Hamiltonian Cycles in Sparse Graphs

By

Alexander Hertel

A Thesis Submitted in Conformity With the Requirements
For the Degree of Master of Science
Graduate Department of Computer Science
University of Toronto

Abstract

Hamiltonian Cycles

in Sparse Graphs

Alexander Hertel

Master of Science

Graduate Department of Computer Science

University of Toronto

2004

The subject of this thesis is the Hamiltonian Cycle problem, which is of interest in many areas including graph theory, algorithm design, and computational complexity. Named after the famous Irish mathematician Sir William Rowan Hamilton, a Hamiltonian Cycle within a graph is a simple cycle that passes through each vertex exactly once. This thesis provides a history of the problem, a survey of major results, as well as a detailed account of the author's original contributions with respect to sparse graphs. The first of these is the "Stonecarver's Algorithm", which is successful in finding Hamiltonian Cycles in random regular graphs. The second gives upper and lower bounds on the creation of a specific obstruction to Hamiltonicity under the context of the Stonecarver Algorithm. Finally, the third is a theorem which strengthens Barnette's Conjecture.

Acknowledgements

First and foremost, I would like to thank my girlfriend Tanya Pearson for supporting me while working on my Master's degree at the University of Toronto.

I would like to thank my brother Philipp with whom I have worked very closely for many years. The Stone Carver's algorithm was developed jointly with him, and he also deserves credit.

Next I would like to thank my mother for making the sacrifices to pay for my expensive private school education.

A number of professors at the University of Victoria also deserve my gratitude. In the Philosophy Department, I would like to thank Dr. Charles Morgan for being an exceptional teacher, and for encouraging me to attend graduate school; were it not for him, I would probably have entered industry.

In the Computer Science Department, I would like to thank Dr. Valerie King for likewise encouraging me to attend graduate school, but also for supervising Philipp and myself during the two Directed Studies courses in which we developed the Stone Carver's algorithm. Her input was extremely valuable.

Also in the Computer Science Department, I would like to thank Dr. Ulrike Stege and Dr. John Ellis for supervising the research which we performed while holding our NSERC Undergraduate Student Research Awards. That research was both fruitful and enjoyable. Ulrike in particular was always as helpful and friendly as possible, and even got us involved in some of her research. I can only hope that we made a fraction as positive an impression on her as she did on us.

On a similar note, I would also like to thank Dr. Hausi Mueller for always encouraging us and also for supervising our Directed Studies course in Software Engineering.

In the Mathematics Department, I would like to thank Dr. Gary MacGillivray for his helpfulness and willingness to answer what must have been trivial questions early on during our study of Hamiltonian Cycles.

At the University of Toronto, I would like to thank my supervisor, Dr. Alasdair Urquhart in the Philosophy Department for taking me on as his graduate student and also for spending the time meeting with me, lending me his books, photocopying papers on my behalf, and for otherwise giving me advice concerning my research.

I would like to thank Dr. Derek Corneil from the Computer Science Department. Even though he hadn't met me yet, he generously spent his time reading and giving feedback on my work with Barnette's Conjecture. When I finally met him for the first time, I audaciously asked if he would be willing to act as my second reader for this thesis. Once again, he generously agreed to guide me despite having considerable time pressures from elsewhere.

I would likewise like to thank Dr. Toniann Pitassi from the Computer Science Department for all the time she spent meeting with me before I started working with Alasdair. She has always been very supportive and friendly.

I am grateful to Dr. David Johnson at AT&T labs for helping me find a suitable heuristic with which to compete, and to Dr. Keld Helsgaun from the Department of Computer Science at Roskilde University for being so kind as to give me the code for that heuristic.

On a sad note, I would like to acknowledge three researchers who unfortunately all passed away very recently. Claude Berge and William Tutte died in 2002, and Donald Coxeter died in 2003. I was never fortunate enough to meet any of them, yet their work influenced this thesis a great deal.

Last but not least, I would like to thank the Department of Computer Science at the University of Toronto as well as NSERC for supporting me financially.

Contents

List of Figures

Chapter 1

Introduction

1.1 Terminology

This thesis falls in the area of overlap between the fields of graph theory and computer science. As such, we assume that the reader is familiar with elementary graph-theoretic terminology, the basic ideas behind $\mathcal{NP}$-Completeness, as well as tools for algorithm analysis such as asymptotic 'Big-O' notation. We will use [51], [21], and [13] as our respective references for these fundamentals.

For the purpose of this thesis, all graphs are undirected and unweighted, with no self loops nor any multiple edges. Consistent with the standard definition, we use the notation $G = (V, E)$. In addition, we use n to denote $|V|$, and m to denote $|E|$. For the reader's convenience, the end of this thesis contains an alphabetized glossary.

1.2 Subject

The subject of this thesis is the Hamiltonian Cycle problem, which we shall explore as both computer scientists and graph theoreticians. Formally, a Hamiltonian Cycle is a simple cycle which passes through every vertex in a graph exactly once. The Hamiltonian Cycle problem is that of determining whether a given graph contains a Hamiltonian Cycle, and such graphs are termed to be 'Hamiltonian'. More specifically, however, we will focus on Hamiltonian Cycles in sparse graphs. A graph is considered to be sparse if m is $O(n)$, and considered to be dense if m is $\Omega(n^2)$.

1.3 Motivation

There is considerable motivation for studying the Hamiltonian Cycle problem. Firstly, it is a well-studied problem with a long history, and is of interest in its own right. Secondly, it is $\mathcal{NP}$-Complete, so the discovery of an efficient polynomial-time algorithm which solves the Hamiltonian Cycle problem would immediately yield efficient algorithms for solving hundreds of other $\mathcal{NP}$-Complete problems. Finally, the Hamiltonian Cycle problem lies at the heart of the Traveling Salesman problem (TSP), which is probably the most studied combinatorial optimization problem in history. TSP takes as input a complete weighted graph G, and an integer k, and asks whether the graph contains a Hamiltonian Cycle whose combined edges weigh at most k. The optimization version of TSP leaves out the integer k, and basically amounts to finding the lowest-weight Hamiltonian Cycle. TSP has many real-world applications, the least of which is the routing of vehicles and door-to-door salesmen. Although it is not obvious, problems in the areas of robot control, computer wiring, VLSI design, chronological sequencing, crystallography, and many others can be formulated as instances of TSP [30].

Sparse graphs are of interest for two reasons. Firstly, most graphs seen in practical applications such as computer or road networks are sparse. Secondly, difficult problems tend to attract more attention, and with respect to Hamiltonicity, sparse graphs tend to be more difficult to solve than dense graphs. Intuitively, this is because dense graphs contain many more edges, and the presence of more edges typically increases the likelihood and ease of finding a Hamiltonian Cycle in a graph.

1.4 Purpose

First and foremost, the purpose of this thesis is to present research results. Rather than presenting one major result, however, we focus on several smaller results pertaining to Hamiltonian Cycles in sparse graphs.

In addition, as a Master's thesis, this work has the secondary purpose of demonstrating a capacity for original research, as well as a capacity for organizing ideas in a useful way. Also consistent with the idea of a Master's thesis, the results described herein should not be viewed as a finished product. On the contrary, almost all of the results can be improved, and are meant to be 'stepping stones' on the path to more impressive work. Since these improvements are not trivial, this thesis represents a natural transition to Ph.D. level research.

1.5 Overview

This thesis follows the typical introduction / results / conclusion layout and contains 5 Chapters.

Chapter 2 provides an introduction to Hamiltonian Cycles, and contains two parts. Section 2.1 describes the history of the problem with an emphasis on early history, especially the roles played by Hamilton & Kirkman. This leads into Section 2.2, which categorizes Hamiltonicity research into the three areas of graph theory, algorithms, and computational complexity. Although giving a comprehensive survey of the field is impractical due to the enormous amount of research that has been done, we do give a brief overview, with an emphasis on the necessary conditions for Hamiltonicity that have been discovered so far. The reason for this emphasis is that these necessary conditions are very useful tools for determining non-Hamiltonicity, and will resurface in subsequent sections under various contexts.

Chapter 3 constitutes the bulk of this thesis, and is divided into a number of sections, each containing a different aspect of the author's research. In some ways, Section 3.1 is the most important section of this thesis. This is because it describes several ideas that are critical to understanding the remaining sections. These ideas are 'forced edges', 'pruning', and the 'Look-Ahead Theorem'. A forced edge in a graph G is an edge that must be included in every Hamiltonian Cycle of G. Forcing arbitrary edges is somewhat risky because it can destroy a graph's Hamiltonicity. However, some edges happen to be in every Hamiltonian Cycle of a graph. Identifying such edges and 'safely' forcing them is a great advantage because it provides us with extra information that can be used to determine Hamiltonicity. In contrast, some edges are part of no Hamiltonian Cycle, and should therefore be deleted. The act of identifying such edges and deleting them is referred to as 'pruning'. We use the concept of forced edges to extend the list of necessary conditions given in Section 2.2. Finally, the 'Look-Ahead Theorem' is a powerful tool both algorithmically and theoretically. Algorithmically, it can be used to speed up Hamiltonian Cycle algorithms. Theoretically, it allows us to phrase our necessary conditions as rules that can be implemented within Hamiltonian Cycle algorithms, and allows us to build powerful compound necessary conditions for Hamiltonicity, taking this research into the realm of proof complexity.

Referring to Section 3.1 as 'original research' would be inappropriate, although it does contain some original research. Many of the ideas have been thought of thousands of times before, but rarely published due to their simplicity. Others have been published, but under different contexts. What is original about this section, however, is that this seems to be the first time that anyone has potently and formally combined the ideas of forcing, pruning, and look-ahead simultaneously under the banner of determining Hamiltonicity. We submit that this a powerful and useful way to group these ideas.

Section 3.2 gives a straightforward demonstration of how forced edges can be used by illustrating a simple but effective divide & conquer algorithm for finding Hamiltonian Cycles in graphs which contain sparse cuts. This algorithm is generic in that it can be combined with almost any other Hamiltonian Cycle algorithm.

In Section 3.3, we further develop the ideas of forcing and pruning to define the concept of a ‘deletion sequence’, which we use to develop the Stone Carver’s Hamiltonian Cycle Algorithm, a simple yet effective search heuristic for finding Hamiltonian Cycles to be used primarily on graphs that are strongly expected to be Hamiltonian. We present a partial analysis of the algorithm together with empirical evidence supporting that its running time is $O(n^2)$. The goal of this research is to develop a simple algorithm for which it is possible to show it to be the fastest known proven algorithm for finding Hamiltonian Cycles in random regular graphs. In addition, we compare its running time with that of another well-established heuristic in order to establish its usefulness in practical terms.

In Section 3.4 we continue our study of deletion sequences more formally, and provide two results. The first is that regardless of their length, certain deletion sequences can be shortened to contain at most three edges. This appears to be a weak result, but is vindicated by the second result, which shows that the necessary length of deletion sequences in cubic graphs is unbounded.

Barnette’s Conjecture states that all planar, cubic, 3-connected, bipartite graphs are Hamiltonian, and has stood unresolved for many years. We refer to all such graphs as ‘Barnette graphs’. In Chapter 4 we strengthen Barnette’s Conjecture by proving that it holds if and only if every edge in every Barnette graph is part of some Hamiltonian Cycle. A stronger version may be easier to work with, so the goal of this research is to help settle the conjecture.

Finally, in Chapter 5 we discuss avenues of future research.

Chapter 2

Background

2.1 History

2.1.1 Early History

The Hamiltonian Cycle problem, also known as the Hamilton Circuit problem, has an interesting history. It is predated by two related problems. The first, now more than a thousand years old, is the **Knight's Tour** problem. It is a simple problem relating to the game of chess: given an (arbitrarily sized) chessboard and a knight, move the knight using only standard L-shaped moves such that it visits every square on the board exactly once, and finishes in the starting square. It is not hard to see that if each square is viewed as a vertex, and legal moves are viewed as edges, the Knight's Tour problem is equivalent to finding a Hamiltonian Cycle in the corresponding graph. Rouse Ball and Coxeter provide a good account of early results in [3]. Of course, Knight's Tours were not studied in terms of graph theory before recent times. This, however, is not true of the second related problem, which was solved in 1763 by the famous Swiss mathematician Leonhard Euler (1707-1783) in what may be the first formal use of graph theory [43]. The **Seven Bridges of Königsberg** problem asked if it is possible to cross all seven bridges in the city of Königsberg (modern day Kaliningrad) without using any bridge more than once. Euler proved the answer to be no, but his proof was much more general. An **Euler Circuit** in a multigraph G is a circuit which contains one instance of every edge in G, without explicit restrictions on how many times vertices may be visited. Euler showed that a connected multigraph is **Eulerian** if and only if each of its vertices has even degree.

2.1.2 Hamilton

Despite the widespread knowledge of these two problems, it took quite some time for the Hamiltonian Cycle problem to be formally discovered. It is named for the famous Irish Mathematician Sir William Rowan Hamilton (Please refer to Figure 2.1, below).

Figure 2.1: Sir William Rowan Hamilton (1805 - 1865)

By all accounts, Hamilton was a genius and a child prodigy. He started studying languages at an early age, and was already fluent in more than a dozen by the time he was 13 years old. Hamilton later switched his interest to mathematics and achieved many interesting results. His intellect was so compelling that in 1822 he was appointed Royal Astronomer of Ireland, quite a feat considering that he had not even finished his undergraduate degree yet. When he did graduate, he received top marks in both the arts and sciences, a very rare feat. Hamilton's greatest discovery was the quaternions, the first noncommutative algebraic system. His achievements led him to be knighted in 1835.

Interestingly enough, it was his ongoing interest in noncommutative systems that led to his discovery of Hamiltonian Cycles. Hamilton invented the 'Icosian Calculus', a noncommutative algebra so called because it involved a planar embedding of the graph of a dodecahedron, which has 20 vertices (Please refer to Figure 2.2, below).

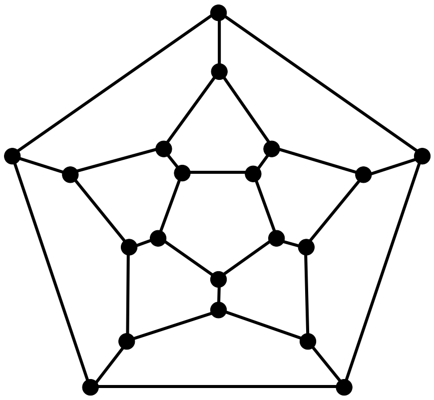

Figure 2.2: Planar Embedding Of A Dodecahedron

The system has two operations: L and R, standing for 'left' and 'right' respectively, the idea being that if one has just arrived at a vertex, one can choose to go left or right, with the value 1 being reserved for an expression which returns to one's point of origin. For example, a path that turns right twice and then left once can be expressed as the term R^2L. Similarly, since each face of a dodecahedron is pentagonal, we know that $R^5 = L^5 = 1$. Hamilton showed that symmetry notwithstanding, the equation $LLLRRRLRLRLLLRRRLRLR = 1$ defines the only Hamiltonian Cycle on a dodecahedron. Since $LR \neq RL$, the Icosian Calculus is clearly noncommutative. However, it is associative. For example, $(LR)L = L(RL)$. A more complete account can be found in [6].

According to [7], Hamilton's first communication about his Icosian Calculus was to his friend Robert Graves (See [26], page 71) in a letter dated Oct. 7th, 1856. Hamilton later published his ideas in [28] and [29]. Hamiltonian Cycles were not only discussed in academic circles, however. Perhaps even more relevant to their proliferation was the fact that Hamilton had his Icosian Calculus turned into a board game. Dubbed "The Icosian Game", it consisted of a board inscribed with the planar embedding of a dodecahedron, as well as twenty pegs. Each of the 20 vertices was inscribed with one of the 20 consonants (excluding y), and each peg was numbered from 1 to 20. (Please refer to Figure 2.3, below.)

Figure 2.3: Hamilton's Icosian Game

The game was played with two players; the first put down the numbers 1 through 5 to form a partial path, and the second player's job was to use the rest of the pegs to complete the cycle, thereby creating a complete tour. In the rule book, Hamilton suggested a number of alternate games that could be played. For example, another was for the second player to cover the board with a Hamiltonian path after the first placed three pegs. The complete rules including patent can be found in [7]. Hamilton sold the rights to the Icosian Game to a businessman for £25. This turned out to be a bad investment on the businessman's part, since it never sold very well. Hamilton also sold a second game called both “The Traveller’s Dodecahedron”, and “A Voyage Round The World”. This game had similar rules, but was played slightly differently. It consisted of a solid dodecahedron with pegs at the vertices representing famous cities ranging from Brussels to Zanzibar. The idea was to create a Hamiltonian Cycle or “voyage ’round the world” by looping a string around successive pegs.

Hamilton lived most of his life at Dunsink Observatory near Dublin. Unfortunately his later years were quite unhappy; his marriage was miserable, which was at least in part due to his alcoholism. Sadly, he died of gout at the age of 60, leaving behind mounds of unpublished research. He lived just long enough to learn that he had become the first foreign associate to be elected to the National Academy of Sciences of the United States.

2.1.3 Kirkman

It may surprise the reader to learn that Hamiltonian Cycles should not have been named after Hamilton at all. In fairness, they should be called ‘Kirkman Cycles’, or perhaps ‘Kirkmanian Cycles’ after Thomas Penyngton Kirkman, the man who *actually* first discovered them. (Please see Figure 2.4, below.)

Figure 2.4: Thomas Penyngton Kirkman (1806 - 1895)

A devoutly religious man, Kirkman was the rector of the small parish of Croft with Southworth for over 50 years. Indeed, he lived to be almost 90 years old, which is well beyond even today's average life expectancy. Elected to the Royal Society in 1857, Kirkman made dozens upon dozens of significant mathematical discoveries during his lifetime. His interest in polyhedra led him to discover Hamiltonian Cycles, a result which he published in [36]. This paper, received by the Royal Society on Aug. 6th, 1855, predates Hamilton's earliest communication, let alone his first publication on the subject, by more than a year. However, precedence is not the only argument on Kirkman's side. Whereas Hamilton considered only the one special case of cycles in the dodecahedron, Kirkman's result was much more general, because he pondered the existence of Hamiltonian Cycles in all graphs corresponding to planar embeddings of solid shapes. In addition, Kirkman was the first to discover an infinite class of non-Hamiltonian polyhedra. He showed that any bipartite graph with an odd number of vertices must be non-Hamiltonian. He gave an example of a planar, 3-connected, bipartite, non-Hamiltonian graph shown in Figure 2.5, below.

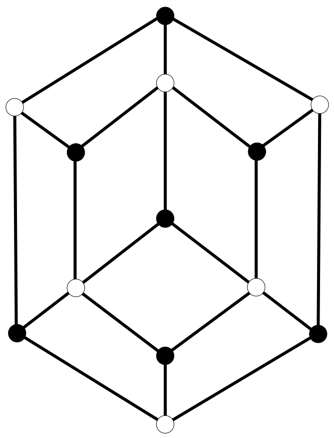

Figure 2.5: Kirkman's Planar, 3-Connected, Bipartite, Non-Hamiltonian Graph

In addition, it can be argued that Kirkman's paper dealt with graph theory which became a fruitful field of future research, whereas Hamilton's noncommutative algebra viewpoint ended up being a dead-end that has not yet led to any major results.

It is somewhat strange that Kirkman was not given credit for the discovery of Hamiltonian Cycles. Both men were from the same generation, and worked in the same geographical area; they both reported their findings to the Royal Society, and indeed even knew each other. According to [7], Hamilton visited Kirkman in 1861, and the two subsequently exchanged flattering letters, showing a mutual professional respect. Perhaps it was Hamilton's greater fame, or the marketing of his board game which upstaged Kirkman. Perhaps it was Kirkman's isolation, or a lack of self-promotion. Perhaps political issues within the Royal Society itself robbed Kirkman of the credit. More than likely it was a combination of these factors, and more. Unfortunately, this was not the only time that poor Kirkman went uncredited. It turns out that he also solved the problem of Steiner Triples more than half a decade before Steiner even formulated the problem [12], and similarly lost that claim to fame.

In any case, things may have turned out for the best. Many technical papers from and before Kirkman's era are poorly written by modern standards. Even so, reading Kirkman's paper is a particularly surreal experience. Graph theorists everywhere should be grateful that his terminology did not proliferate. Instead of using intuitive terms, Kirkman talks about p-edral q-acrons, walls, bases, summits, and constantly uses colourful adjectives such as sympolar, heteropolar, autopolar, syncral, etc. He concludes the paper by arguing that the word 'polyhedron' is misspelled, and should actually be spelled 'polyedron', justifying this argument by stating that the word 'periodic' is not spelled 'perihodic'.

2.1.4 Later History

The only other major interest in the Hamiltonian Cycle problem during its infancy was in connection to the much more famous 4-Colour Conjecture, which states that the countries on any geographical map can each be coloured with one of four colours such that no two adjacent countries have the same colour. This conjecture was widely accepted, but nobody seemed able to prove it as fact. Peter Guthrie Tait thought that he had discovered a short, elegant proof to the problem [31]. In what has become known as the 'Tait Conjecture', he stated that all planar, cubic, 3-connected graphs are Hamiltonian. Tait then argued that all such graphs are 3-Edge-Colourable by referring to their Hamiltonicity; simply colour the edges of a Hamiltonian Cycle alternately red and white, and then colour all non-cycle edges blue. He then appealed to a theorem which states the equivalence of 3-Edge-Colourability and the 4-Colour-Conjecture to complete the 'proof'. Tait's erroneous proof came in 1884 [45], and it did not take long for his assumption to be pointed out. However, more than 60 years passed before Tutte in 1946 [47] constructed the first counterexample to Tait's Conjecture (please refer to Figure 4.1), thereby finally concluding the early history of the Hamiltonian Cycle problem. The 4-Colour-Conjecture, on the other hand, was only finally proven in 1976 with the aid of computers [2].

For most of the 20th century, Hamiltonian Cycles remained a purely graph-theoretic entity. Researchers attempted without success to discover a simple characterization of Hamiltonian graphs that is both necessary and sufficient. They reasoned that since such a characterization exists for Eulerian graphs, one probably also exists for graphs that are Hamiltonian. Only with the birth of $\mathcal{NP}$-Completeness and its corresponding effect on how we view computational complexity did they finally concede that in all likelihood, this is impossible. The Hamiltonian Cycle problem was shown to be $\mathcal{NP}$-Complete by Richard Karp in 1972 [34], proving that it is solvable in polynomial time if and only if $\mathcal{P} = \mathcal{NP}$. Since then, Hamiltonian Cycles have been of interest to computer scientists as well, as will be described in the following section.

2.2 Interesting Results

Although it is tempting to include a complete survey of the problem, this is impractical. Literally hundreds of theorems concerning Hamiltonian Cycles have been published within various different contexts. These fall into three approximate categories: graph theoretic results, algorithmic results, and complexity-related results. Instead of providing a full survey, we will focus on results which are relevant to the research that is described in the next chapter.

2.2.1 Definitions

The following sections contain non-elementary definitions. Before continuing, it may be worthwhile to review some of them. We say that a graph is **1-tough** if it contains no set of k vertices that, if removed, would disconnect the graph into $k + 1$ or more components. A **k-connected** graph is one that cannot be disconnected by removing fewer than k vertices. A **k-vertex-cut** is a subset of k vertices that, if removed, would disconnect the graph, and a **k-edge-cut** is a similar subset of edges. An **r-regular** graph is one in which each vertex has degree r, and 3-regular graphs are typically referred to as **cubic**. Finally, a graph is said to be **Hamiltonian connected** if each pair of vertices has a Hamiltonian Path between them.

2.2.2 Graph Theoretic Results

Sufficient Conditions For Hamiltonicity

Graph theorists have published an incredible amount of research pertaining to Hamiltonian Cycles. Many sufficient conditions as well as necessary conditions for Hamiltonicity have been discovered. One of the most notable sufficient conditions is a theorem by Dirac [14]:

Theorem 2.2.1 *Let δ be the minimum degree of any vertex in a graph G. If $\delta \geq n/2$, then G is Hamiltonian.*

Bondy and Chvátal [8] strengthen this result with the following theorem:

Theorem 2.2.2 *If u and v are non-adjacent vertices in a graph G, and $d(u)+d(v) \geq n$, then G is Hamiltonian if and only if $G+u,v$ is Hamiltonian.*

The concept of the 'Bondy-Chvátal Closure' or 'B-C Closure' follows from this theorem. The idea of the B-C Closure is to take a graph G and recursively add edges between all non-adjacent vertices whose degrees sum to at least n until no such vertices remain. The resulting graph, which is sometimes complete, is called the B-C Closure of G. Although non-constructive, this technique can easily be turned into a constructive algorithm for finding Hamiltonian Cycles.

Necessary Conditions For Hamiltonicity

Unfortunately, the known sufficient conditions tend to be useful only in dense graphs. In contrast, the known necessary conditions for Hamiltonicity are much more useful from an algorithmic point of view, especially with respect to sparse graphs. We will therefore focus on necessary rather than sufficient conditions. The following rules constitute the major necessary conditions for Hamiltonicity:

Necessary Condition 2.2.1 *All Hamiltonian graphs are biconnected.*

Proof: Follows directly from the definition of Hamiltonian Cycle. □

Necessary Condition 2.2.2 *All Hamiltonian graphs are 1-tough.*

Proof: Suppose there exists some Hamiltonian graph G that is not 1-tough. Therefore G contains some Hamiltonian Cycle C as well as some cutset S of vertices of size q that, if removed, would disconnect G into p components where $p \geq q + 1$. Since C is a Hamiltonian Cycle, each vertex in S must be incident on exactly 2 edges of C. The edge-cut between S and the p components crosses exactly $2q$ edges. In other words, at least one of the p components has fewer than 2 of C's edges entering it, showing that C cannot be a Hamiltonian Cycle. This is a contradiction, as required. □

Example 2.2.1

The graphs in Figure 2.6 below are not 1-tough and therefore are not Hamiltonian. The relevant cutsets are indicated with arrows.

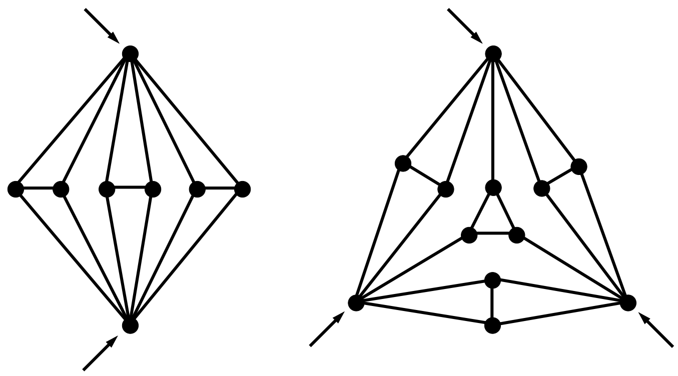

Figure 2.6: Examples Of Graphs Which Are Not 1-Tough

Necessary Condition 2.2.3 *Every Hamiltonian bipartite graph has bipartitions of equal size.*

Proof: Suppose there exists some Hamiltonian bipartite graph G with uneven bipartitions. Such a graph is not 1-tough because the removal of the smaller bipartition would disconnect the graph into too many components. Therefore, by Necessary Condition 2.2.2, G is non-Hamiltonian, which is a contradiction, as required. □

Necessary Condition 2.2.4 (Grinberg's Theorem [9]) *Every planar graph $G = (V, E)$ containing a Hamiltonian Cycle C satisfies the following equation:*

$$\sum_{i=3}^{n}(i-2)(f_i' - f_i'') = 0, \tag{2.1}$$

where f_i' represents the number of faces of degree i on the interior of C, and f_i'' represents the number of faces of degree i on the exterior of C.

Proof: (From [9]) Let E' denote the set of edges on the interior of C which are not part of C itself, and let $\varepsilon' = |E'|$. We know that the interior of C contains exactly $\varepsilon' + 1$ faces. Therefore,

$$\sum_{i=3}^{n} f_i' = \varepsilon' + 1 \tag{2.2}$$

Each edge in E' is incident on exactly two faces in the interior of C, and each edge of C is incident on exactly one face on the interior of C. Therefore,

$$\sum_{i=3}^{n} i f_i' = 2\varepsilon' + n \tag{2.3}$$

We can isolate ε' in equation 2.2 and substitute it into equation 2.3 to get

$$\sum_{i=3}^{n}(i-2) f_i' = n - 2 \tag{2.4}$$

This argument can be similarly applied to the exterior of C to get

$$\sum_{i=3}^{n}(i-2) f_i'' = n - 2 \tag{2.5}$$

To finish, we simply combine equations 2.4 and 2.5 to create equation 2.1, as required.

□

Example 2.2.2

Grinberg gave an example of a non-Hamiltonian graph that violates his theorem. It is shown in Figure 2.7, below.

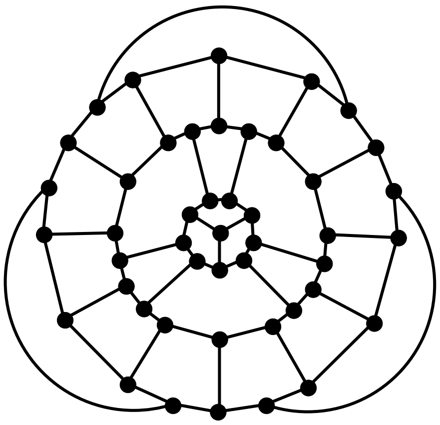

Figure 2.7: The Grinberg Graph

The Grinberg Graph contains multiple faces which have degrees 5 and 8, and a single face which has degree 9. We assume that the graph is Hamiltonian and apply Grinberg's theorem to get the following equation:

$$3(f_5' - f_5'') + 6(f_8' - f_8'') + 7(f_9' - f_9'') = 0 \tag{2.6}$$

Which implies that

$$3[(f_5' - f_5'') + 2(f_8' - f_8'')] = 7(f_9'' - f_9') \tag{2.7}$$

Since we have only one face of degree 9, we know that $f_9'' - f_9' = \pm 1$, so

$$3x = \pm 7, \tag{2.8}$$

where x is some integer. This is a contradiction, so the graph is not Hamiltonian, as required.

Other Graph Theoretic Results

The proof of the Hamiltonian Cycle problem's $\mathcal{NP}$-Completeness caused most graph theorists to abandon their attempts to unify necessary and sufficient conditions in the hope of finding a simple characterization for Hamiltonian graphs. However, they did not simply throw their hands in the air and abandon all Hamiltonian Cycle research. Instead, they prudently expanded the study of Hamiltonicity to restricted cases such as planar graphs, bipartite graphs, n-connected graphs, random graphs, etc., and have achieved some interesting results. For example, in [48], Tutte proves that every planar 4-connected graph is Hamiltonian. In [46], Thomassen extends this result by showing that every planar 4-connected graph is Hamiltonian connected. In [41], Robinson and Wormald prove that asymptotically speaking, almost all cubic graphs are Hamiltonian. Correspondingly, this shows that with high probability, any random 3-regular graphs is Hamiltonian. In [42] they extend this result to show that almost all r-regular (random) graphs are Hamiltonian for all r $\geq$ 3. A number of additional graph classes have been explored; please refer to Chapter 4 for more information.

Although no single source contains a complete survey of all graph theoretic results, an interested reader can easily access many partial surveys. Most modern graph theory textbooks contain a chapter on Hamiltonian Cycles, and some partial survey papers have been written. For example, reading [6], [9], [12], [31], and [51] together provides a fairly detailed exposition.

2.2.3 Algorithmic Results

The literature contains dozens of algorithms for finding Hamiltonian Cycles. Although there are some exceptions, Hamiltonian Cycle algorithms are usually search algorithms. Instead of simply determining whether a graph is Hamiltonian, most are designed to find a Hamiltonian Cycle as a certificate. However, similar certificates for non-Hamiltonicity are typically not given as output, reflecting the common belief that $\mathcal{NP} \neq \text{co}\mathcal{NP}$. In this case, most algorithms simply return the equivalent of 'no'. These many algorithms generally fall into three categories: polynomial-time algorithms for restricted inputs, polynomial-time heuristics, and exponential backtrack algorithms. This trichotomy reflects the $\mathcal{NP}$-Completeness of the problem. Much like modern graph theorists, some algorithmists concede that the problem is difficult, so it makes sense to look at restricted inputs. Others are not daunted by the difficulty but know when to give up and so design heuristics. Others yet are more stubborn and are willing to accept exponential running times in the worst case.

Algorithms For Restricted Inputs

Researchers have published many interesting Hamiltonian Cycle algorithms for restricted inputs. For example, it should be easy to see that if every vertex has degree 2, then the input is simply a 2-factor, which is clearly solvable in linear time [38]. Similarly, most algorithms perform well on complete graphs, which are clearly all Hamiltonian. In many cases, researchers provide corresponding algorithmic results for an earlier graph theoretic results. For example, in [25], Gouyou-Beauchamps puts Tutte's proof that all planar 4-connected graphs are Hamiltonian [48] to practical use by providing a corresponding $O(n^3)$ search algorithm. This result was later improved by Chiba and Nishizeki in [11]. They give an algorithm for solving 4-connected planar graphs in linear time, a surprisingly efficient result. In [19], Frieze gives an $O(n^3 log(n))$ expected time algorithm for finding Hamiltonian Cycles in random r-regular graphs for $r \geq 85$. In [20], Frieze et al. improve this result by giving an $O(n^6)$ expected time algorithm for $r \geq 3$. This corresponds to the theoretic result in [42] by Robinson & Wormald.

Heuristics & Exponential Algorithms

The distinction between the categories of Hamiltonian Cycle algorithms is more for taxonomical purposes than anything else. Indeed, absolute categorization is often misleading. Firstly, it is difficult to define exactly what a heuristic is. Heuristics are general procedures, so by the Church-Turing Thesis, one can argue that all heuristics are also algorithms. Generally, heuristics are given extra leeway in that they are not always expected to return the correct answer. For example, some Hamiltonian Cycle heuristics search for a certain number of steps, and if they are still unsuccessful, they return some value indicating failure rather than Hamiltonicity or non-Hamiltonicity. Secondly, many algorithms in the three groups are easily interchangeable given only minor modifications. For example, most of the algorithms for restricted inputs could be converted into heuristics by simply adding a counter that causes the algorithm to halt after some number of search steps. These could then be run on graph classes for which they were not specifically designed. Likewise, it would not be hard to attach an exponential backtrack tree to most heuristics in order to turn them into true algorithms. Furthermore, most of these algorithms and heuristics can be randomized and tweaked in various other minor ways to make them more or less effective.

There seems to be a stigma against exponential backtrack algorithms because of their poor running times. However, labelling backtrack algorithms as somehow inferior simply because they have worst-case exponential running times is inappropriate. Firstly, unless $\mathcal{P} = \mathcal{NP}$, we will have to settle for exponential runtimes. Furthermore, even the most naive exponential algorithm may do very well on certain input classes. For instance, consider the algorithm which simply looks at all possible permutations of the vertices, and each time checks if that permutation corresponds to a cycle in the graph. With an $O(n!)$ running time, it would be difficult to design a less efficient algorithm without becoming creative. However, this algorithm performs better than some others when given very dense graphs as input.

In any case, the literature contains too many algorithms for us to do them all justice in this exposition. Since they are not particularly relevant to our later results, an interested reader should consult Basil Vandegriend's Master's thesis [50]. He describes many Hamiltonian Cycle algorithms in fair detail and discusses general Hamiltonian Cycle algorithm design techniques.

2.2.4 Complexity-Related Results

After Karp's 1972 proof [34] that the Hamiltonian Cycle problem is $\mathcal{NP}$-Complete, a number of other researchers refined his result to include more restricted graph classes. The first was Krishnamoorthy, who in 1975 proved that the Hamiltonian Cycle problem remains $\mathcal{NP}$-Complete even when the input is restricted to bipartite graphs [37]. In 1976, Garey, Johnson, and Tarjan collaborated to prove that it remains $\mathcal{NP}$-Complete even when restricted to planar, cubic, 3-connected graphs with no face having fewer than 5 sides [22]. Also in 1976, Papadimitriou and Steiglitz showed that it remains $\mathcal{NP}$-Complete even when a Hamiltonian-Path is given as part of the instance [40]. In 1980 Akiyama, Nishizeki, and Saito published a paper [1] in which they showed that the Hamiltonian Cycle problem remains $\mathcal{NP}$-Complete for two classes of graphs: planar, cubic, 2-connected, bipartite graphs, as well as cubic, 3-connected, bipartite graphs. The first of these two classes is of special interest because it is so similar to the class from Barnette's Conjecture (See Chapter 4). More recently, Chvátal [12] and Wigderson [52] independently proved that the problem remains $\mathcal{NP}$-Complete when restricted to triangulations (maximal planar graphs). Interestingly, although planar 4-connected graphs can be solved in linear time (see Subsection 2.2.3), when restricted to 4-connected, 4-regular graphs, the problem once again becomes $\mathcal{NP}$-Complete [44]. On a related note, Karp also proved that the Directed Hamiltonian Cycle problem is $\mathcal{NP}$-Complete [34]. Various results for restricted versions of this problem are also known [21]. The corresponding directed and undirected versions of the Hamiltonian-Path problem are also $\mathcal{NP}$-Complete [21].

Karp reduced from Vertex Cover to Hamiltonian Cycles by converting formulas to graphs using 'widgets', or intermediate graph components which enforce certain properties. Many of the subsequent reductions are from 3-SAT, which is slightly easier to manipulate, with different widgets being used in various clever ways.

Chapter 3

Algorithmic Results

3.1 Preliminaries

The remaining sections constitute the author's original contributions to the field, and contain results which rely heavily on certain notions and terminology which are supplied in this section as preliminaries. The ideas of 'forced edges' and 'pruning' are particularly important. In addition, the 'Look-Ahead Theorem' as well as the additional necessary conditions for Hamiltonicity given here will resurface later.

These ideas are not new, but this is the first time that they have been combined and listed together under the context of finding Hamiltonian Cycles. It should be emphasized that these ideas 'fit' together very naturally, and that they work particularly well when used in concert.

3.1.1 Forced Edges & Pruning

The key idea in many of the following results is the notion of a 'forced edge'. A forced edge in a graph G is an edge that is marked as having to be included in every Hamiltonian Cycle of G, and we therefore define a Hamiltonian Cycle in a graph with forced edges to be one which contains each of the forced edges. Note that forcing an edge which belongs to no Hamiltonian Cycles will lead to the incorrect conclusion that a Hamiltonian graph is non-Hamiltonian. Forced edges should therefore generally be viewed as constraints which can reduce the number of possible Hamiltonian Cycles which can be found. However, not all forced edges should be viewed as constraints. We say that a forced edge in a graph G is 'safely forced' if it is marked as forced but also happens to necessarily be part of every Hamiltonian Cycle in G. In other words, the act of forcing such an edge does not reduce the number of Hamiltonian Cycles in G, but rather serves the role of providing information about G that may be very useful. For example, both edges incident on any degree-2 vertex can be safely forced. Even non-Hamiltonian graphs can have safely forced edges, and in this case it still means that these edges are (vacuously) part of every Hamiltonian Cycle. We indicate forced edges by drawing hash marks across them, as shown in Figure 3.1 below.

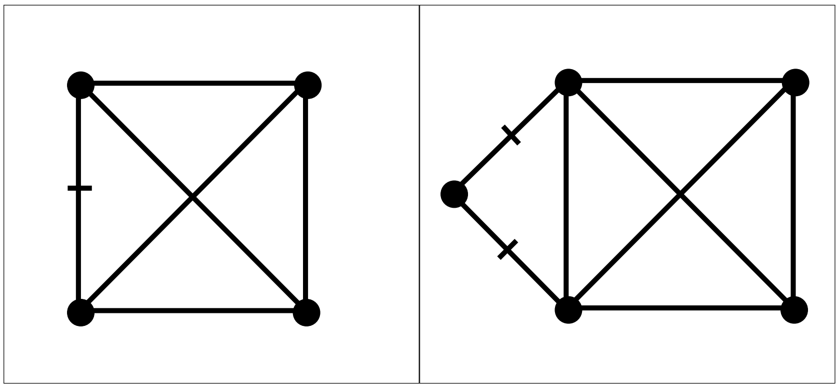

Figure 3.1: The Graph On The Left Contains A Forced Edge, Whereas The Graph On The Right Contains Two (Safely) Forced Edges.

Since the Hamiltonian Cycle problem is so difficult, any extra information about a graph that can be obtained is very precious. Safely forced edges are such a source of information, and should therefore be exploited whenever possible, because they can be used to determine Hamiltonicity, as will be shown in subsection 3.1.2.

In contrast with safely forced edges, an edge in a graph which is necessarily part of no Hamiltonian Cycle may be safely deleted. We use the term 'pruning' to refer to the act of safely deleting an edge. Pruning can be viewed as 'cutting the fat' out of a graph, and much like forcing is an important tool for determining Hamiltonicity.

On a related note, we say that a 'chain' is a path containing q vertices subject to the following constraints: $2 \leq q < n$, and each of the $q-1$ edges along the path is forced, the 2 endpoints each have degree at least 3, and the remaining $q-2$ vertices, if any, have degree of exactly 2. These endpoints are referred to as 'heads' of the chain. (Please refer to Figure 3.2, below.)

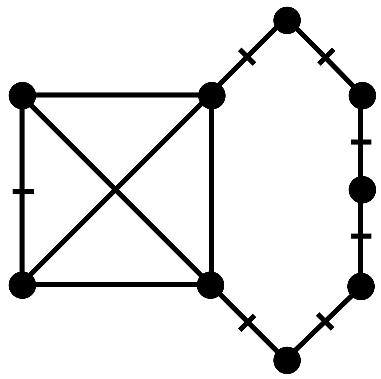

Figure 3.2: A Graph Containing Two Chains

3.1.2 Necessary Conditions Involving Forced Edges

In addition to the necessary conditions for Hamiltonicity given in Subsection 2.2.2, the notion of a forced edge leads us to a number of other necessary conditions. Our first necessary condition pertains to edge-cuts across forced edges. We say that an 'odd-forced-cut' is a cut across an odd number of edges, all of them forced. Figure 3.3 below illustrates a graph containing such cuts.

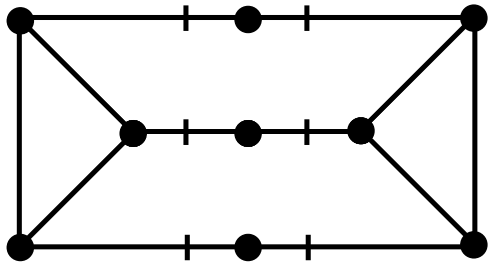

Figure 3.3: A Graph Containing 8 Odd-Forced-Cuts

Necessary Condition 3.1.1 *No Hamiltonian graph contains an odd-forced-cut.*

Proof: The odd-forced-cut separates the graph into two components. Since the cut contains an odd number of forced edges, though, the forced edges have no hope of contributing to a cycle. At best, they might be part of a Hamiltonian Path that has one end in each component. $\square$

Corollary 3.1.1 *Given any p-edge-cut in a graph, any Hamiltonian Cycle contains q edges crossing that cut, where* $2 \leq q \leq p$ *and* q *is even.*

A 'hub', shown in Figure 3.4 below, is a vertex which is incident on 3 or more forced edges.

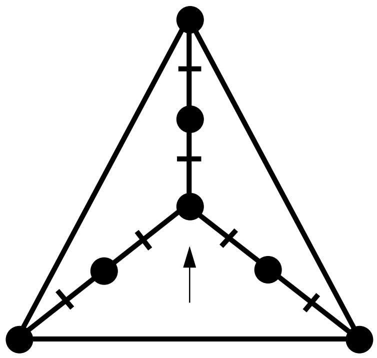

Figure 3.4: A Graph Containing A Hub

Necessary Condition 3.1.2 *No Hamiltonian graph contains a hub.*

Proof: Every vertex in a Hamiltonian Cycle has exactly two Hamiltonian edges incident on it. A hub therefore violates the very definition of Hamiltonian Cycle. □

It is tempting to somehow prove that this Necessary Condition follows as a corollary from Necessary Condition 3.1.1, but hubs can also contain an even number of forced edges, which does not constitute an odd-forced-cut.

Finally, we say that a forced edge incident on the two vertices of a 2-vertex-cut is called a 'barricade', shown below in Figure 3.5.

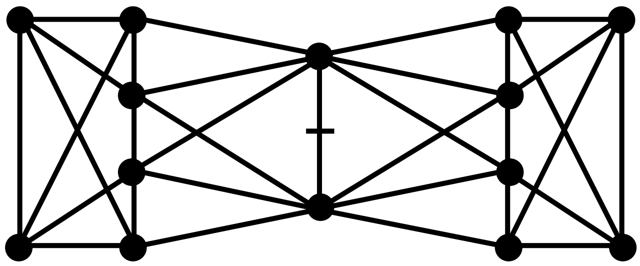

Figure 3.5: A Graph Containing A Barricade

Necessary Condition 3.1.3 *No Hamiltonian graph contains a barricade.*

Proof: Let us label the two endpoints of the barricade x and y. The barricade guarantees that exactly one additional Hamiltonian edge will leave each of x and y. Therefore the forced path through x and y cannot lie on a Hamiltonian Cycle, since x, y forms a 2-cut. □

3.1.3 The Look-Ahead Theorem

With respect to Hamiltonian Cycles, it is the edges of a graph which are of paramount importance. Any given edge falls into exactly one of three groups: edges which are in all Hamiltonian Cycles, edges which are in some Hamiltonian Cycles, and edges which are in no Hamiltonian Cycles. Edges in the first and third groups can be identified using the following theorem:

Theorem 3.1.1 (Look-Ahead Theorem) *If a graph is non-Hamiltonian after deleting an edge, then that edge can be safely forced. Likewise, if a graph is non-Hamiltonian after forcing an edge, then that edge can be safely pruned.*

Proof: If forcing an edge causes the graph to become non-Hamiltonian, it cannot be part of any Hamiltonian Cycle, and therefore may be pruned. Similarly, if deleting an edge causes the graph to become non-Hamiltonian, then it must be part of every Hamiltonian Cycle, and therefore may be forced. □

The Look-Ahead Theorem is so called because the act of testing an edge by forcing it or deleting it is equivalent to a common algorithmist's trick, namely 'looking ahead' one step. Of course, with many graphs one step is not nearly sufficient to show that it can be safely forced or pruned. Nonetheless, even with one step, the Look-Ahead Theorem can be quite useful. For example, it provides some interesting corollaries:

Corollary 3.1.2 *Any edge added to a non-Hamiltonian graph can be safely forced.*

Corollary 3.1.3 *Every edge in a non-Hamiltonian graph can be safely forced.*

Corollary 3.1.4 *Every edge in a non-Hamiltonian graph can be safely pruned.*

The Look-Ahead Theorem is obviously not limited to looking ahead by just one step. It is possible to use an arbitrary number of iterations of the Look-Ahead Theorem in order to create powerful compound necessary conditions for non-Hamiltonicity. This forms the basis for a potentially powerful proof system for non-Hamiltonicity. We will not explore this topic in this thesis, but rather suggest that this is an interesting area for future research.

3.1.4 Algorithmic Impact

When used in concert with the Look-Ahead Theorem, the known necessary conditions for Hamiltonicity in Subsections 2.2.2 and 3.1.2 provide us with a rich assortment of algorithmic tools. The following rules represent some of these tools. They can be considered as 'algorithmic rules', since their wording directly suggests a course of action. Naturally, these rules are all meant to apply only within the context of searching for Hamiltonian Cycles.

Action Rule 3.1.1 *Both edges across any 2-edge-cut can be safely forced.*

Proof: Let us call these two edges e_1 and e_2. If we delete e_1, we are left with a graph that is not biconnected, and therefore non-Hamiltonian by Necessary Condition 2.2.1. By the Look-Ahead Theorem, we may therefore safely force e_1. We may force e_2 for the same reason. □

This rule also leads to a very useful corollary:

Corollary 3.1.5 *Both edges incident on a degree-2 vertex can be safely forced.*

Action Rule 3.1.2 *Suppose a graph contains a cut across q edges, and $q - 1$ of those edges are forced. If q is even, then the remaining edge can be safely forced. If q is odd, then the remaining edge can be safely pruned.*

Proof: Follows from Necessary Condition 3.1.1 together with the Look-Ahead Theorem. □

Action Rule 3.1.3 *If a graph contains an unforced edge between the two vertices of a two-vertex-cut, then that edge can be safely pruned.*

Proof: Follows from Necessary Condition 3.1.3 together with the Look-Ahead Theorem. □

Action Rule 3.1.4 *If a vertex of degree at least 3 has exactly two forced edges incident on it, then all of the remaining edges incident on it can be safely pruned.*

Proof: Follows from Necessary Condition 3.1.2 together with Look-Ahead Theorem. □

In practice, Rule 3.1.4 is perhaps the single most useful pruning rule. A single application can prune many edges, revealing a number of degree-2 vertices. By Corollary 3.1.5, the edges incident on these vertices can be safely forced. These resulting forced edges often set up a number of other vertices where Rule 3.1.4 can once again be applied, and so on. This type of pruning is therefore highly effective, especially in sparse graphs. In many cases, cascading iterations will solve the entire graph quite effectively. An example of this is given in Figure 3.6, below.

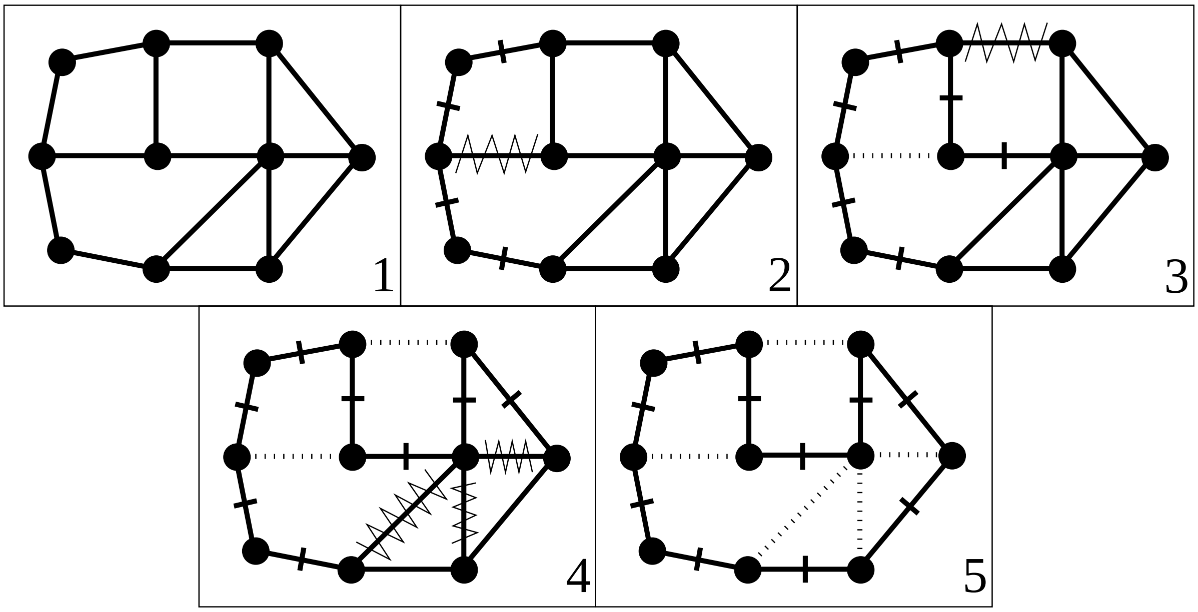

Figure 3.6: Repeated Applications Of Corollary 3.1.5 & Rule 3.1.4

Action Rule 3.1.5 *If a graph contains a chain which has an unforced edge between its two heads, then that edge may be safely pruned.*

Proof: Since a chain can never contain all of the vertices in a graph, if we were to force the edge between the chain heads, the chain would form a closed cycle that is not a Hamiltonian Cycle. Each of its vertices would have exactly 2 forced edges incident on it, so repeated applications of Rule 3.1.4 would disconnect this cycle from the rest of the graph. Since this graph is not biconnected, it is not Hamiltonian. The act of forcing an edge therefore created a graph which is obviously non-Hamiltonian, so the Look-Ahead Theorem allows us to safely prune this edge, as required. □

In practice, Rule 3.1.5 is also an effective pruning rule. An example of its application is given in Figure 3.7, below.

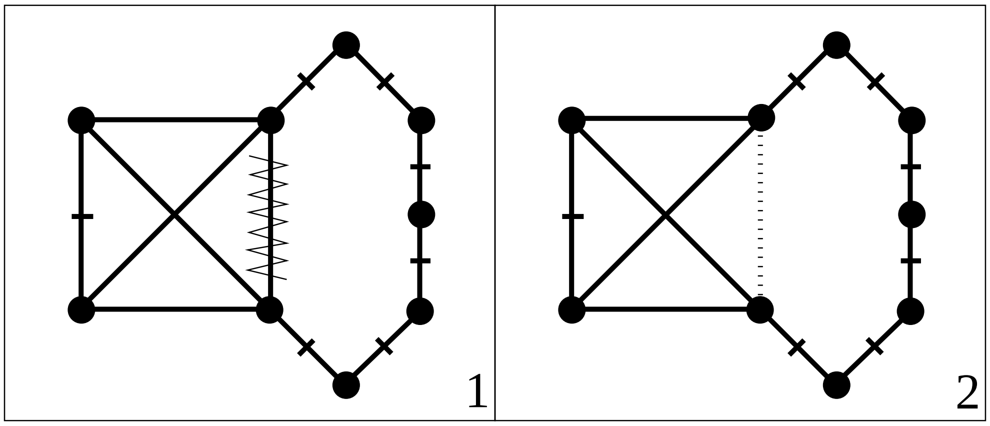

Figure 3.7: An Application Of Rule 3.1.5

In some cases, we can exploit the symmetry of a graph in order to determine whether it is Hamiltonian. More specifically, we may prune edges based on symmetry. For example, this principle can be applied to the Petersen Graph in Figure 3.8, below.

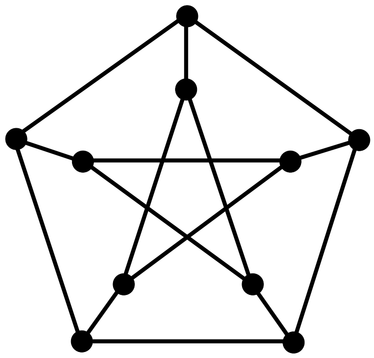

Figure 3.8: The Petersen Graph

The Petersen graph consists of an inner star separated from an outer pentagon by a 5-edge-cut. We know from Corollary 3.1.1 that we may delete one of these edges. Usually this is a problem, because we do not necessarily know which edge to delete. However, in this case, symmetry guarantees that regardless of which edge we choose, we will be left with one of five isomorphic graphs. We can therefore proceed and delete any one of the edges. This principle can be similarly applied to the Tutte Graph, shown in Figure 4.1. Deleting any arbitrary edge incident on its central vertex would leave us with one of three isomorphic graphs, showing that any one of the central edges can be safely pruned.

More formally, this technique can be stated as follows:

Action Rule 3.1.6 *If a graph contains a p-edge-cut, p odd, and each of the p graphs resulting from deleting one of these edges is isomorphic to the others, then we may safely prune any single one of the edges in the cut.*

Proof: We know from Corollary 3.1.1 that one of the p edges must be deleted, and that removing this edge will not reduce the number of Hamiltonian Cycles in the graph. Because of symmetry, it does not matter which of the edges we delete, as required. $\square$

Unfortunately, the graph isomorphism problem is in $\mathcal{NP}$, but is not known to be in $\mathcal{P}$, and is not known to be $\mathcal{NP}$-Complete, which prevents a proper analysis. Nevertheless, this area may yield future results, since there may be special cases which are exploitable in polynomial time.

This is by no means an exhaustive list of necessary conditions, pruning rules, or forcing rules. The literature contains additional rules that are useful under very specific circumstances. Indeed, we can expect many more rules to be developed in the future. Of course, the effectiveness of any Hamiltonicity theorems and rules depends largely on the graph on which they are applied. Implementations of these rules also have various different running times. For example, Necessary Condition 2.2.1 tells us to check for biconnectivity, which is widely known to be solvable using depth first search, which has an $O(n+m)$ running time. In contrast, Necessary Condition 2.2.2 tells us to check if the graph is 1-tough, which is $\mathcal{NP}$-Hard for cubic graphs [5], and Rule 3.1.6 deals with a problem whose difficulty is not yet understood. Therefore, an algorithmist must weigh poor running times with the effectiveness of executing a certain rule when deciding which ones to include in an algorithm. In Section 3.3 we put some of these rules to use in order to develop a fast Hamiltonian Cycle algorithm.

3.2 Divide & Conquer Algorithm

3.2.1 Overview

This section serves a twofold purpose. Firstly, it describes a generic divide & conquer algorithm for finding Hamiltonian Cycles that can be combined with most other algorithms. Secondly, this algorithm makes simple use of the forced edges which were described in Section 3.1, and therefore lends some practical intuition to a concept which will become more complicated in subsequent sections.

3.2.2 Algorithm Description

Using a divide & conquer strategy is an effective algorithmic technique that has been applied to many different problems. It turns out that this can also be applied in order to help determine whether or not a given graph is Hamiltonian. This is a more general technique that can be used to augment other search algorithms. It should therefore be viewed more as a collaborative technique rather than a substitute for them.

Determining whether a graph is Hamiltonian by using a divide & conquer approach is only effective under special conditions, namely when a graph contains a relatively sparse edge-cut separating it into two halves which are approximately equal in size. For example, if the graph contains a 2-edge cut as shown in Figure 3.9 below, then it is a perfect candidate. Technically speaking, such graphs are not necessarily 'sparse' by the standard definition, but sparse graphs are typically much more conducive to the presence of sparse cuts than are dense graphs, so the divide & conquer technique should be seen as an algorithm for sparse graphs.

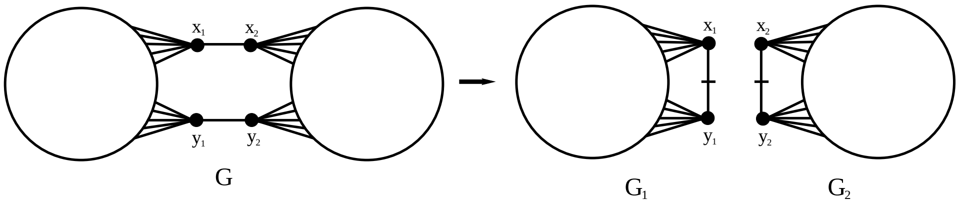

Figure 3.9: Divide & Conquer Being Applied Across A 2-Edge Cut

Note that any graph with a 2-edge-cut also contains a number of 2-vertex cuts. Any edges between vertices x_1 and y_1, x_1 and y_2, x_2 and y_1, or x_2 and y_2 could therefore be pruned by appealing to Rule 3.1.3, and hence they are not shown. Suppose we were to delete the edges of the cut to disconnect the graph into two components. Then the entire graph G contains a Hamiltonian Cycle if and only if its left component contains a Hamiltonian Path between x_1 and y_1, and its right component contains a Hamiltonian Path between x_2 and y_2. Therefore, if we force these edges as indicated in graphs G_1 and G_2, we know that G is Hamiltonian if and only if both G_1 and G_2 are Hamiltonian, suggesting that this style of divide & conquer is prudent. We may now use any available algorithms to solve G_1 and G_2.

We may similarly use this technique across a 2-vertex-cut, as shown below in Figure 3.10, below.

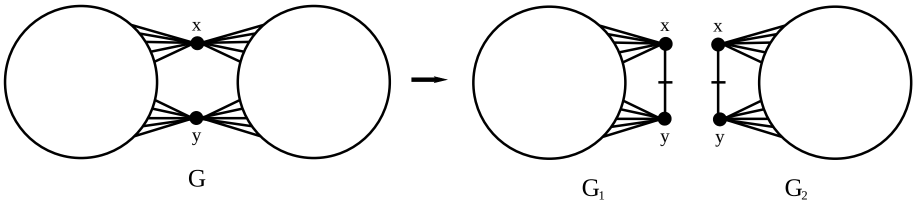

Figure 3.10: Divide & Conquer Being Applied Across A 2-Vertex Cut

The only difference here is that the vertices x and y are copied so that they appear in both G_1 and G_2.

This divide & conquer technique is not limited to 2-edge-cuts and 2-vertex-cuts, however. As shown below in Figure 3.11, it can also be applied across 3-edge-cuts, albeit at a price.

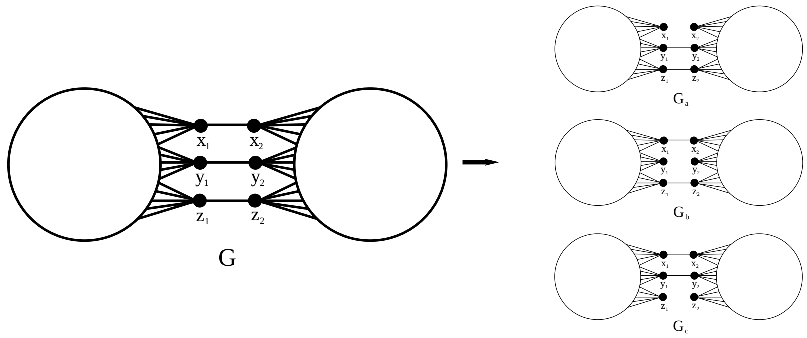

Figure 3.11: Divide & Conquer Being Applied Across A 3-Edge Cut

We know from Corollary 3.1.1 that any Hamiltonian Cycle in G crosses exactly 2 of the edges in the cut. We may therefore create three copies of G, one corresponding to each of the three cases where one of the edges across the cut has been removed. Each of these graphs contains a 2-edge-cut, and may therefore also be divided and conquered.

As the cuts become more complicated, dividing & conquering yields diminishing returns. For example, a 3-vertex-cut requires us to make 6 copies of the graph which may each then be treated using the 2-vertex-cut divide & conquer technique. In general, the number of divided graphs necessary increases exponentially with the size of the cut, regardless of whether it is an edge-cut or a vertex-cut, showing that this technique indeed is only useful given small cuts.

3.3 The Stone Carver's Hamiltonian Cycle Algorithm

3.3.1 Introduction

In this section, we describe the Stone Carver's Hamiltonian Cycle algorithm (SCHCA), a randomized heuristic search algorithm for finding Hamiltonian Cycles in Hamiltonian graphs which implements some of the ideas given in Section 3.1. In Subsection 3.3.2, we start off by describing its main subroutine, the 'Probability Mill'. Since the SCHCA is a very natural algorithm, its pseudocode, given in Subsection 3.3.3, is very simple and requires only a few lines. The SCHCA amounts basically to repeatedly calling the Probability Mill subroutine until a Hamiltonian Cycle is found.

Normally, after an algorithm is described, an analysis of its running time is given. We have been able to prove the running time of the Probability Mill subroutine, but a complete analysis depends very much on the class of graphs that is being used as input. For the purposes of this research, we are particularly interested in sparse random regular graphs. Unfortunately, we have not yet been able to prove the running time of the SCHCA on these graphs, which is why we refer to it as a heuristic. In the absence of such a proof, we give two alternate measures of its effectiveness. The first such measure, given in Subsection 3.3.4, is empirical evidence illustrating its running time. The second, given in Subsection 3.3.5, is an empirical comparison between the SCHCA and a heuristic which has already been established as working well.

It turns out that the SCHCA is not the fastest heuristic available for finding Hamiltonian Cycles in sparse Hamiltonian graphs. However, the goal of this research is not to create a fast practical Hamiltonian Cycle algorithm, but rather to develop a simple algorithm for which it might be possible to prove the running time on random regular graphs. Empirical evidence suggests that the SCHCA has a running time of $O(n^2)$ on random regular graphs, which, if proven, would be a considerable improvement over the current best, $O(n^6)$ [20]. As stated in Subsection 3.3.6, the ultimate future goal of this research is to prove this running time.

The SCHCA is designed to be used on graphs that are Hamiltonian, and fares poorly when given non-Hamiltonian graphs as input. It will only give the correct answer on very simple cases of non-Hamiltonian graphs, whereas complicated ones will cause it to run indefinitely. We are interested primarily in solving sparse random regular graphs. As already stated in Section 2.2, any graph chosen from this class is almost surely Hamiltonian, so we need not worry about infinite running times.

The SCHCA is unique in how it works. Many Hamiltonian Cycle algorithms such as HAM, SparseHAM, MultiPath, and others work by selecting edges and building partial paths in the hope of forming a Hamiltonian Cycle [50]. The SCHCA works in exactly the opposite way; instead of selecting edges to include, it selects edges to exclude, and deletes them. Incidentally, this is the source of its name. A quote attributed to Michelangelo states that every block of marble contains a statue, and that it is the carver's job to 'liberate' it by cutting away the excess pieces. Similarly, the SCHCA deletes the extra edges in a graph in order to 'liberate' a Hamiltonian Cycle. Although it is not unheard of for Hamiltonian Cycle algorithms to be randomized, this is another distinguishing characteristic of the SCHCA.

3.3.2 The Probability Mill Subroutine

Probability Mill is the subroutine that performs the actual edge deletions, and is called repeatedly by the SCHCA. Because it is destroying the graph, we only run this subroutine on a backup copy of the graph. Probability Mill's purpose is to execute a 'deletion sequence' on a graph. A deletion sequence is a sequence of edges that are deleted in order. The resulting graph depends very much on the order in which the edges of a deletion sequence are removed, showing that 'deletion sets' are insufficient for our needs. After each edge is deleted, we apply some subset of the pruning and forcing rules described in the previous sections. Edges deleted by pruning are not considered to be part of the deletion sequence. In effect, any edge that appears in a deletion sequence may be unsafe in that its deletion may reduce the number of possible Hamiltonian Cycles in the graph. In contrast with these 'Probability Mill deletions', pruning deletions are perfectly safe.

Probability Mill is randomized in that it executes random deletion sequences on the graph. The rationale for its randomization is quite simple; Probability Mill deletions are unsafe and may destroy any chance of finding a Hamiltonian Cycle. In other words, any single deletion within a deletion sequence may be a critical mistake. This mistake may come at any point in the sequence, but more importantly, such mistakes are not necessarily detectable. Therefore, if Probability Mill were repeatedly called in a sequential, depth-first manner that filled out a search tree, a mistake early in the very first deletion sequence executed would guarantee that the algorithm wastes an incredible amount of time hopelessly searching a potentially enormous subtree nested below that mistake. Randomization avoids this problem by allowing deletion sequences to 'warp' out of dangerous areas of the search tree. Although this step sacrifices the completeness of the algorithm in a computational sense, empirical evidence confirms that the running time is improved substantially.

The only decision to be made with respect to the Probability Mill subroutine is to determine which pruning and forcing rules to run. It should be intuitively obvious that without any pruning rules, the running time of Probability Mill is $O(m)$ (after all, we can only delete a maximum of m edges). However, it turns out that if we are somewhat clever, we can include some powerful pruning rules without increasing the running time. Intuitively, the strategy here is to include as many pruning rules as possible into the

Probability Mill subroutine without increasing its asymptotic running time so that we strike a balance between its effectiveness while maximizing the number of times it can be executed.

We can include Corollary 3.1.5, which allows us to force both edges incident on any degree-2 vertex. We can also include Rule 3.1.4, which allows us to look at any vertex with exactly two forced edges incident on it, and safely prune all of its other unforced edges. We shall henceforth refer to this pruning rule as 'Lightning', because it works so quickly and effectively. Finally, we can include Theorem 3.1.5, which allows us to safely prune any unforced edge incident on both heads of a chain. We will therefore refer to this pruning rule as 'Head Hunter'.

The only extra data structure that we need is an array of chain data structures, each of which keeps track of its two heads as well as how many vertices it contains. Conceptually, each chain data structure should be thought of as a constant-sized representation of an arbitrarily long chain. The maximum number of chains in a graph is bounded by $O(n)$. In order to minimize the time spent manipulating chains, we assume that our graph is represented using adjacency lists, but that the degree of each vertex is bounded by a constant (as is the case with sparse random regular graphs).

Chains play an important role in ensuring that Probability Mill runs quickly. They can be easily created, deleted, merged and closed to form cycles in $O(1)$ time. They allow us to choose a random edge to be deleted in $O(1)$ time. Part way through our deletion sequence, our graph will contain many forced edges. Instead of searching for unforced edges, we want to be able to find a random, unforced edge more quickly, so we first choose a chain at random, and then randomly choose an edge incident on one of that chain's heads. In addition, chains allow us to execute our Head Hunter pruning rule in $O(1)$ time; every time a chain is modified, we check to see if there is an unforced edge incident between its two heads. If so, we can safely prune it using Head Hunter.

Before running Probability Mill, we assume that all possible chain data structures have first been built. If the graph contains no chains, then we repeatedly delete edges at random from the graph until at least one chain has been created, at which point we switch over to randomly deleting edges from chain heads. Every time a deleted edge creates a degree-2 vertex, we immediately force its two edges in $O(1)$ time. This may create chains, extend chains, or merge chains. We look at the endpoints of any newly-created chain and see if Lightning or Head Hunter apply. If so, we enter the relevant vertices into our 'pruning queue', which is an ordinary fifo queue. We mark these vertices so that they cannot have multiple instances on the queue. The pruning queue is then executed by removing the vertices and deleting the appropriate edges from their adjacency lists. Other than their safety, Lightning and Head Hunter deletions are otherwise the same as normal Probability Mill deletions; they can create degree-2 vertices, affect chains, and put other vertices onto the pruning queue. Likewise, the pruning queue will not add anything to our running time, because all actions associated with it are $O(1)$ operations, and every pruning deletion simply means that we will have to perform one fewer Probability Mill deletion. In accordance with the idea of a deletion sequence, we repeatedly apply a random deletion, followed immediately by an execution of the pruning queue, provided that it is non-empty. We know that the Probability Mill subroutine has completed an iteration when one of its termination conditions has been met.

Probability Mill has only three termination conditions; one represents success, and the other two represent failure. In addition to allowing us to perform a number of actions in $O(1)$ time, chains also allow us to know when a Hamiltonian Cycle has been found; if a chain closes upon itself to form a cycle, we check to see if it contains all of the vertices of the graph in $O(1)$ time. If not, then we have found a disjoint cycle, which is one of our two failure conditions because it shows that the graph is disconnected, which is a special case of Necessary Condition 2.2.1. The other failure condition is a hub, as described by Necessary Condition 3.1.2. Hubs can be detected in $O(1)$ time because we simply need to check the endpoints of any edge that we force. When any one of these conditions is met, Probability Mill immediately terminates.

Example 3.3.1

Figure 3.12.1 below illustrates a graph before any edges have been deleted. Figure 3.12.2 shows this graph after the deletion sequence (e_1, e_2) has created a hub. Figure 3.12.3 shows the graph after the deletion sequence (e_2, e_3) has created a disjoint cycle. Finally, Figure 3.12.4 shows the Hamiltonian Cycle found by deletion sequence (e_4, e_5). In all three cases, notice the edges that were deleted by pruning.

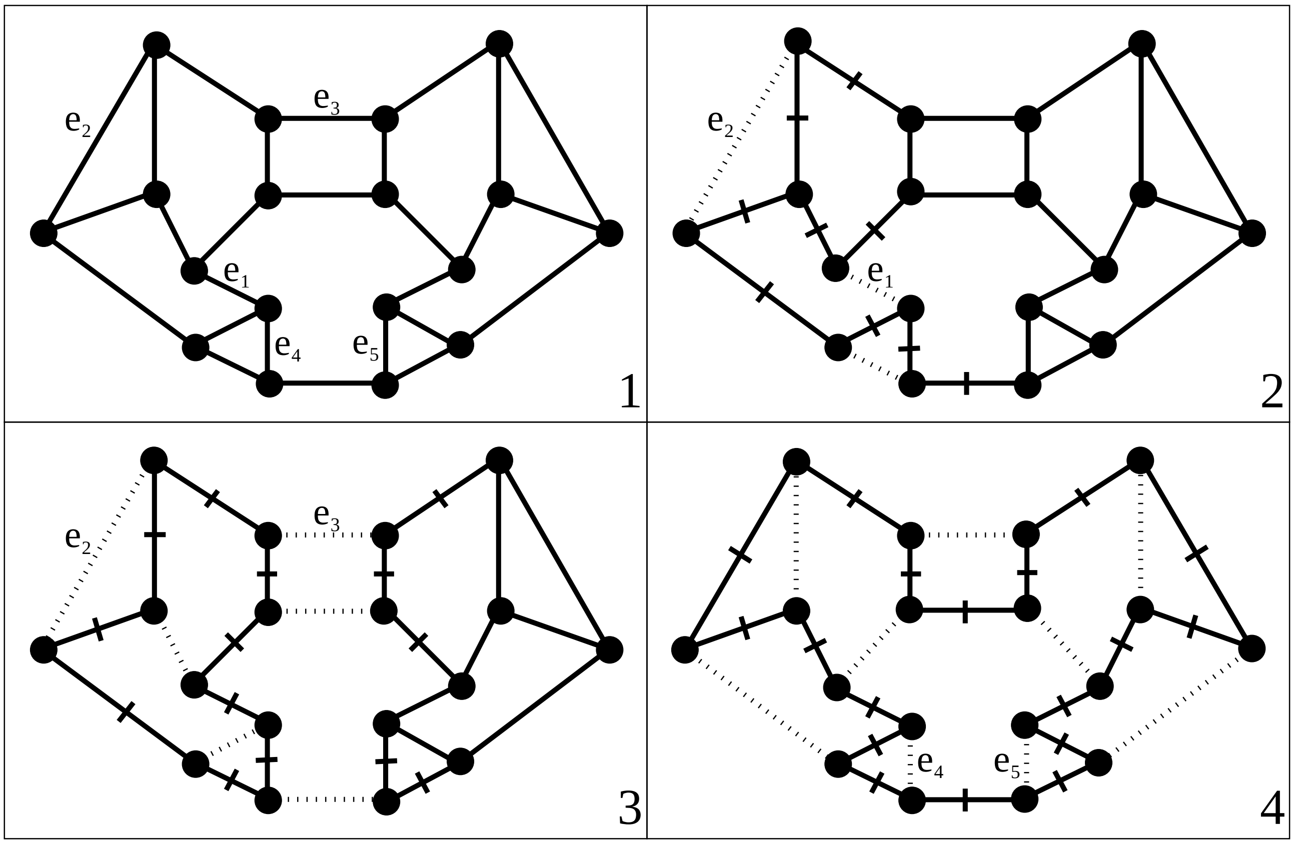

Figure 3.12: The Three Possible Ways For Probability Mill To Terminate

Within the framework of the Probability Mill, every deletion sequence will result in a Hamiltonian Cycle, a hub, or a disjoint cycle. Clearly, it is impossible for the deletion of an edge to cause a vertex to have a degree less than 2, since all degree-2 vertices immediately have their edges forced, and are therefore immune to deletion. Probability Mill is sound in that it cannot find a 'false Hamiltonian Cycle', since it does not add edges. It is also complete, because every Hamiltonian graph obviously has some deletion sequence that will liberate a Hamiltonian Cycle.

Most importantly, the chain manipulation, hubs, disjoint cycles, pruning, etc. can all be implemented for free along with the actual deletion sequence. Probability Mill therefore has an $O(m)$ upper bound on its running time. In fact, this is the best possible, because Probability Mill is clearly bounded from below by $\Omega(m)$, giving us a total running time of $\Theta(m)$. This fast running time is desirable because it allows us to run this subroutine as many times as possible.

3.3.3 SCHCA Pseudocode

The details of Probability Mill are the most complicated aspect of the algorithm. The remaining pseudocode for the SCHCA is quite simple and is shown below.

```
SCHCA(G) {
        Initialize Data Structures
        InitialPruning(G)
        If (solved(G)) then Output HC or return that no HC exists
        Do {
            Copy(G, G*)
            ProbabilityMill(G*)
        } While (not Hamiltonian(G*))
        Output HC or return that no HC exists
}
```

Initializing the data structures on line 2 simply consists of setting up the chains, initializing an empty pruning queue, etc. The InitialPruning subroutine on line 3 is the only chance of determining that the graph is not Hamiltonian; non-Hamiltonian graphs that are too complicated to be solved by InitialPruning will cause the while loop to run an infinite number of times. The InitialPruning subroutine is a one-time cost outside of Probability Mill, so all Hamiltonicity checks and pruning with reasonably fast running times should be included here. For example, at the very least, InitialPruning should build all possible chains, run Lightning and Head Hunter as many times as possible, and check to make sure that the graph contains no hubs, is biconnected, and not unevenly bipartite. Since Probability Mill has already been described, the remainder of the pseudocode is self-explanatory.

Of course, alternate implementations are possible. For example, instead of making copies of the graph, we could simply maintain an 'undo list' that reverses the affects of every failed Probability Mill iteration. In practice, however, this provides little benefit, since the deletion sequences executed by Probability Mill often have $O(m)$ length. In any event, the asymptotic running time of the algorithm would not be affected.

It would not be difficult to attach a backtrack tree to the SCHCA so that it becomes a true algorithm rather than a heuristic by gaining the additional capacity to identify all non-Hamiltonian graphs. There seems little point in doing this, however, since non-Hamiltonicity would correspond to a backtrack tree that has been exhaustively filled out. This would take an exponential amount of time, and would therefore be useful only with very small graphs.

3.3.4 Empirical Running Times

High-Level Empirical Analysis

The running time of the SCHCA depends entirely upon the graph class on which it is executed. For that reason, it does not make sense to ask the question, "What is the running time of the Stone Carver Algorithm?" Instead, one must ask, "What is the running time of the Stone Carver Algorithm on the following class of graphs...?" In general, the expected running time for the SCHCA on a graph class X is $O(IP + PM * EI(X))$ where IP is the running time for Initial Pruning, PM is the running time for Probability Mill, and $EI(X)$ is the expected number of Probability Mill iterations needed to solve a member of X. This running time is contingent on X being Hamiltonian; otherwise if Initial Pruning fails to show the graph to be non-Hamiltonian, then Probability Mill will run indefinitely. Designing Initial Pruning to be so complicated so that it dominates the running time would defeat its purpose, so we may ignore its impact, leaving us with a running time of $O(m * EI(X))$, since we know that the running time of Probability Mill is $O(m)$.

Calculating $EI(X)$, however, is very difficult for most graph classes. Unfortunately, here we have been unsuccessful in rigorously proving any results. Nonetheless, the experimental evidence supporting the SCHCA's strength is quite compelling. In particular, the SCHCA appears to perform very well on random r-regular graphs for any $r \geq 3$. As already stated in Section 2.2, any graph chosen from this class is almost surely Hamiltonian, so we need not worry about infinite running times.

Our empirical evidence illustrating the running time of the SCHCA on random regular graphs was obtained by repeatedly running the algorithm on many different random 3-regular graphs of varying sizes, and analyzing the average number of Probability Mill iterations. Random 3-regular graphs are of particular interest because they are the sparsest non-trivial random regular graphs (random 2-regular graphs are just random

2-factors). All of our 3-regular random graphs were generated according to the standard 'Configuration Model' [33].

Three Versions Of The SCHCA

We tested three progressively stronger versions of the algorithm. The only difference between them is found in the Probability Mill subroutine. We refer to these different implementations as 'Version 1', 'Version 2', and 'Version 3' of the SCHCA, where Version 1 is the most simple, and Version 3 is the most complicated.

Version 1 of the SCHCA is as described above. The Probability Mill subroutine runs in $O(m)$ time, and incorporates the Lightning and Head Hunter pruning rules.

Version 2 of the SCHCA is slightly more complicated. The asymptotic running time of Probability Mill cannot be improved, but it turns out that the number of probability Mill iterations can be reduced significantly while increasing its running time only modestly. By incorporating the Look-Ahead Theorem into Probability Mill, the algorithm gains the ability to sidestep potential mistakes, which significantly cuts down on the number of Probability Mill iterations that terminate in hubs and disjoint cycles. This step is fairly straightforward to implement; every time we are about to perform a Probability Mill deletion, we first run a simulation that deletes the edge in question. If that deletion causes the graph to become non-Hamiltonian, then we force the edge. Otherwise, we delete it. A simulated deletion with pruning could cascade and delete $O(m)$ edges. However, in practice this does not seem to occur with random 3-regular graphs, and pruning cascades tend to have constant length. Therefore, instead of making a copy of the graph, we implement the Look-Ahead simulation using an undo list.

Version 3 also uses the Look-Ahead Theorem, but this time uses it more extensively. It still simulates every deletion, but we call this version 'Variable Look-Ahead' because every time a simulated deletion saves a mistake from occurring, we run a 'Full Look-Ahead' on every unforced edge in the graph. That is, we sequentially simulate the deletion of every unforced edge, and accordingly force those that cause the graph to become non-Hamiltonian. Naturally, all subsequent Lightning and Head Hunter pruning is also performed.

Experimental Evidence & Analysis

Before coming to the charts which compare these three versions and illustrate their suspected running times, let us describe the experiments. All tests were run on random 3-regular graphs. We selected several graph sizes; our first experiments looked at graphs with 100 vertices up to 1,000 vertices separated by increments of 100 vertices. In each of these 10 groups, we generated 1,000 graphs, and solved them using the three versions of the SCHCA. The performance was better than expected, so we ran more tests on larger graphs. With Version 1, we solved graphs having 5,000 vertices up to 50,000 vertices, separated by increments of 5,000. With Versions 2 and 3, we continued up to 100,000 vertices. Again, each group contained 1,000 graphs. All experiments were run under Windows XP Professional on a normal desktop computer containing a single AMD Athlon XP 1800+ processor and 768 MB of RAM. Consistent with the result that any randomly-generated regular graph is Hamiltonian with high probability [42], no non-Hamiltonian graphs were seen.

The chart in Figure 3.13 below shows the average number of Probability Mill iterations that each of the three versions required for finding Hamiltonian Cycles in random 3-regular graphs.

This data strongly suggests that the relationship between graph size and the expected number of Probability Mill iterations is at worst linear for the SCHCA, regardless of which version is being run. In other words, for random 3-regular graphs, $EI(X)$ appears to be $O(n)$. Experiments on 4, 5, 6, and 7-regular graphs as well as complete graphs yielded similar linear relationships. If future research can prove this relationship to hold,

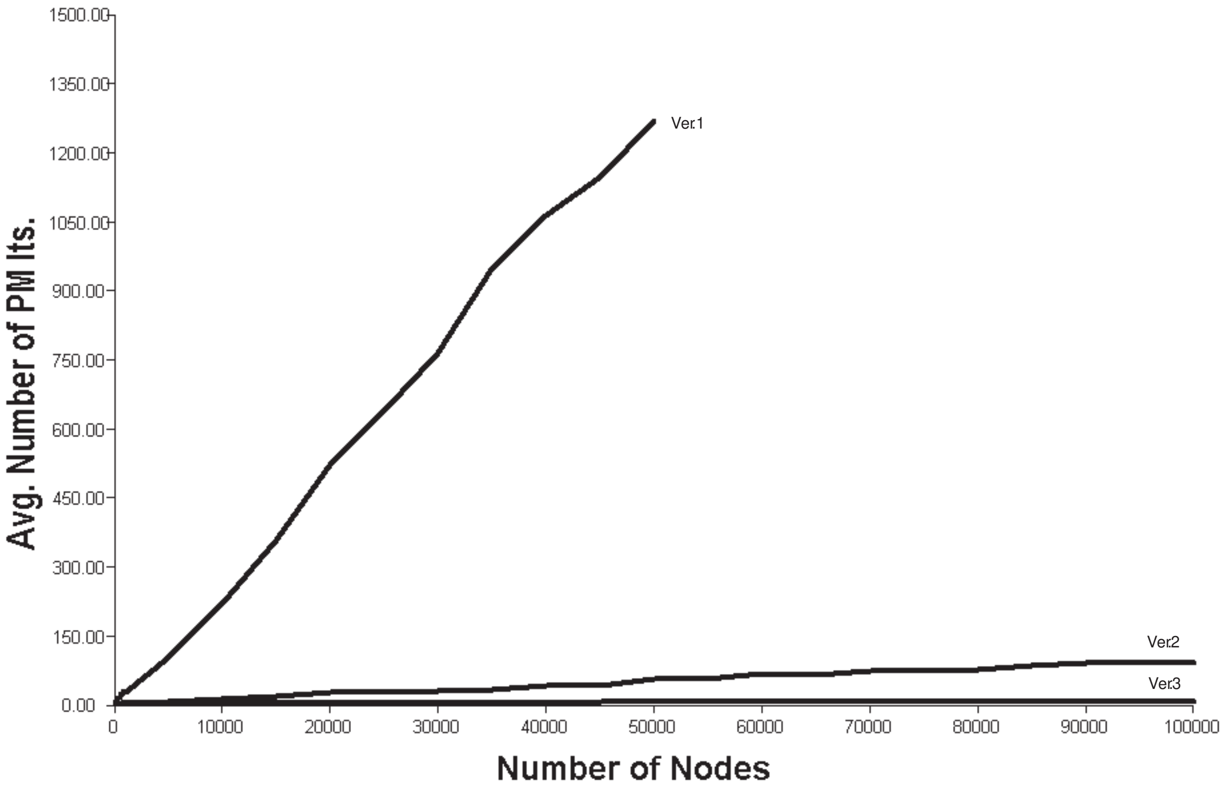

Figure 3.13: Chart Comparing The Average Number Of Probability Mill Iterations Between The Three Versions Of The SCHCA Running On Random 3-Regular Graphs

it would mean that the expected running time of the SCHCA on random regular graphs is $O(n) * O(m) = O(n \cdot m) = O(n^2)$ for sparse random regular graphs. This would be a considerable improvement over the current best, $O(n^6)$, published in [20].

In addition, this data shows that adding the Look-Ahead step into the Probability Mill subroutine of Versions 2 and 3 has a dramatic effect on reducing the number of Probability Mill iterations. Whether or not it is worthwhile to include this step, however, is not a foregone conclusion. There is a possibility that the amount of running time added in each Probability Mill iteration exceeds the time saved by reducing the number of iterations. It turns out that for random regular graphs, this is not a concern, as shown by the chart below in Figure 3.14.

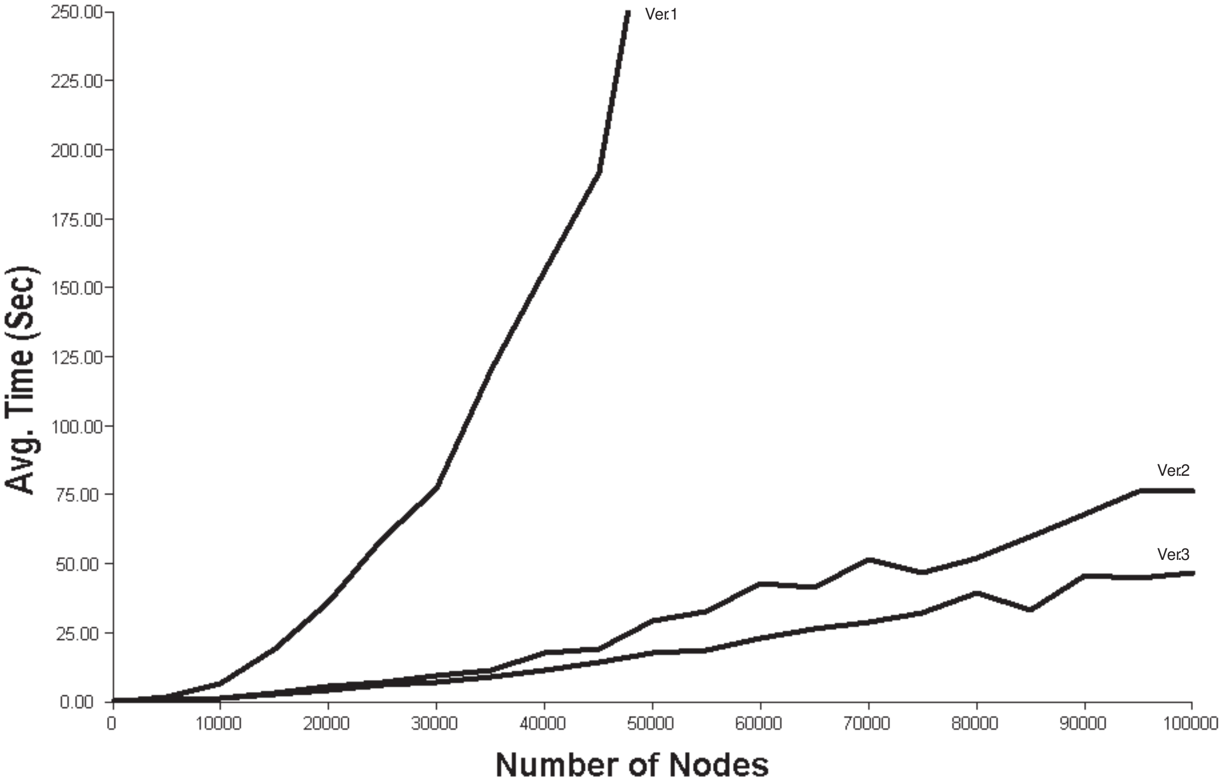

Figure 3.14: Chart Comparing The Running Time Of The Three Versions Of The SCHCA On Random 3-Regular Graphs

As was predicted, the running time of Version 1 forms an obvious parabola. The Look-Ahead steps included in Versions 2 and 3 clearly pay off in terms of running time, and their running times at worst seem to be quadratic, tempered by small constants.

In fact, a closer look at the experimental evidence raises the possibility that the Look-Ahead step may actually improve the asymptotic running time of the SCHCA. The chart in Figure 3.15 below shows the average time per Probability Mill iteration for the three version of the SCHCA.

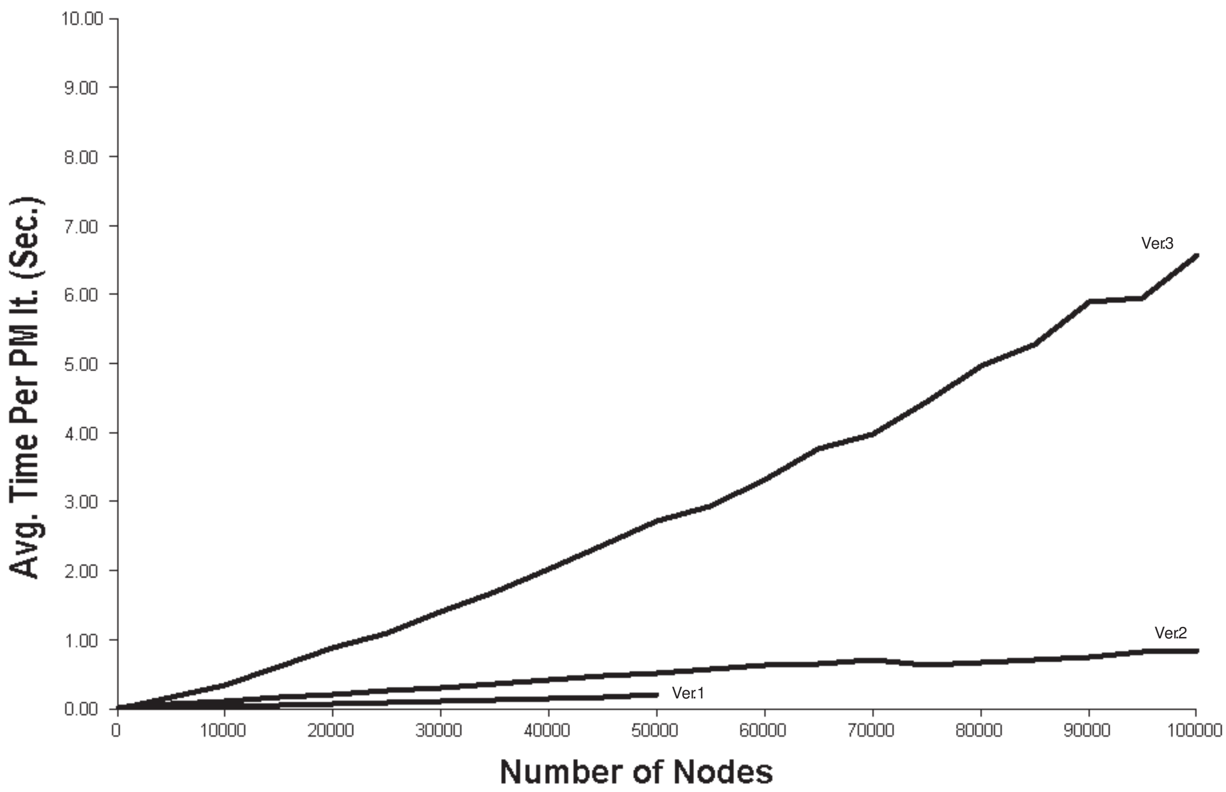

Figure 3.15: Chart Comparing Average Time Per Probability Mill Iteration For The Three Versions Of The SCHCA On Random 3-Regular Graphs

The fact that the average time per iteration for Version 1 is linear should be of no surprise. As already discussed, the fact that the data for Version 2 is linear suggests that the single Look-Ahead step is rarely associated with an $O(m)$ pruning cascade. However, the most interesting data comes from Version 3. Given that we are running the Look-Ahead step on every single edge in the graph, it is somewhat surprising that the average time per iteration does not suffer, and remains linear. This seems to indicate that adding $O(n)$ steps into the Probability Mill subroutine does not necessarily increase its running time.

The data in Chart 3.13 shows that the number of Probability Mill iterations for Versions 1 and 2 grows linearly with graph size. However, the data for Version 3 is so scaled that it is difficult to see. The chart below in Figure 3.16 gives a better view.

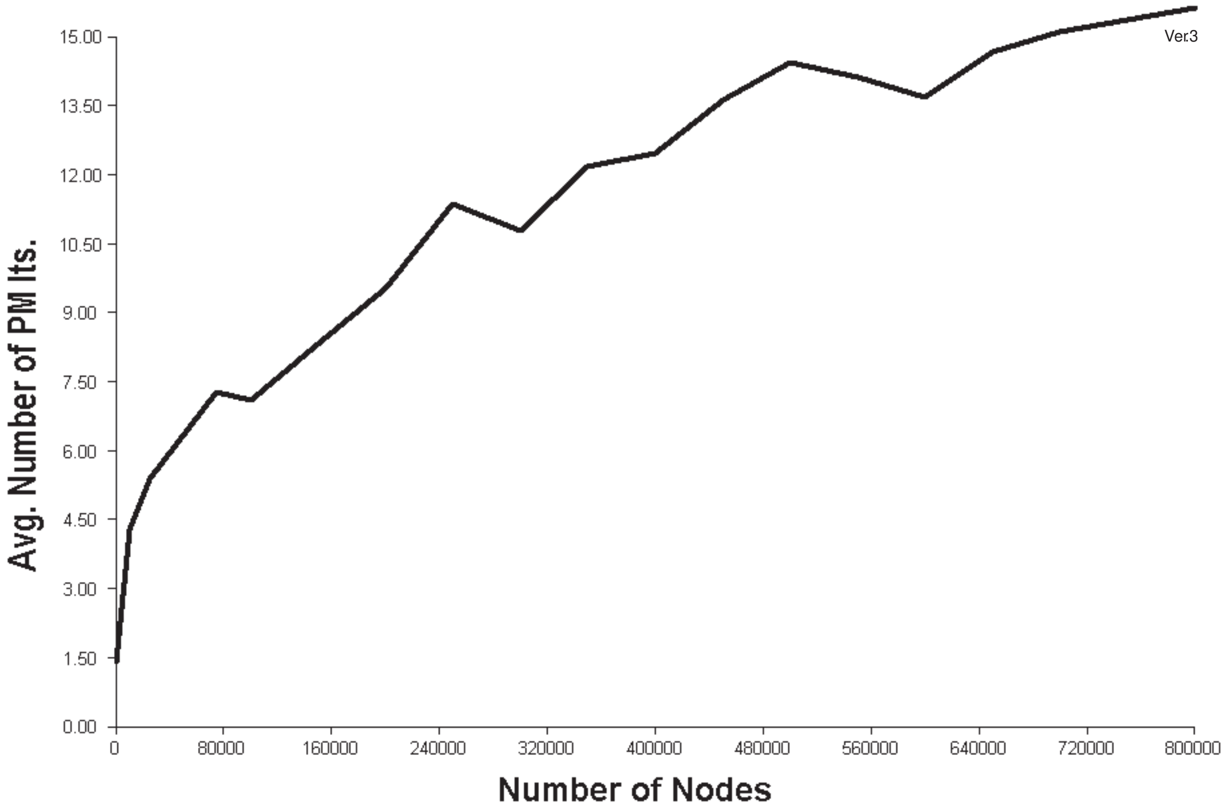

Figure 3.16: Chart Showing The Average Number Of Probability Mill Iterations For Version 3 Of The SCHCA Running On Random 3-Regular Graphs

This chart shows the performance of Version 3 in terms of Probability Mill iterations. Because the curve looked suspicious, we performed experiments including graphs with up to 800,000 vertices separated by increments of 50,000. Again, each data point contains the average number of Probability Mill iterations needed to solve 1000 random graphs.

Version 3 has clearly become very successful, and even large graphs require very few Probability Mill iterations. Although the increase in Probability Mill iterations for

Versions 1 and 2 is linear, $EI(X)$ for Version 3 appears to be sub-linear. This, together with the evidence from Chart 3.15 indicating that the average time per Probability Mill iteration increases linearly for Version 3, seems to suggest that the SCHCA's running time on random regular graphs may be even better than $O(n^2)$.

Interestingly, if we take the Probability Mill subroutine to the next level of complexity by running Full Look-Ahead on every edge after every Probability Mill deletion rather than waiting for a mistake to be saved, the running time on random regular graphs increases by a factor of at least $O(n)$, and the SCHCA loses its advantages.

Performance On Other Graph Classes

The SCHCA was successfully used by Dr. Frank Ruskey and his graduate student, Scott Effler at the University of Victoria to find Hamiltonian Cycles in Cayley graphs [15]. Their work was focused on trying to disprove the conjecture that all Cayley graphs are Hamiltonian by finding a counterexample. They have as of yet been unable to settle the conjecture, but empirically the SCHCA performed very well. It was unable to solve a few Hamiltonian instances, but at the very least one can claim that the SCHCA is a good filter for isolating potential candidates for non-Hamiltonicity among Cayley graphs.

The SCHCA was also tested on square Knight's Tour graphs, the famous class described in Section 2.1. By 'square', we mean that the chessboards corresponding to the graphs have an equal number of rows and columns. As square Knight's Tour graphs with odd-by-odd dimensions are not Hamiltonian [50], we focus on those corresponding to chessboards with an even number of rows and columns. Experimental evidence showing the performance of the SCHCA on square Knight's Tour graphs is shown below in Figure 3.17.

The testing conditions and number of samples taken were identical to those used for random regular graphs. Although the SCHCA did not perform as well on Knight's Tours

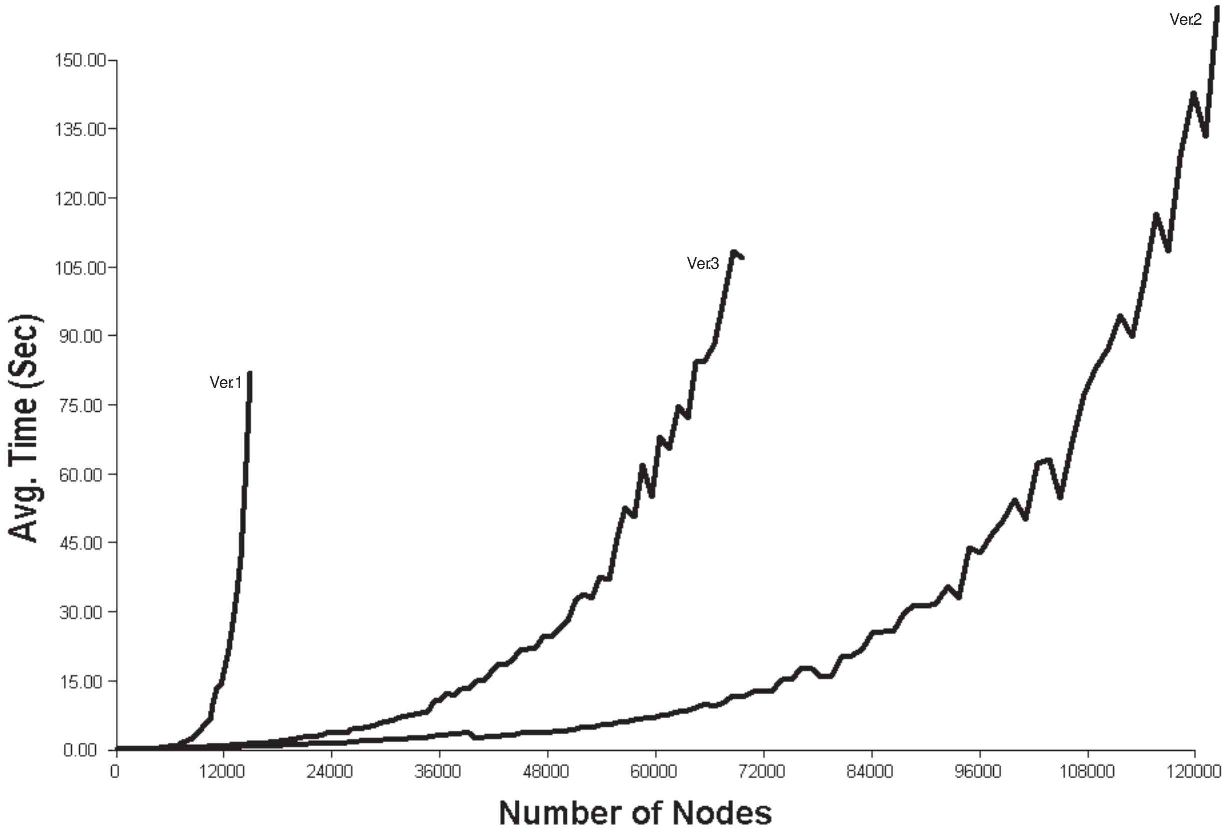

Figure 3.17: Chart Showing The Average Time Required For The SCHCA Running On Square Knight's Tour Graphs

as it did on random regular graphs, and it is difficult to make any claims about asymptotic running times based on this evidence, the results are interesting because Version 2 of the algorithm outperformed Version 3. In other words, the added Look-Ahead steps included in Version 3 seem to be a hindrance rather than an asset for this graph class.

In addition, we tested the SCHCA on square Toroidal Grids, graphs resembling Knight's Tour graphs except that the vertices corresponding to adjacent board squares are themselves adjacent so that each vertex has degree 4. The term 'Toroidal' is used because vertices corresponding to squares on the borders are adjacent to those corresponding to squares on the opposite borders. All toroidal grids are Hamiltonian. The proof of this statement proceeds by induction on the number of rows and columns in the grid, and is not difficult. Experimental evidence showing the performance of the SCHCA on square Toroidal grids is shown below in Figure 3.18.

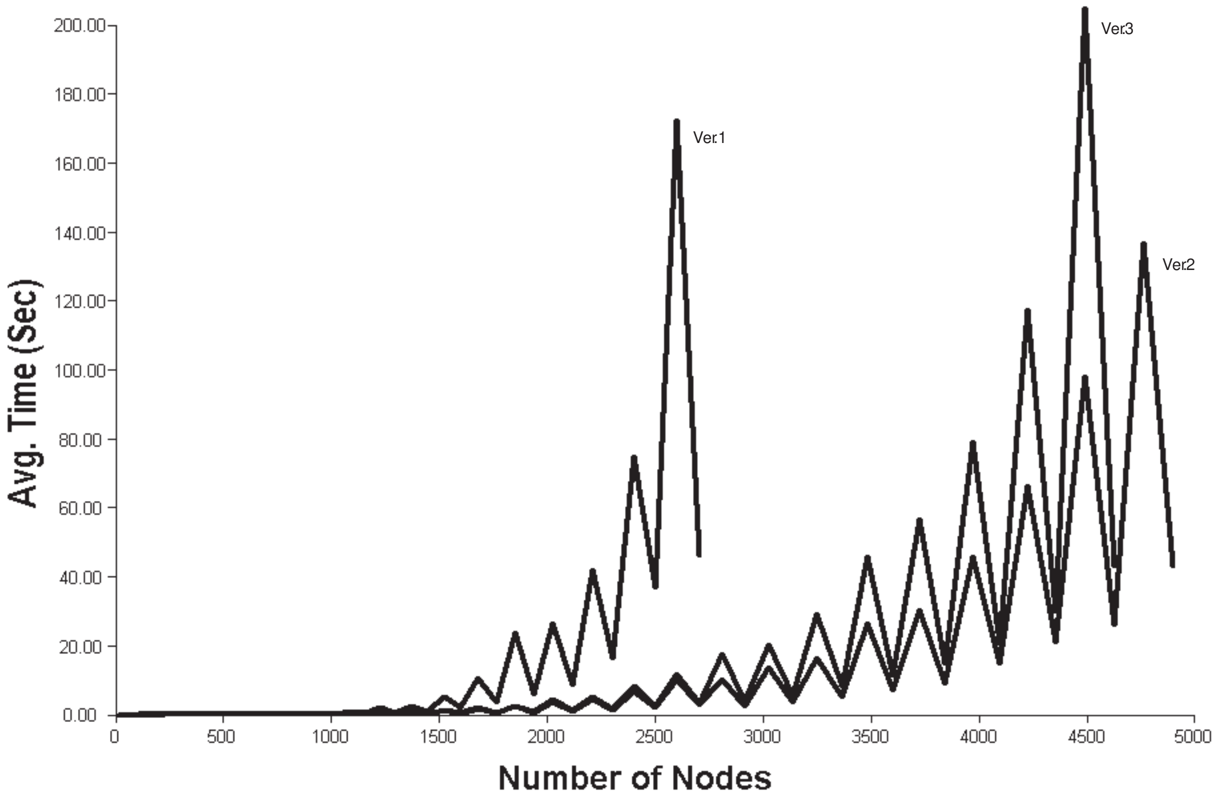

Figure 3.18: Chart Showing The Average Time Required For The SCHCA Running On Square Toroidal Grids

Once again, we used our 'standard' test conditions from before, and once again the SCHCA did not perform as well as it did on random regular graphs. However, this graph class is interesting for two reasons. Firstly, much like with Knight's Tours, Version 2 of the SCHCA outperformed Version 3. Secondly, note the zig-zag pattern of the results. All versions of the SCHCA had more trouble on square Toroidal Grids with odd-by-odd dimensions than they did on those with even dimensions. This may be an indication that square Toroidal Grids with odd dimensions contain fewer Hamiltonian Cycles.

We have seen that the SCHCA fares poorly on non-Hamiltonian graphs, but as implied by the conjecture that $\mathcal{P} \neq \mathcal{NP}$, there are also some Hamiltonian graphs that it cannot handle. For example, the SCHCA takes an exponential amount of time to solve Interconnected Cutset or ICCS graphs. ICCS graphs, described in [50] are graphs built using ICCS components. Such a component is shown in Figure 3.19, below.

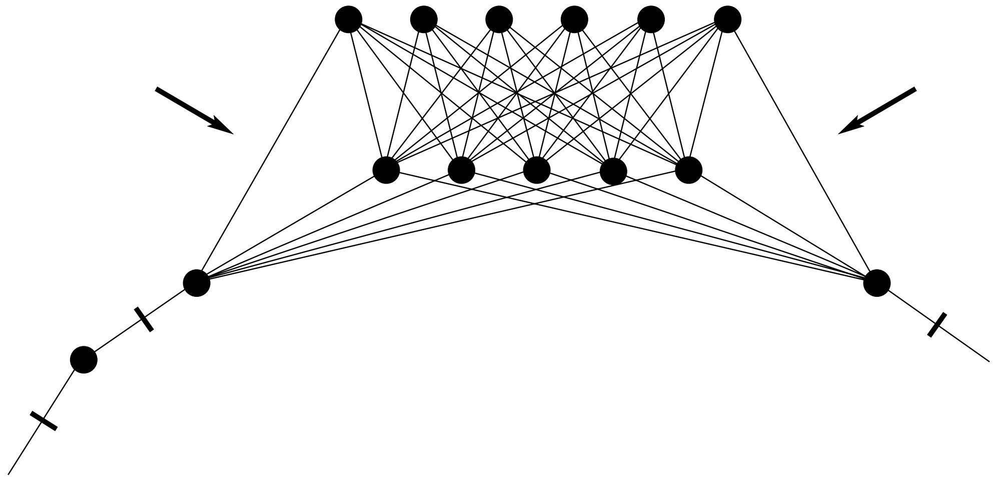

Figure 3.19: An ICCS Component

When these components are joined end to end in order to form a closed ring, the resulting ICCS graphs are all Hamiltonian, but nevertheless pose a great challenge for many Hamiltonian Cycle algorithms. Indeed, joining only 4 or 5 components together will create a graph that causes the SCHCA to slave for days. The reason is that the two indicated edges are in every possible Hamiltonian Cycle. This is not immediately obvious, and no mainstream Hamiltonian Cycle algorithms contain forcing rules that can recognize these edges. In effect, even with the Look-Ahead step, the SCHCA is 'blind' to this pitfall, and with enough ICCS components, the probability of at least one such edge being deleted becomes very high. After that, there is no hope of finding a Hamiltonian Cycle. Similarly, as reported in [50], algorithms that are based on the principle of choosing edges rather than deleting them often fail to choose these edges, and consequently fail to find a Hamiltonian Cycle. ICCS graphs are not special; any graph that contains a similar pitfall that the SCHCA cannot detect would cause a poor running time. For example, since the SCHCA does not check for biconnectivity during each Probability Mill iteration, it stands to reason that a graph which contains many edge-cuts which contain very few edges will lose its biconnectivity at some point during most Probability Mill iterations.

3.3.5 Comparison With Established Heuristic

With algorithms whose running times have not been formally established, it is often useful to compare them with other well-known algorithms in order to establish how fast they really are. It is obviously desirable to compare algorithms that were designed for similar inputs. The difficulty lies in finding an appropriate 'competitor'.

The DIMACS (DIscrete MAthematics & theoretical Computer Science) Center's 'Implementation Challenge' (http://dimacs.rutgers.edu/Challenges/index.html) holds an 'Implementation Challenge' attacking a different problem approximately every year. It is open to anybody who wishes to submit an algorithm, which is then evaluated on different inputs. Because of its competitive nature, this is a wonderful source of fast and practical algorithms. Although they have yet to have a formal Hamiltonian Cycle Implementation Challenge, the eighth DIMACS Challenge, held in 2001, focussed on the Traveling Salesman Problem (TSP). This problem, described in Section 1.3, is strongly related to the Hamiltonian Cycle problem in that any TSP algorithm can be easily adapted to find Hamiltonian Cycles. Recall that the TSP takes a complete weighted graph G as input along with a constant k, and asks whether G contains a Hamiltonian Cycle whose edges sum to at most k. In order to use a TSP algorithm to find Hamiltonian Cycles, simply take the undirected input for the Hamiltonian Cycle problem and turn it into a complete graph by adding all missing edges while setting the weights of all 'real' edges to 0 and those of 'non-edges' to 1. Simply set k to be 0, and input it into any TSP algorithm. Any TSP tour will correspond to a Hamiltonian Cycle in the original graph. Many TSP implementations contain a feature for finding Hamiltonian Cycles and do this automatically so that complete inputs are not required.

The DIMACS TSP Challenge has been divided into nine categories containing dozens of algorithms. Dr. David Johnson, one of the organizers of the Challenge, suggested that Dr. Keld Helsgaun's Lin-Kernighan Heuristic (LKH) would work very well on sparse random graphs. Dr. Helsgaun was kind enough to supply an implementation.

The LKH, published in [35], is one of the most celebrated heuristics for finding TSP tours. It has a reputation for being very fast but nevertheless often finds optimal or near-optimal tours. It is a well-studied 'local search' heuristic with many variants and implementations. Dr. Helsgaun's LKH variant (H-LKH), published in [30], differs somewhat from the original, but in any event the details of the LK Heuristic are beyond the scope of this thesis. We refer an interested reader to [30] and [35].

The Helsgaun variant is very effective. For example, at the time of its publication, it was able to quickly find optimal tours in many non-trivial graphs, including an instance containing 7397 cities, the largest that had been optimally solved to that date. This instance took 1065.6 seconds to solve. In contrast, respected exact algorithms working on the same instance required roughly 3-4 years of CPU time to solve.

Empirical evidence shows that it is also very effective when given random 3-regular graphs as input and asked to find a Hamiltonian Cycle. The chart shown below in Figure 3.20 illustrates the running time of the Helsgaun LKH variant compared with that of the SCHCA, Version 3. The running time data for the SCHCA comes from the same experiments that were described in the previous subsection. The running time data for the LKH comes from experiments conducted on the same computer under the same conditions as before, save that each data point contains the average running time needed to solve 100 graphs instead of 1000. The default parameters were used.

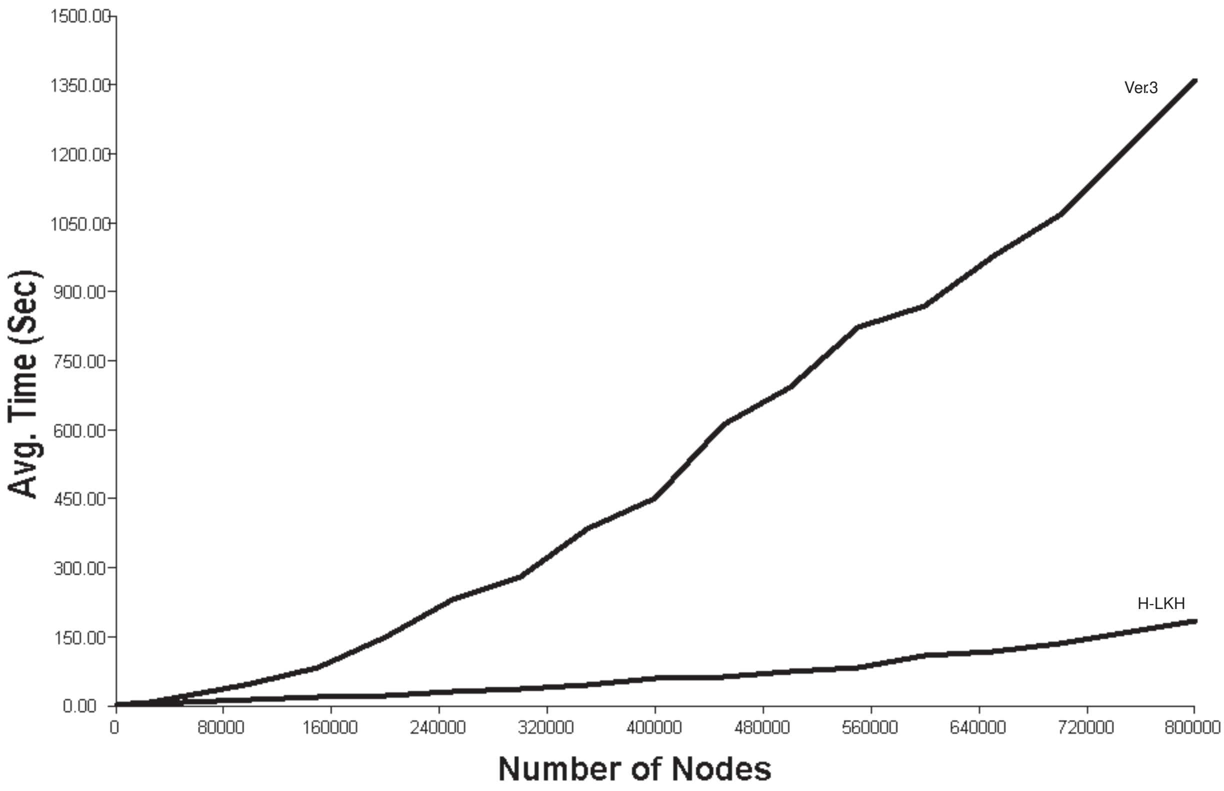

Figure 3.20: Chart Showing Running Time Comparison Between The SCHCA & Keld Helsgaun's Lin Kernighan Variant

The H-LKH variant clearly outperforms the SCHCA on the given inputs. The SCHCA takes a constant factor of approximately 7 times as long to find Hamiltonian Cycles in random regular graphs. Both running times increase at what appear to be slightly super-linear rates. This is consistent with the expected number of Probability Mill iterations and expected time per iteration as discussed in the previous subsection, but is slightly at odds with the reported running time of approximately $O(n^{2.2})$ for H-LKH. It may be the case that the actual running time for H-LKH is married to a very small constant factor. Another explanation is that the reported running time pertains only to TSP, and is not tight for the Hamiltonian Cycle problem on random 3-regular graphs.

It should be noted that the SCHCA was implemented in unoptimized C++, whereas H-LKH was implemented in optimized C for the DIMACS Challenge, suggesting that the implementation is likely as efficient as the algorithm would allow. However, the discrepancy between the running times cannot be attributed to the choice of language; if the SCHCA were reprogrammed in optimized C, the running time probably would not decrease greatly.

It is of course possible that if, as suggested by the results in the previous section, the running time of Version 3 of the SCHCA is faster than $O(n^2)$, and that of H-LKH is approximately $O(n^{2.2})$ as reported, then for progressively larger graphs the gap between the running times should shrink. This, however, is speculation and cannot be verified given the current computer resources of this project.

For 'real world' applications, H-LKH is clearly the choice for finding Hamiltonian Cycles in random regular graphs. In any event, the purpose of this research is not to find the world's fastest practical algorithm, but rather to develop an algorithm whose running time can be proven to be faster than $O(n^6)$, the current fastest. With respect to this goal, the SCHCA is the algorithm of choice because it is simple whereas H-LKH is much more complicated. Since proving even simple things about random graphs tends to be difficult, it seems unlikely that the running time of H-LKH on this graph class will ever be proven. In contrast, there is hope for proving the running time of the SCHCA.

3.3.6 Future Research

The obvious next step for this research to take is to prove the running time of Version 1 of the SCHCA on random regular graphs to be as close to $O(n^2)$ as possible. With a little effort, it should be possible to obtain a tight bound. In addition, proving that the single Look-Ahead step of Version 2 does not affect the expected running time of Probability Mill also seems within the realm of possibility. In contrast, proving that the variable Look-Ahead steps of Version 3 do not affect the running time would most likely be quite difficult due to its complexity.

With greater computer resources, it would be worthwhile to continue to chart the running times of the SCHCA and H-LKH to see how they perform on very large graphs.

Establishing solid running times on other graph classes would also be a worthwhile result. This may involve including more pruning and forcing rules into the framework of Probability Mill. In any case, a formal relationship between the tradeoff of including expensive pruning rules and their effectiveness would be an interesting result.

An alternate course of research would be devise and study more powerful versions of Probability Mill by including additional pruning rules. This would most likely mean sacrificing its fast $O(m)$ running time. Additional pruning might not be worthwhile on random regular graphs, but perhaps other graph classes would benefit greatly. For example, including a check for biconnectivity would at worst increase the running time of Probability Mill by a factor of $O(m)$. As we have seen, $O(m)$ steps do not necessarily increase the running time of the overall algorithm; if biconnectivity were checked every time a mistake were saved by the Look-Ahead step, it may not increase the running time at all. Such things seem very difficult to prove, however.

3.4 More On Deletion Sequences

3.4.1 Introduction

In this section we explore deletion sequences that result in the creation of hubs, and describe two results. With our first result, we show that given certain types of deletion sequences that result in hubs in cubic graphs, we can find shorter deletion sequences that create the same hubs regardless of how long the original sequence was, and prove that the maximum necessary length of this type of deletion sequence is 3. Given this result, it is tempting to try to prove that any hub in a cubic graph can be created by a deletion sequence of length at most 3. However, it turns out that this is not true, and our second result shows that the necessary length of deletion sequences, even in cubic graphs, is unbounded. That is, it is possible to create graphs in which some vertex will only become a hub if subjected to deletion sequences which are necessarily very long.

The framework set up by the SCHC algorithm works to avoid hub creation. During each Probability Mill iteration, the Lightning pruning rule tries to delete all unforced edges incident on a vertex which is already incident on two forced edges. Nonetheless, hub creation is still possible through less direct means. For example, suppose a degree-3 vertex, call it x, has a forced edge incident on it, and the deletion of an edge causes another of its edges to become forced. At this point, the final unforced edge of x is placed at the end of the pruning queue. However, as we execute the pruning queue, it is possible that we are never able to delete this edge, because some other pruning deletion first causes it to be forced, thereby creating a hub and terminating the Probability Mill iteration.

Before continuing, and for the sake of clarity, it would be prudent to emphasize certain attributes of all deletion sequences in this section:

- In order to keep from being bogged down in the complexities of working with too many pruning rules, the only one being applied is the Lightning rule.

- Pruning proceeds in a breadth-first rather than a depth-first manner. That is, when an edge is deleted for whatever reason, all edges incident on degree-2 vertices are immediately forced, and then all edges slated for pruning are placed on the pruning queue before the next edge is removed and pruned. This guarantees that pruning spreads concentrically from the deleted edge that triggered it.

- Because graphs with vertices that have a high degree do not lend themselves well to pruning, we will only consider cubic graphs.

3.4.2 Finding Shorter Deletion Sequences

Given a deletion sequence resulting in a hub, it is often possible to find a shorter deletion sequence resulting in the same hub. The reason for this is that some of the deletions may not have directly contributed to creating the hub, and are therefore extraneous. If these extraneous edges are removed from the deletion sequence, we are left with a deletion sequence which still creates the same hub. For example, consider the graph shown in Figure 3.21.1, below.

In Figure 3.21.2, we execute the deletion sequence $(e_1, e_2, e_3, e_4, e_5, e_6, e_7)$ to create a hub. Clearly, some of these deletions are extraneous, because in Figure 3.21.3 we execute a proper deletion subsequence, namely (e_3, e_7), to create the same hub.

When finding a more efficient deletion sequence, we are not necessarily restricted to proper deletion subsequences. We are often able to include edges that were otherwise pruned as members of a new, shorter deletion sequence. Likewise, we are sometimes able to include edges that were not affected at all by the longer sequence.

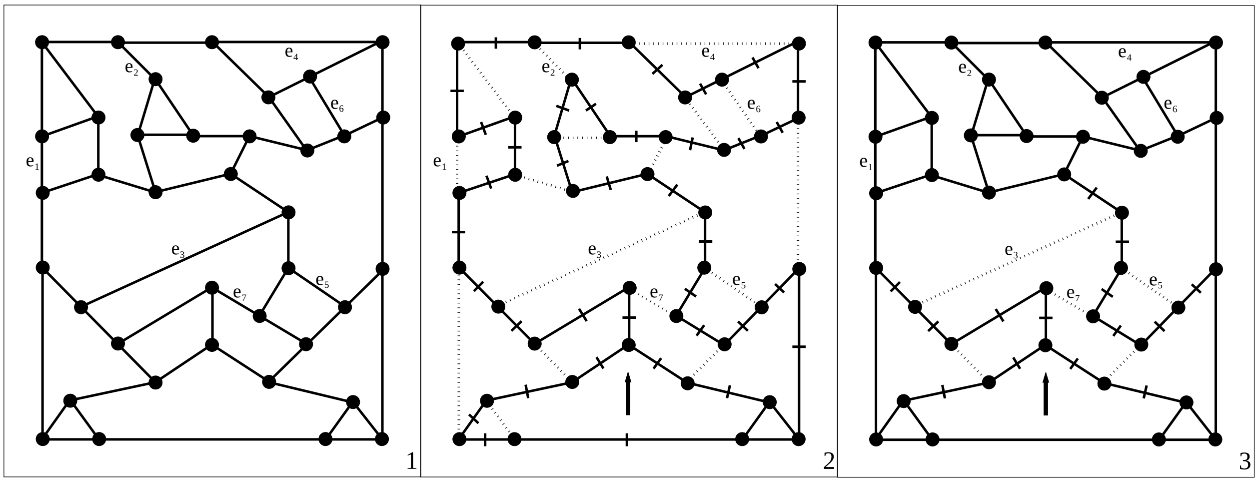

Figure 3.21: Two Different Deletion Sequences Resulting In The Same Hub

The Hub / Deletion Sequence Lemmas

Let $D = (e_1,\ e_2,\ e_3,\ \ldots\ ,\ e_\alpha)$ be a deletion sequence executed on a cubic graph that causes some vertex, call it x, to become a hub. Let us consider the graph containing x before D is executed. Since we know that x will become a hub, the three edges incident on x will be forced in a certain order. Let us label the other endpoint of the first edge to be forced as a, the other endpoint of the second edge to be forced as b, and the other endpoint of the third edge to be forced as c. Furthermore, the only way for an edge to be forced is if one of its neighboring edges is deleted. Therefore let us label the soon-to-be-ex-neighbor of a as $a^{'}$ and its other neighbor as a^*, and similarly for the other neighbors of b and c, as shown in Figure 3.22, below.

If D contains exactly two edges that are incident on the neighbors of x, but not on x, then the following three lemmas hold:

Lemma 3.4.1 *Edge $(b,\ b^{'})$ is the last edge in D.*

Proof: We know that executing D will delete $(a,\ a^{'})$ before it deletes $(b,\ b^{'})$. However, deleting $(b,\ b^{'})$ will cause x to become a vertex with exactly 2 forced edges, and therefore also place $(x,\ c)$ on the pruning queue to be deleted using the Lightning pruning rule.

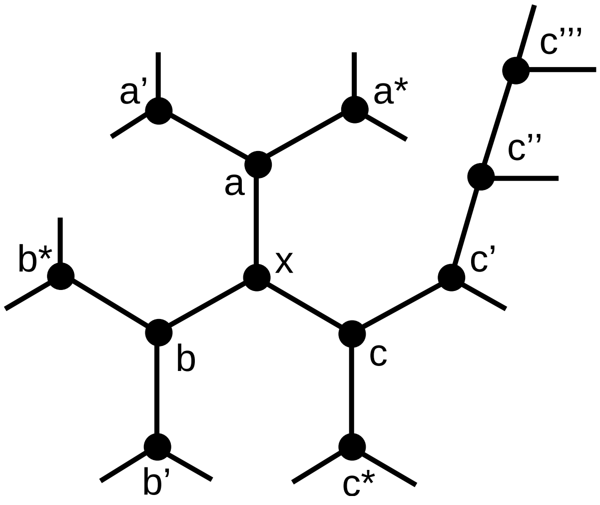

Figure 3.22: A Graph Accompanying The Hub / Deletion Sequence Lemmas.

In effect, the execution of the pruning queue will never reach edge (x, c), since we know that it will first become forced to create the hub at x. Therefore, $(b, b^{'})$ must be the very last edge in D, as required. $\square$

Lemma 3.4.2 *$b^{'}$ and $c^{'}$ must be adjacent.*

Proof: We know that deleting $(b, b^{'})$ sets off the events which force (x, c). Deleting $(b, b^{'})$ places (x, c) on the pruning queue, which becomes forced before its position on the pruning queue is reached. Therefore, deleting $(b, b^{'})$ must place some other edge onto the pruning queue before (x, c). That edge must be $(c, c^{'})$, because if it were any further away from x, the breadth-first nature of the pruning queue would first delete (x, c). The only way in which deleting $(b, b^{'})$ could place $(c, c^{'})$ onto the pruning queue is if $b^{'}$ and $c^{'}$ are adjacent, as required. $\square$

Lemma 3.4.3 *There exists a deletion sequence D^{*} containing at most 3 edges that also causes x to become a hub.*

Proof: Since deleting $(b, b^{'})$ places $(c, c^{'})$ on the pruning queue, we know that $c^{'}$ must already have had one of its edges, call it $(c^{'}, c^{''})$ forced. This edge may have been forced by a pruning deletion, but in the worst case, we assume that it was forced by removing one of the edges in D. Let us call this edge $(c^{''}, c^{'''})$. Therefore, we can construct a new deletion sequence $D^* = (\ (c^{''}, c^{'''}), (a, a^{'}), (b, b^{'})\ )$ which will cause x to become a hub, and D^* has a length of at most 3, as required. $\square$

Some deletion sequences can be even shorter. For example, it is possible for hubs to be created by the removal of only two edges. However, it is impossible for the removal of a single edge to create a hub in a cubic graph, since this could only force a maximum 4 edges, of which only 2 (rather than the required minimum of 3) could be incident on any vertex.

3.4.3 Deletion Sequences Of Unbounded Necessary Length

Given the above result, it is tempting to try to strengthen it by showing that any hub in a cubic graph can be created by a deletion sequence of length at most 3. It turns out that this is not possible because the statement simply is not true. In fact, no constant will do, and the necessary length of deletion sequences in cubic graphs is unbounded.

To illustrate this, we will describe how to construct graphs that require long deletion sequences in order to cause a certain vertex to become a hub. This family of graphs is based loosely on the structure of Tutte's Graph, shown in Figure 4.1. The idea is to create a graph with three components incident on a central vertex, as shown in Figure 3.23, below.

The components have a few important properties. Namely, they are not only isomorphic to one another, but each one is also bilaterally symmetric. By this, we mean that it is possible to divide each one using a line that passes through the vertex incident on x, which passes between the other two corner vertices such that the two halves are also isomorphic to one another. These properties ensure that the deletion sequence which we choose is not dependent on graph orientation.

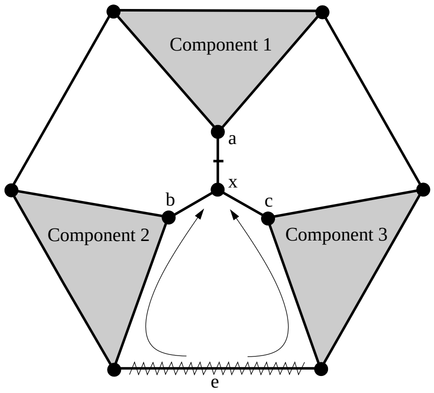

Figure 3.23: A Template For Building Graphs For Which Deletion Sequences Cannot Be Shortened

The idea behind this result is very simple. We will set up the graph so that the following deletion sequence creates a hub at x: we first force edge $(x,\ a)$ by deleting some other edge incident on a. This will be the only edge in component 1 that we need to delete. We then delete a number of 'supporting edges' in components 2 and 3 so that deleting edge e will cause a cascade of Lightning pruning deletions through those components as indicated by the arrows to force the remaining 2 edges incident on x, thereby creating a hub.

The deletion sequence causing x to become a hub therefore is $D = (\ (x,\ a),\ s_1,\ s_2,\ldots\ s_p,\ e\ )$, where each s_i represents a 'supporting edge'. Given any integer k, we can ensure that D's necessary length is longer than k by arbitrarily increasing the size of each component, which in turn increases the number of supporting edges that will be needed. The fact that the components are separated by a single edge guarantees that the necessary number of supporting edges will increase with the size of the components.

Constructing arbitrarily large symmetrical, cubic components that allow for a cascade of Lightning pruning to create a hub at x is a simple task, as illustrated by Figure 3.24, below.

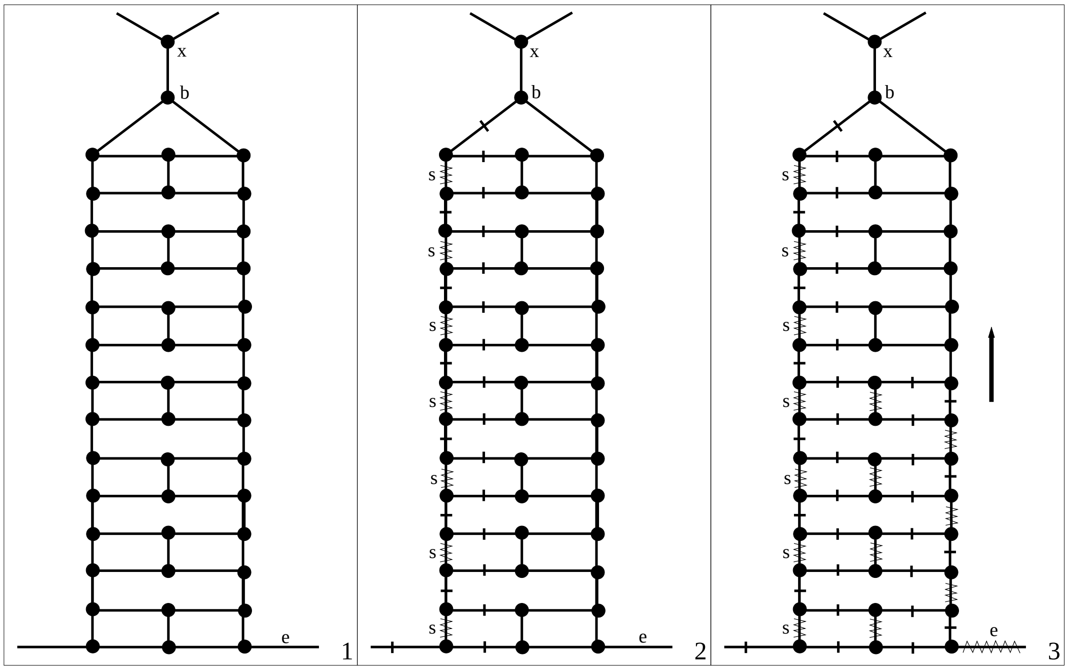

Figure 3.24: One Possible Way To Construct Components That Cause Long Deletion Sequences To Be Necessary

The component shown on the left is drawn to be substituted into Figure 3.23 as 'Component 2'. Vertex x, which is to become the hub, is shown at the top, and edge e, which will ultimately trigger the pruning cascade is shown at the bottom. Such a cascade requires a number of supporting deletions placed periodically along the length of the component. One possible set of supporting deletions is shown by the edges marked with an 's' in the center diagram. In the right-hand diagram, we delete edge e, which triggers the cascade of Lightning deletions. The progress of this cascade is indicated by the arrow, and will ultimately force edge (b, x). The mirror image of this situation unfolds in Component 3. It should be obvious by inspection that without supporting deletions at intervals along the entire length of the component, the pruning cascade will not reach x. Since we have the power to construct such components with arbitrary length, we can use them to build graphs in which vertex x can only be forced with arbitrarily long deletion sequences, as required.

Example 3.4.1

Putting the above method to use, we construct a graph that requires a deletion sequence with a length of at least 4 in order to create a hub at the central vertex. This graph as well as a deletion sequence of length 4 are shown in Figure 3.25, below.

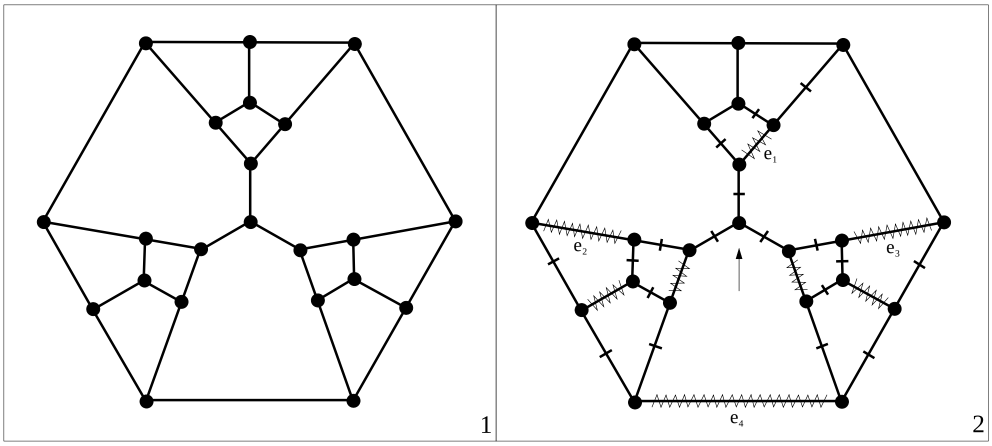

Figure 3.25: The Indicated Hub's Creation Requires The Deletion Of At Least 4 Edges

Ultimately, in order to create a hub, each of the 3 components requires at least one edge to be removed, followed by the final edge, e_4, which triggers a cascade of pruning, resulting in the indicated hub being created. No shorter deletion sequence is possible.

3.4.4 Future Research

It would be interesting to explore deletion sequences that use different pruning rules and find explicit bounds on their necessary length. In addition, it may be possible to further strengthen the first result to include a wider class of deletion sequences.

There are, however, even more fundamental open questions concerning deletion sequences: Does every (planar) cubic 3-connected graph of sufficient size contain at least one vertex that can be transformed into a hub by some deletion sequence (with length at most k for some constant k)? Are some vertices 'immune' to becoming hubs? Are vertices in non-Hamiltonian graphs more likely to become hubs than vertices in Hamiltonian graphs?

We conjecture that the answers to all of these open problems is 'yes'. For the first problem, this is because the alternative is for there to be some graph in which every possible deletion sequence converts the graph into a 2-factor, which intuitively seems unlikely. Furthermore, we have seen that it is possible to design graphs in which some vertex requires long deletion sequences, but to design a family of graphs in which every vertex requires arbitrarily long deletion sequences also seems unlikely. There most likely are vertices which are 'immune' to becoming hubs; the central vertex in the Tutte graph is a good candidate. Finally, since non-Hamiltonian graphs have no associated deletion sequences that are successful in finding Hamiltonian Cycles, a greater proportion of them will produce hubs, showing that hub creation in non-Hamiltonian graphs is more likely.

Chapter 4

Strengthening Barnette's Conjecture

4.1 Introduction

Barnette's Conjecture, published in [4], states that all planar, cubic, 3-connected, bipartite graphs are Hamiltonian. As already discussed in Subsection 2.1.4, this comes as part of a series of conjectures concerning Hamiltonicity, the first of which dates back more than 100 years. In [45] Tait conjectured that all planar, cubic, 3-connected graphs are Hamiltonian. Had this been true, it would have supported a short, elegant proof of the Four-Colour Theorem. Tait's Conjecture stood for more than 60 years, but was disproved by Tutte, who constructed a clever counterexample in [47]. This graph has come to be known as the 'Tutte Graph', and is shown below in Figure 4.1.

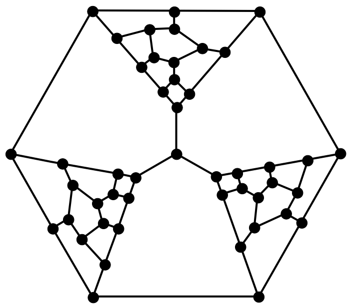

Figure 4.1: The 46-Vertex Tutte Graph

In [49] Tutte then conjectured that all cubic, 3-connected, bipartite graphs are Hamiltonian. Tutte's Conjecture fell due to a counterexample by Horton, published in [9]. The 'Horton Graph' is shown below in Figure 4.2.

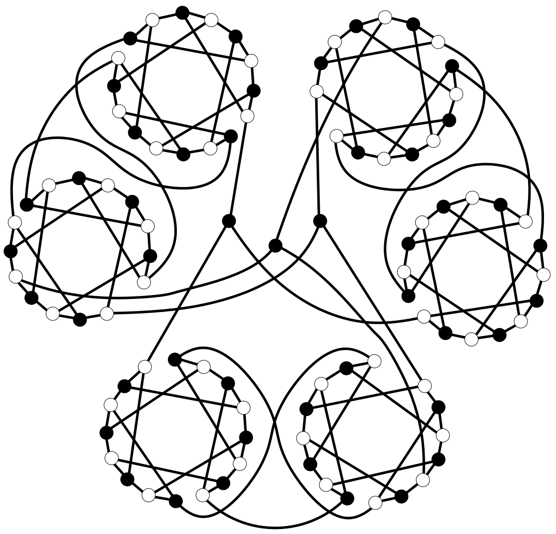

Figure 4.2: The 96-Vertex Horton Graph

Barnette's Conjecture subsumes the other two conjectures in that any counterexample would have to be a simultaneous counterexample to both the Tait and Tutte Conjectures.

Despite considerable effort to prove otherwise, Barnette's Conjecture has stood since its publication in 1969. Many partial and related results have been obtained and are described below, but it still remains unclear whether the conjecture is true or false. The purpose of this paper is to strengthen Barnette's Conjecture in the hope that a stronger conjecture may be easier to disprove. Our main result states that Barnette's Conjecture is true if and only if every Barnette graph is Hamiltonian-Edge-Connected. Hopefully this fact will aid future researchers in constructing a counterexample or a proof. The result is obtained by first describing a procedure that can be used for building successively larger Barnette graphs. This construction may in and of itself be of interest to the reader.

4.2 Partial & Related Results

Although Barnette's Conjecture remains an open problem, many partial results have been achieved. In [23], Goodey proves that any Barnette graph with faces having 6 or fewer sides is Hamiltonian. In [32], Holton et al. confirm through a combination of clever analysis and computer search that all Barnette graphs with up to and including 64 vertices are Hamiltonian. In [39] this result is extended by McKay et al. to 84 vertices, inclusive. In [18] Feder and Subi prove a complicated but interesting result, namely that for any Barnette graph, if the faces can be 3-coloured such that two of the three colour classes contain only squares and hexagons, and the third colour class contains faces surrounded by an even number of squares, then that graph is Hamiltonian. The literature also contains related complexity results; in [1] Akiyama et al. prove that the Hamiltonian Cycle problem remains $\mathcal{NP}$-Complete even when the input is restricted to planar, cubic, 2-connected, bipartite graphs. Nobody has yet been able to show that it remains $\mathcal{NP}$-Complete when restricted to Barnette graphs. Such a result would clearly invalidate the conjecture. It should be noted that instances of non-Hamiltonian, planar, almost cubic, 3-connected, bipartite graphs have been discovered. For example, Kirkman's graph shown in Figure 2.5 contains only three non-cubic vertices. In effect, relaxing any of the Barnette adjectives leaves us with a class that contains non-Hamiltonian graphs. On the other hand, adding further constraints such as Goodey has done leaves us with a class that is Hamiltonian. This seems to suggest that the Barnette graphs lie on the border of Hamiltonicity and non-Hamiltonicity.

Barnette's less-famous Hamiltonicity conjecture states that any planar, cubic, 3-connected (bipartite omitted) graph with faces having 6 or fewer sides is Hamiltonian. Partial results have also been achieved here. In [10] Brinkmann et al. confirm that this conjecture holds for up to 250 vertices. In [24] Goodey proves that a proper subset of these graphs, namely all planar, cubic, 3-connected graphs with faces having 3 or 6 sides, are Hamiltonian. The Fullerenes are another interesting subset [31]. They are planar, cubic, 3-connected graphs with exactly 12 faces of degree 5, and all other faces having degree 6. These graphs represent possible molecular structures of pure carbon.

4.3 Main Result

We show that Barnette's Conjecture holds if and only if every Barnette graph is Hamiltonian-Edge-Connected. While proving the 'reverse direction' is trivial, the 'forward direction' uses a standard proof by contradiction. Under the assumption that Barnette's conjecture is true, the further assumption that some Barnette graph contains an edge which is in no Hamiltonian Cycle allows us to construct a non-Hamiltonian Barnette graph. This construction is given in detail below. The main theorem requires the following lemmas:

Lemma 4.3.1 *A plane graph is bipartite if and only if all of its faces have even degree.*

Proof: Almost every modern graph theory textbook contains a proof of the statement that a graph is bipartite if and only if it contains no odd cycles. All that remains to be shown is that a plane graph contains no odd cycles if and only if all of its faces have even degree.

$\Rightarrow$ Trivial

$\Leftarrow$ Let G be any arbitrary plane graph whose faces all have even degree. Suppose that G contains some odd cycle, call it C. This cycle cannot be a face, so it must enclose two or more faces. However, it is impossible to add even-sided faces together to create an odd-sided perimeter. To see this, suppose one were to attempt to construct C and all of its internal edges by laying even-sided faces together onto a plane in a way similar to how someone might tile a floor. Every new face that was added would subtract some edges from the overall perimeter by turning them into internal edges, but would also add some edges to the perimeter. Since only even-sided faces are added, any even number of edges subtracted from the perimeter will always be balanced by adding an even number; likewise, any odd number of edges subtracted from the perimeter will be balanced by adding an odd number. This guarantees that the perimeter will always maintain an even number of edges around it, showing that it is impossible to construct C using only even-sided faces, in turn showing that C must contain at least one odd-sided face, which is a contradiction. Therefore G contains no odd cycles, as required. □

4.3.1 The Mirroring Procedure

We now describe the steps of an informal procedure that takes as input a cubic planar graph G, and outputs a different cubic planar graph G'.

1. Take as input any arbitrary embedding of any arbitrary cubic planar graph, call it G. Let x be any arbitrary vertex on G's perimeter. Let a, b, and c be x's neighbors in a counter-clockwise order as shown in Figure 4.3 and orient G so that x faces East.

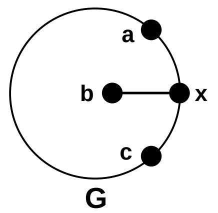

Figure 4.3: The Mirroring Procedure, Step 1

2. Create a mirror-image copy of G called G^* by creating a line of reflection near x as indicated in Figure 4.4. Label G^*'s vertices after their counterparts in G.

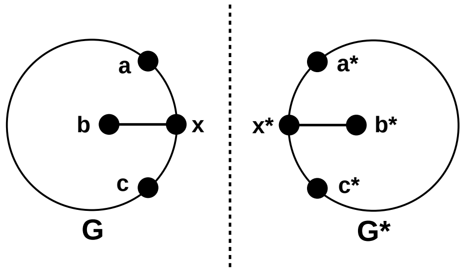

Figure 4.4: The Mirroring Procedure, Step 2

3. Reflect G^* from top to bottom to create an alternate embedding as shown in Figure 4.5.

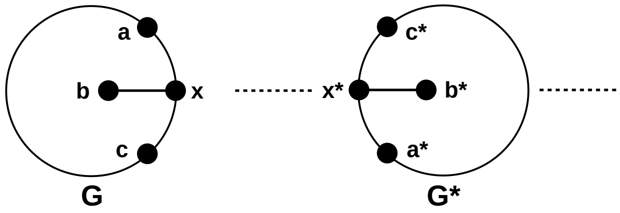

Figure 4.5: The Mirroring Procedure, Step 3

4. Delete x and x^* together with all edges incident on them and create edges (a, c^*), (b, b^*), and (c, a^*), thereby creating a new graph called G' as indicated in Figure 4.6.

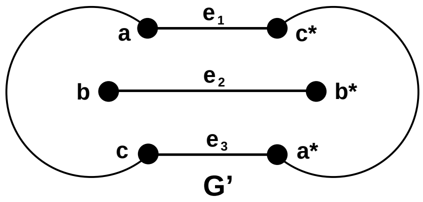

Figure 4.6: The Mirroring Procedure, Step 4

Lemma 4.3.2 *If the Mirroring Procedure is applied to any Barnette graph, the result will also be a Barnette graph.*

Proof: Suppose we apply the Mirroring Procedure to some arbitrary Barnette graph, call it G. By Lemma 4.3.1 we know that every face of G must have even degree. Steps

2 and 3 of the procedure create a Barnette graph to which Lemma 4.3.1 obviously also applies. For the following argument, please refer to Figure 4.7.

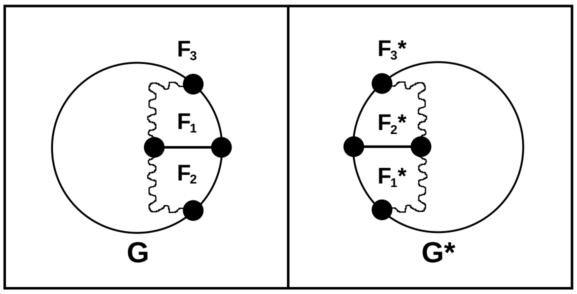

Figure 4.7: The Faces Affected By The Mirroring Procedure

Step 4 joins F_1 and F_2^* to create $F_\alpha^{'}$, F_2 and F_1^* to create $F_\beta^{'}$ as well as F_3 and F_3^* to create $F_\gamma^{'}$. Let f_i represent the face degree of F_i, let f_i^* represent the face degree of F_i^*, and let $f_i^{'}$ represent the face degree of $F_i^{'}$. $f_\alpha^{'} = f_1$ - 2 + f_2^* - 2 + 2, which is even, since f_1 and f_2^* are even. $f_\beta^{'} = f_2$ - 2 + f_1^* - 2 + 2, which is even since f_2 and f_1^* are even. $f_\gamma^{'} = f_3$ - 2 + f_3^* - 2 + 2, which is even, since f_3 and f_3^* are even. Since these are the only faces which have been modified, every face in G' must have even degree. By Lemma 4.3.1, G' is therefore bipartite. In addition, any damage done to 3-connectivity is immediately repaired. The procedure also clearly preserves planarity, and maintains every vertex having degree 3. Therefore, G' is a Barnette graph, as required. $\square$

Theorem 4.3.1 *Barnette's Conjecture holds if and only if every Barnette graph is Hamiltonian-Edge-Connected.*

Proof: $\Rightarrow$ Suppose Barnette's Conjecture holds. For the sake of contradiction, suppose there exists some Barnette graph, call it G, such that G contains an edge between two vertices, call them x and a, which is part of no Hamiltonian Cycle. Since G is planar, it is possible to embed it such that x and a are on the outside face. Let x's other neighbors be called b and c as indicated in Figure 4.3. Suppose we create a new graph G^* by applying the first 3 steps of the mirroring procedure to G. Since there exists no Hamiltonian Path between a and x, if we were to remove x from G, the resulting graph, call it G_1, would neither contain a Hamiltonian Path from a to b, nor would it contain a Hamiltonian Path from a to c. Similarly, if we were to remove x^* from G^*, the resulting graph, call it G_2, would neither contain a Hamiltonian Path from a^* to b^*, nor would it contain a Hamiltonian Path from a^* to c^*. We now complete the Mirroring Procedure to create G', which by Lemma 4.3.2 is also a Barnette graph. Consider the cut across the three newly-created edges labeled e_1, e_2, and e_3 in G' as shown in Figure 4.6. Any Hamiltonian Cycle in G' would have to cross exactly two of those three edges. Such a Hamiltonian Cycle cannot cross e_1 and e_2 because there is no Hamiltonian Path from a to b in G_1. It cannot cross e_1 and e_3 because neither is there a Hamiltonian Path from a to c in G_1, nor is there a Hamiltonian Path from c^* to a^* in G_2. Finally, it cannot cross e_2 and e_3 since there is no Hamiltonian Path from a^* to b^* in G_2. Therefore, G' is a non-Hamiltonian Barnette graph, which is a contradiction. Therefore every edge in any Barnette graph must be part of some Hamiltonian Cycle, as required.

$\Leftarrow$ Trivial. $\square$

Corollary 4.3.1 *If some edge in some Barnette graph is part of no Hamiltonian Cycle, then Barnette's Conjecture does not hold.*

Corollary 4.3.2 *If Barnette's conjecture holds, then every pair of adjacent vertices in any Barnette graph has a Hamiltonian Path between them.*

4.4 Future Research

Even if this result does not yield a counterexample to Barnette's Conjecture, further strengthening results can only help. For example, we conjecture that Barnette's Conjecture holds if and only if every pair of adjacent edges in any Barnette graph is part of some Hamiltonian Cycle. One corollary to this result would be that if Barnette's Conjecture holds, then it is possible to delete any single edge out of a Barnette graph without it losing its Hamiltonicity. In other words, Barnette's Conjecture holds if and only if for every Barnette graph, every edge is in some Hamiltonian Cycle, and no edge is in every Hamiltonian Cycle. Such a robust characteristic in sparse, planar graphs may have some practical applications if the Conjecture turns out to be true.

Since all graphs with 84 or fewer vertices are known to be Hamiltonian, the days of computers being very helpful with Barnette's Conjecture may be drawing to a close. Not only is the number of graphs to be investigated becoming enormous, but any potential candidates are becoming so large that proving non-Hamiltonicity may be very difficult in a computational sense. It stands to reason that future research will involve more theoretical analysis and proportionally less computer power.

Chapter 5

Concluding Remarks

Apart from the potential future research mentioned in Sections 3.3, 3.4, and 4, there are a few other ways in which this research might proceed.

We have seen that the Hamiltonian Cycle problem is currently of interest in the areas of graph theory, algorithms, and complexity theory. The Look-Ahead Theorem gives us a tool for also taking it into the area of proof complexity. If forcing an edge in a graph causes the graph to become non-Hamiltonian, then we may safely prune it. Likewise, if deleting an edge causes the graph to become non-Hamiltonian, then we may safely force it. Corollaries 3.1.3 and 3.1.4 state that for any non-Hamiltonian graph, any edge may be safely pruned, and likewise may be safely forced. Therefore, in order to create a proof of non-Hamiltonicity, we need only force and prune edges in some way that creates some obstruction to Hamiltonicity such as a hub. Of course, we would have to prove that these forcings and prunings are legitimate by showing that their resulting graphs are non-Hamiltonian. This can be done recursively by building a tree whose root contains the original graph in question. The first level of the tree contains $O(m)$ vertices, one corresponding to the graph created by deleting one of each of the unforced edges in the root. Similarly, the second level of the tree contains $O(m^2)$ vertices, and so on. As we explore deeper into the tree, we are likely to find more non-Hamiltonian graphs / tree vertices. The goal of this research would be to see how powerful this proof system is. That is, for any non-Hamiltonian graph, can it always give a polynomially-sized proof of non-Hamiltonicity, or are there some families of non-Hamiltonian graphs that force it to give exponentially large proofs? A similar line of research is to compare this proof system to other proof systems whose complexities are known, and see where it fits in. If it is very powerful, it may be equivalent to a proof system such as Frege for which no lower bounds are known. Ultimately, this line of research is addressing whether $\mathcal{NP} =$ co$\mathcal{NP}$, because polynomially-bounded proofs for non-Hamiltonicity are certificates for 'no' instances similar to how Hamiltonian Cycles are certificates for 'yes' instances of the problem. Many theoretical computer scientists conjecture that $\mathcal{NP} \neq$ co$\mathcal{NP}$, but not with nearly as much conviction as when they conjecture that $\mathcal{P} \neq \mathcal{NP}$.

Bibliography

[1] T. Akiyama, T. Nishizeki, and N. Saito. NP-Completeness of the Hamiltonian Cycle Problem for Bipartite Graphs. *Journal of Information Processing*, Vol. 3 No. 2:73–76, 1980.

[2] K. Appel, W. Haken, and J. Koch. Every Planar Map Is Four-Colorable, Parts I & II. *Illinois J. Math.*, 21:429 – 567, 1977.

[3] W.W. Rouse Ball and H.S.M. Coxeter. *Mathematical Recreations and Essays*. Macmillan, New York, 1962.

[4] D. Barnette. Conjecture 5. In W.T. Tutte, editor, *Recent Progress in Combinatorics*, page 343. Academic Press, New York, 1969.

[5] D. Bauer, J. van den Heuvel, A. Morgana, and E. Schmeichel. The Complexity of Toughness in Regular Graphs. *Congr. Numer.*, 130:47–61, 1998.

[6] C. Berge. *Graphs & Hypergraphs*. North Holland, Amsterdam, 1973.

[7] N.L. Biggs, E.K. Lloyd, and R.J. Wilson. *Graph Theory 1736 - 1936*. Clarendon Press, Oxford, 1976.

[8] J.A. Bondy and V. Chvátal. A Method in Graph Theory. *Discrete Math.*, 15:111–135, 1976.

[9] J.A. Bondy and U.S.R Murty. *Graph Theory With Applications*. Macmillan, London, 1976.

[10] G. Brinkmann, B.D. McKay, and U. von Nathusius. Backtrack Search and Lookahead for the Construction of Planar Cubic Graphs With Restricted Face Sizes. Published On Internet: http://cs.anu.edu.au/~bdm/papers/plantri_ad.pdf.

[11] N. Chiba and T. Nishizeki. The Hamiltonian Cycle Problem is Linear-Time Solvable for 4-Connected Planar Graphs. *Journal of Algorithms*, 10:87–211, 1989.

[12] V. Chvátal. Hamiltonian cycles. In E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, editors, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, pages 403–429. John Wiley & Sons Ltd., New York, 1985.

[13] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* McGraw-Hill, Cambridge, Massachusetts, 1990.

[14] G.A. Dirac. Some Theorems On Abstract Graphs. *Proceedings Of The London Mathematics Society*, 2:69–81, 1952.

[15] S. Effler. Enumeration, Isomorphism and Hamiltonicity of Cayley Graphs: 2-Generated and Cubic. Master's thesis, University of Victoria, 2000.

[16] M.N. Ellingham and J.D. Horton. Non-Hamiltonian 3-Connected Cubic Bipartite Graphs. *Journal of Combinatorial Theory, Series B*, 34:350–353, 1983.

[17] H. Eves. *An Introduction To The History of Mathematics, Fourth Ed.* Saunders College Publishing, Orlando, Florida, 1992.

[18] T. Feder and C. Subi. On Barnette's Conjecture. Published On Internet: http://theory.stanford.edu/ tomas/bar.ps. Submitted to Journal of Graph Theory.

[19] A.M. Frieze. Finding Hamilton Cycles in Sparse Random Graphs. *Journal of Combinatorial Theory, Series B*, 44:230–250, 1988.

[20] A.M. Frieze, M. Jerrum, M. Molloy, R. Robinson, and N. Wormald. Generating and Counting Hamilton Cycles in Random Regular Graphs. *Journal of Algorithms*, 21:176–198, 1996.

[21] M.R. Garey and D.S. Johnson. *Computers and Intractability -A Guide to the Theory of NP-Completeness*. Freeman & Company, New York, 1979.

[22] M.R. Garey, D.S. Johnson, and R.E. Tarjan. The Planar Hamiltonian Circuit Problem is NP-Complete. *SIAM Journal of Computing*, Vol. 5 No. 4:704–714, 1976.

[23] P.R. Goodey. Hamiltonian Circuits in Polytopes With Even Sided Faces. *Israel Journal of Mathematics*, 22:52–56, 1975.

[24] P.R. Goodey. A Class of Hamiltonian Polytopes. *Journal of Graph Theory*, 1:181–185, 1977.

[25] D. Gouyou-Beauchamps. The Hamiltonian Circuit Problem is Polynomial for 4-Connected Planar Graphs. *SIAM Journal of Computing*, 11:529–539, 1982.

[26] R.P. Graves. *Life of Sir William Rowan Hamilton, Vol. 3*. Dublin University Press, Dublin, 1889.

[27] H. Gropp. Configurations and the Tutte Conjecture. *Ars Combinatoria A*, 29:171–177, 1990.

[28] W.R. Hamilton. Account of the Icosian Calculus. *Philosophical Transactions of the Royal Irish Academy*, 6:415–416, 1853-7.

[29] W.R. Hamilton. Memorandum Respecting a New System of Roots of Unity. *Phil. Mag. (4)*, 12:446, 1856.

[30] K. Helsgaun. An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic. *European Journal of Operational Research*, 126:106–130, 2000.

[31] D.A. Holton and E.L. Aldred. Planar Graphs, Regular Graphs, Bipartite Graphs and Hamiltonicity. *Australasian Journal of Combinatorics*, 20:111–131, 1999.

[32] D.A. Holton, B. Manvel, and B.D. McKay. Hamiltonian Cycles in Cubic 3-Connected Bipartite Planar Graphs. *Journal of Combinatorial Theory, Series B*, 38:279–297, 1985.

[33] S. Janson, T. Luczak, and Andrzej Rucinski. *Random Graphs*. John Wiley and Sons, New York, 2000.

[34] R.M. Karp. Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.

[35] B.W. Kernighan and S. Lin. An Efficient Heuristic Algorithm For The Traveling-Salesman Problem. *Oper. Res.*, 2:498–516, 1973.

[36] T.P. Kirkman. On the Representation of Polyedra. *Philosophical Transactions of the Royal Society of London*, 146:413–418, 1856.

[37] M.S. Krishnamoorthy. An NP-Hard Problem in Bipartite Graphs. *SIGACT News*, Vol. 7, No. 1:26, 1975.

[38] C.L. Liu. *Introduction to Combinatorial Mathematics*. McGraw-Hill, New York, 1968.

[39] B.D. McKay, G. Brinkmann, and R. Aldred. Personal Communication, March 2000.

[40] C.H. Papadimitriou and K. Steiglitz. Some complexity results for the traveling salesman problem. In ACM, editor, *Proc. 8th Ann. ACM Symp. on Theory of Computing*, pages 1–9. ACM, New York, 1976.

[41] R.W. Robinson and N.C. Wormald. Almost All Cubic Graphs Are Hamiltonian. *Random Structures and Algorithms*, 3:117–126, 1992.

[42] R.W. Robinson and N.C. Wormald. Almost All Regular Graphs Are Hamiltonian. *Random Structures and Algorithms*, 5:363–374, 1994.

[43] K.H. Rosen. *Discrete Mathematics and Its Applications, Fourth Ed.* McGraw-Hill, New York, 1999.

[44] M. Rosenfeld. Personal Communication, November 2003.

[45] P.G. Tait. Listing's Topologie. *Phil. Mag.*, 17:30–46, 1884.

[46] C. Thomassen. A Theorem on Paths in Planar Graphs. *Journal of Graph Theory*, 7:169–176, 1983.

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

[48] W.T. Tutte. A Theorem on Planar Graphs. *Trans. Amer. Math. Soc.*, 82:99–116, 1956.

[49] W.T. Tutte. On the 2-Factors of Bicubic Graphs. *Discrete Mathematics*, Vol. 1, No. 2:203–208, 1971.

[50] B. Vandegriend. Finding Hamiltonian Cycles: Algorithms, Graphs and Performance. Master's thesis, University of Alberta, 1998.

[51] D.B. West. *Introduction to Graph Theory, 2nd. Edition.* Prentice Hall, Upper Saddle River, New Jersey, 2001.

[52] A. Wigderson. The Complexity Of The Hamiltonian Circuit Problem For Maximal Planar Graphs. *Technical Report, Princeton University, Dept. of EECS*, Vol. 298, February, 1982.

Glossary

The following definitions (in alphabetical order) may be helpful to the reader:

- **1-Tough:** A graph is said to be **1-tough** if it contains no set of k vertices that, if removed, would separate the graph into $k + 1$ or more components.
- **2-Factor:** A **2-factor** of a graph G is a subgraph in which every vertex has a degree of exactly 2. For example, a Hamiltonian Cycle is a 2-factor.
- **Barnette's Conjecture** states that all planar, cubic, 3-connected, bipartite graphs are Hamiltonian. For the sake of not having to constantly repeat these adjectives, these graphs will be referred to as **Barnette graphs** in this thesis.
- **Barricade:** A **barricade** is a forced edge incident on the two vertices of a 2-vertex-cut.
- **Biconnected: Biconnected** is synonymous with 2-Connected.
- **Bicubic:** A **bicubic** graph is both bipartite and cubic.
- **Bipartite:** A graph whose vertices can be coloured using exactly two different colours such that no two adjacent vertices have the same colour is called **bipartite**.
- **Chain:** A **chain** is a path containing q vertices subject to the following constraints: $2 \leq q < n$, and each of the $q - 1$ edges along the path is forced, the 2 endpoints have degree at least 3, and the remaining $q - 2$ vertices, if any, have a degree of exactly 2.

- **Cubic:** Synonymous with 3-regular, the adjective **cubic** indicates that a graph's vertices all have a degree of exactly 3.

- **Cut:** A k**-vertex-cut** is a subset of k vertices that, if removed, would disconnect the graph. A k**-edge-cut** is a similar subset of edges.

- **Connected:** A k**-connected** graph is one that cannot be disconnected by the removal of fewer than k vertices. Similarly, a k**-edge-connected** graph is one that cannot be disconnected by the removal of fewer than k edges.

- **Deletion Sequence:** A **deletion sequence** is a list of edges that are deleted from a graph in order. After each edge in a deletion sequence is removed, some predefined subset of pruning and forcing rules are executed. Edges deleted due to pruning are not considered to be part of the deletion sequence. Furthermore, the resulting graph depends very much on the order in which the edges of a deletion sequence are removed, showing that 'deletion sets' are insufficient for our needs.

- **Face Degree:** The number of **sides** or **degree** of a face both refer to the number of edges surrounding its perimeter.

- **Forced Edge:** Within the context of Hamiltonicity, a **forced edge** is a constraint put on a graph stipulating that every Hamiltonian Cycle must contain that edge. We say that a forced edge in a graph G is **safe** if the act of forcing it does not reduce the number of Hamiltonian Cycles that G contains.

- **Hamiltonian:** A graph that contains a Hamiltonian Cycle is said to be **Hamiltonian**.

- **Hamiltonian Cycle:** A **Hamiltonian Cycle** is a simple cycle which passes through every vertex in a graph exactly once.

- **Hamiltonian Connected:** A graph is said to be **Hamiltonian Connected** if each pair of vertices has a Hamiltonian Path between them.

- **Hamiltonian-Edge-Connected:** A graph is said to be **Hamiltonian-Edge-Connected** if each of its edges is part of some Hamiltonian Cycle.

- **Hamiltonian Path:** A **Hamiltonian Path** between two vertices u and v in a graph G is a path which contains one instance of every vertex and has endpoints u and v.

- **Head:** A **head** is one of the two endpoints of a chain.

- **Head Hunter: Head Hunter** is a pruning rule that, when given a chain whose two heads are adjacent, deletes the edge between them.

- **Hub:** A **hub** is a vertex with three or more forced edges incident on it.

- **Hypo-Hamiltonian:** A graph G is said to be **hypo-Hamiltonian** if it is not Hamiltonian, but the removal of any arbitrary vertex causes it to become Hamiltonian.

- **Lightning:** Named for its fast speed, **Lightning** is a pruning rule that, when given a vertex of degree at least 3 with exactly two forced edges incident on it, deletes all remaining unforced edges.

- **Obstruction to Hamiltonicity:** An **obstruction to Hamiltonicity** is a structure within a graph that acts as a certificate for non-Hamiltonicity. For example, a degree-1 vertex is an obvious obstruction.

- **Odd-Forced-Cut:** An **odd-forced-cut** is a cut across an odd number of edges, all of them forced.

- **Planar:** A **planar** graph is one that can be drawn on a two-dimensional plane such that no two edges cross.

- **Pruning: Pruning** is the act of deleting an edge which is part of no Hamiltonian Cycle. Pruning therefore does not affect the Hamiltonicity of a graph.

- **Regular:** An **r-regular** graph is one in which each vertex has degree r.

- **Sparse:** A graph is considered to be **sparse** if m is $O(n)$.

- **Triangulation:** A **triangulation** is a planar graph in which every face has exactly three sides.

- **Unevenly Bipartite:** A graph is said to be **unevenly bipartite** if it is bipartite, but the two bipartitions do not contain an equal number of vertices.