

ABSTRACT

SHIELDS, IAN BEAUMONT Hamilton Cycle Heuristics in Hard Graphs (Under the direction of Professor Carla D. Savage)

In this thesis, we use computer methods to investigate Hamilton cycles and paths in several families of graphs where general results are incomplete, including Kneser graphs, cubic Cayley graphs and the middle two levels graph. We describe a novel heuristic which has proven useful in finding Hamilton cycles in these families and compare its performance to that of other algorithms and heuristics. We describe methods for handling very large graphs on personal computers. We also explore issues in reducing the possible number of generating sets for cubic Cayley graphs generated by three involutions.

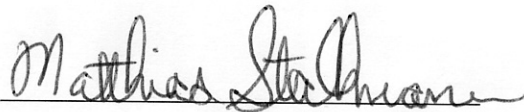
Hamilton Cycle Heuristics in Hard Graphs

by

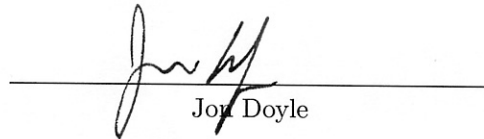
Ian Beaumont Shields

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy
Computer Science
Raleigh
2004

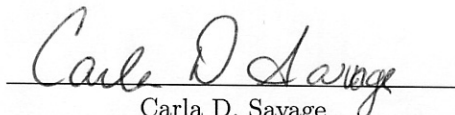
APPROVED BY:



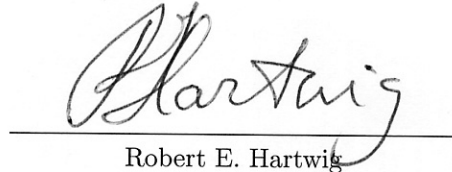
Matthias F. Stallmann



Jon Doyle



Carla D. Savage
Chair of Advisory Committee



Robert E. Hartwig

Dedication

To my wife Pat, and my children, Catherine, Brendan and Michael, for their understanding during these years of study.

To the memory of Hanna Neumann who introduced me to combinatorial group theory.

Biography

Ian Shields was born on March 10, 1949 in Melbourne, Australia. He grew up and attended school near the foot of the Dandenong ranges. After a short time in actuarial work he attended the Australian National University in Canberra, where he graduated in 1973 with first class honours in pure mathematics.

Ian worked for IBM and other companies in Australia, Canada and the United States, before resuming part-time study for a Masters Degree in Computer Science at North Carolina State University which he received in May 1995. He currently works for IBM's Software Group in Research Triangle Park, North Carolina.

Acknowledgements

I would like to thank the members of my committee for their work in reviewing this thesis. Particular thanks are due to my adviser, Dr. Carla Savage, for countless hours of review and encouragement, without which this work would never have been completed.

I would also like to thank IBM for their financial support for this degree, and the managers whose encouragement and understanding was so necessary for completing this work over a long period of time. In particular, I thank Howard Brown for the initial approval to work on a doctorate, Iva Anderson for some time in the middle, and Greg Meyer for his help and encouragement in the completion of this work.

Contents

List of Figures	vii
List of Tables	viii
1 Introduction	1
1.1 Definitions	3
1.2 Outline of Thesis	6
2 Previous Work on Hamilton Paths and Cycles	8
2.1 When is a Graph Hamiltonian?	8
2.1.1 Hamilton Cycles in General Graphs	9
2.1.2 Hamilton Cycles in Vertex Transitive Graphs and Cayley Graphs	13
2.1.3 Hamilton Cycles in Random Graphs	14
2.2 Finding Hamilton Paths and Cycles	16
2.2.1 Complexity	16
2.2.2 Algorithms and Heuristics	17
2.2.3 Using Random Graph Heuristics on Non-Random Graphs	21
3 The Hamilton Path Heuristic	22
3.1 Backtrack and Extension-Rotation Techniques	22
3.2 Improving the Heuristic	25
3.3 Running Time	30
3.4 Practical Enhancements	30
3.4.1 Dynamic Adjacency Calculation	31
3.4.2 Reducing BFS Tree Height	31
3.4.3 Cycle Extension	32
3.4.4 Random Backup	33
3.4.5 Verification	33
3.4.6 Checkpoint and Restart	33
3.5 The Current Heuristic	34
4 The Middle Two Levels Problem	36
4.1 Introduction to the Problem	36
4.2 Reducing the Middle Two Levels Problem	39
4.2.1 Definitions	39
4.2.2 The Reduction	40
4.2.3 Lifting Paths from the Reduced Graphs	40
4.3 Graph Representation	43
4.4 Performance of the Heuristic on the Middle Two Levels Graphs	43

5	Kneser Graphs	47
5.1	Kneser and Bipartite Kneser Graphs Overview	47
5.2	Kneser Graphs	48
5.3	Bipartite Kneser Graphs	49
5.4	Graph Representation	53
5.4.1	Fast Computation of Adjacencies in Kneser Graphs	54
5.5	Performance of the Heuristic on Kneser Graphs	55
6	Cayley Graphs	59
6.1	Introduction to Cayley Graphs	59
6.1.1	Hamiltonian Properties of Cayley Graphs	60
6.2	Hamiltonian Cubic Cayley Graphs	62
6.2.1	First Experiments	62
6.2.2	The Vandegriend Algorithm	63
6.2.3	The Work of Effler and Ruskey	63
6.2.4	Random Number Issues	64
6.3	Graph Representation	65
6.4	First Results for Cubic Cayley Graphs	65
6.5	Performance of the Heuristic on Cubic Cayley Graphs	65
6.6	Generating Graphs with Three Involutions	68
6.7	Further Results for Cubic Cayley Graphs	76
7	The Cover Graph of $M(n)$	79
7.1	Introduction	79
7.2	Necessary Conditions	81
7.3	Construction of Paths	82
8	Conclusion	92
8.1	Future Work	93
	Bibliography	95

List of Figures

1.1	A Hamilton path through the vertices of a dodecahedron.	2
2.1	Graphs related to $K_{1,3}$	12
2.2	The SELECT procedure of Angluin and Valiant	17
3.1	Backtrack search from s has reached u and cannot continue.	23
3.2	A path rotation at x : add edge ux and remove edge xy	23
3.3	The Pósa path rotation heuristic	24
3.4	Limitations of the PosaSearch heuristic	24
3.5	Finding the new position of a vertex	28
3.6	Finding the new vertex in a position	28
3.7	Performing the rotations	28
3.8	The Hamilton path BFS heuristic	29
3.9	Using cycle extension to transform a su -path to a zr -path	32
3.10	Create a cycle from a path or partial path	34
3.11	The Hamilton cycle heuristic	35
4.1	The Hasse diagram of \mathcal{B}_5 showing a Hamilton cycle through the middle two levels.	37
5.1	Inserting bits ($n = 9, k = 4$)	55
5.2	The Kneser adjacency algorithm	56
5.3	Intel assembly for the adjacency loop	56
6.1	The Petersen Graph	60
6.2	S_6 has cycles of length 4 and non-commuting generators	61
6.3	Triangular subgraphs in a cubic Cayley graph.	67
6.4	Finding sets of three involutions in S_n	73
6.5	Cycle of length 6 in a cubic graph.	78
7.1	Two views of the cover graph of $M(4)$	80
7.2	A_5 , the cover graph of $M(5)$	83
7.3	Construction of C_n in Lemma 8 when C_{n-1} contains $\{2, n-1\}\{3, n-1\}$	85
7.4	Construction of C_n in Lemma 8 when C_{n-1} contains $\{1, 2, n-1\}\{1, 3, n-1\}$	86
7.5	The Hamilton path Q_5 in G_5 (read across) in Lemma 9.	86
7.6	Construction of Q_n in Lemma 9.	87
7.7	The Hamilton path Q_5^* in G_5 (read left to right) in Lemma 10.	88
7.8	The Hamilton paths P_6 and R_6 (read across) in Lemma 11	89
7.9	Construction of P_n in Lemma 11 when $\binom{n+1}{2}$ is even and n is (a) odd, (b) even.	89
7.10	Construction of P_n and R_n in Lemma 11 when $\binom{n+1}{2}$ is odd and n is odd.	90

List of Tables

4.1	Running time to find a Hamilton cycle in the middle two levels graph	44
5.1	Known results for connected Kneser graphs $K(n, k)$ for $n \leq 27$	50
5.2	Known results for connected bipartite Kneser graphs $H(n, k)$ for $n \leq 27$	52
5.3	Heuristic running times for Kneser and bipartite Kneser graphs	58
6.1	Cubic Cayley graphs generated by one involution and one non-involution.	66
6.2	Involution and conjugacy statistics for permutation groups S_n	75
6.3	Characteristics of hard Cayley graphs with 5,040 vertices.	77
6.4	Characteristics of hard Cayley graphs with 20,160 vertices.	77

Chapter 1

Introduction

Intuitively, a Hamilton path is a path that visits every vertex of a graph exactly once. If there is an edge between the starting and ending point of a Hamilton path then the Hamilton path is a Hamilton cycle. Such cycles bear the name of the Irish mathematician, Sir William Rowan Hamilton, who demonstrated that the graph formed by the twenty vertices and thirty edges of a dodecahedron has a Hamilton cycle, as indeed do the corresponding graphs for each of the Platonic solids. The case of the dodecahedron is illustrated in Figure 1.1. Hamilton patented a game (the Icosian Game [57]) based on finding such a path, but did not prove any general results and may even have got the idea for the game from Kirkman who had earlier submitted a paper on polyhedra to the Royal Society. Euler had previously studied the knights tour problem on a chess board which is also a Hamilton cycle problem. Biggs, Lloyd and Wilson [8] have an interesting history of graph theory including a history of Hamilton cycles.

The general problem of finding a Hamilton path or cycle is known to be NP-complete. Polynomial time heuristics for searching for Hamilton paths and cycles exist and usually employ random techniques which succeed in finding a Hamilton cycle or path *almost surely* or *with high probability*.

We devised a new Hamilton cycle heuristic to use as a tool for insight into graph structure with the hope of using it to help with recursive constructions or the establishment of basis cases for inductive proofs of Hamiltonicity in special classes of graphs. We also used this heuristic to extend significantly the known results on Hamiltonicity in some classes of graphs where we did not find more general results.

Our heuristic worked extremely well on the *middle two levels problem* (Chapter 4), one example of a well-known class of hard graphs called *vertex-transitive graphs*. We investigated other vertex-transitive

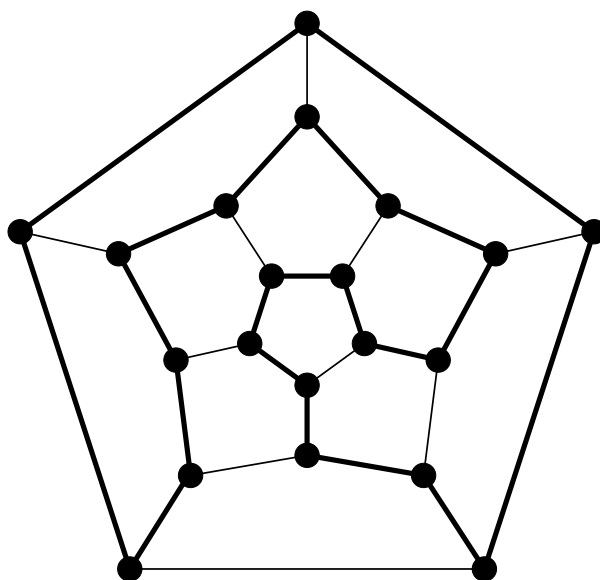


Figure 1.1: A Hamilton path through the vertices of a dodecahedron.

graphs to see how well our heuristic worked on them and what its limitations might be. Vertex-transitive graphs have additional interest because a famous conjecture, due to Lovász [75], that every connected, undirected, vertex transitive graph has a Hamilton path, has stood for over thirty years with neither a proof nor a counterexample.

The heuristic we devised terminates in polynomial time. It can be used to search for either Hamilton paths or Hamilton cycles. It may terminate without finding a path or cycle, although this does not mean that such a path or cycle does not exist.

Using innovative trade-offs in space and time, we have been able to find Hamilton cycles in some huge graphs. We showed the middle two levels graph, M_{2k+1} , has a Hamilton cycle for $k \leq 15$ if a certain Hamilton path exists in a particular quotient graph and used our heuristic to successfully find such paths. The graph M_{2k+1} has over 600 million vertices and the corresponding quotient graph has over 9 million vertices and degree 16.

We also used our heuristic to show that all Kneser graphs $K(n, k)$ (Chapter 5) for $n > 2k$ and their bipartite analogs $H(n, k)$ have Hamilton cycles for $n \leq 27$. These are also very large graphs. $K(27, 13)$ has over 20 million vertices and is of degree 14 while $K(27, 10)$ has over 8 million vertices and degree in excess of 19,000.

These results did not require a supercomputer, but were achieved on an ordinary personal computer.

Finding Hamilton paths or cycles in even relatively small graphs is often a difficult task. We used our heuristic to find certain paths in the *cover graph* A_n (Chapter 7), for small n , and thereby were able to show that, for all n , A_n has a Hamilton path if and only if $\binom{n+1}{2}$ is odd and $n \neq 5$.

The heuristic uses pseudo-random numbers to make some decisions. When a run does not succeed, the program can be restarted with a different seed for the pseudo-random number generator, thus allowing many different attempts at a problem. Using this capability we were able to find cycles in cubic Cayley graphs that were not found with just one run, although some graphs (Chapter 6) did not yield cycles even after 100,000 such attempts.

1.1 Definitions

We use West [108] for basic terminology and notation not defined here.

Definition 1 *An undirected graph G consists of a set of vertices $V(G) = \{v_1, v_2, \dots, v_n\}$ and a set of*

edges $E(G) = \{e_1, e_2, \dots, e_m\}$ where each edge is an unordered pair of vertices. We will write uv , or (u, v) , for the edge $\{u, v\}$ and we say that u and v are the endpoints of this edge. If $uv \in E(G)$ then we say that u and v are adjacent or that they are neighbors. We also write $u \leftrightarrow v$ to mean u is adjacent to v . If there is at most one edge uv for any pair of vertices u and v and there are no edges uu for any vertex u we say that the graph is a simple graph. If an edge uu exists for any vertex u we say that such an edge is a loop.

Let $n = |V(G)|$. Then G is said to be of order n .

If $uv \in E(G)$ for every pair $u, v \in V(G)$ and $n = |V(G)|$, then G is the complete graph, K_n . A complete graph is also called a *clique*.

A graph H is a *subgraph* of a graph G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. A subgraph H of G is an *induced subgraph* of G if, for every pair of vertices, $u, v \in V(H)$, $uv \in E(G)$ implies $uv \in E(H)$.

The *degree*, $\deg(v)$, of a vertex, $v \in G$, is the number of non-loop edges incident with it plus twice the number of loop edges. The minimum degree of any vertex in the graph, G , we denote by $\delta(G)$ and the maximum degree by $\Delta(G)$. A graph, G , is *k-regular* if $\delta(G) = \Delta(G) = k$.

In studying sets of graphs on n vertices, $V(G) = \{v_1, v_2, \dots, v_n\}$, we may assign the *label*, i , to the vertex v_i . A *labeled graph* is then one of the possible graphs on these n vertices. There are $\binom{n}{2}$ possible sets of edges on n vertices, so there are $2^{\binom{n}{2}}$ possible labeled graphs on a set of n vertices. Labeled graphs are used frequently in the study of random graphs. See Chapter 2 for further background material on random graphs.

Unless otherwise noted, all our results are for undirected simple graphs.

Definition 2 Given a graph G , an ordered list $P = (v_1, v_2, \dots, v_l)$ of vertices of $V(G)$ is a v_1, v_l -path of length $l - 1$ in G if $v_i \neq v_j$ for all $i \neq j$ and $(v_i, v_{i+1}) \in E(G)$ for $1 \leq i < l$. A path P of length l in G is a Hamilton path if $l = |V(G)| - 1$. A cycle of length l is a v_1, v_l -path of length $l - 1$ in which $(v_1, v_l) \in E(G)$. A Hamilton cycle is a Hamilton path that is a cycle. We say that a graph G is Hamiltonian if and only if it contains a Hamilton cycle.

Definition 3 If there is a path in G from u to v then the distance from u to v , denoted $d_G(v)$ or, more simply, $d(v)$ is the least length of a u, v -path. If there is no such path then $d(u, v) = \infty$.

If u and v are neighbors, then $d(u, v) = 1$.

A *vertex cut* (or *separating set*) of a graph G is a set $S \subseteq V(G)$ such that the subgraph induced by $\{V(G) \setminus S\}$ has more than one component. As is customary, we define the *connectivity*, $\kappa(G)$ to be the

size of the smallest vertex cut of G , with $\kappa(G) = |V(G)|$ if G has no vertex cut. A graph is k -connected if $\kappa(G) \geq k$. Thus $\kappa(G)$ is the largest value, k , for which G is k -connected.

An *independent set* of a graph G is a set $S \subseteq V(G)$, such that the graph $G(S)$ induced by S has no edges. The *independence number*, $\alpha(G)$, is defined to be the size of the largest independent set in G .

We define the *neighborhood*, $N(v)$, of a vertex v of G as follows:

$$N(v) = \{x \in V(G) | xv \in E(G)\}.$$

If $S \subseteq V(G)$ then the neighborhood of S is defined as $N(S) = \cup_{v \in S} N(v)$ and we say that the (*generalized*) *degree of S* , $\deg(S)$, is $|N(S)|$.

Definition 4 An isomorphism from a graph G to a graph H is a bijection $f : V(G) \rightarrow V(H)$ such that $uv \in E(G)$ if and only if $f(u)f(v) \in E(H)$. We write $G \cong H$ and say that G is isomorphic to H . An isomorphism from G to G is called an automorphism of G .

Evidently an automorphism of G is a permutation of $V(G)$.

Definition 5 A graph G is vertex-transitive if, for every pair $u, v \in V(G)$, there is an automorphism of G that maps u to v .

Evidently, a vertex-transitive graph is regular, that is $\Delta(G) = \delta(G)$. If a graph is vertex-transitive then any property that holds for one vertex holds for all vertices.

We now introduce a special class of vertex-transitive graphs called *Cayley graphs*.

Definition 6 Let G be a group and $X \subseteq G$. The Cayley digraph $\bar{\Gamma} = \text{Cay}(G : X)$ is defined by:

$$V(\bar{\Gamma}) = G; \quad E(\bar{\Gamma}) = \{(u, ux) : u \in G, x \in X\}.$$

The digraph $\bar{\Gamma}$ is strongly connected if and only if S is a generating set for G and loopless if and only if $e \notin S$ where e is the identity element of G . Either or both of these restrictions appear in some definitions of the Cayley digraph. Since our interest is in Hamilton cycles, it will be convenient to assume that both restrictions apply unless explicitly stated otherwise.

If, for every $x \in X$, the inverse $x^{-1} \in X$, then both $\{u, ux\}$ and $\{ux, u\}$ are edges of $\bar{\Gamma}$. This gives rise to the undirected analog of a Cayley digraph.

Definition 7 Let G be a group and $X \subseteq G$ with $e \notin X$. The Cayley graph $\Gamma = \text{Cay}(G : X)$ is defined by:

$$V(\Gamma) = G; \quad E(\Gamma) = \{(u, ux), (ux^{-1}) : u \in G, x \in X\}.$$

We will explicitly state whether the Cayley graph or digraph is involved unless a result applies to both types.

If X is a set of generators for a group G and $e \in G$ is the group identity element, we say that $x \in X, x \neq e$, is an *involution* if $x^2 = e$.

1.2 Outline of Thesis

In the remaining sections of this thesis we cover the following areas.

Hamilton cycles and Hamilton paths have been widely studied. Chapter 2 provides additional background on Hamilton cycle results and the previous work, in both non-random and random graphs, that has motivated our work.

Chapter 3 describes our heuristic which is a refinement of the path rotation technique of Pósa [82], augmented by a breadth-first search technique to determine a sequence of rotations that will allow a partial path to be extended. We add other features to produce an algorithm that has been successful at finding Hamilton cycles in several types of graphs.

Chapter 4 describes our work on the *middle two levels problem*. The middle two levels problem is to find a Hamilton path or cycle in the graph M_{2k+1} induced by the middle two levels of the Hasse diagram, $H(\mathcal{B}_{2K+1})$, which is the covering graph of the $2k + 1$ -atom Boolean lattice, \mathcal{B}_{2k+1} . We show the existence of Hamilton cycles for $k \leq 15$. For $k = 15$ we reduce the problem of finding a Hamilton cycle in a graph with over 600,000,000 vertices to a problem of finding a Hamilton path in a graph with almost 10,000,000 vertices and find such paths in the reduced graph using our heuristic.

Chapter 5 describes our work on connected Kneser graphs $K(n, k)$, and bipartite Kneser graphs, $H(n, k)$. We show that the connected Kneser Graph $K(n, k)$ and the bipartite Kneser graphs, $H(n, k)$ are Hamiltonian for all values of $n \leq 27$. Since the bipartite Kneser graph $H(2k + 1, k)$ is an example of the middle two levels problem, the work of Chapter 4 extends the results for this case to $n \leq 31$. We devise a fast algorithm for calculating neighbors, which allows us to find cycles in graphs with over 80 billion edges.

Chapter 6 describes our work on cubic Cayley graphs derived from subgroups of the symmetric

group S_n which can be generated by three involutions or by one involution and one other group element that is not an involution. These graphs proved hardest for our heuristic and we examine characteristics of graphs in which we did not find cycles. We devise an algorithm to significantly reduce the size of the problem space by creating non-conjugate sets of three involutions in S_n . We show that all Cayley graphs generated by three involutions from S_n with $n \leq 7$ are Hamiltonian and we describe our results for Cayley graphs generated by three involutions from S_8 .

Chapter 7 describes our work on the *cover graph*, A_n , the Hasse diagram of the poset $M(n)$ which has as its elements the n -tuples of integers $\mathbf{a} = (a_1, a_2, \dots, a_n)$ satisfying $0 = a_1 = \dots = a_j < a_{j+1} < \dots < a_n \leq n$ for some $j \geq 0$ with an order relation defined by $\mathbf{a} \leq \mathbf{b}$ if and only if $a_i \leq b_i$ for $1 \leq i \leq n$. We completely characterize those values of n for which A_n has a Hamilton path and those for which it does not.

Chapter 8 summarizes our conclusions and gives suggestions for future work.

Chapter 2

Previous Work on Hamilton Paths and Cycles

Hamilton cycles and Hamilton paths have been widely studied and there are several surveys of the subject in the literature, including those of Curran and Gallian [27], Faudree [44], Gould [52], Witte and Gallian [111], Dean et al. [29] and Lesniak [72]. In this chapter we review some of the main results concerning conditions for a graph to have a Hamilton path or cycle and then review techniques that have been used to find Hamilton paths or cycles.

2.1 When is a Graph Hamiltonian?

Most existence conditions for general graphs depend on *degree sums*, *neighborhood unions*, *independence numbers* or *forbidden subgraphs*. Using these techniques, a graph is usually provably Hamiltonian only when there are sufficiently many edges in the graph. Yet such results are often best possible in the sense that counterexamples exist when the conditions are weakened.

Erdős and Rényi founded the theory of random graphs in a series of papers, starting with [37] and [38]. Results for random graphs are couched in terms of *almost all* graphs of some particular (infinite) class of random graphs being Hamiltonian. In contrast to the general situation, results for random graphs show that even sparse graphs are almost always Hamiltonian for many interesting classes of graphs. Using ideas from the algorithms used to prove facts about random graphs we create a heuristic that is effective at finding Hamilton paths and cycles in several classes of non-random graphs with low

edge density, for which general results are incomplete or unknown.

The general problem of finding a Hamilton path or cycle is known to be NP-complete. As we shall see later, polynomial time heuristics for finding Hamilton paths and cycles are rooted in the theory of random graphs and usually employ techniques which succeed in finding a Hamilton cycle or path *almost surely* or *with high probability*.

We begin with a review of the major Hamilton path and cycle results and then focus on results for random graphs as many of the techniques that we have used come from strategies to prove results about random graphs.

2.1.1 Hamilton Cycles in General Graphs

Degrees, Degree Sums and Neighborhood Unions

Much of the early study of Hamiltonian graphs focused on the edge density of the graph and these results usually apply to fairly dense graphs. One of the earliest such conditions for a graph to be Hamiltonian is due to Dirac [34] and depends solely on the minimum degree of vertices.

Theorem 1 *If G is a graph of order $n \geq 3$ such that $\delta(G) \geq n/2$ then G is Hamiltonian.*

Ore [81] extended this result by considering the degree sum of non-adjacent vertices.

Theorem 2 *If G is a graph of order $n \geq 3$ such that $\deg(u) + \deg(v) \geq n$ for each pair u, v of non-adjacent vertices then G is Hamiltonian.*

Ore's relaxation stimulated several refinements culminating in the important result of Bondy and Chvátal [14] relating Hamiltonian properties to the closure of a graph. Define the k -closure of G , denoted by $C_k(G)$ as the graph obtained by repeatedly adding an edge between pairs of non-adjacent vertices whose degree sum is k until no such pair remains. Bondy and Chvátal showed the following:

Theorem 3 *A graph G of order n is Hamiltonian if and only if $C_n(G)$ is Hamiltonian.*

Henceforth we shall refer to $C_n(G)$ simply as the *closure* of G . The closure property is important because it allows the Hamiltonian problem for one graph to be determined in terms of the Hamiltonian property of a denser graph. In particular, if the closure of a graph is a clique then the graph is Hamiltonian. In other cases, and in particular for sparse graphs, this may not be helpful.

Chvátal [22] further generalized the degree conditions as follows:

Theorem 4 *Every graph on $n \geq 3$ vertices having vertex degree sequence $d_1 \leq d_2 \leq \dots \leq d_n$ satisfying $(d_k \leq k < \frac{1}{2}n) \Rightarrow (d_{n-k} \geq n - k)$ is Hamiltonian.*

Chvátal also showed that this condition was best possible by showing that if the degree sequence d_1, d_2, \dots, d_n fails to satisfy this condition, then there exists a graph G^* with degree sequence $d_1^*, d_2^*, \dots, d_n^*$, each $d_i^* \geq d_i$, such that G^* is not Hamiltonian.

The closure of a graph satisfying the conditions of Dirac, Ore or Chvátal is a clique. Fan [41] further generalized these results and introduced the idea of considering distance and local structure rather than conditions that hold for every pair of non-adjacent vertices.

Theorem 5 *If G is a 2-connected graph of order n such that*

$$\min\{\max(\deg u, \deg v) \mid d(u, v) = 2\} \geq n/2$$

then G is Hamiltonian.

In contrast to the earlier results, Theorem 5 does not imply that the closure of G is complete.

Further work in this direction considered the cardinality of neighborhoods of non-adjacent vertices.

The *neighborhood cardinality* of a graph G , $NC(G)$ is defined as:

$$NC(G) = \min\{|N(u) \cup N(v)| \mid uv \notin E(G)\}.$$

We may restrict this idea of neighborhood cardinality to vertices a specific distance, d , apart, in which case we define $NC_d(G)$ as

$$NC_d(G) = \min\{|N(u) \cup N(v)| \mid d(u, v) = d\}.$$

Faudree, Gould, Jacobson, and Schelp [43], Jackson [62], Broersma, van den Heuvel and Veldman [16], Lindquister [73], and Shen and Tian [97] have used the idea of $NC(G)$ or $NC_2(G)$ to derive Hamiltonian results. Liu, Zhu and Zhang [74] generalized these results to show:

Theorem 6 *Let G be a 2-connected graph of order n with $NC_2(G) \geq n/2$, then G is Hamiltonian, the Petersen graph, or a member of one of three exceptional families of non-Hamiltonian graphs.*

Jackson [61] has shown that a k -regular graph G of order n is Hamiltonian if $k \geq n/3$. The best condition of the neighborhood cardinality condition of Theorem 6 for a k -regular graph G of order n is that $k > n/4$.

Thus, in the general case, we see that proofs of Hamiltonicity in arbitrary graphs require a large average degree, even when a condition of regularity is imposed.

Independence Numbers

Some work has been done on relating the independence number $\alpha(G)$ of a graph to its Hamilton properties. Chvátal and Erdős [23] proved such a theorem which relates the vertex connectivity, $\kappa(G)$, of a graph G and its independence number.

Theorem 7 *Let G be a graph with at least three vertices. If $\kappa(G) \geq \alpha(G)$ then G is Hamiltonian.*

The result was strengthened by Lu [76] who showed:

Theorem 8 *For a non-empty vertex subset S of a simple graph G , let*

$$d(S, \bar{S}) = |\bar{S} \cap N(S)|$$

denote the number of neighbors of S in \bar{S} . Let $\theta(G) = \min\{d(S, \bar{S})/|\bar{S}|; \emptyset \neq S \subset V\}$. If G has order n and $\alpha(G) \leq n\theta(G)$, then G is Hamiltonian.

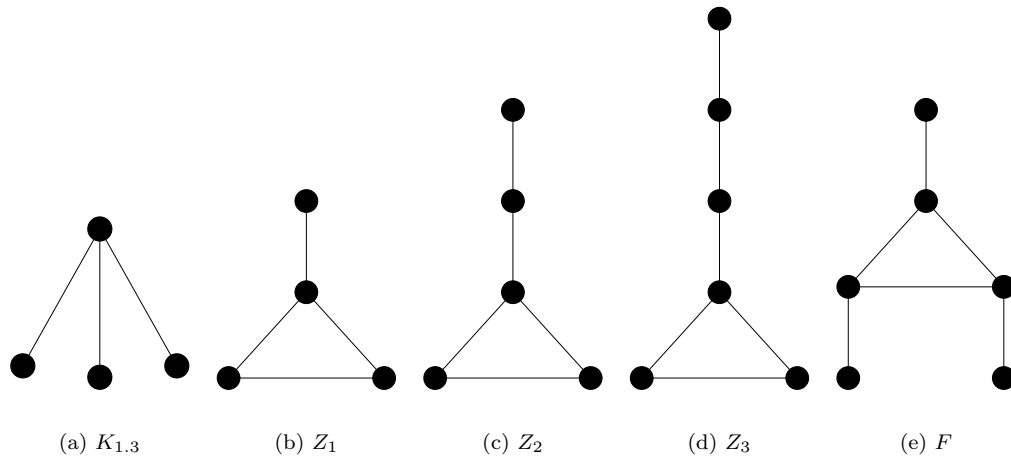
Fournier and Fraisse [47] also investigated independent sets and cycle properties and proved a conjecture due to Bondy.

Theorem 9 *If G is a k -connected graph of order n , where $k \geq 2$, and the degree-sum of any $k + 1$ independent vertices is at least m , then G contains a cycle of length at least $\min(2m/(k + 1), n)$.*

For any connected graph, G , $\kappa(G) \leq \delta(G)$ so, once again, we see that a large average degree is required for these results to provide Hamiltonicity results.

Forbidden Subgraphs

Given graphs F_1, F_2, \dots, F_k , we say that G is $[F_1, F_2, \dots, F_k]$ -free if G contains no induced subgraph isomorphic to any F_i , ($1 \leq i \leq k$). Most forbidden subgraph results to date involve the complete bipartite graph $K_{1,3}$ or closely related graphs such as those shown in Figure 2.1.

Figure 2.1: Graphs related to $K_{1,3}$

These results are mainly inapplicable to our work, as a large number of the vertex transitive graphs that we studied have either $K_{1,3}$ or Z_1 as an induced subgraph. However, we borrow from the idea in our work on cubic Cayley graphs (Chapter 6) to reduce such graphs containing K_3 , or triangles, as induced subgraphs to a smaller triangle-free graph where our heuristic is more successful.

The notion of forbidden subgraphs has been combined with other ideas such as degree sums or additional local neighborhood properties. For a survey of Hamiltonicity and forbidden subgraphs see Faudree [42, 44] or Gould [52].

We have seen here a wide range of approaches to determining whether a graph has a Hamilton cycle or Hamilton path. Despite this breadth of previous knowledge, the graphs that we will study later in this thesis do not yield to these techniques.

2.1.2 Hamilton Cycles in Vertex Transitive Graphs and Cayley Graphs

The set of automorphisms of a graph G forms a group that is a subgroup of the symmetric group S_n where G is of order n . It is not surprising, therefore, that group theory and graph theory are both useful in studying vertex-transitive graphs and Cayley graphs. See Praeger, Li and Niemeyer [83] for a further exploration of the interactions between the two areas.

Lovász [75] conjectured the following:

Conjecture 1 *Every connected, undirected, vertex transitive graph has a Hamilton path.*

To date, only five vertex transitive graphs are known that do not have a Hamilton cycle. They are K_2 , the Petersen graph on 10 vertices, the Coxeter graph on 28 vertices and two graphs obtained from these latter two by substituting a triangle for each vertex. Thomassen (see [6]) conjectured the following:

Conjecture 2 *There are only a finite number of vertex-transitive graphs which are not Hamiltonian.*

Babai [3] gives a lower bound on the length of the longest cycle in a vertex transitive graph, G , of $(3|V(G)|)^{1/2}$.

Scapallato [95] has a lengthy work summarizing classifications, concepts and methods used in working on vertex transitive graphs. Even though vertex transitivity is quite a strong restriction on a graph's structure, the Hamiltonicity of the graphs that we will study is still unknown in the general case.

Much of the work on Hamiltonicity of Cayley graphs involves either a group of special order or a special type of group. We illustrate here with some examples.

Witte [110] showed that every Cayley (di)graph of an arbitrary p -group, p a prime, is Hamiltonian.

Marušič [77] proved that every vertex-transitive (di)graph of order p^k , p a prime, $k \leq 3$, is a Cayley (di)graph. Combining this with Witte's result, every such connected vertex-transitive graph is Hamiltonian.

Chen [21] generalized these two results and proved that every connected vertex-transitive graph and digraph of order p^4 , p a prime, is Hamiltonian.

The strongest such structural result is due to Keating and Witte [65].

Theorem 10 *Let G be a group (other than Z_3) where the commutator subgroup is cyclic of prime-power order. Then G is Hamiltonian.*

The question of whether such graphs are Hamiltonian when the commutator subgroup is cyclic of other than prime-power order is still open.

The surveys of Witte and Gallian [111] and Curran and Gallian [27] provide additional material on Cayley graphs.

We will return to the topic of vertex-transitive graphs when we study the *middle two levels problem* in Chapter 4 and *Kneser graphs* in Chapter 5 and we will study Cayley graphs in Chapter 6.

2.1.3 Hamilton Cycles in Random Graphs

There are three different probability models for random graphs. In model A each of the $\binom{n}{2}$ possible edges between n vertices is present with equal probability. Model A is also called the *edge density model*. In model B, on the other hand, each possible set of edges is chosen with equal probability. Model B is also called the *fixed size model*. Model C was introduced by Erdős and Rényi [38] and deals with digraphs. We will not consider it further here.

Bollobás [10] has shown that, under some very minimal assumptions if \mathcal{A} is a set of graphs with order n such that, in Model B, almost every graph has property Q , then also in model A almost every graph has property Q and vice-versa. For this reason Model A is often used because the computations are generally easier.

A random graph with n vertices and M edges is frequently denoted by $G_{n,M}$.

Erdős and Rényi [38] asked the question: for what function $f(n)$ does the probability that a random graph of order n with $f(n)$ edges is Hamiltonian tend to 1 as n tends to ∞ ?

A *factor* of a graph is a spanning subgraph and a *k-factor* is a spanning *k*-regular subgraph. Evidently a Hamiltonian graph of even order has a 1-factor. A 1-factor is also called a *perfect matching*. Erdős and Rényi [39] proved the following:

Theorem 11 *If $n = 2m$ and, as $n \rightarrow \infty$, $M = \frac{1}{2}n \log n + \omega(n)n$, where $\lim \omega(n) = +\infty$, then the probability that a random graph $G_{n,M}$ contains a 1-factor has limit 1 as $n \rightarrow \infty$.*

Komlós and Szemerédi [67] obtained the first bound on $f(n)$ as

$$f(n) < ne^{2\sqrt{\log n \times \log \log n}}.$$

They also noted that

$$n \log n < ne^{2\sqrt{\log n \times \log \log n}} < n^{1+\epsilon}$$

for large n . The proof first shows sufficient set connectivity properties to ensure that any maximum length path can almost surely be closed into a cycle and if this cycle is not a Hamilton cycle then the connectivity properties guarantee that some vertex is adjacent to a vertex on the cycle and thus the cycle can be broken into a path that is longer than the original one which was assumed to be maximum.

This result was soon improved by Pósa [82] and independently by Koršunov [71].

Theorem 12 *There exists a constant c such that almost all labeled graphs of order n having at least $cn \log n$ edges are Hamiltonian.*

Pósa proved Theorem 12 for random graphs of model A and model B. In doing so he introduced the idea of an *allowable transformation* for a path which we shall hereafter call a *path rotation*. Note that the same basic idea was also used in the Komlós and Szemerédi result. The rotation operation replaces one path with another path of equal length having the same vertices but a new endpoint. Thus a path that was maximal can be replaced by a path that can possibly be extended.

Formally, if v_1, v_2, \dots, v_n is a path in an arbitrary graph G and there is an edge $v_n v_k$ with $1 \leq k < n - 1$ then $v_1, v_2, \dots, v_k, v_n, v_{n-1}, \dots, v_{k+1}$ is also a path of the same length.

Path rotations are at the root of many heuristics for finding Hamilton paths or cycles and the technique is often called the *extension-rotation* approach. The heuristic that we use (Chapter 3) builds on this idea of extension-rotation. We add several improvements to give a very effective heuristic for the classes of graphs that we study.

Let $G_{n,M}$ be a random graph with n vertices and M edges. If $G_{n,M}$ is Hamiltonian then $\delta(G_{n,m}) \geq 2$, so it follows immediately that $Pr(G_{n,M} \text{ is Hamiltonian}) \leq Pr(\delta(G_{n,M}) \geq 2)$.

Komlós and Szemerédi [68] further refined these results by proving the following:

Theorem 13 *Let $G_{n,M}$ be a random graph n vertices and M edges, If the number of edges*

$$M = M(n) = \frac{n}{2}(\log n + \log \log n + c)$$

then

$$Pr(G_{n,M} \text{ is Hamiltonian}) \rightarrow e^{-e^{-c}}.$$

Koršunov [70] gave a new proof of Theorem 13.

The value $e^{-e^{-c}}$ is the limit of the probability that the $\delta(G_{n,M}) \geq 2$, so the existence of vertices of degree less than 2 is the main obstruction to $G_{n,M}$ being Hamiltonian. This is a surprising result when we consider the edge density required for proofs of Hamiltonicity in non-random graphs.

If we consider a random graph process as a process in which a graph starts as an empty graph with vertex set $V = \{v_1, v_2, \dots, v_n\}$ and grows by the random addition of edges, then Erdős and Spencer [40] conjectured that, in fact, almost every graph process is such that the very first edge which increases the minimum degree to 2 also makes the graph Hamiltonian. This was also restated by Komlós and Szemerédi [68].

Thus, for large n , if a random graph contains no vertices of degree 0 or degree 1, then it is almost surely Hamiltonian. Proofs were published by Ajtai, Komlós and Szemerédi [1] and independently by Bollobás [13] This result is even more surprising than that of Theorem 13.

2.2 Finding Hamilton Paths and Cycles

2.2.1 Complexity

A standard backtrack search (see for example [87]) attempts to construct a Hamilton path or cycle by starting at a designated starting vertex, s , and extending the path to a new vertex as long as possible. Whenever it is impossible to further extend the path from a vertex x , the search “backs up” to the predecessor, y , of x on the path and the path is extended (if possible) from y to one of its other neighbors. Such an algorithm grows exponentially and is unsuitable for anything but relatively small

```

procedure SELECT( $u$ )
  begin if  $uv \notin E(G)$  for every  $v \in V$  then return NIL
        else select an edge  $uv \in E(G)$  at random with equal probability
           delete  $uv$  from  $E(G)$  and return the value  $v$ 
  end

```

Figure 2.2: The SELECT procedure of Angluin and Valiant

problems. Not surprisingly, the Hamilton cycle problem is known to be NP-complete and a proof may be found in many standard textbooks on algorithms. See, for example, Cormen, Leiserson and Rivest [25] pp. 953-959.

2.2.2 Algorithms and Heuristics

In Section 2.1.3 we discussed the idea of path rotations used by Pósa [82] and others. If v_1, v_2, \dots, v_n is a path in an arbitrary graph G and there is an edge $v_n v_k$ with $1 \leq k < n - 1$ then $v_1, v_2, \dots, v_k, v_n, v_{n-1}, \dots, v_{k+1}$ is also a path of the same length. Using this basic idea, a Hamilton path (or cycle) search heuristic constructs a path P by selecting a starting vertex s and extending the path until no longer possible. When the path cannot be extended from a vertex u , the algorithm selects a neighbor v of u , performs a path rotation and then attempts to extend this new path. The algorithm terminates when a path (or cycle) has been found or when some terminating condition, such as a specified number of rotations, has been reached.

Angluin and Valiant [2] devised a fast probabilistic algorithm based on using rotations and a random procedure for selecting edges adjacent to a given vertex u of a graph G which then deleted the edge from G .

Angluin and Valiant's algorithm UHC for finding a Hamilton cycle in an undirected graph starts from a vertex s and uses the SELECT procedure, shown in Figure 2.2, to select adjacent vertices either extending, if the new vertex is not on the path, or reversing, if it is, until a Hamilton path is created whose endpoint is adjacent to s . They show the following bound on running time for a graph G_N whose N edges are present with equal probability.

Theorem 14 *For all α there exist values M and c , such that for all $N \geq cn \log n$ the probability that UHC finds a Hamilton cycle before SELECT has been called $Mn \log n$ times is $1 - O(n^{-\alpha})$.*

It is interesting to note that this work of Angluin and Valiant has been cited in over 150 other works.

Shamir [96] gave another extension-rotation algorithm to answer constructively the Erdős-Rényi question on the number of random edges required to make a graph almost surely Hamiltonian. Suppose a graph, G , is an example of a graph, $G_{n,p}$, where $p = p(n) = n^{-1}(\log n + c \log \log n)$ and $c > 3$. Using an extension-rotation method from earlier unpublished work he first constructs a partial path PP on a set $V(G) - \Phi$ of vertices of G where $|\Phi| \leq n(\log n)^{-c}$. A tree structure is then used to determine the possible endpoints after a series of rotations. Shamir bounds the height of the tree to be $\log n$ but shows that it almost surely contains every vertex of the path as a possible new endpoint. With this result the vertices of $\Phi = \{x_1, x_2, \dots, x_{|\Phi|}\}$ can almost surely be added in order with a suitable set of rotations between each new addition. We use a similar tree structure for computing sequences of possible rotations.

Bollobás, Fenner, and Frieze [9] also devised a rotation-extension algorithm, HAM, for finding Hamilton paths and cycles in random graphs that is theoretically best possible in the sense that its probability of finding a circuit matches that of the Komlós-Szemerédi result of Theorem 13. Suppose $G_{n,m}$ is a member of the probability space of random graphs with n vertices and m edges. Suppose further that

$$m = n \log n/2 + n \log \log n/2 + c_n n$$

for some sequence c_n .

Theorem 15

$$\lim_{n \rightarrow \infty} Pr(\text{HAM finds a Hamilton cycle in } G_{n,m}) = \begin{cases} 0 & \text{if } c_n \rightarrow -\infty, \\ e^{-e^{-2c}} & \text{if } c_n \rightarrow c, \\ 1 & \text{if } c_n \rightarrow +\infty. \end{cases}$$

The algorithm, HAM, adds the idea of *cycle extension* to the standard extension-rotation technique of Pósa by checking for the existence of an edge from the last vertex v_n to the first vertex v_1 of the path whenever the current path, P , cannot be extended. Since the graph is connected, some vertex v_i on P is adjacent to a vertex u which is not on the path. The path P can be transformed into a new path P' by adding the edge $v_0 v_n$ and removing an edge incident with v_i . The path P' is of the same length as P but can be extended by the addition of the edge $v_i u$. A counter is used to bound the maximum number of rotations of the path and guarantee polynomial running time.

The authors show that HAM is an $O(n^{4+\epsilon})$ algorithm. They consider the case of graphs on n

vertices, where each of the 2^N , $N = \binom{n}{2}$, possible graphs is equally likely to be chosen. Under this model the probability of failure is so small that the application of a dynamic programming algorithm [56] when HAM fails gives the following theorem as a corollary to Theorem 15.

Theorem 16 *There is an algorithm for solving the Hamilton cycle problem with polynomial expected running time.*

Thomason [105] also used the idea of combining fast algorithms with exponential algorithms to achieve an overall linear expected time algorithm for finding a Hamilton path between two specified vertices, s and t , in a given undirected graph if such a path exists, or saying so if it does not. For a random graph of order n with probability p of occurrence of an edge, this algorithm requires an expected time cn/p and storage cn , where c is a constant. This algorithm works for $p \geq \frac{12}{\sqrt[3]{n}}$, an improvement over another algorithm proposed by Gurevich and Shelah [54]. Thomason uses a pair of fast greedy algorithms, A_1 and A_2 . These work for all but a very small fraction (2^{-n}) of all graphs. This small fraction of graphs is then processed by any exponential-time algorithm for finding Hamilton paths.

Algorithm A_1 starts with one of the two specified vertices, s and t , and extends the partial path by inserting unvisited vertices to eventually obtain a Hamilton path. Algorithm A_2 starts with a set V_0 of vertices of degree at most $pn/4$ in graph G . It finds a ≤ 2 -matching from V_0 to $G - V_0 - \{s, t\}$ of maximum size, and then extends the matching in a greedy fashion to obtain a Hamilton path.

Frieze [48] used the ideas of Theorem 15 together with Depth-First-Search probing of the partial paths generated by rotations and created an $O(n^3 \log n)$ algorithm, for the special case of multigraphs $D_{n,m}$, of order n with each vertex of degree, m , created as for Model C but without regard to orientation and for random regular graphs. He showed the following:

Theorem 17 *There is an $O(n^3 \log n)$ time algorithm HAM1 which satisfies*

$$\lim_{n \rightarrow \infty} Pr(\text{HAM1 finds a Hamilton cycle in } D(n, 10)) = 1.$$

He used another variant of the theorem to construct long cycles in sparse random graphs.

If we let $R(n, r)$ denote an r -regular random graph of order n , then Fenner and Frieze [46] and

Bollobás [12] independently showed that there is a constant r_0 such that for any constant $r \geq r_0$

$$\lim_{n \rightarrow \infty} \text{PR}(R(n, r) \text{ is Hamiltonian}) = 1$$

The smaller value given in [46] was $r_0 = 796$. Frieze [48] reduced this.

Theorem 18 *There is an $O(n^3 \log n)$ time algorithm HAM2 which satisfies*

$$\lim_{n \rightarrow \infty} \text{Pr}(\text{HAM2 finds a Hamilton cycle in } R(n, r)) = 1.$$

for any constant $r \geq 85$.

Bollobás [11] had earlier conjectured that this was true for $r \geq 3$. Robinson and Wormald [89] showed that almost all cubic graphs are Hamiltonian, by subdividing the set of cubic graphs according to the number of short odd cycles. Shortly after they closed the gap and showed that almost all regular graphs are Hamiltonian [90]. The proof of the latter result did not use the same method as for cubic graphs because the subgraph induced by two cycles must have vertices of degree 2 or 3 in a cubic graph, but may have vertices of degree 4 when the graph is regular of higher degree. In the later work an analysis of the distribution of the number M of perfect matchings in a random regular graph is used. They show, in fact, that the edges of a graph G of even order can be decomposed into a set of perfect matchings and a Hamilton cycle with high probability. Graphs with an odd number of vertices are handled by a similar argument with an extra twist.

The results are for fixed r . If $r = r(n) \rightarrow \infty$ sufficiently slowly they still hold. The authors conjecture the following:

Conjecture 3 *Suppose $R(n, r(n))$ is a random regular graph with $3 \leq r(n) \leq n$ and $nr(n)$ is even. Then $R(n, r(n))$ is almost surely Hamiltonian as $n \rightarrow \infty$.*

Robinson and Wormald also make two additional conjectures concerning the number of pairwise edge-disjoint Hamilton cycles in random regular graphs.

Conjecture 4 *Let $r \geq 4$. The probability that a random r -regular graph on $2n$ vertices has $\lfloor r/2 \rfloor$ pairwise edge-disjoint Hamilton cycles tends to 1 as $n \rightarrow \infty$.*

Conjecture 5 *Let $r \geq 4$. The probability that a random r -regular bipartite graph on $2n$ vertices has $\lfloor r/2 \rfloor$ pairwise edge-disjoint Hamilton cycles tends to 1 as $n \rightarrow \infty$.*

In a recent paper Frieze, et al. [49] deviated from the extension-rotation approach and used a rapidly mixing Markov chain approach to provide a constructive solution to the earlier results of Robinson and Wormald [89, 90]. Let $\mathcal{G}(r, n)$ denote the set of r -regular (simple) graphs of order n .

Theorem 19 *Let $r \geq 3$ be fixed and let G be chosen uniformly at random from $\mathcal{G}(r, n)$. There is a polynomial time algorithm *FIND* which constructs a Hamilton cycle in G with high probability.*

Suppose a graph G , of order n , is obtained by adding a random perfect matching M to a random Hamilton cycle H . Broder, Frieze and Shamir [15] had left open the problem of finding a Hamilton cycle without knowing H . A motivation for this problem is the design of authentication protocols. Theorem 19 solves the problem and suggests that such a problem may not be useful for authentication protocols.

2.2.3 Using Random Graph Heuristics on Non-Random Graphs

Shields and Savage [98] used an extension-rotation approach together with a Breadth-First-Search tree search for optimal sequences of rotations similar to that used by Shamir [96] to investigate the middle two levels problem for the graph M_{2k+1} . They were able to improve the best previously known result that M_{2k+1} is Hamiltonian for $k \leq 11$ due to Moews and Reid [86]. This was improved to show M_{2k+1} Hamiltonian for $k \leq 15$. Shields and Savage reduced the problem of finding a Hamilton cycle in M_{2k+1} to that of finding a Hamilton path between two particular vertices in a smaller graph. For $k = 15$, M_{2k+1} has 601,080,390 vertices. The reduced graph still has 9,694,845 vertices and the algorithm was able to find the desired Hamilton path in the reduced graph in three weeks on a 400MHz Pentium II system. Further details are provided in Chapter 4.

A variant of the same algorithm was used by Shields and Savage [99] to find cycles in all Kneser graphs, $K(n, k)$, and bipartite Kneser graphs $H(n, k)$, with $n \leq 27$. See Chapter 5.

The heuristic was used to find basis cases for an inductive proof by Savage, Shields and West [93] that the cover graph, $M(n)$, has a Hamilton path if and only if $\binom{n+1}{2}$ is odd and $n \neq 5$. See Chapter 7.

Extensions of this work to cubic Cayley graphs are described in this thesis. See Chapter 6.

Chapter 3

The Hamilton Path Heuristic

3.1 Backtrack and Extension-Rotation Techniques

Given a graph G and vertices $s, t \in V(G)$ we would like to find, if possible, a Hamilton path starting at s and ending at t .

A standard backtrack search (see for example [88]) attempts to construct such a path by starting at s and extending the path to a new vertex as long as possible (avoiding t until all other vertices have been included). Whenever it is impossible to further extend the path from a vertex x , the search “backs up” to the predecessor, y , of x on the path and the path is extended (if possible) from y to one of its other neighbors.

A backtrack search is guaranteed to find a Hamilton path from s to t if such a path exists. The drawback is that it is an exhaustive search, exponential time in the worst case, and it quickly becomes too slow to be effective for anything but relatively small problems. Many pruning techniques have been devised to reduce the search space, but even these improvements do not make any kind of backtrack search practical for large graphs.

Pósa [82] observed that reaching a dead end in backtrack search could be used as an opportunity to modify the current path by a *rotation* and thus possibly to continue. Figure 3.1 illustrates a path P from a starting vertex s to a vertex u whose only neighbors in G are s , x and v , which are already on P . In this case, backtrack search would back up to w , then v and eventually back to z , at which time t would be added to the path.

Instead, we can transform the current path, P , into a path, P' , of equal length by a *rotation*

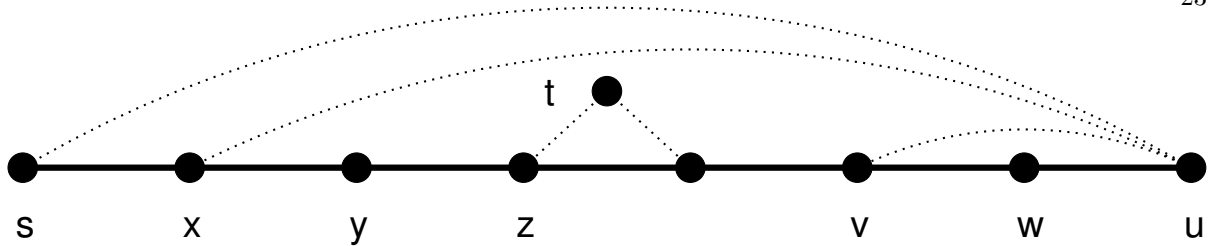


Figure 3.1: Backtrack search from s has reached u and cannot continue.

at x , which inserts the edge ux into P and removes the existing edge xy from P as illustrated in Figure 3.2. Vertex y becomes the new endpoint of P' . More generally, if $P = (u_1, u_2, \dots, u_k)$ is a path in G and there is an edge $u_k u_j$ for some $0 < j < k$, then the *rotation of p at u_j* , namely $P' = (u_1, u_2, \dots, u_j, u_k, u_{k-1}, \dots, u_{j+1})$, is also a path in G . If $j \neq k - 1$ then P' and P share a common starting point but have different endpoints.

Define $\text{PosaSearch}(G, s, t)$ to be the Hamilton cycle search heuristic which constructs a path P by starting at s and extending the path until no longer possible (avoiding t until the end). When the path cannot be extended from a vertex u , the procedure selects a neighbor v of u and performs a path rotation at v by calling $\text{ReversePath}(v, P)$. PosaSearch does not backtrack. See Figure 3.3 for the pseudo-code which uses a style similar to the BFS algorithm in Cormen, Leiserson and Rivest [26].

Of course, there are drawbacks to the PosaSearch . It may not succeed in finding a path even if one exists. Furthermore, it may run indefinitely, even if we eliminate the obvious case where the rotation candidate chosen is the immediate predecessor of the endpoint on the path.

If we consider the path, P , shown in Figure 3.1, we see that the PosaSearch heuristic can only transform P into one of the paths shown in Figure 3.4. None of these transformations will ever allow the vertex, t , to be added to the path.

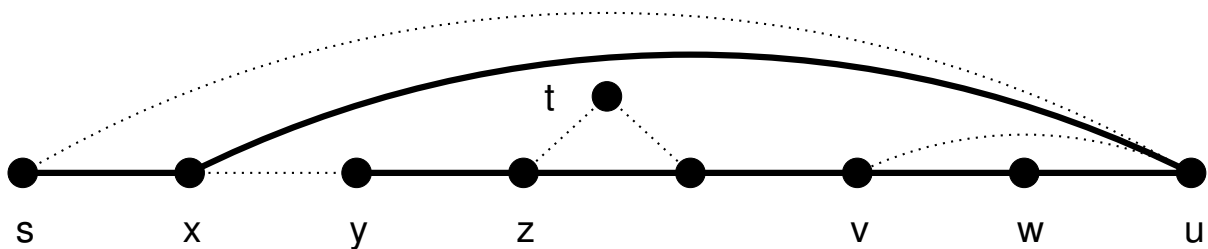


Figure 3.2: A path rotation at x : add edge ux and remove edge xy .


```

PosaSearch(G,s,t)
  P ← (s)
  while (|P| < |V(G)| - 1)
    do u ← end(P)
    ▷ Extend P from u if possible
    extension_candidates ← neighbors of u, other than t, not on P
    if extension_candidates not empty
      then select x from extension_candidates
      P ← Px    ▷ Add x to the end of P
    else ▷ Do a rotation, if possible
      rotation_candidates ← neighbors of u on P
      if rotation_candidates not empty
        then select v from rotation_candidates
        P ← ReversePath(v, P)
      else return FAIL
  ▷ Now P is missing only t so perform rotations until t is a neighbor
  ▷ of end(P)
  while t is not a neighbor of end(P) do
    if end(P) has no neighbor
      then return FAIL
    else ▷ all neighbors are on P
      select a neighbor v of end(P)
      P ← ReversePath(v, P)
  P ← Pt    ▷ Add t to the end of P
  return P

```

Figure 3.3: The Pósa path rotation heuristic

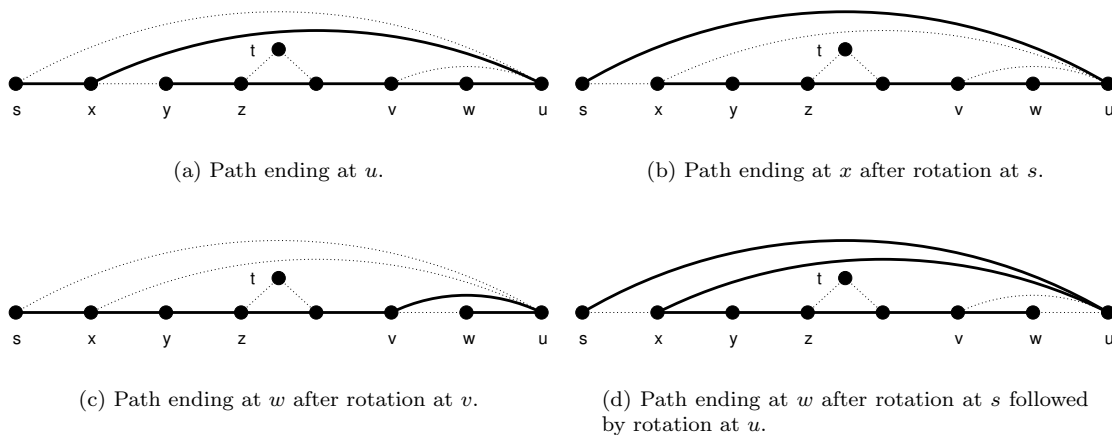


Figure 3.4: Limitations of the PosaSearch heuristic

Nevertheless, the path rotation approach has proven useful in studying random graphs. Angluin and Valiant [2] devised a fast variant that always terminated because edges were removed from the graph once they had been included in a path. This was shown to almost surely find a Hamilton cycle in random graphs with n vertices and $cn \log n$ edges. The middle two levels graph, which we will study further in Chapter 4, is certainly not random, but it has a relationship of vertices to edges of approximately this form. Although we tried the heuristic on it, we did not succeed in finding a Hamilton cycle using it.

3.2 Improving the Heuristic

The heuristic we propose uses refinements of the path rotation technique of Pósa [82] which is the basis of several Hamilton path heuristics and algorithms for random graphs.

To overcome the potential non-termination of PosaSearch and to increase the chances that a path rotation would lead to an extension of the path, we improved the way in which candidates were chosen for the rotation operation whenever all neighbors of the endpoint are already on the path. Figure 3.4 shows four paths obtainable by one or more rotations from the path shown in Figure 3.1. In this example, there are only four possible endpoints, u , v , w , and x . The path cannot be extended from any of these. As we shall describe, our heuristic evaluates possible endpoints without performing the rotations. If a sequence of one or more rotations guaranteed to result in a path which can be further extended is found, then the sequence of rotations is performed and the path is extended. If no such sequence exists, the heuristic terminates. The worst-case running time is polynomial in the number of vertices.

Rotations transform a path and alter the order of vertices on it. We therefore define several functions to handle arbitrary paths. Let $P = (u_1, u_2, \dots, u_n)$ be an arbitrary path of length $n - 1$ in a graph G . We denote the last vertex u_n as $end(P)$. We denote the i th vertex u_i by $P(i)$ and the position of u_i as $pos(P, u_i) = i$. We denote the number of vertices on P as $|P|$ and thus $end(P) = P(|P|)$. The successor of a vertex u on P is defined for all vertices other than $end(P)$ as $succ(P, u) = P(1 + pos(P, u))$. We may then rewrite a rotation of P at u_j , denoted $r(P, j)$, as $(P(1), P(2), \dots, P(j), P(n), \dots, P(j+1))$. Evidently $r(P, j)$ is a path if and only if $end(P)$ is adjacent to $P(j)$. Let $P' = r(P, j)$ be a rotation of P at $P(j)$ and let u be a vertex on P . Let $p = pos(P, u)$ be the position of u on P . Then the position p' of u on P' is

$$p' = pos(P', u) = \begin{cases} p & \text{if } p \leq j \\ |P| - p + j + 1 & \text{otherwise} \end{cases} \quad (3.1)$$

Rewriting Equation 3.1 we may determine the vertex of P that is now in position p' on P' as

$$P'(p') = \begin{cases} P(p') & \text{if } p' \leq j \\ P(|P| - p' + j + 1) & \text{otherwise} \end{cases} \quad (3.2)$$

The successor of $u = P(p')$ on P' is defined if and only if $p' < |P|$ and is obtained by substituting $p' + 1$ for p' in Equation 3.2.

If P is a path with endpoint $u = \text{end}(P)$ and v is a vertex on P adjacent to u we may reverse P at v to arrive at a new path $P' = r(P, \text{pos}(P, v))$. We use Equation 3.1 to find the position of v on P' and then use Equation 3.2 to determine the successor w on P' of v . If the edge uv is on P then $P = P'$, otherwise we say that P' is *derivable* from P in one rotation and we say that the new endpoint w of P' is *reachable in one rotation of P* . Inductively, we say that a path Q is derivable from P in $n > 1$ rotations if it is derivable in one rotation from a path R which is derivable from P in $n - 1$ rotations. We define a vertex of P as reachable in $n > 1$ rotations if it is reachable in one rotation from a path R which is derivable from P in $n - 1$ rotations.

Our approach is to extend the path as is done in `PosaSearch` until all neighbors of the endpoint are already on the path and we cannot extend further. Whenever we reach such a point we start building a tree of the endpoints that are reachable from the current path P . If u is in the tree and if y is the parent of u , then u and y are endpoints of paths P_u and P_y reachable from P and, furthermore, $P_u = r(P_y, j)$ for some index j . We save this value j as $J[u]$ and save the parent of u as $\pi[u]$. The root of the tree is $\text{end}(P)$. $\pi[\text{end}(P)]$ is set to `NIL` and $J[\text{end}(P)]$ is set to 0. The sequence of values $J[\pi[u]], J[\pi[\pi[u]]], \dots, J[\text{end}(P)]$ along the path in the tree from u to the root define the sequence of rotations required to derive P_u from P .

The search proceeds using a standard breadth-first search (BFS). Initially, all vertices on P are marked unvisited. The endpoint of P (the root of the tree) is marked visited and placed on a first-in first-out queue, Q . As each reachable endpoint is discovered, it is placed on the queue, Q . While Q is not empty we remove the front vertex, u , from Q . Note that because u is on Q , $u = \text{end}(R)$ for some path R derivable from P . When u is dequeued each neighbor v of u in G is examined. If v is on P we use the stored π and J values to determine the endpoint w that would result from a rotation of R at v . If w has not yet been visited then w is reachable from P and we add it to the tree, set its π and J values, mark it visited, and insert it into Q . If v is not on P our search has succeeded. We terminate the search process and perform the sequence of rotations required to derive R with endpoint u from P .

Again, the stored π and J values will be sufficient to find R . Since u is adjacent to v in G and v is not on P we are now guaranteed to be able to extend the path R by at least one edge. If the BFS search fails to find a vertex having a neighbor not on P then the heuristic terminates without having found a Hamilton path.

The pseudo-code, FindPosition, implementing the algorithm of Equation 3.1 to find the position of the vertex u on a path R derivable from path P such that u is currently in position p on path P and $u = \text{end}(R)$ is shown in Figure 3.5.

The pseudo-code, FindVertex, implementing the algorithm of Equation 3.2 to find the vertex v that will be in position r on the path R derivable from P such that $u = \text{end}(R)$ is shown in Figure 3.6.

The function DoRotations shown in Figure 3.7 performs the rotations once a suitable new endpoint has been found.

The pseudo-code for BFSExtend is shown in Figure 3.8.

We do not assume any particular representation of the graph, only that a function $Adj(u)$ is available to return a list of vertices adjacent to u . We maintain several data structures for each vertex in the tree. For each vertex u a Boolean variable $visited[u]$ indicates whether u has been visited or not. The ancestor of u in the tree is stored in $\pi[u]$ and the position of the vertex about which a rotation is done to make u the new endpoint of a path currently ending in $\pi[u]$ is stored in $J[u]$. A first-in, first-out queue Q , initially empty, is used to manage the list of vertices that have been discovered but whose neighbors have not yet been examined. We also assume that the current position of a vertex u on the path P is available as $Pos[u]$.

The basic heuristic will attempt to find a Hamilton path. If a path is found, and the first and last vertices are adjacent then we have also found a cycle. Otherwise, we use our BFS logic and attempt to find a series of rotations that will make the first and last vertices of the path adjacent.

Our heuristic can be made non-deterministic by allowing choices from the adjacency list to be made randomly. This allows additional attempts to find a Hamilton path to be made with a new random number seed if the heuristic fails on a particular attempt. In practice, successive calls to the pseudo random number generators used in computer applications always generate the same sequence of random numbers, so our code permits a user to specify the seed for the random number generation process.

```

FindPosition ( $P, u, p$ )
  if  $J[u] \geq 0$ 
    then  $p \leftarrow \text{FindPosition}(P, \pi[u], p)$ 
    if  $J[u] < p$ 
      then  $p \leftarrow |P| - p + J[u] + 1$ 
  return  $p$ 

```

Figure 3.5: Finding the new position of a vertex

```

FindVertex ( $P, u, r$ )
  while  $J[u] \geq 0$ 
    if  $J[u] < r$ 
      then  $r \leftarrow (|P| - r) + J[u] + 1$ 
     $u \leftarrow \pi[u]$ 
  return  $P[r]$ 

```

Figure 3.6: Finding the new vertex in a position

```

DoRotations( $P, u$ )
  if  $J[u] \geq 0$ 
    then  $\text{DoRotations}(P, \pi[u])$ 
     $l \leftarrow J[u] + 1$ 
     $h = |P|$ 
    while  $l < h$ 
       $t = P[h]$ 
       $\text{Pos}[P[h]] \leftarrow l$ 
       $\text{Pos}[P[l]] \leftarrow h$ 
       $P[h] \leftarrow P[l]$ 
       $P[l] \leftarrow t$ 
       $h \leftarrow h - 1$ 
       $l \leftarrow l + 1$ 

```

Figure 3.7: Performing the rotations

```

BFSExtend( $P, s, t$ )
  for each vertex  $u$  on  $P$ 
    do  $visited[u] \leftarrow \text{false}$ 
   $z \leftarrow \text{end}(P)$ 
  Enqueue( $Q, z$ )
   $visited[z] \leftarrow \text{true}$ 
   $J[z] \leftarrow 0$ 
   $\pi[z] \leftarrow \text{NIL}$ 
  while  $Q$  is not empty
    do  $u = \text{Head}(Q)$ 
      Dequeue( $Q$ )
      for each vertex  $v$  in  $Adj(u)$ 
        do if  $v \neq t$ 
          then if  $v \in P$ 
            then  $p \leftarrow \text{FindPosition}(P, u, Pos[v])$ 
               $w \leftarrow \text{FindVertex}(P, u, p + 1)$ 
              if  $\neg visited[w]$  and  $w \neq t$ 
                then  $J[w] \leftarrow p$ 
                   $\pi[w] \leftarrow u$ 
                   $visited[w] \leftarrow \text{true}$ 
                  Enqueue( $Q, w$ )
              else  $P \leftarrow \text{DoRotations}(P, u)$ 
          return success
  return fail

```

Figure 3.8: The Hamilton path BFS heuristic

3.3 Running Time

If V and E are the number of vertices and edges respectively in G and H is the maximum height of the BFS tree then we may determine the worst-case running time as follows. The running time for each of FindPosition and FindVertex is $O(H)$. BFSExtend performs some initial housekeeping for each vertex on P and then examines each edge in G that is incident with a reachable vertex of P and computes a reachable endpoint for each such edge, so the time spent building the (partial) BFS tree is at worst $O(V) + O(E)O(H)$. The path rotations performed in BFSExtend are $O(H)O(V)$ at worst, so the running time of BFSExtend is at worst $O(E)O(H)$. The main program performs some initial housekeeping for each vertex in G and then examines each edge in G until a path is found or the heuristic terminates. Since a call to BFSExtend either results in termination of the heuristic or extension of the current path, the total running time for the heuristic is at worst $O(V)O(E)O(H)$. In an extreme case this is $O(V^4)$. However, if the number of edges were, say $O(V \log V)$ and the maximum height of the BFS tree was $O(\log V)$ the worst case would become $O(V^2 \log^2 V)$, a significant improvement.

For comparison, the running time of the deterministic algorithm of Bollobás, Fenner and Frieze [9] for finding paths almost surely in random graphs is $O(V^{4+\epsilon})$.

As noted earlier, our search heuristic does not assume any particular representation of a graph G , only that a function $Adj(u)$ is available to return a list of vertices adjacent to u . The search heuristic assumes vertices are numbered from 0 to $n = |G|$ without regard to their internal representation. The calculation of worst-case running time has the underlying assumption that the adjacency list can be calculated in time linear in the size of the list. When adjacency lists are calculated once and stored for later use, this assumption is easily satisfied. As we shall see in Section 3.4.1, this was not always practical for large graphs. We discuss an alternative strategy in that section. Additional information on our graph representations can be found in the chapter on each graph.

3.4 Practical Enhancements

In the preceding sections we have described our heuristic as it was initially used on the middle two levels problem (Chapter 4). As we worked on Kneser graphs (Chapter 5) and cubic Cayley graphs (Chapter 6) we made several improvements. Although these do not affect the theoretical running times described in Section 3.3, they do significantly affect the actual time taken by the heuristic. We describe these enhancements in the following sections.

3.4.1 Dynamic Adjacency Calculation

For the middle two levels problem and for Kneser and bipartite Kneser graphs, we were dealing with very large graphs with up to 2×10^7 vertices and up to 10^{11} edges. We found that storing adjacency information in memory for larger graphs resulted in significant memory swapping. Although we did not experiment with storing adjacency information on disk and accessing it as needed, we felt that the operating system memory swapping mechanism was likely to be at least as efficient as any disk implementation we could devise. Nevertheless, when swapping did occur, we observed a significant reduction in speed of the heuristic. To offset this effect, we experimented with dynamic generation of adjacency lists. Whenever an adjacency list for a vertex was required, we recalculated it. Only if it was the most recently calculated one did we reuse saved adjacency values. For graphs with the kind of regular structure of the middle two levels or Kneser graphs, this calculation could be done quite quickly. In Section 5.4.1 of Chapter 5 we describe a very fast algorithm for calculating adjacencies in Kneser graphs using a loop with only 10 register instructions. Not only were we able to handle much larger graphs than would have been possible using stored adjacency information, the recalculation time was more than offset by the lack of time spent swapping memory. Many factors, including CPU speed, available memory, operating system swap algorithm and the compiler optimization of our compiled code affect the actual performance. The results for the middle two levels illustrate the relative slowdown of larger problems on a system where swapping was still required, even with our dynamic adjacency calculation.

3.4.2 Reducing BFS Tree Height

We observed that, for our large graphs, the BFS trees tended to be broad but not very high. If we added a vertex, v , to the tree at the time the tree contained n nodes, there might be several times n nodes on the tree before we dequeued v and discovered that it was the vertex we were searching for. To alleviate the delay caused by this problem we implemented a *free adjacency* array containing a counter for each vertex of the graph to count of the number of neighbors of that vertex not already on the path. Thus, we were able to know at the time that a vertex, v , was enqueued that it would be a suitable new end point, rather than waiting for it to be dequeued before we could determine this. Our first implementation of this was for our work on Kneser graphs (Chapter 5). As a result, we saw significant speed increase compared with the original version of the heuristic.

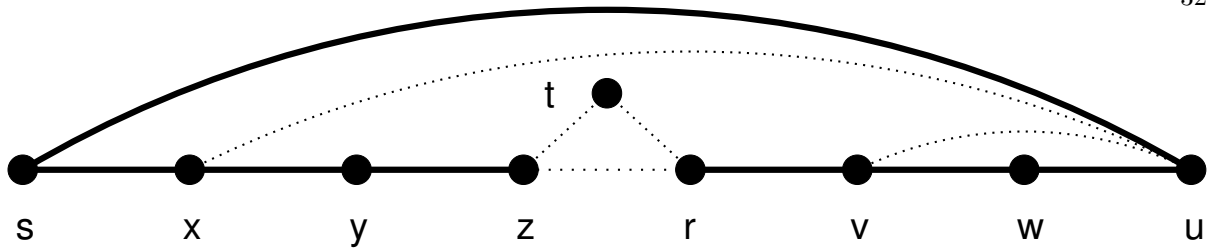


Figure 3.9: Using cycle extension to transform a su -path to a zr -path

3.4.3 Cycle Extension

When we started work on cubic Cayley graphs (Chapter 6) we were frequently unable to find a cycle, even though we were dealing with much smaller graphs than we had seen in the middle two levels or Kneser cases.

We enhanced the cycle extension technique of Bollobás, Fenner, and Frieze [9] (see Chapter 2) when a path could not be extended. Recall that their algorithm checks whether a partial path has an edge between the first and last vertices. If so, the resulting cycle can be broken open at any point that will allow the path to be extended. We use our BFS logic on the partial path in order to determine if a sequence of rotations can transform the partial path so that the two ends are adjacent, thus transforming it into a cycle. If the path can be converted to a cycle we find the last vertex on the path that was adjacent to a vertex not on the path and break the cycle at that point to form a new path that can be extended. Figure 3.9 shows the effect of cycle extension applied to the path of Figure 3.1.

We also updated our heuristic to take full advantage of another technique which was also used by Bollobás, Fenner, and Frieze. This permits extension of a partial path from either end. Our original heuristic attempted to extend the path until it could no longer extend it. If the search was for a cycle, then the heuristic reversed the whole path so that the current end point became the start and the original start became the new end point. Originally, we performed this reversal only once, extending from one end until no further extensions could be made and then attempting once only to extend from the other end. We enhanced our logic to repeat this path reversal process until a Hamilton cycle was found or no further extensions could be made.

These two changes significantly improved our success rate on cubic Cayley graphs.

3.4.4 Random Backup

As we saw in Section 3.2 our heuristic uses a random number generator when choosing which edge to use in extending the path. Starting the heuristic with a different random seed results in a different sequence of edges being added, with the possibility that a Hamilton cycle (or Hamilton path) may be found where it was not found in a previous attempt. This approach requires a completely new path to be generated. Rather than throwing away the whole path, we experimented with removing a random number of edges from a path that could not be extended and then attempting to extend the shorter path to a cycle. We found this approach beneficial for cubic Cayley graphs.

We chose to limit the number of vertices removed to a maximum of 400 and to attempt this random edge removal up to 400 times before returning with no cycle found. Our choice of 400 for each of these values was arbitrary. Exploring strategies for determining optimal values is a suggestion for further study.

3.4.5 Verification

For the first implementation of our heuristic, we separated most of the graph related computation from the path search logic. Faced with an output file of approximately 800MB of data for a 10,000,000 vertex graph, we realized that we needed some means of verifying that the output truly represented a Hamilton path. For the middle two levels graph we wrote a separate program to verify the output.

For later work, we formalized our graph structures using two abstract base classes in C++. One class was for graphs which did not include an adjacency matrix and the other was for graphs that did. These base classes include functions such as the ability to check whether two vertices are adjacent. We also modified the path heuristic to use these functions and verify that a vertex being added to a path was, indeed, adjacent to the current end point of a path. Routines to perform internal consistency checks on various structures used in the path heuristic were also found to be invaluable, particularly when large parts of the path were being manipulated.

3.4.6 Checkpoint and Restart

Our code has shown itself to be extremely robust, but it was not able to defend against events such as power outages. With runs that take days or weeks, we found it advantageous to implement a simple checkpoint mechanism. Periodically, we dump the current state of the path, along with some

```

int HAMSEARCH::BFSMakeCycle(GRAPHBASE& g) {
    int num_r = 0;
    if (path[0] != adj_list_vertex) {
        adj_list_vertex = path[0];
        adj_list_degree = g.adj_func(adj_list_vertex, adj_list);
    }
    for (int ix = 0; ix < adj_list_degree; ix++) {
        FreeAdj[adj_list[ix]]++;
    }
    num_r = BFSExtend( g, NIL_VERTEX);
    if (path[0] != adj_list_vertex) {
        adj_list_vertex = path[0];
        adj_list_degree = g.adj_func(adj_list_vertex, adj_list);
    }
    for (int ix = 0; ix < adj_list_degree; ix++) {
        FreeAdj[adj_list[ix]]--;
    }
    return num_r;
}

```

Figure 3.10: Create a cycle from a path or partial path

key variables that allow restart from the point of interruption, to a disk file. If such a file exists when the program is started, the program reloads the path from the file and continues from the point of interruption. We chose to checkpoint approximately every 30 minutes.

3.5 The Current Heuristic

The logic used by our heuristic for creating a path from a cycle assumes that a path cannot be extended using the `BFSExtend` function of Figure 3.8. We take advantage of the free adjacency array described in Section 3.4.2 by incrementing the counters for the initial vertex of the path to make it appear to the BFS logic that the initial vertex could be added to the path. The BFS logic will then try to find a sequence of rotations that make a neighbor of the initial vertex the new endpoint of the path, thus creating a cycle. The code for attempting to create a cycle for a path that cannot be extended is shown in Figure 3.10.

The code used for finding a Hamilton cycle starts by attempting to find a Hamilton path from a designated starting vertex. When the Hamilton path function terminates because the path can no longer be extended, we add the cycle extension and random backup techniques described in Sections 3.4.3 and 3.4.4 and attempt to further extend the path with the eventual goal of finding a Hamilton cycle. The top level code, combining all these techniques to find a Hamilton cycle, is shown in Figure 3.11.

```

int HAMSEARCH::HamCycle(int s, GRAPHBASE& g) {
    pathlen = HamPath( s, NIL-VERTEX, g);
    for (int ix = 0; (ix < 400) && pathlen <= sizeG ; ix++) {
        int lastSize = 0;
        while ((pathlen < sizeG) && (pathlen > lastSize)) {
            lastSize = pathlen;
            ReversePath(); // Try reversing and extending existing path
            pathlen = ExtendPath(g, probname, NIL-VERTEX);
            if(pathlen == lastSize) {
                int num_rots = BFSMakeCycle(g);
                if(num_rots >= 0) {
                    int newEnd = 0;
                    bool foundnbr = false;
                    for (int checkix = pathlen-1;
                        (!foundnbr) && (checkix > 0); checkix--) {
                        foundnbr = (FreeAdj[path[checkix]] > 0);
                        newEnd = checkix;
                    }
                    RotateCycle(pathlen - newEnd - 1, g);
                    pathlen = ExtendPath(g, probname, NIL-VERTEX);
                } else { // Try dropping last vertex and reversing path
                    RemoveLastFromPath(g);
                    ReversePath();
                    pathlen = ExtendPath(g, probname, NIL-VERTEX);
                }
            }
        }
    }
    if (pathlen == sizeG) { // Try to make a path into a cycle.
        if (!g.isEdge(path[0], path[pathlen-1])) {
            int num_r = BFSMakeCycle(g);
            if (num_r <= 0) { // No luck to try the other end
                ReversePath();
                num_r = BFSMakeCycle(g);
            }
        }
        if (g.isEdge(path[0], path[pathlen-1])) {
            path[pathlen++] = path[0];
        }
    }
    if (pathlen <= sizeG) {
        int backup = rand()%400;
        if (backup > pathlen) {
            backup = pathlen%2;
        }
        for (int ix=0; ix < backup; ix++) {
            RemoveLastFromPath(g);
        }
    }
}
return pathlen;
}

```

Figure 3.11: The Hamilton cycle heuristic

Chapter 4

The Middle Two Levels Problem

4.1 Introduction to the Problem

Let \mathcal{B}_n be the n -atom Boolean lattice, that is, the partially ordered set of subsets of $[n] = \{1, 2, \dots, n\}$, ordered by inclusion. The Hasse diagram, $H(\mathcal{B}_n)$, is the covering graph of \mathcal{B}_n with two subsets adjacent when they differ in only one element. We define the i th level $\mathcal{B}_n(i)$ of \mathcal{B}_n as the set of i -element subsets of $[n]$, so there are $\binom{n}{i}$ elements in $\mathcal{B}_n(i)$.

The notorious *middle two levels problem* is to determine if there is a Hamilton path or cycle in the subgraph M_{2k+1} of $H(\mathcal{B}_{2k+1})$ induced by the middle two levels $\mathcal{B}_{2k+1}(k)$ and $\mathcal{B}_{2k+1}(k+1)$. The problem has been variously attributed to Dejter, Erdős, Trotter, Havel, and Kelley (see [31], [35], [60], [66], [92]).

The Boolean lattice is isomorphic to the set of n -bit binary numbers with two numbers being adjacent in the Hasse diagram if and only if they differ in exactly one bit position. For $n = 2k + 1$, the middle two levels are the $2k + 1$ -bit binary numbers whose representations are permutations of $0^k 1^{k+1}$ or $0^{k+1} 1^k$. We illustrate the Hasse diagram of \mathcal{B}_5 in Figure 4.1, with the heavier lines defining a Hamilton cycle through the middle two levels graph, M_5 .

Since M_{2k+1} is connected and vertex transitive, the nonexistence of a Hamilton path in M_{2k+1} for some k would provide a counterexample to the Lovász conjecture [75] that every connected, undirected, vertex transitive graph has a Hamilton path. However, both constructive and existential approaches have so far been unsuccessful in establishing the existence of a Hamilton path or cycle in M_{2k+1} in the general case. A result of Babai [3] gives a lower bound on the length of the longest cycle in a vertex

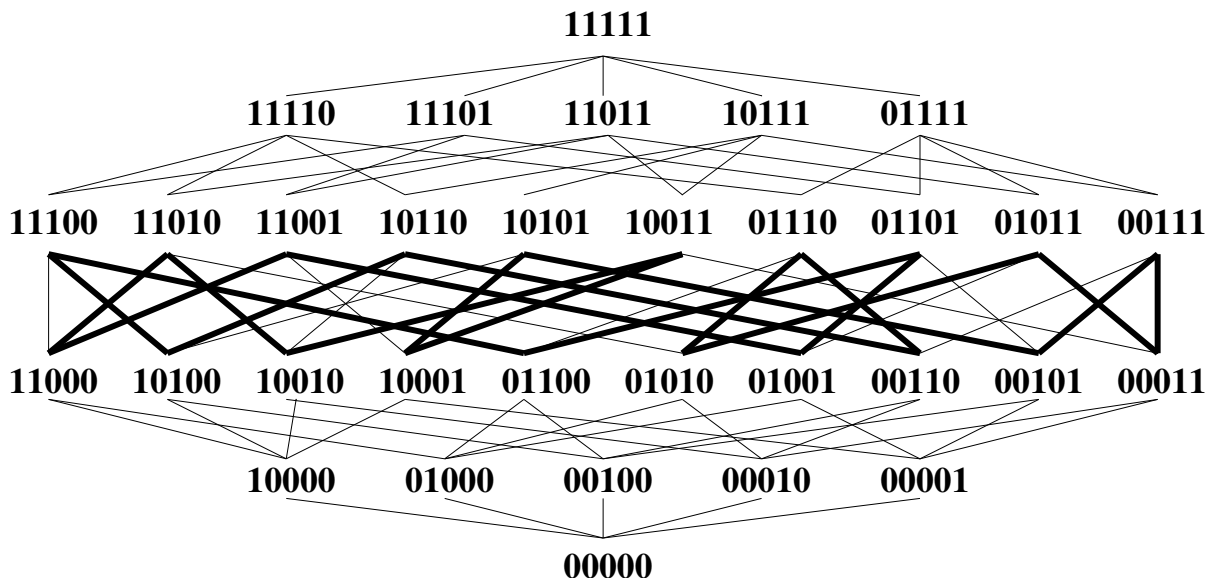


Figure 4.1: The Hasse diagram of \mathcal{B}_5 showing a Hamilton cycle through the middle two levels.

transitive graph, G , of $(3|V(G)|)^{1/2}$. The number of vertices, $V(M_{2k+1})$, in the middle two levels graph M_{2k+1} is $2\binom{2k+1}{k}$. This lower bound was improved to $0.25V_k$ by Felsner and Trotter [45]. The best lower bounds are currently given by the following theorems of Savage and Winkler [94] (Theorem 20) and Johnson [63] (Theorem 21).

Theorem 20 *For any $\epsilon > 0$, there is an $h \geq 1$ so that if M_{2i+1} has a Hamilton cycle for $1 \leq i \leq h$, then the middle levels graph, M_{2k+1} , has a cycle of length at least $(1 - \epsilon)V_k$ for all $k \geq 1$.*

Theorem 21 *For k sufficiently large there is a cycle of length at least $(1 - o(1))|M_{2k+1}|$ in M_{2k+1} .*

Since it was shown by Moews and Reid [86] in 1990 that M_{2k+1} is Hamiltonian for $1 \leq k \leq 11$, it follows from Theorem 20 that M_{2k+1} has a cycle of length at least $0.838V_k$.

In this chapter, we extend the result of Moews and Reid to show that M_{2k+1} is Hamiltonian for $1 \leq k \leq 15$. Combining this with Theorem 20 slightly improves the lower bound.

Corollary 1 *For every $k > 1$, the middle two levels graph, M_{2k+1} has a cycle of length at least $0.86V_k$.*

Our efforts to find Hamilton cycles in these graphs with up to millions of vertices focus on:

1. transforming M_{2k+1} to a *reduced graph* R_{2k+1} , smaller than M_{2k+1} by a factor of $2(2k + 1)$,
2. identifying *distinguished vertices* u and v in R_{2k+1} with the property that any Hamilton path in R_{2k+1} from u to v is guaranteed to lift to a Hamilton cycle in M_{2k+1} , and
3. devising a heuristic for finding Hamilton paths or cycles, that is fast enough and powerful enough to succeed in Step 2.

The reduction of Step 1 was used by Moews and Reid. The reduction to *necklaces*, was suggested earlier by Dejter [31] who also recognized the distinguished vertices of Step 2. Step 3 is new.

In Section 4.2 we describe the reduction and the Hamilton path in the reduced graph that can be lifted to a Hamilton cycle in the full graph.

The heuristic was described in Chapter 3 and its performance on the middle levels graph is discussed in Section 4.4. As we shall see, the Hamilton path/cycle heuristic was surprisingly successful on the middle levels problem.

4.2 Reducing the Middle Two Levels Problem

4.2.1 Definitions

Given a string X of n symbols, $X = x_1x_2 \dots x_n$, we define a function $\sigma(X)$ by

$$\sigma(x_1x_2 \dots x_n) = x_2x_3 \dots x_nx_1.$$

Evidently $\sigma^n(X) = X$ for any n -element string X and indeed $\sigma^i(X) = \sigma^j(X)$ whenever $i \equiv j \pmod n$. For convenience, we may think of $\sigma^i(X)$ as cyclically shifting the symbols X to the left when $i > 0$ and to the right when $i < 0$.

For the remainder of this chapter we will restrict our attention to strings where the elements are taken from $\{0, 1\}$. We will refer to n -element strings as n -bit binary numbers.

Given a set S of n -bit binary numbers we define a relation \sim on S such that $X \sim Y$ if and only if $Y = \sigma^i(X)$ for some integer i . The relation \sim is an equivalence relation and we refer to the equivalence classes as *necklaces*. We denote the equivalence class of X by $\nu(X)$. Now suppose $X = \sigma^i(X)$ for some $i < n$. Then it follows easily that if $g = \gcd(i, n)$ then it is also true that $X = \sigma^g(X)$ and X is simply the concatenation of n/g copies of a substring of length g .

Lemma 1 *The necklace equivalence classes for the middle two levels for $n = 2k+1$ each have n members.*

Proof. Clearly there can be at most n distinct members of each necklace equivalence class. If there are fewer in some class C , then for $X \in C$ there is some $i < n$ for which $X = \sigma^i(X)$. Let $g = \gcd(i, n)$. Then X is composed of n/g copies of some substring Y of length g . Thus n/g is a factor of both k and $k+1$ so $g = n$ contradicting the assumption of $i < n$. \square

For a binary digit x we define the *complement* \bar{x} as

$$\bar{x} = \begin{cases} 1 & \text{if } x = 0 \\ 0 & \text{if } x = 1 \end{cases}$$

Now suppose $X = x_1x_2 \dots x_n$ is an n -bit binary number. Define \bar{X} in the obvious way as $\bar{X} = \bar{x}_1\bar{x}_2 \dots \bar{x}_n$. We define the complement of a necklace by $\overline{\nu(X)} = \nu(\bar{X})$. Note that this is well-defined since $Y = \sigma^i(X)$ if and only if $\bar{Y} = \sigma^i(\bar{X})$.

4.2.2 The Reduction

As mentioned in Section 4.1, for $n = 2k + 1$ we will denote by M_n or M_{2k+1} the graph corresponding to the middle two levels problem where the vertices are the n -bit binary numbers with k or $k + 1$ ones and $u \leftrightarrow v$ if u and v differ in exactly one bit position. Evidently M_n is $k + 1$ -regular and has $2\binom{n}{k} = 2\binom{n}{k+1}$ vertices.

Let us now consider the graph N_n of necklace equivalence classes with $V(N_n) = \{\nu(u) : u \in M_n\}$, and $\{\nu(u), \nu(v)\} \in E(N_n)$ if and only if there is an i such that u differs from $\sigma^i(u)$ in exactly one bit. By Lemma 1 it follows that N_n has $2\binom{n}{k}/n$ vertices which makes it smaller by a factor of n than M_n . (Note, in fact, that the number of vertices of N_{2k+1} is twice the Catalan number $\binom{2k+1}{k}/(2k+1)$.)

We further reduce the problem size by observing that in N_n , $\nu(X) \leftrightarrow \nu(Y)$ if and only if $\overline{\nu(X)} \leftrightarrow \overline{\nu(Y)}$. We define another equivalence relation $\overset{\circ}{\sim}$ on $V(N_n)$ by $X \overset{\circ}{\sim} Y$ if either $X = Y$ or $X = \overline{Y}$ and denote the equivalence class of $\overset{\circ}{\sim}$ containing X by $\rho(X)$. Since every string in X has odd length, $X \neq \overline{X}$, every equivalence class $\rho(X)$ has exactly 2 elements. We construct a reduced graph R_n whose vertices are the equivalence classes $\{\rho(X) \mid X \in V(N_n)\}$ with edges $\{\rho(X)\rho(Y)\} \in E(R_n)$ if $XY \in E(N_n)$ or $X\overline{Y} \in E(N_n)$. Observe that if $Z = \nu(0^k 1^{k+1})$ then $\overline{Z} = \overline{\nu(0^k 1^{k+1})} = \nu(1^k 0^{k+1})$, and so $Z \leftrightarrow \overline{Z}$ in N_n . Hence $\rho(Z) = \rho(\overline{Z})$ in R_n and R_n has loops, so it is not a simple graph.

4.2.3 Lifting Paths from the Reduced Graphs

In Section 4.2.2 we saw how to reduce the size of the middle two levels problem by constructing smaller graphs from the original. We now show that if a suitably constructed Hamilton path in R_n exists then it can be lifted to construct a Hamilton cycle in N_n . We then show how that cycle in N_n can be lifted to construct a Hamilton cycle in M_n .

We observed earlier that R_n is not a simple graph because some edges are loops.

Lemma 2 *If there is a Hamilton path $P_R = (r_1, r_2, \dots, r_l)$ in R_n and the loops $r_1 r_1$ and $r_l r_l$ are edges in R_n then there is a Hamilton cycle in N_n*

Proof. We construct such a cycle P_N . First choose s_1 in N_n such that $r_1 = \rho(s_1)$ and let $P_{N,1} = (s_1)$ be the path in N_n consisting of this single vertex. Now suppose we have constructed a path $P_{N,k} =$

(s_1, s_2, \dots, s_k) of $k < |N_n|$ vertices in N_n , and suppose $r_{k+1} = \rho(t) = \rho(\bar{t})$ for some t in N_n . Now set

$$s_{k+1} = \begin{cases} t & \text{if } s_k \leftrightarrow t \\ \bar{t} & \text{if } s_k \leftrightarrow \bar{t} \end{cases}$$

Proceeding in this manner we construct the path $P_{N,l} = (s_1, s_2, \dots, s_l)$ of l vertices, where $r_l = \rho(s_l)$.

The desired cycle is then

$$(s_1, s_2, \dots, s_l, \overline{s_l}, \overline{s_{l-1}}, \dots, \overline{s_1})$$

which is a cycle because $\rho(v) \leftrightarrow \rho(v)$ in R_n if and only if $v \leftrightarrow \bar{v}$ in N_n and, by assumption, $r_1 \leftrightarrow r_1$ and $r_l \leftrightarrow r_l$ in R_n . \square

For the special case where $r_1 = \rho(\nu(0^k 1^{k+1}))$ and $r_l = \rho(\nu(0(01)^k))$ we now show that we can lift the Hamilton cycle in the necklace graph to a Hamilton cycle in the full middle two levels graph M_n ,

Lemma 3 *If there is a Hamilton path $P_R = (r_1, r_2, \dots, r_l)$ in R_n and $r_1 = \rho(\nu(0^k 1^{k+1}))$ and $r_l = \rho(\nu(0(01)^k))$ then there is a Hamilton cycle in M_n*

Proof. We first choose a sequence of vertices (Y_1, Y_2, \dots, Y_l) in N_n , as representatives in N_n of the equivalence classes (r_1, r_2, \dots, r_l) in R_n . Similarly, choose a sequence of vertices (X_1, X_2, \dots, X_l) in M_n , as representatives in M_n of the equivalence classes $(N_1, N_2, \dots, N_{2l})$ in N_n . By the construction of N_n and R_n and the assumption of the lemma these vertices may be chosen subject to the following constraints.

1. $r_i = \rho(Y_i) = \rho(\overline{Y_i})$
2. $Y_i = \nu(X_i)$
3. $X_1 = 0^k 1^{k+1}$.
4. X_i is a permutation of $0^k 1^{k+1}$.
5. $X_l = \sigma^j(0(01)^k)$ for some $j < n$.
6. $X_i \leftrightarrow \overline{X_{i+1}}$ in M_n for $1 \leq i < l$

Suppose first that l is odd. By Lemma 2 we may construct a Hamilton cycle

$$\nu(X_1), \nu(\overline{X_2}), \nu(X_3), \dots, \nu(X_l), \nu(\overline{X_l}), \dots, \nu(\overline{X_3}), \nu(X_2), \nu(\overline{X_1})$$

in N_n . Now X_l was chosen so that $X_l = \sigma^j(0(01)^k)$ for some $j < n$ and so we have $\sigma(\overline{X_l}) = \sigma(\overline{\sigma^j(0(01)^k)}) = \sigma(\sigma^j(\overline{0(01)^k})) = \sigma^{j+1}(\overline{0(01)^k}) = \sigma^{j+1}(1(10)^k) = \sigma^j(1(01)^k)$. This differs in one bit from $\sigma^j(0(01)^k) = X_l$ so that $\sigma(\overline{X_l}) \leftrightarrow X_l$ in M_n . We may thus construct a path

$$q = X_1, \overline{X_2}, X_3, \dots, X_l, \sigma(\overline{X_l}), \sigma(X_{l-1}), \dots, \sigma(\overline{X_3}), \sigma(X_2), \sigma(\overline{X_1})$$

of length $2l$ in M_n . Now

$$\sigma(\overline{X_1}) = \sigma(\overline{0^k 1^{k+1}}) = \sigma(1^k 0^{k+1}) = \sigma(\sigma^k(0^k 1^k 0)) = \sigma^{k+1}(0^k 1^k 0)$$

which differs in exactly one bit from $\sigma^{k+1}(0^k 1^k 0) = \sigma^{k+1}(X_1)$. Thus $\sigma(\overline{X_1}) \leftrightarrow \sigma^{k+1}(X_1)$ in M_n . Now $k+1$ and n are relatively prime so we may extend our partial path q in M_n to a Hamilton path:

$$q, \sigma^{k+1}(q), \sigma^{2(k+1)}(q), \dots, \sigma^{(n-1)(k+1)}(q),$$

that is:

$$\begin{aligned} & X_1, \dots, \sigma(\overline{X_1}), \\ & \sigma^{k+1}(X_1), \dots, \sigma^{k+2}(\overline{X_1}), \\ & \vdots \\ & \sigma^{(n-1)(k+1)}(X_1), \dots, \sigma^{(n-1)(k+1)+1}(\overline{X_1}). \end{aligned}$$

Since $\sigma(\overline{X_1}) \leftrightarrow \sigma^{k+1}(X_1)$ we have that

$$\sigma^{(n-1)(k+1)+1}(\overline{X_1}) \leftrightarrow \sigma^{(n-1)(k+1)+k+1}(X_1) = X_1$$

and our Hamilton path is a Hamilton cycle.

If l is even we use Lemma 2 to construct a Hamilton cycle

$$\nu(X_1), \nu(\overline{X_2}), \nu(X_3), \dots, \nu(\overline{X_l}), \nu(X_l), \dots, \nu(\overline{X_3}), \nu(X_2), \nu(\overline{X_1})$$

in N_n .

Since $X_l = \sigma^j(0(01)^k)$ for some $j < n$ we have $\overline{X_l} = \overline{\sigma^j(0(01)^k)} = \sigma^j(\overline{0(01)^k}) = \sigma^j(1(10)^k) = \sigma^{j+1}(011(01)^{k-1})$. This differs in one bit from $\sigma(\sigma^j(0(01)^k)) = \sigma(X_l)$ so that $\overline{X_l} \leftrightarrow \sigma(X_l)$ in M_n and

we may construct a path

$$q' = X_1, \overline{X_2}, X_3, \dots, \overline{X_l}, \sigma(X_l), \sigma(\overline{X_{l-1}}), \dots, \sigma(\overline{X_3}), \sigma(X_2), \sigma(\overline{X_1})$$

As in the odd case, we may extend our path q' in M_n to a Hamilton cycle in M_n :

$$q', \sigma^{k+1}(q'), \sigma^{2(k+1)}(q'), \dots, \sigma^{(n-1)(k+1)}(q').$$

□

4.3 Graph Representation

For the reduced version of the middle levels graph our internal vertex representation was as the canonical (lexically least) necklace using k one bits and $k + 1$ zero bits in the low order $2k + 1$ bits of a 32-bit computer word. We stored these vertex representations in an array of size $|G| = 2\binom{n}{k}$, indexed from zero to $|G| - 1$ so that the representation of the i th vertex was available as the i th element of the array. We implemented a set of utility functions to manipulate n -bit subwords of a 32-bit word for functions such as rotating the n bits or finding the canonical necklace.

We calculated adjacent vertices by successively setting each of the $k + 1$ zero bits to one. Each result was then complemented and the $2k + 1$ low order bits were rotated to find the lexically least necklace. We used a binary search lookup strategy to find the index corresponding to a particular vertex representation. In systems where memory was constrained, this often caused additional swapping because the array references were not localized. For Kneser graphs (Chapter 5), we changed to using an index array which provided comparable performance with better locality of reference.

4.4 Performance of the Heuristic on the Middle Two Levels Graphs

Our BFS heuristic, described in Chapter 3, was applied to search the reduced graphs R_{2k+1} , $k = 1, 2, \dots, 15$, to try to find a Hamilton path from vertex $\rho(\nu(0^{k+1}1^k))$ to vertex $\rho(\nu(0(01)^k))$. A Hamilton path meeting these requirements was found for each $k \leq 15$. The case $k = 15$, which has about 9.6 million vertices in the reduced graph and over 600 million vertices in the full graph, took almost 3

weeks on 400MHz Intel Pentium-II system with 192MB RAM. The timing results are summarized in Table 4.1. We measured elapsed time using a timer with a resolution of 1 second. Runs that start and complete in the same second show a time of 0 seconds. We later ran the program again on a 2.4GHz Intel Pentium 4 system with 512MB of RAM and the table includes those results. Many factors affect the difference in performance between the two systems, including operating system and compiler differences as well as processor, RAM, and disk speed. For $k = 15$, disk activity on the smaller system was almost constant, indicating significant amounts of memory swapping. The larger system ran silently with no disk activity. The dramatic speedup with adequate memory is evident from the timing differences compared with the speedup for cases where $k < 15$.

Table 4.1: Running time to find a Hamilton cycle in the middle two levels graph

Running time for the heuristic to find a Hamilton path in R_n (and thus a Hamilton cycle in the middle two levels of B_n).

k	$n = 2k + 1$	# vertices in R_n	# vertices in M_n	Time (Secs)	
				192MB	512MB
8	17	1,430	48,620	0	0
9	19	4,862	184,756	1	0
10	21	16,796	705,432	2	1
11	23	58,786	2704,156	18	5
12	25	208,012	10,400,600	235	105
13	27	742,900	40,116,600	2,941	1,732
14	29	2,674,440	155,117,520	36,859 (10 hrs)	24,138 (6.7 hrs)
15	31	9,694,845	601,080,390	1,756,134 (3 weeks)	307,976 (3.6 days)

For the middle two levels graph, the number of edges, E is $O(V \log V)$ where V is the number of vertices. For $k = 15$ the maximum height of the BFS tree was ≤ 6 for all searches except possibly the final one to add t for which our instrumentation did not check the tree height. This is much better than the theoretical worst case of $O(V)$. If the maximum height were bounded by $O(\log V)$ then the worst case running time on graphs of this class would be $O(V^2 \log^2 V)$ rather than the general worst case of $O(V^4)$. In contrast to the small height actually seen, the largest BFS tree that was generated had over 6M of the 9.6M graph vertices enqueued at once.

One would expect that on any given run, the algorithm would eventually get stuck and fail to find a Hamilton path; the search for a Hamilton path would then be re-started, from the same vertex in our case, but making different random choices of neighbors. The hope would be that if a Hamilton path exists, it would be found without too many re-starts.

It was striking, however, that in our trials on the reduced middle two levels graph, the algorithm always found a Hamilton path on the first run.

For comparison, consider a less drastic modification of the PosaSearch algorithm which we tried when we first started using it to find a path from a starting vertex s to an ending vertex t . We eliminated the possibility of a trivial rotation but we did not further restrict the algorithm by excluding the desired termination vertex.

Instrumenting this modified PosaSearch algorithm showed that the bulk of our time was spent in processing the final few vertices and then performing rotations to make t the actual endpoint. We made two simple enhancements. The first was to exclude t as a candidate for extension until it was the only vertex not on the path. The second was to choose rotations by giving preference to a rotation in which the new endpoint had a neighbor not already on the path. These two changes alone reduced the running time for $k = 12$ by a factor of four. Furthermore, the second enhancement was the nucleus of what eventually became our BFS search heuristic.

Even with these enhancements the PosaSearch algorithm slowed considerably as larger graphs were searched. Initially we were using a Pentium Pro system with only 128MB of RAM and the PosaSearch algorithm ran for about 8 days on the $k = 14$ case and had still not finished. The first version of our heuristic found a path for $k = 14$ in only four days.

As k grows, the number of vertices, V in M_n grows exponentially. The memory used for the data structures in our implementation is linear in V so our memory requirement grows exponentially, along with V . For $k = 15$ even the reduced graph has almost 10 million vertices. We stored most items as 32-bit integers so that the original vertex list, the path itself and the data structures we used in the BFS search each required about 40MB of storage. For smaller values we were able to generate adjacency lists once and store them in a large array. For larger values we computed adjacency lists each time they were required as this proved to be faster than allowing the operating system to swap our large arrays. Even so, swapping on the $k = 15$ case caused significant slowdown for the last two of the three weeks of the run. Extending the algorithm to handle cases larger than $k = 15$ would mean processing bit strings of at least 33 bits and would probably be easier to do on a 64-bit machine with much greater memory than the 192MB on the 32-bit system that we used.

Our program printed the path in the reduced graph R_n , in order, as output. Each output line included vertex number, the bit representation of the vertex and the complement of that representation. Even this relatively small amount of information resulted in an output file of around 800MB for $k = 15$.

Obviously, such large output cannot reasonably be verified manually, so we wrote a separate verification program not reusing any of the logic from the original program to process the output file and verify that the claimed path was indeed a proper Hamilton path.

Chapter 5

Kneser Graphs

5.1 Kneser and Bipartite Kneser Graphs Overview

In this chapter we use our heuristic (Chapter 3) to extend the known results for Hamilton cycles in Kneser graphs, $K(n, k)$, and bipartite Kneser graphs, $H(n, k)$. With the exception of the Petersen graph, $K(5, 2)$, these have long been conjectured to have Hamilton cycles for $n > 2k$, but neither a constructive nor an existential proof is known. For $n > 2k$, both $K(n, k)$ and $H(n, k)$ are connected and vertex transitive, so the nonexistence of a Hamilton path in $K(n, k)$ or $H(n, k)$ for some n, k would provide a counterexample to the Lovász conjecture [75] that every connected, undirected, vertex transitive graph has a Hamilton path.

The *Kneser graph* $K(n, k)$ has as vertices the k -subsets of $[n] = \{1, 2, \dots, n\}$, where two vertices are adjacent if the k -subsets are disjoint. A related graph, the *uniform subset graph* $G(n, k, t)$, also has as vertices the k -subsets of $\{1, 2, \dots, n\}$ but here two vertices are adjacent if their intersection has cardinality t . So $K(n, k) = G(n, k, 0)$. The *bipartite Kneser graph* $H(n, k)$ has as its partite sets the k - and $(n - k)$ -subsets of $\{1, 2, \dots, n\}$, respectively. Two vertices from different partite sets are adjacent if and only if one is a subset of the other.

There is a one-to-one correspondence between the k -subsets of $\{1, 2, \dots, n\}$ and the set of n -bit binary numbers with exactly k ones and $n - k$ zeros, which simplifies the computer representation and manipulation of these graphs. The graph $K(n, k)$ has $\binom{n}{k}$ vertices while $H(n, k)$ has twice as many vertices and both graphs are regular of degree $\binom{n-k}{k}$.

Simpson [100] studied bipartite graphs $B(G)$ constructed from a graph G with vertex set $V(G) =$

$\{x_1, x_2, \dots, x_n\}$. The vertex set of $B(G)$ consists of two copies of $V(G)$ denoted $S = \{y_1, y_2, \dots, y_n\}$ and $T = \{z_1, z_2, \dots, z_n\}$ and (y_i, z_j) is an edge in $B(G)$ if and only if (x_i, x_j) is an edge in G . He showed that $H(n, k) = B(K(n, k))$.

Chen and Lih [17] showed that $G(n, k, t)$, and therefore $K(n, k)$, are both vertex transitive and edge transitive. Simpson showed that if G has no loops and is vertex (edge) transitive then $B(G)$ is also vertex (edge) transitive and therefore $H(n, k)$ is both vertex and edge transitive.

The graph $K(5, 2)$ is the Petersen graph, which has a Hamilton path but not a Hamilton cycle, although the bipartite graph $H(5, 2)$ does have a Hamilton cycle. It has long been conjectured that $K(n, k)$ and $H(n, k)$ (with the exception of $K(5, 2)$) have Hamilton cycles when $n > 2k$.

In this chapter, we use our heuristic from Chapter 3, with additional modifications described in Section 5.5, to complete the verification of the Hamiltonicity of all the $K(n, k)$ (except for the Peterson graph) and $H(n, k)$ graphs for $n \leq 27$.

In Section 5.2 we describe the previous work and our results for Kneser graphs, $K(n, k)$. In Section 5.3 we extend these results to the corresponding bipartite Kneser graphs $H(n, k)$ using a result due to Simpson and results from our prior work. We show that the Simpson technique cannot be used when $n = 2k + 1$. In Section 5.4.1 we discuss a fast algorithm for generating adjacency lists in Kneser graphs. In Section 5.5 we discuss algorithmic trade-offs used in reaching these results.

5.2 Kneser Graphs

The Kneser graph $K(2k - 1, k - 1)$ is also known as the odd graph O_k . The graph, O_2 , is a triangle, which has a Hamilton cycle, while O_3 is the Petersen graph, which has no Hamilton cycle. Balaban [4] studied the odd graph as the “ k -valent halved combination graph” and exhibited Hamilton cycles for $k = 4$ and $k = 5$. Meredith and Lloyd [79, 80] established Hamilton cycles in O_k for $k = 6$ and $k = 7$ and Mather [78] showed a Hamilton cycle for $k = 8$.

Heinrich and Wallis [55] showed that infinitely many of the Kneser graphs, $K(n, k)$ have Hamilton cycles. In particular, $K(n, k)$ has a Hamilton cycle for $k = 1$, $n \geq 3$ (the complete graph K_n), for $k = 2$, $n \geq 6$ and for $k = 3$, $n \geq 7$. Using a theorem of Baranyai [5] they derived the more general result that $K(n, k)$ has a Hamilton cycle for

$$n \geq k + \frac{k \sqrt[k]{2}}{\sqrt[k]{2} - 1}$$

which tends asymptotically to $k + k^2/\log_2 e$.

The circumference of a graph is the length of its longest cycle. Chen and Lih [17] studied the uniform subset graphs $G(n, k, t)$ and obtained results about the circumference of such graphs, from which they proved that $K(n, k)$ has a Hamilton cycle when $n \geq e(k)$ where

$$e(k) = \min\{n \mid n > 2k \text{ and } \binom{n-1}{k-1} / \binom{n-k}{k} \leq 1\}.$$

This implies that $K(n, k)$ has a Hamilton cycle for $n \geq (1 + o(1))k^2 / \log k$ (Clark and Ismail [24]).

Chen [19] used Baranyai's theorem [5] again to show that $K(n, k)$ has a Hamilton cycle for $n \geq 3k$, a dramatic improvement. She [20] subsequently improved this to show that $K(n, k)$ has a Hamilton cycle whenever

$$n \geq \frac{3k + 1 + \sqrt{5k^2 - 2k + 1}}{2}.$$

We have summarized the known results for Kneser graphs $K(n, k)$ with $n \leq 27$ in Table 5.1. The previous work leaves gaps when

$$2k + 1 < n < \frac{3k + 1 + \sqrt{5k^2 - 2k + 1}}{2} \quad (k = 5, 6, 7)$$

and

$$2k + 1 \leq n < \frac{3k + 1 + \sqrt{5k^2 - 2k + 1}}{2} \quad (k \geq 8).$$

Our Hamilton cycle program was able to find Hamilton cycles in all of the graphs in Table 5.1, thus filling in these gaps.

5.3 Bipartite Kneser Graphs

The bipartite Kneser graph $H(n, k)$ has also been studied under several names. Dejter, Cordova and Quintana [33] constructed Hamilton cycles in $H(16, 7)$ and $H(19, 9)$. Dejter, Cedeño, and Jáuregui [32] found Hamilton cycles using similar methods in $H(2k + 1, k)$ for $k \leq 8$, in $H(2k + 2, k)$ for $k \leq 6$ and in $H(6, 3)$. Simpson [100] showed that if $|G|$ is odd and G has a Hamilton cycle then so does $B(G)$. If $|G|$ is even and $C = x_1, x_2, \dots, x_n, x_1$ is a cycle in G then $B(G)$ has a Hamilton cycle if there is (i) a vertex x_i with i odd, adjacent to x_1 and (ii) x_n is adjacent to either $x_i - 1$ or $x_i + 1$. He showed

Table 5.1: Known results for connected Kneser graphs $K(n, k)$ for $n \leq 27$.

n	$K(n, k) \ k < n/2$													
	1	2	3	4	5	6	7	8	9	10	11	12	13	
3	Δ													Δ - Triangle graph
4	H													P - Petersen Graph
5	H	P												A - Balaban [4]
6	H	H												L - Meredith & Lloyd [79]
7	H	H	A											M - Mather [78]
8	H	H	H											H - Heinrich & Wallis [55]
9	H	H	H	A										B - Chen & Lih [17]
10	H	H	C_2	C_1										C_1 - Chen [18]
11	H	H	H	C_1	L									C_2 - Chen [19]
12	H	H	H	C_2	S									C_3 - Chen [20]
13	H	H	H	C_2	S	L								S - Current Results [99]
14	H	H	H	C_2	S	S								
15	H	H	H	C_2	C_2	S	M							
16	H	H	H	B	C_2	S	S							
17	H	H	H	B	C_2	S	S	S						
18	H	H	H	B	C_2	C_2	S	S	S					
19	H	H	H	B	C_2	C_2	S	S	S	S				
20	H	H	H	B	C_2	C_2	S	S	S	S				
21	H	H	H	B	C_2	C_2	C_2	S	S	S	S			
22	H	H	H	B	B	C_2	C_2	C_3	S	S	S			
23	H	H	H	B	B	C_2	C_2	C_3	S	S	S	S		
24	H	H	H	B	B	C_2	C_2	C_2	S	S	S	S		
25	H	H	H	B	B	C_2	C_2	C_2	C_3	S	S	S	S	
26	H	H	H	B	B	C_2	C_2	C_2	C_3	S	S	S	S	
27	H	H	H	B	B	C_2	C_2	C_2	C_2	C_3	S	S	S	S

that $H(n, k)$ has a Hamilton cycle when

$$2\binom{n-1}{k-1} \leq 1 + \binom{n-k}{k}.$$

A slightly weaker condition was given in [101] where it was shown that $H(n, k)$ has a Hamilton cycle when $n \geq (3k^2 + k + 2)/2$.

Hurlbert [60] also studied $H(n, k)$ calling it the antipodal layers problem and he showed that $H(n, k)$ has a Hamilton cycle for $n > ck^2 + k$ for large enough k .

Chen [19] showed that $H(n, k)$ has a Hamilton cycle for $n > 3k$, and $H(3k, k)$ has one when $\binom{3k}{k}$ is odd, by extending her results for Kneser graphs in a manner similar to that used by Simpson for extending the results for a general graph G to the bipartite graph $B(G)$. She [20] subsequently improved this to show that $H(n, k)$ has a Hamilton cycle whenever

$$n \geq \frac{3k + 1 + \sqrt{5k^2 - 2k + 1}}{2}.$$

The special case of $H(2k + 1, k)$ is the *middle two levels problem*. As we showed in Chapter 4, we obtained the current best result, that $H(2k + 1, k)$ has a Hamilton cycle for all $k \leq 15$.

We have summarized the known results for bipartite Kneser graphs $H(n, k)$ with $n \leq 27$ in Table 5.2. The previous work leaves gaps when

$$2k + 2 < n < \frac{3k + 1 + \sqrt{5k^2 - 2k + 1}}{2} \quad (k = 4, 5, 6, 7)$$

and

$$2k + 1 < n < \frac{3k + 1 + \sqrt{5k^2 - 2k + 1}}{2} \quad (k \geq 8).$$

We were able to fill in all of these gaps using our Hamilton cycle program. Although the heuristic could be run independently for Kneser and bipartite Kneser graphs, instead we added code to perform the Simpson [100] test whenever a Hamilton cycle was found in a Kneser graph under test. This provided a relatively quick check to determine if the bipartite analog, $H(n, k)$, of a Kneser graph, $K(n, k)$, was Hamiltonian. The cases in which this test did not prove $H(n, k)$ to be Hamiltonian when $K(n, k)$ was Hamiltonian were all cases where $n = 2k + 1$ or, in other words, $H(n, k)$ was an instance of the middle two levels problem. In Chapter 4 we showed $H(n, k)$ to be Hamiltonian for $n \leq 31$ or $k \leq 15$.

Table 5.2: Known results for connected bipartite Kneser graphs $H(n, k)$ for $n \leq 27$.

n	$K(n, k) \ k < n/2$													
	1	2	3	4	5	6	7	8	9	10	11	12	13	
3	D ₁													
4	D ₁													
5	I	D ₁												
6	I	D ₁												
7	I	H	D ₁											
8	I	I	D ₁											
9	I	I	D ₁	D ₁										
10	I	I	C ₂	D ₁										
11	I	I	H	S	D ₁									
12	I	I	H	C ₂	D ₁									
13	I	I	C ₂	C ₂	S	D ₁								
14	I	I	H	C ₂	C ₃	D ₁								
15	I	I	H	C ₂	C ₂	S	D ₁							
16	I	I	I	C ₂	C ₂	C ₃	D ₂							
17	I	I	I	C ₂	C ₂	C ₃	S	D ₁						
18	I	I	I	C ₂	C ₂	C ₃	S	S						
19	I	I	I	C ₂	C ₂	C ₂	C ₃	S	D ₂					
20	I	I	I	H	C ₂	C ₂	C ₃	S	S					
21	I	I	I	H	C ₂	C ₂	C ₃	S	S	M				
22	I	I	I	H	C ₂	C ₂	C ₂	C ₃	S	S				
23	I	I	I	H	C ₂	C ₂	C ₂	C ₃	S	S	M			
24	I	I	I	H	C ₂	C ₂	C ₂	C ₂	C ₃	S	S			
25	I	I	I	H	C ₂	C ₂	C ₂	C ₂	C ₃	S	S	S ₁		
26	I	I	I	H	C ₂	C ₂	C ₂	C ₂	C ₃	S	S	S		
27	I	I	I	I	C ₂	C ₂	C ₂	C ₂	C ₂	C ₃	S	S	S ₁	

Since the cases where the Simpson test failed represented just one of many possible Hamilton cycles in $K(n, k)$ the question arises as to whether a Hamilton cycle in $K(n, k)$ does exist for which the Simpson test would work. We show this is not possible.

Theorem 22 *If $n = 2k + 1$ then no Hamilton cycle in $K(n, k)$ can satisfy the Simpson test.*

Proof. The Simpson test requires that four vertices of $K(n, k)$ form a cycle. If $k = 1$ then $|K(n, k)| = 3$ and no 4-cycle exists. Suppose $k > 1$ and that four vertices, a, b, c, d of $K(n, k)$ form a cycle. Now a, b, c, d are k -subsets of $\{1, 2, \dots, n\}$. Since $n = 2k + 1$ there exists $i \in \{1, 2, \dots, n\}$ such that $i \notin a$ and $i \notin b$. Hence $i \in c$ otherwise $c = a$. Since $c \cap d = \phi$ we have $i \notin d$. Since $n = 2k + 1$ this is impossible since $d \neq b$. \square

Thus it can be seen that the Simpson test may fail to show a graph $B(G)$ to be Hamiltonian even when a Hamilton cycle in the underlying graph G is known.

5.4 Graph Representation

For Kneser graphs we stored the representation of each vertex as a word with k one bits and $n - k$ zero bits in an array of 32-bit words. These were stored in lexicographic order, and we used the array index as the vertex number. In contrast to the binary search method for lookup of a particular vertex representation to find the corresponding vertex number used for the middle two levels graph in Chapter 4, we switched to using an index table lookup. If the largest bit representation of a vertex was a binary number that exceeded the size of our index table, we truncated enough low order bits of each vertex so that the resulting largest representation was smaller than the number of elements of our index array. Using the index array we could locate a vertex close to the desired one by direct lookup. A linear search was then used to locate the actual vertex. With a lookup table containing 2^{20} or approximately 1,000,000 entries, a largest graph containing approximately 20,000,000 vertices, and a reasonably even distribution of vertices among the index entries, the linear search had performance comparable to a binary search with the advantage of much better locality of memory reference.

Although the Simpson test eliminated any need for us to generate bipartite Kneser graphs, we had, in fact, implemented such code before implementing the Simpson test. Logically, we added the vertices with $n - k$ ones and k zeroes to the vertex array for the Kneser graphs. If a particular vertex was stored in the i th position from the beginning of the array, its complement was stored in the i th

position from the end. Given this fact, and the ease of complementing words in a computer, we did not actually store these additional vertices, but rather did all computations on the vertices of the Kneser graph, complementing as necessary when vertices in the other partite set were involved.

We discuss our fast algorithm for generating adjacency lists for Kneser graphs in the next section.

5.4.1 Fast Computation of Adjacencies in Kneser Graphs

As we saw with the middle two levels problem in Section 4, it was advantageous to recalculate adjacency lists when needed rather than store the complete adjacency lists either as an array or a set of linked lists. The Kneser graph $K(27, 10)$ has 8,436,285 vertices, each of degree 19,448 making a total of 82,034,353,340 edges. Even using only 3 bytes of storage to represent a vertex would require over 480 gigabytes of memory for the adjacency lists (assuming, as is customary, that each edge would be represented in two adjacency lists). This is very far out of the range of typical computer memory systems. Even storing this large a structure on disk would require much larger disks than are common on typical desktop systems, to say nothing of the performance effect of so much disk activity. We will now describe a very fast algorithm that we devised for generating the adjacencies of a particular vertex of a Kneser graph whenever needed. The only adjacency list ever stored was the last one computed.

The *Kneser graph* $K(n, k)$ has as vertices the k -subsets of $\{1, 2, \dots, n\}$, where two vertices are adjacent if the k -subsets are disjoint. Thus, for a vertex $v \in V(K(n, k))$ the neighbors of v are the k -subsets of $\{1, 2, \dots, n\} \setminus v$. For a vertex v and integer i we define the function $f(v, i) = |\{(j|j \in v, j < i)\}|$, the size of the subset of v whose elements are smaller than i .

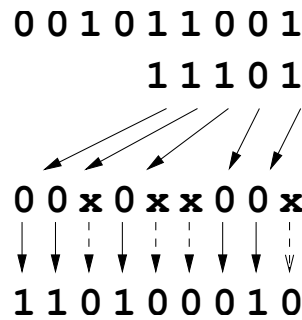
For any k -subset v of $\{1, 2, \dots, n\}$ we define a function $g_v : \{1, 2, \dots, n\} \setminus v \rightarrow \{1, 2, \dots, n - k\}$ by $g_v(i) = i - f(v, i)$.

Lemma 4 *The function g_v is a bijection.*

Proof. $f(v, i) < i$ so $g_v(i) \geq 1$. Now let $i \in \{1, 2, \dots, n\} \setminus v$. Then

$$g_v(i) = i - f(v, i) = |\{1, 2, \dots, i\}| - |\{(j|j \in v, j < i)\}| = |\{1, 2, \dots, i\} \setminus \{j|j \in v, j < i\}| \leq n - k$$

since $\{1, 2, \dots, i\} \setminus \{j|j \in v, j < i\} \subseteq v^c$ where v^c is the complement of v in $\{1, 2, \dots, n\}$. therefore, g_v is a function. When $i < j$ then $j - i > f(v, j) - f(v, i)$ and so $g_v(i) < g_v(j)$. Therefore g_v is an injection and hence a bijection. \square

Figure 5.1: Inserting bits ($n = 9, k = 4$)

Now suppose v is vertex of $K(n, k)$. Let $N(v)$ be the set of vertices adjacent to v (the *neighbors* of v). Let $w = \{w_1, w_2, \dots, w_k\} \in N(v)$ be a neighbor of v . Then w is a k -subset of $\{1, 2, \dots, n\} \setminus v$. We define a function $h_v : N(v) \rightarrow V(K(n - k, k))$ from $N(v)$ to the vertices of $K(n - k, k)$ by

$$h_v(w) = \{g_v(w_1), g_v(w_2), \dots, g_v(w_k)\}.$$

It follows from Lemma 4 that h_v is also an bijection.

For computational purposes we represented the vertex $v = \{v_1, v_2, \dots, v_k\}$ by the unsigned integer $(2^{v_1} + 2^{v_2} + \dots + 2^{v_k})/2$. We stored the vertices of $K(n, k)$ in ascending lexicographic order and so the first $\binom{n-k}{k}$ vertices had the same representation as the vertices of $K(n - k, k)$. We used this fact to derive a fast algorithm for generating the neighbors of a vertex v using the inverse of the function h_v . The $n - k$ bits of a vertex of $K(n - k, k)$ are inserted into the the $n - k$ positions occupied by zeroes in v while the k ones serve as a mask to control the operation as illustrated in Figure 5.1.

The C++ code used for this is shown in Figure 5.2. The loop is executed k times to generate each neighbor. The code was compiled for an Intel Pentium processor. As shown in Figure 5.3, the compiled loop was very tight with one jump instruction, and the other 9 instructions using only register operations.

5.5 Performance of the Heuristic on Kneser Graphs

As described in Section 5.1, the heuristic we used to find Hamilton cycles in the Kneser graphs is the rotation-extension heuristic described in Chapter 3. The Kneser graph, $K(27, 13)$, has 20,058,300 vertices, each of degree 14, making 140,408,100 edges. The Kneser graph, $K(27, 10)$, has 8,436,285


```

unsigned int AddMaskedBitstoWord(unsigned int s,unsigned int mask){
    while (mask != 0) {
        unsigned int lo = mask ^ (mask - 1);
        mask &= ~lo;
        lo >>=1;
        s += (s & ~lo);
    }
    return s;
}

```

Figure 5.2: The Kneser adjacency algorithm

```

.L70:
    lea    %eax, [%ecx-1]
    xor    %eax, %ecx
    mov    %edx, %eax
    shr   %eax, 1
    not   %eax
    and   %eax, %ebx
    not   %edx
    add   %ebx, %eax
    and   %ecx, %edx
    jne   .L70

```

Figure 5.3: Intel assembly for the adjacency loop

vertices, each of degree 19,448, making over $82 * 10^9$ edges.

Handling such large graphs required improvements in both the space and time requirements for the heuristic. In a system where the program requires more memory than the system has available, memory is swapped to disk by the operating system. When this occurred, execution times tended to increase significantly.

Some optimization of storage used for intermediate work was done initially to reduce storage requirements. Part of this was later taken back to significantly reduce execution time in the tree building phase (see Section 3.4.2 of Chapter 3). When storage again became a problem with large graphs of high degree, another trade-off was required. Even if virtual memory was large enough to hold all adjacency lists, keeping them resulted in significant memory swapping, particularly during the tree building phase. We used the fast adjacency generation algorithm of Section 5.4.1 to generate adjacency lists as required. The significant saving in swapping more than offset the cost of the extra computations for these problems. These improvements allowed us to obtain our results on a 1.6GHz personal computer with 640MB of memory.

The long running times required for larger graphs also made a checkpoint function desirable to allow restarting a problem part way through. To accomplish this we wrote a file containing the partial path along with some key graph information every half hour.

We have summarized the running times and number of rotation operations needed in Table 5.3. Note that the times shown here include (i) the time to test Hamiltonicity of $K(n, k)$, (ii) the time to take a checkpoint every half hour and (iii) the time required to perform the Simpson test on all possible rotations of the cycle in $K(n, k)$ to determine if $H(n, k)$ has a Hamilton cycle.

Table 5.3: Heuristic running times for Kneser and bipartite Kneser graphs

Times and number of rotations required for determining that the Kneser graph $K(n, k)$ and bipartite Kneser graph $H(n, k)$ are Hamiltonian for $10 \leq n \leq 27$.

n	k	$ V $	Deg	Time (sec)	Rotations
10	4	210	15	0	16
11	4	330	35	0	10
11	5	462	6	0	49
12	4	495	70	0	5
12	5	792	21	0	34
13	5	1287	56	0	33
13	6	1716	7	0	201
14	5	2002	126	0	16
14	6	3003	28	0	90
15	5	3003	252	0	11
15	6	5005	84	0	64
15	7	6435	8	0	1178
16	6	8008	210	0	47
16	7	11440	36	1	288
17	6	12376	462	1	36
17	7	19448	120	0	181
17	8	24310	9	1	4020
18	6	18564	924	3	39
18	7	31824	330	3	142
18	8	43758	45	1	947
19	7	50388	792	11	87
19	8	75582	165	6	506
19	9	92378	10	31	14628
20	7	77520	1716	33	57
20	8	125970	495	24	303
20	9	167960	55	39	2932
21	7	116280	3432	84	36
21	8	203490	1287	82	192
21	9	293930	220	64	1493
21	10	352716	11	711	52336
22	8	319770	3003	309	139
22	9	497420	715	191	802
22	10	646646	66	582	10081
23	8	490314	6435	750	108
23	9	817190	2002	570	493
23	10	1144066	286	611	4457
23	11	1352078	12	10753	94270
24	8	735471	12870	2069	73
24	9	1307504	5005	1968	315
24	10	1961256	1001	1316	2199
24	11	2496144	78	8206	33051
25	9	2042975	11440	6171	228
25	10	3268760	3003	3699	1212
25	11	4457400	364	6155	13291
25	12	5200300	13	159817	344435
26	9	3124550	24310	19032	145
26	10	5311735	8008	15137	765
26	11	7726160	1365	9799	6164
26	12	9657700	91	106054	111573
27	10	8436285	19448	52759	583
27	11	13037895	4368	29192	3361
27	12	17383860	455	81878	42059
27	13	20058300	14	2476824	1299446

Chapter 6

Cayley Graphs

6.1 Introduction to Cayley Graphs

Cayley graphs occur frequently as models of interconnection networks. See the article by Heydemann [59], for example, which also contains excellent diagrams of several common Cayley graphs.

Cayley graphs also occur in change ringing of bells, which has been studied in Britain since the seventeenth century. Ringing all n bells in a bell tower in one of the $n!$ possible ways is called a *change*. Ringing the bells in sequence from highest pitch to lowest is called a *round*. An *extent* involves ringing all $n!$ changes of n bells. The original constraints placed on an extent were:

1. The extent begins with a round and ends with a round.
2. No other change is repeated
3. No bell changes its order of ringing by more than one position from one change to the next.

Modern extents add three additional requirements. For an introduction to the algebra of change ringing see Dechéne [30]. White [109] has shown that the constraints give rise to a set of involutions in S_n and that an extent exists if and only if the Cayley graph on S_n generated by these involutions is Hamiltonian.

If G is a group and $\Gamma = \text{Cay}(G : X)$ is the Cayley graph generated by some generating set $X \subseteq G$, then for any $a, b \in G$ we have $b = x_1 x_2 \dots x_k a$, where $x_i \in X$ or $x_i^{-1} \in X$ and it follows that Γ is vertex transitive. However, not every vertex transitive graph is a Cayley graph. For example, the Petersen

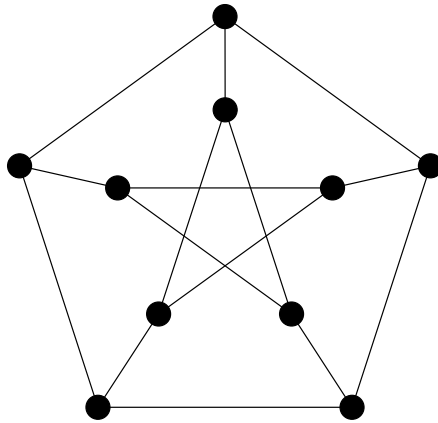


Figure 6.1: The Petersen Graph

graph on 10 vertices (see Figure 6.1) is the smallest example of a vertex-transitive graph which is not a Cayley graph.

Let us suppose the Cayley graphs $\Gamma_1 = \text{Cay}(G_1 : X_1)$ and $\Gamma_2 = \text{Cay}(G_2 : X_2)$ are isomorphic. Then the groups G_1 and G_2 are isomorphic groups. However, the converse is not the case. Let $G = \{0, 1, 2, 3, 4, 5\}$ be the group of integers under addition modulo 6. Then $X_1 = \{1\}$ and $X_2 = \{2, 3\}$ are generating sets for G . But $\text{Cay}(G : X_1)$ is a cycle while $\text{Cay}(G : X_2)$ is a cubic graph.

Since Cayley graphs are vertex transitive, the existence of a non-Hamiltonian Cayley graph would provide a counterexample to the Lovász conjecture [75] that every connected, undirected, vertex transitive graph has a Hamilton path. As with the middle two levels problem, both constructive and existential approaches have so far been unsuccessful in establishing the existence of a non-Hamiltonian Cayley graph.

6.1.1 Hamiltonian Properties of Cayley Graphs

It is well known that the elements of the symmetric group S_n can be arranged such that successive elements differ only by a transposition and also that this group can be generated by a minimal generating set, or basis, B , of $n - 1$ transpositions. Kompel'maher and Liskovec [69] showed that $\text{Cay}(S_n : B)$ is Hamiltonian for any basis B consisting of transpositions.

Let $\text{Cay}(S_n : X)$ be a Cayley graph with any generating set of transpositions X . And let

$$M_x = \{(g, gx) | g \in S_n\}.$$

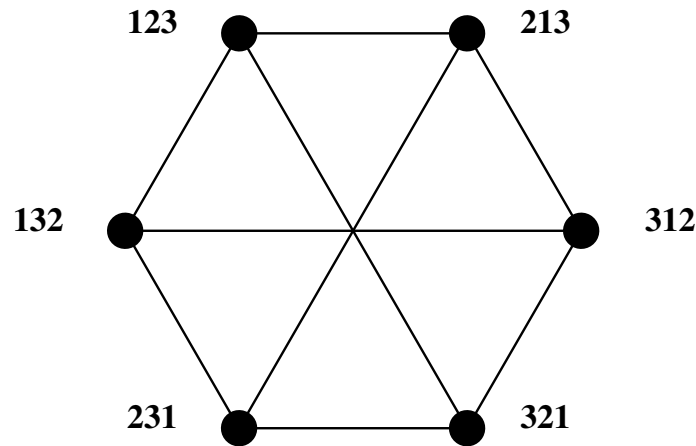


Figure 6.2: S_6 has cycles of length 4 and non-commuting generators

Then M_x is a perfect matching in S_n . Ruskey and Savage [91] investigated the problem of finding Hamilton cycles in Cayley graphs by extending such transposition matchings to a cycle. They proved the following.

Theorem 23 *Let X be a generating set of transpositions for S_n , where $n > 4$. Then for any $x \in X$, M_x extends to a Hamilton cycle in $\text{Cay}(S_n : X)$.*

More generally, if X is a set of generators for a group G and $x \in X$ is an *involution*, that is, an element of order 2 then x induces a perfect matching in $\text{Cay}(G : X)$. Ruskey and Savage show that there is, in general, no x -alternating Hamilton cycle in $\text{Cay}(G : X)$. They ask the question as to whether there is an x -alternating Hamilton path in $\text{Cay}(G : X)$.

Sjerve and Cherkassoff [102] also studied Cayley graphs where the group generating set consisted of three involutions. They completely determine those alternating groups A_n , symmetric groups S_n and projective groups $\text{PSL}_2(q)$ and $\text{PGL}_2(q)$ which can be generated by three involutions, two of which commute. They also show, by embedding the graph in a surface, that, for a finite group having this property, the Cayley graph corresponding to the generators has a Hamilton cycle. The question of whether a Cayley graph generated by three involutions no two of which commute is still open.

When two involutions commute, it follows that the smallest even cycle in the graph is of length 4. The converse is not the case as is illustrated by the Cayley graph generated by the three involutions $(1)(23)$, $(12)(3)$ and $(13)(2)$ from the symmetric group S_6 . The graph is illustrated in Figure 6.2.

6.2 Hamiltonian Cubic Cayley Graphs

We will now turn our attention to *cubic* Cayley graphs, that is, those where every vertex has degree three. We are particularly interested in the symmetric group, S_n , and its subgroups. Suppose X is a set of generators for a group $G \subseteq S_n$. If $\text{Cay}(G : S)$ is cubic then either X is a set of three involutions or a set consisting of one involution and one element of G that is not an involution. The latter type can be studied as either a directed graph or an undirected graph but our research focuses on the undirected form.

6.2.1 First Experiments

We initially applied our algorithm to find Hamilton cycles in Cayley graphs generated by three involutions. As noted above, Cayley graphs generated by three involutions where two or more of the involutions commute were already known to be Hamiltonian [102]. Accordingly, we focused our attention on the case where no two of the involutions commuted. We initially studied graphs $\text{Cay}(S_n : X_n)$ where $X_n = (\alpha_n, \beta_n, \gamma_n)$ with α a transposition, β a reversal, and γ a reversal of all but one symbol as shown below.

	One line notation	Cycle notation
α	$(2, 1, 3, \dots, n)$	$(1, 2)$
β	$(n, n-1, \dots, 1)$	$(1, n)(2, n-1), \dots, (\frac{n}{2}, \frac{n}{2} + 1)$ n even $(1, n)(2, n-1), \dots, (\frac{n+1}{2} - 1, \frac{n+1}{2} + 1)(\frac{n+1}{2})$ n odd
γ	$(1)(n, n-1, \dots, 2)$	$(1)(2, n)(3, n-1), \dots, (\frac{n}{2}, \frac{n}{2} + 2)(\frac{n}{2} + 1)$ n even $(1)(2, n)(3, n-1), \dots, (\frac{n+1}{2}, \frac{n+1}{2} + 1)$ n odd

We will use either cycle notation or one line notation for permutations as may be convenient for a particular case. For computer use we used an internal representation similar to one line notation. Before the advent of computers, it was customary to use the symbols $(1, 2, \dots, n)$ when describing permutations. Since the advent of computers some writers use the symbols $(0, 1, \dots, n-1)$ instead. Despite this variety of possible representation it will always be evident from the context which form is being used.

After our success with other types of vertex transitive graphs we were very surprised when we were not able to find results. Our heuristic, even when allowed to try a large number of times with a different random number seed each time, was unable to find Hamilton paths or cycles in $\text{Cay}(S_7, X_7)$ which has only 5,040 vertices.

We decided to run a simple backtrack algorithm with no pruning or other enhancements as a comparison. This eventually ran for over a year on a 300MHz Intel processor without finding a Hamilton cycle.

6.2.2 The Vandegriend Algorithm

Vandegriend [107] had investigated the performance of several different Hamilton cycle algorithms on a variety of graphs with up to 1,000 vertices. One of the algorithms used was a backtrack algorithm which used extensive pruning techniques to reduce the size of the search tree.

We modified this code to use dynamically allocated data structures so that we could handle larger graphs than those studied by Vandegriend without needing compiled maximum graph sizes. We also added code based on our existing code to handle generation of permutation groups using three involutions or an involution and another generator and its inverse. We used this modified code to show that several Cayley graphs generated by three involutions, no two of which commuted, were Hamiltonian.

Although we had some success with smaller graphs generated by three involutions, this code was still problematic when graphs contained a couple of thousand vertices. Nevertheless, we were able to answer some previously unsolved questions as we shall see in Section 6.2.3.

6.2.3 The Work of Effler and Ruskey

At about this time Effler and Ruskey [36] started cataloging results for isomorphism and Hamiltonicity of Cayley graphs. They had confirmed that all but two of the cubic Cayley graphs on the alternating group, A_7 , with generators from S_7 were Hamiltonian. They had spent thousands of hours of computer time on one of these using algorithms that had succeeded on similar graphs without finding a Hamilton cycle and were about to embark on an exhaustive search on a supercomputer.

We were invited to try our algorithm on these graphs. The graph A_7 has 2520 vertices and the first set of generators was $(0)(1)(2)(3,4)(5,6)$ and $(0,1,2,3)(4,5)(6)$. We made further modifications to the Vandegriend code to handle graphs generated by one involution and one non-involution. Using this version of the Vandegriend algorithm we found a Hamilton cycle within a minute or so of computation time. Effler and Ruskey had also been unable to find a Hamilton cycle in the Cayley graph of A_7 generated by $(0)(1)(2)(3,4)(5,6)$ and $(0,3)(1,4,2,5)(6)$. Again, the modified Vandegriend code found a Hamilton cycle within a minute or so of computation time.

Nevertheless, the Vandegriend algorithm slowed down significantly on larger problems. For example, we did not find a cycle using this algorithm on the graph with 2520 vertices generated by the involution (03)(14)(25)(6) and the element (0)(1)(2)(3456) and its inverse which is result 7.294 in the table of Effler and Ruskey (Section 6.4).

6.2.4 Random Number Issues

We had been running our algorithm mostly on IBM OS/2 and Microsoft Windows systems up until this time and we realized that the 16 bit random number generator commonly used in older ANSI C systems had an unfortunate side effect of generating sequences that alternated between positive and negative numbers. In exploring 3-regular graphs, such as those under study, we arrive at a vertex via one edge and have only two possible choices for a departure edge. Choosing such an edge using a random number modulo 2 when such random numbers alternate between the two possible remainders was clearly inadequate.

A full discussion of random number generation and use is beyond the scope of this paper. Press et al. [84] suggest the following:

If you want to generate a random integer between 1 and 10, you should always do it by using high-order bits, as in

$$j = 1 + (\text{int})(10.0 * \text{rand}) / (\text{RAND}_{MAX} + 1.0)$$

and never by anything resembling

$$j = 1 + (\text{rand}) \% 10$$

(which uses lower-order bits).”

If the largest possible random number returned by the random number generator is R and the number actually returned is r and we desire a random number with a maximum value of D , then the value d that we should use is calculated so that $d/D = r/R$. Rather than use floating point computation as would be required for the suggested expression above, or implementing a function to calculate the desired value using integer arithmetic, we decided to switch to Linux where the `rand()` function is implemented using the same random number generator as `random()` and `srandom()`. On our target 32-bit Intel machines this meant a 32-bit random number where the lower-order bits should be as random as the higher-order bits. We felt that we could continue with the simpler modulo arithmetic and eliminate, or at least lessen, the impact of poor random number generation in this way. Furthermore, no code

changes were required to implement this.

6.3 Graph Representation

The cubic Cayley graphs that we studied had fewer vertices and smaller degree than our large Kneser and middle two levels graphs. The graphs all have degree three, so it is feasible to store these as arrays with the row index representing the vertex number and the three columns representing the adjacent vertices.

Generation of permutations was done using an algorithm due to Johnson [64] and Trotter [106] which generates permutations such that each permutation differs from the previous one by a transposition of adjacent symbols. Simple algorithms to compute the rank of a permutation generated in this way, or to compute a permutation given its rank, exist. See, for example, Stanton and White [104]. We created a class in C++ to handle these and other aspects of permutation manipulation that we needed, including composing permutations, finding inverses and conjugates.

Let G be the group generated by a set, $X \subseteq S_n$. We use a breadth first search technique to generate G . In the event that $G \neq S_n$ we relabel the vertices of G from 0 to $|G| - 1$ for storing in an adjacency array.

6.4 First Results for Cubic Cayley Graphs

With this change we had more success with our heuristic, although we often required multiple attempts with a different random number seed on each attempt. Our successful results for some of the graphs used by Effler and Ruskey [36] generated by an involution and an element of S_n that is not an involution are shown in Table 6.1. The name shown is the internal name for the graph as used by Effler and Ruskey. In Section 6.5 we shall discuss characteristics of graphs where our heuristic was unsuccessful.

6.5 Performance of the Heuristic on Cubic Cayley Graphs

When a cubic Cayley graph has a minimum odd cycle of length 3, then every vertex lies on such a cycle and a Hamilton path through the graph must pass along two of the edges of such a triangle in succession as shown in Figure 6.3.

Table 6.1: Cubic Cayley graphs generated by one involution and one non-involution.

Running time and number of attempts before successfully finding a Hamilton cycle in Cubic Cayley graphs generated by one involution and one non-involution.

$ G $	σ τ	Diameter	Shortest Odd Cycle	Shortest Even Cycle	Time (seconds)	Attempts	Name
1,512	(0)(12)(34)(56)(78) (013)(2)(457)(6)(8)	20	3	18	3	9	result9.1853
2,160	(0)(1)(2)(34)(56)(78) (03)(1478)(25)(6)	18	none	4	1	1	result9.1166
2,160	(0)(1)(2)(34)(56)(78) (03)(1564)(27)(8)	18	none	4	1	1	result9.1337
2,520	(0)(1)(2)(34)(56) (0123)(45)(6)	19	21	4	1	1	result7.102
2,520	(0)(1)(2)(34)(56) (03)(1425)(6)	19	27	4	2	5	result7.154
2,880	(0)(1)(2)(34)(56)(78) (0134)(2576)(8)	24	none	4	1	1	result9.874
5,040	(0)(12)(34)(5 6) (01)(2356)(4)	23	27	4	4	7	result7.1000
5,040	(0)(12)(34)(56) (0123)(45)(6)	23	25	4	1	1	result7.1009
5,040	(0)(12)(34)(56) (0132)(45)(6)	20	27	4	1	1	result7.1020
5,040	(0)(12)(34)(56) (0135)(2)(46)	19	27	4	1	2	result7.1034
5,040	(0)(12)(34)(56) (013)(245)(6)	28	3	20	36	2211	result7.1037
5,040	(0)(12)(34)(56) (0135)(24)(6)	19	27	4	3	3	result7.1044
5,040	(0)(12)(34)(56) (013)(254)(6)	28	3	20	27	1472	result7.1048
5,040	(0)(12)(34)(56) (0134)(25)(6)	23	25	4	2	3	result7.1058
5,040	(0)(12)(34)(56) (0136)(25)(4)	19	25	4	1	1	result7.1069
5,040	(0)(1)(2)(3456) (03)(14)(25)(6)	25	none	4	1	2	result7.294
5,040	(0)(12)(34)(56) (01)(2345)(6)	23	27	4	1	1	result7.996
5,040	(0)(12)(34)(56) (01)(2354)(6)	20	29	4	3	4	result7.998
5,040	(0)(1)(2)(34)(56)(78) (03)(1425)(6)(7)(8)	20	29	4	2	2	result9.1129
5,040	(0)(1)(2)(34)(56)(78) (03)(1425)(6)(78)	27	none	4	2	2	result9.1130
5,040	(0)(1)(2)(3)(4)(56)(78) (01)(25)(3647)(8)	21	27	4	2	2	result9.124
5,040	(0)(12)(34)(56)(78) (01)(2345)(6)(7)(8)	23	27	4	1	1	result9.1501
5,040	(0)(12)(34)(56)(78) (01)(2354)(6)(7)(8)	20	29	4	4	4	result9.1515
5,040	(0)(12)(34)(56)(78) (01)(2356)(4)(7)(8)	23	27	4	5	7	result9.1533
5,040	(0)(12)(34)(56)(78) (0123)(45)(6)(7)(8)	23	25	4	1	1	result9.1623
5,040	(0)(12)(34)(56)(78) (0132)(45)(6)(7)(8)	20	27	4	1	1	result9.1733
5,040	(0)(12)(34)(56)(78) (0135)(2)(4)(6)(78)	25	none	4	1	1	result9.1894
10,080	(0)(12)(34)(56)(78) (01)(2356)(4)(78)	27	none	4	21	51	result9.1534
10,080	(0)(12)(34)(56)(78) (0123)(45)(6)(78)	26	none	4	9	15	result9.1624
10,080	(0)(12)(34)(56)(78) (0132)(45)(6)(78)	27	none	4	5	3	result9.1734

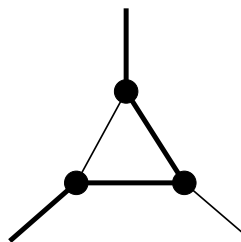


Figure 6.3: Triangular subgraphs in a cubic Cayley graph.

We added code to our program to detect this situation and reduce the graph to a smaller equivalent triangle-free graph. The running times for those graphs with a minimum odd cycle of length 3 shown in Table 6.1 reflect this enhancement. We also added a parameter to our algorithm to allow specification of the starting random number seed and another to specify the maximum number of attempts that the program should make to find a Hamilton cycle.

From the results in Table 6.1 it can be seen that this version of our algorithm was usually able to find a Hamilton cycle in a relatively small number of attempts, when it could find one at all. For this size of cubic Cayley graph, the graph was constructed and the cycle usually found in a matter of a few seconds. Nevertheless, two of these graphs required over 1,000 attempts to find a Hamilton cycle.

This version of the algorithm was able to find a Hamilton cycle quickly in the graph with 2520 vertices generated by the involution $(03)(14)(25)(6)$ and the element $(0)(1)(2)(3456)$ and its inverse which had defied our efforts with the Vandegriend algorithm (Sec 6.2.2). However, our algorithm was still unsuccessful in finding a Hamilton cycle in many graphs, even with as many as 100,000 attempts as was the case with the graph generated by the three involutions $(0)(1)(24)(36)(5)$, $(01)(2)(3)(4)(56)$ and $(02)(13)(4)(5)(6)$ which has diameter 16, a smallest odd cycle of 21, and a smallest even cycle of 6. This is the graph, result7.10108, in the table of Effler and Ruskey (Section 6.2.3).

A common feature of most of the graphs where we were unsuccessful seems to be the diameter of the graph which tends to be among the larger possible diameters for these graphs, often with a small minimum even cycle.

Every time we terminated with a path that was longer than any previous path, we printed the path length and the random number seed that was used to find that path. This allowed the program to be restarted with the best seed to date.

The tables of Effler and Ruskey contain many graphs that are isomorphic to each other. We observed an interesting side-effect of the partial path and random number seed output mentioned above.

For some graphs the sequences of path lengths and random number seeds were identical. Subsequent verification showed that the graphs were, in fact, isomorphic. In the next section we will describe our approach to generating sets of three involutions so that the number of isomorphic graphs is significantly reduced.

6.6 Generating Graphs with Three Involutions

As noted in Section 6.5, some of the graphs we investigated were subsequently found to be isomorphic. If we consider the symmetric group S_7 with $7!$, or 5,040, vertices, we find that it has 231 involutions. There are therefore $\binom{231}{3}$, or 2,027,795 possible sets of three involutions in S_7 , many of which generate a proper subgroup. The tables of Effler and Ruskey contain 32,755 sets of three involutions that generate the whole group, S_7 .

We investigated the problem of determining sets of three involutions that generated as few isomorphic Cayley graphs as possible. We were successful in devising an algorithm for creating such generating sets that was both time and space efficient, reducing the 2,027,795 possible sets of three involutions in S_7 down to just 676 sets, of which only 254 generate the whole group, S_7 .

If g, h are elements of a group G then the element $h^{-1}gh$ is a *conjugate* of g . Similarly, for a set $X \subseteq G$ we define the *conjugate set* $h^{-1}Xh = \{h^{-1}xh : x \in X\}$. In Lemma 5 we show that conjugate generating sets generate isomorphic groups. In Lemma 6 we show that the group isomorphism also induces an isomorphism in the corresponding Cayley graphs.

Lemma 5 *Let X be a generating set for a group G . For $h \in G$ let G' be the group generated by $h^{-1}Xh$. Then G' is isomorphic to G .*

Proof. Let ϕ be the mapping from G to G' defined by $\phi(g) = h^{-1}gh$. Let $a = x_1x_2 \dots x_m \in G$; $x_i \in X$ and let $b = y_1y_2 \dots y_n \in G$; $y_i \in X$ Then

$$\phi(ab) = h^{-1}(ab)h = h^{-1}x_1hh^{-1}x_2h \dots h^{-1}x_mhh^{-1}y_1hh^{-1}y_2h \dots h^{-1}y_nh = \phi(a)\phi(b)$$

so ϕ is a homomorphism. If $\phi(a) = \phi(b)$ then $h^{-1}ah = h^{-1}bh$ and so $a = b$ and ϕ is one-to-one. \square

Lemma 6 *Let X be a generating set for a group G . For $h \in G$ and let G' be the group generated by $h^{-1}Xh$. Then $\text{Cay}(G : X)$ and $\text{Cay}(G' : h^{-1}Xh)$ are isomorphic.*

Proof. Suppose $g \in G$ and $x \in X$. The mapping ϕ of Lemma 5 is a bijection that preserves adjacency for $\phi(gx) = \phi(g)\phi(x) = \phi(g)h^{-1}xh$. \square

Conjugacy defines an equivalence relation on the elements of G and the *conjugacy class* $cl(g)$ of an element $g \in G$ is the set $\{h^{-1}gh : h \in G\}$. Evidently, conjugacy also defines an equivalence relation on subsets $X \subseteq G$. In an analogous way we define the *conjugacy class* $cl(X)$ of a set $X \subseteq G$ as the set $\{h^{-1}Xh : h \in G\}$.

Lemma 6 shows that all Cayley graphs generated by conjugate sets of generators are isomorphic. To reduce the number of isomorphic generating sets our strategy will be choose a canonical representative from each conjugacy class, $cl(X)$ of three involutions. We now use some more group theory to help us find this canonical generating set. The necessary group theoretic results may be found in a standard algebra text such as Herstein [58].

Every permutation in S_n can be written as a product of disjoint cycles. If a permutation $\sigma \in S_n$ can be written as the product of disjoint cycles of lengths n_1, n_2, \dots, n_r with $n_1 \leq n_2 \leq \dots \leq n_r$ we say that σ has a *cycle decomposition* $\{n_1, n_2, \dots, n_r\}$. Evidently, n_1, n_2, \dots, n_r is a partition of n . The cycle decomposition for a permutation is unique and two permutations in S_n are conjugate if and only if they have the same cycle decomposition.

An involution in S_n is the product of disjoint transpositions and so it has a cycle decompositions of the form $\{1, \dots, 1, 2, \dots, 2\}$. If the multiplicity of 2 in the cycle decomposition of $\sigma \in S_n$ is k , where $1 \leq k \leq \lfloor \frac{n}{2} \rfloor$, then we will denote $cl(\sigma)$ by cl_k .

The size of cl_k is given by

$$|cl_k| = \frac{\prod_{i=1}^k \binom{n+2-2i}{2}}{k!} \quad (6.1)$$

Summing over the conjugacy classes, the total number of involutions in S_n is therefore $\sum_{k=1}^{\lfloor \frac{n}{2} \rfloor} |cl_k|$.

Let $I_n = \{\tau_1, \tau_2, \dots, \tau_{|I_n|}\}$ be the set of involutions in S_n . We order the elements of I_n such that if $\tau_i \in cl_k$ and $\tau_j \in cl_l$ where $k < l$ then $i < j$. For any $g = \tau_i \in I_n$ we define $\text{rank}(g) = i$. For each conjugacy class cl_k , let the involution τ_k of lowest rank in cl_k be the representative of that class.

Given this ordering, we can sort any generating set, $\{g_1, g_2, g_3\}$, of three distinct involutions such that $\text{rank}(g_1) < \text{rank}(g_2) < \text{rank}(g_3)$. This induces a lexicographic ordering of sorted generating sets in the obvious way. For any generating set, X , of three involutions, we will use the lexically least generating set from $cl(X)$ as our canonical representative of $cl(X)$. For individual generators $g \in I_n$ we will use the generator g' as the representative of $cl(g)$ where g' is chosen so that $\text{rank}(g') \leq \text{rank}(h)$ for all $h \in cl(g)$.

To help us find these canonical generating sets we use some more group theory.

The *centralizer*, $C_G(g)$, of an element $g \in G$ is defined as $C_G(g) = \{h \in G \mid h^{-1}gh = g\}$. The set $C_G(g)$ is a subgroup of G which consists of all the elements of G that commute with g .

The size of the conjugacy class of an element and the size of its centralizer are related by the equation

$$|cl(g)| = \frac{|G|}{|C_G(g)|}. \quad (6.2)$$

The well-known class equation for G is the sum

$$|G| = \sum \frac{|G|}{|C_G(g)|} \quad (6.3)$$

where the sum runs over one representative g from each conjugacy class.

If $g \in cl_k \subseteq S_n$ then $C_G(g) = n!/|cl_k|$, so a large conjugacy class will have a small centralizer and vice-versa.

For each $g \in G$, we define a relation $\overset{g}{\sim}$ on the elements of G such that $a \overset{g}{\sim} b$ if $a = h^{-1}bh$ for some $h \in C_G(g)$. Since $C_G(g)$ is a group we have the following:

1. If e is the identity element of G , then $a = e^{-1}ae$ and so $a \overset{g}{\sim} a$.
2. If $a = h^{-1}bh$ then $b = h^{-1}ah = (h^{-1})^{-1}bh^{-1}$ and $a \overset{g}{\sim} b$.
3. If $a = h^{-1}bh$ and $b = k^{-1}ck$ then $a = h^{-1}(k^{-1}ck)h = (kh)^{-1}(ckh)$ and $a \overset{g}{\sim} c$.

Hence, $\overset{g}{\sim}$ is an equivalence relation.

We are now in a position to describe our strategy for finding canonical generating sets of three involutions.

For any set of involutions $X = \{g_1, g_2, g_3\} \subset S_n$ there is an $h \in S_n$ where the conjugate generating set $h^{-1}Xh = \{g'_1, g'_2, g'_3\}$ has g'_1 as the canonical representative of $cl(g)$. Therefore, we choose our first generator g_1 from the set, $\{r_k : 1 \leq k \leq \lfloor n/2 \rfloor\}$, of canonical representatives of the conjugacy classes of involutions.

Given that we can always conjugate a generating set $X = \{g_1, g_2, g_3\} \subset S_n$ so that g_1 is a class representative and given that g_1 is invariant under conjugation by elements of $C_G(g_1)$, we need only consider a second generator g_2 that is chosen as a canonical representative of the equivalence class defined by the $\overset{g_1}{\sim}$ relation described above. Again, we use the least such generator as our canonical representative.

We then choose our third generator g_3 and check that the resulting set $\{g_1, g_2, g_3\}$ is lexically least among its conjugate sets. If g_1 is not in the same conjugacy class as g_2 , then g_1 must remain invariant under the conjugation, so we need check only those generating sets conjugated by elements of $C_G(g_1)$.

The pseudo-code for our algorithm to construct all non-conjugate triples, $\{g_1, g_2, g_3 : g_i \in I_n\}$ is shown in Figure 6.4. The algorithm consists of the following steps.

1. Create the set X_1 of class representatives $\{r_k : 1 \leq k \leq \lfloor n/2 \rfloor\}$.
2. While X_1 is not empty.
 - (a) Select g_1 of minimum rank from X_1 and remove g_1 from X_1 .
 - (b) Compute the centralizer subgroup $C_{S_n}(g_1)$ of g_1 .
 - (c) Create the set of involutions $X_2 = \{i \in I_n : \text{rank}(i) > \text{rank}(g_1)\}$.
 - (d) While X_2 is not empty.
 - i. Select g_2 of minimum rank from X_2 .
 - ii. For each element $c \in C(g_1)$ compute $c^{-1}g_2c$. If $\text{rank}(c^{-1}g_2c) > \text{rank}(g_2)$ then remove the element $c^{-1}g_2c$ from X_2 so that it will not be selected in a future iteration of Step 2(d)i.
 - iii. Create the set of involutions $X_3 = \{i \in I_n : \text{rank}(i) > \text{rank}(g_2)\}$.
 - iv. While X_3 is not empty.
 - A. Select g_3 of minimum rank from X_3 .
 - B. If the conjugacy class of g_1 is the same as that of g_2 then for each $h \in G$ compute the conjugates $h^{-1}g_1h$, $h^{-1}g_2h$ and $h^{-1}g_3h$ and sort this to form a new generating set $\{a_1, a_2, a_3\}$. If the conjugacy class of g_1 is not the same as that of g_2 (and therefore also that of g_3), do this for each $h \in C(g_1)$ instead of for each $h \in G$.
 - C. For each set $\{a_1, a_2, a_3\}$ computed in Step 2(d)ivB, if $g_1 = a_1$ and $\text{rank}(a_2) < \text{rank}(g_2)$, then remove g_3 from X_3 so we will not output this generating set. Otherwise, if $g_1 = a_1$ and $g_2 = a_2$ and $\text{rank}(a_3) > \text{rank}(g_3)$, then remove a_3 from X_3 so that it will not be selected as a third generator in a future iteration of Step 2(d)ivA.
 - D. If g_3 is still in X_3 then output the generating set $\{g_1, g_2, g_3\}$.

We implemented two additional optimizations which reduced running time by approximately 40%, at the expense of extra memory requirement. The first was to compute and store all the conjugates of the element g_1 selected in Step 2a and necessary conjugates of the element g_2 in Step 2(d)i. The second was to cut short the computation of conjugates sets in Step 2(d)ivB as soon as it became clear that the set could not pass either of the tests in Step 2(d)ivC

There are only $\lfloor \frac{n}{2} \rfloor$ choices of an involution for g_1 . Furthermore, if the conjugacy class size for g_1 (Equation 6.1) is small then the centralizer of g_1 is large, so we might expect large classes of conjugates under the equivalence relation $\overset{g_1}{\sim}$.

We illustrate just how effective this algorithm is in reducing the number of non-isomorphic groups generated by three involutions by considering S_7 . Recall that S_7 with $7!$, or 5,040, vertices, has 231 involutions and $\binom{231}{3}$, or 2,027,795 possible sets of three involutions. The involutions fall into one of $\lfloor 7/2 \rfloor = 3$ equivalence classes so we have only three choices for g_1 . When g_1 is from the equivalence class of permutations consisting of a single transposition, the relation $\overset{g_1}{\sim}$ reduces the number of possible values for g_2 to just 9. The corresponding number of possible values for g_2 , when g_1 consists of two or three transpositions, are 14 and 6 respectively. The size of the class of single transpositions is 21, while the other two each have 105 elements. Taking into account the sorting of our generating sets, we have now reduced the possible number of generating sets to

$$(9 \times 230) + (14 \times 209) + (6 \times 104) = 5620$$

sets to be checked for conjugates rather than 2,027,795 sets. Checking these 5,620 sets for conjugates reduces the actual number of non-conjugate generating sets in S_7 to just 676 sets. Table 6.2 shows the number of involutions and the number of non-conjugate generating sets for S_4 through S_{10} , along with the sizes of the equivalence classes and sizes of the centralizer groups for elements of each equivalence class.

Curtin [28] also used conjugate generating sets while studying the (Δ, D) problem which seeks the largest value n , such that a graph on n vertices with degree Δ and diameter D exists. He derives a generating function for the number of generating sets of particular types and the number of sets found by our algorithm agrees with this theoretical computation.

The running time on a 2.4GHz Pentium 4 system with 512MB of RAM was one second or less for all groups up to S_7 . For S_8 it was 12 seconds, for S_9 approximately six minutes, and for S_{10} approximately

```

ThreeInvGenSets(n)
G ← Sn
In ← {involutions of G}
X1 ← {class representatives of each i ∈ I}
while X1 not empty
  select g1 from X1 where rank(g1) is minimal
  X1 = X1 \ {g1}
  C ← {c ∈ G : cg1 = g1c} ▷ Centralizer of g1
  X2 ← {i : i ∈ I, rank(i) > rank(g1)}
  while X2 not empty
    select g2 from X2 where rank(g2) is minimal
    X2 = X2 \ {g2}
    ▷ Remove g1 equivalent involutions
    for each c ∈ C
      if c-1g2c ≠ g2
        then X2 ← X2 \ ({g2} ∩ X2)
    X3 ← {i : i ∈ I, rank(i) > rank(g2)}
    while X3 not empty select g3 from X3 where rank(g3) is minimal
    if cl(g1) = cl(g2)
      then H ← G
      else H ← C
    for each h ∈ H
      a1 ← h-1g1h
      a2 ← h-1g2h
      a3 ← h-1g3h
      SortGenerators(a1, a2, a3)
      if a1 = g1
        then
          if rank(a2) < rank(g2)
            then X3 = X3 \ {g3}
          else
            if a2 = g2
              then if a3 ≠ g3
                then X3 = X3 \ ({a3} ∩ X3)
    if g3 ∈ X3
      then output({g1, g2, g3})
      X3 = X3 \ {g3}

```

Figure 6.4: Finding sets of three involutions in S_n

4.5 hours. The program ran entirely in memory, without swapping, requiring approximately 100MB of memory. CPU utilization stayed around 99%. As we first saw with the middle two levels problem (Chapter 4), Memory swapping can seriously degrade performance on such large problems. We attempted to run the program for S_{11} and the system indicated that 1GB of RAM, or perhaps slightly more, would be required to run without swapping. Running in a 512MB system caused severe swapping and the system became very sluggish, even for such tasks as moving the cursor. The program itself was not even using 10% of the CPU, because the system was using so much to handle the memory swapping.

For each set of generators, our algorithm also used a breadth first search to determine several graph characteristics, including number of vertices, diameter and shortest odd and even cycles. The running time for this version on the same 2.4GHz Pentium 4 system was three seconds or less for all groups up to S_7 . For S_8 it was approximately one minutes, for S_9 approximately one hour, and for S_{10} approximately 54 hours.

We have thus reduced the number of Cayley graphs to be considered by removing groups that have conjugate sets of generators. However, we still have some graphs that are isomorphic as can readily seen in S_4 . Consider the two sets of generators X_1 and X_2 where

$$X_1 = \{(1, 2), (3, 4), (1, 2)(3, 4)\}$$

and

$$X_2 = \{(1, 2)(3, 4), (1, 3)(2, 4), (1, 4)(2, 3)\}.$$

Each of these sets generates an abelian group of order 4 where each generator is the product of the other two. The Cayley graphs are both isomorphic to K_4 , the complete graph on 4 vertices. The group generated by X_2 is a subgroup of the alternating group A_4 since the generators are all even permutations, but the group generated by X_1 is not a subgroup of A_4 , so the generators cannot be conjugate sets, even though the generated groups are isomorphic.

Furthermore, the sets of three involutions from S_n include sets that are isomorphic to the sets of three involutions from S_{n-1} .

Table 6.2: Involution and conjugacy statistics for permutation groups S_n .

Size, number of involutions, conjugacy class sizes and centralizer sizes for selected permutation groups.

	Group						
	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
$ G! = n $	24	120	720	5,040	40,320	362,880	3,628,800
Involutions	9	25	75	231	763	2619	9495
Generating Sets	10	37	197	676	3094	12022	55912
Class 1 sizes							
Class	6	10	15	21	28	36	45
Centralizer	4	12	48	240	1440	10080	80640
Class 2 sizes							
Class	3	15	45	105	210	378	630
Centralizer	8	8	16	48	192	960	5760
Class 3 sizes							
Class			15	105	420	1260	3150
Centralizer			48	48	96	288	1152
Class 4 sizes							
Class					105	384	4725
Centralizer					945	384	768
Class 5 sizes							
Class							945
Centralizer							3840

6.7 Further Results for Cubic Cayley Graphs

The cycle extension and random backup capabilities described in Section 3.4 of Chapter 3 were developed as we sought to find cycles in Cayley graphs generated by the sets of three involutions found by the algorithm of Section 6.6. We were successful in finding Hamilton cycles in all of these graphs with generators from S_n where $n \leq 7$.

As with our earlier results, it seemed harder to find cycles in some graphs than other graphs. We did three runs with the 5,040 vertex graphs from S_7 , using a different initial random seed for the runs in each set. From the results, we selected the graphs that took 50 or more attempts in any of these runs and these are shown in Table 6.3. Each attempt at a graph now included up to 400 random backup steps. We have not yet re-instrumented our code to reflect this increase, so these attempts are not directly comparable with the attempts in Table 6.1.

Although our data is not yet extensive, the hardest graphs appear to be those with a minimum even cycle of length 6 followed by those with a minimum even cycle of length 8. Of the 254 graphs with 5040 vertices generated by three involutions from S_7 , 47 have a minimum even cycle of 6 and 80 have a minimum even cycle of length 8. There are 35 with a minimum even cycle of length 4 and the remainder have minimum even cycles of lengths 10, 12 or 14.

If we consider the prime factors of $8!$, we might expect to find graphs with 10,080 vertices generated by three involutions from S_8 . In fact, there are no such graphs. The smallest subgraph of S_8 larger than S_7 and generated by three involutions from S_8 , is the alternating group, A_8 , with 20,160 vertices.

We also attempted to find cycles in the 21 graphs from S_8 having 20,160 vertices. The groups for these are, in fact, the alternating group, A_8 . We were unsuccessful in finding Hamilton cycles in three of these graphs as shown in Table 6.4, although we did succeed with the other 18.

All of these graphs have a minimum even cycle of length 6, and these are the only graphs on A_8 generated by our sets that have a minimum even cycle of length 6. Given that graph 931 and graph 2159 from the sets from S_8 have the same diameter and minimum odd and even cycle lengths, it is possible that they are isomorphic, although neither is isomorphic to graph 928 as the diameters differ.

Note also that most of the hard graphs in Table 6.3 also have a minimum even cycle of length 6.

We conjecture that the problem with our heuristic on graphs with a minimum even cycle of length 6 is that the heuristic may be generating a partial path containing many pairs of edges that are opposites on a cycle of length 6 as illustrated in Figure 6.5. If the path includes edges ab , bc , cd , gh , hi , and ij ,

Table 6.3: Characteristics of hard Cayley graphs with 5,040 vertices.

All graphs are generated by three involutions from S_7 .

Our Graph Number	Generators	Shortest Odd Cycle	Shortest Even Cycle	Diameter	Number of Attempts		
					Run 1	Run 2	Run 3
60	(1)(2)(3)(4)(5)(67) (1)(2)(3)(46)(57) (15)(26)(37)(4)	21	8	20	164	187	139
183	(1)(2)(3)(4)(5)(67) (1)(25)(36)(47) (13)(27)(4)(56)	-	8	21	171	127	50
183	(1)(2)(3)(46)(57) (1)(24)(37)(5)(6) (14)(27)(36)(5)	17	6	20	329	174	498
396	(1)(2)(3)(46)(57) (1)(24)(37)(5)(6) (15)(27)(36)(4)	21	6	16	120	80	119
404	(1)(2)(3)(46)(57) (1)(24)(37)(5)(6) (15)(27)(3)(46)	15	6	19	78	50	50
407	(1)(2)(3)(46)(57) (1)(24)(37)(5)(6) (13)(27)(45)(6)	25	6	19	53	138	108
627	(1)(25)(36)(47) (1)(24)(37)(56) (14)(25)(36)(7)	-	6	20	341	945	272
635	(1)(25)(36)(47) (1)(24)(37)(56) (14)(23)(57)(6)	-	6	18	50	194	50
637	(1)(25)(36)(47) (15)(26)(37)(4) (14)(25)(36)(7)	-	6	17	99	333	106

Table 6.4: Characteristics of hard Cayley graphs with 20,160 vertices.

Our algorithm did not find a cycle in these three graphs with 20,160 vertices generated by three involutions from A_8 .

Our Graph Number	Generators	Diameter	Shortest Odd Cycle	Shortest Even Cycle
928	(1)(2)(3)(4)(57)(68) (1)(2)(35)(48)(6)(7) (14)(26)(37)(58)	20	21	6
931	(1)(2)(3)(4)(57)(68) (1)(2)(35)(48)(6)(7) (14)(28)(35)(67)	21	21	6
2159	(1)(2)(3)(4)(57)(68) (14)(25)(38)(67) (13)(28)(45)(67)	21	21	6

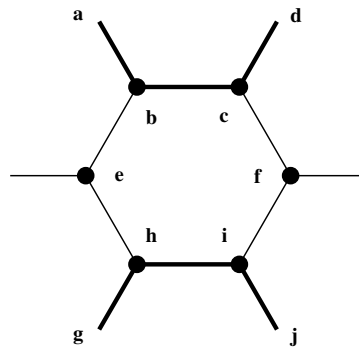


Figure 6.5: Cycle of length 6 in a cubic graph.

then the vertices e and f cannot be added to the path unless one of their neighbors becomes the new path end point by a series of rotations.

We also conjecture that this problem may occur in other rotation-extension heuristics. Further study would be required to confirm this conjecture.

Chapter 7

The Cover Graph of $M(n)$

7.1 Introduction

Introduced by Stanley [103], the poset $M(n)$ has as its elements the n -tuples of integers $\mathbf{a} = (a_1, a_2, \dots, a_n)$ satisfying $0 = a_1 = \dots = a_j < a_{j+1} < \dots < a_n \leq n$ for some $j \geq 0$. The order relation is defined by $\mathbf{a} \leq \mathbf{b}$ if and only if $a_i \leq b_i$ for $1 \leq i \leq n$. The poset $M(n)$ has a number of order-related properties of interest in graph theory and combinatorics. A discussion of order properties of posets appears in [104]; a nice description of $M(n)$ in particular appears in [85].

In contrast to the order-theoretic properties of $M(n)$, in this chapter we consider graph-theoretic properties of its Hasse diagram. When viewed as an undirected graph, the Hasse diagram is called the *cover graph* of the poset.

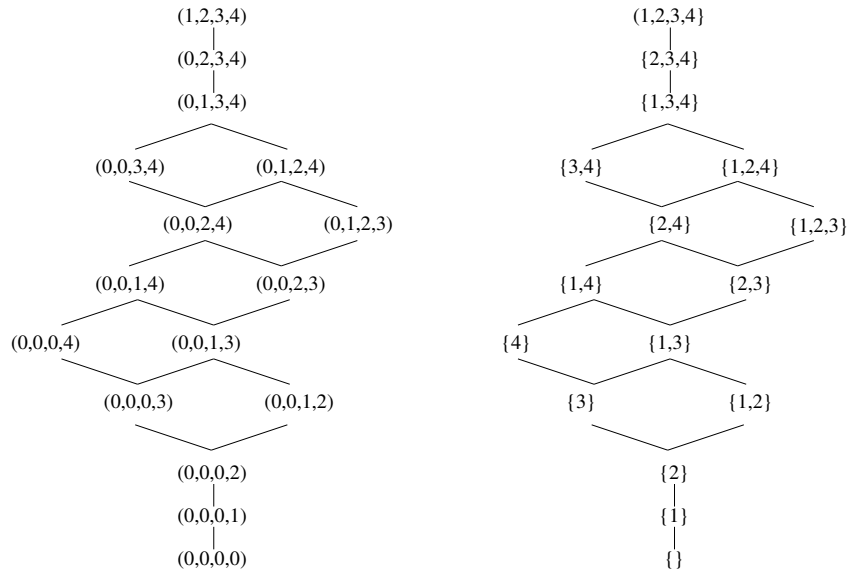
Let A_n be the cover graph of $M(n)$. For compactness, we will represent the vertices of A_n as subsets of $[n] = \{1, 2, \dots, n\}$. We write element $\mathbf{a} \in M(n)$ satisfying $0 = a_1 = \dots = a_j < a_{j+1} < \dots < a_n \leq n$ as the vertex $\{a_{j+1}, \dots, a_n\}$ of A_n , as in Figure 7.1.

For sets X, Y , let $X \oplus Y = (X \cup Y) \setminus (X \cap Y)$. In A_n vertices X and Y are adjacent if and only if

- (i) $|X| = |Y|$ and $X \oplus Y = \{i, i + 1\}$ for some i or
- (ii) $X \oplus Y = \{1\}$.

For a finite set S , let $\sigma(S)$ denote the sum of the elements of S . Note that if X and Y are adjacent in A_n then $\sigma(X)$ and $\sigma(Y)$ differ by 1. Thus A_n is bipartite with bipartition (E_n, O_n) , where

$$E_n = \{X \in 2^{[n]} : \sigma(X) \text{ is even}\}$$

Figure 7.1: Two views of the cover graph of $M(4)$.

$$O_n = \{X \in 2^{[n]} : \sigma(X) \text{ is odd}\}$$

Savage and West conjectured in May 1991 that A_n has a Hamilton path whenever $n \geq 1$, and $\binom{n+1}{2}$ is odd and $n \neq 5$. They had determined that A_5 does not have a Hamilton path and that A_n does not have a Hamilton path for $n \geq 1$ when $\binom{n+1}{2}$ is even. The edge results of Lemma 7 and Lemma 8 are also due to Savage and West. At that time, the only non-trivial result for $\binom{n+1}{2}$ odd was that the graph has a Hamilton path for $n = 6$. No further progress was made for several years.

For comparison and motivation, we mention a more familiar partial order on the same set: the inclusion relation. This defines a poset on the subsets of $[n]$, called the *Boolean lattice* $B(n)$. Its cover graph is the n -cube. The n -cube is Hamiltonian and has Hamilton paths satisfying a wide variety of constraints (for example, [7, 50, 53, 51, 94]). It is easy to obtain a Hamilton path in $B(n)$ by induction. The cover graph of $M(n)$ has a somewhat more complicated structure than that of $B(n)$ and fewer edges: $(n+1)2^{n-2}$ instead of $n2^{n-1}$.

In 2000, we used our algorithm to find Hamilton paths for the cases $n = 9, 10, 13, 14, 17$ and 18 . This knowledge inspired work on the problem by Savage, Shields and West [93] leading to a proof that the cover graph of $M(n)$ has a Hamilton path if and only if $\binom{n+1}{2}$ is odd and $n \neq 5$.

In the remainder of this chapter, we outline the proof of the following theorem and describe the role of our algorithm in developing the proof.

Theorem 24 *For $n \geq 1$, if $\binom{n+1}{2}$ is odd and $n \neq 5$, then A_n has a Hamilton path.*

The proof is a constructive proof by induction. Once a likely construction was identified, our algorithm was employed for the cases $n = 5, 6$ to find the paths satisfying more stringent conditions, that would become the basis cases for the induction. We show the proofs of the necessary conditions in Section 7.2. In Section 7.3 we show the construction technique used and describe our contributions to the series of lemmas leading to the final result. For the complete proof refer to [93].

7.2 Necessary Conditions

Proposition 1 *(Savage and West 1991). For $n \geq 1$, if $\binom{n+1}{2}$ is even, then A_n does not have a Hamilton path.*

Proof. Always $|E_n| = |O_n|$, so a Hamilton path in A_n must have one endpoint in E_n and the other in O_n . On the other hand, A_n has two vertices of degree 1, namely \emptyset and $[n]$, which therefore must be the

endpoints of any Hamilton path. If $\sigma([n]) = \binom{n+1}{2}$ is even, then the vertices that must be the endpoints both lie in E_n . \square

When $n = 5$, $\binom{n+1}{2}$ is odd, but A_5 does not have a Hamilton path. To see this, note in Figure 7.2 that vertices of degree 2 in A_n successively force a Hamilton path in A_5 to use the edges on the disjoint paths (shown in boldface on the figure):

$$\emptyset, \{1\}, \{2\}, \{1, 2\}, \{1, 3\}, \{3\}, \{4\}, \{5\}, \{1, 5\}, \{1, 4\}, \{2, 4\}, \{2, 3\}, \{1, 2, 3\}, \{1, 2, 4\}$$

and

$$\{1, 2, 3, 4, 5\}, \{2, 3, 4, 5\}, \{1, 3, 4, 5\}, \{3, 4, 5\}, \{2, 4, 5\}, \{1, 2, 4, 5\}, \{1, 2, 3, 5\},$$

$$\{1, 2, 3, 4\}, \{2, 3, 4\}, \{2, 3, 5\}, \{1, 3, 5\}, \{1, 4, 5\}, \{4, 5\}, \{3, 5\}.$$

However, the subgraph of A_n induced by $\{1, 2, 4\}, \{3, 5\}, \{2, 5\}, \{3, 4\}, \{1, 2, 5\}$, and $\{1, 3, 4\}$ is a 6-cycle in which $\{1, 2, 4\}$ and $\{3, 5\}$ are diametrically opposite. Hence the path cannot be completed.

7.3 Construction of Paths

For $n \geq 3$, let G_n be the subgraph of A_n induced by the vertices

$$V(G_n) = V(A_n) \setminus \{\emptyset, \{1\}, \{2, 3, \dots, n\}, [n]\}.$$

Note that A_n has a Hamilton path if and only if G_n has a Hamilton path from $\{2\}$ to $\{1, 3, \dots, n\}$. We will prove via a sequence of lemmas that such a path exists in G_n whenever $\binom{n+1}{2}$ is odd and $n \neq 5$. Our strategy is to build up paths and cycles in subgraphs of G_n that we will combine to form the desired Hamilton path.

For $n \geq 3$, let e_n and a_n denote the edges in G_n :

$$e_n = \{1, 2, \dots, n-1\}\{2, \dots, n-1\}$$

$$a_n = \{n\}\{1, n\}.$$

(We use uv to denote the edge joining adjacent vertices u and v .)

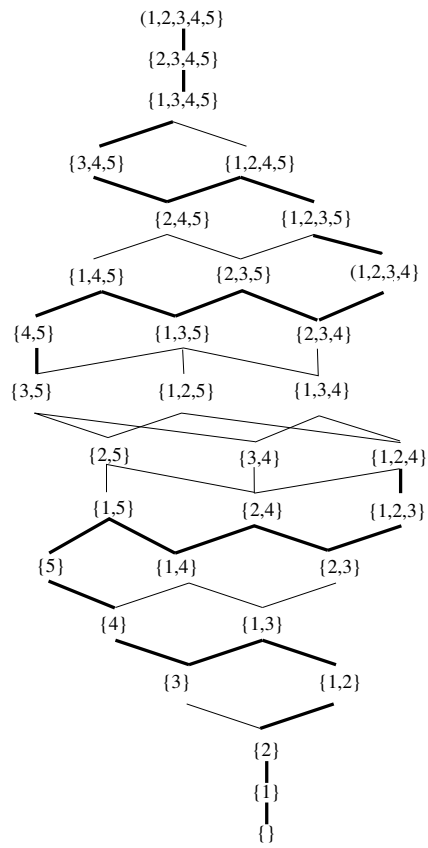


Figure 7.2: A_5 , the cover graph of $M(5)$.

Lemma 7 (Savage and West 1991). For $n \geq 3$, any Hamilton cycle in G_n contains the following edges:

- (a) $\{2\}\{3\}$
- (b) $\{2\}\{1, 2\}$
- (c) $\{1, 2\}\{1, 3\}$
- (d) $\{n\}\{1, n\} = a_n$
- (e) $[n - 1]\{2, 3, \dots, n - 1\} = e_n$.

Furthermore, for each edge uv in (a)-(e) any Hamilton path in G_n that does not start or end at u or v contains the edge uv .

Proof. Each of the vertices $\{1, 2, \dots, n - 1\}$, $\{n\}$, $\{2\}$, and $\{1, 2\}$ has degree 2 in G_n . \square

When H is a subgraph of G_{n-1} , let nH denote the subgraph of G_n obtained from the graph H by replacing each vertex X of H by the vertex $X \cup \{n\}$. For an edge $e = XY$, let $V(e)$ denote the set $\{X, Y\}$. Then the vertex set of G_n is the disjoint union

$$V(G_n) = V(G_{n-1}) \cup V(nG_{n-1}) \cup V(e_n) \cup V(a_n). \quad (7.1)$$

Let P be a path in G_n containing edge uv , and let $e = wz$ be an edge of G_n with neither w nor z on P . If uw and vz are also edges of G_n , then by “pulling e into P ” we will mean replacing the edge uv of P by the path u, w, z, v to obtain a new path P' with $V(P') = V(P) \cup V(e)$. We will apply this technique to combine cycles and paths into larger paths.

Note that if P and Q are Hamilton paths in G_{n-1} , then by Lemma 7 we can pull a_n into P as long as P does not start or end with $\{n - 1\}$ or $\{1, n - 1\}$ and we can pull e_n into nQ as long as Q does not start or end with $[n - 2]$ or $\{2, 3, \dots, n - 2\}$.

Lemma 8 (Savage and West 1991). For $n \geq 3$, G_n has a Hamilton cycle C_n . For $n \geq 4$, there is such a cycle containing the edge $\{2, n\}\{3, n\}$ and such a cycle containing the edge $\{1, 2, n\}\{1, 3, n\}$.

Proof. For $n = 3$, G_3 itself is the cycle C_3 (of length 4). For $n = 4$, the cycle C_4 below contains both special edges.

$$\{2\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}, \{1, 2, 4\}, \{1, 3, 4\}, \{3, 4\}, \{2, 4\}, \{1, 4\}, \{4\}, \{3\}$$

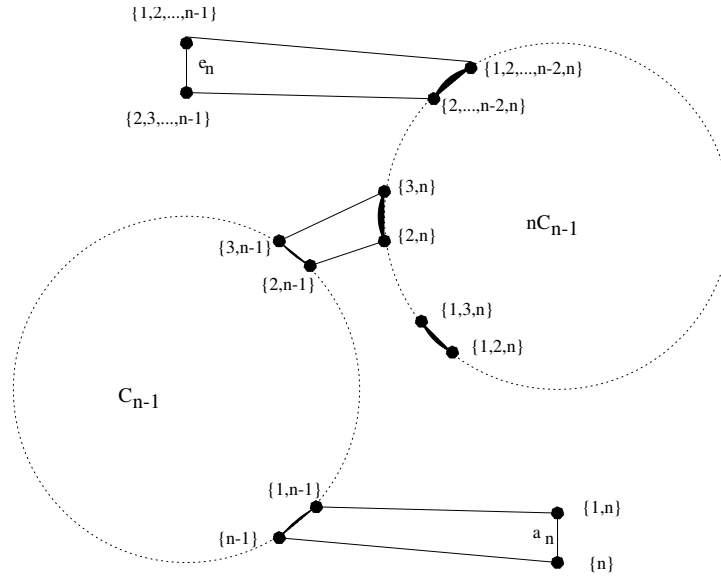


Figure 7.3: Construction of C_n in Lemma 8 when C_{n-1} contains $\{2, n-1\}\{3, n-1\}$.

Assume that $n > 4$, and let C_{n-1} be a Hamilton cycle in G_{n-1} satisfying one of the claimed conditions of the theorem. We first pull edge e_n into nC_{n-1} and edge a_n into C_{n-1} and then link these two cycles together to form C_n as described below.

If C_{n-1} contains edge $\{2, n-1\}\{3, n-1\}$, then delete this edge from C_{n-1} and delete edge $\{2, n\}\{3, n\}$ from nC_{n-1} (guaranteed to exist by Lemma 7). Add edges $\{2, n-1\}\{2, n\}$ and $\{3, n-1\}\{3, n\}$. (See Figure 7.3. In this and other figures, vertices need not occur on cycles and paths in the order shown.) The resulting cycle contains edge $\{1, 2, n\}\{1, 3, n\}$, since Lemma 7 implies that C_{n-1} contains edge $\{1, 2\}\{1, 3\}$.

In the other case, C_{n-1} contains the edge $\{1, 2, n-1\}\{1, 3, n-1\}$. A similar argument to the above yields a cycle containing edge $\{2, n\}\{3, n\}$. (See Figure 7.4.) \square

Lemma 9 For $n \geq 5$, G_n has a Hamilton path Q_n from $\{2\}$ to $\{2, n\}$ if n is odd and from $\{2\}$ to $\{1, 2, n\}$ if n is even.

Proof. For $n = 5$, we used our algorithm to find the path Q_5 satisfying the requirements is shown in Figure 7.5 (compare with Figure 7.2).

For the inductive construction we combine a path Q_{n-1} in G_{n-1} with the Hamilton cycle nC_{n-1} in nG_{n-1} of Lemma 8 and pull in the edges e_n and a_n to construct the required path Q_n . The constructions for n even and n odd are shown in Figure 7.6. \square

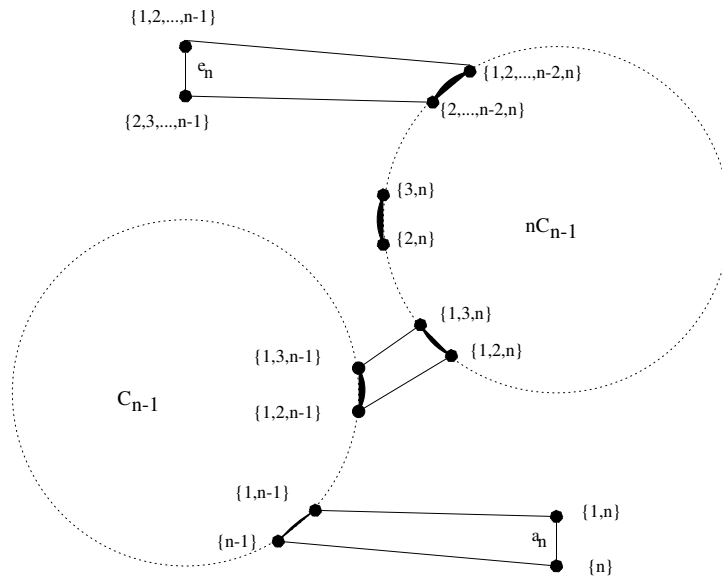


Figure 7.4: Construction of C_n in Lemma 8 when C_{n-1} contains $\{1, 2, n - 1\}\{1, 3, n - 1\}$.

$Q_5 =$

$\{2\}$,	$\{1, 2\}$,	$\{1, 3\}$,	$\{3\}$,	$\{4\}$,	$\{5\}$,	$\{1, 5\}$,
$\{1, 4\}$,	$\{2, 4\}$,	$\{2, 3\}$,	$\{1, 2, 3\}$,	$\{1, 2, 4\}$,	$\{1, 3, 4\}$,	$\{3, 4\}$,
$\{3, 5\}$,	$\{4, 5\}$,	$\{1, 4, 5\}$,	$\{2, 4, 5\}$,	$\{3, 4, 5\}$,	$\{1, 3, 4, 5\}$,	$\{1, 2, 4, 5\}$,
$\{1, 2, 3, 5\}$,	$\{1, 2, 3, 4\}$,	$\{2, 3, 4\}$,	$\{2, 3, 5\}$,	$\{1, 3, 5\}$,	$\{1, 2, 5\}$,	$\{2, 5\}$

Figure 7.5: The Hamilton path Q_5 in G_5 (read across) in Lemma 9.

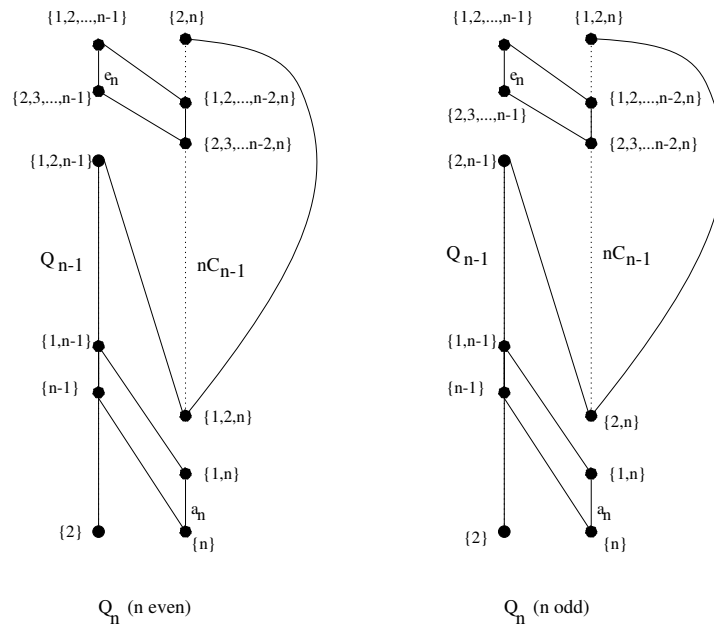


Figure 7.6: Construction of Q_n in Lemma 9.

$Q_5^* =$

{1, 2},	{2},	{3},	{4},	{5},	{1, 5},	{1, 4},
{1, 3},	{2, 3},	{1, 2, 3},	{1, 2, 4},	{1, 3, 4},	{3, 4},	{2, 4},
{2, 5},	{3, 5},	{4, 5},	{1, 4, 5},	{2, 4, 5},	{3, 4, 5},	{1, 3, 4, 5},
{1, 2, 4, 5},	{1, 2, 3, 5},	{1, 2, 3, 4},	{2, 3, 4},	{2, 3, 5},	{1, 3, 5},	{1, 2, 5}

Figure 7.7: The Hamilton path Q_5^* in G_5 (read left to right) in Lemma 10.

Lemma 10 For $n \geq 5$, G_n has a Hamilton path Q_n^* from $\{1, 2\}$ to $\{1, 2, n\}$ if n is odd and from $\{1, 2\}$ to $\{2, n\}$ if n is even.

Proof. For $n = 5$, we used our algorithm to find a path, Q_5^* , satisfying the requirements as shown in Figure 7.7 (compare with Figure 7.2). For $n > 5$, Q_n^* is constructed from Q_{n-1}^* and nC_{n-1} exactly as Q_n is constructed from Q_{n-1} and nC_{n-1} in the proof of Lemma 9. \square

Lemma 11 For $n \geq 6$, G_n has a Hamilton path P_n from $\{2\}$ to $\{1, 3, 4, \dots, n\}$ if $\binom{n+1}{2}$ is odd and from $\{2\}$ to $\{3, 4, \dots, n\}$ if $\binom{n+1}{2}$ is even. In addition, when $n \geq 6$ and $\binom{n+1}{2}$ is odd, G_n has a Hamilton path R_n from $\{1, 2\}$ to $\{3, 4, \dots, n\}$.

Proof. For the basis case $n = 6$, we again used our algorithm to find paths P_6 and R_6 satisfying the requirements as shown in Figure 7.8. Assume $n > 6$. We consider four cases, according to whether $\binom{n+1}{2}$ and n are, independently, odd or even.

Case 1. If $\binom{n+1}{2}$ is even and n is odd, then $\binom{(n-1)+1}{2}$ is odd. We construct P_n by pulling edge e_n into nR_{n-1} and edge a_n into Q_{n-1} . Now link Q_{n-1} to nR_{n-1} by adding edge $\{1, 2, n-1\}\{1, 2, n\}$. (See Figure 7.9(a).)

Case 2. If $\binom{n+1}{2}$ is even and n is even, then we construct Hamilton path P_n in G_n by pulling edge e_n into nP_{n-1} and edge a_n into Q_{n-1} and then adding edge $\{2, n-1\}\{2, n\}$. (See Figure 7.9(b).)

Case 3. If $\binom{n+1}{2}$ is odd and n is odd, then we must construct both P_n and R_n . We first construct P_n . We take the complement in $[n-1]$ of every set on the path P_{n-1} to get a path $\overline{P_{n-1}}$ from $\{1, 3, 4, \dots, n-1\}$ to $\{1, 2\}$. Now $n\overline{P_{n-1}}$ is a Hamilton path in nG_{n-1} from $\{1, 3, 4, \dots, n-1, n\}$ to $\{1, 2, n\}$. Note that $n\overline{P_{n-1}}$ must have edge $\{1, 2, \dots, n-2, n\}\{2, 3, \dots, n-2, n\}$ by Lemma 7. To construct P_n , pull edge a_n into Q_{n-1} and edge e_n into $n\overline{P_{n-1}}$, and then join the two paths with the edge from $\{1, 2, n-1\}$ to $\{1, 2, n\}$ (See Figure 7.10.)

$P_6 =$

{2},	{1, 2},	{1, 3},	{3},	{4},	{1, 4},	{2, 4},
{2, 3},	{1, 2, 3},	{1, 2, 4},	{1, 2, 5},	{2, 5},	{1, 5},	{5},
{6},	{1, 6},	{2, 6},	{1, 2, 6},	{1, 3, 6},	{3, 6},	{4, 6},
{5, 6},	{1, 5, 6},	{2, 5, 6},	{1, 2, 5, 6},	{1, 2, 4, 6},	{1, 2, 3, 6},	{2, 3, 6},
{2, 4, 6},	{1, 4, 6},	{1, 4, 5},	{4, 5},	{3, 5},	{3, 4},	{1, 3, 4},
{1, 3, 5},	{2, 3, 5},	{2, 3, 4},	{1, 2, 3, 4},	{1, 2, 3, 5},	{1, 2, 4, 5},	{2, 4, 5},
{3, 4, 5},	{1, 3, 4, 5},	{2, 3, 4, 5},	{1, 2, 3, 4, 5},	{1, 2, 3, 4, 6},	{2, 3, 4, 6},	{1, 3, 4, 6},
{3, 4, 6},	{3, 5, 6},	{4, 5, 6},	{1, 4, 5, 6},	{1, 3, 5, 6},	{2, 3, 5, 6},	{1, 2, 3, 5, 6},
{1, 2, 4, 5, 6},	{2, 4, 5, 6},	{3, 4, 5, 6},	{1, 3, 4, 5, 6}			

$R_6 =$

{1, 2},	{2},	{3},	{4},	{5},	{6},	{1, 6},
{2, 6},	{1, 2, 6},	{1, 2, 5},	{2, 5},	{1, 5},	{1, 4},	{1, 3},
{2, 3},	{1, 2, 3},	{1, 2, 4},	{2, 4},	{3, 4},	{1, 3, 4},	{2, 3, 4},
{1, 2, 3, 4},	{1, 2, 3, 5},	{1, 2, 3, 6},	{1, 2, 4, 6},	{1, 2, 5, 6},	{2, 5, 6},	{1, 5, 6},
{5, 6},	{4, 6},	{3, 6},	{1, 3, 6},	{1, 4, 6},	{2, 4, 6},	{2, 3, 6},
{2, 3, 5},	{1, 3, 5},	{3, 5},	{4, 5},	{1, 4, 5},	{2, 4, 5},	{1, 2, 4, 5},
{1, 3, 4, 5},	{3, 4, 5},	{3, 4, 6},	{1, 3, 4, 6},	{1, 3, 5, 6},	{3, 5, 6},	{4, 5, 6},
{1, 4, 5, 6},	{2, 4, 5, 6},	{2, 3, 5, 6},	{2, 3, 4, 6},	{2, 3, 4, 5},	{1, 2, 3, 4, 5},	{1, 2, 3, 4, 6},
{1, 2, 3, 5, 6},	{1, 2, 4, 5, 6},	{1, 3, 4, 5, 6},	{3, 4, 5, 6}			

Figure 7.8: The Hamilton paths P_6 and R_6 (read across) in Lemma 11 .

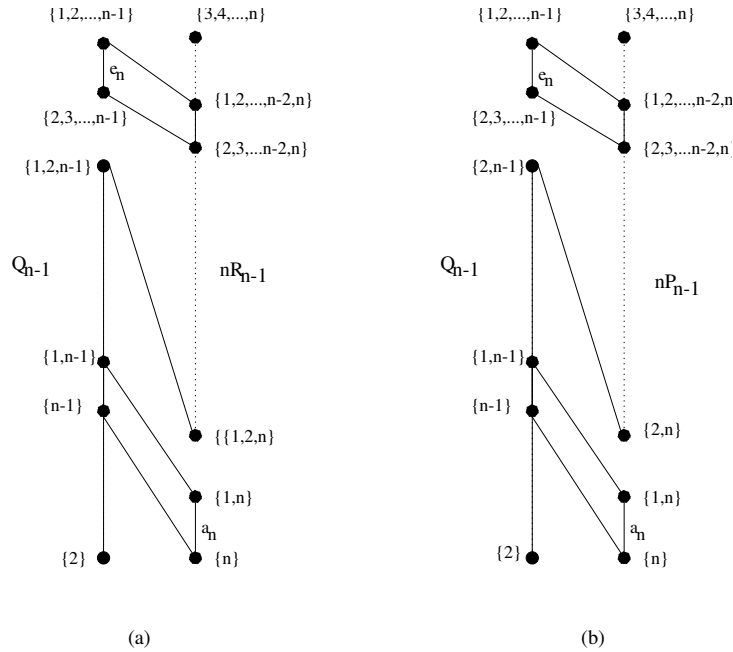


Figure 7.9: Construction of P_n in Lemma 11 when $\binom{n+1}{2}$ is even and n is (a) odd, (b) even.

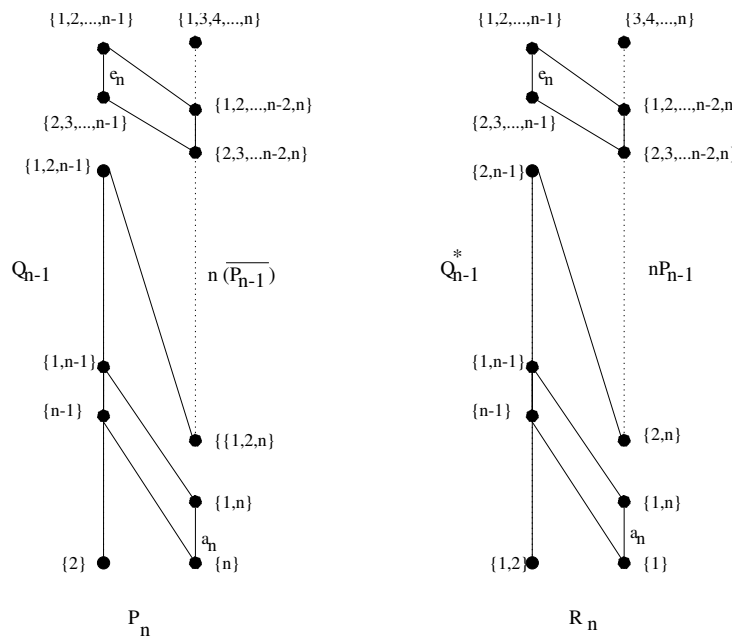


Figure 7.10: Construction of P_n and R_n in Lemma 11 when $\binom{n+1}{2}$ is odd and n is odd.

We construct R_n using Q_{n-1}^* and nP_{n-1} . First, pull edge e_n into nP_{n-1} and edge a_n into Q_{n-1}^* , and then join the paths by adding the edge $\{2, n-1\}\{2, n\}$ (See Figure 7.10.)

Case 4. If $\binom{n+1}{2}$ is odd and n is even, then we again construct both P_n and R_n . We pull edge e_n into nP_{n-1} and edge a_n into Q_{n-1} , and then join the paths by adding the edge $\{2, n-1\}\{2, n\}$ to form P_n . We pull edge e_n into nR_{n-1} and edge a_n into Q_{n-1}^* , and then join the paths by adding the edge $\{1, 2, n-1\}\{1, 2, n\}$ to form R_n . \square

Theorem 24 is clear for $n \leq 2$. For $n > 5$ and $\binom{n+1}{2}$, the result follows from Lemma 11 by inserting $\emptyset, \{1\}$ at the beginning and appending $\{2, 3, \dots, n\}, \{1, 2, \dots, n\}$ to the end of P_n to complete a Hamilton path in the full graph A_n .

Chapter 8

Conclusion

In this thesis we have described our work on devising a heuristic, using a combination of previous ideas and new, innovative techniques to create a useful tool for finding Hamilton cycles in large graphs. We have used this tool to extend to body of knowledge on Hamilton cycles in several classes of graphs.

Chapter 2 reviewed the main Hamilton path and cycle results for non-random and random graphs and introduced us to techniques that had been used for finding Hamilton cycles.

Chapter 3 described our heuristic which synthesizes several extension-rotation ideas used in the study of random graphs into a heuristic that we used for finding Hamilton cycles in non-random graphs. We described features that we added to the previous ideas, as well as techniques we used to allow the algorithm to work on some very large graphs that would normally be considered beyond the scope of the personal computer equipment we used for this work.

Chapter 4 described our work on the middle two levels problem, M_{2k+1} . We showed the existence of Hamilton cycles for $k \leq 15$. For $k = 15$ we reduced the problem of finding a Hamilton cycle in a graph with over 600,000,000 vertices to a problem of finding a Hamilton path in a graph with almost 10,000,000 vertices and found these paths in the reduced graph using our heuristic.

Chapter 5 described our work on connected Kneser graphs $K(n, k)$, and bipartite Kneser graphs, $H(n, k)$. We showed that the connected Kneser Graph $K(n, k)$ and its bipartite analog, $H(n, k)$ are Hamiltonian for all values of $n \leq 27$. Since the bipartite Kneser graph $H(2k + 1, k)$ is also an example of the middle two levels problem, the work of Chapter 4 extends the results for this case to $n \leq 31$. We created and used a very fast algorithm for finding neighbors in the Kneser graphs, trading some computation time for a significant space saving, allowing us to find cycles in graphs with over 80 billion

edges.

Chapter 6 described our work on cubic Cayley graphs derived from subgroups of the symmetric group S_n . These are generated by three involutions or by one involution and one other group element that is not an involution. These graphs proved hardest for our heuristic and we examined the characteristics of graphs in which we did not find cycles. We devised an algorithm to significantly reduce the size of the problem space by creating non-conjugate sets of three involutions in S_n . We showed that all Cayley graphs generated by three involutions from S_n with $n \leq 7$ are Hamiltonian and we described our results for Cayley graphs generated by three involutions from S_8 , in which we have shown Hamilton cycles in all but three graphs generated by the alternating group A_8 .

Chapter 7 described our work on the cover graph, A_n , of the poset $M(n)$ which has as its elements the n -tuples of integers $\mathbf{a} = (a_1, a_2, \dots, a_n)$ satisfying $0 = a_1 = \dots = a_j < a_{j+1} < \dots < a_n \leq n$ for some $j \geq 0$ with an order relation defined by $\mathbf{a} \leq \mathbf{b}$ if and only if $a_i \leq b_i$ for $1 \leq i \leq n$. We used our heuristic to confirm that several suspected graphs did, in fact, have Hamilton paths, and then used it again to find specific paths that served as the basis cases in several induction arguments. With these, we completely characterized those values of n for which A_n has a Hamilton path and those for which it does not.

8.1 Future Work

All of our work to date has been done on a 32-bit personal computer. For the middle levels problem, we are at the practical limit of vertex representations that will fit conveniently in a 32-bit word and for Kneser graphs we are close to the limit. Work on larger problems of this type is likely to require a 64-bit machine.

We have not investigated any possible advantages from a parallel version of our heuristic. An obvious form of parallel operation would be to start several copies of a problem on different processors using different starting random number seeds. Other possibilities might include distributing work on calculating adjacency lists, particularly during the BFS search phase of the heuristic.

Work on addressing the problem with small even cycles in cubic Cayley graphs remains to be done. We believe our heuristic could be modified, albeit at the expense of running time, to avoid what we conjecture to be the problem. Further study of these graphs may provide insight into problems for heuristics such as ours.

Our work on conjugate generating sets could be extended to cubic Cayley graphs generated by one involution and one non-involution. It may also form a basis for further work on graph isomorphism in S_n , or possibly other Cayley graphs.

Bibliography

- [1] M. Ajtai, J. Komlós, and E. Szemerédi. First occurrence of Hamilton cycles in random graphs. In *Cycles in graphs (Burnaby, B.C., 1982)*, pages 173–178. North-Holland, Amsterdam, 1985.
- [2] D. Angluin and L. G. Valiant. Fast probabilistic algorithms for Hamiltonian circuits and matchings. *J. Comput. System Sci.*, 18(2):155–193, 1979.
- [3] László Babai. Long cycles in vertex-transitive graphs. *J. Graph Theory*, 3(3):301–304, 1979.
- [4] A. T. Balaban. Chemical graphs, Part XIII; Combinatorial patterns. *Rev. Roumain Math. Pures Appl.*, 17:3–16, 1972.
- [5] Zs. Baranyai. On the factorization of the complete uniform hypergraph. In *Infinite and finite sets (Colloq., Keszthely, 1973; dedicated to P. Erdős on his 60th birthday), Vol. I*, pages 91–108. Colloq. Math. Soc. János Bolyai, Vol. 10. North-Holland, Amsterdam, 1975.
- [6] J-C Bermond. *Hamiltonian graphs*. Academic Press Inc. [Harcourt Brace Jovanovich Publishers], London, 1978.
- [7] Girish S. Bhat and Carla D. Savage. Balanced Gray codes. *Electron. J. Combin.*, 3(1):Research Paper 25, approx. 11 pp. (electronic), 1996.
- [8] Norman L. Biggs, E. Keith Lloyd, and Robin J. Wilson. *Graph theory. 1736–1936*. The Clarendon Press Oxford University Press, New York, second edition, 1986.
- [9] B. Bollobás, T. I. Fenner, and A. M. Frieze. An algorithm for finding Hamilton paths and cycles in random graphs. *Combinatorica*, 7(4):327–341, 1987.
- [10] Béla Bollobás. *Graph theory*. Springer-Verlag, New York, 1979. An introductory course.
- [11] Béla Bollobás. Random graphs. In *Combinatorics (Swansea, 1981)*, volume 52 of *London Math. Soc. Lecture Note Ser.*, pages 80–102. Cambridge Univ. Press, Cambridge, 1981.
- [12] Béla Bollobás. Almost all regular graphs are Hamiltonian. *European J. Combin.*, 4(2):97–106, 1983.
- [13] Béla Bollobás. The evolution of sparse graphs. In *Graph theory and combinatorics (Cambridge, 1983)*, pages 35–57. Academic Press, London, 1984.
- [14] J. A. Bondy and V. Chvátal. A method in graph theory. *Discrete Math.*, 15(2):111–135, 1976.
- [15] Andrei Z. Broder, Alan M. Frieze, and Eli Shamir. Finding hidden Hamiltonian cycles. *Random Structures Algorithms*, 5(3):395–410, 1994.

- [16] H. J. Broersma, J. van den Heuvel, and H. J. Veldman. A generalization of Ore's theorem involving neighborhood unions. *Discrete Math.*, 122(1-3):37–49, 1993.
- [17] Bor Liang Chen and Ko-Wei Lih. Hamiltonian uniform subset graphs. *J. Combin. Theory Ser. B*, 42(3):257–263, 1987.
- [18] Ya-Chen Chen. *Extremal Problems in Graph Theory: Hamiltonicity, Minimum vertex-Diameter-2-Critical Graphs and Decomposition*. PhD thesis, University of Illinois, Urbana, 2000.
- [19] Ya-Chen Chen. Kneser graphs are Hamiltonian for $n \geq 3k$. *J. Combin. Theory Ser. B*, 80(1):69–79, 2000.
- [20] Ya-Chen Chen. Triangle-free Hamiltonian Kneser graphs. *J. Combin. Theory Ser. B*, 89(1):1–16, 2003.
- [21] Yu Qing Chen. On Hamiltonicity of vertex-transitive graphs and digraphs of order p^4 . *J. Combin. Theory Ser. B*, 72(1):110–121, 1998.
- [22] V. Chvátal. On Hamilton's ideals. *J. Combinatorial Theory Ser. B*, 12:163–168, 1972.
- [23] V. Chvátal and P. Erdős. A note on Hamiltonian circuits. *Discrete Math.*, 2:111–113, 1972.
- [24] W. Edwin Clark and Mourad E. H. Ismail. Binomial and Q -binomial coefficient inequalities related to the Hamiltonicity of the Kneser graphs and their Q -analogues. *J. Combin. Theory Ser. A*, 76(1):83–98, 1996.
- [25] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to algorithms*. MIT Press, Cambridge, MA, 1990.
- [26] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.
- [27] Stephen J. Curran and Joseph A. Gallian. Hamiltonian cycles and paths in Cayley graphs and digraphs—a survey. *Discrete Math.*, 156(1-3):1–18, 1996.
- [28] Eugene Curtin. Cubic Cayley graphs with small diameter. *Discrete Math. Theor. Comput. Sci.*, 4(2):123–131 (electronic), 2001.
- [29] Alice M. Dean, C. J. Knickerbocker, Patti Frazer Lock, and Michael Sheard. A survey of graphs Hamiltonian-connected from a vertex. In *Graph theory, combinatorics, and applications, Vol. 1 (Kalamazoo, MI, 1988)*, pages 297–313. Wiley, New York, 1991.
- [30] Lucy Dechéne. Ringing the changes: an aural permutation group. In *Innovations in teaching abstract algebra*, volume 60 of *MAA Notes*, pages 137–142. Math. Assoc. America, Washington, DC, 2002.
- [31] I. J. Dejter. Hamilton cycles and quotients of bipartite graphs. In *Graph theory with applications to algorithms and computer science (Kalamazoo, Mich., 1984)*, Wiley-Intersci. Publ., pages 189–199. Wiley, New York, 1985.
- [32] I. J. Dejter, W. Cedeño, and V. Jáuregui. Frucht diagrams, Boolean graphs and Hamilton cycles. *Sci. Ser. A Math. Sci. (N.S.)*, 5:21–37 (1995), 1992/93.
- [33] I. J. Dejter, J. Cordova, and J. A. Quintana. Two Hamilton cycles in bipartite reflective Kneser graphs. In *Proceedings of the First Japan Conference on Graph Theory and Applications (Hakone, 1986)*, volume 72, pages 63–70, 1988.
- [34] G. A. Dirac. Some theorems on abstract graphs. *Proc. London Math. Soc. (3)*, 2:69–81, 1952.

- [35] Dwight Duffus, Bill Sands, and Robert Woodrow. Lexicographic matchings cannot form Hamiltonian cycles. *Order*, 5(2):149–161, 1988.
- [36] Scott Effler and Frank Ruskey. Enumeration, isomorphism and Hamiltonicity of Cayley graphs: 2-generated and cubic. <http://www.theory.csc.uvic.ca/~cos/cayley>, 2002.
- [37] P. Erdős and A. Rényi. On random graphs. I. *Publ. Math. Debrecen*, 6:290–297, 1959.
- [38] P. Erdős and A. Rényi. On the evolution of random graphs. *Magyar Tud. Akad. Mat. Kutató Int. Közl.*, 5:17–61, 1960.
- [39] P. Erdős and A. Rényi. On the existence of a factor of degree one of a connected random graph. *Acta Math. Acad. Sci. Hungar.*, 17:359–368, 1966.
- [40] Paul Erdős. Problems and results in combinatorial analysis and combinatorial number theory. In *Proceedings of the Ninth Southeastern Conference on Combinatorics, Graph Theory, and Computing (Florida Atlantic Univ., Boca Raton, Fla., 1978)*, pages 29–40, Winnipeg, Man., 1978. Utilitas Math.
- [41] Geng Hua Fan. New sufficient conditions for cycles in graphs. *J. Combin. Theory Ser. B*, 37(3):221–227, 1984.
- [42] R. Faudree. Forbidden subgraphs, closure and Hamiltonian properties - recent results. In *Graph theory and combinatorial biology (Balatonlelle, 1996)*, pages 9–27. János Bolyai Math. Soc., Budapest, 1999.
- [43] R. J. Faudree, Ronald J. Gould, Michael S. Jacobson, and R. H. Schelp. Neighborhood unions and Hamiltonian properties in graphs. *J. Combin. Theory Ser. B*, 47(1):1–9, 1989.
- [44] Ralph Faudree. Forbidden subgraphs and Hamiltonian properties: a survey. *Congr. Numer.*, 116:33–52, 1996. Surveys in graph theory (San Francisco, CA, 1995).
- [45] Stefan Felsner and William T. Trotter. Colorings of diagrams of interval orders and α -sequences of sets. *Discrete Math.*, 144(1-3):23–31, 1995. Combinatorics of ordered sets (Oberwolfach, 1991).
- [46] T. I. Fenner and A. M. Frieze. Hamiltonian cycles in random regular graphs. *J. Combin. Theory Ser. B*, 37(2):103–112, 1984.
- [47] I. Fournier and P. Fraisse. On a conjecture of Bondy. *J. Combin. Theory Ser. B*, 39(1):17–26, 1985.
- [48] A. M. Frieze. Finding Hamilton cycles in sparse random graphs. *J. Combin. Theory Ser. B*, 44(2):230–250, 1988.
- [49] Alan Frieze, Mark Jerrum, Michael Molloy, Robert Robinson, and Nicholas Wormald. Generating and counting Hamilton cycles in random regular graphs. *J. Algorithms*, 21(1):176–198, 1996.
- [50] E. N. Gilbert. Gray codes and paths on the n -cube. *Bell System Tech. J.*, 37:815–826, 1958.
- [51] Luis Goddyn, George M. Lawrence, and Evi Nemeth. Gray codes with optimized run lengths. *Utilitas Math.*, 34:179–192, 1988.
- [52] Ronald J. Gould. Updating the Hamiltonian problem—a survey. *J. Graph Theory*, 15(2):121–157, 1991.
- [53] F. Gray. Pulse code communication, U.S patent no. 2.632.058, March 1953.

- [54] Yuri Gurevich and Saharon Shelah. Expected computation time for Hamiltonian path problem. *SIAM J. Comput.*, 16(3):486–502, 1987.
- [55] Katherine Heinrich and W. D. Wallis. Hamiltonian cycles in certain graphs. *J. Austral. Math. Soc. Ser. A*, 26(1):89–98, 1978.
- [56] Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. *J. Soc. Indust. Appl. Math.*, 10:196–210, 1962.
- [57] A. S. Herschel. Sir Wm. Hamilton’s Icosian Game. *Quart. J. Pure Applied Math*, 5:305, 1862.
- [58] I. N. Herstein. *Topics in Algebra*. Blaisdell Publishing Company., Waltham, Massachusetts, 1964.
- [59] Marie-Claude Heydemann. Cayley graphs and interconnection networks. In *Graph symmetry (Montreal, PQ, 1996)*, pages 167–224. Kluwer Acad. Publ., Dordrecht, 1997.
- [60] Glenn Hurlbert. The antipodal layers problem. *Discrete Math.*, 128(1-3):237–245, 1994.
- [61] Bill Jackson. Hamilton cycles in regular 2-connected graphs. *J. Combin. Theory Ser. B*, 29(1):27–46, 1980.
- [62] Bill Jackson. Neighborhood unions and Hamilton cycles. *J. Graph Theory*, 15(4):443–451, 1991.
- [63] J. Robert Johnson. Long cycles and the middle two layers of the discrete cube. To appear.
- [64] Selmer M. Johnson. Generation of permutations by adjacent transposition. *Math. Comp.*, 17:282–285, 1963.
- [65] Kevin Keating and David Witte. On Hamilton cycles in Cayley graphs in groups with cyclic commutator subgroup. In *Cycles in graphs (Burnaby, B.C., 1982)*, pages 89–102. North-Holland, Amsterdam, 1985.
- [66] H. A. Kierstead and W. T. Trotter. Explicit matchings in the middle levels of the Boolean lattice. *Order*, 5(2):163–171, 1988.
- [67] J. Komlós and E. Szemerédi. Hamilton cycles in random graphs. In *Infinite and finite sets (Colloq., Keszthely, 1973; dedicated to P. Erdős on his 60th birthday)*, Vol. II, pages 1003–1010. Colloq. Math. Soc. János Bolyai, Vol. 10. North-Holland, Amsterdam, 1975.
- [68] János Komlós and Endre Szemerédi. Limit distribution for the existence of Hamiltonian cycles in a random graph. *Discrete Math.*, 43(1):55–63, 1983.
- [69] V. L. Kompel’maher and V. A. Liskovec. Successive generation of permutations by means of a transposition basis. *Kibernetika (Kiev)*, (3):17–21, 1975.
- [70] Aleksej D. Korshunov. A new version of the solution of a problem of Erdős and Rényi on Hamiltonian cycles in undirected graphs. In *Random graphs ’83 (Poznań, 1983)*, pages 171–180. North-Holland, Amsterdam, 1985.
- [71] A. D. Koršunov. Solution of a problem of P. Erdős and A. Rényi on Hamiltonian cycles in undirected graphs. *Soviet Math Dokl.*, 17(3):760–764, 1976.
- [72] Linda M. Lesniak. Neighborhood unions and graphical properties. In *Graph theory, combinatorics, and applications. Vol. 2 (Kalamazoo, MI, 1988)*, pages 783–800. Wiley, New York, 1991.
- [73] Terri England Lindquister. The effects of distance and neighborhood union conditions on Hamiltonian properties in graphs. *J. Graph Theory*, 13(3):335–352, 1989.

- [74] X. Liu, Yong Jin Zhu, and Lian Zhu Zhang. Distance, neighborhood unions and Hamiltonian properties in graphs. In *Combinatorics, graph theory, algorithms and applications (Beijing, 1993)*, pages 255–268. World Sci. Publishing, River Edge, NJ, 1994.
- [75] L. Lovász. Problem 11. In *Combinatorial Structures and their Applications*. Gordon and Breach, 1970.
- [76] Xiaoyun Lu. A Chvátal-Erdős type condition for Hamiltonian graphs. *J. Graph Theory*, 18(8):791–800, 1994.
- [77] D. Marušič. Vertex transitive graphs and digraphs of order p^k . In *Cycles in graphs (Burnaby, B.C., 1982)*, pages 115–128. North-Holland, Amsterdam, 1985.
- [78] Michael Mather. The Rugby footballers of Croam. *J. Combinatorial Theory Ser. B*, 20(1):62–63, 1976.
- [79] Guy H. J. Meredith and E. Keith Lloyd. The Hamiltonian graphs O_4 to O_7 . In *Combinatorics (Proc. Conf. Combinatorial Math., Math. Inst., Oxford, 1972)*, pages 229–236. Inst. Math. Appl., Southend, 1972.
- [80] Guy H. J. Meredith and E. Keith Lloyd. The footballers of Croam. *J. Combinatorial Theory Ser. B*, 15:161–166, 1973.
- [81] Oystein Ore. Note on Hamilton circuits. *Amer. Math. Monthly*, 67:55, 1960.
- [82] L. Pósa. Hamiltonian circuits in random graphs. *Discrete Math.*, 14(4):359–364, 1976.
- [83] Cheryl E. Praeger, Cai Heng Li, and Alice C. Niemeyer. Finite transitive permutation groups and finite vertex-transitive graphs. In *Graph symmetry (Montreal, PQ, 1996)*, pages 277–318. Kluwer Acad. Publ., Dordrecht, 1997.
- [84] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical recipes in C*. Cambridge University Press, Cambridge, second edition, 1992. The art of scientific computing.
- [85] Robert A. Proctor. Solution of two difficult combinatorial problems with linear algebra. *Amer. Math. Monthly*, 89(10):721–734, 1982.
- [86] Mike Reid. E-mail to J Gallian. Correspondence describing work performed by David Moews and Mike Reid on middle levels problem, 1990.
- [87] Edward M. Reingold, Jurg Nievergelt, and Narsingh Deo. *Combinatorial algorithms: theory and practice*. Prentice-Hall Inc., Englewood Cliffs, N.J., 1977.
- [88] Edward M. Reingold, Jurg Nievergelt, and Narsingh Deo. *Combinatorial Algorithms: Theory and Practice*. Prentice-Hall, Englewood Cliffs, New Jersey, 1977.
- [89] R. W. Robinson and N. C. Wormald. Almost all cubic graphs are Hamiltonian. *Random Structures Algorithms*, 3(2):117–125, 1992.
- [90] R. W. Robinson and N. C. Wormald. Almost all regular graphs are Hamiltonian. *Random Structures Algorithms*, 5(2):363–374, 1994.
- [91] Frank Ruskey and Carla Savage. Hamilton cycles that extend transposition matchings in Cayley graphs of S_n . *SIAM J. Discrete Math.*, 6(1):152–166, 1993.
- [92] Carla D. Savage. Long cycles in the middle two levels of the Boolean lattice. *Ars Combin.*, 35(A):97–108, 1993.

- [93] Carla D. Savage, Ian Shields, and Douglas B. West. On the existence of Hamiltonian paths in the cover graph of $M(n)$. *Discrete Math.*, 262(1-3):241–252, 2003.
- [94] Carla D. Savage and Peter Winkler. Monotone Gray codes and the middle levels problem. *J. Combin. Theory Ser. A*, 70(2):230–248, 1995.
- [95] Raffaele Scapellato. Vertex-transitive graphs and digraphs. In *Graph symmetry (Montreal, PQ, 1996)*, pages 319–378. Kluwer Acad. Publ., Dordrecht, 1997.
- [96] Eli Shamir. How many random edges make a graph Hamiltonian? *Combinatorica*, 3(1):123–131, 1983.
- [97] Ru Qun Shen and Feng Tian. Neighborhood unions and Hamiltonicity of graphs. *Discrete Math.*, 141(1-3):213–225, 1995.
- [98] Ian Shields and Carla D. Savage. A Hamilton path heuristic with applications to the middle two levels problem. In *Proceedings of the Thirtieth Southeastern International Conference on Combinatorics, Graph Theory, and Computing (Boca Raton, FL, 1999)*, volume 140, pages 161–178, 1999.
- [99] Ian Shields and Carla D. Savage. A note on Hamilton cycles in Kneser graphs. *Bull. Inst. Combin. Appl.*, 40:13–22, 2004.
- [100] J. E. Simpson. Hamiltonian bipartite graphs. In *Proceedings of the Twenty-second Southeastern Conference on Combinatorics, Graph Theory, and Computing (Baton Rouge, LA, 1991)*, volume 85, pages 97–110, 1991.
- [101] J. E. Simpson. On uniform subset graphs. *Ars Combin.*, 37:309–318, 1994.
- [102] Denis Sjerne and Michael Cherkassoff. On groups generated by three involutions, two of which commute. In *The Hilton Symposium 1993 (Montreal, PQ)*, pages 169–185. Amer. Math. Soc., Providence, RI, 1994.
- [103] Richard P. Stanley. Weyl groups, the hard Lefschetz theorem, and the Sperner property. *SIAM J. Algebraic Discrete Methods*, 1(2):168–184, 1980.
- [104] Dennis Stanton and Dennis White. *Constructive combinatorics*. Springer-Verlag, New York, 1986.
- [105] Andrew Thomason. A simple linear expected time algorithm for finding a Hamilton path. *Discrete Math.*, 75(1-3):373–379, 1989. Graph theory and combinatorics (Cambridge, 1988).
- [106] H. F. Trotter. Algorithm 115: Perm. *Commun. ACM*, 5(8):434–435, 1962.
- [107] Basil Vandegriend. Finding hamilton cycles: Algorithms, graphs and performance. Master’s thesis, University of Alberta, 1998.
- [108] Douglas B. West. *Introduction to graph theory*. Prentice Hall Inc., Upper Saddle River, NJ, 1996.
- [109] Arthur T. White. Ringing the cosets. *Amer. Math. Monthly*, 94(8):721–746, 1987.
- [110] Dave Witte. Cayley digraphs of prime-power order are Hamiltonian. *J. Combin. Theory Ser. B*, 40(1):107–112, 1986.
- [111] David Witte and Joseph A. Gallian. A survey: Hamiltonian cycles in Cayley graphs. *Discrete Math.*, 51(3):293–304, 1984.