exact algorithm that solves it in $\mathcal{O}^*(\alpha^n)$ time for some constant $\alpha < 2$ is a notorious open problem. For a claw-free graph $G$, finding a hamiltonian cycle is equivalent to finding a closed trail (eulerian subgraph) that dominates the edges of some associated graph $H$. Using this translation we obtain two exact algorithms that solve the Hamiltonian Cycle problem for the class of 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

In this paper we study the well-known NP-complete decision problem Hamiltonian Cycle (cf. [7]) that asks whether a graph $G$ has a *hamiltonian cycle*, i.e., a cycle that passes 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 & Karp [11] 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 indicates that we suppress factors of polynomial order, and we use this notation throughout the paper. The slightly easier Hamiltonian Cycle problem can be solved using $\mathcal{O}^*(2^n)$ time and polynomial space, as was shown by Karp [13] and independently by Bax [1]. It is a major and long outstanding open problem if the Hamiltonian Cycle and the Traveling Salesman problem can be solved in $\mathcal{O}^*(\alpha^n)$ time for some constant $\alpha < 2$, even if the polynomial space restriction is dropped.

For some graph classes for which the Hamiltonian Cycle, and consequently the Traveling Salesman problem, remains NP-complete, faster algorithms

* This work has been supported by EPSRC (EP/D053633/1).

C. Paul and M. Habib (Eds.): WG 2009, LNCS 5911, pp. 44–53, 2010.

have been designed. For planar graphs, the HAMILTONIAN CYCLE problem can be solved in $\mathcal{O}^*(c^{\sqrt{n}})$ for some constant $c$ (cf. [19]). The TRAVELING SALESMAN problem can be solved in $\mathcal{O}^*(1.251^n)$ time for cubic graphs [12] and in $\mathcal{O}^*(1.890^n)$ time for graphs with maximum degree 4 [5]. Both algorithms use polynomial space. For graphs with maximum degree 4, an algorithm with time complexity $\mathcal{O}^*(1.733^n)$ is known [8], 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 an $\mathcal{O}^*((2-\epsilon)^n)$ 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 [6] for a survey. The HAMILTONIAN CYCLE problem is NP-complete for claw-free graphs; the authors of [14] show that the problem is already NP-complete for 3-connected cubic planar claw-free graphs. We present two exact algorithms that solve the HAMILTONIAN CYCLE problem for 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. Our techniques are based on a (known) transformation of the problem to the problem of finding a dominating closed trail in a graph and a new, more careful study of such trails. Hence, these techniques are different from the ones used in the already known algorithms, and as such may be of independent interest.

**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 [4]. Let $G = (V(G), E(G))$ be a graph. The *neighborhood* of a vertex $v$ in $G$ is denoted by $N_G(v) := \{w \in V(G) \mid vw \in E\}$, and $d_G(v) = |N_G(v)|$ denotes the *degree* of $v$. 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 $U \subseteq V$ is denoted by $G[U]$.

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 $N_G(x)$ is connected but not complete, we can perform an operation called *local completion of $G$ at $x$* by adding edges joining all pairs of nonadjacent vertices in $N_G(x)$.

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. [9]). 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* (or *eulerian*) if it is a connected even graph. Let $T$ be a closed trail in a graph $H$. If $V(H) \setminus V(T)$ is an independent set in $H$, then we say that $T$ is a *dominating closed trail*, abbreviated DCT. Note that the latter means that every edge of $H$ has at least one vertex in $T$, so in this context "dominating" means "edge-dominating". For any integer $k \geq 1$, a graph $H$ is called *$k$-degenerate* if every non-empty subgraph of $H$ has a vertex of degree at most $k$. We say that $H$ is *$k$-ordered* if $H$ allows a vertex ordering $\pi = v_1, \ldots, v_{|V(H)|}$ such that for $1 \leq i \leq |V(H)|$, $H[\{v_1, \ldots, v_i\}]$ is connected and $v_i$ has at most $k$ neighbors in $H[\{v_1 \ldots, v_i\}]$.

**Paper organization.** In Section 2 we translate the Hamiltonian Cycle problem for claw-free graphs into the problem of finding a dominating closed trail in triangle-free graphs. In Section 3 we show that every graph with a spanning closed trail has a 2-degenerate 3-ordered spanning closed trail. We use this structural result in Section 4, where we present two exact algorithms for finding a dominating closed trail in a graph. Section 5 contains the conclusions and mentions some open problems.

## 2 The Two Exact Algorithms

Here we explain our two algorithms that solve the Hamiltonian Cycle problem for a claw-free graph $G$ on $n$ vertices. For the first step 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 [17] 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 [17] also showed that $G$ 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. [16]). Hence, we can efficiently compute the unique graph $H$ with $L(H) = cl(G)$.

**Step 2: find a DCT of $H$**

Harary and Nash-Williams [10] showed that the line graph of any connected graph with at least three vertices is hamiltonian if and only if the graph itself contains a DCT. This result combined with the results from the previous step implies that $G$ has a hamiltonian cycle if and only if $H$ has a DCT. In Section 4 we present two exact algorithms for finding such a DCT in a 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 DCT of $H$ back into a hamiltonian cycle of $cl(G)$**

Suppose we have obtained a DCT $T$ in Step 2. Then we construct a hamiltonian cycle of $cl(G)$ 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 an even connected graph (cf. [4]).

**Step 4: translate the hamiltonian cycle in $cl(G)$ to one in $G$**

We can do this in polynomial time by using exactly the same method as described in [3]. There, we show how to translate a 2-factor of $cl(G)$ into a 2-factor of $G$. Since a hamiltonian cycle is a connected 2-factor we are done.

From the above it is clear that all steps except the third one can be performed in polynomial time. Hence, we have found the following.

**Theorem 1.** *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.*

## 3 Closed Trails of Low Degeneracy and Ordering

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 useful result is due to Thomassen and Toft [18].

**Theorem 2 ([18]).** *Any connected graph with minimum degree 3 has an induced non-separating cycle.*

Theorem 2 immediately yields the following result.

**Corollary 1.** *Any connected graph with minimum degree 3 has a removable cycle.*

*Proof.* Let $H$ be a connected graph with minimum degree 3. By Theorem 2, $H$ 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 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 theorem, which will help us to obtain the time complexity of the exact algorithms described in Section 4.

**Theorem 3.** *Every graph with a spanning closed trail contains a 2-degenerate 3-ordered spanning closed trail.*

*Proof.* We first show that every graph with a spanning closed trail contains a 2-degenerate spanning closed trail. Let $H^*$ be a counterexample with $|E(H^*)|$ minimum. Let $T$ be a spanning closed trail in $H^*$. We repeatedly remove vertices from $T$ with degree at most 2 in $T$ as long as possible. Let $T'$ be the subgraph of $T$ we obtain this way. Since $H^*$ is a counterexample, $T'$ is not empty. Let $T_1$ be a component of $T'$. Since $T'$ has minimum degree at least 3, $T_1$ has a removable cycle $C$ by Corollary 1. Then $C$ is also a removable cycle of $H^*$, since $H^*$ is a supergraph of $T_1$. This contradicts the minimality of $|E(H^*)|$.

So, every graph $H$ with a spanning closed trail contains a 2-degenerate spanning closed trail $T$. Suppose $T$ is not 3-ordered. We repeatedly remove vertices from $T$ with degree at most 3 in $T$ until $T$ becomes disconnected. Let $T'$ be the resulting (connected) subgraph of $T$. Since $T$ is not 3-ordered, $T'$ is not empty. Let $U$ consist of all vertices of degree at most 3 in $T'$. By our procedure, every vertex of $U$ is a cut-vertex of $T'$, and since $T$ is 2-degenerate, $U$ is nonempty. Let $u \in U$ be such that $T'[V(T') \setminus \{u\}]$ contains a component $D$ without vertices of $U$. Then all vertices of $D$ have degree at least 3, contradicting the 2-degeneracy of $T$. □

## 4 Two Exact Algorithms for Finding a DCT

We present two exact algorithms for solving the following problem.

Dominating Closed Trail (DCT)
*Instance:* a connected graph $H$.
*Question*: does $H$ have a dominating closed trail?

To solve the DCT problem for an instance $H$, both algorithms start by branching on vertices of low degree by the same branching procedure, explained in Section 4.1. 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 dominating closed trail. Our first algorithm, described in Section 4.2, solves each of the subproblems using dynamic programming. Our second algorithm, described in Section 4.3, solves each of the subproblems by guessing the remaining edges of a possible dominating closed trail.

### 4.1 Branching on Vertices of Low Degree

Let $H = (V, E)$ be an instance of the DCT problem. We assign a so-called *parity label* $\ell(v) \in \{0, 1\}$ to each vertex $v$ of $H$. Note that if $H$ has a dominating closed trail $T$, then $d_T(v)$ is even for every $v \in V$. After all, a vertex is either not in $T$ (i.e., $d_T(v) = 0$, in which case all of its neighbors must be in $T$), or a vertex has an even number of incident edges in $T$ (since $T$ is a closed trail). Hence we initially set $\ell(v) = 0$ for every $v \in V$.

The first stage of both algorithms consists of branching 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 $\ell$-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 dominating closed trail. 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 dominating closed trail we are looking for, whereas edges in $R(H')$ will be assumed not to be in the dominating closed trail. Suppose $R(H')$ contains an edge $e = xy$ with $x, y \in V(H) \setminus V(H')$ such that both $x$ and $y$ are not incident with any old trail edge. Then $e$ will not be dominated by any closed trail that we might discover in the next stage. Hence, we discard this subproblem. For the same reason, we also discard the subproblem if there is a vertex $v \in V(H) \setminus V(H')$ incident with an odd number of old trail edges. If these two cases do not occur, we keep the subproblem and call the tuple $(H', W(H'), \ell)$ a *stage-2 tuple*.

**Lemma 1.** *The branching phase of the algorithm 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 1 is $\mathcal{O}^*(1.6818^{n_1})$ if $d^* = 4$ and $\mathcal{O}^*(1.8878^{n_1})$ if $d^* = 12$.

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

Let $H = (V, E)$ be an input of the DCT problem. In case $H$ has vertices of degree at most 4, we apply the branching procedure described in Section 4.1. Suppose that during the branching process $n_1$ edges were deleted (possibly $n_1 = 0$). Then, by Lemma 1, $\mathcal{O}^*(1.6818^{n_1})$ 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. If at least one of them leads to a dominating closed trail of $H$, then the algorithm outputs YES; the algorithm outputs NO otherwise.

Let $(H', W(H'), \ell)$ be a stage-2 tuple. We write $H' = (V', E')$. We output YES if $W(H')$ forms a dominating closed trail of $H$. If this is not the case, we enter the dynamic programming phase. In this procedure, we consider each $u \in V'$ and say that $(\{u\}, \ell(u))$ is an *option* if $u \in V'$ is incident with at least one old trail edge. Otherwise $(\{u\}, \ell(u))$ is not an option. Furthermore, $(\{u\}, \bar{\ell}(u))$ with $\bar{\ell}(u) = \ell(u) + 1 \pmod 2$ is not an option.

Suppose we know for all sets $S \subseteq V'$ of size at most $k$ and all labelings $\ell' : S \to \{0, 1\}$ whether $(S, \ell')$ is an option or not. Then for each set $S \subseteq V'$ of size $k$, for each vertex $v \in V' \setminus S$, and for each $\{0, 1\}$-labeling $\ell'$ 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 perform the following three tests.

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

(4) each old trail edge allows a path to a vertex in $S \cup \{v\}$ that uses only old trail edges;
(5) each vertex $x$ in $S \cup \{v\}$ has label $\ell'(x) = 0$ and each vertex $y \in V' \setminus (S \cup \{v\})$ incident with an old trail edge has label $\ell(y) = 0$;
(6) there is no edge $e = ab$ in $H'$ for some $a, b \in V' \setminus (S \cup \{v\})$ such that both $a$ and $b$ are not incident with an old trail edge.

If the answers to tests (4), (5), (6) are all three affirmative, the algorithm concludes that $H$ has a dominating closed trail (cf. Theorem 4) and returns YES. If no YES-answer has been returned and $k < |V'|$, the algorithm considers all sets $S \subseteq V'$ of size $k + 1$, all vertices $v \in V' \setminus S$ and all $\{0, 1\}$-labelings $\ell'$ of $S \cup \{v\}$. Otherwise, the algorithm outputs NO.

**Theorem 4 (Correctness).** *When run on a connected graph $H$, the algorithm returns* YES *if $H$ has a dominating closed trail, and returns* NO *otherwise.*

*Proof.* Our algorithm only returns a YES-answer if it has found a stage-2 tuple $(H', W(H'), \ell)$ with some option $(S, \ell)$ for which tests (4), (5), (6) are all positive. In that case, let $T$ be the subgraph of $H$ consisting of all old trail edges in $W(H')$ plus all new trail edges that have been added between 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, and (6) ensures that $T$ is dominating. Hence, $T$ is a dominating closed trail.

It remains to show that if $H$ has a dominating closed trail, then the algorithm outputs YES. Suppose $H$ has a dominating closed trail $T$. Due to Theorem 3 we may assume that $T$ is 3-ordered. We show that our algorithm finds $T$, unless it finds another dominating closed trail of $H$ first. Let $V'$ consist of all vertices that are not removed in the branching procedure, so $V(H') = V'$ 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 $\ell'$ such that $(T_i, \ell')$ 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 $(T', \ell)$ will be formed. Then tests (1)-(6) will all be successful and a YES-answer is returned. □

Below we give the overall running time of our algorithm.

**Theorem 5 (Running time).** *The algorithm runs in $\mathcal{O}^*(1.6818^n)$ time.*

*Proof.* We first prove that the dynamic programming procedure runs in $\mathcal{O}^*(3^p)$ time on any $p$-vertex graph. Let $H' = (V', E')$ be a graph on $p$ vertices. There are $\binom{p}{k}$ sets $S \subseteq V'$ of cardinality $k$, each of those sets has $2^k$ possible labelings $\ell$, 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)-(6) can be done in polynomial time. 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 DCT problem having $n$ edges. Suppose we repeatedly branch on vertices of degree at most $d^* = 4$, and suppose $n_1$ is the number of edges we delete during this branching phase. Then we obtain $\mathcal{O}^*(1.6818^{n_1})$ stage-2 tuples by Lemma 1. Let $(H', W(H'), \ell)$ be such a stage-2 tuple, where $H' = (V', E')$ 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 running time 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 upper bound is no longer guaranteed. □

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

We describe our second algorithm in the proof of the following theorem.

**Theorem 6.** *The DCT problem for a graph $H$ on $n$ edges can be solved in $\mathcal{O}^*(1.8878^n)$ time, using polynomial space.*

*Proof.* Let $H$ be an instance of the DCT problem with $n$ edges. We execute the branching procedure described in Section 4.1, 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 1, 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 minimum degree 13 and $n_2 = n - n_1$ edges. Note that $n_2 \geq 13p/2$, or equivalently $p \leq 2n_2/13$.

If $H$ has a dominating closed trail $T$, then $T$ may be assumed to be 2-degenerate, due to Theorem 3. Let $T'$ denote the (2-degenerate) subgraph of $T$ that remains after the branching procedure; note that $T'$ is a subgraph of some graph $H'$. A 2-degenerate graph on $p$ vertices has at most $2p$ edges. This means that we only have to check in every $H'$ for every possible subset of edges up to cardinality $2p$ whether this subset together with the old trail edges in $W(H')$ forms a dominating closed trail of $H$. Using Sterling's approximation $n_2! \approx n_2^{n_2} e^{-n_2} \sqrt{2\pi n_2}$ and the fact $p \leq 2n_2/13$, the total number of checks 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 leads to $\mathcal{O}^*(1.8539^{n_2})$ checks. Since each of them can be performed in polynomial time, the overall running time is

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

If we choose $d^* \neq 12$, then the above upper bound is no longer guaranteed. It is clear that this algorithm only needs polynomial space. □

## 5 Conclusions

We presented two exact algorithms for the Hamiltonian Cycle problem. Can we speed up these algorithms by making use of the triangle-freeness of the preimage graph? Another (more) 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 requires some new ideas as our current approach that takes the closure of a graph and then makes a transformation to the domain of triangle-free graphs 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. [15]) 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 [15], this question is interesting for that class as well. We can also try to design fast exact algorithms for superclasses of claw-free graphs such as $K_{1,4}$-free graphs.

## References


1. Bax, E.T.: Inclusion and exclusion algorithm for the Hamiltonian path problem. Information Processing Letters 47, 203–207 (1993)
2. Björklund, A., Husfeldt, T., Kaski, P., Koivisto, M.: The travelling salesman problem in bounded degree graphs. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 198–209. Springer, Heidelberg (2008)
3. Broersma, H.J., Paulusma, D.: Computing sharp 2-factors in claw-free graphs. In: Ochmański, E., Tyszkiewicz, J. (eds.) MFCS 2008. LNCS, vol. 5162, pp. 193–204. Springer, Heidelberg (2008)
4. Diestel, R.: Graph Theory, 2nd edn. Graduate Texts in Mathematics, vol. 173. Springer, Heidelberg (2000)
5. Eppstein, D.: The traveling salesman problem for cubic graphs. Journal of Graph Algorithms and Applications 11, 61–81 (2007)
6. Faudree, R., Flandrin, E., Ryjáček, Z.: Claw-free graphs—a survey. Discrete Mathematics 164, 87–147 (1997)
7. Garey, M.R., Johnson, D.S.: Computers and Intractability. W.H. Freeman and Co., New York (1979)
8. Gebauer, H.: On the number of hamilton cycles in bounded degree graphs. In: 4th Workshop on Analytic and Combinatorics (ANALCO 2008), SIAM, Philadelphia (2008)
9. Harary, F.: Graph Theory. Addison-Wesley, Reading (1969)
10. Harary, F., Nash-Williams, C.S.J.A.: On eulerian and hamiltonian graphs and line graphs. Canadian Mathematical Bulletin 8, 701–709 (1965)
11. Held, M., Karp, R.M.: A dynamic programming approach to sequencing problems. Journal of SIAM 10, 196–210 (1962)
12. Iwama, K., Nakashima, T.: An improved exact algorithm for cubic graph TSP. In: Lin, G. (ed.) COCOON 2007. LNCS, vol. 4598, pp. 108–117. Springer, Heidelberg (2007)
13. Karp, R.M.: Dynamic programming meets the principle of inclusion and exclusion. Operations Research Letters 1, 49–51 (1982)
14. Li, M., Corneil, D.G., Mendelsohn, E.: Pancyclicity and NP-completeness in planar graphs. Discrete Applied Mathematics 98, 219–225 (2000)
15. Müller, H.: Hamiltonian circuits in chordal bipartite graphs. Discrete Mathematics 156, 291–298 (1996)
16. Roussopoulos, N.D.: A max {m,n} algorithm for determining the graph H from its line graph G. Information Processing Letters 2, 108–112 (1973)
17. Ryjáček, Z.: On a closure concept in claw-free graphs. Journal of Combinatorial Theory, series B 70, 217–224 (1997)
18. Thomassen, C., Toft, B.: Non-separating induced cycles in graphs. Journal of Combinatorial Theory, Series B 31, 199–224 (1981)
19. Woeginger, G.J.: Open problems around exact algorithms. Discrete Applied Mathematics 156, 397–405 (2008)