Finding a Second Hamiltonian Cycle in Barnette Graphs

by

Arash Haddadan

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Combinatorics and Optimization

Waterloo, Ontario, Canada, 2015

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

We study the following two problems: (1) finding a second room-partitioning of an oik, and (2) finding a second Hamiltonian cycle in cubic graphs. The existence of solution for both problems is guaranteed by a parity argument.

For the first problem we prove that deciding whether a 2-oik has a room-partitioning is NP-hard, even if the 2-oik corresponds to a planar triangulation.

For the problem of finding a second Hamiltonian cycle, we state the following conjecture: for every cubic planar bipartite graph finding a second Hamiltonian cycle can be found in time linear in the number of vertices via a standard pivoting algorithm. We fail to settle the conjecture, but we prove it for cubic planar bipartite WH(6)-minor free graphs.

Acknowledgements

First and foremost, I thank my supervisor Laura Sanità for her incredible guidance and mentorship. This work could not be completed without her support. I would also like to express my gratitude towards University of Waterloo for their financial support. I thank Bruce Richter and Penny Haxell for spending time to read this thesis and for their insightful comments. Finally, I thank my family and friends for their support and kindness during all these years.

Table of Contents

List of Figures

Introduction

This thesis is concerned with problems where we are given one solution and the existence of another solution is guaranteed by a simple parity argument. In this case, the second solution can be algorithmically obtained via some sort of *exchange algorithm*. However, these algorithms are often not efficient; i.e. there are instances for which they require a number of steps exponential in the size of the instance.

The two problems we study in this thesis are (1) the problem of finding a second room-partitioning of a d-oik, and (2) the problem of finding a second Hamiltonian cycle in a cubic graph.

0.0.1 Second room-partitioning of d-oiks

Consider a planar triangulation G on set V of vertices, and let R be the set of faces (triangles) of G. It is an easy observation that every edge of a triangulation is in exactly two triangles. Hence, object (V, R) consists of a set V of vertices and a set R of triangles (called *rooms*), where each room is a subset of size 3 of V. Moreover, (V, R) has the following property: every subset of size 2 of V, is either in 0 or 2 rooms in R. If the 2-subset corresponds to an edge of the triangulation, it appears in exactly two rooms, otherwise it appears in no room. A *room-partitioning* of (V, R) is a subset of rooms of R that partitions the vertices in V. A theorem of Edmonds [14] shows that room-partitionings come in pairs.

What Edmonds shows is indeed more general. He uses the concept of d-dimensional Euler complexes or d-oiks, that was introduced independently by Todd [39], as follows.

Definition 1.1.1. A *d-dimensional Euler complex* $C = (V, R)$, or a *d-oik*, for $d \geq 1$, is given by a set V of *vertices* of C, and a set $R \subseteq \{A \subseteq V \ : \ |A| = d + 1\}$, called the set of *rooms* of C, satisfying the following property: every $B \subseteq V$, $|B| = d$, appears in an even number of rooms.

For a planar triangulation G, it is easy to see that $(V(G), F(G))$ is a 2-oik, where $V(G)$ and $F(G)$ are the vertex set and the face set of G, respectively.

A room-partitioning of d-oik $C = (V, R)$ is $P \subseteq R$, such that each vertex $v \in V$ appears in exactly one room in P. Edmonds proves that every d-oik has an even number of room-partitionings. The proof is simple, and yields an algorithm for finding a second room-partitioning given a first one, known as the *pivoting algorithm* or the *exchange algorithm*.

Interesting examples of room-partitionings of d-oiks are perfect matchings in Eulerian graphs. A graph is Eulerian if all its vertices have even degree. For an Eulerian graph $G = (V, E)$, it is easy to see that $C = (V, E)$ is in fact a 1-oik. Perfect matchings of G correspond to room-partitionings of C. Hence, the exchange algorithm can solve the problem of finding a second perfect matching in Eulerian graphs, given a first one. Edmonds and Sanità [16] constructed an infinite family of Eulerian graphs for which the exchange algorithm takes an exponential number of steps in the number of vertices of the graph before termination. However, in his Ph.D. thesis, Merschen [30] shows that if the Eulerian graph is bipartite, then the exchange algorithm computes a second perfect matching in time linear in the number of vertices of the graph.

Another example of d-oiks is the one we started off with: planar triangulations. A room-partitioning of a planar triangulation is a set of faces that partitions the vertices. Again, one could apply the exchange algorithm to find a second room-partitioning of a planar triangulation, given one, but the algorithm might have a running time exponential in the number of vertices in the triangulation. There is a possibility that this problem is intractable in the following sense: if there is an efficient algorithm for finding a second room-partitioning of planar triangulations, then there exists an efficient algorithm for finding a Nash equilibrium in bimatrix games.

The problem of finding a perfect matching of a given graph is easy by the means of Edmonds's well-known blossom algorithm [13]. Hence, one does not need to rely on an exchange algorithm for finding a second perfect matching in Eulerian graphs. For the case of planar triangulations, we show that a similar thing is unlikely to happen. In particular, we prove that the problem of deciding whether a planar triangulation has a (first) room-partitioning is NP-complete.

Theorem 1.3.2. *Determining whether a planar triangulation has a room-partitioning is NP-complete.*

0.0.2 Second Hamiltonian cycle in cubic graphs

Consider the following problem: given a cubic graph G and a Hamiltonian cycle H of G containing an edge e, is there another Hamiltonian cycle that contains e?

A classical result of Smith [40] shows that the answer to this question is always yes. However, his proof is not constructive. Thomason [37] later found a short and constructive proof for finding a second Hamiltonian cycle through a given edge in a cubic graph. The algorithm proposed by Thomason, called *the lollipop algorithm*, is based on a basic parity argument, and iterates by exchanging edges with simple rules.

In his seminal paper of 1992, Papadimitriou [32] reviewed Thomason's proof to illustrate the complexity class PPA (for Polynomial Parity Argument). Although no exponential lower bounds were known on the number of iterations required by the lollipop algorithm to find a second Hamiltonian cycle, Papadimitriou conjectured that this problem is in fact hard to solve, and that the lollipop algorithm is not efficient. A year later, Krawczyk [28] constructed an infinite family of cubic graphs with $8n + 2$ vertices, for which the lollipop algorithm takes at least 2^n iterations. Krawczyk's graphs are cubic and planar.

Barnette's conjecture [3] states that all 3-connected cubic planar bipartite graphs (Barnette graphs) have a Hamiltonian cycle. Hamiltonicity of these graphs is intensively studied in the context of graph theory. The conjecture remains open after more than four decades. The complexity of finding a (first) Hamiltonian cycle in Barnette graphs is also open. About the complexity of finding a second Hamiltonian cycle, we propose the following conjecture.

Conjecture 3.2.2. *Given a Hamiltonian cycle H in a cubic planar bipartite graph G and edge e in H, the lollipop algorithm finds a second Hamiltonian cycle through e in time linear in the number of vertices of G.*

A corollary of Conjecture 3.2.2 would be that the lollipop algorithm is an efficient algorithm for finding a second Hamiltonian cycle in a Barnette graph. We fail to settle Conjecture 3.2.2, however, we prove that the lollipop algorithm terminates in time linear in the number of vertices in cubic planar bipartite WH(6)-minor free graphs. Graph WH(6) consists of a cycle of five vertices together with a vertex connected to all the vertices in the cycle (See Figure 0.0.2.). A graph G has a WH(6)-minor if one could obtain from G a graph isomorphic to WH(6) by doing minor operations (edge deletion and edge contraction). Precisely, we prove the following theorem.

Theorem 3.3.1. *Given a Hamiltonian cycle H in a cubic planar bipartite WH(6)-minor free graph G, and edge e in H, the lollipop algorithm finds a second Hamiltonian cycle through e in time linear in the number of vertices of G.*

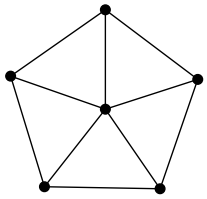


Figure 1: Graph WH(6).

0.0.3 Organization of the thesis

The organization of the thesis is as follows. We begin in Chapter 1 with recalling basic properties of d-oiks. We will see the proof that room-partitionings in d-oiks come in pairs. Next, we will review related works. In particular, we discuss the problem of finding a Nash equilibrium in bimatrix games, and the problem of finding a second room-partitioning with opposite sign. Then, we review the suitable complexity classes when dealing with these types of problems. Section 1.2 is dedicated to the problem of finding a second perfect matching in Eulerian graphs. A detailed description of the exchange algorithm for solving this problem is provided in this section. Later in this section, we focus on bipartite Eulerian graphs, and see how pairing the edges guarantees polynomiality of the exchange algorithm in these graphs. Section 1.3 elucidates the problem of finding a room-partitioning of a planar triangulation. We prove that deciding whether a planar triangulation has a room-partitioning is NP-complete.

Chapter 2 focuses on the problem of finding a second Hamiltonian cycle in cubic graphs. We start in Section 2.1 by proving Smith's theorem using a similar parity technique we used for room-partitionings of oiks. We will describe the algorithm, and fix notations that we will need later. Then, we will briefly review related works in Section 2.2. As in the first chapter, we discuss complexity classes that seem to capture the complexity of the problem. In the same section, we will see Krawczyk's construction that shows that the lollipop algorithm has exponential running time in cubic planar graphs.

Chapter 3 contains the motivation for proposing Conjecture 3.2.2 and the proof of Theorem 3.3.1. This latter proof is divided into three subsections.

Finally, in Chapter 4 we will close the thesis with a brief conclusion and future challenges.

Chapter 1

Finding a second room-partitioning of an oik

In this chapter we study Eulerian complexes, or oiks.

1.1 Preliminaries

The main motivation to study oiks is to generalize the parity argument for showing the exitence of an even number of room-partitionings. A famous example of this parity argument is given by the Lemke-Howson algorithm for computing a Nash equilibrium in bimatrix games. Before getting into this relationship, let us introduce oiks in more detail.

1.1.1 d-oiks and the exchange algorithm

We start by recalling the definition of an oik.

Definition 1.1.1 (Oik)**.** A *d-dimensional Euler complex* $C = (V, R)$, or a *d-oik*, for $d \geq 1$, is given by a set V of *vertices* of C, and a set $R \subseteq \{A \subseteq V \ : \ |A| = d + 1\}$, called the set of *rooms* of C, satisfying the following property: every $B \subseteq V$, $|B| = d$, appears in an even number of rooms.

Definition 1.1.2. A *room-partitioning of a d-oik* (V, R) is a set $P \subseteq R$, such that each vertex $v \in V$, appears in exactly one room in P.

Example 1.1.3. Let $G = (V, E)$ be an Eulerian graph. Then $C = (V, E)$ is a 1-oik. A room-partitioning of C corresponds to a perfect matching of G.

A special case of d-oiks are simplicial pseudo manifolds.

Definition 1.1.4. A *rank-d simplicial pseudo manifold*, for $d \geq 1$, is a pair (V, R), where V is a set of *vertices*, and R is a collection of subsets of V with size $d+1$, called the set of *rooms*, such that every $B \subseteq V$, $|B| = d$, appears in either zero or two rooms.

For convenience, we would refer to a rank-d simplicial pseudo manifold as a d-manifold.

Example 1.1.5. Let $G = (V, E)$ be a planar triangulation, and let F be the set of faces of G. Then, (V, F) is a 2-manifold.

The heart of the exchange algorithm is a parity argument that we will later see many times in this thesis.

Theorem 1.1.6 (Edmonds [14])**.** *A d-oik has an even number of room-partitionings.*

Before proving the theorem, we need a definition.

Definition 1.1.7. For a d-oik $C = (V, R)$ and $w \in V$, a *w-almost room-partitioning of C* is a set $P \subseteq R$, such that (1) w is not in any room of P, (2) there is a vertex $v \in V \setminus \{w\}$ that is in exactly two rooms of P, called the *duplicate vertex of P*, and (3) every vertex in $V \setminus \{v, w\}$ appears in exactly one room of P.

Proof of Theorem 1.1.6. Let $C = (V, R)$ be a d-oik. Choose $w \in V$ arbitrarily. Now we will construct a graph G_C^w, whose vertex set is given by all w-almost room-partitionings of C and the set of all room-partitionings of C. Two vertices P_1 and P_2 of G_C^w are connected by an edge if they can be obtained from each other by replacing one room with another, i.e. by swapping two rooms. We call G_C^w the *exchange graph*.

It is easy to see that each vertex in G_C^w that corresponds to a w-almost room-partitioning has an even degree. To see this, consider a w-almost room-partitioning P, with duplicate vertex v. Let r_1, r_2 be the rooms in P that contain v. A neighbour of P in G_C^w is a w-almost room-partitioning of C or a room-partitioning of C, that is obtained from P by removing one room in P, and adding another room in $R \setminus P$ to P. Thus, the room that we remove must be either r_1 or r_2. Otherwise, by adding any other room we would end up with two duplicate vertices. Now consider $P \setminus r_1$. By the property of d-oiks, there is an even number of rooms that have $r_1 \setminus \{v\}$ in common. Hence, there is an odd number of them different

from r_1. A similar argument holds for $P \setminus r_2$. Therefore, all w-almost room-partitionings have even degree in G_C^w.

In addition, it is easy to see that a room-partitioning P has odd degree in G_C^w. This is because one has to remove the room that contains w, namely $r \in P$. Since there is an even number of rooms that have $r \setminus w$ in common, the number of vertices in G_C^w that are neighbours of P is odd.

The statement of the theorem then follows by the following simple observation: A graph has an even number of odd degree vertices! □

The parity argument in the proof of Theorem 1.1.6 will come forward many times later in this chapter and in the next chapter. Theorem 1.1.6 has many corollaries. Consider the following computational problem.

Definition 1.1.8 (Finding a second room-partitioning of an oik)**.** Given a d-oik C and room-partitioning P of C, find a room-partitioning of C that is different from P.

Recall the exchange graph in the proof of Theorem 1.1.6. In case our d-oik is in fact a d-manifold, all vertices in the exchange graph have degree at most two. Thus, components of the exchange graph are cycles and paths. Moreover, a room-partitioning corresponds to a leaf in a path. Trivially, there is another leaf (room-partitioning) at the other end of the path. Edmonds's exchange algorithm basically moves along this path to obtain another room-partitioning.

For finding a second room-partitioning of a d-oik (V, R), we can pair the rooms as follows. By definition, for each d-subset B of V, there is an even number of rooms that contain B. Thus, we can pair them up for each d-subset. Now, consider the exchange graph for some $w \in V$. Recall that at vertex P of the exchange graph that corresponds to a w-almost room-partitioning, we have two choices for dropping a room, namely r_1 and r_2. However, for adding we could have plenty of choices. We can restrict ourselves to the rooms paired with the d-subsets $P \setminus r_1$ and $P \setminus r_2$, respectively. A similar thing can be done for vertices in the exchange graph that correspond to room-partitionings of C. We will describe the exchange algorithm for 1-oiks in more detail in Section 1.2.1.

1.1.2 Related Works

The parity argument mentioned in the proof of Theorem 1.1.6 is a central idea in many existential proofs. The most well-known one is probably the existence of a Nash equilibrium

in non-degenerate bimatrix games. A classical result of 1951 by Nash shows that a Nash equilibrium always exists in bimatrix games. A bimatrix game is specified by two matrices A and B of the same dimension. Two players a and b simultaneously choose a row index i and a column index j, respectively. Player a receives payoff A_{ij} and player b receives payoff B_{ij}. Each player aims to maximize his expected payoff with respect to a probability distribution over the set of possible choices, which is called a *mixed strategy*. A Nash-equilibrium is a pair of mixed strategies such that no player can change his strategy in order to increase his expected payoff while his opponent mixed strategy is fixed.

Finding such equilibrium is a long standing open problem, and after many decades there are no constructive solutions that output a Nash equilibrium in time polynomial in the size of the input. A well-known algorithm for computing a Nash equilibrium in bimatrix games is the Lemke-Howson algorithm [29]. The algorithm is based on the same parity argument as in the proof of Theorem 1.1.6. In particular, Lemke and Howson show that finding a Nash equilibrium could be solved by finding a second completely labelled vertex in a labelled polytope. One could show that these completely labelled vertices come in pairs.

Edmonds and Sanità [15] describe a common generalization for room-partitionings of oiks and Nash equilibria in bimatrix games. In particular, they proved that non-degenerate bimatrix games could be described via two manifolds on the same set of vertices, where Nash equilibria correspond to room-partitionings obtained by selecting one room from each manifold.

Végh and von Stengel [41] introduce an abstract framework of *complementary pivoting with direction*, that also brings Nash equilibria in bimatrix games and room-partitionings of oiks under one common general framework. They extend a concept of sign defined by Shapley [34] to this common framework (and hence to room-partitionings) to show that room-partitioning of different signs come in pairs. They show that the exchange algorithm computes a room-partitioning with opposite sign for their notion of sign. While the exchange algorithm in general has an exponential worst-case running time, they provide a different algorithm that efficiently finds a perfect matching of opposite sign in Eulerian graphs (Theorem 12 in [41]).

1.1.3 Related complexity classes

TFNP (for Total Function Nondeterministic Polynomial) is the class of all search problems where a solution is always guaranteed to exist. A consequence of Nash's theorem is that the problem of finding a Nash equilibrium in non-degenerate bimatrix games (2-Nash) is

in TFNP. By an easy argument one can show that there is no NP-complete problem in TFNP unless NP=co-NP. Moreover, it seems difficult to find a polynomial-time algorithm for computing a Nash equilibrium for bimatrix games. This, together with the nature of the Lemke-Howson algorithm for solving 2-Nash, was the motivation to introduce a complexity class that captures the complexity of 2-Nash. To this end, Papadimitriou [32] introduced the complexity class PPAD (for Polynomial Parity Argument on a Directed graph). The definition of PPAD, unlike other classes like NP is based on the following artificial problem.

Definition 1.1.9 (End of the line [8])**.** An instance of End of the line is a pair $(M, 0^n)$ with the following properties. M is a polynomial-time Turing machine that encodes a directed graph G with vertex set $\{0,1\}^n$, such that every vertex has in-degree and out-degree at most one, in the following sense: for $v \in \{0,1\}^n$, M outputs $M(v)$ in time polynomial in n. $M(v)$ consists of a unique ingoing neighbour of v and a unique outgoing neighbour of v, if they exist. Moreover, in G, vertex 0^n (*standard leaf*) has out-degree one and in-degree zero. The problem asks to find a leaf of G, different from 0^n.

It is easy to see that End of the line is in fact in TFNP. The proof is by the exact same argument as the one in the proof of Theorem 1.1.6: a graph has an even number of odd degree vertices!

But how hard is End of the line? The directed graph G has an exponential number of vertices, thus, one could not compute the whole graph in polynomial time. However, since M is a polynomial-time Turing machine, we could always find the outgoing and ingoing neighbours of a vertex in polynomial-time. Therefore, a natural way for solving End of the line is to move along the path that contains the standard leaf until we find another leaf. However, this has exponential worst case.

We are now ready to define PPAD.

Definition 1.1.10 (PPAD)**.** A problem A is in the complexity class PPAD if there is a polynomial-time reduction from A to End of the line.

It is known that P $\subseteq$ PPAD $\subseteq$ NP. A distinction between PPAD and P or NP would immediately imply that P$\neq$ NP. In 2006, Chen and Deng [6] proved that 2-Nash is PPAD-complete. It is widely believed that PPAD-complete problems are hard to solve efficiently (see [21] for a well-written survey about PPAD and the complexity of computing a Nash equilibrium). Notice that the Lemke-Howson algorithm is not efficient for 2-Nash. Savani and von Stengel [33] find an infinite family of bimatrix games for which the Lemke-Howson algorithm requires a number of steps exponential in the dimension of the game, before terminating.

Another complexity class concerned with parity argument is PPA (for Polynomial Parity Argument). PPA is defined similarly to PPAD, with a similar canonical problem.

Definition 1.1.11 (End of the line without direction)**.** An instance of End of the line without direction is a pair $(M, 0^n)$ with the following properties. M is a polynomial-time Turing machine that generates a graph G with vertex set $\{0,1\}^n$, such that every vertex has degree at most two, in the following sense: for $v \in \{0,1\}^n$, M outputs $M(v)$ in time polynomial in n. $M(v)$ consists of at most 2 neighbours of v. Moreover, in G, vertex 0^n has only one neighbour. The problem asks to find a leaf of G, different from 0^n.

Definition 1.1.12 (PPA)**.** A problem A is in the complexity class PPA if there is a polynomial-time reduction from A to End of the line without direction.

It is known that P $\subseteq$ PPAD $\subseteq$ PPA $\subseteq$ NP. Any distinction between PPA and PPAD would prove P $\neq$ NP. In the second chapter, we will study the problem of finding a second Hamiltonian cycle in cubic graphs. We try to talk about potential differences of these two classes in that context.

It is not a difficult task to show that finding a second room-partitioning of an oik falls in the complexity class PPA by the means of the exchange algorithm.

1.2 Second perfect matching in Eulerian graphs

The main idea in this section is to show polynomiality of the exchange algorithm on bipartite Eulerian graphs, proved in [30], since some of these ideas will be later used in Chapter 2. First, we need to formally describe the algorithm.

1.2.1 Exchange algorithm on Eulerian graphs

Recall the problem of finding a second room-partitioning of an oik, and the interesting special case of room-partitionings of 1-oiks, which correspond to perfect matchings in Eulerian graphs. In particular, observe the following corollary of Theorem 1.1.6 .

Corollary 1.2.1. *Every Eulerian graph has an even number of perfect matchings.*

The exchange algorithm applied in this setting outputs another perfect matching in Eulerian graphs. Since the graph has even degree at each vertex, we can pair up the edges

incident to any vertex. Hence, we construct a pairing function. Let us formally define pairing functions. For a vertex v in a graph G, we let $\delta(v)$ denote the set of edges of G incident to v.

Definition 1.2.2. For an Eulerian graph $G = (V, E)$, a *pairing function for* G is $\mathcal{F} : (V \times E) \to E$, such that: (i) $\mathcal{F}$ is defined for all pairs (v, e), where $v \in V$ and $e \in \delta(v)$, and (ii) for $v \in V$ and $e \in \delta(v)$, $\mathcal{F}(v, e) = e'$ with $e' \neq e$, and $\mathcal{F}(v, e') = e$.

A pairing function for an Eulerian graph can be easily obtained e.g. by following an Eulerian tour in the graph.

Algorithm 1 Exchange algorithm on Eulerian graphs

Input : Eulerian graph $G = (V, E)$, a perfect matching M of G and a pairing function $\mathcal{F}$ for G.
Output: A perfect matching $M' \neq M$ of G.

```
Choose w ∈ V
e ← M ∩ δ(w)
M ← M \ {e}
exposed ← endpoint of e that is not w
while (1 = 1) do
    e' ← F(exposed, e)
    M ← M ∪ {e'}
    duplicate ← endpoint of e' that is not exposed
    if duplicate = w then
        M' ← M
        return M' and terminate
    end
    e ← (δ(duplicate) ∩ M) \ {e'}
    M ← M \ {e}
    exposed ← endpoint of e that is not duplicate
end
```

Termination of Algorithm 1.2.1 is guaranteed by the same parity argument as in the proof of Theorem 1.1.6.

1.2.2 Bipartite Eulerian graphs

Recall that the exchange algorithm is not efficient for general Eulerian graphs as shown by Edmonds and Sanità [15]. Would it be efficient if we restrict the graph a bit more? As we

will see in the following theorem, the answer is yes for bipartite Eulerian graphs.

Theorem 1.2.3 (Merschen [30]). *Let $G = (V, E)$ be a bipartite Eulerian graph, and M be any perfect matching of G. For any pairing function $\mathcal{F} : (V \times E) \to E$, the exchange algorithm terminates in at most $O(|V|)$ iterations.*

Before proving Theorem 1.2.3 let us define the add-remove walk for an execution of the exchange algorithm.

Definition 1.2.4. For Eulerian graph $G = (V, E)$, perfect matching M of G, pairing function $\mathcal{F}$ of G, and vertex $w \in V$, we define the add-remove walk $W(G, M, \mathcal{F}, w)$ as follows.

$$W(G, M, \mathcal{F}, w) = (w = a_0, f_0, r_0, e_1, a_1, f_1, r_1, e_2, \ldots, a_{n-1}, f_{n-1}, r_{n-1}, e_n, a_n = w), \quad (1.1)$$

where in the execution of Algorithm 1.2.1 on input $(G, M, \mathcal{F})$ and chosen vertex w, for $i = \{1, \ldots, n\}$, a_i is the duplicate vertex in iteration i and e_i is the edge that is added in iteration i and for $j \in \{0, \ldots, n-1\}$, r_j is the exposed vertex in iteration j, and f_j is the edge that was removed in iteration j.

Proof of Theorem 1.2.3. Consider $W(G, M, \mathcal{F}, w)$ (shortly W) as in (1.1). Notice that since the graph is bipartite, $A = \{a_i \ : \ i \in \{0, \ldots, n-1\}\}$ and $R = \{r_i \ : \ i \in \{0, \ldots, n-1\}\}$ are in two different parts of the bipartition of G, and hence disjoint sets. This also means that W always *arrives* to visit vertices in A by *adding* an edge, and vertices in R by *removing* an edge.

Label the vertices in W with $y_0, \ldots, y_{2n}$. Let $j \in \{0, \ldots, 2n\}$ be the smallest j, such that there exists $i \in \{0, \ldots, j-1\}$ for which $y_i = y_j$. The existence of j is guaranteed by the fact that $y_{2n} = w$.

If $y_j = w$, then we are done. Because the algorithm would terminate at iteration $\frac{j}{2}$, and vertices $y_1, \ldots, y_{j-1}$ are distinct. Hence, $j \leq |V(G)| + 1$.

Now let us show that it is impossible to have $y_j = r_k$, for some k. This would imply that at iteration k the algorithm removes an edge incident to $r_k = y_i$. However, notice that in the set of edges that the algorithm maintains there is only one edge incident to r_k, and that is (y_i, y_{i+1}). On the other hand, the algorithm removes (r_k, a_k) to visit r_k again. This implies $y_{i+1} = y_{j-1} = a_k$, which is a contradiction to the choice of j.

Thus, we may assume that $y_j = a_k$, where $k = \frac{j}{2}$. Also, let $k' = \frac{i}{2}$. We have $a_{k'} = a_k$. The following claim concludes the proof.

Claim 1.2.5. *Let $i, j \in \{0, \ldots, n\}$ such that $i < j$ and $a_i = a_j$. For $\ell \in \{j+1, \ldots, j+i\}$, we have $a_{j+\ell} = a_{i-\ell}$.*

Proof. We proceed by induction on i. If $i = 1$, in iteration $j+1$, the algorithm removes edge e_1 and then since $\mathcal{F}(r_0, f_0) = e_1$ (the algorithm added e_1 in iteration 1), we have $\mathcal{F}(r_0, e_1) = f_0$. Thus, the algorithm adds f_0 to reach $a_0 = a_{j+1} = w$. Clearly, the claim holds in this case.

Now for $i > 1$, the algorithm removes edge e_i and then since $\mathcal{F}(r_{i-1}, f_{i-1}) = e_i$ (the algorithm added e_i in iteration i), we have $\mathcal{F}(r_{i-1}, e_i) = f_{i-1}$. Thus, the algorithm adds f_{i-1} to reach $a_{i-1} = a_{j+1}$. Applying the induction hypothesis on $i-1$, for $\ell \in \{j+1, \ldots, j+1+i-1\}$, we have $a_{j+1+\ell} = a_{i-1-\ell}$. □

The conditions in Claim 1.2.5 holds for $i = k'$ and $j = k$. Hence, Claim 1.2.5 concludes the proof, because it would imply that $a_{k+k'} = a_{k'-k'} = a_0 = w$. Hence, the algorithm would terminate after $k + k'$ iterations. As before we know that $a_0, \ldots, a_{k-1}$ are distinct vertices. By the claim, $a_{k+1}, \ldots, a_{k+k'}$ are also distinct vertices. Thus, $k+k' \leq 2|V(G)|$. □

1.3 Room-partitionings of planar triangulations

Finding a second perfect matching in Eulerian graphs can be efficiently done by using Edmonds's blossom algorithm for finding a perfect matching in a graph. We only need to drop one edge in the initial perfect matching, and apply blossom algorithm, trying out all possible edges in the initial matching one by one until a perfect matching is found. In this section, we want to show that a similar thing is unlikely to happen for finding a second room-partitioning of planar triangulations. We show that determining whether a planar triangulation has a room-partitioning is NP-complete. Consequently, in some sense, if one wishes to give a polynomial-time algorithm for finding a second room-partitioning of a planar triangulation, she is obliged to use more information from the given initial room-partitioning.

We will use a reduction to the planar 3-dimensional matching (planar 3D matching) problem, which is a restriction of the 3-dimensional matching (3D matching) problem. Let us first define the problem.

Suppose we are given three disjoint sets A, B, and C, with $|A| = |B| = |C|$, and set $H \subseteq \{(a, b, c) : a \in A, b \in B, c \in C\}$, called *3D-edges*. Then, the 3D matching problem asks whether there exists a subset $J \subseteq H$, such that (1) for distinct 3D-edges

$(a, b, c), (a', b', c') \in J$, $a \neq a'$, $b \neq b'$ and $c \neq c'$, and (2) J covers all the elements in A, B, and C. If such J exists, we call it a *3D-matching* for the instance given by A, B, C, and H.

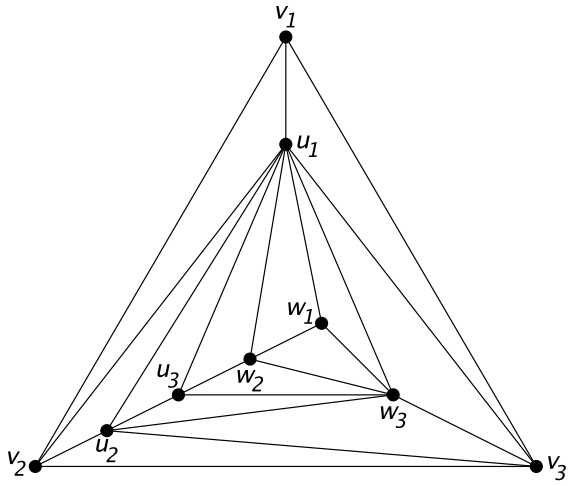
We can associate a bipartite graph $G = (V, E)$ with an instance of 3D matching, such that for every element in A, B, and C, and each 3D-edge in H, we have a vertex in G. A vertex corresponding to $(a, b, c) \in H$, is connected with an edge to vertices corresponding to a, b, and c.

An instance of planar 3D matching is similar to an instance for 3D matching with the restriction that the bipartite graph corresponding to the instance is planar.

Dyer and Frieze [12] showed that planar 3D matching is NP-hard. We will reduce this problem to the problem of finding a room-partitioning of a planar triangulation.

First we have the following straightforward observation.

Observation 1.3.1. Let $G = (V, E)$ be the graph depicted below. Every room-partitioning of (V, F), where F is the set of faces of G, uses room $\{v_1, v_2, v_3\}$.



Proof. We will show that (V, F), admits exactly two room-partitionings:

$$P_1 = \{\{v_1, v_2, v_3\}, \{u_1, u_2, u_3\}, \{w_1, w_2, w_3\}\},$$

and

$$P_2 = \{\{v_1, v_2, v_3\}, \{u_2, u_3, w_3\}, \{u_1, w_1, w_2\}\}.$$

This would prove the observation, since both P_1 and P_2 have $\{v_1, v_2, v_3\}$ as a room.

It is easy to see that P_1 and P_2 are indeed room-partitionings of (V, F). Thus, it suffices to prove there are no more room-partitionings of (V, F). Assume $P' \neq P_1$ and $P' \neq P_2$ is a room-partitioning of (V, F). There are three possibilities for a room-partitioning to cover w_1.

Case 1. $\{u_1, w_1, w_3\} \in P'$. However, this cannot happen, since all the four rooms that cover w_2 contain at least one vertex that is already covered by $\{u_1, w_1, w_3\}$.

Case 2. $\{w_1, w_2, w_3\} \in P'$. Among the four rooms that cover u_3, the only one that can possibly be in P' is $\{u_1, u_2, u_3\}$, since all other rooms contain a vertex that is already covered by $\{w_1, w_2, w_3\}$. However, if $\{w_1, w_2, w_3\} \in P'$ and $\{u_1, u_2, u_3\} \in P'$, we would clearly have $\{v_1, v_2, v_3\} \in P'$, which is a contradiction with $P' \neq P_1$.

Case 3. $\{u_1, w_1, w_2\} \in P'$. Among the four rooms that cover u_3, the only one that can possibly be in P' is $\{u_2, u_3, w_3\}$, since all other rooms contain a vertex that is already covered by $\{u_1, w_1, w_2\}$. However, if $\{u_1, w_1, w_2\} \in P'$ and $\{u_2, u_3, w_3\} \in P'$, we would clearly have $\{v_1, v_2, v_3\} \in P'$, which is a contradiction with $P' \neq P_2$. □

Theorem 1.3.2. *Determining whether a planar triangulation has a room-partitioning is NP-complete.*

Proof. Let disjoint sets A, B, and C and $H \subseteq \{(a, b, c) : a \in A, b \in B, c \in C\}$ be an instance I of planar 3D matching. We show that instance I is a yes-instance if and only if a certain planar triangulation has a room-partitioning.

Let $B_I = (V, E)$, be the bipartite graph corresponding to instance I. Fix a planar embedding of B_I. For each vertex $v \in V$ corresponding to a 3D-edge (a, b, c), we have three neighbours corresponding to elements $a \in A$, $b \in B$, and $c \in C$. Remove vertex v and change the 3D-edge into a 3-face as shown in Figure 1.1. After doing this operation on all vertices that correspond to 3D-edges, we obtain a planar graph G. Planarity of G is because the operation mentioned above clearly preserves planarity.

For each 3D-edge $h \in H$, there is a 3-face in G, whose vertices correspond to the elements in h. Choose an arbitrary 3-face f of G, that corresponds to a 3D-edge and consider the embedding of G on the plane that has f as its outer face. Now we add edges to G while preserving planarity. We call the final graph G'. It is well-known that an edge-maximal planar graph is a planar triangulation, i.e. every face is a 3-face. Remember that each vertex of G' corresponds to an element of $A \cup B \cup C$.

Note that G' has at most $3n - 2$ edges, where $n = |A| + |B| + |C|$, and by Euler's formula it has $2n$ faces, making it possible to enumerate all the faces in polynomial time.

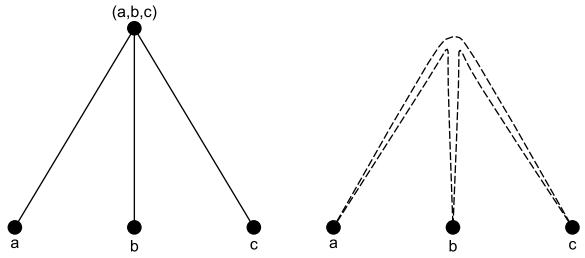


Figure 1.1: Changing 3D-edge $h = (a, b, c)$ into a 3-face

Consider a 3-face f of G'. Let u_1, u_2, and u_3 be the elements in $A \cup B \cup C$, that correspond to the vertices in face f. If $\{u_1, u_2, u_3\} \notin H$, we call f a *bad triangle of* G'. By adding the gadget in Observation 1.3.1 inside every bad triangle of G' we obtain a new graph G_I.

Notice that for each 3D-edge $(a, b, c) \in H$, there is a 3-face f in G_I that contains vertices corresponding to a, b, and c. The bad triangles in G' correspond to triangles in G_I, but they are not 3-faces. We call the subgraph of G_I induced by these triangles and the gadget added inside, a *non-matching triangle of* G_I.

Notice that G_I is a planar triangulation.

Claim 1.3.3. *G_I has a room-partitioning if and only if I is a yes-instance.*

Proof. Suppose I is a yes-instance. Let $J \subseteq H$, be a 3D-matching for I. For each 3D-edge j in J, let f_j be the 3-face in G_I. Let $F = \{f_j : j \in J\}$. Notice that F partitions all the elements in $A \cup B \cup C$. Also, clearly, F does not cover any vertex $v \notin A \cup B \cup C$. Hence, for each non-matching triangle of G_I, it suffices to add a pair of 3-faces suggested by Observation 1.3.1.

For the other direction, assume that P is a room-partitioning of G_I. For element $a \in A$, let f_a be the 3-face (room) in P that covers a. We claim that the vertices in f_a correspond to elements of a 3-edge $h \in H$. If not, first of all observe that we cannot have that all the vertices in f_a correspond to elements in $A \cup B \cup C$, as in construction of G_I, we added a gadget inside each such face. Hence, it must be that f_a contains some vertices that do not correspond to elements in $A \cup B \cup C$. However, this is also a contradiction, since by Observation 1.3.1 all the vertices in G_I that do not correspond to elements in $A \cup B \cup C$ must be covered by rooms that only contain vertices that do not correspond to elements in $A \cup B \cup C$, while f_a contains $a \in A$. □

Claim 1.3.3 completes the proof. □

Chapter 2

Finding a second Hamiltonian cycle

A Hamiltonian cycle of a graph is a sequence of vertices of the graph, where the first and last vertices in the sequence are the same, and all other vertices in the sequence are distinct. The sequence includes all the vertices in the graph and all consecutive vertices in the sequence are adjacent. Finding a Hamiltonian cycle was amongst the first problems proven to be NP-hard. However, in this chapter we will address another problem: we will consider the problem of finding a second Hamiltonian cycle in cubic graphs. For cubic graphs, the existence of a second Hamiltonian cycle is guaranteed if one Hamiltonian cycle of the graph is given.

We will start the chapter by proving the existence of a second Hamiltonian cycle in cubic graphs. Then, we review related works. In particular, we briefly review Hamiltonicity of general regular graphs. Next, we review the literature related to finding Hamiltonian cycles in cubic graphs, and the conjecture of Barnette. In Section 2.3 we study the complexity of the problem.

2.1 The lollipop algorithm

The goal of this section is to introduce the *lollipop algorithm*. We will introduce the prerequisites we need later in the chapter when analyzing the algorithm. To this end, we will first see Thomason's proof for Smith's theorem about the existence of a second Hamiltonian cycle in cubic graphs. After proving Smith's theorem, we will explicitly introduce the lollipop algorithm that is implicitly stated in Thomason's proof.

Thomason's proof of Smith's theorem

We will prove the following theorem.

Theorem 2.1.1 (Smith [40])**.** *For a cubic graph G and edge e of G, there is an even number of Hamiltonian cycles through e.*

Let us first define a lollipop in a cubic graph.

Definition 2.1.2. Given a cubic graph $G = (V, E)$, vertex $w \in V$, and edge $e \in \delta(w)$, a *(w, e)-lollipop of G* is a connected spanning subgraph H of G, such that (i) for exactly one vertex $u \in V \setminus \{w\}$ we have $deg_H(u) = 3$, (ii) $deg_H(w) = 1$, (iii) for all $v \in V \setminus \{u, w\}$ we have $deg_H(v) = 2$, and (iv) $e \in E(H)$.

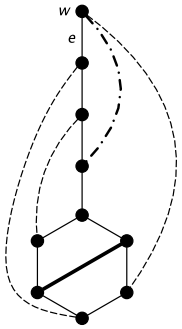


Figure 2.1: A (w, e)-lollipop: the solid thin lines are the edge set of a (w, e)-lollipop H of G, where G is the graph with all the edges depicted in the figure. Note that the embedding of H resembles an upside-down lollipop.

We will drop (w, e) in the definition of a (w, e)-lollipop when it is clear from the context.

We call the degree 3 vertex in a (w, e)-lollipop H the *base of the lollipop*, or $base(H)$. Note that a (w, e)-lollipop consists of a path and a cycle. For (w, e)-lollipop H of G, $P(H)$ is the wu-path in H, where $u = base(H)$, and $C(H)$ is the unique cycle in H.

Proof of Theorem 2.1.1 [37]. Fix an endpoint w of e. Consider a graph X defined as follows. Each vertex of X is either a (w, e)-lollipop of G, or a Hamiltonian cycle containing edge e.

Now consider a lollipop H. Since $u = base(H)$ has degree 3 in H, there are two edges (u, x) and (u, y), incident to u in $C(H)$. Notice that $H - (u, x)$ is a Hamiltonian path. Let (x, z) be the third edge incident to x, such that $(x, z) \notin E(H)$. Observe that $H - (u, x) + (x, z)$ is a (w, e)-lollipop if $z \neq w$, and is a Hamiltonian cycle containing e if $z = w$. A similar argument can be stated for (u, y) as well.

The argument above shows that $H - (u, x) + (x, z)$ and $H - (u, y) + (y, t)$, where (y, t) is the edge incident to y, $(y, t) \notin E(H)$, correspond to vertices of X. Graph X has an edge from H to $H - (u, x) + (x, z)$ and $H - (u, y) + (y, t)$.

Consider a Hamiltonian cycle H containing edge e. Since w has degree two in H, there is an edge $(w, v) = f \neq e$ in H incident to w. Let g be the third edge incident to v, such that $g \notin E(H)$. Notice that $H - f + g$ is either a (w, e)-lollipop or a Hamiltonian cycle of G containing e. Graph X has an edge between H and $H - f + g$.

It is easy to see that the relation defined by the edges of X is symmetric. Therefore every vertex of X has degree at most 2. In particular, a vertex in X has degree one if and only if it corresponds to a Hamiltonian cycle containing e. Recall that a graph has an even number of odd degree vertices. This concludes the proof. □

The algorithm

Graph X in the proof of Theorem 2.1.1 is called the exchange graph for input (G, w, e). The exchange graph consists of paths and cycles. Every leaf in X correspond to a Hamiltonian cycle containing a specific edge e. Moreover, the proof shows how to move from one vertex in X to its neighbouring vertices. Now the problem of finding a second Hamiltonian cycle through a specific edge e, basically boils down to finding a second leaf in X.

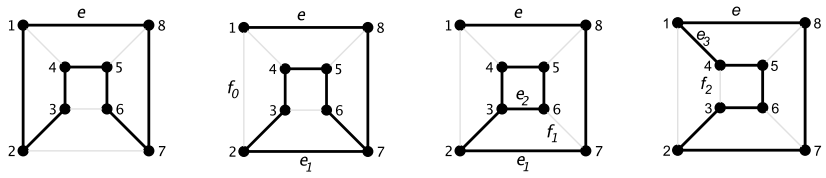
Given one leaf of X, a natural way to find another leaf is to move along the path, until we reach another endpoint. Since all vertices in X have degree at most two, there is no chance of cycling, so we will eventually arrive at another leaf. Let us formalize this.

Tuple (G, H_0, w, e) is a *valid input* for the lollipop algorithm if (1) $G = (V, E)$ is a cubic graph, (2) H_0 is a Hamiltonian cycle of G, (3) $w \in V$, and (4) $e \in \delta(w) \cap E(H_0)$.

For the remainder of this section, we let tuple (G, H_0, w, e) be a valid input for the lollipop algorithm. For this input the algorithm works as follows. At iteration $i \geq 0$, the algorithm finds (w, e)-lollipop H_{i+1}, from H_i.

If $i = 0$, let f_0 be the unique edge in $\delta(w) \cap E(H_0) \setminus \{e\}$, and r_0 be the endpoint of f_0 different from w, and let $e_1 = \delta(r_0) \setminus E(H_0)$. Then $H_1 = H_0 - f_0 + e_1$. If the endpoint of e_1 that is not r_0 is w, we terminate and output H_1, otherwise we go to the next iteration.

If $i \geq 1$, let $H_i = H_{i-1} - f_{i-1} + e_i$, where f_{i-1} is the edge incident to $base(H_{i-1})$, that is in $C(H_{i-1})$ and is different from e_{i-1}, and $e_i \notin E(H_{i-1})$ is the edge incident to the endpoint of f_{i-1} that is not $base(H_{i-1})$, namely r_{i-1}. Label the endpoint of e_i that is not r_{i-1} by a_i. Terminate and output H_i, if $a_i = w$, otherwise we move to the next iteration. Figure 2.2 illustrates the iterations of the lollipop algorithm for a given instance.



(a) H_0 (b) H_1 (c) H_2 (d) H_3

Figure 2.2: The iterations of the lollipop algorithm for the cube graph, with Hamiltonian cycle H_0, and vertex 1 and edge $e = (1, 8)$ as depicted above.

Algorithm 2.1 summarizes this section.

Algorithm 2 Lollipop algorithm for cubic graphs

Input : Valid input (G, H, w, e)
Output: A Hamiltonian cycle of G different from H
$e' \leftarrow (E(H) \cap \delta(w)) \setminus \{e\}$
$H \leftarrow H - e'$
$degree1 \leftarrow$ endpoint of e' that is not w
while $(1 = 1)$ **do**
  $e'' \leftarrow (\delta(degree1) \setminus \{e'\}) \setminus E(H)$
  $H \leftarrow H + e''$
  $base \leftarrow$ endpoint of e'' that is not $degree1$
  if $base = w$ **then**
    return H and terminate
  end
  $e' \leftarrow (E(C(H)) \cap \delta(base)) \setminus \{e''\}$
  $H \leftarrow H - e'$
  $degree1 \leftarrow$ endpoint of e' that is not $base$
end

The add-remove walk for instance (G, H_0, w, e)

We define the add-remove walk to simplify notation when proving our theorem about the lollipop algorithm later on. Suppose that the execution of the lollipop algorithm on a valid input (G, H_0, w, e) terminates at iteration n.

Definition 2.1.3. The add-remove walk for input (G, H_0, w, e) is

$$W(G, H_0, w, e) = (w = a_0, f_0, r_0, e_1, a_1, f_1, r_1, e_2, \ldots, a_{n-1}, f_{n-1}, r_{n-1}, e_n, a_n = w).$$

Notice that for $i = 0, \ldots, n-1$, edges f_i and e_{i+1} have a common endpoint r_i, and for $i = 1, \ldots, n-1$, edges e_i and f_i have a common endpoint a_i. Hence $W(G, H_0, w, e)$ is walk in graph G. Figure 2.3 shows the add-remove walk for the input illustrated in Figure 2.2.

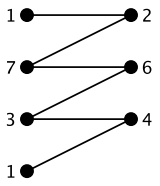


Figure 2.3: The add remove walk for the input in Figure 2.2.

In the execution of the lollipop algorithm for a valid input (G, H_0, w, e), we say that an edge e' is *added in iteration* i, if $e' = e_{i+1}$. We say that e' is *removed in iteration* i, if $e' = f_i$. Moreover, we say that edge e' is *touched in iteration* i, if it is either added or removed in iteration i. Finally, for a vertex v we say that *the algorithm arrives at* v *in iteration* i, if $v = a_{i+1}$ or $v = r_i$.

For a subgraph J of G, we say that the algorithm *enters* J *in iteration* i, if the algorithm does not arrive at a vertex $v \in V(J)$ in iteration $i-1$, but it arrives at a vertex $v \in V(J)$ in iteration i. In particular, we say that the algorithm *enters* J *through* v *in iteration* i. Similarly, we say that the algorithm *leaves* J *in iteration* i, if the algorithm arrives at a vertex in J in iteration $i-1$ and arrives at a vertex not in J in iteration i. Also, we say that the algorithm *leaves* J *through* v *in iteration* i.

We are particularly interested in the portion of the add-remove walk between a pair of lollipops, or between a Hamiltonian cycle and a lollipop.

Definition 2.1.4. For any $i, j \in \{0, \ldots, n\}$ where $j \geq i$, the *add-remove walk between* H_i *and* H_j for input (G, H_0, w, e), is

$$W(G, H_0, w, e)_{H_i, H_j} = (a_i, f_i, r_i, e_{i+1}, a_{i+1}, f_{i+1}, r_{i+1}, \ldots, a_{j-1}, f_{j-1}, r_{j-1}, e_j, a_j).$$

We will abbreviate $W(G, H_0, w, e)$ by W and $W(G, H_0, w, e)_{H_i,H_j}$ by W_{H_i,H_j} when the context is clear.

2.2 Related works

The problem of finding a Hamiltonian cycle in graphs is one of the most fundamental problems in graph theory and computer science. For regular graphs, the existence of a second Hamiltonian cycle is also well-studied. In addition, in Hamiltonian cycles are intensively studied in the context of Traveling Salesman Problem (TSP). We will review some related results in this section.

2.2.1 Second Hamiltonian cycle in regular graphs

Generalizing Theorem 2.1.1

Suppose that H_0 is a given Hamiltonian cycle of a graph G, where all vertices have odd degrees. Now consider the exchange graph X for input (G, H_0, w, e), where $w \in V(G)$ and $e \in E(H_0) \cap \delta(w)$.

Similar to the proof of Theorem 2.1.1 in Section 2, in X the vertices that correspond to Hamiltonian cycles have odd degrees, while the vertices that correspond to (w, e)-lollipops have even degrees. By a simple parity argument, one can derive the following theorem.

Theorem 2.2.1 (Thomason [37])**.** *Every graph G where all vertices have odd degree, has an even number of Hamiltonian cycles containing a given edge e.*

Sheehan's conjecture

Theorem 2.2.1 raised a question about the existence of a second Hamiltonian cycles in k-regular graphs. In particular, the question concerns the existence of *uniquely Hamiltonian graphs*. A graph is uniquely Hamiltonian if it contains exactly one Hamiltonian cycle. In 1975, Sheehan [35] proposed the following conjecture.

Conjecture 2.2.2. *There are no uniquely Hamiltonian 4-regular graphs.*

Suppose Sheehan's conjecture holds. Then, consider a k-regular graph G for some even number $k \geq 4$. If G has one Hamiltonian cycle C, then G has a second Hamiltonian cycle.

The proof is by induction on k. The base case follows from Sheehan's conjecture. For the induction step notice that $G - C$ is a $(k-2)$-regular graph. By Petersen's Theorem [10], $G - C$ has a 2-factor F. Now $G - F + C$ is $(k-2)$-regular and Hamiltonian, hence by induction it has a second Hamiltonian cycle C' that is also a second Hamiltonian cycle for G.

Therefore, Sheehan's conjecture together with Theorem 2.2.1, would imply that no k-regular graph with $k \geq 3$ is uniquely Hamiltonian. The conjecture remains open after about four decades. Thomassen [38] proved that no k-regular graph with $k \geq 72$ is uniquely Hamiltonian. The proof relies on the probabilistic method. Later, this result was improved to $k \geq 23$ [25].

2.2.2 Hamiltonicity of cubic graphs

In 1884, Tait [36] conjectured that all 3-connected cubic planar graphs are Hamiltonian. His motivation was to find a proof for the four-color theorem. After 60 years, his conjecture was disproved by Tutte [40]. Tutte then proposed his own conjecture, that all 3-connected cubic bipartite graphs are Hamiltonian. The conjecture of Tutte was later disproved by Horton [4]. Barnette's conjecture is a combination of Tait and Tutte conjectures, asserting that all 3-connected cubic planar bipartite graphs (Barnette graphs) are Hamiltonian. Barnette's conjecture is still open after 50 years.

Barnette's conjecture

Holton, Manvel, and McKay [9] proved that Barnette's conjecture holds for graphs with at most 64 vertices. Their proof uses computers combined with smart combinatorial ideas for pruning the search space. Later, they announced they have a proof for up to 84 vertices [26]. Goodey [23] proved that if all the faces of a Barnette graph are 4-faces or 6-faces, then the graph is Hamiltonian.

Another approach for proving the conjecture is proving its equivalence with more restrictive statements. An example would be Theorem 2.2.3.

Theorem 2.2.3 (Kelmans [27])**.** *Barnette's conjecture holds if and only if for every face f of a Barnette graph G, and edge e and e' on f, there is a Hamiltonian cycle that uses e and avoids e'.*

Alt et al. [2] use the planar dual graph of a Barnette graph, which in fact is an Eulerian triangulation, to give a sufficient condition for the existence of Hamiltonian cycles in Barnette graphs.

Theorem 2.2.4 (Alt et al. [2]). *Suppose G^* is an Eulerian triangulation. If the vertices of G^* can be properly colored with colors red, green, and blue, such that every cycle whose vertices are not colored with blue, contains a vertex of degree 4, then the planar dual G of G^* is Hamiltonian*

They use a lemma from [19] that translates Hamiltonian cycles of cubic planar graphs into special subgraphs in planar triangulations, called permeating subtrees.

Definition 2.2.5. Let G be a planar triangulation. Subgraph H of G is a *permeating subtree of* G, if (i) H is an induced subgraph, (ii) H is a tree, and (iii) each face of G is incident to a vertex of H.

Theorem 2.2.6 (Florek [19]). *Let G be a cubic planar graph, and G^* be the planar dual of G. Then G is Hamiltonian if and only if G^* has two disjoint permeating subtrees that partition $V(G^*)$.*

Proof sketch. Fix a planar embedding of G. Suppose that G is Hamiltonian. Let C be a Hamiltonian cycle of G. Cycle C partitions the plane into two regions. Let F_1 be the faces that are inside the cycle and F_2 be the faces outside C. F_1 and F_2 correspond to a set of vertices V_1 and V_2 in G^*, respectively. Notice that $F_1 \cup F_2 = F$, hence $V_1 \cup V_2 = V(G^*)$. Then $G^*[F_1]$ and $G^*[F_2]$ are two disjoint permeating subtrees that partition $V(G^*)$.

Now suppose G^* has two permeating subtrees that partition the vertices of G^*, namely T_1 and T_2. Contract the vertices in T_1 and T_2 to a single vertex to obtain G'. There is one-to-one correspondence between the faces in G' and G^*. Graph G' consists of two vertices with many multiple edges, but no loops. The planar dual of G' is a cycle. This cycle corresponds to a Hamiltonian cycle for G. □

Figure 2.4 shows an example that illustrates the correspondence between permeating subtrees and a Hamiltonian cycle, as in Theorem 2.2.6.

Short TSP tours in cubic graphs

Since not all cubic graphs are Hamiltonian, problems related to finding Hamiltonian cycles in cubic graphs are also well-studied. A natural one is the Traveling Salesman Problem (TSP). A TSP tour in a graph G is a connected spanning Eulerian multi-subgraph of G. Here by multi-subgraph we mean that a TSP tour can use each edge in G multiple times.

Boyd et al. [5] find a TSP tour of a cubic graph G with at most $\frac{4}{3}|V(G)|-2$ edges. This was improved by Correa et al. [7] to $(\frac{4}{3}-\epsilon)|V(G)|-2$ edges, where $\epsilon \geq \frac{1}{61236}$. This result

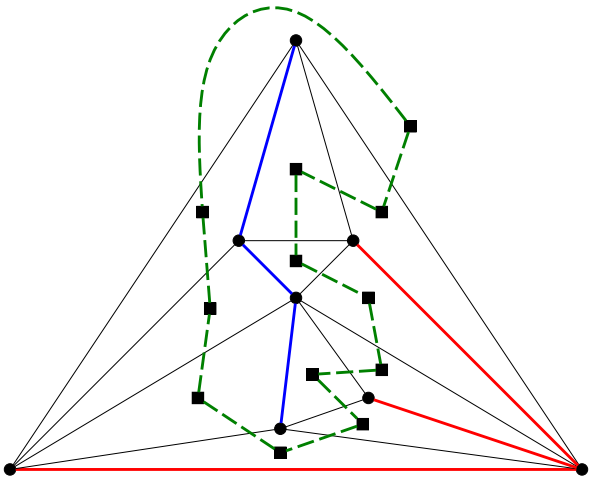


Figure 2.4: The green edges correspond to the edge set of a Hamiltonian cycle that yields the permeating subtrees in the planar triangulation shown by red and blue edges.

is particularly interesting since it is conjectured that the integrality gap of the sub-tour elimination linear programming relaxation for TSP is $\frac{4}{3}$ [20]. The result of Correa et al. shows that the integrality gap is in fact strictly below $\frac{4}{3}$ in cubic graphs. Their approach also yields the following theorem.

Theorem 2.2.7 ([7])**.** *Let G be a Barnette graph. Then G has a TSP tour of length at most* $1.28|V(G)|$.

Their proof uses the fact that the faces of a Barnette graph can be properly colored with 3 colors. They obtain three different 2-factors of a Barnette graph G, by considering the edge set of all the faces colored by a specific color. In the next step, they devise an efficient algorithm that performs local swapping operations on the cycles in each of the 2-factors. Their algorithm outputs a TSP tour, for each of the different 2-factors. By an averaging argument, they show that the minimum of the three has length at most $\frac{23n-22}{18}$, where $n = |V(G)|$.

This yields another approach to prove Barnette's conjecture. Barnette's conjecture holds if all Barnette graphs have a TSP tour of length $|V(G)|$.

2.3 Complexity

2.3.1 The complexity of finding a second Hamiltonian cycle in cubic graphs

Recall the definition of PPA and PPAD (Definitions 1.1.12 and 1.1.10). By the argument in the proof of Theorem 2.1, we have that the problem of Finding a Second Hamiltonian Cycle in a Cubic graph (FSHCC) is in PPA. It is not known whether the problem is in PPAD or not.

Generally speaking, it is not known whether PPA≠PPAD or not. In [32] Papadimitriou explains the possible contrast between these two classes. The following problem will give an idea of this contrast.

Suppose we are given a valid input (G, H_0, w, e) of FSHCC. Consider a vertex x in the exchange graph X (for input (G, H_0, w, e)), that is on the path P containing the vertex of X corresponding to H_0 (standard leaf). Is there a polynomial-time algorithm to determine which direction from x on P would lead to the standard leaf?

For the problems in PPAD, the answer to the above question is indeed yes. However, for FSHCC, it is not yet known whether such a direction is possible to set up. In fact, the problems in PPAD have an algebraic and combinatorial nature, where FSHCC seems more geometric. Thus, it might be that FSHCC is PPA-complete.

2.3.2 An exponential lower bound for the lollipop algorithm

Papadimitriou [32] conjectured that no efficient algorithm for finding a second Hamiltonian cycle exists unless PPA=P. He thought it was unlikely that the lollipop algorithm would solve the problem of finding a second Hamiltonian cycle through a given edge efficiently. This was proved by Krawczyk [28], who came up with a construction showing that the number of steps required by the lollipop algorithm is exponential in the number of vertices, in cubic graphs.

Consider the graph G_1 shown in Figure 2.5.

It would take 12 iterations for the lollipop algorithm to find a second Hamiltonian cycle, if it starts with the Hamiltonian cycle $H_0 = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1)$, $w = 1$, and $e = (1, 10)$ (See Figure 2.6.). Now consider the add-remove walk $W(G_1, H_0, w, e)$ or shortly W_1. In W_1, edge $(6, 5)$ is touched 4 times. First it is removed in iteration 1, then it is added

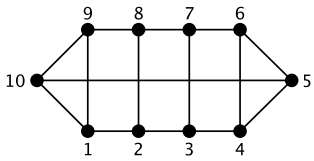
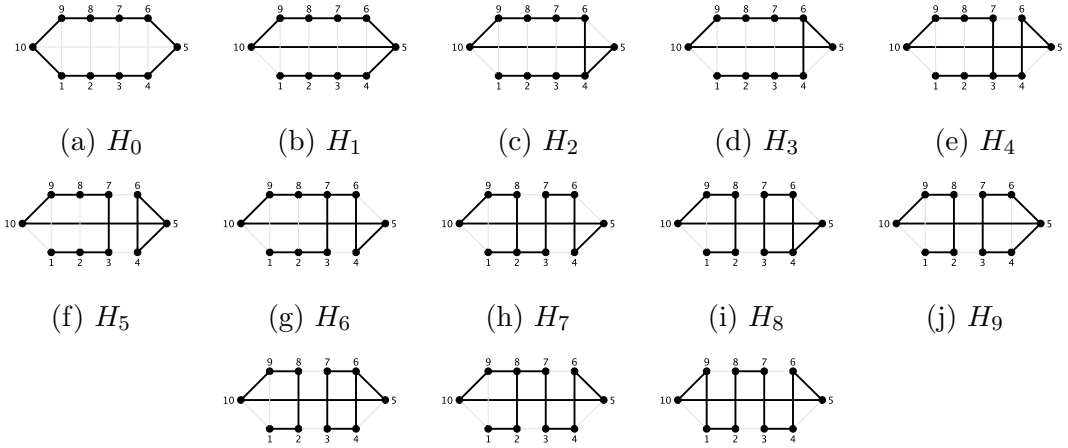


Figure 2.5: Graph G_1



(a) H_0 (b) H_1 (c) H_2 (d) H_3 (e) H_4

(f) H_5 (g) H_6 (h) H_7 (i) H_8 (j) H_9

(k) H_{10} (l) H_{11} (m) H_{12}

Figure 2.6: The iterations of the lollipop algorithm for input (G_1, H_0, w, e)

in iteration 2, again removed in iteration 5, and finally added again in iteration 8. Hence, the algorithm arrives at vertex 6 four times through edge $(6, 5)$. Label 6 with u_1 and edge $(6, 5)$ with h_1. Moreover, label vertices $7, 5$, and 4 with x_1, y_1, and z_1, respectively. The idea of the construction is to build a graph G_i, such that the algorithm for input (G_i, H'_i, w, e), where H'_i is a Hamiltonian cycle of G_i ($H'_1 = H_0$), arrives at vertex u_i through h_i, at least 2^i times, for $i \geq 1$.

To this end, we define a graph B that is obtained from G_1 by removing vertex w. We label the vertices as depicted in Figure 2.7. Graph B would help us construct the desired graph. First, the main property of B is stated in the following observation.

Observation 2.3.1. For $t, t' \in \{x', y', z'\}$, there is exactly one Hamiltonian path in B starting from t and ending at t'.

Graph G_i is obtained from G_{i-1} for $i > 1$ as follows. Delete vertex u_i from the graph

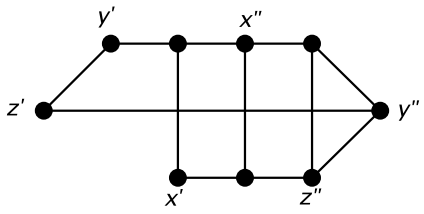


Figure 2.7: Graph B

G_{i-1} to get $G_{i-1} - u_i$. Add graph B, and connect x', y', and z' of B to x_{i-1}, y_{i-1}, and z_{i-1} of $G_{i-1} - u_i$, respectively. The graph obtained by this operation is called G_i. Label vertices x'', y'', and z'' with x_i, y_i, and z_i, respectively. Also label the vertex that is the common neighbour of x_i, y_i, and z_i with u_i. Finally let $h_i = (u_i, y_i)$. The construction of G_2 from G_1 is shown in Figure 2.8.

In addition, H'_i is obtained from H'_{i-1} as follows. Remove u_{i-1} from H'_{i-1}. Suppose that in H'_{i-1}, u_{i-1} is adjacent to $t, t' \in \{x_{i-1}, y_{i-1}, z_{i-1}\}$. To define H'_i, connect both t and t' to the unique vertex of B adjacent to them, namely $s, s' \in \{x', y', z'\}$. By Observation 2.3.1, there is a unique Hamiltonian path from s to s' in B. Extend H'_i to a Hamiltonian cycle for G_i by adding the Hamiltonian ss'-path in B. In Figure 2.8, H'_2 is shown by dashed edges.

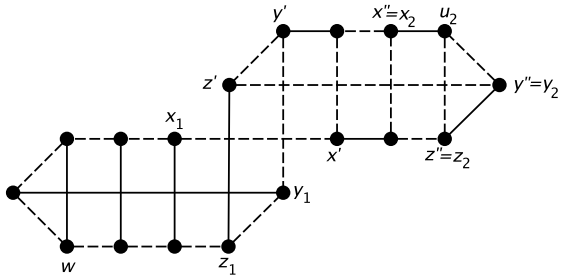


Figure 2.8: Graph G_2

Krawczyk's main result is the following theorem.

Theorem 2.3.2 ([28]). *In the execution of the lollipop algorithm with input* (G_i, H'_i, w, e)*, the algorithm arrives at vertex* u_i *at least* 2^i *times through edge* h_i*.*

This immediately implies that the lollipop algorithm has an exponential worst case running time.

The proof of Theorem 2.3.2 is by induction, and in two steps: (1) edge (y_{i-1}, y') of G_i appears in $W_i = W(G_i, H'_i, w, e)$ at least 2^{i-1} times, and (2) for each time edge (y_{i-1}, y') appears in W_i , edge h_i appears twice in W_i. Step (1) holds because the algorithm would act similarly if we contract the rightmost isomorph of B in G_i. Hence, since by induction h_{i-1} appears 2^{i-1} in W_{i-1}, we have that (y_{i-1}, y') appears 2^{i-1} times in W_i. Step (2) follows directly from Observation 2.3.1. Because we can consider two cases of adding and removing (y_{i-1}, y'). Then we can follow the steps of the algorithm and see that h_i is in fact touched twice.

2.3.3 The complexity of deciding Hamiltonicity of Barnette graphs

Deciding whether a graph admits a Hamiltonian cycle is one of the most well-known NP-complete problems. Takanori, Takao, and Nobuji [1] proved that deciding Hamiltonicity is NP-complete even for cubic planar bipartite graphs. However, the complexity of deciding whether there exists a Hamiltonian cycle for Barnette graphs (HBG) is still unknown.

If the conjecture holds, then the problem would fall into the complexity class TFNP (See Section 1.1.3.), and then it would be unlikely that HBG is NP-complete [31]. More precisely, the following theorem holds.

Theorem 2.3.3. *If Barnette's conjecture holds, then HBG is not NP-complete, unless NP=co-NP.*

Feder and Subi [18] proved that if the conjecture is false, then HBG is NP-complete.

Chapter 3

The lollipop algorithm on planar bipartite graphs

Theorem 2.3.2 gives an infinite class of cubic graphs for which the lollipop algorithm is not efficient. Our goal is to find a subclass of cubic graphs, for which the lollipop algorithm is efficient.

3.1 Why bipartite graphs?

In Section 2.3.1, we presented Krawczyk's graphs where it takes exponential time to obtain a second Hamiltonian cycle using the lollipop algorithm. It is easy to see that Krawczyk's graphs are in fact planar. Looking closely at edges added and removed by the lollipop algorithm on Krawczyk's graphs gives us the feeling that the odd cycles are central for exponentiality of the algorithm on these instances.

Another motivation to consider the lollipop algorithm on cubic bipartite graphs is due to its possible similarities with how the exchange algorithm for finding a second perfect matching behaves on bipartite Eulerian graphs, as illustrated in Theorem 1.2.3. Recall that Theorem 1.2.3 proves that the exchange algorithm terminates after a number of iterations that is linear in number of vertices. In fact, in that context, the pairing function in Eulerian graphs establishes this linear running time performance. Hence, we seek for an analogue of pairing functions in the context of cubic bipartite graphs and the lollipop algorithm.

To this end, we introduce some definitions. Let $H_1, \ldots, H_{n-1}$ be the (w, e)-lollipops considered by the lollipop algorithm for a valid input (G, H_0, w, e).

Definition 3.1.1. Lollipops H_i and H_j are *same based*, for $i, j \in \{1, \ldots, n-1\}$, and $i \neq j$, if $base(H_i) = base(H_j)$.

Definition 3.1.2. Two same based lollipops H_i and H_j are *compatible* if

$$\delta(base(H_i)) \cap C(H_i) = \delta(base(H_j)) \cap C(H_j).$$

Compatibility establishes our analogy with the exchange algorithm in Theorem 1.2.3.

To start, let us prove the following easy lemma, that shows what might be different when dealing with bipartite graphs. From now on, suppose (G, H_0, w, e) is a valid input for the lollipop algorithm, and G is a cubic bipartite graph. Recall

$$W(G, H_0, w, e) = (w = a_0, f_0, r_0, e_1, a_1, f_1, r_1, e_2, \ldots, a_{n-1}, f_{n-1}, r_{n-1}, e_n, a_n = w).$$

We denote $W(G, H_0, w, e)$ shortly by W.

Lemma 3.1.3. *Suppose A and R are the parts of a bipartition of G, such that $w \in A$. Then $a_i \in A$, and $r_i \in R$ for $i \in \{0, \ldots, n-1\}$.*

Proof. By induction on i. For $i = 0$, by definition $a_0 = w \in A$, and since w and r_0 are connected via edge f_0, we have $r_0 \in R$. For the induction step, consider a_i. The induction hypothesis implies that r_{i-1} is in R, since $e_i = (r_{i-1}, a_i)$ we have $a_i \in A$. Similar to the base case we have $r_i \in R$. □

An immediate corollary of Lemma 3.1.3 is the following.

Corollary 3.1.4. *There is no $i \in \{0, \ldots, n\}$ and $j \in \{0, \ldots, n-1\}$ such that $a_i = r_j$.*

The following theorem is our analogue of Theorem 1.2.3.

Theorem 3.1.5. *Suppose that $H_1, \ldots, H_{n-1}$ are the lollipops considered by the lollipop algorithm for input (G, H_0, w, e). If all same based lollipops among $H_1, \ldots, H_{n-1}$ are compatible, then the lollipop algorithm terminates in at most $2|V(G)|$ iterations for input (G, H_0, w, e).*

Proof. Relabel the vertices in W by $y_0, \ldots, y_{2n}$, respecting the order. All we need to do is showing that $n \leq 2|V(G)|$. Suppose $n > |V(G)|$. By the pigeonhole principle, there is a vertex of G that is repeated in $y_0, \ldots, y_{2n}$. Let $j \in \{1, \ldots, 2n\}$ be the smallest possible index for which there exists $i \in \{0, \ldots, j-1\}$ such that $y_i = y_j$.

Now suppose $y_i \in A$. We show that $y_i = w$. If not, then by Corollary 3.1.4, there are $i' < j'$, $i', j' \in \{1, \ldots, n-1\}$, such that $y_i = a_{i'}$ and $y_j = a_{j'}$. But this implies that in iteration $j' - 1$ of the algorithm, the algorithm added edge $e_{j'}$ to arrive at $a_{j'}$. If $e_{j'} = f_{i'}$, then $y_{i+1} = y_{j-1}$ which is a contradiction to the choice of j. Also notice that since $a_{j'}$ is different from w, it has degree two in $H_{i'}, \ldots, H_{j'-1}$. Hence, the two edges in $\delta(a_{j'}) \setminus \{f_{i'}\}$ are already in $H_{j'-1}$, and cannot be added by the algorithm.

On the other hand, if $y_i = y_j = w$, then it must be the case that $i = 0$ and $j = 2n$. Moreover, $y_1, \ldots, y_{2n-1}$ are distinct vertices. Hence $n - 1 \leq |V(G)| - 1$, which is a contradiction.

We are left with $y_i \in R$. We prove the following claim.

Claim 3.1.6. *Suppose j is any index in $\{1, \ldots, n-1\}$, for which there is $i \in \{0, \ldots, j-1\}$, such that $r_i = r_j$. The add-remove walk after j-th iteration is as follows.*

$$(r_j = r_i, f_i, a_i, e_i, \ldots, e_1, r_0, f_0, a_0). \tag{3.1}$$

Proof. We proceed by induction on i. The base is that $r_0 = r_j$ for some $j \in \{1, \ldots, n-1\}$. In iteration j the algorithm adds an edge that is incident to $r_0 = r_j$, and the only candidate edge to add is f_0. This would be the last step of the algorithm, since by adding f_0 the algorithm arrives at w. In this case the add-remove walk after iteration j is

$$(r_0, f_0, a_0 = w), \tag{3.2}$$

as desired.

Now suppose that $i > 0$. Thus, we have $r_i = r_j$ for some $j \in \{1, \ldots, n-1\}$. At iteration j of the algorithm, edge f_j is removed and similar to the base case edge f_i is added. Hence, the algorithm arrives at a_i. Now consider lollipops H_i and H_{j+1}. Since $base(H_i) = a_i$ and $base(H_{j+1}) = a_i$, H_i and H_{j+1} are compatible, hence the algorithm removes e_i to go back to r_{i-1}, which in fact corresponds to r_{j+1} in the add-remove walk. Thus, we can apply induction to show that the add-remove walk after iteration $j + 1$ is

$$(r_{i-1}, f_{i-1}, a_{i-1}, e_{i-1}, \ldots, e_1, r_0, f_0, a_0). \tag{3.3}$$

This concludes the proof. □

Now let j be the smallest index in $\{1, \ldots, n-1\}$ for which there exists $i \in \{0, \ldots, j-1\}$, such that $r_i = r_j$. By Claim 3.1.6, the add-remove walk is

$$(a_0, f_0, r_0, e_1, \ldots, f_j, r_j = r_i, f_i, a_i, e_i, \ldots, e_1, r_0, f_0, a_0).$$

Notice that by the choice of j, $a_0, r_0, \ldots, r_{j-1}$ are distinct vertices of G. Hence, $i < j \leq |V(G)|$. But notice that the algorithm has $j + i$ iterations and $i + j \leq 2|V(G)|$. □

3.2 Why planar bipartite graphs?

Theorem 3.1.5 provides a tool to analyze the lollipop algorithm. The connection between the exchange algorithm for bipartite Eulerian graphs and the lollipop algorithm for cubic bipartite graphs that satisfies the condition stated in Theorem 3.1.5 is now clear. However, not all cubic bipartite graphs satisfy the statement in Theorem 3.1.5. A counterexample is illustrated in Figure 3.1.

Notice that the graph depicted in Figure 3.1 is not planar. In fact, we believe that obtaining two same based lollipops that are not compatible requires some sort of twist in the graph, which motivates us to assert the following conjecture.

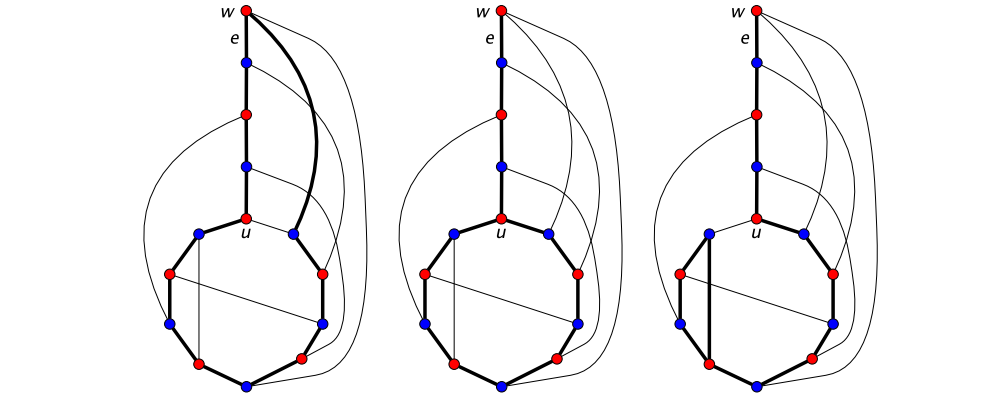
Conjecture 3.2.1. *Let $H_1, \ldots, H_{n-1}$ be the lollipops considered when executing the lollipop algorithm on a valid input (G, H_0, w, e). If G is planar and bipartite, then all same based lollipops in $H_1, \ldots, H_{n-1}$ are compatible.*

Then, by Theorem 3.1.5, this implies the following.

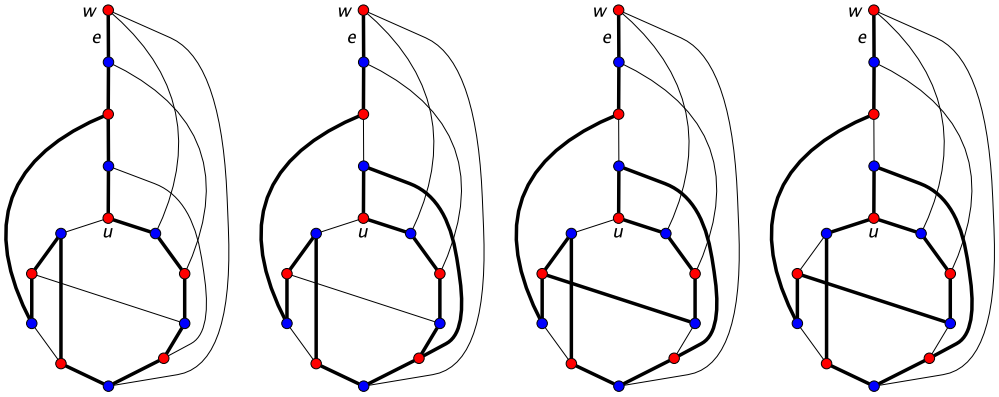
Conjecture 3.2.2. *Given a Hamiltonian cycle H in a cubic planar bipartite graph G and edge e in H, the lollipop algorithm finds a second Hamiltonian cycle through e in time linear in $|V(G)|$.*

Another motivation towards proving Conjecture 3.2.2 would be in the context of Barnette's conjecture. After five decades the conjecture remains (wide) open. The complexity of finding a Hamiltonian cycle in a 3-connected cubic planar bipartite graph is still not known. A corollary of Conjecture 3.2.2 would be that finding a second Hamiltonian cycle in a Barnette graph can be efficiently solved. It would also mean that the exchange graph has polynomially long paths. Hence, if one manages to find a lollipop on a path of the exchange graph, after polynomially many pivotings we would arrive at a Hamiltonian cycle.

Barnette's conjecture has motivated the study of Hamiltonicity in cubic bipartite graphs, and some observations in this context might be useful in proving Conjecture 3.2.2. In particular, recall from Section 2.2.2 the definition of permeating subtrees (Definition 2.2.5). By Theorem 2.2.6 the problem of finding a second Hamiltonian cycle in a cubic planar graph could be now described as finding a second pair of permeating subtrees in a planar triangulation. An analogue of the lollipop algorithm for these instances would consider almost-permeating subtrees as "intermediate vertices". In this case, it might be easier to attack Conjecture 3.2.2.



(a) H_0 (b) H_1 (c) H_2



(d) H_3 (e) H_4 (f) H_5 (g) H_6

Figure 3.1: The iterations of the lollipop algorithm for input (G_1, H_0, w, e). Notice that $base(H_1) = base(H_6)$. However, H_1 and H_6 are not compatible.

3.3 The lollipop algorithm on cubic planar bipartite WH(6)-minor free graphs

This section is dedicated to the proof of Theorem 3.3.1. Let us first recall this theorem.

Theorem 3.3.1. *Given a Hamiltonian cycle H in a cubic planar bipartite WH(6)-minor free graph G, and edge e in H, the lollipop algorithm finds a second Hamiltonian cycle through e in time linear in $|V(G)|$.*

To avoid repeating all the adjectives we will refer to all cubic planar bipartite WH(6)-minor free graphs as class $\mathcal{A}$.

In order to prove Theorem 3.3.1, we will prove the following theorem.

Theorem 3.3.2. *Let $G \in \mathcal{A}$. Let (G, H_0, w, e) be a valid input for the lollipop algorithm, and let $H_1, \ldots, H_{n-1}$ be the (w, e)-lollipops considered by the algorithm. Then any two same based lollipops in $\{H_1, \ldots, H_{n-1}\}$ are compatible.*

Theorem 3.3.1 is an immediate corollary of Theorems 3.1.5 and 3.3.2.

In order to prove Theorem 3.3.2 we will consider plenty of cases. However, before that we will prove some useful lemmas.

3.3.1 Useful lemmas

Let us begin this section with introducing some notation.

Let G be a graph, and P be a path in G. For two vertices u, v in P, $P(u, v)$ is the uv-path in P.

For the remainder of this section let $(G = (V, E), H_0, w, e)$ be a valid input for the lollipop algorithm, where G is a cubic graph. Since (w, e) is fixed for the whole section, we will refer to a (w, e)-lollipop of G by a lollipop.

For a lollipop H, a *jump of H* is an edge $j \in E \setminus E(H)$, such that j has one endpoint in $C(H)$ and one in $P(H)$. A *chord of H* is an edge $j \in E \setminus E(H)$ that has both endpoints in $C(H)$. We call

$$jump(H) := \{j \ : \ j \text{ is a jump of } H\},$$

and

$$chord(H) := \{j \ : \ j \text{ is a chord of } H\}.$$

In Figure 3.2 the bold solid line is a chord of H and the dashed lines are jumps of H.

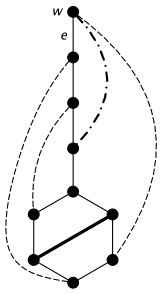


Figure 3.2: A lollipop

Lemma 3.3.3. *For any lollipop H, if G is WH(6)-minor free, then $|jump(H)| \leq 3$.*

Proof. For the sake of contradiction let $e_1, e_2, e_3, e_4 \in jump(H)$. Suppose that $u = base(H)$. Let u_1, u_2, u_3, and u_4 be the endpoints of e_1, e_2, e_3, and e_4 in $C(H)$, respectively. Note that by cubicness of G, $u, u_1, \ldots, u_4$ are distinct. Contracting $P(H) - u$ into a single vertex yields a WH(6) minor, a contradiction. □

Lemma 3.3.4. *For any lollipop H, if $C(H)$ is an even cycle, then $|jump(H)|$ is odd.*

Proof. Let

$$E' = \{e \in E \setminus E(H) \ : \ e \text{ has at least one endpoint in } C(H)\}.$$

The edges in E' cover the vertices in $C(H) \setminus \{base(H)\}$. Also notice that $E' = chord(H) \cup jump(H)$. Each chord e of $C(H)$ covers exactly two vertices in $C(H)$, and each jump covers exactly one vertex in $C(H)$. Thus, the total number of jumps is

$$|V(C(H))| - 1 - 2|chord(H)|,$$

which is an odd number. □

If a graph is planar and WH(6)-minor free, there are many patterns on the edges that are forbidden.

Definition 3.3.5. For a lollipop H, let

$$S = \{v \in C(H) \ : \ v \text{ is an endpoint of an edge in } jump(H)\}.$$

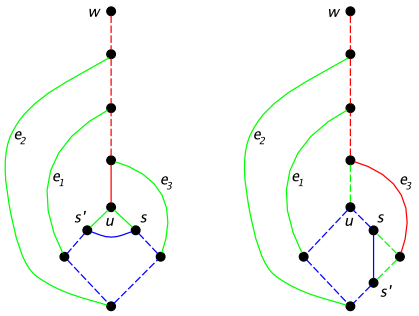
Then $\ell \in chord(H)$ with endpoints s and s' is

- a *short chord* of H, if at least one of the two ss'-paths in $C(H)$ contains exactly one vertex in $S \cup \{base(H)\}$,
- a *non-crossing chord* of H, if at least one of the two ss'-paths in $C(H)$ contains no vertex in $S \cup \{base(H)\}$.

Lemma 3.3.6. *For any lollipop H, if G is WH(6)-minor free and $|jump(H)| \geq 3$, then H has no short chords.*

Proof. By Lemma 3.3.3, since G is WH(6)-minor free, we have $|jump(H)| \leq 3$. Thus, we have $|jump(H)| = 3$. Let e_1, e_2, and e_3 be the three edges in $jump(H)$. Let p_i be the endpoint of e_i in $P(H)$, and c_i be its endpoint in $C(H)$, for $i = 1, 2, 3$. Let $A = \{c_1, c_2, c_3, u\}$, where $u = base(H)$.

For the sake of contradiction, suppose there is a short chord ℓ of H with endpoints s and s'. By definition, at least one of the ss'-paths, namely $P_{ss'}$ in $C(H)$ contains exactly one vertex $x \in A$. We consider the following two cases.



(a) Case 1. (b) Case 2.

Figure 3.3: An example for the two possible cases in Lemma 3.3.6: Dashed edges correspond to paths in the graph. In each of the subcases, blue edges correspond to the edge set of C', red edges correspond to the edge set of P', and green edges correspond to the five disjoint paths that connect P' to C'.

Case 1 ($x = u$)**.** Consider $C' = C(H) - P_{ss'} + \ell$ and $P' = P(H)$. Notice that C' is a cycle. Then $e_1, e_2, e_3, P_{ss'}(s, u)$, and $P_{ss'}(u, s')$, are five disjoint paths from C' to P', contradicting that G is WH(6)-minor free (See Figure 3.3a.).

Case 2 ($x \in \{c_1, c_2, c_3\}$)**.** Without loss of generality suppose $x = c_3$. Consider cycle $C' = C(H) - P_{ss'} + \ell$, and the connected subgraph $P' = (P(H) + e_3) - u$. There are five

disjoint paths $e_1, e_2, e_4, P_{ss'}(s, c_3)$ and $P_{ss'}(c_3, s')$, where $e_4 = \delta(u) \setminus E(C(H))$. This is a contradiction to the fact that G is WH(6)-minor free (See Figure 3.3b.). □

Let f be a jump of a lollipop H, and v be its endpoint in $P(H)$. Jump f of H is the *lowest jump* of H, if no vertex other than v in $P(H)(v, base(H))$ is incident to an edge in $jump(H)$, and f is the *highest jump* of H if no vertex other than v in $P(H)(v, w)$ is incident to an edge in $jump(H)$.

In Figure 3.3a, e_2 is the highest jump and e_3 is the lowest jump of the lollipop in this figure.

Lemma 3.3.7. *Let H be a lollipop with $base(H) = u$ and lowest jump f. Let c and p be the endpoints of f in $C(H)$ and $P(H)$, respectively. If G is planar, then there is a cu-path P in $C(H)$ such that there is no edge in $jump(H)$ with one endpoint being an internal vertex of P.*

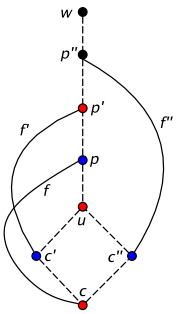


Figure 3.4: The picture for the proof of Lemma 3.3.7.

Proof. Suppose for contradiction that there is no such path P. Let c' and c'' be internal vertices of the two cu-path in $C(H)$ that are incident to jumps f' and f'', respectively. Let p' and p'' be the endpoints on f' and f'' in $P(H)$. Without loss of generality, suppose p' is a vertex of $P(H)(p'', p)$.

Now it is easy to see that G has a subdivision of $K_{3,3}$. The bipartition of the subdivision is $A = \{c', c'', p''\}$ and $B = \{p', u, c\}$ (Figure 3.4). □

In the following lemmas, we will try to analyze the behavior of the lollipop algorithm on some specific parts of a graph.

Definition 3.3.8. A path P in G is *matched inside* if for every edge $e' \in E(G) \setminus E(P)$ if the following hold: e' is incident to an internal vertex of P, then its other endpoint is also an internal vertex of P.

The following lemma shows that if the algorithm enters a matched inside path P in an iteration i through vertex v, it will leave P through vertex v, unless P is a single edge.

Lemma 3.3.9. *Let $P = (u_0, \ldots, u_k)$ be a matched inside path with $k \geq 2$ in G, such that $w \notin V(P)$. Let $f = (u_0, u_1)$ and $g = (u_{k-1}, u_k)$. If at some iteration the algorithm arrives at vertex u_1 by removing edge f, then the algorithm will add back f, before touching g.*

Proof. The algorithm will ultimately arrive at w, and $w \notin V(P)$. However, since f is removed at some iteration i, it implies that the algorithm will enter path P. By the property of P, the only way to leave P after iteration i is through u_0 or u_k. In order to leave through u_0, one has to add f, as desired.

On the other hand, for leaving P through u_k, one has to remove g. But notice that $G - f - g$ is disconnected, so g would never be removed before f is added back. □

Suppose that $H_1, \ldots, H_{n-1}$ are the lollipops considered by the algorithm for (G, H_0, w, e).

Lemma 3.3.10. *Let H_i and H_j be same based lollipops, for some $i < j$, where $i, j \in \{1, \ldots, n-1\}$. If $|E(H_j) \cap jump(H_i)| < 2$, then H_i and H_j are compatible.*

Proof. Suppose H_i and H_j are not compatible. Let f and f' be the edges incident to $base(H_i)$ in $C(H_i)$, and g be the edge incident to $base(H_i)$ in $P(H_i)$. Let x, y, and z, be the endpoints of f, f' and g different from $base(H_i)$, respectively. Since H_i and H_j are not compatible, either f or f' are in $P(H_j)$. Without loss of generality we can assume that f is in $P(H_j)$.

Hence, there is a wx-path in H_j disjoint from y and z. This means there is an edge ℓ in H_j from a vertex in $P(H_i) - z$ to a vertex in $C(H_i) - y$, which is a jump of H_i. Furthermore, in H_j, f' and g are in cycle $C(H_j)$. Thus, there is an edge in H_j that is in $jump(H_i)$, and is different from ℓ. Therefore, $E(H_j)$ contains at least two edges in $jump(H_i)$. □

Figure 3.5 illustrates the proof of Lemma 3.3.10.

Lemma 3.3.11. *Let H_i and H_j be same based lollipops, for some $i < j$, where $i, j \in \{1, \ldots, n-1\}$. Moreover, suppose that any two same based lollipops in $\{H_{i+1}, \ldots, H_{j-1}\}$ are compatible. If there are same based lollipops $H_{i'}$ and $H_{j'}$, with $i' < j'$ and $i', j' \in \{i, \ldots, j\}$, such that $(E(H_{j'}) \setminus E(H_{i'})) \cap jump(H_i) = \emptyset$, then H_i and H_j are compatible.*

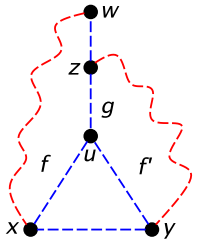


Figure 3.5: Dashed lines correspond to paths and $u = base(H_i) = base(H_j)$. The edge set of H_i is colored blue. Red edges correspond to the paths that must exist in H_j, for H_i and H_j not to be compatible.

Proof. We can assume without loss of generality that no lollipops in $\{H_{i+1}, \ldots, H_{j-1}\}$ has the same base as H_i. Let $j - i = m$, and $j' - i' = k$. If $k = m$, we have $H_i = H_{i'}$, and $H_j = H_{j'}$, and by Lemma 3.3.10, H_i and H_j are compatible.

Thus, we may assume $k < m$. We may also assume that either $i' = i$ or $j' = j$. However, if $i' = i$ and $j' \neq j$, then $base(H_i) = base(H_{i'}) = base(H_{j'})$, which is a contradiction. Similarly, we can assume $j' \neq j$. We will show that $E(H_j) \setminus E(H_i) = E(H_{j'}) \setminus E(H_{i'})$. This will conclude the proof since $E(H_{j'}) \setminus E(H_{i'})$ contains no edge in $jump(H_i)$. By Lemma 3.3.10, H_i and H_j are compatible.

Let $W = W_{H_i,H_{i'}}(G, H_0, w, e)$. We proceed by induction on the length of W. Since $k < m$, W has length at least 2.

For the base case, suppose W has length 2. This means that $i' = i+1$. In particular, in iteration i the algorithm removes edge f_i, and adds edge $e_{i'}$ to obtain $H_{i'}$. Since $H_{i'}$ and $H_{j'}$ are compatible by assumption, the algorithm removes $e_{i'}$ and adds f_i in iteration j' to obtain lollipop H_j. In other words $H_j = H_{j'} - e_{i'} + f_i$. Hence, $E(H_j) \setminus E(H_i) = E(H_{j'}) \setminus E(H_{i'})$.

Now suppose W has length 2ℓ for some integer $\ell > 1$. We have $i' = i+\ell$. Lollipop $H_{i'}$ is obtained from lollipop $H_{i'-1}$ by removing edge $f_{i'-1}$ and adding edge $e_{i'}$. Since $H_{i'}$ and $H_{j'}$ are compatible, the algorithm removes $e_{i'}$ and adds $f_{i'-1}$ in iteration j' to obtain lollipop $H_{j'+1}$. Observe that $base(H_{i'-1}) = base(H_{j'+1})$. Notice that $W' = W_{H_i,H_{i'-1}}(G, H_0, w, e)$ has length $2(\ell - 1)$. Thus, by the induction hypothesis, $E(H_j) \setminus E(H_i) = E(H_{j'+1}) \setminus E(H_{i'-1}) = E(H_{j'}) \setminus E(H_{i'})$. □

The following lemma is an immediate corollary of Lemmas 3.3.10 and 3.3.11.

Lemma 3.3.12. *Let H_i and H_j be same based lollipops, for some $i < j$, where $i, j \in \{1, \ldots, n-1\}$. Let $P = (u_0, u_1, \ldots, u_k)$ be a path in G with $|V(P)| \geq 3$. If the following properties hold, then H_i and H_j are compatible.*

1. *P is matched inside;*
2. *path P completely lies in $P(H_i)$ or $C(H_i)$;*
3. *during the execution of the lollipop algorithm when going from iteration i to j, $f = (u_0, u_1)$ is removed to arrive at vertex u_1;*
4. *all same based lollipops in $\{H_{i+1}, \ldots, H_{j-1}\}$ are compatible.*

Proof. Let $i' \in \{i, \ldots, j\}$ be the iteration in which the algorithm removes f to arrive at u_1. We have $base(H_{i'}) = u_0$. By Lemma 3.3.9, in some iteration $j' \in \{i'+1, \ldots, j\}$ the algorithm adds f to obtain lollipop $H_{j'}$. In addition, $base(H_{j'}) = u_0$. Lollipops $H_{j'}$ and $H_{i'}$ are same based. We have $E(H_{j'}) \setminus E(H_{i'}) \subseteq E(P)$ and $E(P)$ contains no edge in $jump(H_i)$ from properties (1) and (2). The result follows from Theorem 3.3.11. □

The following lemmas would help us to analyze the algorithm easier. For the following lemmas (Lemmas 3.3.13, 3.3.14, 3.3.15, 3.3.16) suppose that G is WH(6)-minor free. Also let H be a lollipop with $base(H) = u$ and with exactly three jumps, namely f, g, and h. Let c, c', and c'' be the endpoints of f, g, and h in $C(H)$, respectively. Also let p, p', p'' be the endpoints of f, g, and h in $P(H)$, respectively. The four vertices u, c, c', and c'' partition $C(H)$ to a uc''-path P_1, a $c''c$-path P_2, a cc'-path P_3, and a $c'u$-path P_4.

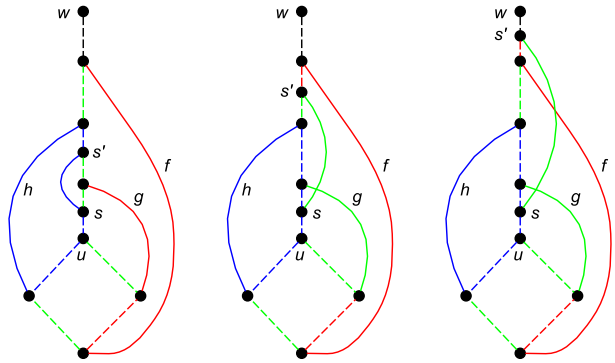
Lemma 3.3.13. *If (1) f is the highest jump of H, (2) g is the lowest jump of H, and (3) each uc-path in $C(H)$ contains exactly one of c' and c'' as internal vertices, then $P(H)(p', u)$ is matched inside.*

Proof. Suppose not. Let m be an edge with exactly one endpoint s being an internal vertex of $P(H)(p', u)$. Consider these cases for the other endpoint of m, namely s'.

Case 1 (s' in $P(H)(p', p'')$). Consider cycle $C' = h + P(H)(p'', s') + m + P(H)(s, u) + P_1$ and path $P' = f + P_3 + g$. Paths $P_2, P_4, P(H)(s, p'), P(H)(p', s')$, and $P(H)(p'', p)$ are five disjoint paths from P' to C'. However, G is WH(6)-minor free. This is a contradiction (Figure 3.6a).

Case 2 (s' in $P(H)(p'', p)$). Consider cycle $C' = h + P(H)(p'', u) + P_1$, and path $P' = P(H)(s', p) + f + P_3$. Paths P_2, P_4, g, m, and $P(H)(p'', s')$ are five disjoint paths from P' to C', which is a contradiction as G is WH(6)-minor free (Figure 3.6b).

Case 3 (s' in $P(H)(p,w)$). Consider cycle $C' = h + P(H)(p'',u) + P_1$, and path $P' = P(H)(p,s') + f + P_3$. Paths P_2, P_4, g, m, and $P(H)(p,p'')$ are five disjoint paths from P' to C'. This is a contradiction since G is WH(6)-minor free (Figure 3.6c). □



(a) Case 1 (b) Case 2 (c) Case 3

Figure 3.6: The three cases in the proof of Lemma 3.3.13.

Lemma 3.3.14. *If g is the highest jump of H and h is the lowest jump of H, then there is no edge in $E \setminus E(H)$ with one endpoint in $P(H)(u,p'')$ and one in $P(H)(p'',p)$.*

Proof. Suppose for contradiction that there is an edge $\ell = (s,s') \in E \setminus E(H)$ with s in $P(H)(u,p'')$ and s' in $P(H)(p'',p)$. Then consider cycle $C' = g + P_4 + P(H)(u,s) + \ell + P(H)(s',p')$ and path $P' = h + P_2$. Now $f, P_1, P_3, P(H)(p'',s')$, and $P(H)(p'',s)$ are five disjoint paths from P' to C'. This is a contradiction to the fact that G is WH(6)-minor free (See Figure 3.7.). □

Recall the definition of non-crossing chords in Definition 3.3.5.

Lemma 3.3.15. *Lollipop H has at most one chord that is not non-crossing.*

Proof. Assume for contradiction that H has at least two chords that are not non-crossing, namely $\ell = (s,s')$ and $\ell' = (t,t')$. By Lemma 3.3.6, ℓ and ℓ' are not short chords. Hence, without loss of generality we can assume that s is in P_1 and s' is in P_3. We have to consider the following cases.

Case 1 (t in P_1 and t' in P_3). We consider two subcases: (i) if h is the lowest jump, consider cycle $C' = P_1 + h + P(H)(p'',u)$ and path $P' = g + P_3$. Observe that P_2, P_4, ℓ, ℓ',

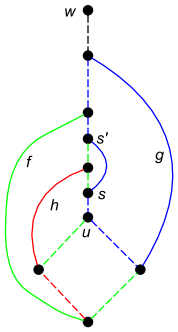
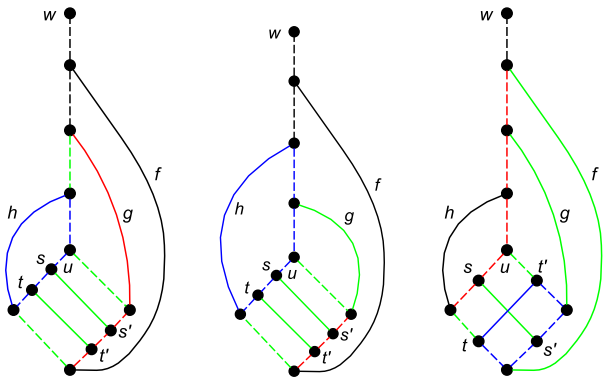


Figure 3.7: The picture in the proof of Lemma 3.3.14.

and $P(H)(p'', p')$ are five disjoint paths from P' to C' (Figure 3.8a), and (ii) if h is not the lowest jump, consider cycle $C' = P_1 + h + P(H)(p'', u)$, path $P' = P_3$. Then P_2, P_4, ℓ, ℓ', and ℓ'', where ℓ'' is the lowest jump of H, are five disjoint paths from P' to C' (Figure 3.8b). In both cases we reach a contradiction.



(a) Case 1i (b) Case 1ii (c) Case 2

Figure 3.8: The cases in the proof of Lemma 3.3.15.

Case 2 (t in P_2 and t' in P_4). Without loss of generality suppose that f is the highest jump of H. Consider cycle $C' = P_2(c, t) + \ell' + P_4(t', c') + P_3$, and path $P(H)(p, u) + P_1$. Then $f, g, \ell, P_4(t', u)$, and $P_2(t, c'')$ are five disjoint paths from P' to C', which is a contradiction (Figure 3.8c). □

Lemma 3.3.16. *If H has one chord $\ell = (s, s')$ that is not non-crossing, then each of the six paths created by u, c, c', c'', s, s' on $C(H)$ are matched inside.*

Proof. Suppose without loss of generality that s is in P_1. By Lemma 3.3.6, s' is in P_3. First notice that P_2 and P_4 are matched inside. This immediately follows by Lemma 3.3.6 and 3.3.15. Thus, we need to show that $P_1(u, s), P_1(s, c''), P_3(c, s')$, and $P_3(s', c')$ are matched inside.

Suppose for contradiction that $P_1(u, s)$ is not matched inside. The proof for the other paths is similar. So suppose there is an edge $\ell' = (t, t') \in E \setminus E(H)$ with one endpoint t in internal vertices of $P_1(u, s)$ and one endpoint t' not in $P_1(u, s)$. By Lemma 3.3.6, t' is not in P_2 or P_4. Also by Lemma 3.3.15, t' is not in P_3. Thus, we may assume that t' is in $P_1(s, c'')$. Let ℓ'' be the edge incident to u in $P(H)$.

Consider cycle $C' = P_1(u, s) + \ell + P_3(s', c') + P_4$, and connected subgraph $P' = (P(H) - u) + f + P_2 + P_1(c'', t')$. Observe that $P_1(s, t'), \ell', P_3(s', c), g$, and ℓ'' are five disjoint paths from P' to C'. This is a contradiction (Figure 3.9). □

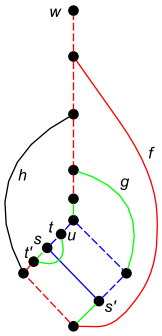


Figure 3.9: The picture in the proof of Lemma 3.3.16.

3.3.2 Proof of Theorem 3.3.2

Let us recall the theorem once again.

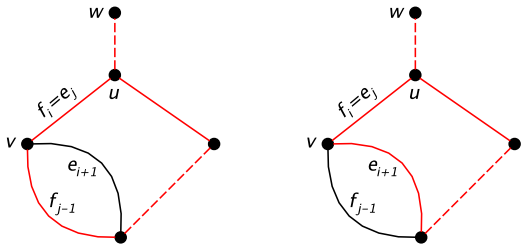
Theorem 3.3.2. *Let $G \in \mathcal{A}$. Let (G, H_0, w, e) be a valid input for the lollipop algorithm, let $H_1, \ldots, H_{n-1}$ be the (w, e)-lollipops considered by the algorithm. Then any two same based lollipops in $\{H_1, \ldots, H_{n-1}\}$ are compatible.*

Let H_i and H_j be same based lollipops, for some $i < j$, where $i, j \in \{1, \ldots, n-1\}$. We will proceed by induction on $m = j - i$.

The base case is $m = 2$, since it is impossible to obtain two same based lollipops in one iteration of the lollipop algorithm. Let $base(H_i) = base(H_j) = u$. In this case, $W' = W_{H_i,H_j}(G, H_0, w, e)$ has four edges. The edges are $f_i, e_{i+1}, f_{j-1}, e_j$ in this order. The algorithm removes f_i, f_{j-1} and adds e_{i+1}, e_j. Notice that $e_j = f_i$. Otherwise, the algorithm adds edge e_{i+1} incident to u. But, e_{i+1} was already an edge in H_{i+1}. Now since $e_j = f_i$, it must be the case that e_{i+1} and f_{j-1} are parallel edges.

If $e_{i+1} \in jump(H_i)$, then let y be its endpoint on $P(H_i)$. Since e_{i+1} and f_{j-1} are parallel edges, we have $f_{j-1} \in jump(H_i)$ as well. This means that y has degree one in $P(H_i)$. Hence $y = w$. However, this is a contradiction, for the algorithm would terminate upon arriving at w.

Now observe that $C(H_j) = C(H_i) - f_{j-1} + e_{i+1}$. This implies that H_i and H_j are compatible (See Figure 3.10.).



(a) H_i (b) H_{i+2}

Figure 3.10: The base case, where $j = i + 2$

For the inductive step, we will consider many cases. First notice that the induction hypothesis is that for same based lollipops $H_{i'}$ and $H_{j'}$, where $j' > i'$, $i', j' \in \{1, \ldots, n-1\}$, if $j' - i' < m$, then $H_{i'}$ and $H_{j'}$ are compatible. The induction hypothesis provides the ingredients for applying Lemmas 3.3.11 and 3.3.12.

Let f and g be the edges incident to $u = base(H_i)$ in $C(H_i)$, and h be the edge incident to u in $P(H_i)$. Let x, y, and z be the endpoints of f, g, and h that are different from u, respectively. Without loss of generality, suppose the algorithm adds edge f in iteration $i - 1$.

Suppose for the sake of contradiction that H_i and H_j are not compatible. Moreover, assume that the lollipops that the algorithm considers from iteration i to $j - 2$

are $H_{i+1}, \ldots, H_{j-1}$. Without loss of generality, we can assume that u is not the base of any lollipop H_k, $k \in \{i+1, \ldots, j-1\}$. Let $W = W_{H_i,H_j}(G, H_0, w, e)$.

By Lemma 3.3.10, we have $|jump(H_i)| \geq 2$. In addition, by Lemma 3.3.4, we have $|jump(H_i)| \geq 3$. Finally, by Lemma 3.3.3 we have $|jump(H_i)| \leq 3$. Thus, $jump(H_i)$ consists of three edges e_1, e_2, and e_3.

Let p_1, p_2, and p_3 be the endpoints of e_1, e_2, and e_3 in $P(H_i)$, respectively. Also let c_1, c_2, and c_3, be the endpoints of e_1, e_2, and e_3 in $C(H_i)$, respectively. Without loss of generality, suppose c_1, c_2, c_3, happen in this order, when walking from x to y on $C(H_i) - u$. The four vertices u, c_1, c_2, and c_3 partition $C(H)$ into a uc_1-path P_1, a c_1c_2-path P_2, a c_2c_3-path P_3, and a c_3u-path P_4 (See Figure 3.11.).

By Lemma 3.3.7, since G is planar, e_2 cannot be the lowest jump of H_i. Depending on the position of p_1, p_2, and p_3 on $P(H_i)$ we consider the following cases.

Case 1. (e_2 is the highest jump, e_3 is the lowest jump of H_i)

Let Q_1, Q_2, Q_3, and Q_4 be the up_3-path, p_3p_1-path, p_1p_2-path, and p_2w-path on $P(H_i)$, respectively (Figure 3.11). Consider the cycle $C = Q_1 + P_4 + e_3$. Since G is bipartite, C is an even cycle. We have $w \notin V(C)$. Thus, the edges in $E \setminus E(H_i)$ that have both endpoints in $V(C)$ form a matching of G. Hence, an even number of vertices in C are incident to such edges. In addition, u is not incident to any of the edges in $E \setminus E(H_i)$. Therefore, there is an edge k with endpoint s in $V(C) \setminus \{u, p_3, c_3\}$ and endpoint s' not in $V(C)$.

Notice that by Lemma 3.3.13, Q_1 is matched inside. Hence, the only possibility is that s is in P_4. By Lemma 3.3.6, k is not a short chord of H_i. This means that s' is in P_2.

Consider the following claim.

Claim 3.3.17. *Paths $P_1, Q_2, P_4(c_3, s)$, and $P_2(s', c_2)$ are single edges.*

Before proving Claim 3.3.17, let us see how it would lead us to conclude Case 1.

Observe that W is a closed walk that starts from u and ends at u. Recall that the algorithm removes edge g in iteration i. By Claim 3.3.17, $g = P_1$.

Following the steps of the algorithm, after removing $g = P_1$ the algorithm alternately adds and removes $e_1, Q_2, e_3, P_4(c_3, s), k, P_2(s', c_2)$, and finally adds e_2 to arrive at vertex p_2 and lollipop H_{i+4} (Figure 3.12b).

This means that the algorithm enters $Q_3 + Q_4$ in iteration $i + 3$. However, since the algorithm arrives at u in iteration $j - 1$, it has to leave $Q_3 + Q_4$ before iteration j.

We now need to make a quick observation.

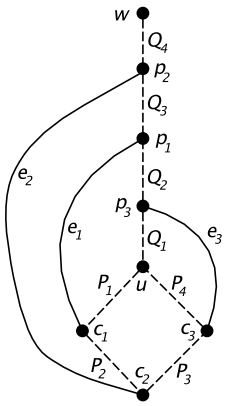


Figure 3.11: Case 1: In all the figures in this section dashed edges correspond to paths in the graph.

Observation 3.3.18. There is no edge in $E \setminus E(H_i)$ that is different from e_1 and e_2, with one endpoint in $Q_3 + Q_4$ and one not in $Q_3 + Q_4$.

The proof of Observation 3.3.18 is deferred to Section 3.3.3.

By Observation 3.3.18, the only way W can leave $Q_3 + Q_4$ is either through p_1 or p_2. Since p_1 and p_2 are in the same part of the bipartition as w, by Lemma 3.1.3, the algorithm leaves $Q_3 + Q_4$ by removing e_1 or e_2.

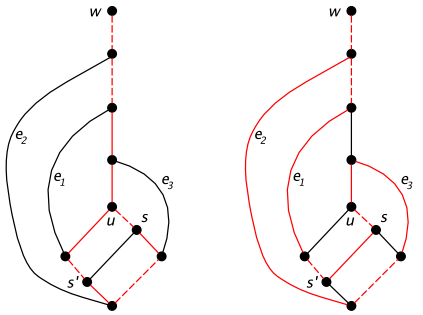
In the latter case, let $H_{k'}$, for some $i + 4 < k' < j$, be the lollipop considered by the algorithm before removing e_2. Note that $base(H_{k'}) = p_2$, and in fact H_{i+4} and $H_{k'}$ are same based lollipops. Since the algorithm does not leave $Q_3 + Q_4$ between iterations $i + 4$ and k', $E(H_{i+4}) \setminus E(H_{k'}) \cap jump(H_i) = \emptyset$. This, plus the induction hypothesis allows us to conclude that H_i and H_j are compatible using Lemma 3.3.11. This is a contradiction to the original assumption.

On the other hand, if the algorithm leaves $Q_3 + Q_4$ by removing e_1, then the algorithm adds $g = P_1$ to return to u. However, in this case $C(H_j) = g + P_4(u, s) + m + P_2(s', c_1)$, which contains both f and g. Hence, H_i and H_j are compatible.

It remains to prove Claim 3.3.17. Before that we have to make the following observation. To preserve the coherence of the proof, we will prove this observation later in Section 3.3.3.

Observation 3.3.19. The following properties hold:

1. P_1 is matched inside;



(a) H_i (b) H_{i+4}

Figure 3.12: If Claim 3.3.17 holds: The red edges in (a) and (b) correspond to the edge set of H_i and H_{i+4}, respectively.

2. Q_2 is matched inside;
3. $P_4(c_3, s)$ is matched inside; and
4. $P_2(s', c_2)$ is matched inside.

Now to prove Claim 3.3.17 we just need to show for each

$$P \in \{P_1, Q_2, P_4(c_3, s), P_2(s', c_2)\},$$

such that $P = (u_0, u_1, \ldots, u_t)$, the algorithm removes edge (u_0, u_1) to arrive at u_1. This completes the proof of Claim 3.3.17, because: (1) P is matched inside, (2) P completely lies in $P(H_i)$ or in $C(H_i)$, (3) edge (u_0, u_1) is removed to arrive at u_1 between iterations i and j, and (4) by induction hypothesis any two same based lollipops in $\{H_{i+1}, \ldots, H_{j-1}\}$ are compatible. Then if $|V(P)| \geq 3$, properties (1)-(4) provide the ingredients of Lemma 3.3.12, proving that H_i and H_j are compatible. This is a contradiction to the original assumption.

Proof of Claim 3.3.17. Let $P \in \{P_1, Q_2, P_4(c_3, s), P_2(s', c_2)\}$, and $P = (u_0, u_1, \ldots, u_t)$.

If $P = P_1$, then $u_0 = u$ and the algorithm removes $g = (u_0, u_1)$ in iteration i to arrive at u_1. So P_1 is an edge.

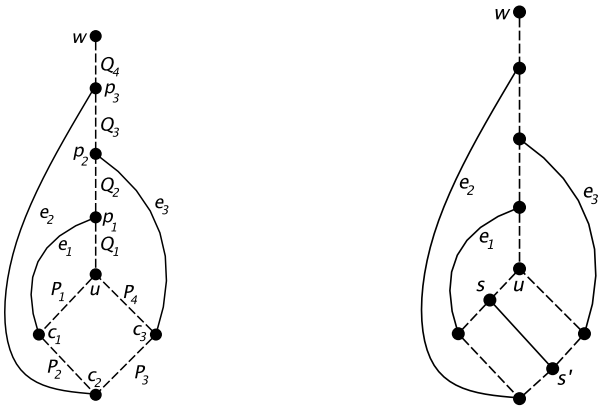
If $P = Q_2$, then $u_0 = p_1$. Considering that P_1 is an edge, the algorithm removes (u_0, u_1) in iteration $i + 1$ to arrive at u_1. Thus, Q_2 is an edge.

Taking the fact that Q_2 is an edge into account, if $P = P_4(c_3, s)$, the algorithm removes (u_0, u_1), where $u_0 = c_3$, to arrive at u_1 in iteration $i + 2$. Hence, $P_4(c_3, s)$ is an edge.

Finally, if $P = P_2(s', c_2)$, the algorithm removes $P_4(c_3, s)$ and adds k. Then the algorithm removes edge (u_0, u_1), where $u_0 = s'$, in iteration $i + 3$. This implies that $P_4(c_3, s)$ is an edge. □

Case 2. (e_2 is the highest jump, e_1 is the lowest jump of H_i)

Let Q_1, Q_2, Q_3, and Q_4 be the up_1-path, p_1p_3-path, p_3p_2-path, and the p_2w-path in $P(H_i)$ (See Figure 3.13a).



(a) Case 2 (b) Edge k on H_i

Figure 3.13

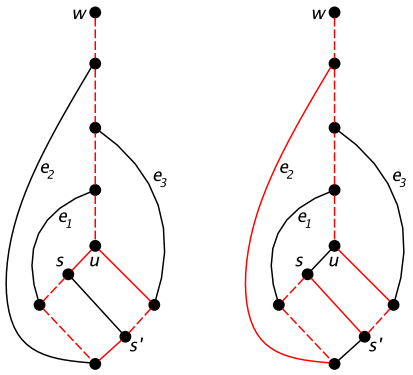
In cycle $C = P_1 + e_1 + Q_1$ there is a vertex $s \notin \{p_1, c_1, u\}$ that is incident to an edge $k \in E \setminus E(H_i)$, such that the other endpoint s' of k is not in $V(C)$. Otherwise C is an odd cycle, which is a contradiction to bipartiteness of G.

Vertex s is either in P_1 or in Q_1. However, by Lemma 3.3.13, Q_1 is matched inside. Thus, s is in P_1. Moreover, by Lemma 3.3.6, s' is in P_3 (Figure 3.13b).

We prove the following claim later.

Claim 3.3.20. *Paths $P_1(u, s), P_3(s', c_2)$, and P_4 are single edges.*

If Claim 3.3.20 holds the rest of the proof works as follows. The algorithm removes $g = P_1(u, s)$ and then alternately adds and removes $k, P_3(s', c_2), e_2$. After adding e_2 we obtain lollipop H_{i+2}. Observe that $base(H_{i+2}) = p_2$ (Figure 3.14b). Hence, at this iteration the algorithm enters $Q_3 + Q_4$. Similar to Case 1, we have to show that the only way W



(a) H_i (b) H_{i+2}

Figure 3.14: If Claim 3.3.20 holds: The red edges in (a) and (b) correspond to the edge set of H_i and H_{i+2}, respectively.

can leave $Q_3 + Q_4$ is by adding e_3 or removing e_2.

Observation 3.3.21. There is no edge in $E \setminus E(H_i)$ that is different from e_2 and e_3, with one endpoint in $Q_3 + Q_4$, and one not in $Q_3 + Q_4$.

We prove Observation 3.3.21 in Section 3.3.3. By Observation 3.3.21, the algorithm leaves $Q_3 + Q_4$ either by removing e_2 or by adding e_3. In the first case, let $H_{k'}$, for some $i + 2 < k' < j$, be the lollipop before removing e_2. We have $base(H_{k'}) = p_2$. Note that H_{i+2} and $H_{k'}$ are same based lollipops. Moreover, by Lemma 3.3.21, $(E(H_{k'}) \setminus E(H_{i+2})) \cap jump(H_i) = \emptyset$. Therefore, by Lemma 3.3.11, H_i and H_j are compatible.

Otherwise, if the algorithm leaves $Q_3 + Q_4$ by adding e_3, in the next iteration the algorithm removes edge P_4 to arrive at u. In this case W is an odd closed walk, since the algorithm leaves u by removing an edge, then alternately adds and removes edges, until finally removes an edge to arrive at u. This is a contradiction to bipartiteness of G.

Again in order to prove Claim 3.3.20 we need to show that the paths $P_1(u, s), P_3(s', c_2)$, and P_4 are matched inside. The proof of this observation is provided in Section 3.3.3.

Observation 3.3.22. We have the following properties:

1. $P_1(u, s)$ is matched inside;

2. $P_3(s', c_2)$ is matched inside; and

3. P_4 is matched inside.

Similar to Case 1, we just show that for each $P \in \{P_1(u, s), P_3(s', c_2), P_4\}$, where $P = (u_0, \ldots, u_t)$, for some t, the algorithm removes edge (u_0, u_1) to arrive at u_1 between iterations i and j in the algorithm.

Proof of Claim 3.3.20. If $P = P_1(u, s)$, then $u_0 = u$ and $g = (u_0, u_1)$. Hence, $P_1(u, s)$ is an edge. If $P = P_3(s', c_2)$, since $P_1(u, s)$ is an edge, the algorithm removes $P_1(u, s)$ to leave vertex u. Then it adds edge k, and then removes (u_0, u_1), where $u_0 = s'$, to arrive at u_1. Thus, $P_3(s', c_2)$ is an edge.

Finally, suppose $P = P_4$. We know that $P_1(u, s)$ and $P_3(s', c_2)$ are single edges. So the algorithm leaves u by removing $P_1(u, s)$, adding k, removing $P_3(s', c_2)$, and adding e_2 to obtain lollipop H_{i+2}. At this iteration, the algorithm enters $Q_3 + Q_4$. Observation 3.3.21 ensures that W leaves $Q_3 + Q_4$ by adding e_3 or removing e_2. The latter case would imply that H_i and H_j are compatible, which is a contradiction. So the algorithm adds e_3, and removes edge (u_0, u_1), where $u_0 = c_3$. Therefore, P_4 is an edge. □

Case 3. (e_1 is the highest jump, e_3 is the lowest jump of H_i)

Let Q_1, Q_2, Q_3, and Q_4 be the up_3-path, p_3p_2-path, p_2p_1-path, and p_1w-path on $P(H_i)$, respectively (Figure 3.15). Consider the cycle $C = Q_1 + P_4 + e_3$. Since G is bipartite, C has an even number of vertices. Furthermore, w is not in C, hence there is an even number of vertices in C that are incident to edges in $E \setminus E(H_i)$ with both endpoints in C, but u is not incident to an edge in $E \setminus E(H_i)$. Thus, there is an edge $k = (s, s') \in E \setminus E(H_i)$, such that $s \in V(C) \setminus \{u, p_3, c_3\}$ and $s' \notin V(C)$.

Vertex s' is either in Q_1 or in P_4. Notice that if s' is in Q_1, then by Lemma 3.3.14, s' cannot be in Q_2. We consider the following cases.

***Case 3i.* s in Q_1 and s' in Q_3.**

In this case, we have the following claim.

Claim 3.3.23. *Paths $P_1, Q_1(s, p_3)$, and P_4 are single edges.*

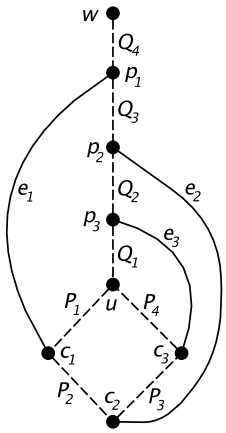


Figure 3.15: Case 3.

Let us first show how to conclude Case 3i using Claim 3.3.23. The algorithm removes $g = P_1$ in iteration i, and then adds edge e_1, to obtain lollipop H_{i+1}. Observe that $base(H_{i+1}) = p_1$ (Figure 3.16b). In iteration $i+1$, the algorithm will remove an edge in $Q_3(p_1, s') + Q_4$. Hence, the algorithm enters $Q_3(p_1, s') + Q_4$.

Observation 3.3.24. There is no edge in $E \setminus E(H_i)$, that is different from e_1 and k, with one endpoint in $Q_3(p_1, s') + Q_4$, and one not in $Q_3(p_1, s') + Q_4$.

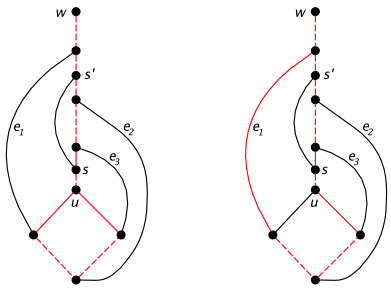
We prove this observation later in Section 3.3.3. By Observation 3.3.24, the only way to leave $Q_3(p_1, s') + Q_4$ is either by adding k or by removing e_1. The latter will give us a contradiction, since in this case the algorithm adds $g = P_1$ back. Then $C(H_j) = C(H_i)$, and hence H_i and H_j are compatible, contradicting the original assumption.

Thus, we may assume that the algorithm adds edge k. Then, the algorithm alternately removes and adds $Q_1(s, p_3), e_3, P_4$ to arrive at u in iteration $j-1$. But since W leaves u in the i-th iteration by removing an edge (P_1) and arrives at u in iteration $j-1$ by again removing an edge (P_4), W is an odd closed walk. This is a contradiction to bipartiteness of G.

For proving Claim 3.3.23 we need the following observation.

Observation 3.3.25. The following properties hold:

1. P_1 is matched inside;
2. $Q_1(s, p_3)$ is matched inside; and



(a) H_i (b) H_{i+1}

Figure 3.16: If Claim 3.3.23 holds, G has a subgraph as depicted: the red edges in (a) and (b) are the edge set of H_i and H_{i+1}.

3. P_4 is matched inside.

The proof of Observation 3.3.23 is provided in Section 3.3.3. The proof of Claim 3.3.23 is now similar to the other cases. The only thing one needs to show is that for $P \in \{P_1, Q_1(s, p_3), P_4\}$, where $P = (u_0, \ldots, u_t)$ for some integer $t \geq 1$, the algorithm removes edge (u_0, u_1) to arrive at vertex u_1, between iterations i and j.

Case 3ii. **s in Q_1 and s' in Q_4.**

We have the following claim in this case.

Claim 3.3.26. *Paths P_1, Q_3, P_3, and $Q_1(p_3, s)$ are single edges.*

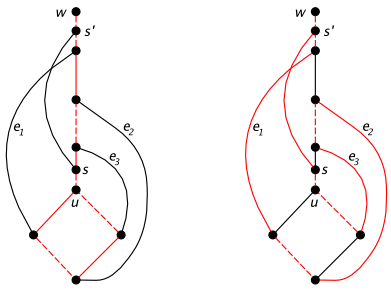
If Claim 3.3.26 holds, then after iteration i, the algorithm alternately removes and adds $P_1 = g, e_1, Q_3, e_2, P_3, e_3, Q_3(p_3, s)$, and k. The lollipop obtained after adding k is H_{i+4}. Notice that $base(H_{i+4}) = s'$.

Now we have to make a quick observation.

Observation 3.3.27. Let $\ell \in E \setminus E(H_i)$, and $\ell \neq k, e_1$. If ℓ has one endpoint in Q_4, the other endpoint of ℓ is also in Q_4.

We will prove Observation 3.3.27 in the next section.

Consider W. Since W ends at u, the algorithm has to leave path Q_4 before iteration j. By Observation 3.3.27, the only possibilities are by removing e_1 or by removing k. The



(a) H_i (b) H_{i+4}

Figure 3.17: If Claim 3.3.26 holds: the red edges in (a) and (b) are the edge set of H_i and H_{i+4}, respectively.

latter case results in a contradiction. In particular, let $H_{k'}$, $i+4 < k' < j$, be the lollipop right before removing k and leaving Q_4. Since $base(H_{k'}) = s'$, H_{i+4} and $H_{k'}$ are same based. Moreover, $E(H_{k'}) \setminus E(H_{i+4}) \cap jump(H_i) = \emptyset$. This implies that H_i and H_j are compatible by Lemma 3.3.11. This is a contradiction.

On the other hand, if the algorithm removes e_1, then it adds $P_1 = g$ back to obtain lollipop H_j. In this case, again both f and g are in $C(H_j)$. Therefore, H_i and H_j are compatible, which is a contradiction.

Proving Claim 3.3.26 is straightforward after next observation.

Observation 3.3.28. We have the following properties:

1. P_1 is matched inside;
2. Q_3 is matched inside;
3. P_3 is matched inside; and
4. $Q_1(p_3, s)$ is matched inside.

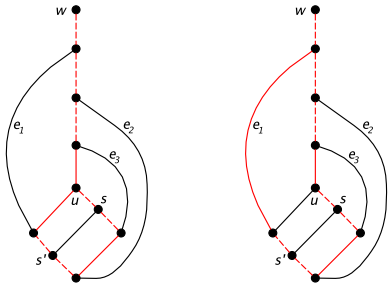
We will prove this observation in Section 3.3.3.

Case 3iii. **s in P_4 and s' in P_2.**

In this case, we will use the following claim.

Claim 3.3.29. *Paths P_1, Q_1, and P_3 are matched inside.*

Claim 3.3.29 implies that the algorithm removes $g = P_1$ and adds e_1 to obtain lollipop H_{i+1}. In iteration i the algorithm enters $Q_3 + Q_4$. However, the algorithm has to leave this $Q_3 + Q_4$ in some subsequent iteration.



(a) H_i (b) H_{i+4}

Figure 3.18: If Claim 3.3.29 holds, G has a subgraph as depicted: the red edges in (a) and (b) are the edge set of H_i and H_{i+1}.

Observation 3.3.30. Let $\ell \in E \setminus E(H_i)$, and $\ell \neq e_1, e_2$. If ℓ has one endpoint in $Q_3 + Q_4$, the other endpoint of ℓ is also in $Q_3 + Q_4$.

We prove Observation 3.3.30 in Section 3.3.3. Observation 3.3.30 implies that the only way that the algorithm can leave $Q_3 + Q_4$ is either to remove e_1 or to add e_2. In the first case, the algorithm adds $P_1 = g$ back after removing e_1. Notice that then $C(H_j) = C(H_i)$. Hence, H_i and H_j are compatible which is a contradiction.

In the second case, the algorithm adds e_2, removes P_3, adds e_3, and removes Q_1 to arrive at u. But W cannot go back to u by deleting an edge, since G is bipartite.

Claim 3.3.29 can be proved similar to the other cases, using the following observation.

Observation 3.3.31. The following properties hold:

1. P_1 is matched inside;
2. Q_1 is matched inside; and
3. P_3 is matched inside.

We defer the proof of Observation 3.3.31 to Section 3.3.3.

Case 4. (e_3 **is the highest jump,** e_1 **is the lowest jump of** H_i)

Let Q_1, Q_2, Q_3, and Q_4, be the up_1-path, p_1p_2-path, p_2p_3-path, and the p_3w-path in $P(H_i)$. This case is shown in Figure 3.19.

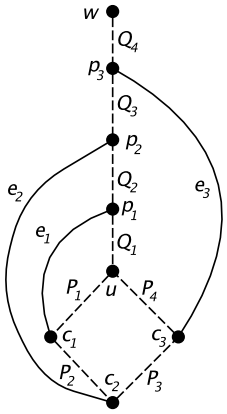


Figure 3.19: Case 4.

Consider $C = Q_1 + P_1 + e_1$. There must be an edge $k \in E \setminus E(H_i)$ with one endpoint $s \notin \{p_1, c_1, u\}$ in C, and one endpoint s' not in C.

Observe that s is either in P_1 or in Q_1. If s is in Q_1, then by Lemma 3.3.14, s' cannot be in Q_2. Also if s in P_1, then by Lemma 3.3.6, s' is in P_3.

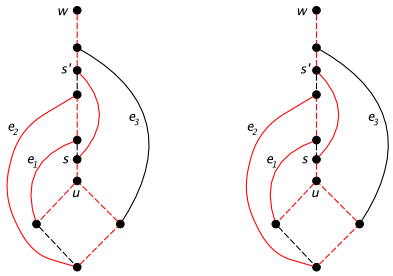
Case 4i. s **in** Q_1 **and** s' **in** Q_3**.**

We start by proposing the following claim.

Claim 3.3.32. *Paths* $P_1, Q_1(p_1, s), Q_3(s', p_2)$, *and* P_2 *are matched inside.*

Assuming that Claim 3.3.32 holds, the algorithm alternately removes and adds $g = P_1, e_1, Q_1(p_1, s), k, Q_3(s', p_2), e_2, P_2, g = P_1$, to obtain H_j. Clearly since $C(H_j)$ contains both g and f, H_i and H_j are compatible (See Figure 3.20). This contradicts our to the original assumption. Proof of Claim 3.3.32 goes similar as discussed in Case 1 after the following observation.

Observation 3.3.33. The following properties hold:



(a) H_i (b) H_j

Figure 3.20: If Claim 3.3.32 holds, G has a subgraph as depicted: the red edges in (a) and (b) are the edge set of H_i and H_j, respectively.

1. P_1 is matched inside;
2. $Q_1(p_1, s)$ is matched inside;
3. $Q_3(s', p_2)$ is matched inside; and
4. P_2 is matched inside.

We prove Observation 3.3.33 in the next section.

Case 4ii. s in Q_1 and s' in Q_4.

As in the other cases, we start with a claim.

Claim 3.3.34. *Paths $P_1, Q_1(p_1, s), P_3$, and Q_2 are single edges.*

Using Claim 3.3.34, the algorithm removes $g = P_1$ and adds e_1 to obtain lollipop H_{i+1}. Notice that $base(H_{i+1}) = p_1$. Then, the algorithm removes $Q_1(p_1, s)$ and adds k to arrive at vertex s'. Let us make a quick observation.

Observation 3.3.35. Let $\ell \in E \setminus E(H_i)$, and $\ell \neq k, e_3$. If ℓ has one endpoint in Q_4, the other endpoint of ℓ is also in Q_4.

We defer the proof of Observation 3.3.35 to the next section.

Since the algorithm arrives at u in iteration $j-1$, it has to leave Q_4 at some iteration between iterations $i+2$ and $j-1$. By Observation 3.3.35, the only way is either by removing

k, or by adding e_3. In the first case, let $H_{k'}$, $i+2 < k' < j$ be the lollipop in iteration k' before removing k. Notice that $base(H_{k'}) = base(H_{i+2}) = s'$. However, by Observation 3.3.35 we have

$$(E(H_{k'}) \setminus E(H_{i+2})) \cap jump(H_i) = \emptyset.$$

Therefore, by Lemma 3.3.11, H_i and H_j are compatible, which is a contradiction.

Thus, we may assume that the algorithm adds e_3. After adding e_3, the algorithm removes P_3. In the next iteration the algorithm adds e_2 and removes Q_2. At this point the algorithm arrives at p_1. Recall that p_1 was $base(H_{i+1})$. The algorithm leaves p_1 by removing an edge in iteration $i+1$, and arrives to p_1 again by removing an edge in some iteration after $i+1$. Hence, W has an odd closed walk as a subgraph. This is a contradiction to bipartiteness of G.

The proof of Claim 3.3.34 is similar to the other cases using the following observation.

Observation 3.3.36. The following properties hold:

1. P_1 is matched inside;
2. $Q_1(p_1, s)$ is matched inside;
3. P_3 is matched inside; and
4. Q_2 is matched inside.

We prove Observation 3.3.36 in Section 3.3.3.

Case 4iii. s in P_1 and s' in P_3.

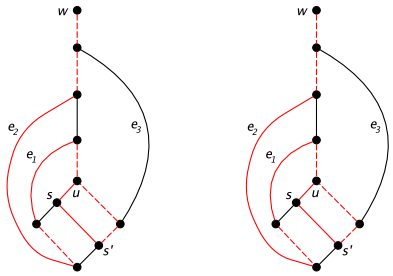
We start by the following claim,

Claim 3.3.37. *Paths $P_1(u, s), P_3(s', c_2), Q_2$, and $P_1(c_1, s)$ are single edges.*

If Claim 3.3.37 holds (Figure 3.21), the lollipop algorithm alternately removes and adds $g = P_1(u, s), k, P_3(s', c_2), e_2, Q_2, e_1, P_1(c_1, s), g = P_1(u, s)$ to obtain H_j. Notice that g and f are in $C(H_j)$. Therefore H_i and H_j are compatible. This is a contradiction to the original assumption.

The proof of Claim 3.3.37 is an immediate consequence of the following observation.

Observation 3.3.38. The following properties hold:



(a) H_i (b) H_j

Figure 3.21: If Claim 3.3.37 holds: the red edges in (a) and (b) are the edge set of H_i and H_j.

1. $P_1(u, s)$ is matched inside;
2. $P_3(s', c_2)$ is matched inside;
3. Q_2 is matched inside; and
4. $P_1(c_1, s)$ is matched inside.

We will prove Observation 3.3.38 later in Section 3.3.3. This finishes the proof. □

3.3.3 Proofs of observations

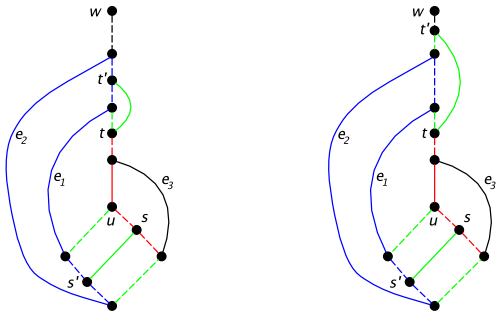
This section is dedicated to the proofs of the observations in the proof of Theorem 3.3.2.

Observations in Case 1

In Observations 3.3.18, 3.3.19 there is an edge k with one endpoint s in the internal vertices of P_4, and one endpoint s', in the internal vertices of P_2. Notice that by Lemma 3.3.13, Q_1 is matched inside.

Observation 3.3.18. There is no edge in $E \setminus E(H_i)$, that is different from e_1 and e_2, with one endpoint in $Q_3 + Q_4$, and one not in $Q_3 + Q_4$.

Proof. For contradiction let $\ell \in E \setminus E(H_i)$ be an edge with endpoints t not in $Q_3 + Q_4$ and t' in $Q_3 + Q_4$. Hence t is in Q_2.



(a) t' in Q_3 (b) t' in Q_4

Figure 3.22: Figures for the proof of Observation 3.3.18

If t' is in Q_3, let $C' = e_2 + P_2 + e_1 + Q_3$, and $P' = Q_2(t, p_3) + Q_1 + P_4$. Observe that P_1, k, P_3, ℓ, and $Q_2(t, p_1)$ are five disjoint paths from C' to P'. Also if t' in Q_4, then $P_1, k, P_3, Q_2(t, p_1)$, and $\ell + Q_4(t', p_2)$ are five disjoint paths from C' to P', contradiction to $G \in \mathcal{A}$ (See Figures 3.22a and 3.22b). □

Observation 3.3.19. The following properties hold:

1. P_1 is matched inside;
2. Q_2 is matched inside;
3. $P_4(c_3, s)$ is matched inside; and
4. $P_2(s', c_2)$ is matched inside.

Proof.

1. By Lemma 3.3.16.
2. Immediately from Lemma 3.3.13 and Observation 3.3.18.
3. By Lemma 3.3.16.
4. By Lemma 3.3.16.

□

Observations in Case 2

In this case, the graph H_i (together with its jumps) is isomorphic to the one in Case 1.

Observation 3.3.21. There is no edge in $E \setminus E(H_i)$, that is different from e_2 and e_3, with one endpoint in $Q_3 + Q_4$, and one not in $Q_3 + Q_4$.

Proof. Similar to Observation 3.3.18 □

Observation 3.3.22. We have the following properties:

1. $P_1(u, s)$ is matched inside;
2. $P_3(s', c_2)$ is matched inside; and
3. P_4 is matched inside.

Proof. By Lemma 3.3.16. □

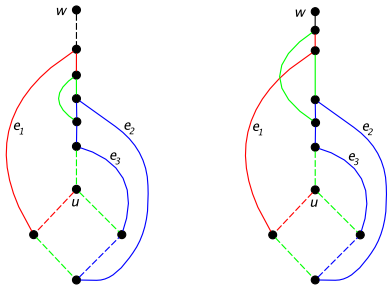
Observations in Case 3

In Case 3, e_1 is the highest jump of H_i and e_3 is the lowest jump. Hence, by Lemma 3.3.14, there is no edge $k \in E \setminus E(H_i)$ with one endpoint in Q_1 and one in Q_2.

The following observation holds for all the subcases of Case 3.

Observation 3.3.39. If $\ell \in E \setminus E(H_i)$, $\ell \neq e_1$, has one endpoint t in $Q_3 + Q_4$, then its other endpoint t' is not in Q_2.

Proof. Suppose there is such an edge ℓ for contradiction. Let $C' = Q_2 + e_3 + P_3 + e_2$. If t is in Q_3, then let $P' = P_1 + e_1 + Q_3(p_1, t)$. Then $\ell, Q_3(p_2, t), Q_1, P_4$, and P_1 are five disjoint paths from P' to C'. If instead t is in Q_4, let $P' = P_1 + e_1 + Q_4(p_1, t)$. In this case ℓ, Q_3, Q_1, P_4, and P_2 are five disjoint paths from P' to C'. Thus, in both cases G would have a WH(6)-minor, which is a contradiction (See Figure 3.23.). □



(a) if t is in Q_3 (b) If t is in Q_4

Figure 3.23: Different cases in the proof of Observation 3.3.39

Case 3i

Recall from the previous section that in Observations 3.3.24 and 3.3.25 we assume there is an edge $k \in E \setminus E(H_i)$ with one endpoint s in Q_1 and one endpoint s' in Q_3.

Observation 3.3.24. There is no edge in $E \setminus E(H_i)$, that is different from e_1 and k, with one endpoint in $Q_3(p_1, s') + Q_4$, and one endpoint not in $Q_3(p_1, s') + Q_4$.

Proof. Suppose for contradiction that there is $\ell \in E \setminus E(H_i)$, $\ell \neq e_1, k$, with one endpoint t in $Q_3(p_1, s') + Q_4$, and one endpoint t' not in $Q_3(p_1, s') + Q_4$. By Observation 3.3.39, it must be the case that either t' is in Q_1 or in $Q_3(p_2, s')$.

If t' is in Q_1 and t is in $Q_3(p_1, s')$, then let $C' = Q_1 + P_4 + e_3$ and $P' = P_2 + e_1 + Q_3$. In this case ℓ, k, Q_2, P_1, and P_3 are five disjoint paths from P' to C'. Hence, $G \notin \mathcal{A}$, which is a contradiction (Figure 3.24a).

If t' is in Q_1 and t is in Q_4, similar to the other case we can find a WH(6)-minor since t is in Q_1 (see Figure 3.24b.).

Thus, we may assume that t' is in $Q_3(p_2, s')$. Now, if t is in $Q_3(p_1, s')$, we can consider cycle $C' = P(H_i)(s, s') + m$, and connected subgraph $P' = Q_3(t, p_1) + e_1 + P_2 + P_3 + P_4$. Then $e_2, e_3, Q_1(s, u), \ell$, and $Q_3(s', t)$ are five disjoint paths from P' to C' (Figure 3.24c). Similarly, if t is in Q_4 (Figure 3.24d), there is a WH(6)-minor in G, which is a contradiction.

□

Observation 3.3.25. The following properties hold:

1. P_1 is matched inside;

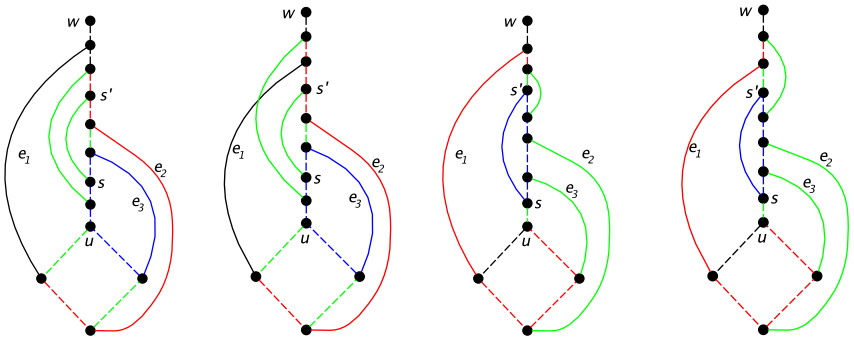


Figure 3.24: Different cases in the proof of Observation 3.3.24

2. $Q_1(s, p_3)$ is matched inside; and
3. P_4 is matched inside.

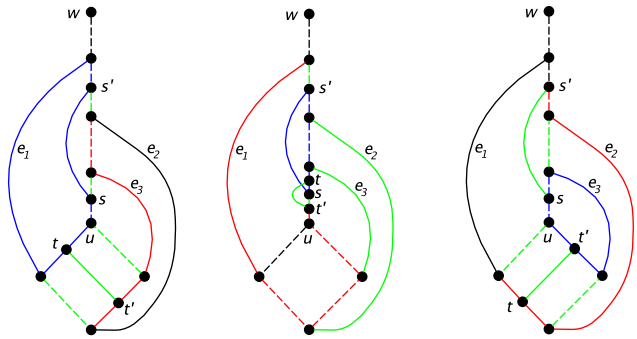


Figure 3.25: Different cases in the proof of Observation 3.3.25

Proof.

1. Suppose for contradiction that there is an ℓ, with endpoints t in P_1 and t' in P_3. Consider $C' = e_1 + P_1 + Q_1(u, s) + m + Q_(s', p_1)$, and $P' = Q_2 + e_3 + P_3$. One could easily check that $P_2, P_4, \ell, Q_1(s, p_3)$, and $Q_3(p_2, s')$ are five disjoint paths from P' to C'. Contradiction to $G \in \mathcal{A}$ (Figure 3.25a).

2. Assume otherwise. The only possibility is to have an edge with one endpoint t in $Q_1(s, p_3)$, and one t' in $Q_1(u, s)$. In this case, consider cycle $C' = m + P(H_i)(s, s')$, $P' = P_4 + P_3 + P_2 + e_1$. Observe that $e_2, e_3, \ell, Q_3(s', p_1)$, and $Q_1(s, t')$ are five disjoint paths from P' to C'. This is a contradiction, since $G \in \mathcal{A}$. This case is illustrated in Figure 3.25b.

3. If not, let $\ell = (t, t')$, where t is in P_2 and t' in P_4. Consider $C' = e_3 + Q_1 + P_4$, and $P' = Q_3(s', p_2) + e_2 + P_2$. Now k, Q_2, P_1, P_3, and ℓ are five disjoint paths from P' to C'. This implies that G has a WH(6)-minor, which is a contradiction (See Figure 3.25c.).

□

Case 3ii

For the proof of Observations 3.3.27 and 3.3.28 we suppose there is an edge $k \in E \setminus E(H_i)$ with one endpoint s in Q_1 and one endpoint s' in Q_4.

Observation 3.3.27. Let $\ell \in E \setminus E(H_i)$, and $\ell \neq k, e_1$. If ℓ has one endpoint in Q_4, the other endpoint of ℓ is also in Q_4.

Proof. Similar to Observation 3.3.24. □

Observation 3.3.28. We have the following properties:

1. P_1 is matched inside;
2. Q_3 is matched inside;
3. P_3 is matched inside; and
4. $Q_1(p_3, s)$ is matched inside.

Proof.

1. Similar to Observation 3.3.25.

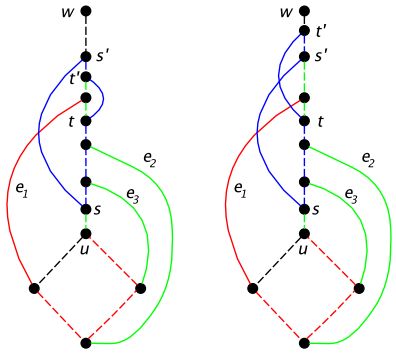


Figure 3.26: Different cases in proof of the Observation 3.3.28

2. Assume not. Then there is an edge $\ell \in E \setminus E(H_i)$, with one endpoint t in Q_3 and another endpoint t' not in Q_3. Notice that by Observation 3.3.39, t' cannot be in Q_2. Furthermore, since we reached a contradiction in Case 2i, we can assume that t' is not in Q_1 either. Thus, it must be that t' is in Q_4.

   We consider two separate cases. If t' is in $Q_4(s', p_1)$, then let $C' = m + Q_4(s', t') + \ell + Q_3(t, p_2) + Q_2 + Q_1(p_3, s)$ and $P' = e_1 + P_2 + P_3 + P_4$. Then $Q_1(u, s), e_3, e_2, Q_3(p_1, t)$, and $Q_4(p_1, t')$ are five disjoint paths from C' to P'. Similarly, if t' is in $Q_4(w, s')$, then $Q_1(u, s), e_3, e_2, Q_3(p_1, t)$, and $Q_4(p_1, s')$ are five disjoint paths from C' to P' (See Figure 3.26.).

3. Similar to Observation 3.3.25.

4. Similar to Observation 3.3.25.

□

Case 3iii

Recall that Observations 3.3.30 and 3.3.31 were stated in Case 3iii. Thus, we may assume for the proof of the two observations that there is an edge $k \in E \setminus E(H_i)$ with one endpoint s in P_4 and one endpoint s' in P_2.

Observation 3.3.30. Let $\ell \in E \setminus E(H_i)$, and $\ell \neq e_1, e_2$. If ℓ has one endpoint in $Q_3 + Q_4$, the other endpoint of ℓ is also in $Q_3 + Q_4$.

Proof. By Observation 3.3.39, there is no edge $\ell \in E \setminus E(H_i)$ with one endpoint in Q_2 and one endpoint in $Q_3 + Q_4$. □

Observation 3.3.31. We have the following properties:

1. P_1 is matched inside;
2. Q_1 is matched inside; and
3. P_3 is matched inside.

Proof.

1. By Lemma 3.3.16.
2. By Lemma 3.3.14, there is no edge in $E \setminus E(H_i)$ from Q_1 to Q_2. By Case 3i and Case 3ii, there is no edge in $E \setminus E(H_i)$ from Q_1 to Q_3 and Q_4, respectively. This means that Q_1 is matched inside.
3. By Lemma 3.3.16.

□

Observations in Case 4

In this case, the subgraph of G that consists of H_i and its jumps is isomorphic to the one in the previous case. Hence, we are able to use arguments similar to the ones in Case 3.

The following observation holds for all the subcases of Case 4.

Observation 3.3.40. If $\ell \in E \setminus E(H_i)$, $\ell \neq e_3$, has one endpoint t in $Q_3 + Q_4$, then its other endpoint t' is not in Q_2.

Proof. Same as Observation 3.3.39. □

Case 4i

In this case, there is an edge $k \in E \setminus E(H_i)$ that has one endpoint s in Q_1 and one endpoint s' in Q_3. Suppose without loss of generality that among all edges in $E \setminus E(H_i)$ with one endpoint in Q_1 and one in Q_3, k is the one whose endpoint in Q_3 is closest to p_2.

Observation 3.3.33. We have the following properties:

1. P_1 is matched inside;
2. $Q_1(p_1, s)$ is matched inside;
3. $Q_3(s', p_2)$ is matched inside; and
4. P_2 is matched inside.

Proof.

1. Similar to the proof in Observation 3.3.25 for showing that P_4 is matched inside.
2. Similar to the proof in Observation 3.3.25 for showing that $Q_1(p_1, s)$ is matched inside.
3. Suppose not. Then there is an edge $\ell \in E \setminus E(H_i)$ with one endpoint t in $Q_3(s', p_2)$ and another endpoint t' not in $Q_3(s', p_2)$. By choice of k, t' cannot be in Q_1. Moreover, by Observation 3.3.40, t' cannot be in Q_2. Hence, t' is either in $Q_3(s', p_3)$ or in Q_4.

   The argument here is similar to the one used for Observation 3.3.24 in Case 3i (isomorphic to this case), where we showed that there is no edge in $E \setminus E(H_i)$ with one endpoint in $Q_3(s', p_3) + Q_4$ and one in $Q_3(s', p_2)$.
4. Similar to the proof in Observation 3.3.25 for showing that P_1 is matched inside.

□

Case 4ii

In this case, there is an edge $k \in E \setminus E(H_i)$ with one endpoint s in Q_1 and another endpoint t' in Q_4. The graph H_i together with its jumps and edge k is isomorphic to the one in Case 3ii.

Observation 3.3.35. Let $\ell \in E \setminus E(H_i)$, and $\ell \neq k, e_3$. If ℓ has one endpoint in Q_4, the other endpoint of ℓ is also in Q_4.

Proof. Same as Observation 3.3.27. □

Observation 3.3.36. We have the following properties:

1. P_1 is matched inside;
2. $Q_1(p_1, s)$ is matched inside;
3. P_3 is matched inside; and
4. Q_2 is matched inside.

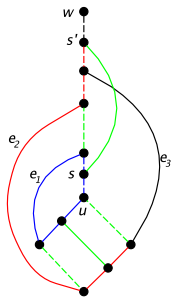


Figure 3.27: Observation 3.3.36: P_1 and P_3 are matched inside.

Proof.

1. Suppose not. Then there is $\ell = (t, t') \in chord(H_i)$, with t in P_1 and t' in P_3. Now consider the connected subgraph $P' = P_3 + e_2 + P(H_i)(p_2, s')$, and cycle $C' = e_1 + Q_1 + P_1$. Observe that Q_2, k, ℓ, P_2, and P_4 are five disjoint paths from P' to C'. This is a contradiction to the fact that G is WH(6)-minor free.
2. Identical to the proof in Observation 3.3.28 for showing $Q_1(p_3, s)$ is matched inside.
3. Immediately from the fact that P_1 is matched inside.
4. Let $\ell \in E \setminus E(H_i)$ be an edge with endpoint t in internal vertices of Q_2 and endpoint t' not in Q_2. By Lemma 3.3.14, t' is not in Q_1. By Observation 3.3.40, t' is not in $Q_3 + Q_4$.

□

Case 4iii

In this case there is an edge $k = (s, s') \in E \setminus E(H_i)$, such that s is in P_1 and s' is in P_3.

Observation 3.3.38. The following properties hold:

1. $P_1(u, s)$ is matched inside;
2. $P_3(s', c_2)$ is matched inside;
3. Q_2 is matched inside; and
4. $P_1(c_1, s)$ is matched inside.

Proof.

1. By Lemma 3.3.16.
2. By Lemma 3.3.16.
3. Suppose otherwise. Hence, there is an edge $\ell = (t, t') \in E \setminus E(H_i)$, such that t is in Q_2 and t' in not in Q_2. Now by Observation 3.3.40, t' is not in $Q_3 + Q_4$. Moreover, by Lemma 3.3.14, t' cannot be in Q_1.
4. By Lemma 3.3.16.

□

3.4 An infinite family of graphs of class $\mathcal{A}$

Although the class $\mathcal{A}$ of graphs mentioned in Theorem 3.3.2 looks very specific we show it contains infinitely many graphs. To do this, we propose a constructive procedure.

Let G_1 be the cube and label two adjacent vertices of G_1 with v_1 and w_1. In Figure 3.28, graph G_1 is shown. Graph G_1 is cubic, planar, bipartite and WH(6)-minor free.

We recursively construct graph G_i from graph G_{i-1} for $i > 1$. Consider a graph H isomorphic to G_1, and let v and w be the vertices of H corresponding to v_1, and w_1 in G_1. Moreover, consider a planar embedding of H where v and w are on the outer face (Embedding shown in Figure 3.28.). Let w' be the other neighbor of v on the outer face.

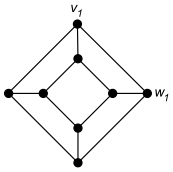


Figure 3.28: Graph G_1.

In G_{i-1} remove the edge (v_{i-1}, w_{i-1}). Also, remove the edge (v, w') in H. Add an edge between the vertices w_{i-1} and w', namely e_i, and between v_{i-1} and v, namely f_i. We call the resulting graph G_i. Label vertex v of G_i with v_i and w with w_i. Finally, label the edge (v_i, w_i) with g_i. See Figure 3.29 for G_3.

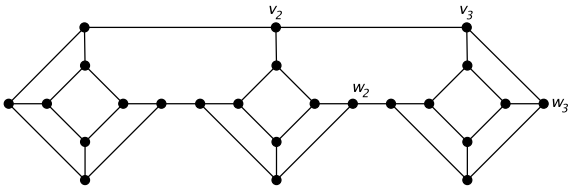


Figure 3.29: Graph G_3.

It is easy to see that G_i is cubic, planar and bipartite for $i \geq 1$. We will prove the following theorem to show that G_i is WH(6)-minor free.

Theorem 3.4.1. *G_i is WH(6)-minor free for $i \geq 1$.*

Proof. We will proceed by induction on i. The base case G_1 is clearly WH(6)-minor free. Suppose that G_{i-1} is WH(6)-minor free, for $i > 1$. We will prove that G_i is also WH(6)-minor free.

Suppose for contradiction that G_i has a WH(6)-minor. This means that there are connected subgraphs H and C of G_i, such that C is a cycle, and H and C are vertex disjoint. Moreover, there are five subgraphs P_j, $j \in \{1, \ldots, 5\}$ of G_i, such that (1) P_j is internally vertex disjoint from C, H, and P_k, where $k \in \{1, \ldots, 5\} \setminus \{j\}$, (2) P_j is a path, (3) P_j has one endpoint in C and one in H for $j \in \{1, \ldots, 5\}$, and (4) for $j, k \in \{1, \ldots, 5\}$, if $j \neq k$, then P_j and P_k have distinct endpoints in C.

Notice that by construction of G_i, $G_i - e_i - f_i$ has two connected components. Label the one isomorphic to $G_{i-1} - g_{i-1}$ with J_1, and the other component that is ismorphic to G_1 with one edge removed with J_2.

Since G_{i-1} and G_1 are WH(6)-minor free, the subgraphs H and C are not completely lying in J_1 or J_2. Hence, either C or H contains e_i or f_i.

Note that, since C is a cycle, if e_i is an edge in C, then it must be the case that also f_i is an edge of C. This implies that H, P_j for $j \in \{1, \ldots, 5\}$ completely lie in J_1. Contract J_2 in G_i to get a minor H_i of G_i. It is easy to see that H_i is isomorphic to G_{i-1}. Moreover, H_i has a subgraph C' that is obtained from C by contracting the part of C that lies in J_2. Clearly C', H, and P_j, for $j \in \{1, \ldots, 5\}$ constitute a WH(6)-minor in H_i, which is a contradiction.

Thus, we may assume that C completely lies in J_1. If H contains both e_i and f_i, then contracting J_1 would give a WH(6)-minor in G_{i-1}, which is a contradiction.

Suppose without loss of generality that $e_i \in E(H)$ and $f_i \notin E(H)$. Now if no P_j, for $j \in \{1, \ldots, 5\}$ contains the edge f_i, the graph obtained by deleting J_2 from G_i would have a WH(6)-minor, but this graph is a subgraph of G_{i-1} which is a contradiction. However, even if f_i is in P_j for some $j \in \{1, \ldots, 5\}$, we could delete J_2 from G_i. Then if P'_j is the part of P_j in J_1, we can let $P_j = P'_j + g_{i-1}$ and let $H' = H \cap J_1$. Then C, H', and P_j for $j \in \{1, \ldots, 5\}$ constitute a WH(6)-minor. □

Finally, we need to show that G_i is Hamiltonian for $i \geq 1$.

Lemma 3.4.2. *G_i has a Hamiltonian cycle through g_i for $i \geq 1$.*

Proof. Proceed by induction on i. Clearly G_1 has a Hamiltonian cycle through g_1. For $i \geq 1$, suppose G_{i-1} has a Hamiltonian cycle H through g_{i-1}. Remove g_{i-1} from H and extend it to obtain a Hamiltonian cycle for G_i through g_i. □

Chapter 4

Conclusion

We provide an infinite class of cubic graphs for which the number of steps required by the lollipop algorithm is a polynomial in the number of vertices of the graph. A corollary of this is that the problem of finding a second Hamiltonian cycle in cubic graphs (FSHCC) can be efficiently solved by the lollipop algorithm. However, finding a first Hamiltonian cycle for a graph in class $\mathcal{A}$ does not seem to be intractable. In the case of Barnette graphs, the complexity of finding the first Hamiltonian cycle is not known, therefore, it would be interesting to find out the complexity of FSHCC in Barnette graphs.

The main open question of this thesis would be to settle Conjecture 3.2.2. It might be that analyzing how the algorithm behaves over the dual graphs, i.e. Eulerian triangulations, give more tools in proving Conjecture 3.2.2.

In fact, for dual graphs of cubic planar graphs, the lollipop algorithm moves along the triangles in a similar way as the exchange algorithm for finding a second room-partitioning does on 2-manifolds. Is it possible to prove that FSHCC is in PPAD if the graph is planar?

For general cubic bipartite graphs, we do not know any cubic bipartite graphs for which the lollipop algorithm requires a number of steps exponential in the size of the graph. Recall that Krawczyk's graph is not bipartite. Is it possible to use the graph in Figure 3.1 to create a bipartite exponential example?

Another interesting question would be to analyze the power of the lollipop algorithm. Disser and Skutella [11] proposed a classification of algorithms by the complexity of the problems they can *implicitly* solve.

Definition 4.0.3. An algorithm given by a Turing machine T *implicitly* solves problem P, if for a given instance I of P, there is a efficient algorithm that computes an input $\mathcal{I}$ for T,

and a bit b in the complete configuration of T (a binary string that represents machine's state, the content of the tape, and position of machine's head with respect to the content), such that instance I is a yes-instance if and only if bit b is flipped at some point during the execution of T with input $\mathcal{I}$.

An algorithm is *NP-mighty*, if it can implicitly solve an NP-complete problem. In [11], the authors show that the Simplex method (with Dantzig's original pivot rule) is NP-mighty. This result was later improved to show that the Simplex method (with Dantzig's pivot rule) is PSPACE-mighty [17]. Similarly, it is proved that the Lemke-Howson algorithm solves a PSPACE-complete problem [22]. Is the lollipop algorithm NP-mighty? Is it PSPACE-mighty? A proof for NP-mightiness would require a counting gadget, like the one in Krawczyk's graph. What about special cases of exchange algorithm? Could NP-mightiness justify the exponential behavior of the exchange algorithm for finding a second perfect matching in Eulerian graphs, given the problem itself can be solved in polynomial time by the blossom algorithm?

Finding a second room-partitioning of a planar triangulation, however, seems to be much harder. Is it PPAD-complete?

Since 1990, complexity class PPA has resisted "natural" complete problems. Grigni [24], showed that Sperner's lemma for non-orientable 3-manifolds is complete for PPA. However, this problem has a Turing machine embedded in the input. A more natural problem seems to be the one of finding a second Hamiltonian cycle in odd-degree graphs.

Bibliography

[1] S. N. A. Takanori, N. Takao. NP-completeness of the hamiltonian cycle problem of bipartite graphs. *Journal of Information Processing*, 3, 1980.

[2] H. Alt, M. Payne, J. Schmidt, and D. Wood. Thoughts on Barnette's Conjecture. *CoRR*, abs/1312.3783, 2013.

[3] D. W. Barnette. Conjecture 5. *W. J. Tutte (Ed.) Recent Progress in Combinatorics, Proceedings of the 3rd Waterloo conference on Combinatorics*, 3:343, 1969.

[4] J.-A. Bondy and U. S. R. Murty. *Graph theory*. Graduate texts in mathematics. Springer, New York, London, 2007. OHX.

[5] S. Boyd, R. Sitters, S. van der Ster, and L. Stougie. TSP on cubic and subcubic graphs. *Integer Programming and Combinatoral Optimization*, 6655:65–77, 2011.

[6] X. Chen, X. Deng, and S.-H. Teng. Settling the complexity of computing two-player Nash equilibria. *J. ACM*, 56(3):14:1–14:57, May 2009.

[7] J. R. Correa, O. Larré, and J. A. Soto. TSP tours in cubic graphs: Beyond 4/3. *CoRR*, abs/1310.1896, 2013.

[8] C. Daskalakis, P. W. Goldberg, and C. H. Papadimitriou. The complexity of computing a Nash equilibrium. *SIAM Journal on Computing*, 39(1):195–259, 2009.

[9] B. M. D.Holton, B. Manvel. Hamiltonian cycles in cubic 3-connected bipartite planar graphs. *Journal of Combinatorial Theory, Series B*, 38:279–297, 1985.

[10] R. Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

[11] Y. Disser and M. Skutella. In defense of the simplex algorithm's worst-case behavior. *CoRR*, abs/1311.5935, 2013.

[12] M. Dyer and A. Frieze. Planar 3DM is NP-complete. *Journal of Algorithms*, 7(2):174 – 184, 1986.

[13] J. Edmonds. Paths, trees, and flowers. In I. Gessel and G.-C. Rota, editors, *Classic Papers in Combinatorics*, Modern Birkhuser Classics, pages 361–379. Birkhuser Boston, 1987.

[14] J. Edmonds. Euler complexes. *Research Trends in Combinatorial Optimization eds. W. Cook, L. Lovász, J. Vygen, Springer, Berlin*, pages 65–68, 2009.

[15] J. Edmonds and L. Sanità. On finding another room-partitioning of the vertices. In *Electronic Notes in Discrete Mathematics*, volume 36, pages 1257–1264, 2010.

[16] J. Edmonds and L. Sanità. Exponentiality of the exchange algorithm for finding another room-partitioning. *Discrete Applied Mathematics*, 164, Part 1:86 – 91, 2014. Combinatorial Optimization.

[17] J. Fearnley and R. Savani. The Complexity of the Simplex Method. In *Proc. of the ACM Symposium on Theory of Computing (STOC)*, pages 201–208, 2015.

[18] T. Feder and C. S. Subi. On barnette's conjecture. *Electronic Colloquium on Computational Complexity (ECCC)*, (015), 2006.

[19] J. Florek. On Barnettes Conjecture and property. *Electronic Notes in Discrete Mathematics*, 43:375 – 377, 2013.

[20] M. Goemans. Worst-case comparison of valid inequalities for the tsp. *Mathematical Programming*, 69(1-3):335–349, 1995.

[21] P. W. Goldberg. A survey of PPAD-completeness for computing Nash equilibria. *CoRR*, abs/1103.2709, 2011.

[22] P. W. Goldberg, C. H. Papadimitriou, and R. Savani. The complexity of the homotopy method, equilibrium selection, and Lemke-Howson solutions. *ACM Transactions on Economics and Computation*, 1(2):Article 9, 25 pages, 2013. Preliminary conference version appeared at FOCS 2011.

[23] P. R. Goodey. Hamiltonian circuits on polytopes with even sides. *Israel Journal of Mathematics*, 22:52–56, 1975.

[24] M. Grigni. A Sperner lemma complete for {PPA}. *Information Processing Letters*, 77(5–6):255 – 259, 2001.

[25] P. Haxell, B. Seamone, and J. Verstraete. Independent dominating sets and hamiltonian cycles. *Journal of Graph Theory*, 54(3):233–244, 2007.

[26] A. Hertel. A survey and strengthening of Barnette's conjecutre. Unpublished manuscript, 2005.

[27] A. K. Kelmans. Constructions of cubic bipartite 3-connected graphs without hamiltonian cycles. *Americal Mathematical Society Translations*, 158(Series 2):127–140, 1994.

[28] A. Krawczyk. The complexity of finding a second Hamiltonian cycle in cubic graphs. *J. Comput. System Sci.*, 58:641–647, 1999.

[29] C. E. Lemke and J. T. Howson. Equilibrium points of bimatrix games. *Journal of Society for Industrial and Applied Mathematics*, 12:413–423, 1964.

[30] J. Merschen. *Nash Equilibria, Gale Strings, and Perfect Matchings.* PhD thesis, London School of Economics, 2012.

[31] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani. *Algorithmic Game Theory.* Cambridge University Press, New York, NY, USA, 2007.

[32] C. H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. System Sci.*, 48(498-532), 1994.

[33] R. Savani and B. von Stengel. Hard-to-solve bimatrix games. *ECONOMETRICA*, 74(2):397–429, 2006.

[34] L. Shapley. A note on the Lemke-Howson algorithm. In M. Balinski, editor, *Pivoting and Extension*, volume 1 of *Mathematical Programming Studies*, pages 175–189. Springer Berlin Heidelberg, 1974.

[35] J. Sheehan. The mutiplicity of hamiltonian circuits in a graph. *M. Fiedler (Ed.) Recent Advances in Graph Theory, Academia, Prague*, pages 447–480, 1975.

[36] P. G. Tait. Listing's topologie. *Philosophical Magazine*, 17(30-46), 1884.

[37] A. G. Thomason. Hamiltonian cycles and uniquely edge colourable graphs. *Ann. Discrete. Math.*, 3:259–268, 1978.

[38] C. Thomassen. Independent dominating sets and a second hamiltonian cycle in regular graphs. *Journal of Combinatorial Theory, Series B*, 72(1):104 – 109, 1998.

[39] M. J. Todd. A generalized complementary pivoting algorithm. *Mathematical Programming*, 6(1):243–263, 1974.

[40] W. T. Tutte. On Hamiltonian circuits. *J. London Math. Soc.*, 21:98–101, 1946.

[41] L. Végh and B. von Stengel. Oriented Euler complexes and signed perfect matchings. *Mathematical Programming*, 150(1):153–178, 2015.