Hypercube Coloring and the Structure of Binary Codes

by

James Gregory Rix

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

The College of Graduate Studies

(Interdisciplinary Studies)

The University Of British Columbia

(Okanagan)

July 2008

Abstract

A coloring of a graph is an assignment of colors to its vertices so that no two adjacent vertices are given the same color. The chromatic number of a graph is the least number of colors needed to color all of its vertices. Graph coloring problems can be applied to many real world applications, such as scheduling and register allocation. Computationally, the decision problem of whether a general graph is m-colorable is NP-complete for $m \geq 3$.

The graph studied in this thesis is a well-known combinatorial object, the k-dimensional hypercube, Q_k. The hypercube itself is 2-colorable for all k; however, coloring the square of the cube is a much more interesting problem. This is the graph in which the vertices are binary vectors of length k, and two vertices are adjacent if and only if the Hamming distance between the two vectors is at most 2.

Any color class in a coloring of Q_k^2 is a binary $(k, M, 3)$ code. This thesis will begin with an introduction to binary codes and their structure. One of the most fundamental combinatorial problems is finding optimal binary codes, that is, binary codes with the maximum cardinality satisfying a specified length and minimum distance. Many upper and lower bounds have been produced, and we will analyze and apply several of these. This leads to many interesting results about the chromatic number of the square of the cube.

The smallest k for which the chromatic number of Q_k^2 is unknown is $k = 8$; however, it can be determined that this value is either 13 or 14. Computational approaches to determine the chromatic number of Q_8^2 were performed. We were unable to determine whether 13 or 14 is the true value; however, much valuable insight was learned about the structure of this graph and the computational difficulty that lies within. Since a 13-coloring of Q_8^2 must have between 9 and 12 color classes being $(8, 20, 3)$ binary codes, this led to a thorough investigation of the structure of such binary codes.

Table of Contents

List of Tables

List of Figures

Acknowledgements

First and foremost, I would like to thank my supervisor, Dr. Donovan Hare, for both introducing me to the beauty of optimization from a real world perspective, and for giving me the opportunity to pursue my research in this field. His infinite patience was extremely helpful over the course of my educational experience. In addition, I would like to give thanks to everyone who has had an invaluable impact on my educational experience: Dr. Wayne Broughton, Dr. Heinz Bauschke, Dr. Shawn Wang, Dr. Yves Lucet, Dr. Yong Gao, and Dr. Blair Spearman.

This thesis was partially supported by the Natural Sciences and Engineering Research Council of Canada, the Pacific Institute for the Mathematical Sciences and the U.B.C. Okanagan College of Graduate Studies.

Chapter 1

Coding Theory

In this chapter, we recall background material from coding theory that will be used in subsequent chapters. The chapter will conclude with the sphere-packing bound and Delsarte's linear programming bound, two very useful bounds in determining optimal binary code size. We will also include a proof of $A(9,4) = 20$ that uses Delsarte's linear programming bound. This value will be very important in later chapters. We will now introduce the notation and definitions that will be used throughout, which closely follow those used in [20] and [13].

It will be assumed throughout this thesis that reader has a basic knowledge of field theory, set theory, linear and integer programming, and probability.

1.1 Preliminaries

Notation 1.1. Let $\mathbb{Z}$ denote the set of all integers, and $\mathbb{Z}^+$ denote the set of all positive integers.

Notation 1.2. For any positive integer n, let $[n]$ denote the set of integers $\{1, 2, \ldots, n\}$.

Consider a finite field F with addition and multiplication defined, satisfying $|F| = q$. Recall that all fields have an additive identity, 0, and a multiplicitave identity, 1. Now consider the vector space F^n. Clearly, $|F^n| = q^n$. A vector u in F^n is of the following form:

$$u = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix}.$$

For notational convenience, this will also be denoted $u_1 u_2 \ldots u_n$. We will refer to u_i as the ith coordinate of u.

Notation 1.3. Let $\vec{0}^n$ and $\vec{1}^n$ denote the vectors in F^n in which every coordinate is 0 and 1, respectively.

Notation 1.4. Let μ_i^n denote the vector in F^n in which the ith coordinate is 1, and every other coordinate is 0.

Definition 1.5. Consider two vectors u and v in F^n. The *(Hamming) distance* between u and v, denoted $d(u, v)$, is defined to be the number of coordinates in which u and v differ.

Let $u, v, w \in F^n$. We state without proof the following properties of the Hamming distance function:

- $0 \leq d(u, v) \leq n$,
- $d(u, v) = 0$ if and only if $u = v$,
- $d(u, v) = d(v, u)$,
- $d(u, w) \leq d(u, v) + d(v, w)$ (the triangle inequality).

Definition 1.6. Let $u \in F^n$ and t be a positive integer. The *(Hamming) sphere* of radius t centered at u, denoted $S_t(u)$, is the following set of vectors:

$$S_t(u) = \{v \in \{0, 1\}^n \mid d(u, v) \leq t\}.$$

Definition 1.7. Consider a vector u in F^n. The *(Hamming) weight* of u, denoted $wt(u)$, is defined to be the number of coordinates of u that are nonzero. The *parity* of u is even if $wt(u)$ is even, and odd if $wt(u)$ is odd.

Consider two vectors u and v in F^n. The following properties of the Hamming weight function are immediate:

- $0 \leq wt(u) \leq n$,
- $wt(u) = 0$ if and only if $u = \vec{0}^n$,
- $wt(u) = n$ if and only if $u = \vec{1}^n$,
- $wt(u - v) = d(u, v)$.

Definition 1.8. Consider a vector u in F^n. We define p_u and q_u to be following elements of F:

$$p_u = \begin{cases} 0 & wt(u) \text{ is even,} \\ 1 & wt(u) \text{ is odd,} \end{cases}$$

$$q_u = \begin{cases} 1 & wt(u) \text{ is even,} \\ 0 & wt(u) \text{ is odd.} \end{cases}$$

Lemma 1.9. *Let $u, v, w \in F^n$. Then $d(u, v) = d(u + w, v + w)$.*

Proof.

$$\begin{aligned} d(u + w, v + w) &= wt((u + w) - (v + w)) \\ &= wt((u - v) + (w - w)) \\ &= wt(u - v) \\ &= d(u, v). \end{aligned}$$

□

Notation 1.10. Let S_n, for all positive integers n, denote the symmetric group of degree n (i.e. the set of all permutations on the integers $[n]$).

Definition 1.11. Let $\sigma \in S_n$ and $u \in F^n$ with coordinates defined by $u = u_1 u_2 \dots u_n$. Then $\sigma(u) \in F^n$ is defined as follows:

$$\sigma(u) = u_{\sigma(1)} u_{\sigma(2)} \dots u_{\sigma(n)}.$$

For any vectors u and v in F^n, and permutation $\sigma \in S_n$, the following properties are immediate:

- $wt(\sigma(u)) = wt(u)$,
- $d(\sigma(u), \sigma(v)) = d(u, v)$.
- $\sigma(u) + \sigma(v) = \sigma(u + v)$.

Definition 1.12. Let $u \in F^m$ and $v \in F^n$ have coordinates defined by $u = u_1 u_2 \dots u_m$ and $v = v_1 v_2 \dots v_n$. We define u *concatenated with* v, denoted u_v, to be:

$$u\_v = u_1 u_2 \dots u_m v_1 v_2 \dots v_n,$$

an element of the vector space F^{m+n}.

Let $u, u' \in F^m$ and $v, v' \in F^n$. Clearly, from the distance functions of their respective vector spaces:

$$d(u\_v, u'\_v') = d(u, u') + d(v, v').$$

One technique used often in the study of codes is to concatenate a vector $u \in F^n$ with a parity coordinate, that is, create the vector u_p_u. It is clear that $wt(u\_p_u)$ is even for any $u \in F^n$.

Definition 1.13. Consider two vectors u and v in F^n with their coordinates defined by $u = u_1u_2 \dots u_n$ and $v = v_1v_2 \dots v_n$. The *inner product* of u and v is defined as follows:

$$< u, v > = \sum_{i=1}^{n} u_i v_i.$$

Note: the inner product of u and v is an element of the field F.

Let $u, v, w \in F^n$ and $c \in F$. As this is an inner product, it satisfies several key properties:

- $< u, v >=< v, u >$,
- $< u, v + w >=< u, v > + < u, w >$,
- $c < u, v >=< cu, v >$.

In this thesis, we will often be dealing with the field $\mathbb{Z}_2 = \{0, 1\}$, with binary addition and multiplication defined normally. When dealing with this field, we can define the intersection of two vectors. This will give rise to several useful results.

Definition 1.14. Let u and v be binary vectors in $\{0, 1\}^n$ with their coordinates defined by $u = u_1u_2 \dots u_n$ and $v = v_1v_2 \dots v_n$. The *intersection* of u and v is defined to be:

$$u \cap v = \begin{pmatrix} u_1v_1 \\ u_2v_2 \\ \vdots \\ u_nv_n \end{pmatrix}.$$

Thus the ith coordinate of $u \cap v$ is 1 if and only if the ith coordinate of both u and v are 1.

Let u and v be binary vectors in $\{0, 1\}^n$. We have the following properties:

- $wt(u \cap v) \leq wt(u)$ and $wt(u \cap v) \leq wt(v)$,
- $u \cap \vec{0}^n = \vec{0}^n$ and $u \cap \vec{1}^n = u$.

Lemma 1.15. *Let u and v be binary vectors in $\{0, 1\}^n$. Then $d(u, v) = wt(u) + wt(v) - 2wt(u \cap v)$.*

Proof. Let U be the number of coordinates in which u is 1 and v is 0, and let V be the number of coordinates in which v is 1 and u is 0. Then $d(u,v) = U + V$. Since $wt(u \cap v)$ represents the number of coordinates in which u and v are both 1, we have:

$$\begin{aligned} d(u,v) &= U + V \\ &= (wt(u) - wt(u \cap v)) + (wt(v) - wt(u \cap v)) \\ &= wt(u) + wt(v) - 2wt(u \cap v). \end{aligned}$$ □

Corollary 1.16. *Let u and v be binary vectors in $\{0,1\}^n$. Then u and v are of the same parity if and only if $d(u,v)$ is even.*

Proof.

$$\begin{aligned} u \text{ and } v \text{ are of the same parity} &\Leftrightarrow wt(u) + wt(v) \text{ is even} \\ &\Leftrightarrow wt(u) + wt(v) - 2wt(u \cap v) \text{ is even} \\ &\Leftrightarrow d(u,v) \text{ is even} \qquad \text{by Lemma 1.15. } \square \end{aligned}$$

For notational convenience, we will often represent a binary vector by its integer representation.

Definition 1.17. Let v be a binary vector in $\{0,1\}^n$ whose coordinates are defined by $v_1 v_2 \dots v_n$. Consider the nonnegative integer i satisfying:

$$i = 2^{n-1}v_1 + 2^{n-2}v_2 + \ldots + 2v_{n-1} + v_n.$$

This integer i is the *integer representation* of the vector v.

Definition 1.18. For any positive integer k, the *binary representation matrix* of order k is the $\lceil \log_2(k+1) \rceil \times k$ matrix in which column i, for $i \in [k]$, is the binary vector of length $\lceil \log_2(k+1) \rceil$ whose integer representation is i. This matrix is denoted BRM_k. For example, the binary representation matrix of order 6 is defined as follows:

$$BRM_6 = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}.$$

1.2 An Introduction to Codes

We will now define a code over a finite field F.

Definition 1.19. Let F be a field. A *code* C over F is a subset of F^n. The elements $u \in C$ are the *codewords* of C. If $F = \{0, 1\}$, then C is a *binary code*.

Definition 1.20. Let $C = \{u_1, u_2, \ldots, u_M\}$, where $u_i \in F^n$ for all $i \in [M]$. The *(code) length* is the length of the vectors, n, whereas the *(code) size* is the number of codewords in the code, $|C| = M$.

In addition to code length and size, a third parameter is usually specified when referring to a code.

Definition 1.21. Let C be an (n, M) code over a field F. The *minimum distance* of C, denoted $d(C)$, is defined to be:

$$d(C) = \min\{d(u, v) \mid u, v \in C, u \neq v\}.$$

If $M = 1$ then $d(C) = \infty$.

Notation 1.22. An (n, M) code is a code with length n and size M. An (n, M, d) code is a code with length n, size M, and minimum distance at least d.

Definition 1.23. Let C be an (n, M) code over a field F, and let $\sigma \in S_n$. Then $\sigma(C)$ is defined to be:

$$\sigma(C) = \{\sigma(u) \mid u \in C\}.$$

Clearly, $\sigma(C)$ is an (n, M) binary code. Moreover, $d(\sigma(C)) = d(C)$.

Definition 1.24. Let C be an (n, M) code over a field F, and let $w \in F^n$. Then $C + w$ is the (n, M) code over F satisfying:

$$C + w = \{u + w \mid u \in C\}.$$

Proposition 1.25. *Let C be an (n, M) code over a field F, and let $w \in F^n$. Then $d(C + w) = d(C)$.*

Proof. To determine the minimum distance of $C + w$, we have:

$$\begin{aligned} d(C + w) &= \min\{d(u, v) \mid u, v \in C + w, u \neq v\} \\ &= \min\{d(x + w, y + w) \mid x, y \in C, x \neq y\} \\ &= \min\{d(x, y) \mid x, y \in C, x \neq y\} \qquad \text{by Lemma 1.9} \\ &= d(C). \end{aligned}$$

□

In the following definitions, we will introduce techniques to create longer or shorter codes from a given code.

Definition 1.26. Let C be an (n, M) code over a field F, and let $v \in F^m$ for some $m \in \mathbb{Z}^+$. Then C_v is the $(n + m, M)$ code over F defined as follows:

$$C\_v = \{u\_v \mid u \in C\}.$$

It is clear that $d(C\_v) = d(C)$.

Definition 1.27. Let C be an $(n, M, 2)$ code over a field F, and let $i \in [n]$. The *punctured code* $C_{(i)}$ is defined to be the $(n-1, M)$ code over F containing all codewords of C with the ith coordinate removed. That is:

$$C_{(i)} = \{u_1 u_2 \dots u_{i-1} u_{i+1} \dots u_n \mid u \in C\}.$$

Clearly, $d(C) - 1 \leq d(C_{(i)}) \leq d(C)$.

Definition 1.28. Let C be an (n, M) code over a field F. Moreover, let $x \in F$ and $i \in [n]$. The *shortened code* $C_{i,x}$ is defined to be the $(n-1, M' \leq M)$ code over F containing all codewords of C with the field element x in coordinate i, then punctured at i. That is:

$$C_{i,x} = \{u_1 u_2 \dots u_{i-1} u_{i+1} \dots u_n \mid u_1 u_2 \dots u_{i-1} x u_{i+1} \dots u_n \in C\}.$$

Clearly, $d(C_{i,x}) \geq d(C)$.

Codes are used in real world applications every day. In practice, information is sent in the form of codewords from some (n, M) code C over a field F, where different codewords represent different meanings. The required codeword $u \in C$ is sent over a channel, over which errors can occur in any and all of the coordinates that change the coordinate to a different field element. Therefore, it is possible that one or more errors have transformed the original codeword into a vector in F^n that is not in the code. When a vector $x \in F^n$ is received at the other end of this channel, x is then decoded to some codeword $v \in C$, generally the one that maximizes the likelihood of receiving x (in the hopes that $u = v$). Assuming the probability of error in every coordinate is the same, this codeword v will be the $v \in C$ that minimizes $d(x, v)$ with ties broken arbitrarily. Formally, this is expressed as in the following definition.

Definition 1.29. Let C be an (n, M) code over a field F. A *decoder* is a function $\mathcal{D} : F^n \to C$. The decoder $\mathcal{D}$ is a *nearest codeword decoder* if it satisfies:

$$\mathcal{D}(x) = v \Rightarrow d(x, v) = \min\{d(x, v') \mid v' \in C\}.$$

It is clear that the higher the minimum distance of a code, the higher the probability that a received vector will be decoded correctly. The result of Corollary 1.31 will specify explicitly the error correcting capability of a code with fixed minimum distance. The following two proofs are similar to those used in [13, Ch. 1, Theorem 2].

Proposition 1.30. *Let C be an (n, M, d) code over a field F, and $t = \left\lfloor \frac{d-1}{2} \right\rfloor$. For any $u, v \in C$ such that $u \neq v$, $S_t(u) \cap S_t(v) = \emptyset$.*

Proof. Suppose, by way of contradiction, that there exists $w \in F^n$ such that $d(u, w) \leq t$ and $d(v, w) \leq t$. We have:

$$\begin{aligned} d(u, v) &\leq d(u, w) + d(w, v) \qquad \text{by the triangle inequality} \\ &\leq \left\lfloor \frac{d-1}{2} \right\rfloor + \left\lfloor \frac{d-1}{2} \right\rfloor \\ &\leq \frac{d-1}{2} + \frac{d-1}{2} \\ &= d - 1, \end{aligned}$$

a contradiction to the minimum distance of C. □

Corollary 1.31. *Let C be an (n, M, d) code over a field F. A nearest codeword decoder will correctly recover any pattern of up to $\left\lfloor \frac{d-1}{2} \right\rfloor$ errors.*

Proof. Let C be an (n, M, d) code over F. Suppose codeword $u \in C$ is sent over a channel and $t \leq \left\lfloor \frac{d-1}{2} \right\rfloor$ errors occur to yield vector $w \in F^n$. Thus $w \in S_t(u)$. Take any $v \in C \setminus \{u\}$. By the result of Proposition 1.30, $w \notin S_t(v)$, implying:

$$d(v, w) > t = d(u, w).$$

Therefore a nearest codeword decoder will decode w to u. □

Definition 1.32. A *constant weight code* C is an (n, M, d) code in which every codeword in C has the same weight w. C can then be referred to as an (n, M, d, w) code.

Definition 1.33. An *even weight code* C is a code in which every codeword $u \in C$ is of even weight. Similarly, an *odd weight code* C is a code in which every codeword $u \in C$ is of odd weight.

For the remainder of this thesis, we will be strictly using binary codes. For binary vectors u and v, note that $u - v = u + v$.

Corollary 1.34. *Even weight binary codes and odd weight binary codes have even minimum distance.*

Proof. Since every codeword is of the same parity, the proof of this corollary follows directly from Corollary 1.16. □

Definition 1.35. The *weight distribution* of an (n, M) binary code C with respect to a vector $u \in \{0,1\}^n$ is the sequence of nonnegative integers $(W_i(u))_{i=0}^n$, where $W_i(u)$ for $i = 0, 1, \ldots, n$ is the number of codewords $v \in C$ such that $d(u, v) = i$. Symbolically, this definition may be represented as:

$$\begin{aligned} W_i(u) &= \sum_{\substack{v \in C \\ d(u,v)=i}} 1 \\ &= \sum_{\substack{v \in C \\ wt(u+v)=i}} 1. \end{aligned}$$

When u is not specified, it will be assumed that $u = \vec{0}^n$. We will denote $(W_i(\vec{0}^n))_{i=0}^n$ as simply $(W_i)_{i=0}^n$.

Let C be an (n, M) binary code, $u \in \{0,1\}^n$, and $\sigma \in S_n$. Moreover, let $(W_i(u))_{i=0}^n$ denote the weight distribution of C with respect to u. The weight distribution satisfies the following properties:

- $\sum_{i=0}^{n} W_i = M$,
- The weight distribution of $\sigma(C)$ with respect to u is also $(W_i(u))_{i=0}^n$,
- Letting $(W_i')_{i=0}^n$ denote the weight distribution of $C + u$, $(W_i')_{i=0}^n = (W_i(u))_{i=0}^n$.

Definition 1.36. The *distance distribution* of an (n, M) binary code C is the list of nonnegative, rational numbers $(A_i)_{i=0}^n$, where A_i represents the average number of pairs of codewords in C at distance i from each other. Thus each A_i satisfies:

$$\begin{aligned} A_i &= \frac{1}{M} \sum_{\substack{u,v \in C \\ d(u,v)=i}} 1 \\ &= \frac{1}{M} \sum_{u \in C} \sum_{\substack{v \in C \\ d(u,v)=i}} 1 \\ &= \frac{1}{M} \sum_{u \in C} W_i(u). \end{aligned}$$

The following statements are clear for any (n, M, d) binary code with distance distribution $(A_i)_{i=0}^n$:

- $A_0 = 1$,
- $A_i \geq 0$,
- $A_i = 0$ for $i \in [d-1]$,
- $\sum_{i=0}^{n} A_i = M$, or equivalently, $\sum_{i=d}^{n} A_i = M - 1$.

Definition 1.37. Let C and D be (n, M, d) binary codes. C and D are *equivalent* if $C = \sigma(D) + v$ for some $\sigma \in S_n$ and $v \in \{0,1\}^n$. C is *unique* if every (n, M, d) binary code is equivalent to C.

It is clear that code equivalence is reflexive, symmetric, and transitive. Therefore, it is an equivalence relation on the set of all binary codes.

The notion of equivalent codes will be very important throughout this thesis. One very important property of equivalent codes is that they share the same distance distribution.

Lemma 1.38. *Let C and D be equivalent (n, M, d) binary codes. Then C and D have the same distance distributions.*

Proof. Let $\sigma \in S_n$ and $w \in \{0,1\}^n$ satisfy $D = \sigma(C) + w$. Moreover, let $(A_i^C)_{i=0}^n$ and $(A_i^D)_{i=0}^n$ denote the distance distributions of C and D, respectively. For any $i \in [n]$, we have:

$$
\begin{aligned}
A_i^C &= \frac{1}{M} \sum_{\substack{u,v \in C \\ d(u,v)=i}} 1 \\
&= \frac{1}{M} \sum_{\substack{u,v \in C \\ d(\sigma(u),\sigma(v))=i}} 1 \\
&= \frac{1}{M} \sum_{\substack{u,v \in C \\ d(\sigma(u)+w,\sigma(v)+w)=i}} 1 \qquad \text{by Lemma 1.9} \\
&= \frac{1}{M} \sum_{\substack{x,y \in D \\ d(x,y)=i}} 1 \\
&= A_i^D. \qquad \square
\end{aligned}
$$

1.3 Optimal Binary Codes

Finding binary codes of optimal size when length and minimum distance are held fixed has become a well-known and difficult combinatorial problem. Such optimal codes allow for a maximum number of possible messages to be sent efficiently, while maintaining a specified error correcting ability.

Notation 1.39. The size of the largest possible binary code of length n and minimum distance at least d is denoted $A(n, d)$. An (n, M, d) binary code in which $M = A(n, d)$ is *optimal*.

The following trivial values of $A(n, d)$ can easily be verified for any positive integers n and d:

- $A(n, 1) = 2^n$,
- $A(n, 2) = \dfrac{2^n}{2} = 2^{n-1}$,
- $A(n, d) = 1$ if $d > n$.

When d is sufficiently large ($\dfrac{2n}{3} < d \leq n$), it is known that $A(n, d) = 2$. Proposition 1.40 proves this result.

Proposition 1.40. *$A(n, d) = 2$ for $n, d \in \mathbb{Z}^+$ with $\dfrac{2n}{3} < d \leq n$.*

Proof. Fix n and d as specified. First, we will show the existence of an $(n, 2, d)$ binary code. Consider any $v \in \{0,1\}^n$ with $wt(v) = d$ (if $d = n$ then $v = \vec{1}^n$). The code $\{\vec{0}^n, v\}$ is an example of an $(n, 2, d)$ binary code.

We now show that an $(n, 3, d)$ code can't exist. Suppose, by way of contradiction, that it does, and is defined by $C = \{u, v, w\}$. We have:

$$\begin{aligned}
d(u,v) &\leq d(u, w+\vec{1}^n) + d(v, w+\vec{1}^n) \qquad \text{by the triangle inequality} \\
&= wt(u+w+\vec{1}^n) + wt(v+w+\vec{1}^n) \\
&= d(u+w, \vec{1}^n) + d(v+w, \vec{1}^n) \\
&= wt(u+w) + wt(\vec{1}^n) - 2wt((u+w) \cap \vec{1}^n) \\
&\qquad + wt(v+w) + wt(\vec{1}^n) - 2wt((v+w) \cap \vec{1}^n) \\
&= wt(u+w) + wt(\vec{1}^n) - 2wt(u+w) \\
&\qquad + wt(v+w) + wt(\vec{1}^n) - 2wt(v+w) \\
&= 2n - d(u,w) - d(v,w) \\
&< 2n - \frac{2n}{3} - \frac{2n}{3} \\
&= \frac{2n}{3},
\end{aligned}$$

a contradiction to the minimum distance of C. □

In fact, this result is tight with respect to d, as will be proven in Proposition 1.41.

Proposition 1.41. *$A(n, d) \geq 3$ for $n, d \in \mathbb{Z}^+$ with $d \leq \left\lfloor \frac{2n}{3} \right\rfloor$.*

Proof. Fix n and d as specified, and define $t = \left\lceil \frac{n}{3} \right\rceil$. We will prove this result by exhibiting an $(n, 3, d)$ binary code. Consider $C = \{u_1, u_2, u_3\}$, where the vectors u_i are defined as follows:

$$u_1 = \sum_{i=1}^{t} \mu_i^n, \quad u_2 = \sum_{i=t+1}^{2t} \mu_i^n, \quad u_3 = \sum_{i=2t+1}^{n} \mu_i^n.$$

For any $i, j \in [3]$ with $i \neq j$ we have:

$$u_i \cap u_j = \vec{0}^n,$$

and hence by Lemma 1.15:

$$d(u_i, u_j) = wt(u_i) + wt(u_j).$$

The minimum distance of C is calculated as follows:

$$\begin{aligned} d(C) &= \min\{d(u_1,u_2), d(u_1,u_3), d(u_2,u_3)\} \\ &= \min\{wt(u_1)+wt(u_2), wt(u_1)+wt(u_3), wt(u_2)+wt(u_3)\} \\ &= \min\left\{2\left\lceil\frac{n}{3}\right\rceil, 2\left\lceil\frac{n}{3}\right\rceil, \left\lceil\frac{n}{3}\right\rceil + \left(n - 2\left\lceil\frac{n}{3}\right\rceil\right)\right\} \\ &= n - \left\lceil\frac{n}{3}\right\rceil \\ &= \left\lfloor\frac{2n}{3}\right\rfloor. \end{aligned}$$

Therefore C is an $(n, 3, d)$ binary code. □

Propositions 1.42 and 1.43 show how values of $A(n, d)$ can be bounded above by $A(n+1, d)$ and $A(n-1, d)$.

Proposition 1.42. *$A(n, d) \leq A(n+1, d)$ for any $n, d \in \mathbb{Z}^+$.*

Proof. Let C be an (n, M, d) binary code of optimal size $M = A(n, d)$. Consider the code C_0, an $(n+1, M, d)$ binary code. The existence of this code proves $A(n+1, d) \geq A(n, d)$. □

Proposition 1.43. *$A(n, d) \leq 2A(n-1, d)$ for any $n, d \in \mathbb{Z}^+$.*

Proof. Let C be an (n, M, d) binary code of optimal size $M = A(n, d)$. Consider the shortened codes $C_{i,0}$ and $C_{i,1}$ for any $i \in [n]$. It is clear that $|C_{i,0}| \geq \frac{M}{2}$ or $|C_{i,1}| \geq \frac{M}{2}$. Let D be the code formed by taking the larger of $C_{i,0}$ and $C_{i,1}$. D is clearly an $(n-1, M', d)$ binary code with size $M' \geq \frac{M}{2}$, and hence:

$$A(n-1, d) \geq \frac{A(n, d)}{2}.$$ □

The search of the values of $A(n, d)$ for less trivial n and d is a classic combinatorial problem. The result of Theorem 1.44 will show that we need only find $A(n, d)$ for even d.

Theorem 1.44. *$A(n, d) = A(n+1, d+1)$ for $n, d \in \mathbb{Z}^+$ with $d \leq n$ and d odd.*

Proof. We will prove equality holds by first proving $A(n,d) \leq A(n+1,d+1)$, then by proving $A(n,d) \geq A(n+1,d+1)$.

Let C be an (n,M,d) binary code of optimal size $M = A(n,d)$. To prove $A(n,d) \leq A(n+1,d+1)$, we will show the existence of an $(n+1,M,d+1)$ binary code. Let C_p defined as follows:

$$C\_p = \{u\_p_u \mid u \in C\}.$$

It is clear that C_p is an $(n+1,M,d)$ binary code. However, C_p is also an even weight code, and therefore $d(C\_p)$ is even. C_p is hence an $(n+1,M,d+1)$ binary code, which completes the first half of the proof.

Let D be an $(n+1,M,d+1)$ binary code of optimal size $M = A(n+1,d+1)$. To prove $A(n,d) \geq A(n+1,d+1)$, we will show the existence of an (n,M,d) binary code. Since d is odd, $d+1 \geq 2$, and hence the punctured code $D_{(i)}$ for any $i \in [n+1]$ is an (n,M) binary code. In addition, since only one coordinate has been punctured, $d(D_{(i)}) \geq d$. Therefore $D_{(i)}$ is an (n,M,d) binary code for any $i \in [n+1]$. □

A table of values of $A(n,d)$ for $n \leq 28$ and $d \leq 14$ (for even d) can be found in [2, Table I]. The listed entries of $A(n,4)$ have been included in Appendix A (included as $A(n-1,3)$). When $n \geq 17$, many of these values are currently unknown. That so few values are actually known is a testament to the difficulty of this problem. However, various bounds have been developed. Lower bounds most often arise due to construction of a specific binary code attaining the given parameters, and upper bounds can be proven through a number of different methods, two of which we will show in the following sections.

Furthermore, many optimal binary codes are unique, that is, every binary code attaining the given parameters is equivalent. For example, consider the $(16,256,6)$, $(15,256,5)$, $(15,128,6)$, $(14,128,5)$, $(14,64,6)$, $(13,64,5)$, $(13,32,6)$, and $(12,32,5)$ binary codes. All are seen to be optimal in [2, Table I]; moreover, all are unique with the exception of the $(12,32,5)$ code [13, pp.74-75].

Östergård, Baicheva, and Kolev developed an algorithm in [18] that can determine the number of nonequivalent binary codes for certain code parameters. The number of nonequivalent $(n,M,3)$ and $(n,M,4)$ binary codes were then calculated for various values of n and M, as seen in [18, Table I].

In addition to bounds on the size of binary codes, another common problem in coding theory is determining the size of the largest possible constant weight binary code attaining given parameters.

Notation 1.45. The size of the largest possible constant weight binary code of length n, minimum distance at least d, and codeword weight w, for $w \leq n$, is denoted $A(n, d, w)$. An (n, M, d, w) constant weight binary code in which $M = A(n, d, w)$ is *optimal*.

The following properties of $A(n, d, w)$ can be easily verified for any positive integers n and d, and nonnegative integer $w \leq n$:

- $A(n, 2, w) = \binom{n}{w}$,
- $A(n, d, w) = 1$ if $d > n$,
- $A(n, d, w) = A(n, d + 1, w)$ for d odd,
- $A(n, d, w) = A(n, d, n - w)$.

The final statement is proven to hold by adding $\vec{1}^n$ to either optimal code. Due to these properties, the search for $A(n, d, w)$ to be restricted to even d and $w \leq \left\lfloor \frac{n}{2} \right\rfloor$. We also make use of the results of Propositions 1.46 and 1.47 (see [13, Ch. 17, Theorem 1]):

Proposition 1.46. *$A(n, d, w) = 1$ for $n, d \in \mathbb{Z}^+$ and $w \in \mathbb{Z}^+ \cup \{0\}$ satisfying $2w < d \leq n$.*

Proof. The existence of a code meeting this bound is trivial (as it contains a single codeword). Thus we need only prove $A(n, d, w) \leq 1$. To prove $A(n, d, w) \leq 1$, assume by way of contradiction there are two vectors $u, v \in \{0, 1\}^n$ satisfying $wt(u) = wt(v) = w$ and $d(u, v) \geq d$. Then, by the triangle inequality:

$$\begin{aligned} d(u, v) &\leq d(u, \vec{0}^n) + d(\vec{0}^n, v) \\ &= wt(u + \vec{0}^n) + wt(u + \vec{0}^n) \\ &= wt(u) + wt(v) \\ &< \frac{d}{2} + \frac{d}{2} \\ &= d, \end{aligned}$$

a contradiction to the minimum distance of d, and hence $A(n, d, w) = 1$. □

Proposition 1.47. *$A(n, 2w, w) = \left\lfloor \frac{n}{w} \right\rfloor$ for $n, w \in \mathbb{Z}^+$.*

Proof. To show $A(n, 2w, w) \geq \left\lfloor \frac{n}{w} \right\rfloor$, we need only show the existence of an $(n, t, 2w, w)$ binary code where $t = \left\lfloor \frac{n}{w} \right\rfloor$. Consider $C = \{v_1, v_2, \ldots, v_t\}$, where $v_i \in \{0,1\}^n$ is the vector with a 1 in coordinates $(i-1)w+1, (i-1)w+2, \ldots, (i-1)w+w$. It is clear that these vectors all exist as, for $i \leq t$:

$$(i-1)w + w \leq \left(\left\lfloor \frac{n}{w} \right\rfloor - 1 \right) w + w \leq n - w + w = n.$$

Moreover, $d(v_i, v_j) = 2w$ for $i \neq j$. Therefore C is an $(n, t, 2w, w)$ binary code.

To show $A(n, 2w, w) \leq \left\lfloor \frac{n}{w} \right\rfloor$, assume by way of contradiction there exists an $(n, t+1, 2w, w)$ binary code D, with t defined as above. Let $D = \{v_1, v_2, \ldots, v_{t+1}\}$. For any $i, j \in [t+1], i \neq j$, by Lemma 1.15 we have:

$$\begin{aligned} wt(v_i \cap v_j) &= \frac{wt(v_i) + wt(v_j) - d(v_i, v_j)}{2} \\ &\leq \frac{w + w - 2w}{2} \\ &= 0. \end{aligned}$$

Thus $wt(v_i \cap v_j) = 0$, and hence $\sum_{i=1}^{t+1} wt(v_i) \leq n$. However:

$$\sum_{i=1}^{t+1} wt(v_i) = \sum_{i=1}^{t+1} w = \left(\left\lfloor \frac{n}{w} \right\rfloor + 1 \right) w > n,$$

a contradiction. Therefore $A(n, 2w, w) = \left\lfloor \frac{n}{w} \right\rfloor$. □

Agrell, Vardy, and Veger improved bounds on $A(n, d, w)$ for many values in [1]. This article includes tables of $A(n, d, w)$ for values of and bounds on $A(n, d, w)$ for even values of d between 4 and 14. Recall that for odd values of d, $A(n, d, w) = A(n, d+1, w)$. Later in this thesis, we will refer back to these tables.

The following theorem and corollary show how the weight and distance distribution of any binary code can be bounded above by the sizes of appropriate optimal constant weight codes.

Theorem 1.48. *Let C be an (n, M, d) binary code and fix $u \in \{0,1\}^n$. Moreover, let $W_i(u)$ be the weight distribution of C with respect to u. Then $W_i(u) \leq A(n, d, i)$ for $i = 0, 1, \ldots, n$.*

Proof. Assume, by way of contradiction, that $W_i(u) > A(n, d, i)$ for some i. Let $k = W_i(u)$. Next, let $\{v_1, v_2, \ldots, v_k\} \subseteq C$ be the set of codewords at distance i from u. Consider the set of vectors $D = \{w_1, w_2, \ldots, w_k\}$ defined by $w_j = v_j + u$ for $j \in [k]$. We claim that D is an (n, k, d, i) constant weight binary code with code size $k > A(n, d, i)$, a contradiction.

To show that w_j has weight i for all $j \in [k]$, take arbitrary $j \in [k]$. We have:

$$\begin{aligned} wt(w_j) &= wt(v_j + u) \\ &= d(v_j, u) \\ &= i. \end{aligned}$$

Thus D is constant weight with weight i. Finally, for $j_1, j_2 \in [k], j_1 \neq j_2$:

$$\begin{aligned} d(w_{j_1}, w_{j_2}) &= d(v_{j_1} + u, v_{j_2} + u) \\ &= d(v_{j_1}, v_{j_1}) \qquad \text{by Lemma 1.9} \\ &\geq d. \end{aligned}$$

□

Corollary 1.49. *Let C be an (n, M, d) binary code, and let $(A_i)_{i=0}^n$ be the distance distribution of C. Then $A_i \leq A(n, d, i)$ for all $i \in \{0, 1, \ldots, n\}$.*

Proof. For any $i \in \{0, 1, \ldots, n\}$ we have:

$$\begin{aligned} A_i &= \frac{1}{M} \sum_{u \in C} W_i(u) \\ &\leq \frac{1}{M} \sum_{u \in C} A(n, d, i) \qquad \text{by Theorem 1.48} \\ &= \frac{1}{M}(M) A(n, d, i) \\ &= A(n, d, i). \end{aligned}$$

□

We will now introduce two useful upper bounds on $A(n, d)$, the sphere-packing bound and Delsarte's linear programming bound.

1.3.1 The Sphere-Packing Bound

The sphere-packing bound is a well-known bound on optimal code size, and can be useful when d is small. While the bound can be calculated for codes over any field, we will offer the binary simplification. This bound will then give rise to the notion of perfect codes. The proof of the sphere-packing bound used follows that in [20, Theorem 4.3].

Theorem 1.50 (Sphere-Packing Bound)**.** *For any $n, d \in \mathbb{Z}^+$ with $d \leq n$, let $t = \left\lfloor \frac{d-1}{2} \right\rfloor$. Then:*

$$A(n,d) \leq \left\lfloor 2^n \left(\sum_{i=0}^{t} \binom{n}{i} \right)^{-1} \right\rfloor.$$

Proof. Let C be any (n, M, d) binary code, with optimal code size $M = A(n,d)$. Proposition 1.30 implies that the spheres $S_t(u)$ are distinct for all $u \in C$. Thus we have the following:

$$\begin{aligned} 2^n &\geq \sum_{u \in C} |S_t(u)| \\ &= \sum_{u \in C} |\{v \in \{0,1\}^n \mid d(u,v) \leq t\}| \\ &= \sum_{u \in C} \sum_{i=0}^{t} \binom{n}{i}, \end{aligned}$$

since for a fixed i, there are $\binom{n}{i}$ ways to choose i of the n coordinates of u to be different. Thus we have:

$$2^n \geq A(n,d) \sum_{i=0}^{t} \binom{n}{i}.$$

Since $A(n,d)$ must be an integer, we arrive at the desired result. □

Definition 1.51. An (n, M, d) binary code C is called *perfect* if it attains the sphere-packing bound.

One well-known class of codes are the class of *(binary) Hamming codes*, as defined in [13, p.25]. The Hamming code is defined for any $n \in \mathbb{Z}^+$ satisfying $n \geq 2$, and is a $(2^n - 1, 2^{2^n - n - 1}, 3)$ binary code. Note that as the Hamming code is a linear code (not defined in this thesis), its second

parameter is usually expressed in terms of code dimension, rather than code size. The dimension of a linear code of size M over a field F is defined to be $\log_{|F|}(M)$. The Hamming code attains the sphere-packing bound, and therefore gives rise to the following result.

Theorem 1.52. *$A(2^n - 1, 3) = 2^{2^n - n - 1}$ for any $n \in \mathbb{Z}^+$.*

Proof. First, note that $A(2^1 - 1, 3) = A(1, 3) = 1 = 2^{2^1 - 1 - 1}$ is trivially satisfied. We now fix $n \in \mathbb{Z}^+$ with $n \geq 2$. The sphere-packing bound (Theorem 1.50) implies:

$$\begin{aligned} A(2^n - 1, 3) &\leq \left\lfloor 2^{2^n - 1} \left(\sum_{i=0}^{1} \binom{2^n - 1}{i} \right)^{-1} \right\rfloor \\ &= \frac{2^{2^n - 1}}{1 + 2^n - 1} \\ &= 2^{2^n - n - 1}. \end{aligned}$$

However, the existence of the Hamming code implies that $A(2^n - 1, 3) = 2^{2^n - n - 1}$. □

1.3.2 Delsarte's Linear Programming Bound

Another useful upper bound on $A(n, d)$ is Delsarte's linear programming bound, developed by Delsarte in [8] and [7]. Recall that, for any (n, M, d) binary code with distance distribution $(A_i)_{i=0}^n$:

$$\sum_{i=d}^{n} A_i = M - 1.$$

These A_i will thus be treated as variables in Delsarte's linear program. The result of Theorem 1.55 will produce the constraints of this linear program. We closely follow the notation used in [5].

Definition 1.53. The *Krawtchouk polynomial* $K_k(i)$, for $i, k \in \{0, 1, \ldots, n\}$, is defined to be:

$$K_k(i) = \sum_{j=0}^{k} (-1)^j \binom{n-i}{k-j} \binom{i}{j}.$$

The Krawtchouk polynomial can be rewritten as we will see in Theorem 1.54. The proof given is similar to that used in [14, Lemma 3.5.5].

Theorem 1.54. *For any $w \in \{0,1\}^n$ satisfying $wt(w) = i$, the Krawtchouk polynomial, for $k \in \{0, 1, \ldots, n\}$, satisfies:*

$$K_k(i) = \sum_{\substack{x \in \{0,1\}^n \\ wt(x)=k}} (-1)^{<w,x>}.$$

Proof. Take arbitrary $w \in \{0,1\}^n$ with $wt(w) = i$. Define $K \subseteq \{0,1\}^n$ to be:

$$K = \{x \in \{0,1\}^n \mid wt(x) = k\}.$$

Next, define $K_j \subseteq K$ for $j \in \{0, 1, \ldots, \max\{i,k\}\}$ to be:

$$K_j = \{x \in K \mid wt(w \cap x) = j\}.$$

It is clear that the K_j form a partition of K. This implies:

$$\begin{aligned}
\sum_{\substack{x \in \{0,1\}^n \\ wt(x)=k}} (-1)^{<w,x>} &= \sum_{\substack{x \in \{0,1\}^n \\ wt(x)=k}} (-1)^{wt(w \cap x)} \\
&= |K_0| - |K_1| + |K_2| - \ldots \pm |K_{\max\{i,k\}-1}| \mp |K_{\max\{i,k\}}| \\
&= \sum_{j=0}^{\max\{i,k\}} (-1)^j |K_j|. \qquad (1.1)
\end{aligned}$$

For any $j \in \{0, 1, \ldots, \max\{i,k\}\}$, we determine $|K_j|$. Fix j and consider $x \in K_j$. Of the i coordinates where w is 1, we have j choices of coordinates where we can let x be 1. Then, of the $n - i$ coordinates where w is 0, we have $n - j$ choices of coordinates where we can let x be 1. From equation (1.1):

$$\begin{aligned}
\sum_{\substack{x \in \{0,1\}^n \\ wt(x)=k}} (-1)^{<w,x>} &= \sum_{j=0}^{\max\{i,k\}} (-1)^j |K_j| \\
&= \sum_{j=0}^{\max\{i,k\}} (-1)^j \binom{i}{j} \binom{n-i}{k-j} \\
&= \sum_{j=0}^{k} (-1)^j \binom{i}{j} \binom{n-i}{k-j} \\
&= K_k(i).
\end{aligned}$$

□

The constraints of Delsarte's linear program are proven to hold with a proof similar to that used in [5, Theorem 3], but rearranged and expressed in the same manner as in [2].

Theorem 1.55. *Let C be an (n, M, d) binary code with distance distribution $(A_i)_{i=0}^n$. Then, for $k \in \{0, 1, \ldots, n\}$:*

$$\sum_{i=d}^{n} A_i K_k(i) \geq -\binom{n}{k}.$$

Proof. For arbitrary $w \in \{0,1\}^n$ satisfying $wt(w) = i$, we have:

$$\begin{aligned}
\sum_{i=d}^{n} A_i K_k(i) &= \sum_{i=0}^{n} A_i K_k(i) - A_0 K_k(0) \\
&= \sum_{i=0}^{n} \left(\frac{1}{M} \sum_{\substack{u,v \in C \\ d(u,v)=i}} 1 \right) \left(\sum_{\substack{x \in \{0,1\}^n \\ wt(x)=k}} (-1)^{<w,x>} \right) \\
&\qquad - (1) \left(\sum_{\substack{x \in \{0,1\}^n \\ wt(x)=k}} (-1)^{<\vec{0},x>} \right).
\end{aligned}$$

Moreover, since $wt(u+v) = d(u,v) = i$:

$$\begin{aligned}
\sum_{i=d}^{n} A_i K_k(i) &= \frac{1}{M} \sum_{i=0}^{n} \sum_{\substack{u,v \in C \\ d(u,v)=i}} \sum_{\substack{x \in \{0,1\}^n \\ wt(x)=k}} (-1)^{<u+v,x>} - \sum_{\substack{x \in \{0,1\}^n \\ wt(x)=k}} (-1)^0 \\
&= \frac{1}{M} \sum_{\substack{x \in \{0,1\}^n \\ wt(x)=k}} \sum_{i=0}^{n} \sum_{\substack{u,v \in C \\ d(u,v)=i}} (-1)^{<u,x>} (-1)^{<v,x>} - \binom{n}{k} \\
&= \frac{1}{M} \sum_{\substack{x \in \{0,1\}^n \\ wt(x)=k}} \sum_{u,v \in C} (-1)^{<u,x>} (-1)^{<v,x>} - \binom{n}{k}
\end{aligned}$$

$$= \frac{1}{M} \sum_{\substack{x \in \{0,1\}^n \\ wt(x)=k}} \left(\sum_{u \in C} (-1)^{<u,x>} \right) \left(\sum_{v \in C} (-1)^{<v,x>} \right) - \binom{n}{k}$$

$$= \frac{1}{M} \sum_{\substack{x \in \{0,1\}^n \\ wt(x)=k}} \left(\sum_{u \in C} (-1)^{<u,x>} \right)^2 - \binom{n}{k} \tag{1.2}$$

$$\geq -\binom{n}{k}. \qquad \square$$

In the case where M is odd, these constraints can be strengthened. This will be seen in the following corollary, a result first shown in [5, Theorem 5].

Corollary 1.56. *Let C be an (n, M, d) code with M odd. Then, for $k \in \{0, 1, \ldots, n\}$:*

$$\sum_{i=d}^{n} A_i K_k(i) \geq \frac{1-M}{M} \binom{n}{k}.$$

Proof. Recall from equation (1.2) in the proof of Theorem 1.55:

$$\sum_{i=d}^{n} A_i K_k(i) \geq \frac{1}{M} \sum_{\substack{x \in \{0,1\}^n \\ wt(x)=k}} \left(\sum_{u \in C} (-1)^{<u,x>} \right)^2 - \binom{n}{k}.$$

If M is odd, $\sum_{u \in C} (-1)^{<u,x>}$ is odd and thus nonzero. Hence we reach the following conclusion:

$$\begin{aligned} \sum_{i=d}^{n} A_i K_k(i) &\geq \frac{1}{M} \sum_{\substack{x \in \{0,1\}^n \\ wt(x)=k}} 1 - \binom{n}{k} \\ &= \frac{1}{M} \binom{n}{k} - \binom{n}{k} \\ &= \frac{1-M}{M} \binom{n}{k}. \end{aligned}$$

$\square$

Delsarte's linear programming bound (Version 1) of $A(n, d)$ is then stated in Theorem 1.57.

Theorem 1.57. *[Delsarte's Linear Programming Bound (Version 1)] $A(n,d) \leq 1+\lfloor M^* \rfloor$, where M^* is the optimal objective value of the following linear program:*

Maximize $M^* = \sum_{i=d}^{n} A_i$ *subject to:*

$$\sum_{i=d}^{n} A_i K_k(i) \geq -\binom{n}{k} \qquad \text{for } k \in \left\{0, 1, \ldots, \left\lfloor \frac{n}{2} \right\rfloor\right\},$$

$$A_i \geq 0 \qquad \text{for } i \in \{d, d+1, \ldots, n\}.$$

However, this linear program can indeed be further simplified. The following theorem will show how all odd-indexed variables of the linear program can be eliminated.

Theorem 1.58. *Let $d \in \mathbb{Z}^+$ be even. Any (n, M, d) binary code can be transformed into an (n, M, d) even weight binary code.*

Proof. Let C be an (n, M, d) binary code, and take any $i \in [n]$. Since d is even, we have $d \geq 2$. Thus we consider the $(n-1, M, d-1)$ code $C_{(i)}$. We now (as in the first half of the proof of Theorem 1.44) create the (n, M, d) even weight code $C_{(i)}{\_}p$ that results from concatenating p_w with every codeword w of $C_{(i)}$. Since $d-1$ is odd, $C_{(i)}{\_}p$ is an (n, M, d) even weight binary code. □

We now have, by Theorem 1.44, that the search for the values of $A(n,d)$ can be restricted to even d. Moreover, Theorem 1.58 implies that the search with even d can be restricted to codes in which $A_i = 0$ for odd i. Therefore, the first version of Delsarte's linear programming bound given in Theorem 1.57 can be restricted as in Theorem 1.59.

Theorem 1.59. *[Delsarte's Linear Programming Bound (Version 2)] Let $d \in \mathbb{Z}^+$ be even. $A(n,d) \leq 1 + \lfloor M^* \rfloor$, where M^* is the optimal objective value of the following linear program:*

Maximize $\sum_{i=d/2}^{\lfloor n/2 \rfloor} A_{2i}$ *subject to:*

$$\sum_{i=d/2}^{\lfloor n/2 \rfloor} A_{2i} K_k(2i) \geq -\binom{n}{k} \qquad \text{for } k \in \left\{0, 1, \ldots, \left\lfloor \frac{n}{2} \right\rfloor\right\},$$

$$A_{2i} \geq 0 \qquad \text{for } i \in \left\{\frac{d}{2}, \frac{d}{2}+1, \ldots, \left\lfloor \frac{n}{2} \right\rfloor\right\}.$$

Additional bounds can also be imposed, which can help lower the bound. For example, if $A(n, d, w)$ is known for some (or all) w, the result of Corollary 1.49 can be imposed as a constraint (or constraints). This will be seen in Theorem 1.60 in the following section.

1.4 Applications of Delsarte's Linear Programming Bound

Delsarte's linear programming bound has been crucial in determining many new bounds for $A(n, d)$. One result that will be used often throughout this thesis is the result $A(9, 4) = 20$ (and hence by Theorem 1.44 $A(8, 3) = 20$). This result was first proved in [5, Theorem 6], and our proof will be analogous.

Theorem 1.60. $A(9, 4) = 20$.

Proof. To prove that $A(9, 4) \geq 20$, we will exhibit a $(9, 20, 4)$ binary code. Let C be the code where each codeword is a column of the following matrix:

$$\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0
\end{pmatrix}.$$

The vectors in C can also be represented by their integer representations as

follows:

$$C = \{30, 57, 77, 83, 102, 149, 160, 175, 202, 252, 257, 300, 311, 340,\\ 363, 390, 408, 479, 485, 498\}.$$

It is easily verified that C is a $(9, 20, 4)$ binary code.

We will now apply Version 2 of Delsarte's linear programming bound given in Theorem 1.59. We will also be using the result of Proposition 1.46 (with Corollary 1.48) to impose the additional constraint $A_8 \leq 1$. We see that:

$$A(9, 4) \leq 1 + \lfloor \max\{M^*\} \rfloor,$$

where M^* is the optimal objective value of the following linear program:

Maximize $A_4 + A_6 + A_8$ subject to:

$$\begin{aligned} A_4 + A_6 + A_8 &\geq -1, \\ A_4 - 3A_6 - 7A_8 &\geq -9, \\ -4A_4 + 20A_8 &\geq -36, \\ -4A_4 + 8A_6 - 28A_8 &\geq -84, \\ 6A_4 - 6A_6 + 14A_8 &\geq -126, \\ A_8 &\leq 1, \\ A_4, A_6, A_8 &\geq 0. \end{aligned}$$

Upon finding the optimal objective value of this linear program, we obtain:

$$\begin{aligned} A(9, 4) &\leq 1 + \left\lfloor \frac{61}{3} \right\rfloor \\ &= 21. \end{aligned}$$

We now claim that a $(9, 21, 4)$ binary code can't exist. Assume, by way of contradiction, that C is a $(9, 21, 4)$ binary code with distance distribution $(A_i)_{i=0}^{9}$. Since 21 is odd, the result of Corollary 1.56 allows us to transform our first five constraints to:

$$\begin{aligned} A_4 K_k(4) + A_6 K_k(6) + A_8 K_k(8) &\geq \frac{1 - 21}{21} \binom{9}{k} \\ \Leftrightarrow \left(\frac{21}{20} A_4\right) K_k(4) + \left(\frac{21}{20} A_6\right) K_k(6) + \left(\frac{21}{20} A_8\right) K_k(8) &\geq -\binom{9}{k}. \end{aligned}$$

Also, by definition of A_i, we can also similarly transform our sixth constraint. Since $\sum_{\substack{u,v\in C \\ d(u,v)=8}} 1$ must be even (due to the symmetry of every pair of codewords u and v), we have:

$$\begin{aligned} A_8 \leq 1 &\Leftrightarrow A_8 \leq \frac{20}{21} \\ &\Leftrightarrow \left(\frac{21}{20}A_8\right) \leq 1. \end{aligned}$$

Thus, the optimal solution of the new linear program is simply $\frac{20}{21}$ times the original optimal solution. Therefore:

$$\begin{aligned} A(9,4) &\leq 1 + \left\lfloor \left(\frac{20}{21}\right)\left(\frac{61}{3}\right) \right\rfloor \\ &= 20. \end{aligned}$$

□

The optimal $(9, 20, 4)$ code is not unique, as 2 nonequivalent $(9, 20, 4)$ codes were exhibited in [13, p.57]. In fact, there exist exactly 3, as seen in [18, Table I].

Recall the result of Theorem 1.52, $A(n, d) = A(2^n - 1, 3) = 2^{2^n-n-1}$ for $n \in \mathbb{Z}^+$. This result can be generalized as in Theorem 1.61. This will be stated without proof, but is the main result proved by Best and Brouwer in [4]. The 0, 1, 2, and 3 times shortened Hamming codes are the optimal codes that attain this value, and it was proven to hold as an upper bound through the use of Delsarte's linear programming bound.

Theorem 1.61. *Let $i, n \in \mathbb{Z}^+$ satisfy $n \geq 3$ and $i \in [4]$. Then:*

$$A(2^n - i, 3) = 2^{2^n-n-i}.$$

The result of Theorem 1.61 allows for the values of $A(n, 3)$ to be calculated for $28 \leq n \leq 31$, which are included in Appendix A.

Chapter 2

Graph Theory

In this chapter we will review parts of graph theory needed throughout this thesis. The definitions and notation closely follow those used in [22]. We will also introduce the concept of vertex coloring, and introduce three algorithms that solve various vertex coloring problems.

2.1 Elementary Concepts

Definition 2.1. A *graph* G is defined by a *vertex set* and an *edge set*, where elements of the edge set are unordered pairs of vertices.

Notation 2.2. For any graph G, $V(G)$ denotes the vertex set of G, and $n(G) = |V(G)|$, the number of vertices of G. In addition $E(G)$ denotes the edge set of G, and $e(G) = |E(G)|$, the number of edges of G.

Definition 2.3. The *null graph* is the graph whose vertex set and edge set are both empty.

Definition 2.4. Let $u, v \in V(G)$ and $e = \{u, v\} \in E(G)$. Then u and v are the *endpoints* of e, and u and v are *incident* to e. Equivalently, u and v are *adjacent* to each other. For notational convenience, we will commonly write an edge $\{u, v\}$ as uv.

Definition 2.5. In a graph G, a *loop* is an edge $e \in E(G)$ where both endpoints of e are the same vertex $v \in V(G)$. *Multiple edges* are edges $e_1, e_2 \in E(G)$ satisfying $e_1 = e_2$. If G has no loops or multiple edges, then G is a *simple graph*.

Definition 2.6. A *finite graph* is a graph with both its vertex set and edge set finite.

For the remainder of this paper, we will assume all graphs are finite, simple graphs.

Definition 2.7. Consider a graph G, and let $v \in V(G)$. The *neighbourhood* of v, denoted $N_G(v)$, is the set of all vertices adjacent to V. That is:

$$N_G(v) = \{u \in V(G) \mid uv \in E(G)\}.$$

The *degree* of vertex v, denoted $d_G(v)$, is defined to be $|N_G(v)|$. This is the number of edges in $E(G)$ that are incident to v. If, for some $k \in \mathbb{Z}^+ \cup \{0\}$, $d_G(v) = k$ for every $v \in V(G)$, then G is *k-regular*.

The following proposition allows for the number of edges of a k-regular graph to be calculated directly.

Proposition 2.8. *Let G be an n-vertex, k-regular graph. Then:*

$$e(G) = \frac{nk}{2}.$$

Proof. Since summing the degrees of all vertices of a graph counts each edge twice, the following result is clear (and commonly referred to as the degree-sum formula):

$$2e(G) = \sum_{v \in V(G)} d_G(v).$$

However, since G is k-regular, we have the following:

$$\begin{aligned} 2e(G) &= \sum_{v \in V(G)} k \\ &= nk. \end{aligned}$$

□

Definition 2.9. A graph G' is a *subgraph* of a graph G (denoted $G' \subseteq G$) if and only if:

$$\begin{aligned} V(G') &\subseteq V(G), \\ E(G') &\subseteq E(G). \end{aligned}$$

Definition 2.10. Let G be a graph and let $V' \subseteq V(G)$ be a subset of the vertices of G. The subgraph of G *induced by* V', denoted $G[V']$, is the graph satisfying the following:

$$\begin{aligned} V(G[V']) &= V', \\ E(G[V']) &= \{uv \in E(G) \mid u, v \in V'\}. \end{aligned}$$

Definition 2.11. The *complement* of a graph G, denoted $\overline{G}$, is the graph that satisfies the following criteria:

$$V(\overline{G}) = V(G),$$
$$E(\overline{G}) = \{uv \mid u, v \in V(G), uv \notin E(G)\}.$$

Definition 2.12. A *clique* in a graph G is a set of vertices $V' \subseteq V(G)$ such that any two distinct vertices in V' are adjacent in G. The *clique number* of G, denoted $\omega(G)$, is the number of vertices in a clique of largest cardinality of G.

Definition 2.13. An *independent set* in a graph G is a set of vertices $V' \subseteq V(G)$ such that any two distinct vertices in V' are non-adjacent in G. V' is a *maximal independent set* if $V' \cup \{v\}$ is not independent for any $v \in V(G) \backslash V'$. The *independence number* of G, denoted $\alpha(G)$, is the number of vertices in an independent set of largest cardinality G.

Definition 2.14. A graph G is *bipartite* if and only if the vertex set of G can be expressed as $V(G) = V_1 \cup V_2$, where both V_1 and V_2 are independent.

Definition 2.15. The n-vertex *complete graph*, denoted K_n, is the graph such that any two distinct vertices in $V(K_n)$ are adjacent.

Since every pair of distinct vertices in the complete graph are adjacent, $\omega(K_n) = n$ for all $n \in \mathbb{Z}^+$.

Definition 2.16. Let G be a graph. A *walk in* G is an alternating sequence of vertices and edges $W = (v_0, e_1, v_1, e_2, v_2, \ldots, e_n, v_n)$ such that $e_i = v_{i-1}v_i$ for all $i \in [n]$. The *length* of W is n, the number of edges in W. W is denoted a v_0, v_n-walk in G, and the vertices v_0 and v_n are called the *endpoints* of W. The vertices v_i, for $i \in [n-1]$, are called the *internal vertices* of W.

Definition 2.17. A *path* in a graph G is a walk in G such that every edge and every vertex in the walk are distinct. A *cycle* in G is a path whose endpoints are the same vertex.

Definition 2.18. Let $u, v \in V(G)$ be two vertices in a graph G. The *distance* from u to v in G, denoted $d_G(u, v)$, is the length of the shortest u, v-path in G (the path of least length). If there exists no u, v-path in G, then $d_G(u, v) = \infty$.

Let $u, v, w \in V(G)$ for some graph G. Similarly to the Hamming distance, the graph theoretical distance satisfies the following properties:

- $d_G(u, v) = 0$ if and only if $u = v$,
- $d_G(u, v) = d_G(v, u)$,
- $d_G(u, w) \leq d_G(u, v) + d_G(v, w)$ (the triangle inequality).

Definition 2.19. The *kth power* of G is defined for any graph G and $k \in \mathbb{Z}^+$. Denoted G^k, it is the graph that satisfies:

$$V(G^k) = V(G),$$
$$E(G^k) = \{uv \mid u, v \in V(G), d_G(u, v) \leq k\}.$$

2.2 Graph Isomorphisms and Automorphisms

Definition 2.20. An *isomorphism* from a graph G to a graph H is a bijection $\Phi : V(G) \to V(H)$ that satisfies:

$$uv \in E(G) \Leftrightarrow \Phi(u)\Phi(v) \in E(H).$$

If there exists an isomorphism from G to H, then G is *isomorphic* to H. This is denoted $G \cong H$.

If there is an isomorphism Φ from G to H, then the function Φ^{-1} is an isomorphism from H to G. In addition, $\cong$ is both reflexive (through the identity function) and transitive (through functional composition). Therefore, $\cong$ is an equivalence relation on the set of all graphs. It is clear that, in order for $G \cong H$, both of the following must hold:

$$n(G) = n(H),$$
$$e(G) = e(H).$$

Definition 2.21. For any graphs G and H, a *copy* of H in G is a subgraph $G' \subseteq G$ such that $H \cong G'$.

Definition 2.22. An *automorphism* of a graph G is an isomorphism from G onto itself. The *automorphism group* of a graph G is the set of all automorphisms of G, with functional composition as the group operation. It is denoted $Aut(G)$.

Clearly, any permutation of the vertices in the complete graph K_n is an automorphism. Therefore $|Aut(K_n)| = |S_n| = n!$.

Definition 2.23. A graph G is *vertex transitive* if, for all $u, v \in V(G)$, there exists an automorphism $\Phi : V(G) \to V(G)$ such that $\Phi(u) = v$.

The notion of vertex transitivity will be very important later in this thesis. If a graph is vertex transitive, then it "looks the same" from every vertex. What this means is that many times we can prove a result for every vertex in the graph by simply proving the result for a single, arbitrary vertex.

Definition 2.24. Let $V' \subseteq V(G)$ be a set of vertices of a graph G, and $\Phi : V(G) \to V(G)$ be a function of the vertices of G. Then the *image of* V' *under* Φ, denoted $\Phi(V')$, is defined to be:

$$\Phi(V') = \{\Phi(v) \mid v \in V'\}.$$

We will now prove several results about automorphisms that will be needed later in this thesis.

Proposition 2.25. *Let $V' \subseteq V(G)$ be a set of vertices of a graph G, and $\Phi : V(G) \to V(G)$ be an automorphism of G. Then V' is independent if and only if $\Phi(V')$ is independent. Similarly, V' is a clique if and only if $\Phi(V')$ is a clique.*

Proof. Take any $u, v \in V'$. We have the following:

$$\begin{aligned} V' \text{ is independent} &\Leftrightarrow uv \notin E(G) \\ &\Leftrightarrow \Phi(u)\Phi(v) \notin E(G) \qquad \text{since } \Phi \text{ is an automorphism} \\ &\Leftrightarrow \Phi(V') \text{ is independent.} \end{aligned}$$

The result, as applied to a clique rather than an independent set, is proven in an analogous manner. □

Theorem 2.26. *For any graph G, $Aut(G) = Aut(\overline{G})$.*

Proof. We will show that $\Phi : V(G) \to V(G)$ is an automorphism of G if and only if Φ is an automorphism of $\overline{G}$. We first assume that Φ is an automorphism of G. We now take arbitrary $uv \in E(\overline{G})$. We have:

$$\begin{aligned} uv \in E(\overline{G}) &\Leftrightarrow uv \notin E(G) \\ &\Leftrightarrow \Phi(u)\Phi(v) \notin E(G) \\ &\Leftrightarrow \Phi(u)\Phi(v) \in E(\overline{G}), \end{aligned}$$

and thus Φ is an automorphism of $\overline{G}$.

We now assume that Φ is an automorphism of $\overline{G}$, and take arbitrary $uv \in E(G)$. We have:

$$\begin{aligned} uv \in E(G) &\Leftrightarrow uv \notin E(\overline{G}) \\ &\Leftrightarrow \Phi(u)\Phi(v) \notin E(\overline{G}) \\ &\Leftrightarrow \Phi(u)\Phi(v) \in E(G). \end{aligned}$$

Therefore Φ is an automorphism of G. □

Theorem 2.27. *Let $\Phi : V(G_1) \to V(G_2)$ be an isomorphism between graphs G_1 and G_2. Then, for all $u, v \in V(G_1)$, $d_{G_1}(u, v) = d_{G_2}(\Phi(u), \Phi(v))$.*

Proof. This theorem will be proven by induction on $d_{G_1}(u, v)$.

- **Base Case:**
  For all $u, v \in V(G_1)$ such that $d_{G_1}(u, v) = 1$, we prove that $d_{G_2}(\Phi(u), \Phi(v)) = 1$.
  Take arbitrary $u, v \in V(G_1)$ such that $d_{G_1}(u, v) = 1$. Thus $uv \in E(G_1)$, and this case is trivial by the definition of an isomorphism.

- **Inductive Hypothesis:**
  Assume, for some $k \in \mathbb{Z}^+$, that if $u, v \in V(G_1)$ satisfy $d_{G_1}(u, v) \leq k$, then $d_{G_2}(\Phi(u), \Phi(v)) = d_{G_1}(u, v)$.

We now prove the result for $k + 1$. Take arbitrary $u, v \in V(G_1)$ such that $d_{G_1}(u, v) = k + 1$ (if $d_{G_1}(u, v) < k + 1$, then the statement is true by the inductive hypothesis).

First, to show that $d_{G_2}(\Phi(u), \Phi(v)) \leq k + 1$, we will show the existence of a $\Phi(u), \Phi(v)$-path in G_2 of length at most $k + 1$. Let P be a shortest u, v-path in G_1 having length $k + 1$, and let w be the vertex on P adjacent to v. We have that $d_{G_1}(u, w) = k$; otherwise there would be a shorter u, v-path. By the inductive hypothesis, $d_{G_2}(\Phi(u), \Phi(w)) = k$. Also, since Φ is an isomorphism, $\Phi(w)\Phi(v) \in E(G_2)$. Therefore, there is a $\Phi(u), \Phi(v)$-path of length at most $k + 1$, and thus $d_{G_2}(\Phi(u), \Phi(v)) \leq k + 1$.

Next, to show that $d_{G_2}(\Phi(u), \Phi(v)) \geq k + 1$, we will show that there exists no $\Phi(u), \Phi(v)$-path in G_2 of length less than $k + 1$. Assume, by way of contradiction, that there exists some $\Phi(u), \Phi(v)$ path P of length at most k. Let w be the vertex on P adjacent to $\Phi(v)$. Thus we have $d_{G_2}(\Phi(u), w) \leq k - 1$, and therefore:

$$\begin{aligned} d_{G_1}(u,v) &\leq d_{G_1}(u,\Phi^{-1}(w)) + d_{G_1}(\Phi^{-1}(w),v) \qquad \text{by the triangle inequality} \\ &= d_{G_2}(\Phi(u),w) + d_{G_2}(w,\Phi(v)) \qquad \text{by the inductive hypothesis} \\ &\leq (k-1)+1 \\ &= k, \end{aligned}$$

a contradiction to $d_{G_1}(u,v) = k+1$. Therefore $d_{G_2}(\Phi(u),\Phi(v)) = k+1$, and the induction proof is complete. □

Note that the previous result also holds when $G_1 = G_2$ and Φ is an automorphism of this graph. This specific case of Theorem 2.27 leads to the following corollary.

Corollary 2.28. *Let $\Phi : V(G) \to V(G)$ be an automorphism of a graph G. Then, for any $v \in V(G)$, $\Phi(N_G(v)) = N_G(\Phi(v))$.*

Proof. Recall that:

$$N_G(v) = \{u \in V(G) \mid d_G(u,v) = 1\},$$

and therefore the result is immediate from Theorem 2.27. □

Theorem 2.29. *For any graph G, $Aut(G) \subseteq Aut(G^k)$ for all $k \in \mathbb{Z}^+$.*

Proof. Let $\Phi : V(G) \to V(G)$ be an automorphism of a graph G. Consider the graph G^k for arbitrary $k \in \mathbb{Z}^+$, and take arbitrary $u, v \in V(G^k)$. The following then holds:

$$\begin{aligned} uv \in E(G^k) &\Leftrightarrow d_G(u,v) \leq k \\ &\Leftrightarrow d_G(\Phi(u),\Phi(v)) \leq k \qquad \text{by Theorem 2.27} \\ &\Leftrightarrow \Phi(u)\Phi(v) \in E(G^k), \end{aligned}$$

and thus Φ is an automorphism of G^k. □

2.3 Vertex Coloring

The field of graph coloring is one that is very interesting and heavily studied in discrete mathematics, and it contains many unsolved problems. In addition, many applications, such as timetabling, scheduling, register allocation, and frequency assignment problems, can be formulated as graph coloring

problems. Vertex coloring problems are the most common type of graph coloring problems. For definitions of other graph coloring problems, such as edge and total colorings, we refer the reader to [22]. For the remainder of this thesis, it will be assumed that graph coloring refers specifically to vertex coloring.

A coloring of a graph can be thought of as assigning a color to each vertex of the graph so that no two adjacent vertices are given the same color. Formally, this can be represented as in the following definition.

Definition 2.30. A *k-coloring* of a graph G, for any $k \in \mathbb{Z}^+$, is a function $f : V(G) \to [k]$ such that:

$$uv \in E(G) \Rightarrow f(u) \neq f(v).$$

G is *k-colorable* if and only if a k-coloring of G exists.

Definition 2.31. Let G be a graph with k-coloring f. The *ith color class* of G (with respect to f) is the set of vertices $\{v \in V(G) \mid f(v) = i\}$ for $i \in [k]$.

It is clear that any color class of a coloring of G is an independent set of vertices.

Proposition 2.32. *A graph G is 2-colorable if and only if it is bipartite.*

Proof. Assume G is 2-colorable, and let V_1 and V_2 be the color classes of a 2-coloring of G. Then $V(G) = V_1 \cup V_2$, the union of two independent sets of vertices. Thus G is bipartite.

Next, assume G is bipartite with $V(G) = V_1 \cup V_2$, with V_1 and V_2 independent sets of vertices. Next, define $f : V(G) \to [2]$ as follows:

$$f(v) = \begin{cases} 1 & \text{if } v \in V_1, \\ 2 & \text{if } v \in V_2. \end{cases}$$

Clearly the function f is a 2-coloring of G. □

Let G be an n-vertex graph. Since a set consisting of a single vertex is trivially independent, it is clear that G has an n-coloring, the coloring where every vertex of G forms a distinct color class. However, in general, we can find k-colorings of G with $k < n$.

Definition 2.33. The *chromatic number* of a graph G, denoted $\chi(G)$, is the least integer k such that G is k-colorable.

Theorem 2.34. *Let G be an n-vertex graph. Then $\chi(G) \geq \left\lceil \frac{n}{\alpha(G)} \right\rceil$.*

Proof. Let $f : V(G) \to [k]$ be an optimal coloring of G, that is, $k = \chi(G)$. We have:

$$\begin{aligned} n &= \sum_{i=1}^{k} |\{v \in V(G) | f(v) = i\}| \\ &\leq \sum_{i=1}^{k} \alpha(G) \\ &= k\alpha(G). \end{aligned}$$

Rearranging, and applying the fact that $k \in \mathbb{Z}^+$:

$$k \geq \left\lceil \frac{n}{\alpha(G)} \right\rceil. \qquad \square$$

The chromatic number of a graph can be bounded below by its clique number, as we will see in Theorem 2.35.

Theorem 2.35. *For any graph G, $\chi(G) \geq \omega(G)$.*

Proof. Let $V \subseteq V(G)$ be an optimal clique of $V(G)$, that is $|V| = \omega(G)$. Then, since every pair of vertices in V are adjacent, every vertex $v \in V(G)$ must be in a distinct color class of any coloring of G. Thus $\chi(G) \geq \omega(G)$. $\square$

Corollary 2.36. *For any $n \in \mathbb{Z}^+$, $\chi(K_n) = n$.*

Proof. Since $\omega(K_n) = n$, by the result of Theorem 2.35, $\chi(K_n) \geq n$. However, as $n(K_n) = n$, K_n is clearly n-colorable. Hence we have $\chi(K_n) = n$. $\square$

2.4 Algorithms to Solve Vertex Coloring Problems

Suppose we wish to determine the chromatic number of a graph, the smallest integer k such that G is k-colorable. This, and a variety of other problems in the field of graph theory, are often solved algorithmically. Much effort is put into the study and development of these algorithms. The desired result is, of course, to determine the solution in as computationally efficient manner as possible.

2.4.1 An Introduction to Algorithms and Efficiency

Definition 2.37. The *running time* of an algorithm, expressed as a function of the size of the input, is the maximum possible number of basic computational steps used in the execution of the algorithm.

When dealing with graph theoretic problems, the size of the input refers to the number of vertices in the graph. The running time of an algorithm is commonly measured as follows.

Definition 2.38. Suppose the running time of an algorithm with input size n is bounded by the expression $g(n)$. Let $f(n)$ be an expression such that, for any $N \in \mathbb{Z}^+$, there exists $c \in \mathbb{Z}^+$ such that $g(n) \leq cf(n)$ for all $n > N$. Then the running time of this algorithm is $O(f(n))$.

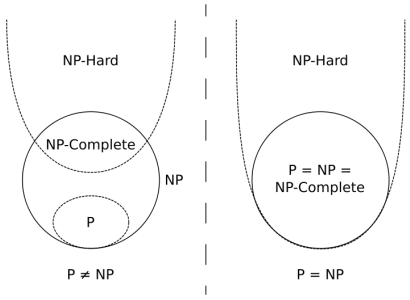
It is widely considered that an algorithm is computationally efficient if it has a polynomial running time. There are methods to determine if a graph is 2-colorable that are computationally efficient. It is known that a graph is 2-colorable if and only if it contains no odd cycles (for a proof of this statement we refer the reader to [22, Theorem 1.2.18]). Because of this nice characterization of 2-colorability, determining if any n-vertex graph G is 2-colorable is solvable in polynomial time. One algorithm that accomplishes this is the DSatur Algorithm [6, p.252], whose running time is $O(n^2)$. However, for $k \geq 3$, whether or not the problem of determining if a graph is k-colorable can be solved efficiently is unknown.

Definition 2.39. A *P (polynomial) problem* is a problem that is solvable in polynomial time, that is, there exists an efficient algorithm to solve it. An *NP (non-deterministic polynomial) problem* is a problem such that a potential solution can be verified in polynomial time. An *NP-hard problem* is a problem that, if solvable by a polynomial time algorithm, then this algorithm could be used to construct a polynomial time algorithm for all NP problems. An *NP-complete problem* is a problem that is both NP and NP-hard.

The class of NP-hard problems can informally be thought of as the class of problems at least as hard as the hardest problems in NP, and possibly harder. The class of NP-hard problems also include not only decision problems, but those problems that are posed as optimization problems. Clearly, all P problems are also NP problems, as a P problem can verify a solution in polynomial time. It is currently unknown if that converse is true, that is, if any of the many NP-complete problems can be solved in polynomial time.

However, if there exists a polynomial time algorithm for one then there exists a polynomial time algorithm for all NP problems, and hence the class of NP problems is equal to the class of all P problems. This P=NP question has become one of the most famous problems in applied mathematics and computer science. The Clay Mathematics Institute, a nonprofit education foundation in Cambridge, Mass., is currently offering a million dollar prize for a solution to this problem.[1] A Venn diagram for the P, NP, NP-complete, and NP-hard set of problems is included in Figure 2.1[2].

Figure 2.1: Venn Diagram of the Complexity Classes



Determining if an n-vertex graph G is k-colorable, for $k \geq 3$, is an NP-complete problem[9, p.191]. A solution to the problem can be verified in polynomial time. Given a function $f : V(G) \to [k]$, to determine if this satisfies the definition of a vertex coloring, one merely needs to check every edge of the graph and make sure its endpoints are not in the same color class. Since a simple, finite graph has at most $\binom{n}{2} < n^2$ edges, this can be done efficiently. However, the decision problem of finding a k-coloring in a graph is NP-complete. Hence it is not known, for a fixed $k \geq 3$, if there exists a polynomial time algorithm to determine whether or not a graph has a k-coloring. Currently, the only general algorithm to prove that a graph is not k-colorable, for $k \geq 3$, is equivalent to an exhaustive search. In fact, one may have to test every function from $V(G)$ to $[k]$, of which there are k^n, to prove the existence or non-existence of a k-coloring.

[1] http://www.claymath.org/millennium/

[2] Source: http://en.wikipedia.org/wiki/Np-hard/

2.4.2 Integer Programming

Integer programming is also an NP-complete problem. Therefore, as discussed in the previous section, it is computationally equivalent to graph coloring with respect to efficiency. Modeling a graph coloring problem as an integer program (IP), however, can provide some practical computational improvements over an exhaustive search for a given graph. We explore this model here.

Let G be a n-vertex graph (for simplicity, let $V(G) = [n]$). The first, and most simple, algorithm to determine if G is k-colorable is a basic integer programming model. The variables of the IP are $x_{i,j}$ where $i \in [n]$ and $j \in [k]$. We let $x_{i,j}$ be a binary variable that equals 1 if vertex i is assigned color j and 0 otherwise. The problem is then equivalent to determining if the following system (which we will refer to as VC) has a feasible solution.

$$\begin{aligned} x_{i_1,j} + x_{i_2,j} &\leq 1 \qquad \text{for all } i_1 i_2 \in E(G) \text{ and } j \in [k], \\ \sum_{j=1}^{k} x_{i,j} &\geq 1 \qquad \text{for all } i \in [n], \\ x_{i,j} &\in \{0,1\} \qquad \text{for all } i \in [n] \text{ and } j \in [k]. \end{aligned}$$

The first constraint forces that no two adjacent vertices can be assigned the same color. The second constraint forces each vertex to be assigned to at least one color (if it is assigned two colors then it can be assigned either color arbitrarily). Trivially, $\chi(G)$ is the smallest value of k such that this integer program has a feasible solution. The model is simple to design; however, it can be very computationally inefficient in execution. Graph coloring problems are a very specific kind of integer program, and this basic model has inherent symmetry associated with it. For example, suppose V_1 and V_2 are color classes in a coloring of G. Assigning V_1 color j_1 and V_2 color j_2, according to the VC model, is a completely different solution than assigning V_1 color j_2 and V_2 color j_1. It becomes apparent that a basic integer program can waste much computational time searching for "new" solutions that are actually the same, or search branches of the search tree where prior knowledge should imply that these branches contain no feasible solution. The following algorithm attempts to rectify this problem.

2.4.3 Column Generation

In this section, we will introduce a model that avoids some of the symmetry inherent in graph coloring problems. This model is introduced in [15], and we will adopt the same notation. Consider a graph G and let S be the set of all maximal independent sets of G. Now define x_s to be a binary variable for all $s \in S$ that equals 1 if that set is used as a color class in an optimal coloring of G, and 0 otherwise. Now, since all of these sets are by definition independent, so long as every vertex is in some set s that is assigned a color, we have a coloring of G. Note that a vertex can appear in more than one set s that is assigned a label, as we are able to break the tie of which label it gets arbitrarily. The problem, denoted IS, that determines $\chi(G)$ is outlined below.

Minimize $\sum_{s \in S} x_s$ subject to:

$$\sum_{s \in S | i \in s} x_s \geq 1 \qquad \text{for all } i \in V(G),$$

$$x_s \in \{0, 1\} \qquad \text{for all } i \in V(G).$$

Minimizing this objective function minimizes the number of colors used. The first constraint forces each vertex to be in at least one color class of the optimal coloring found, and the second constraint forces the x_s to be binary.

Note that this model treats a coloring as a collection of maximal independent sets with no specific color assigned to each set. Therefore, it loses the symmetry problem discussed in the integer programming approach. However, the number of variables (one for every maximal independent set) is exponential in general and expensive to determine. Thus, we will look at a column generation method that reduces the number of variables immensely at the start, and generates new ones as needed. In this method, S is extended to be all independent sets (rather than only maximal independent sets).

First, we must come up with a collection of sets that produces a coloring. Many times there is a coloring previously found that we are trying to improve on, so its color classes could be a good starting point. Or, the simple approach is to let the initial S be all the singleton sets of vertices, which trivially results in an n-coloring of an n-vertex graph.

We then solve a linear relaxation of IS with the restriction that S is our predetermined collection of independent sets. Solving this relaxation generated a dual value, π_i, for each constraint (each of which represents a vertex of G). We then use these dual values as input for the following integer program, denoted MWIS, below.

Maximize $\sum_{i \in V(G)} \pi_i z_i$ subject to:

$$z_{i_1} + z_{i_2} \leq 1 \qquad \text{for all } i_1 i_2 \in E(G),$$
$$z_i \in \{0, 1\} \qquad \text{for all } i \in V(G).$$

The optimal z_i values correspond to a new independent set of vertices (the vertex i is in this set if and only if $z_i = 1$). The first constraint forces that no two adjacent vertices can be included in this set, and the second forces the z_i to be binary. The algorithm solves the IS relaxation with the predetermined set of independent sets S, and uses the dual values of this linear program as input for MWIS. Solving MWIS generates an independent set, which is added to S. IS is then solved with the new set S, and its dual values are again used as input for MWIS.

The algorithm terminates when the optimal objective function value of MWIS is less than or equal to 1. Linear programming duality, discussed in the next paragraph, tells us that no new independent set will improve the coloring of the graph.

To understand why this algorithm works, suppose the Simplex Algorithm on the relaxation of IS was applied with all independent sets included in S. An independent set (a variable) would enter the basis (thus improving our solution) if and only if it has a negative reduced cost. The reduced cost of a variable s is given by $1 - \pi A_s$, where π is the row vector of dual values from the previous Simplex iteration and A_s is the column of the constraint matrix A corresponding to possible entering set s. The entries of the columns of A_s are 1 if that vertex is in the set and 0 otherwise. This, however, is exactly what the z_i from MWIS represent. Thus we have:

$$1 - \sum_{i \in v} \pi_i z_i < 0, \text{or equivalently,}$$
$$\sum_{i \in v} \pi_i z_i > 1.$$

Upon termination of the algorithm, we then look at the final IS relaxation solved. If it has an integral solution, we are done the original problem and $\chi(G)$ is determined. Otherwise, integrality must be enforced.

To enforce integrality, we treat the solved problem as the initial node on a branch and bound tree. We then examine our fractional solution achieved in the algorithm, and consider an independent set s_1 such that x_{s_1} is fractional. There must exist another independent set $s_2 \neq s_1$ such that $s_1 \cap s_2 \neq \emptyset$; otherwise the vertices in s_1 would have their only nonnegative variable being $x_{s_1} < 1$, an infeasible solution to the relaxed linear program. Consider vertices $i_1, i_2 \in V(G)$ such that $i_1 \in s_1 \cap s_2$ and $i_2 \in s_1 \setminus s_2$ or $i_2 \in s_2 \setminus s_1$. We then create two subproblems, denoted $DIFFER(i_1, i_2)$ and $SAME(i_1, i_2)$. $DIFFER(i_1, i_2)$ requires that i_1 and i_2 are given different colors, whereas $SAME(i_1, i_2)$ requires that i_1 and i_2 are given the same color. It is clear that any feasible coloring (and thus, an optimal coloring) of G occurs in one of these two cases. These subproblems can be thought of graph theoretically: $DIFFER(i_1, i_2)$ corresponds to adding the edge $i_1 i_2$ to the graph, and $SAME(i_1, i_2)$ corresponds to collapsing vertices i_1 and i_2 into a single vertex j, with $N_G(j) = N_G(i_1) \cup N_G(i_2)$.

The column generation algorithm is then solved in each subproblem, in the hopes that an integral solution is found. If not, subproblems are again created and solved through column generation, and the branch and bound tree is explored until an optimal value is found. This optimal value is $\chi(G)$.

This algorithm is in most cases superior to the straight integer programming approach discussed in Chapter 2.4.2 (see [15] for a comparison on a test bed of graphs). However, the algorithm can still be very computationally expensive. While it is a simpler integer program than the one outlined in Chapter 2.4.2, the MWIS model is still an integer program which are in the class of NP-Complete problems. Since MWIS may be repeated many times over the course of the algorithm, it is vital to be able to solve it in as efficient a manner as possible. Mehrotra and Trick discuss ways of doing this in [15, pp.345–346].

2.4.4 The Petford-Welsh Algorithm

Petford and Welsh introduced an algorithm in [19] that searches for a 3-coloring of a graph. The generalization of the algorithm to search for a k-coloring is trivial, and we offer this generalization. Consider an n-vertex

graph G. The algorithm first assigns every vertex of G a random number (color) in $[k]$ (note that this is simply a random assignment of k colors and not likely to be a proper k-coloring). At any instant, for all vertices v and every i in $[k]$, we define $S_i(v)$ to be the subset of $N_G(v)$ that are colored i. In addition, $s_i(v)$ is defined to be $|S_i(v)|$. A transition function p is also defined that satisfies, for $s_1, s_2 \ldots, s_k \in [n]$:

$$p(s_1, s_2, \ldots, s_k; j) \geq 0 \qquad \text{for all } j \text{ in } [k],$$
$$\sum_{j=1}^{k} p(s_1, s_2, \ldots, s_k; j) = 1.$$

Note that this satisfies the definition of a probability mass function $P(X = j)$ of a discrete random variable X that can take on values in $[k]$. One example of such a transition function is:

$$p(s_1, s_2, \ldots, s_k; j) = \frac{1}{k-1}\left(1 - s_j\left(\sum_{i=1}^{k} s_i\right)^{-1}\right).$$

Every iteration, a vertex v is chosen at random out of all "bad" vertices, that is, all vertices adjacent to another vertex currently assigned the same color. Next, a (not necessarily new) color is randomly generated according to the probability mass function defined by $p(s_1(v), s_2(v), \ldots, s_k(v); j)$, and this color is assigned to the vertex v. The algorithm continues until there are no more bad vertices to choose from, in which case it has found a k-coloring, or the running time goes beyond a predetermined stopping time. If it does not find a k-coloring in the allotted time, then the algorithm is inconclusive.

Obviously, this algorithm can't be used to prove that a graph is not k-colorable. However, for "good" choices of the transition function p, this algorithm was shown to find 3-colorings on randomly generated 3-colorable graphs in a relatively short length of time (see [19] for more details on these graphs and choices of transition functions). The preliminary results for larger k were not very promising, and one possible reason that was given was that much more experimentation was needed to find a good transition function. However, this algorithm will be revisited in Chapter 3.3.3, in which it is applied successfully to find a 14-coloring of Q_8^2, a graph that will be defined next.

Chapter 3

The Hypercube

We now discuss the main combinatorial object studied in this thesis.

Definition 3.1. For any $k \in \mathbb{Z}^+$, the *k-dimensional hypercube (k-cube)*, denoted Q_k, is the graph satisfying the following:

$$V(Q_k) = \{0,1\}^k,$$
$$E(Q_k) = \{uv \in \{0,1\}^k \mid d(u,v) = 1\}.$$

This is the graph in which the vertices are all binary vectors of length k, and two vertices are adjacent if and only if the Hamming distance between them is 1. Often, we will be dealing with the jth power of the k-cube (when $j \leq k$), Q_k^j. This graph satisfies:

$$V(Q_k^j) = \{0,1\}^k,$$
$$E(Q_k^j) = \{u,v \in \{0,1\}^k \mid d(u,v) \leq j\}.$$

It is clear that $n(Q_k^j) = 2^k$ for any $j,k \in \mathbb{Z}^+$. The result of the following proposition will allow us to enumerate $e(Q_k^j)$.

Proposition 3.2. *Let $j,k \in \mathbb{Z}^+$ with $j \leq k$. Then $e(Q_k^j) = 2^{k-1}t$ where t satisfies:*

$$t = \sum_{i=1}^{j} \binom{k}{i}.$$

Proof. First, we note that Q_k^j is t-regular since, by definition, t sums up all binary vectors of distance at most j from any vertex. Thus Proposition 2.8 implies:

$$\begin{aligned} e(Q_k^j) &= \frac{n(Q_k^j)t}{2} \\ &= \frac{2^k t}{2} \\ &= 2^{k-1}t. \end{aligned}$$

□

Note that if $j \geq k$ then every vertex of Q_k^j is adjacent. Therefore $Q_k^j \cong K_{2^k}$, and hence:

$$e(Q_k^j) = \binom{2^k}{2} = \frac{(2^k)(2^k-1)}{2} = 2^{k-1}(2^k-1).$$

This agrees with the result of Proposition 3.2 as:

$$\begin{aligned} 2^{k-1}\sum_{i=1}^{j}\binom{k}{i} &= 2^{k-1}\sum_{i=1}^{k}\binom{k}{i} \\ &= 2^{k-1}(2^k-1), \end{aligned}$$

by the binomial theorem.

We will most often be discussing the case $j = 2$, the graph Q_k^2. In this case:

$$t = \binom{k}{1} + \binom{k}{2} = \binom{k+1}{2},$$

and hence

$$e(Q_k^2) = 2^{k-1}\binom{k+1}{2}.$$

The number of vertices and edges of Q_k^2 have been calculated for all $k \leq 31$ in Appendix A.

We now note the following results, and our first connections between Chapter 1 and Chapters $2-3$.

Lemma 3.3. *$\alpha(Q_k^j) = A(k, j+1)$ for any $j, k \in \mathbb{Z}^+$ with $j \leq k$.*

Proof. As two vertices of Q_k^j are adjacent if and only if their Hamming distance is at most j, any independent set V of Q_k^j is a set of vertices satisfying $d(u,v) \geq j+1$ for all $u, v \in V$. This is a binary code with minimum distance at least $j+1$, which leads to the desired result. □

This leads to the following proposition.

Proposition 3.4. *Let $\Phi : \{0,1\}^k \to \{0,1\}^k$ be an automorphism of Q_k^j, and C be a $(k, M, j+1)$ binary code. $\Phi(C)$ is then a $(k, M, j+1)$ binary code, where $\Phi(C)$ is defined as follows:*

$$\Phi(C) = \{\Phi(u) \mid u \in C\}.$$

Proof. Clearly, C is an independent set of Q_k^j. Moreover, by Proposition 2.25, $\Phi(C)$ is an independent set of Q_k^j. Therefore $\Phi(C)$ is a $(k, M, j+1)$ binary code. □

3.1 Automorphisms of the Cube and its Square

In this section, we will define the automorphism groups of Q_k and Q_k^2. The result of Theorem 2.29 implies that $Aut(Q_k) \subseteq Aut(Q_k^j)$ for all $j \in \mathbb{Z}^+$. Therefore every automorphism of Q_k is, in addition, an automorphism of Q_k^j. Consider the following function.

Definition 3.5. Let $x \in \{0,1\}^k$. Then the *vector addition function*:

$$\Psi_x : \{0,1\}^k \to \{0,1\}^k,$$

is defined to be:

$$\Psi_x(v) = v + x.$$

This function is an automorphism of Q_k, as will be seen in the Theorem 3.6.

Theorem 3.6. *For any $x \in \{0,1\}^k$, Ψ_x is an automorphism of the graph Q_k .*

Proof. Consider an arbitrary $x \in \{0,1\}^k$. To prove Ψ_x is an automorphism of Q_k, we first prove that Ψ_x is a permutation of $\{0,1\}^k$.

To prove Ψ_x is a permutation, assume (for $u, v \in \{0,1\}^k$) that $\Psi_x(u) = \Psi_x(v)$. We have:

$$\begin{aligned} \Psi_x(u) = \Psi_x(v) &\Leftrightarrow u + x = v + x \\ &\Leftrightarrow u = v, \end{aligned}$$

and hence Ψ_x is one-to-one. Since $\{0,1\}^k$ is finite, it is therefore onto and thus a permutation. Note that throughout this thesis, we will prove a function from $\{0,1\}^k$ to $\{0,1\}^k$ is a permutation by simply proving it is one-to-one.

To prove Ψ_x is an automorphism of Q_k, take $u, v \in \{0,1\}^k$ such that $uv \in E(Q_k)$. We have:

$$\begin{aligned} uv \in E(Q_k) &\Leftrightarrow d(u,v) = 1 \\ &\Leftrightarrow d(u+x, v+x) = 1 \qquad \text{(by Lemma 1.9)} \\ &\Leftrightarrow d(\Psi_x(u), \Psi_x(v)) = 1 \\ &\Leftrightarrow \Psi_x(u)\Psi_x(v) \in E(Q_k), \end{aligned}$$

and therefore Ψ_x preserves edges and is hence an automorphism. □

Corollary 3.7. *Q_k^j is vertex transitive for any $j, k \in \mathbb{Z}^+$.*

Proof. Consider arbitrary $u, v \in \{0,1\}^k$. By Theorem 3.6, the vector addition function Ψ_{u+v} is an automorphism of Q_k. Theorem 2.29 then implies that $\Psi_{u+v} \in Aut(Q_k^j)$ for all $j \in \mathbb{Z}^+$. Moreover:

$$\begin{aligned} \Psi_{u+v}(u) &= u + (u+v) \\ &= v, \end{aligned}$$

and hence, Ψ_{u+v} maps u to v. Since this holds for any choices of $u, v \in \{0,1\}^k$, Q_k^j is vertex transitive. □

Permuting the coordinates of the vertices of Q_k is also an automorphism, as we will prove in Theorem 3.8.

Theorem 3.8. *Permuting the vertex coordinates under the permutation σ, for any $\sigma \in S_k$, is an automorphism of Q_k.*

Proof. To prove σ is an automorphism of Q_k, we first prove that permuting the coordinates under σ also permutes the vertices in $\{0,1\}^k$. Assume, for $u, v \in \{0,1\}^k$, that $\sigma(u) = \sigma(v)$. Since σ is a permutation, it by definition has an inverse σ^{-1}, satisfying $\sigma^{-1}(\sigma(u)) = u$ and $\sigma^{-1}(\sigma(v)) = v$. Now since $\sigma(u)_i = \sigma(v)_i$ for all coordinates $i \in [k]$, $u_i = v_i$ for all coordinates i. Therefore σ is a permutation.

To prove σ is an automorphism of Q_k, consider $u, v \in \{0,1\}^k$ such that $uv \in E(Q_k)$. By definition of Q_k, $d(u,v) = 1$. Thus there is exactly one coordinate $i \in [k]$ where $u_i \neq v_i$. Therefore $\sigma(u)$ and $\sigma(v)$ differ only in coordinate $\sigma(i)$, and hence $d(\sigma(u), \sigma(v)) = 1$. Clearly $\sigma(u)\sigma(v) \in E(Q_k)$.

Now consider $u, v \in \{0,1\}^k$ such that $\sigma(u)\sigma(v) \in E(Q_k)$. By definition of Q_k, $d(\sigma(u), \sigma(v)) = 1$. Thus there is exactly one coordinate $i \in [k]$ where

$\sigma(u)_i \neq \sigma(v)_i$. Therefore u and v differ only in coordinate $\sigma^{-1}(i)$, and hence $d(u,v) = 1$. Clearly $uv \in E(Q_k)$. □

Lemma 3.9. *Let $x, y \in \{0,1\}^k$ and $\sigma_1, \sigma_2 \in S_k$ such that $x \neq y$ or $\sigma_1 \neq \sigma_2$. Then:*

$$\sigma_1 \circ \Psi_x \neq \sigma_2 \circ \Psi_y.$$

Proof. Assume that $\sigma_1 \circ \Psi_x = \sigma_2 \circ \Psi_y$ and, by way of contradiction, $x \neq y$ or $\sigma_1 \neq \sigma_2$. Thus the following holds:

$$\begin{aligned}(\sigma_1 \circ \Psi_x)(x) = (\sigma_2 \circ \Psi_y)(x) &\Leftrightarrow \sigma_1(\vec{0}) = \sigma_2(x+y)\\ &\Leftrightarrow \vec{0} = \sigma_2(x+y)\\ &\Leftrightarrow \vec{0} = x+y\\ &\Leftrightarrow x = y,\end{aligned}$$

and therefore $\sigma_1 \neq \sigma_2$, so it must be the case that $x = y$. Then there is some $v \in \{0,1\}^k$ such that:

$$\begin{aligned}\sigma_1(v) \neq \sigma_2(v) &\Leftrightarrow \sigma_1(\Psi_x(v+x)) \neq \sigma_2(\Psi_y(v+y))\\ &\Leftrightarrow \sigma_1(\Psi_x(v+x)) \neq \sigma_2(\Psi_y(v+x)) \qquad \text{since } x = y\\ &\Rightarrow \sigma_1 \circ \Psi_x \neq \sigma_2 \circ \Psi_y,\end{aligned}$$

a contradiction. □

Since $|S_k| = k!$ and $|\{0,1\}^k| = 2^k$, the result of Lemma 3.9 implies that we can create $2^k k!$ distinct automorphisms of Q_k through composing vector addition and coordinate permutation automorphisms. As we will see in Theorem 3.11, every automorphism of Q_k is of this form. The proofs of the following lemma and theorem are of a combinatorial nature similar to that used in the solution manual to [22, Exercise 1.3.29]. An alternative proof of the result that has an algebraic flavor can be found in [16].

Lemma 3.10. *Let $j, k \in \mathbb{Z}^+$ with $j \leq k$. Every copy of Q_j in Q_k is $Q_k[V]$ where $V \subseteq \{0,1\}^k$ satisfies:*

$V = \{v \in \{0,1\}^k \mid v$ *has specified values in a fixed set of $k-j$ coordinates*$\}$.

Proof. Let H be any subgraph of Q_k such that H is a copy of Q_j in Q_k. There then exists some isomorphism Φ from Q_j (defined normally) to H. Since H is a subgraph of Q_k, for every $x_1, x_2 \in V(H)$, $d_H(x_1, x_2) \geq d_{Q_k}(x_1, x_2)$. Let $u = \Phi(\vec{0}^j)$ and $v = \Phi(\vec{1}^j)$. We will prove by induction on j that $d(u,v) = j$.

- **Base Case(s):**
  If $j = 1$, then $H \cong Q_1$, which contains simply two vertices connected by an edge. Thus $d_{Q_k}(u, v) = 1$ and hence $d(u, v) = 1$. If $j = 2$, then u and v have the common neighbour $\Phi(01)$ in H since $\vec{0}^2$ and $\vec{1}^2$ have the common neighbour 01 in Q_2. Now if $d(u, v) = 1$ then $uv \in E(Q_k)$, and since $H \subseteq Q_k$, $uv\Phi(01)u$ would form a cycle of length 3 in the bipartite graph Q_k, a contradiction. Thus $d(u, v) = 2$.

- **Inductive Hypothesis:**
  Suppose for some positive integer $j > 2$, if ξ is an isomorphism from Q_l, for some $l \in [j-1]$, to a subgraph of Q_k, then $d(\xi(\vec{0}^l), \xi(\vec{1}^l)) = l$.

We now prove the result for j. Consider the isomorphism $\Phi : Q_j \to H$. For all $i \in [j]$, let $w_i = \Phi(\mu_i^j)$. Now since $\vec{0}^j\mu_i^j \in E(Q_j)$, $uw_i \in E(H)$, and hence $d(u, w_i) = 1$.

Now let $W_i = \{x \in V(Q_j) \mid x_i = 1\}$. Then $Q_{j-1} \cong Q_j[W_i]$ under the isomorphism ξ_i that lengthens the vector by inserting a 1 in the ith coordinate. Thus $\Phi_i \circ \xi_i$ is an isomorphism from Q_{j-1} to a subgraph of Q_k where Φ_i is the restriction of Φ to W_i. By the induction hypothesis, for $i \in [j]$:

$$d(\Phi_i \circ \xi_i(\vec{0}^{j-1}), \Phi_i \circ \xi_i(\vec{1}^{j-1})) = j - 1.$$

Moreover, $\xi_i(\vec{0}^{j-1}) = \mu_i^j$ and $\xi_i(\vec{1}^{j-1}) = \vec{1}^j$. Therefore we have $d(w_i, v) = j - 1$ for $i \in [j]$. Thus for each $i \in [j]$, w_i is different from v in $j - 1$ coordinates and is different from u in one coordinate.

If for some $i \in [j]$, the $j - 1$ coordinates in which w_i is different from v do not include the one coordinate in which w_i is different from u, then u and v differ in all these j coordinates and hence $d(u, v) = j$. So, suppose to the contrary that there are distinct $i_1, i_2 \in [j]$ such that w_{i_1} and w_{i_2} differ from u only in coordinates j_1 and j_2, respectively, and additionally that $u_{j_1} = v_{j_1}$ and $u_{j_2} = v_{j_2}$.

Since w_{i_1} differs from v in $j-1$ coordinates and differs from u in exactly one of these coordinates, w_{i_1} differs from v in $j - 2$ coordinates but is the same as u in these same $j-2$ coordinates. Thus u and v differ in these $j-2$

coordinates. Hence $d(u, v) \geq j - 2$.

Consider now $\alpha = \mu^j_{i_1} + \mu^j_{i_2}$ and its image $z = \Phi(\mu^j_{i_1} + \mu^j_{i_2})$. Since $\mu^j_{i_1}\alpha, \mu^j_{i_2}\alpha \in E(Q_j)$, $w_{i_1}z, w_{i_2}z \in E(H)$. Thus:

$$d(w_{i_1}, z) = 1 = d(w_{i_2}, z).$$

If $d(u, z) = 1$, then $uw_{i_1}zu$ is a cycle of length 3 in the H. Since $H \subseteq Q_k$ this is also a cycle of length 3 in the bipartite graph Q_k, a contradiction. Thus $d(u, z) = 2$.

Suppose z differs from u in coordinate j_3 where $j_3 \notin \{j_1, j_2\}$. Then u and w_{i_1}, and u and w_{i_2}, all agree in coordinate j_3. However:

$$d(w_{i_1}, z) = 1 = d(w_{i_2}, z),$$

and so $w_{i_1} = w_{i_2}$, a contradiction. Therefore $j_3 \in \{j_1, j_2\}$.

Since $d(u, z) = 2$, z therefore differs from u only in coordinates j_1 and j_2. Since $u_{j_1} = v_{j_1}$ and $u_{j_2} = v_{j_2}$, z differs from v in coordinates j_1 and j_2. Moreover, z differs from v in all the coordinates in which v differs from u, since in those coordinates z agrees with u. Thus:

$$\begin{aligned} d(v, z) &\geq d(u, z) + d(u, v) \\ &\geq 2 + (j - 2) \\ &= j. \end{aligned}$$

With $\alpha \in V(Q_j)$, we have:

$$\begin{aligned} j - 2 &= d_{Q_j}(\vec{1}^j, \alpha) \\ &= d_H(\Phi(\vec{1}^j), \Phi(\alpha)) \qquad \text{by Theorem 2.27} \\ &= d_H(v, z) \\ &\geq d_{Q_k}(v, z) \qquad \text{since } H \subseteq Q_k \\ &= d(v, z) \\ &\geq j, \end{aligned}$$

a contradiction. Thus no such i_1 and i_2 exist, and $d(u, v) = j$. Hence by induction, the statement holds for all j.

Next, let S be the set of coordinates in which u and v are the same. By the preceding inductive proof, $|S| = k - j$. Consider $w \in V(H)$. Since $\Phi^{-1}(w)$ is in Q_j and for all $y_1, y_2 \in V(Q_j)$, $d(y_1, y_2) = d_{Q_j}(y_1, y_2)$, we have:

$$\begin{aligned} d_{Q_j}(\vec{0}^j, \Phi^{-1}(w)) + d_{Q_j}(\vec{1}^j, \Phi^{-1}(w)) &= d(\vec{0}^j, \Phi^{-1}(w)) + d(\vec{1}^j, \Phi^{-1}(w)) \\ &= j. \end{aligned}$$

By Theorem 2.27, it then follows that $d_H(u, w) + d_H(v, w) = j$. Hence $d(u, w) + d(v, w) \leq j$. Moreover, the coordinates in $[k] \backslash S$ (as they are the coordinates in which u and v differ) contribute j to the sum $d(u, w) + d(v, w)$. Thus u and w must be the same in all coordinates in S. Hence every vertex in H has the same values in S and therefore H is of the form $Q_k[V]$ for V defined as in the theorem statement. □

Theorem 3.11. *$|Aut(Q_k)| = 2^k k!$ for $k \in \mathbb{Z}^+$.*

Proof. As shown earlier, the result of Lemma 3.9 implies that $|Aut(Q_k)| \geq 2^k k!$. Thus, we need only prove that $|Aut(Q_k)| \leq 2^k k!$, that is, that every automorphism of Q_k can be expressed as a composition of a vector addition and coordinate permutation automorphism.

Let Φ be an automorphism of Q_k, and consider $\vec{0}^k$. Define $v = \Phi(\vec{0}^k)$. Corollary 2.28 implies that $\Phi(N_{Q_k}(\vec{0}_k)) = N_{Q_k}(v)$. Consider $w \in \{0, 1\}^k$ such that $wt(w) \geq 2$, and let $j = wt(w)$. Then $\vec{0}^k$ and w are vertices in a copy of Q_j in Q_k, call this copy G. Define $H = Q_k[\Phi(G)]$. Since Φ is an automorphism, $H \cong Q_j$. By Lemma 3.10, H is a subgraph induced by a set of 2^j vertices having specified values on a fixed set of $k - j$ coordinates, call this set S. Let x be the vector attaining these specified values in S and with a 0 in the j coordinates not in S. Moreover, consider $x + \mu_i^k$ for all i not in S. These vertices are in H since their coordinates in S are as specified, and their images under Φ provide j vertices that differ from v in exactly 1 place. Thus once these images are chosen, the $k - j$ coordinates that are the same in all vertices of H have been determined. Since $\Phi(w)$ is different from v in exactly j places, and the $k - j$ coordinates in which v and $\Phi(w)$ are the same have already been determined by the choice of the images of $N_{Q_k}(v)$, $\Phi(w)$ is completely determined.

Thus every automorphism of Q_k has at most 2^k choices to map $\vec{0}^k$, and at most $k!$ choices to map $N_{Q_k}(\vec{0}^k)$, after which the rest of the automorphism is determined. Therefore, $|Aut(Q_k)| \leq 2^k k!$, and we conclude with the desired result. □

Definition 3.12. For every i in $[k]$, the function:

$$\tau_i : \{0,1\}^k \to \{0,1\}^k,$$

satisfies the following, for all v in $\{0,1\}^k$ with coordinates defined by $v = v_1 v_2, \ldots, v_n$:

$$\tau_i(v) = \begin{cases} v, & \text{if } wt(v) \text{ is even and } v_i = 0, \\ v, & \text{if } wt(v) \text{ is odd and } v_i = 1, \\ v + \mu_i^k, & \text{otherwise.} \end{cases}$$

Note that if $v_i = 0$, then $\tau_i(v)$ is of even parity. Moreover, if $v_i = 1$, then $\tau_i(v)$ is of odd parity.

It is clear that τ_i is not an automorphism of Q_k, as it can't be expressed as a composition of vector addition and coordinate permutation. However, τ_i is in fact an automorphism of Q_k^2, as we will see in Theorem 3.13, first proved by Miller in [16, Lemma 3.1].

Theorem 3.13. *The function τ_i is an automorphism of Q_k^2 for all $i \in [k]$.*

Proof. Fix $i \in [k]$. To prove τ_i is an automorphism of Q_k^2, we first prove that τ_i is a permutation of $\{0,1\}^k$.

To prove τ_i is a permutation, assume by way of contradiction that there exist $u, v \in \{0,1\}^k$ such that $\tau_i(u) = \tau_i(v)$ and $u \neq v$. We must have $u_j = v_j$ for all $j \in [k] \setminus i$ and $u_i \neq v_i$. Assume, without loss of generality, that $u_i = 0$ and $v_i = 1$. Thus $d(u,v) = 1$, and therefore u and v are of differing parity. If $wt(u)$ is even, then $\tau_i(u) = u$ and $\tau_i(v) = v$ by defintion and hence $u = v$, a contradiction. Similarly, if $wt(u)$ is odd, then $\tau_i(u) = u + \mu_i^k$ and $\tau_i(v) = v + \mu_i^k$, and hence $u + \mu_i^k = v + \mu_i^k$. Then by Lemma 1.9, $u = v$, also a contradiction. Thus τ_i is a permutation.

We will now prove that τ_i is an automorphism of Q_k^2. Consider $u, v \in \{0,1\}^k$ such that $uv \in E(Q_k^2)$. We consider two cases:

- **Case 1:** $d(u,v) = 1$**.**
  This case is trivial since τ_i only alters one coordinate of both u and v. Thus $d(\tau_i(u), \tau_i(v)) \leq 2$ and hence $\tau_i(u)\tau_i(v) \in E(Q_k^2)$.

- **Case 2:** $d(u,v) = 2$**.**
  By Corollary 1.16, u and v are either both of even weight or both of

odd weight. Thus, if u and v differ in coordinate i we have:

$$\begin{aligned} d(\tau_i(u), \tau_i(v)) &= wt(\tau_i(u) + \tau_i(v)) \\ &= wt(u + v + \mu_i^k) \\ &\qquad \text{as } \mu_i^k \text{ will be added to exactly one of } u, v \\ &= wt(u + v) - 1 \qquad \text{as } (u+v)_i = 1 \\ &= d(u, v) - 1 \\ &\leq 2, \end{aligned}$$

and therefore $\tau_i(u)\tau_i(v) \in E(Q_k^2)$.

Now, if u and v are the same in coordinate i, since they are of the same parity, τ_i performs the same operation (either adding μ_i^k or not adding μ_i^k) on both u and v. Therefore $d(\tau_i(u), \tau_i(v)) = d(u, v) = 2$, and hence $\tau_i(u)\tau_i(v) \in E(Q_k^2)$.

Now consider $u, v \in \{0,1\}^k$ such that $\tau_i(u)\tau_i(v) \in E(Q_k^2)$. We again consider two cases:

- **Case 1:** $d(\tau_i(u), \tau_i(v)) = 1$**.**
  This case is trivial since τ_i only alters one coordinate of both u and v. Thus $d(u, v) \leq 2$ and hence $uv \in E(Q_k^2)$.

- **Case 2:** $d(\tau_i(u), \tau_i(v)) = 2$**.**
  Again by Corollary 1.16, $\tau_i(u)$ and $\tau_i(v)$ are either both of even weight or both of odd weight. Thus, if $\tau_i(u)$ and $\tau_i(v)$ differ in coordinate i we have:

$$\begin{aligned} d(u, v) &= wt(u + v) \\ &= wt(\tau_i(u) + \tau_i(v) + \mu_i^k) \\ &\qquad \text{as } \mu_i^k \text{ will be added to exactly one of } u, v \\ &= wt(\tau_i(u) + \tau_i(v)) - 1 \qquad \text{as } (\tau_i(u) + \tau_i(v))_i = 1 \\ &= d(u, v) - 1 \\ &\leq 2, \end{aligned}$$

  and therefore $uv \in E(Q_k^2)$.

Now, if $\tau_i(u)$ and $\tau_i(v)$ are the same in coordinate i, since they also are of the same parity, τ_i performs the same operation (either adding μ_i^k or not adding μ_i^k) on both u and v. Thus $d(u,v) = d(\tau_i(u), \tau_i(v)) = 2$, and hence $uv \in E(Q_k^2)$. □

For any $i \in [k]$, since $\tau_i \in Aut(Q_k^2) \setminus Aut(Q_k)$, we have $Aut(Q_k) \subset Aut(Q_k^2)$, with equality not holding. In fact, Miller proved in [16] that the automorphism group of the square of the k-cube can be generated by vector addition, coordinate permutation, and a single τ_i automorphism. Using this, he was then able to prove the following result.

Theorem 3.14. *For any positive integer k at least 4:*

$$|Aut(Q_k^2)| = (|\{0,1\}^k|)(|S_{k+1}|) = 2^k(k+1)!.$$

3.2 The Chromatic Number of the Cube and its Square

Determining the chromatic number of a graph, as discussed in Chapter 2.4, can be very computationally expensive. This, however, does not hold for the k-cube, Q_k.

Theorem 3.15. $\chi(Q_k) = 2$ *for all* $k \in \mathbb{Z}^+$.

Proof. Define $V_e, V_o \subseteq \{0,1\}^k$ as follows:

$$V_e = \{v \in \{0,1\}^k | wt(v) \text{ is even}\},$$
$$V_o = \{v \in \{0,1\}^k | wt(v) \text{ is odd}\}.$$

Trivially, $V_e \cup V_o = \{0,1\}^k$. No two vertices in either set can be adjacent, since $d(u,v) = 1$ implies that u and v are of differing parity. Thus Q_k is bipartite, and hence 2-colorable. Moreover, we can trivially have no 1-coloring of Q_k as there is at least one pair of adjacent vertices (for example, consider $\vec{0}^k$ and μ_1^k). Therefore $\chi(Q_k) = 2$. □

The chromatic number of the k-cube is thus 2 for any $k \in \mathbb{Z}^+$. However, higher powers of the k-cube are not bipartite (for $k \geq 2$). Thus, determining $\chi(Q_k^j)$ for $j, k \geq 2$ and $j \leq k-2$ can't necessarily be done in polynomial time.

We consider specifically the square of the k-cube, whose chromatic number is unknown for many k. Included in Appendix A is a table of known

values of and bounds on $\chi(Q_k^2)$ for $k \leq 31$, as well as other relevant information. Due to the result of Lemma 3.3, any coloring of Q_k^2 partitions the vertices into binary codes of length k and minimum distance at least 3. Theorem 2.34 then allows us to determine the lower bound on $\chi(Q_k^2)$ seen in Proposition 3.16.

Proposition 3.16. *$\chi(Q_k^2) \geq \left\lceil \dfrac{2^k}{A(k,3)} \right\rceil$ for any $k \in \mathbb{Z}^+$.*

Proof. The proof of this proposition is immediate from the results of Theorem 2.34 and Lemma 3.3. □

As discussed in Chapter 1, $A(k,3)$ is not known for many $k \geq 16$. If not known, we can still apply the sphere-packing bound to Proposition 3.16, as will be seen in Theorem 3.19 and Corollary 3.20. However, consider the case where $k = 2^n - i$ for some $i, n \in \mathbb{Z}^+$ with $n \geq 3$ and $i \in [4]$. $A(k,3)$ is known explicitly from the result of Theorem 1.61. This allows for the following corollary to Proposition 3.16.

Corollary 3.17. *Let $k = 2^n - i$ for some $i, n \in \mathbb{Z}^+$ with $n \geq 3$ and $i \in [4]$. Then:*

$$\chi(Q_k^2) \geq 2^n.$$

Proof. By Theorem 1.61 and Proposition 3.16 we have:

$$\begin{aligned} \chi(Q_k^2) &\geq \left\lceil \frac{2^{2^n-i}}{2^{2^n-n-i}} \right\rceil \\ &= 2^n. \end{aligned}$$

□

Another way to determine a lower bound on $\chi(Q_k^2)$ is by exhibiting an optimal clique in Q_k^2, and then applying the result of Theorem 2.35.

Lemma 3.18. *$\omega(Q_k^2) = k + 1$ for any $k \in \mathbb{Z}^+$ with $k \geq 3$.*

Proof. Let $k \geq 3$. We will first exhibit a clique of size $k+1$. Consider the set of vertices $V \subseteq \{0,1\}^k$ defined as follows:

$$V = \{\vec{0}^k, \mu_1^k, \mu_2^k, \ldots, \mu_k^k\}.$$

It is clear that any two vertices in V have distance at most 2, and thus V is a clique of size $k+1$ in Q_k^2.

Now assume, by way of contradiction, that $\omega(Q_k^2) \geq k+2$. Let K be a clique of Q_k^2 with $|K| = k+2$. Since Q_k^2 is vertex transitive, we can assume $\vec{0}^k \in K$. Thus every vertex in k has weight at most 2. Since there are at most k vertices in K of weight 1, namely ones from $\{\mu_1^k, \mu_2^k, \ldots, \mu_k^k\}$, some vertex $v \in K$ has weight 2. Let $v = \mu_{i_1}^k + \mu_{i_2}^k$ for $i_1, i_2 \in [k]$ with $i_1 < i_2$.

Now let $\sigma_1 \in S_k$ be a permutation that satisfies $\sigma_1(i_1) = 1$ and $\sigma_1(i_2) = 2$. Clearly $\sigma_1(v) = \mu_1^k + \mu_2^k$. Define $K' = \sigma_1(K)$ and $v' = \sigma_1(v)$. It is clear that $\vec{0}^k, v' \in K'$, $|K'| = k+2$, and, by Proposition 2.25, K' is also a clique of Q_k^2. Thus $\mu_i^k \notin K'$ for $i \in [k]$ with $i \geq 3$, since then:

$$d(v', \mu_i^k) = wt(v') + wt(\mu_i^k) - 2wt(v' \cap \mu_i^k) = 2 + 1 + 0 = 3.$$

Therefore, the only vertices of weight 1 possibly contained in K' are μ_1^k and μ_2^k. Since $k + 2 - 4 \geq 1$, K' has another vertex of weight 2, $w' \neq v'$. Let $w' = \mu_{j_1}^k + \mu_{j_2}^k$ for some $j_1, j_2 \in [k]$ with $j_1 < j_2$. Since $d(v', w') \leq 2$ and $w' \neq v'$, $|\{j_1, j_2\} \cap \{1, 2\}| = 1$. Therefore $j_1 \in \{1, 2\}$ and $j_2 \geq 3$. Define $j_3 \in \{1, 2\}$ where $j_3 \neq j_1$.

Now let $\sigma_2 \in S_k$ be a permutation that satisfies $\sigma_2(j_1) = 1$, $\sigma_2(j_3) = 2$, and $\sigma_2(j_2) = 3$. Clearly $\sigma_2(w') = \mu_1^k + \mu_3^k$ and $\sigma_2(v') = v'$. We define $w'' = \sigma_2(w')$, $v'' = \sigma_2(v') = v'$, and $K'' = \sigma_2(K)$. It is clear that $\vec{0}^k, v' \in K'$, $|K'| = k+2$ and, again by Proposition 2.25, K'' is a clique of Q_k^2. Hence, the only vertex of weight 1 possibly contained in K'' is μ_1^k as $d(w'', \mu_2^k) = 3$ and $d(v'', \mu_i^k) = 3$ for $i \geq 3$.

If $\mu_1^k \in K''$, then all vertices of weight 2 in K'' have the form $\mu_1^k + \mu_i^k$ with $2 \leq i \leq k$; otherwise we would have two of distance 3 in K''. But there are only $k-1$ such weight 2 vertices forcing $|K''| \leq k+1$, a contradiction.

Meanwhile, if $\mu_1^k \notin K''$, we consider any two vertices of weight 2, $x, y \in K'' \setminus \{v, w\}$ with $x \neq y$. Let $x = \mu_a^k + \mu_b^k$ and $y = \mu_c^k + \mu_d^k$ for some $a, b, c, d \in [k]$ with $a < b$ and $c < d$. Since $d(v'', x) \leq 2$ and $d(v'', y) \leq 2$, $a, c \in \{1, 2\}$ and $b, d \geq 3$. If $a = 2$, then since $d(x, w'') \leq 2$, it must be the case that $b = 3$. Similarily, if $c = 2$, then $d = 3$. If $a = 2$ then $c = 1$; moreover $d(x, y) \leq 2$ implies that $d = 3$ giving $y = w''$, a contradiction. Thus $a = 1$. A similar argument shows that $c = 1$. Since $a = c = 1$, then every vertex of weight 2 in K'' has the form $\mu_1^k + \mu_i^k$ for $2 \leq i \leq k$. Again, there are only $k-1$ such weight 2 vertices, and hence $|K''| \leq k$, a contradiction. Therefore, every clique in Q_k^2, for $k \geq 3$, has at most $k+1$ vertices. □

Note that for $k = 1, 2$, since $\chi(Q_k^2) \cong K_{2^k}$, we have $\omega(Q_1^2) = 2$ and $\omega(Q_2^2) = 4$.

The result of Theorem 2.25 then implies that $\chi(Q_k^2) \geq k + 1$ for all $k \in \mathbb{Z}^+$. This will be stated again in Theorem 3.19; however, a second proof will be offered that leads to the strengthened bound of Corollary 3.20.

Theorem 3.19. *$\chi(Q_k^2) \geq k + 1$ for any $k \in \mathbb{Z}^+$.*

Proof 1. The result of Lemma 3.18 proved that $\omega(Q_k^2) = k + 1$. Thus, by the result of Theorem 2.35, $\chi(Q_k^2) \geq k + 1$. □

Proof 2. The result of Proposition 3.16 implies that:

$$\begin{aligned}
\chi(Q_k^2) &\geq \left\lceil \frac{2^k}{A(k,3)} \right\rceil \\
&\geq \left\lceil \frac{2^k}{\lfloor 2^k(k+1)^{-1} \rfloor} \right\rceil \qquad \text{by the sphere-packing bound (Theorem 1.50)} \qquad (3.1) \\
&\geq \left\lceil \frac{2^k}{2^k(k+1)^{-1}} \right\rceil \\
&= k + 1.
\end{aligned}$$

□

Corollary 3.20. *Consider any $k \in \mathbb{Z}^+$ such that $k \neq 2^n - 1$ for any $n \in \mathbb{Z}$. Then $\chi(Q_k^2) \geq k + 2$.*

Proof. Consider line (3.1) in the second proof of Theorem 3.19. We have, if $k + 1 \neq 2^n$, that $2^k(k + 1)^{-1}$ is non-integral. We then have the desired result. □

In addition to the aforementioned lower bounds on $\chi(Q_k^2)$, it is also possible to determine upper bounds on $\chi(Q_k^2)$, as will be seen in the following theorems.

Theorem 3.21. *$\chi(Q_k^2) \leq \chi(Q_{k+1}^2)$ for any $k \in \mathbb{Z}^+$.*

Proof. Assume, by way of contradiction, that $\chi(Q_k^2) > \chi(Q_{k+1}^2)$, and let $\chi(Q_{k+1}^2) = n$. Consider the copy of Q_k in Q_{k+1} induced by the vectors with a 0 in the last coordinate. Applying an n-coloring of Q_{k+1}^2 to this copy of Q_k yields an m-coloring of this copy for some $m \leq n$, a contradiction to the chromatic number of Q_k. □

Another well-known upper bound on $\chi(Q_k^2)$ will be proved in Theorem 3.22. Linial, Meshulam, and Tarsi proved this result in [12], and another proof was offered by Wan in [21, Theorem 2] that exhibits an appropriate coloring with the use of the binary representation matrix, as defined in Definition 1.18. We will prove this result in an analogous method to Wan, and show an example of this coloring applied to Q_3^2 in Example 3.23. Note that we have defined our binary representation matrix to be the transpose of that used by Wan, as we have represented our vectors as columns rather than rows.

Theorem 3.22. *$\chi(Q_k^2) \leq 2^{\lceil \log_2(k+1) \rceil}$ for any $k \in \mathbb{Z}^+$.*

Proof. Consider the binary representation matrix of order k, BRM_k, and define $n = \lceil \log_2(k+1) \rceil$. Now define the following function for all $v \in \{0,1\}^k$:

$$f(v) = (BRM_k)(v),$$

with matrix vector multiplication defined normally over the binary field. Note that $(BRM_k)(v)$ is a binary vector of length n, and hence $f : \{0,1\}^k \to \{0,1\}^n$. Since $|\{0,1\}^n| = 2^n$, we claim f is a 2^n-coloring of Q_k^2. In fact, f could be an m-coloring of Q_k^2 for some $m < 2^n$, which still yields the upper bound of 2^n. Consider arbitrary vertices u and v with $uv \in E(Q_k^2)$, that is, $1 \leq d(u,v) \leq 2$. We prove $f(u) \neq f(v)$. We consider two cases.

If $d(u,v) = 1$, u and v differ in exactly one coordinate in $[k]$, which we denote i. We have:

$$\begin{aligned}(BRM_k)(u) - (BRM_k)(v) &= (BRM_k)(u - v) \\ &= (BRM_k)(\mu_i^k) \\ &= (BRM_k)_i.\end{aligned}$$

where $(BRM_k)_i$ represents the ith column of BRM_k. However, $(BRM_k)_i$ is the binary vector of length n whose integer representation is i. Since $i \neq 0$ by definition of the binary representation matrix, $(BRM_k)(u) - (BRM_k)(v) \neq \vec{0}^n$ and hence $f(u) \neq f(v)$.

If $d(u,v) = 2$, u and v differ in exactly two coordinates in $[k]$, which we denote i_1 and i_2. We have:

$$\begin{aligned}(BRM_k)(u) - (BRM_k)(v) &= (BRM_k)(u - v) \\ &= (BRM_k)(\mu_{i_1}^k + \mu_{i_2}^k) \\ &= (BRM_k)_{i_1} + (BRM_k)_{i_2},\end{aligned}$$

which represents the sum of the binary vectors of length n with integer representations i_1 and i_2. However, since $i_1 \neq i_2$, $(BRM_k)(u) - (BRM_k)(v) \neq \vec{0}^n$. Therefore $f(u) \neq f(v)$. □

Example 3.23. Consider the case $k = 3$. Theorem 3.22 implies that:

$$\chi(Q_3^2) \leq 2^{\lceil \log_2(4) \rceil} = 2^2 = 4.$$

We will therefore exhibit a 4-coloring of Q_3^2. The binary representation matrix of order 3 is:

$$BRM_3 = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}.$$

The coloring f described in Theorem 3.22 is applied as in Table 3.1 (for simplicity we will convert $f(v)$ to its integer representation i).

Table 3.1: A 4-coloring of Q_3^2 using Wan's construction

$v \in \{0,1\}^3$	$(BRM_3)(v)$	i		$v \in \{0,1\}^3$	$(BRM_3)(v)$	i
$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \end{pmatrix}$	0		$\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 1 \end{pmatrix}$	3
$\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 0 \end{pmatrix}$	2		$\begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 1 \end{pmatrix}$	1
$\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 1 \end{pmatrix}$	1		$\begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 0 \end{pmatrix}$	2
$\begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 1 \end{pmatrix}$	3		$\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \end{pmatrix}$	0

This divides $\{0,1\}^3$ into the following 4 color classes:

$$\{v \mid (BRM_3)(v) = 0\} = \{000, 111\},$$
$$\{v \mid (BRM_3)(v) = 1\} = \{011, 100\},$$
$$\{v \mid (BRM_3)(v) = 2\} = \{010, 101\},$$
$$\{v \mid (BRM_3)(v) = 3\} = \{001, 110\}.$$

It is easily verified that this is a 4-coloring of Q_3^2. In fact, by Theorem 3.19, $\chi(Q_3^2) \geq 4$ so this coloring is optimal.

Theorem 3.22 will now allow us to determine exactly $\chi(Q_k^2)$ when $k = 2^n - i$ for some $i, n \in \mathbb{Z}^+$ with $n \geq 3$ and $i \in [4]$.

Theorem 3.24. *Let $k = 2^n - i$ for any $i, n \in \mathbb{Z}^+$ with $n \geq 3$ and $i \in [4]$. Then:*

$$\chi(Q_k^2) = 2^n.$$

Proof. The result of Theorem 3.22 implies that:

$$\begin{aligned}\chi(Q_k^2) &\leq 2^{\lceil \log_2(2^n - i + 1) \rceil} \\ &\leq 2^{\lceil \log_2(2^n) \rceil} \\ &= 2^n,\end{aligned}$$

and therefore, in addition with the result of Corollary 3.17, we have the desired result. □

One useful upper bound on $\chi(Q_n^2)$ occurs when n is odd. In this case, defining $n = 2k+1$, $\chi(Q_{2k+1}^2) \leq 2\chi(Q_k^2)$, as we will prove in Theorem 3.26 by means of the technique used in [17, Theorem 1]. An example of this bound applied to Q_3^2 will be seen in Example 3.27. The $u\_(u+v)$ construction used in the following lemma and theorem was introduced in [13, p.76], and is a common technique in building longer codes from shorter codes.

Lemma 3.25. *Let V be a $(k, M, 3)$ binary code for positive integers k and M. Define the following sets (where p_u and q_u are defined as in Chapter 1):*

$$\begin{aligned}V_p &= \{u\_p_u\_(u+v) \mid u \in \{0,1\}^k, v \in V\}, \\ V_q &= \{u\_q_u\_(u+v) \mid u \in \{0,1\}^k, v \in V\},\end{aligned}$$

V_p and V_q are $(2k+1, 2^k M, 3)$ binary codes.

Proof. The code length and size of V_p and V_q are trivially $2k+1$ and $2^k M$, respectively. We will prove the minimum distance of V_p is at least 3. Since $V_q = V_p + \mu_{k+1}^{2k+1}$, the result for V_q will follow immediately from Proposition 1.25.

Consider distinct $v_1^{2k+1}, v_2^{2k+1} \in V_p$. For $v_1, v_2 \in V$ and $u_1, u_2 \in \{0,1\}^k$, let:

$$v_1^{2k+1} = u_1\_p_{u_1}\_(u_1 + v_1),$$
$$v_2^{2k+1} = u_2\_p_{u_2}\_(u_2 + v_2).$$

We will show that $d(v_1^{2n+1}, v_2^{2n+1}) \geq 3$. First note the following:

$$\begin{aligned} d(u_1, u_2) + d(u_1 + v_1, u_2 + v_2) &= d(u_1, u_2) + wt(u_1 + v_1 + u_2 + v_2) \\ &= d(u_1, u_2) + d(u_2, u_1 + v_1 + v_2) \\ &\geq d(u_1, u_1 + v_1 + v_2) \\ &\qquad \text{by the triangle inequality} \\ &= wt(u_1 + v_1 + u_1 + v_2) \\ &= d(u_1 + v_1, u_1 + v_2) \\ &= d(v_1, v_2) \qquad \text{by Lemma 1.9.} \end{aligned}$$

Therefore, depending on whether or not $v_1 = v_2$ is satisfied, we have:

$$\begin{aligned} d(v_1^{2k+1}, v_2^{2k+1}) &= d(u_1, u_2) + d(p_{u_1}, p_{u_2}) + d(u_1 + v_1, u_2 + v_2) \\ &\geq \max\{2d(u_1, u_2) + d(p_{u_1}, p_{u_2}), d(v_1, v_2)\}. \end{aligned}$$

If $v_1 = v_2$, then $u_1 \neq u_2$. Moreover, $d(u_1, u_2) = 1$ implies $d(p_{u_1}, p_{u_2}) = 1$ (by Corollary 1.16), and thus:

$$\begin{aligned} d(v_1^{2k+1}, v_2^{2k+1}) &\geq 2d(u_1, u_2) + d(p_{u_1}, p_{u_2}) \\ &\geq 3. \end{aligned}$$

If $v_1 \neq v_2$, then $d(v_1, v_2) \geq 3$ since $v_1, v_2 \in V$. Hence:

$$\begin{aligned} d(v_1^{2k+1}, v_2^{2k+1}) &\geq d(v_1, v_2) \\ &\geq 3. \end{aligned}$$

□

Theorem 3.26. *$\chi(Q_{2k+1}^2) \leq 2\chi(Q_k^2)$ for any positive integer k.*

Proof. Fix $k \in \mathbb{Z}^+$. We will show this upper bound on $\chi(Q_{2k+1}^2)$ by creating a $2\chi(Q_k^2)$-coloring of Q_{2k+1}^2. Take an optimal coloring of Q_k^2 and denote its color classes as $V_j^k \subseteq \{0,1\}^k$ for $j \in [\chi(Q_k^2)]$. Clearly, we have:

$$\sum_{j=1}^{\chi(Q_k^2)} |V_j^k| = 2^k.$$

We will use these color classes to build color classes in a coloring of Q^2_{2k+1}. To create our color classes of Q^2_{2k+1}, we define (for $j \in [\chi(Q^2_k)]$):

$$V^{2k+1}_{p,j} = \{u\_p_u\_(u+v) \mid u \in \{0,1\}^k, v \in V^k_j\},$$
$$V^{2k+1}_{q,j} = \{u\_q_u\_(u+v) \mid u \in \{0,1\}^k, v \in V^k_j\}.$$

Obviously, we have created $2\chi(Q^2_k)$ such sets. We claim these sets are color classes in a $2\chi(Q^2_k)$-coloring of Q^2_{2k+1}. The sum of the vertices in the sets can be determined as follows:

$$\begin{aligned}
\sum_{\rho\in\{p,q\}} \sum_{j=1}^{\chi(Q^2_k)} |V^{2k+1}_{\rho,j}| &= \sum_{\rho\in\{p,q\}} \sum_{j=1}^{\chi(Q^2_k)} 2^k |V^k_j| \\
&= \sum_{\rho\in\{p,q\}} (2^k)(2^k) \\
&= \sum_{\rho\in\{p,q\}} 2^{2k} \\
&= (2)(2^{2k}) \\
&= 2^{2k+1} \\
&= n(Q^2_{2k+1}).
\end{aligned}$$

Moreover, Lemma 3.25 implies that these sets have minimum distance 3, and are thus independent in Q^2_{2k+1}. Therefore, we need only prove that they are pairwise disjoint, and as such their union contains every vertex of Q^2_{2k+1}. Assume, by way of contradiction, that for some $\rho, \theta \in \{p,q\}$ and $j_1, j_2 \in [\chi(Q^2_k)]$:

$$V^{2k+1}_{\rho,j_1} \cap V^{2k+1}_{\theta,j_2} \neq \emptyset,$$
$$V^{2k+1}_{\rho,j_1} \neq V^{2k+1}_{\theta,j_2}.$$

Take $v^{2k+1} \in V^{2k+1}_{\rho,j_1} \cap V^{2k+1}_{\theta,j_2}$. We have, for some $u_1, u_2 \in \{0,1\}^k$, $v_1 \in V^k_{j_1}$, and $v_2 \in V^k_{j_2}$:

$$u_1\_\rho_{u_1}\_(u_1+v_1) = u_2\_\theta_{u_2}\_(u_2+v_2).$$

Thus we must have $u_1 = u_2$. Trivially, u_1 and u_2 then are of the same parity; so for $\rho_{u_1} = \theta_{u_2}$, we must have $\rho = \theta$. Finally, since $u_1 = u_2$ and $u_1 + v_1 = u_2 + v_2$, Lemma 1.9 implies that $v_1 = v_2$. Since $v_1 = v_2$, we must

have $j_1 = j_2$ as the V_j^k form a partition of $\{0,1\}^k$. Thus $V_{\rho,j_1}^{2k+1} = V_{\theta,j_2}^{2k+1}$, a contradiction. Therefore these sets form color classes of a $2\chi(Q_k^2)$-coloring of Q_{2k+1}^2. □

Example 3.27. Let us look at the case $k = 1$. $\chi(Q_1^2) = 2$, with the optimal coloring having the following trivial color classes:

$$V_1^1 = \{0\},$$
$$V_2^1 = \{1\}.$$

We use these sets to build our color classes for an optimal coloring of Q_3^2. So, we apply our general formula:

$$V_{p,j}^3 = \{u\_p_u\_(u+v) | u \in \{0,1\}, v \in V_j^1\},$$
$$V_{q,j}^3 = \{u\_q_u\_(u+v) | u \in \{0,1\}, v \in V_j^1\},$$

to build:

$$V_{p,1}^3 = \{000, 111\},$$
$$V_{q,1}^3 = \{010, 101\},$$
$$V_{p,2}^3 = \{001, 110\},$$
$$V_{q,2}^3 = \{011, 100\}.$$

It is easily determined that these are the color classes of a 4-coloring of Q_3^2. In fact, these color classes are the same as those realized in the optimal coloring of Q_3^2 determined in Example 3.23.

3.2.1 The Asymptotic Behavior of the Chromatic Number of the Square of the Cube

The result of Theorem 3.21 states that $\chi(Q_k^2)$ is nondecreasing as k increases. In fact, Östergård proved the following result in [17, Theorem 5].

Theorem 3.28.

$$\lim_{k\to\infty} \frac{\chi(Q_k^2)}{k} = 1.$$

Therefore, as k increases, $\chi(Q_k^2)$ approaches k. From Theorem 3.24, if $k = 2^n - i$ for $n \geq 3$ and $i \in [4]$, then $\chi(Q_k^2) = 2^n$. Moreover, if $k = 2^n$ for $n \in \mathbb{Z}^+$, then Corollary 3.20 yields $\chi(Q_k^2) \geq 2^n + 2$. Therefore we can think of

$2^n - i$ as a "tail" of values satisfying $\chi(Q^2_{2^n-i}) = 2^n$, with $i = 1$ representing the uppermost value at the base of the tail. We will now describe the length of this tail, T. Let $T : \mathbb{Z}^+ \setminus \{1, 2\} \to \mathbb{Z}^+$ be the maximum positive integer i such that $\chi(Q^2_{2^n-i}) = 2^n$. It must be the case that:

$$4 \leq T(n) \leq 2^n - 2^{n-1} = 2^{n-1}.$$

Based on the bounds in Appendix A we have $T(3) = 4$, $4 \leq T(4) \leq 7$, and $4 \leq T(5) \leq 14$. The result of Theorem 3.28 implies that $T(n)$ gets relatively smaller as n tends to ∞, as we will show in Corollary 3.29. In other words, the lengths of these tails of values of k in which the chromatic number of Q^2_k is a power of 2 gets relatively smaller as k increases.

Corollary 3.29. *For every $\lambda \in (0, 1)$, there exists some $N \in \mathbb{Z}^+$ such that $\dfrac{T(n)}{2^n} < \lambda$ for all $n > N$.*

Proof. Take arbitrary $\lambda \in (0, 1)$ and define $\lambda_0 = 2^{\lfloor \log_2(\lambda) \rfloor}$. Clearly $\lambda_0 \leq \lambda$. Given a positive integer n, let $k = 2^{n-1} + (1 - 2\lambda_0)2^{n-1}$. Clearly, there exists a positive integer N_1 such that k is an integer for all $n > N_1$. Moreover, by Theorem 3.19, we have:

$$\frac{\chi(Q^2_k)}{k} - 1 \geq \frac{k+1}{k} - 1 > 0.$$

We now apply Theorem 3.28. For all $\epsilon > 0$ there exists a positive integer N_2 such that $n > N_2$ implies:

$$\frac{\chi(Q^2_k)}{k} - 1 < \epsilon. \tag{3.2}$$

Take $\epsilon = \dfrac{2}{2 - 2\lambda_0} - 1$ and fix the N_2 that satisfies (3.2). Let $N = \max\{N_1, N_2\}$, and hence $n > N$ implies:

$$\begin{aligned}
\chi(Q^2_k) &< \left(1 + \frac{2}{2 - 2\lambda_0} - 1\right) k \\
&= \left(\frac{2}{2 - 2\lambda_0}\right) \left(2^{n-1} + (1 - 2\lambda_0)2^{n-1}\right) \\
&= \left(\frac{2}{2 - 2\lambda_0}\right) (2 - 2\lambda_0) \left(2^{n-1}\right) \\
&= (2)2^{n-1} \\
&= 2^n.
\end{aligned}$$

Since $\chi(Q_k^2) < 2^n$, by the definition of $T(n)$, $n > N$ implies:

$$\begin{aligned} T(n) &< 2^n - k \\ &= 2^n - \left(2^{n-1} + (1 - 2\lambda_0)2^{n-1}\right) \\ &\leq 2^n - (2 - 2\lambda_0)(2^{n-1}) \\ &= 2^n - 2^n + 2\lambda_0(2^{n-1}) \\ &= \lambda_0 2^n \\ &\leq \lambda 2^n. \end{aligned}$$

□

3.3 Applying Computational Methods to Determine the Chromatic Number of the Square of 8-Dimensional Cube

As seen in Appendix A, the smallest k such that $\chi(Q_k^2)$ is unknown is $k = 8$. This graph has $2^8 = 256$ vertices and $2^7\binom{9}{2} = 4608$ edges. Although the actual value of $\chi(Q_k^2)$ is unknown, we know from Theorem 1.60 that $A(8,3) = 20$ (by Theorem 1.44 we have $A(8,3) = A(9,4)$). Thus, we can use Theorem 3.16 to determine a lower bound on $\chi(Q_8^2)$ as follows:

$$\chi(Q_8^2) \geq \left\lceil \frac{256}{20} \right\rceil = \lceil 12.8 \rceil = 13.$$

Moreover, 14-colorings of Q_8^2 are known to exist. This was first exhibited by Hougardy in 1991 [23, p.168] from an adaptation of the Petford-Welsh algorithm described in Chapter 2.4.4. This adaptation will be covered in more detail in Chapter 3.3.3. Another 14-coloring was found independently by Royle in 1993 by means of a randomized search [11, p.157]. Hence, these colorings provide an upper bound on its chromatic number, so $\chi(Q_8^2)$ is either 13 or 14. Therefore, in order to determine the chromatic number, one either needs to find a 13-coloring, or prove that one does not exist.

Also, since there can be no color class of size greater than 20, if a 13-coloring exists, its color classes must be of one of the following 5 cases:

- 12 independent sets of size 20 and 1 of size 16,
- 11 independent sets of size 20, 1 of size 19, and 1 of size 17,
- 11 independent sets of size 20 and 2 of size 18,

- 10 independent sets of size 20, 2 of size 19, and 1 of size 18, and
- 9 independent sets of size 20 and 4 of size 19.

These cases show that, if searching for a 13-coloring of Q_8^2, independent sets of size 15 or less can be ignored, as they could not possibly be a color class of a 13-coloring.

The three algorithms introduced in Chapter 2.4 were then applied to the graph Q_8^2 in an effort to determine the chromatic number. However, the computational difficulty associated with this graph turned out to be far greater than expected.

3.3.1 Integer Programming

Consider the IP model to determine the existence of an m-coloring in a graph described in Chapter 2.4.2. We can denote every vertex of Q_k^2 by its integer representation, and create the binary variables $x_{i,j}$ for all i in $\{0, 1, \ldots, 2^k - 1\}$ and j in $[m]$. A feasible solution to the following IP would then be an m-coloring of Q_k^2 (letting $d(i_1, i_2)$ denote the Hamming distance between the binary representations of i_1 and i_2):

$$x_{i_1,j} + x_{i_2,j} \leq 1 \qquad \text{for all } i_1 \text{ and } i_2 \text{ in } \{0, 1, \ldots, 2^k - 1\} \text{ and } j \text{ in } [m], \text{ such that } 1 \leq d(i_1, i_2) \leq 2$$

$$\sum_{j=1}^{m} x_{i,j} \geq 1 \qquad \text{for all } i \text{ in } \{0, 1, \ldots, 2^k - 1\},$$

$$x_{i,j} \in \{0, 1\} \qquad \text{for all } i \text{ in } \{0, 1, \ldots, 2^k - 1\} \text{ and } j \text{ in } [m].$$

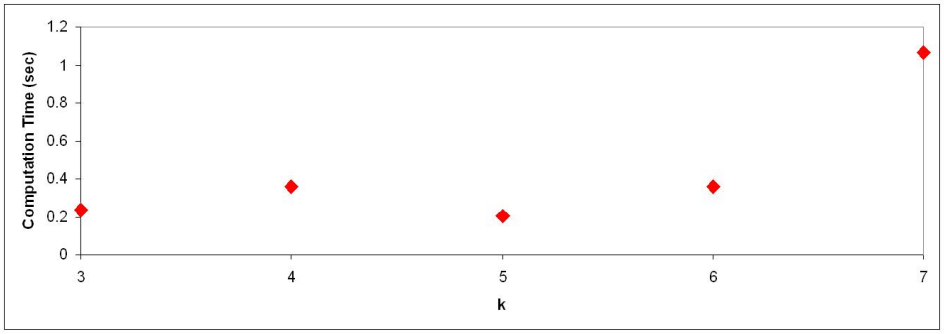
We then applied this model to determine if Q_8^2 is 13-colorable (i.e. the case where $k = 8$ and $m = 13$). Computationally, we modeled this IP in C++ with CPLEX 9.0, a powerful mixed integer program optimizer designed by ILOG, the world's leading combinatorial optimization tool provider. However, the number of variables in this IP are $13n(Q_8^2) = (13)(256) = 3,328$ and the number of constraints are $e(Q_8^2) + n(Q_8^2) = 4,608 + 256 = 60,160$. After CPLEX's internal presolver was applied, the reduced IP matrix contained $13,607$ rows, $3,328$ columns, and $72,722$ nonzero entries. However, it is possible to impose several additional constraints.

Consider the clique in Q_k^2 defined as $V = \{\vec{0}^8, \mu_1^8, \mu_2^8, \ldots, \mu_k^8\}$ (this was proven to be a clique in Lemma 3.18). It is clear that every vertex in V must be in a separate color class. Without loss of generality, we can therefore add the following constraints:

$$x_{0,1} = 1, \quad x_{2^0,2} = 1, \quad x_{2^1,3} = 1, \quad x_{2^2,4} = 1, \ldots \quad , \quad x_{2^k,k+1} = 1.$$

Including these constraints allowed CPLEX's presolve to eliminate many more variables and constraints (such the variables representing vertices adjacent to vertex μ_i^k in color class $i+1$, which must of course equal 0). We applied this IP to find optimal colorings for all $3 \leq k \leq 7$ (that is, a 4-coloring of Q_3^2 and 8-colorings of Q_k^2 for $4 \leq k \leq 7$). The times to find these colorings are included in Figure 3.1, and were obviously very reasonable.

Figure 3.1: Computation Times Using Integer Programming



However, applied to Q_8^2, the new reduced IP matrix contained $8,431$ rows, $2,959$ columns, and $47,784$ nonzero entries. This problem was unfortunately too large to compute in a reasonable amount of time. After running for over 23 hours, CPLEX had only explored $1,040$ out of a maximum possible $2^{2,959}$ nodes of the branch and bound tree. No effort was made to help CPLEX determine the variable choice in the branch and bound. Instead, the defaults were used. Given the vertex transitivity of the graph and large number of symmetries of the IP model previously discussed, it seemed more worthwhile to use another computational method.

3.3.2 Column Generation

We then applied the column generation algorithm described in Chapter 2.4.3, again implemented in C++ with CPLEX 9.0. This was done first with the initial set S being the singleton sets of vertices. Moreover, we could impose the restriction on the MWIS phase of the algorithm that we only needed to generate independent sets of size 16 or greater, as prior knowledge indicated that a 13-coloring of Q_k^2 could not contain any color classes of size less than 16. This approach showed much promise; however, the computation time at each node of the tree turned out to be prohibitive.

For example, to solve the initial node of branch and bound tree, it took $2,388$ iterations of the IS-MWIS column generation method. This process took just over 35 minutes on a 2 G.Hz. Pentium processor with 2 G.B. of R.A.M. After approximately 19 hours, 18 nodes had been explored. The branch and bound tree for this method has maximum size of $2^{32,640}$ nodes (as $\binom{256}{2} = 32,640$). This approach was abandoned since it seemed that an optimal coloring was unlikely to be determined in a reasonable about of time. Varying the choice of the initial set of independent sets S showed no noticeable improvement.

On the other hand, when applied to Q_7^2, the initial node took only 18 iterations and 1 second to perform on the same machine. It also turned out to find the optimal coloring with the independent sets generated in the first node.

Part of the reason we were unable to bring the computation time at each node down to a manageable length of time is due to the immense number of columns available for the MWIS portion of the algorithm to choose from. From Proposition 2.25, we know that, given an independent set V and automorphism Φ, that $\Phi(V)$ is independent. Therefore, given any independent set V, we can generate a collection of independent sets by producing the independent set $\Phi(V)$ for every automorphism Φ. It is unlikely that every $\Phi(V)$ will be distinct. However, if a graph has many automorphisms, the number of generated independent sets that are distinct will still likely be very large. Recall from Theorem 3.14 that $|Aut(Q_k^2)| = 2^k(k+1)!$. Therefore, there exist $2^8 9! = 92,897,280$ automorphisms of Q_8^2, and given a single independent set V, we can immediately generate over 90 million (again, not necessarily distinct) independent sets of the same size.

The challenges associated with the column generation algorithm for determining a 13-coloring of Q_8^2 led us naturally to looking at properties of these independent sets. In Chapter 4, we will attempt to determine their structure.

3.3.3 The Petford-Welsh Algorithm

Recall the Petford-Welsh algorithm described in Chapter 2.4.4. As stated previously, Hougardy exhibited a 14-coloring of Q_8^2 with an adaptation of this algorithm. We were able to reproduce his results, and find a 14-coloring of Q_8^2 with a running time of within several seconds. The C++ code of our implementation of Hougardy's adaptation is included in Appendix D, under the file name Hougardy.cpp. The 14-coloring generated with this algorithm is included in Appendix B. However, attempts with varying values for the probability input parameter did not yield a 13-coloring. This, of course, does not prove the non-existence of such a coloring.

Chapter 4

The Structure of Optimal Binary Codes

Recall from the Chapter 3.3 that if a 13-coloring of Q_8^2 exists, it contains at least 9 independent sets of size 20, and possibly as many as 12. Therefore, it contains at least 9 disjoint $(8, 20, 3)$ binary codes. The knowledge of the structure of these codes is thus very important in proving the existence or non-existence of a 13-coloring, or providing MIP cuts to speed up the exploration of the branch and bound (cut) tree.

4.1 Computationally Determining all Weight Distributions

We know from Chapter 1 that every binary code has a weight distribution with respect to any vector u. One may wish to determine how many sequences of integers are a feasible weight distribution of some (n, M, d) binary code C. Since $C + u$ is an (n, M, d) binary code whose weight distribution (with respect to $\vec{0}^n$) is the same as the weight distribution of C with respect to u, a sequence of integers is a feasible weight distribution with respect to any vector u if and only if it is a feasible weight distribution with respect to $\vec{0}^n$.

Given a specific sequence of integers, $(W_i)_{i=0}^n$, it is easy to set up an IP that determines if there is some (n, M, d) binary code with that weight distribution. We create a binary variable v_j for all j in $\{0, 1, \ldots, 2^n - 1\}$. This variable is set to equal 1 if the binary representation of integer j is included in C, and 0 otherwise. We will let $wt(j)$ and $d(j_1, j_2)$ be defined with respect to the appropriate binary representations. We then determine if the following system of constraints has a feasible solution:

$$\begin{aligned}
\sum_{j=0}^{2^n-1} v_j &= M, \\
\sum_{j|wt(j)=i} v_j &= W_i \qquad \text{for all } i \text{ in } \{0,1,\ldots,n\}, \\
v_{j_1} + v_{j_2} &\leq 1 \qquad \text{for all } j_1 \text{ and } j_2 \text{ in } \{0,1,\ldots,n\} \\
&\qquad\qquad \text{satisfying } 1 \leq d(j_1,j_2) \leq d-1.
\end{aligned}$$

It is clear that if there is a feasible solution to these constraints, then $C = \{j|v_j = 1\}$ is an (n, M, d) binary code. Otherwise, if this IP is infeasible, then no such code exists. However, this can be a very computationally expensive procedure. Fortunately, the knowledge of permutations can allow us to fix at least one of the codewords in C and speed this up dramatically.

Proposition 4.1. *Let $C \subseteq \{0,1\}^n$ be a binary (n, M, d) code with weight distribution $(W_i)_{i=0}^n$. Let $v \in C$ with $wt(v) = w$ and consider any u in $\{0,1\}^n$ with $wt(u) = w$. Then there exists another binary (n, M, d) code C' such that $u \in C'$ and C' also has weight distribution $(W_i)_{i=0}^n$.*

Proof. Consider the permutation σ in S_k such that $\sigma(v) = u$, which exists since $wt(u) = wt(v)$. Define $C' = \sigma(C)$. It is clear that C' is an (n, M, d) binary code that also has weight distribution $(W_i)_{i=0}^n$. In addition, since $v \in C$, we have $u \in C'$. □

In this chapter, we will be determining the feasible weight distributions of optimal binary codes of length 8 and minimum distance 3. It was proven in Theorem 1.60 that these are $(8, 20, 3)$ binary codes. From [1, Table II], it is known that $A(8, 6, 4) = 2$. Therefore, in determining if $(W_i)_{i=0}^8$ is a feasible weight distribution of an $(8, 20, 3)$ binary code C, consider the case where $W_4 \geq 3$. In this case, we can in fact fix two codewords to any u and v in C such that $wt(u) = wt(v) = 4$ and $d(u, v) = 4$. We will be using the codewords 00001111 and 00110011, whose integer representations are 15 and 51, respectively. The IP to determine if $(W_i)_{i=0}^8$ is a feasible weight distribution of an $(8, 20, 3)$ binary code C is then defined by the following system of constraints, denoted (FD) for feasible distributions:

$$\begin{aligned}
\sum_{j=0}^{255} v_j &= 20, \\
\sum_{j|wt(j)=i} v_j &= W_i \qquad \text{for all } i \text{ in } \{0,1,\ldots,8\}, \\
v_{j_1} + v_{j_2} &\leq 1 \qquad \text{for all } j_1 \text{ and } j_2 \text{ in } \{0,1,\ldots,8\} \\
&\qquad\qquad \text{satisfying } 1 \leq d(j_1,j_2) \leq 2, \\
v_{15} &= 1 \qquad \text{if } W_4 \geq 1, \\
v_{51} &= 1 \qquad \text{if } W_4 \geq 3.
\end{aligned}$$

Thus we could run this IP on every sequence of positive integers $(W_i)_{i=0}^8$, where $\sum_{i=0}^{8} W_i = 20$. Unfortunately, there are $\binom{28}{8} = 3,108,105$ such sequences. However, the result of Theorem 1.48 allows us to impose $W_i \leq A(8,3,i) = A(8,4,i)$ for all i.

Proposition 1.46 implies that $A(8,3,0) = A(8,3,1) = 1$, and hence $A(8,3,8) = A(8,3,7) = 1$. In addition, Proposition 1.47 implies that $A(8,3,2) = A(8,4,2) = \left\lfloor \frac{8}{2} \right\rfloor = 4$, and hence $A(8,3,6) = 4$. Finally, [1, Table I] contains the values $A(8,3,3) = A(8,3,5) = 8$ and $A(8,3,4) = 14$. Applying all of these constraints on the W_i dramatically reduces the number of sequences that must be tested. In addition, we will apply the result of the following theorem and corollary.

Theorem 4.2. *Let C be an $(8,M,3)$ binary code with distance distribution $(W_i)_{i=0}^8$. Then $W_0 + W_1 + W_2 \leq 4$.*

Proof. Assume, by way of contradiction, there exists an $(8,M,3)$ binary code with weight distribution $(W_i)_{i=0}^8$ with $W_0 + W_1 + W_2 \geq 5$. If $\vec{0}^8 \in C$, $W_1 = W_2 = 0$ as all v in $\{0,1\}^8$ with $wt(v) \leq 2$ is at distance at most 2 from $\vec{0}^8$. Thus $\vec{0}^8 \notin C$ as $W_1 + W_2 \leq 4$.

Since $W_1 \leq A(8,3,1) = 1$ and $W_2 \leq A(8,3,2) = 4$, we must have $W_1 = 1$ and $W_2 = 4$. Let $u, v_1, v_2, v_3, v_4 \in C$ with $wt(u) = 1$ and $wt(v_i) = 2$ for all i

in [4]. We have $wt(v_i \cap v_j) = 0$ for i and j in [4] with $i \neq j$ as otherwise:

$$\begin{aligned} d(v_i, v_j) &= wt(v_i) + wt(v_j) - 2wt(v_i \cap v_j) \qquad \text{by Lemma 1.15} \\ &\leq 2 + 2 - 2 \\ &= 2. \end{aligned}$$

Therefore $\sum_{i=1}^{4} v_i = \vec{1}^8$, and hence $wt(u \cap v_i) = 1$ for some i in [4]. This implies:

$$d(u, v_i) = wt(u) + wt(v_i) - 2wt(u \cap v_i) = 1 + 2 - 2 = 1,$$

a contradiction. □

Corollary 4.3. *Let C be an $(8, M, 3)$ binary code with weight distribution $(W_i)_{i=0}^8$. Then $W_6 + W_7 + W_8 \leq 4$.*

Proof. Assume, by way of contradiction, there exists an $(8, M, 3)$ binary code with weight distribution $(W_i)_{i=0}^8$ with $W_6 + W_7 + W_8 \geq 5$. Let $(W_i')_{i=0}^8$ be the weight distribution of $C + \vec{1}^8$. We have $W_0' + W_1' + W_2' = W_8 + W_7 + W_6 \geq 5$; however, $C + \vec{1}^8$ is an $(8, M, 3)$ binary code by Proposition 1.25. This contradicts the result of Theorem 4.2. □

Therefore, to determine all sequences of integers to test as feasible weight distributions for an $(8, 20, 3)$ binary code, we must enumerate all feasible solutions to the following system, denoted (WD) for weight distributions:

$$\sum_{i=0}^{8} W_i = 20 \tag{4.1}$$

$$W_0 \leq 1, \tag{4.2}$$

$$W_1 \leq 1, \tag{4.3}$$

$$W_2 \leq 4, \tag{4.4}$$

$$W_3 \leq 8, \tag{4.5}$$

$$W_4 \leq 14, \tag{4.6}$$

$$W_5 \leq 8, \tag{4.7}$$

$$W_6 \leq 4, \tag{4.8}$$

$$W_7 \leq 1, \tag{4.9}$$

$$W_8 \leq 1, \tag{4.10}$$

$$W_0 + W_1 + W_2 \leq 4, \tag{4.11}$$

$$W_6 + W_7 + W_8 \leq 4, \tag{4.12}$$

$$W_0 = 1 \Rightarrow W_1 + W_2 = 0 \tag{4.13}$$

$$W_8 = 1 \Rightarrow W_6 + W_7 = 0 \tag{4.14}$$

$$W_i \in \mathbb{Z}^+ \cup \{0\} \qquad \text{for all } i \text{ in } \{0, 1, \ldots, 8\},$$

$$z_i \in \{0, 1\} \qquad \text{for all } i \text{ in } [4].$$

Constraint (4.1) forces the code to have size 20. Constraints (4.2) through (4.10) enforce the $W_i \leq A(8, 3, i)$ inequalities. Constraints (4.11) and (4.12) follow from Theorem 4.2 and Corollary 4.3, respectively. Moreover, constraint (4.13) encodes the statement that if $W_0 = 1$, then $W_1 = W_2 = 0$, as was determined in the proof of Theorem 4.2. Finally, constraint (2.14) analogously encodes the statement that if $W_8 = 0$, then $W_6 = W_7 = 0$.[3]

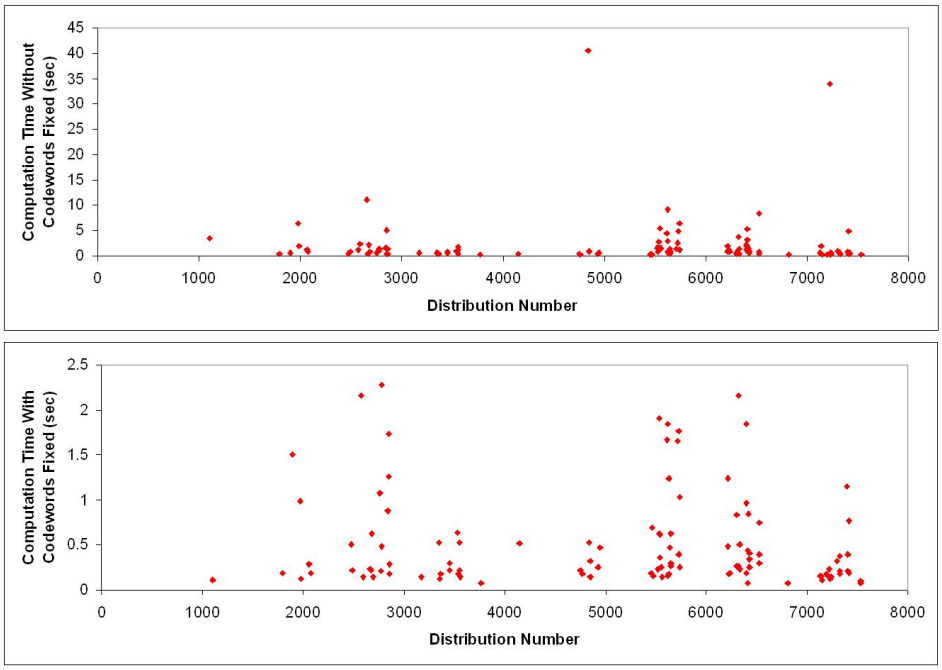
These constraints were then inputted into the constraint programming solver ILOG Solver 6.0, and $7,539$ possible solutions were enumerated. Each of these solutions were then inputted into the (FD) IP, and solved for feasibility as a weight distribution using the mixed IP optimizer ILOG CPLEX 9.0. The C++ code that solved both (WD), and every (FD) iteration, is named CPDist.cpp, and included in Appendix E. In the end, 111 of the distributions were proven feasible, whereas $7,428$ of the distributions were proven infeasible. All of the feasible weight distributions found are then included in Appendix C. All computation was done on a computer with a 2.19 G.Hz. A.M.D. processor with 1.0 G.B. of R.A.M.

To show how important the codeword fixing constraints are in the (FD) model, we reexecuted the IP without the two conditional constraints for all 111 feasible distributions, and 102 of the $7,428$ infeasible distributions selected at random. The computation times with and without the conditional constraints for these 111 feasible and 102 infeasible distributions were compared.

[3]Implications are allowed in CP models and are implemented quite differently from the standard ways they are modeled in MIPs.

The computation times with and without the conditional constraints for the feasible distributions are shown in Figure 4.1.

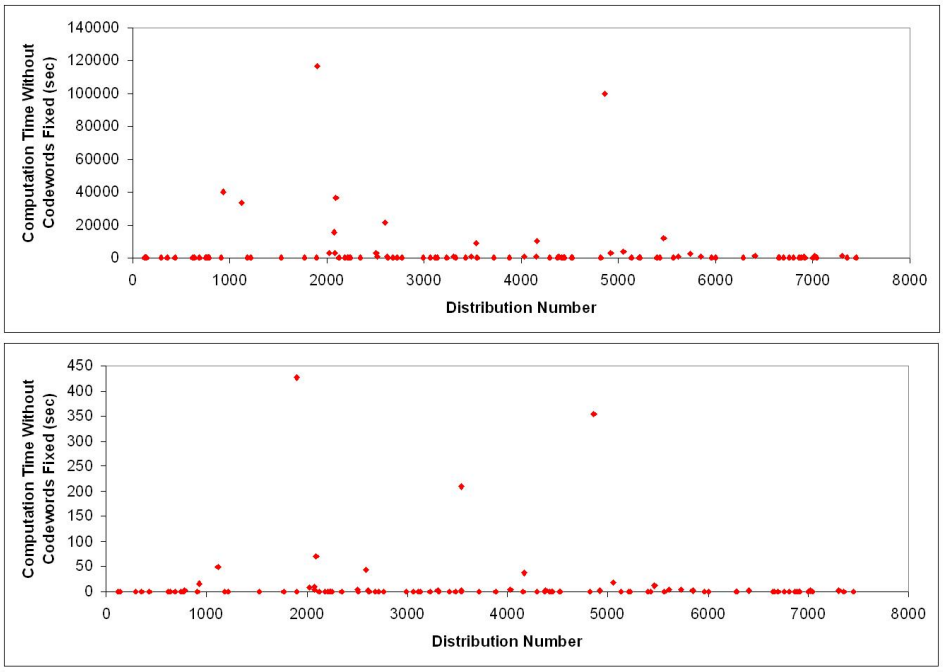
Figure 4.1: Computation Times for Feasible Distributions



The mean computation time with the constraints was 0.52 seconds, with a maximum of 2.28 seconds. The mean computation time with the constraints removed was 2.16 seconds, with a maximum of 40.49 seconds. Clearly the relative increase in times is significant; however, with only 111 distributions it did not make much difference in absolute terms.

The computation times with and without the conditional constraints for the randomly selected infeasible distributions are shown in Figure 4.2.

Figure 4.2: Computation Times for Infeasible Distributions



Here, it can be seen that including these constraints had a substantial impact on computation time. The mean computation time with the constraints was 12.62 seconds, with a maximum of 425.64 seconds, or approximately 7 minutes. The mean computation time with the constraints removed was $2,064$ seconds, or about 34 minutes. The longest computation took $116,872$ seconds, or over 32 hours. With over 7000 distributions in which to determine infeasibility, it is apparent that determining infeasibility in all such distributions would be extremely impractical without applying the knowledge of permutations to fix codewords.

4.2 Structural Results

4.2.1 On the Size of Shortened Codes

It is easily determined from the table in Appendix C that every $(8, 20, 3)$ binary code contains between 8 and 12 codewords of even weight (and thus between 8 and 12 codewords of odd weight). This knowledge can be used to prove the result that every $(8, 20, 3)$ binary code has between 8 and 12 codewords with a specified value in each coordinate, as will be proved in Corollary 4.5.

Theorem 4.4. *Let C be an $(n, M, 3)$ binary code with $n \geq 3$, and fix $t = \left\lfloor \frac{n}{2} \right\rfloor$. For any i in $[n]$ and x in $\{0, 1\}$, C can be transformed into an $(n, M, 3)$ binary code D whose weight distribution satisfies, letting $(W_i)_{i=0}^{n}$ denote the weight distribution of D:*

$$\sum_{i=0}^{t} W_{2i} = |\{u \in C \mid \textit{the } i\textit{th coordinate of } u \textit{ is } x\}|.$$

Proof. First consider the case where $x = 0$. Fix an arbitrary i in $[n]$, and let D be defined as follows:

$$D = \{\tau_i(u) \mid u \in C\}.$$

Recall from the definition of τ_i that if the ith coordinate of u is 0, then $\tau_i(u)$ is of even parity. Moreover, if the ith coordinate of u is 1, then $\tau_i(u)$ is of odd parity. In addition, by Theorem 3.13, τ_i is an automorphism of Q_n^2. Therefore, by Proposition 3.4, since C is an $(n, M, 3)$ binary code, D is an $(n, M, 3)$ binary code. We then have:

$$\begin{aligned}\sum_{i=0}^{t} W_{2i} &= |\{v \in D \mid wt(v) \text{ is even}\}| \\ &= |\{u \in C \mid \text{the } i\text{th coordinate of } u \text{ is } 0\}|.\end{aligned}$$

In the case where $x = 1$, we instead define D to be:

$$D = \{\tau_i(u) + \mu_i^n \mid u \in C\}.$$

D is again an $(n, M, 3)$ binary code and in this case:

$$\begin{aligned}\sum_{i=0}^{t} W_{2i} &= \{v \in D \mid wt(v) \text{ is even}\} \\ &= |\{u \in C \mid \text{the } i\text{th coordinate of } u \text{ is } 1\}|.\end{aligned}$$

□

Corollary 4.5. *Let C be an $(8, 20, 3)$ binary code. Then, for any i in $[8]$ and x in $\{0, 1\}$,:*

$$8 \leq |\{u \in C \mid \textit{the } i\textit{th coordinate of } u \textit{ is } x\}| \leq 12.$$

Proof. Assume by way of contradiction, for arbitrary i in $[8]$ and x in $\{0, 1\}$ that:

$$|\{u \in C \mid \text{the } i\text{th coordinate of } u \text{ is } x\}| = M',$$

where $M' \leq 7$ or $M' \geq 13$. Then, by Theorem 4.4, C can be transformed into an $(8, 20, 3)$ binary code D with M' codewords of even weight. This, however, can't exist due to the computational results of Chapter 4.1. □

Another way of expressing this result is that for any $(8, 20, 3)$ binary code C, consider the shortened code $C_{i,x}$ for any i in $[8]$ and x in $\{0, 1\}$. The code size of this shortened code must satisfy:

$$8 \leq |C_{i,x}| \leq 12.$$

4.2.2 Bounds on the Weight Distributions of Color Classes

The weight distributions listed in Appendix C can provide many interesting results about the color classes of what a 13-coloring of Q_8^2 must look like. First and foremost, assume there exists a 13-coloring of Q_8^2 and denote its color classes C_j for $j \in [13]$. Moreover, let $(W_i^j)_{i=0}^8$ denote the weight distribution of C_j. It is clear that:

$$\sum_{j=0}^{13} W_i^j = \binom{8}{i},$$

for all $i \in \{0, 1, \ldots, 8\}$. Since between 9 and 12 of these C_j must be of size 20, this greatly restricts the combinations of weight distributions that could form a possible 13-coloring of Q_8^2. In fact, the algorithm described in Chapter 4.1 can trivially be modified to generate the weight distributions of all $(8, M, 3)$ codes for $M < 20$. Unfortunately, this led to too many cases to reasonably keep track of. However, useful bounds can still be found.

For example, recall from [1, Table I] that $A(8, 3, 4) = 14$. Therefore, any binary code with length 8 and minimum distance at least 3 can have a maximum of 14 codewords whose weight is 8. However, as can be seen in

Appendix C, any optimal $(8, 20, 3)$ binary code C with weight distribution $(W_i)_{i=0}^{8}$ must satisfy $W_4 \leq 10$. Moreover, if $W_4 \geq 9$, then $W_0 + W_8 \geq 1$. Therefore at least one of $\vec{0}^8$ and $\vec{1}^8$ must belong to C. Therefore, if a 13-coloring of Q_8^2 exists, at most 2 of the minimum 9 color classes of size 20 can have at least 9 codewords of weight 4. Therefore, at least 7 color classes of size 20 must have no more than 8 codewords of weight 4.

Moreover, if $W_3 = A(8, 3, 3) = 8$, then C follows weight distribution 109 or 110. Therefore $W_0 = 1$, and hence there can be at most one color class with 8 codewords of weight 3. An analogous argument shows that there can be at most one color class with 8 codewords of weight 5.

Bounds like these can be added to MIPs or other models for finding, or proving the non-existence of, a 13-coloring of Q_8^2. Attempts were made in this direction.

4.3 The Number of Nonequivalent Codes

Consider the following five $(8, 20)$ binary codes (given in terms of their integer representations), which can be easily verified to be $(8, 20, 3)$ binary codes.

$$\begin{aligned}
C_1 = \{&0, 7, 25, 30, 42, 53, 75, 84, 97, 108, 114, 127, 140, 147, 169, 182,\\
&194, 221, 231, 248\},\\
C_2 = \{&0, 7, 25, 30, 42, 53, 75, 84, 97, 108, 114, 127, 140, 147, 166, 169,\\
&176, 194, 197, 216\}.\\
C_3 = \{&0, 43, 53, 62, 79, 83, 92, 102, 121, 135, 154, 157, 172, 179, 201, 214,\\
&229, 234, 240, 255\},\\
C_4 = \{&15, 26, 38, 51, 66, 73, 84, 101, 120, 127, 133, 144, 172, 185, 206, 211,\\
&221, 224, 235, 246\},\\
C_5 = \{&15, 20, 26, 37, 40, 51, 67, 76, 118, 121, 134, 145, 171, 188, 201, 210,\\
&223, 224, 238, 245\}.
\end{aligned}$$

The distance distributions of these five codes are easily calculated, and seen in Table 4.1.

Table 4.1: Distance Distributions of $(8, 20, 3)$ Binary Codes

Code	Distance Distribution								
	A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8
C_1	1	0	0	5.8	7.4	3.4	1.4	0.8	0.2
C_2	1	0	0	6.1	7.5	2.9	1.5	0.9	0.1
C_3	1	0	0	5.6	8	3.2	1.2	0.8	0.2
C_4	1	0	0	6	7.2	3	1.8	1	0
C_5	1	0	0	5.6	7.6	3.2	1.6	0.8	0.2

As these distance distributions are all distinct, by the result of Lemma 1.38, these codes are pairwise nonequivalent. Therefore there are at least five nonequivalent $(8, 20, 3)$ binary codes. In fact, five was the exact number determined computationally in [18] (Baicheva and Kolev first proved this result in [3]). Thus every $(8, 20, 3)$ binary code is equivalent to one of these five aforementioned codes.

Therefore every weight distribution listed in Appendix C is realized in a code (or codes) equivalent to at least one C_i for i in [5]. To determine which C_i, we calculated $C_i + v$ for all i in [5] and v in $\{0, 1\}^8$. This yielded $1,280$ (not necessarily distinct) $(8, 20, 3)$ binary codes, and the weight distribution of each one was calculated. Each of the 111 weight distributions appeared at least once in these codes, and for each of the 111, the C_i that generated it were also recorded in this appendix. Note that, as some weight distributions can be generated by more than one C_i, two $(8, 20, 3)$ binary codes with the same weight distribution need not be equivalent.

4.4 Another Definition of Equivalence

The hypercube coloring problem leads to another definition of code equivalence.

Definition 4.6. Let C and D be (n, M, d) binary codes. C and D are Q_n^j-*equivalent* if $C = \Phi(D)$ for some Φ in $Aut(Q_n^j)$. C is Q_n^j-*unique* if every (n, M, d) binary code is Q_n^j-equivalent to C.

Analogous to equivalence in the code theoretical sense, Q_n^j-equivalence is an equivalence relation on the set of all binary codes.

Since coordinate permutation and vector addition functions are automorphisms of Q_n, and hence Q_n^j, for any choices of positive integers j and n, it is clear that if two codes are equivalent in the code theoretical sense, then they are also Q_n^j-equivalent. In fact, for $j = 1$, these two definitions are equal.

While there are five nonequivalent $(8, 20, 3)$ binary codes, Hougardy determined via an exhaustive search that there are only two that are Q_8^2-nonequivalent [10]. The two codes exhibited by Hougardy are C_1 and C_2 above. Therefore C_3, C_4, and C_5 must each be Q_8^2-equivalent to exactly one of C_1 and C_2. This can indeed be verified. Let σ_3, σ_4, and σ_5 be the permutations in S_8 defined by (2 6)(5 8), (1 3 2)(4 7 8)(5 6), and (2 8)(3 4 7 6), respectively. We then have:

$$\begin{aligned} \sigma_3 \circ \tau_1(C_3 + 255) &= C_2, \\ \sigma_4 \circ \tau_1(C_4 + 73) &= C_1, \\ \sigma_5 \circ \tau_1(C_5 + 67) &= C_1, \end{aligned}$$

where the addition, of course, corresponds to the addition of the binary vector with the appropriate integer representation. Since the functions used are all automorphisms of the square of the cube (as seen in Chapter 3.1), C_2 and C_3 are Q_8^2-equivalent. Moreover, so are C_1, C_4, and C_5.

As stated previously, two $(8, 20, 3)$ binary codes with the same weight distribution need not be equivalent from a code theoretical perspective. In fact, they need not even be Q_8^2-equivalent. Consider, for example, weight distribution 14 in Appendix C. It can be realized in codes equivalent to both C_3 and C_5. C_3 is Q_8^2-equivalent to C_2 and C_5 is Q_8^2-equivalent to C_1. Therefore, these codes are not Q_8^2-equivalent.

Knowing the structure of equivalent codes invites further computational investigation into resolving whether or not a 13-coloring of Q_8^2 exists. We have looked at this only briefly. One can produce a large number of $(8, 20, 3)$ codes chosen randomly by first choosing at randomly one of the C_i, for i in [5], then choosing randomly a v in [255], and then choosing randomly a permutation from S_8. The set of these codes are then put into a MIP similar to the IS method of the column generation algorithm described in

Chapter 2.4.3 to see if they can produce a 13-coloring. A column generation approach could be used to increase the size of the set using this random method. Unfortunately, even if no such 13-coloring is found, it will not prove that one doesn't exist. On the other hand, the only algorithm known to find even a 14-coloring of Q_8^2 was the modified Petford-Welsh randomized algorithm. Therefore if a 13-coloring exists, then it may very well be found by a random method.

Chapter 5

Conclusion

We introduced the study of graph theory, and specifically, vertex coloring problems. The decision problem of determining whether or not a general graph is m-colorable for $m \geq 3$ is NP-complete. Our main focus was on determining the chromatic number of the square of the hypercube, Q_k^2. Our computational effort was focused on the smallest value of k for which this is currently unknown, $k = 8$. To determine the chromatic number of Q_8^2, based on current knowledge, requires either exhibiting a 13-coloring, or proving that one can't exist.

While our computational approaches did not determine this value, they did serve to direct our attention to the study of binary codes. The knowledge of binary codes and their structure further strengthened our understanding of the combinatorial nature of this class of graphs. We then directed our computational efforts on determining the structure of optimal binary codes with length 8 and minimum distance at least 3, of which there must be at least 9 representing the color classes of a 13-coloring of Q_8^2.

Future work may focus on the following:

- Investigating further the link between weight and distance distributions of binary codes,
- Applying these results to develop new adaptations of graph coloring algorithms to determine the chromatic number of Q_8^2,
- Generalize our research to determine values of, or narrow bounds on, other unknown values of the chromatic number of Q_k^2.

Bibliography

[1] E. Agrell, A. Vardy, and K. Zeger. Upper bounds for constant-weight codes. *IEEE Trans. Inform. Theory*, 46(7):2373–2395, 2000.

[2] E. Agrell, A. Vardy, and K. Zeger. A table of upper bounds for binary codes. *IEEE Trans. Inform. Theory*, 47(7):3004–3006, 2001.

[3] T. Baicheva and E. Kolev. Binary codes of length eight, minimum distance three and twenty codewords. In *Proc. II Int. Workshop Optimal Codes and Related Topics*, pages 5–8, Sozopol, Bulgaria, 1998.

[4] M. R. Best and A. E. Brouwer. The triply shortened binary Hamming code is optimal. *Discrete Math.*, 17(3):235–245, 1977.

[5] M. R. Best, A. E. Brouwer, F. J. MacWilliams, F. Jessie, A. M. Odlyzko, and N. J. A. Sloane. Bounds for binary codes of length less than 25. *IEEE Trans. Infomation Theory*, IT-24(1):81–93, 1978.

[6] D. Brélaz. New methods to color the vertices of a graph. *Comm. ACM*, 22(4):251–256, 1979.

[7] P. Delsarte. Bounds for unrestricted codes, by linear programming. *Philips Res. Rep.*, 27:272–289, 1972.

[8] P. Delsarte. An algebraic approach to the association schemes of coding theory. *Philips Res. Rep. Suppl.*, (10):vi+97, 1973.

[9] M. R. Garey and D. S. Johnson. *Computers and intractability*. W. H. Freeman and Co., San Francisco, Calif., 1979. A guide to the theory of NP-completeness, A Series of Books in the Mathematical Sciences.

[10] S. Hougardy. Personal email communication, 2008.

[11] T. R. Jensen and B. Toft. *Graph coloring problems*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons Inc., New York, 1995. A Wiley-Interscience Publication.

[12] N. Linial, R. Meshulam, and M. Tarsi. Matroidal bijections between graphs. *J. Combin. Theory Ser. B*, 45(1):31–44, 1988.

[13] F. J. MacWilliams and N. J. A. Sloane. *The theory of error-correcting codes. II.* North-Holland Publishing Co., Amsterdam, 1977. North-Holland Mathematical Library, Vol. 16.

[14] S. McKinley. The hamming codes and delsarte's linear programming bound. Master's thesis, Portland State University, May 2003.

[15] A. Mehrotra and M. A. Trick. A column generation approach for graph coloring. *INFORMS Journal on Computing*, 8:344–354, 1996.

[16] Z. Miller and M. Perkel. A stability theorem for the automorphism groups of powers of the n-cube. *Australas. J. Combin.*, 10:17–28, 1994.

[17] P. R. J. Östergård. On a hypercube coloring problem. *J. Combin. Theory Ser. A*, 108(2):199–204, 2004.

[18] P. R. J. Östergård, T. Baicheva, and E. Kolev. Optimal binary one-error-correcting codes of length 10 have 72 codewords. *IEEE Trans. Inform. Theory*, 45(4):1229–1231, 1999.

[19] A. D. Petford and D. J. A. Welsh. A randomised 3-colouring algorithm. *Discrete Math.*, 74(1-2):253–261, 1989. Graph colouring and variations.

[20] R. M. Roth. *Introduction to coding theory.* Cambridge University Press, New York, second edition, 2006.

[21] P.-J. Wan. Near-optimal conflict-free channel set assignments for an optical cluster-based hypercube network. *J. Comb. Optim.*, 1(2):179–186, 1997.

[22] D. B. West. *Introduction to graph theory.* Prentice Hall, Inc., Upper Saddle River, New Jersey, second edition, 2001.

[23] G. M. Ziegler. Coloring Hamming graphs, optimal binary codes, and the 0/1-Borsuk problem in low dimensions. In *Computational discrete mathematics*, volume 2122 of *Lecture Notes in Comput. Sci.*, pages 159–171. Springer, Berlin, 2001.

Appendices

Appendix A

Current Status of Coloring the Square of the Cube

Table A.1: Current Status of Coloring the Square of the Cube

k	$n(Q_k^2)$	$e(Q_k^2)$	$A(k,3)$	$\omega(Q_k^2)$	$\frac{n(Q_k^2)}{A(k,3)}$	$\chi(Q_k^2)$
1	2	1	1^a	2	2	2
2	4	6	1^a	4	4	4
3	8	24	2^a	4	4	4
4	16	80	2^a	5	4	8^e
5	32	240	4^a	6	8	8^e
6	64	672	8^a	7	8	8^e
7	128	1,792	16^a	8	8	8^e
8	256	4,608	20^a	9	12.8	$13^d - 14^c$
9	512	11,520	40^a	10	12.8	$13^d - 16^g$
10	1,024	28,160	72^a	11	$14.\overline{2}$	$15^d - 16^g$
11	2,048	67,584	144^a	12	$14.\overline{2}$	$15^d - 16^g$
12	4,096	159,744	256^a	13	16	16^e
13	8,192	372,736	512^a	14	16	16^e
14	16,384	860,160	$1,024^a$	15	16	16^e
15	32,768	1,966,080	$2,048^a$	16	16	16^e

Continued on next page

[a] [2, Table I].
[b] Theorem 1.61.
[c] Exhibited 14-colorings.
[d] Proposition 3.16.
[e] Theorem 3.24.
[f] Theorem 3.26.
[g] Theorem 3.21.

Table A.1– continued from previous page

k	$n(Q_k^2)$	$e(Q_k^2)$	$A(k,3)$
16	$65,536$	$4,456,448$	$2,720^a - 3,276^a$
17	$131,072$	$10,027,008$	$5,312^a - 6,552^a$
18	$262,144$	$22,413,312$	$10,496^a - 13,104^a$
19	$524,288$	$49,807,360$	$20,480^a - 26,208^a$
20	$1,048,576$	$110,100,480$	$36,864^a - 43,689^a$
21	$2,097,152$	$242,221,056$	$73,728^a - 87,378^a$
22	$4,194,304$	$530,579,456$	$147,456^a - 173,491^a$
23	$8,388,608$	$1,157,627,904$	$294,912^a - 344,308^a$
24	$16,777,216$	$2,516,582,400$	$524,288^a - 599,185^a$
25	$33,554,432$	$5,452,595,200$	$1,048,576^a - 1,198,370^a$
26	$67,108,864$	$11,777,605,632$	$2,097,152^a - 2,396,740^a$
27	$134,217,728$	$25,367,150,592$	$4,194,304^a - 4,793,480^a$
28	$268,435,456$	$54,492,397,568$	$8,388,608^b$
29	$536,870,912$	$116,769,423,360$	$16,777,216^b$
30	$1,073,741,824$	$249,644,974,080$	$33,554,432^b$
31	$2,147,483,648$	$532,575,944,704$	$67,108,864^b$

k	$\omega(Q_k^2)$	$\frac{n(Q_k^2)}{A(k,3)}$	$\chi(Q_k^2)$
16	17	$20.004884 - 24.094$	$21^d - 28^g$
17	18	$20.004884 - 24.675$	$21^d - 28^f$
18	19	$20.004884 - 24.976$	$21^d - 32^g$
19	20	$20.004884 - 25.6$	$21^d - 32^g$
20	21	$24.00091556 - 28.\overline{4}$	$25^d - 32^g$
21	22	$24.00091556 - 28.\overline{4}$	$25^d - 32^g$
22	23	$24.17591691 - 28.\overline{4}$	$25^d - 32^g$
23	24	$24.36367438 - 28.\overline{4}$	$25^d - 32^g$
24	25	$28.00006008 - 32$	$29^d - 32^g$
25	26	$28.00006008 - 32$	$29^d - 32^g$
26	27	$28.00006008 - 32$	$29^d - 32^g$
27	28	$28.00006008 - 32$	$29^d - 32^g$
28	29	32	32^e
29	30	32	32^e
30	31	32	32^e
31	32	32	32^e

Appendix B

A 14-coloring of the Square of the 8-Cube

Table B.1: A 14-coloring of Q_8^2

V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}	V_{11}	V_{12}	V_{13}	V_{14}
13	4	2	22	12	3	16	5	14	6	15	0	7	1
19	24	17	27	18	8	29	11	23	9	21	41	10	20
34	31	30	48	35	37	39	28	36	32	40	46	25	26
62	42	47	61	57	59	44	50	43	55	54	51	33	38
75	53	56	78	65	60	79	63	49	58	70	71	52	45
80	77	69	88	94	68	83	64	67	74	73	81	76	66
103	82	91	98	106	87	96	86	72	95	90	92	85	93
104	97	99	105	116	89	117	110	84	113	101	100	107	115
125	102	108	119	139	111	130	121	109	124	127	122	126	120
142	123	118	132	149	112	133	134	114	144	138	131	128	135
152	147	137	143	166	141	158	136	129	155	156	159	150	140
171	160	151	145	176	148	169	161	146	174	173	165	172	153
183	175	164	163	191	154	179	205	157	195	187	188	186	170
194	182	178	168	196	162	180	209	167	213	192	200	201	181
197	185	189	190	216	177	204	218	184	228	215	214	211	203
223	199	207	210	237	193	217	235	198	233	227	226	231	208
244	202	220	221	243	206	230	247	219	242	238	239	240	222
250	212	234	229		232	255	252	224		248	249		225
	236	241	251		246			245					
					253			254					

The V_i for i in [14] given are the color classes of the 14-coloring of Q_8^2 determined by the Hougardy.cpp code of Appendix D, with the vectors given by their integer representations.

Appendix C

Weight Distributions of $(8, 20, 3)$ Binary Codes

Table C.1: Weight Distributions of $(8, 20, 3)$ Binary Codes

Distribution	W_0	W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8	Realized in code(s) equivalent to
0	0	0	1	5	5	5	3	1	0	C_2
1	0	0	2	4	6	4	3	1	0	C_2
2	0	0	2	5	6	4	2	1	0	C_1, C_4
3	0	0	2	6	4	4	3	1	0	C_2
4	0	0	2	6	5	3	3	1	0	C_1, C_4
5	0	0	2	7	3	4	3	1	0	C_3
6	0	0	2	7	4	4	2	1	0	C_5
7	0	0	3	3	4	6	4	0	0	C_2
8	0	0	3	3	6	4	3	1	0	C_3
9	0	0	3	4	4	6	3	0	0	C_1
10	0	0	3	4	6	4	3	0	0	C_5
11	0	0	3	5	3	5	4	0	0	C_4
12	0	0	3	5	4	4	3	1	0	C_1, C_4
13	0	0	3	5	5	3	3	1	0	C_2
14	0	0	3	5	6	2	3	1	0	C_3, C_5
15	0	0	3	6	3	4	3	1	0	C_2
16	0	0	3	6	4	4	3	0	0	C_1
17	0	0	3	6	5	2	3	1	0	C_2
18	0	0	3	6	5	3	2	1	0	C_1, C_4
19	0	0	3	7	2	4	3	1	0	C_5
20	0	0	3	7	3	3	3	1	0	C_2
21	0	0	3	7	3	4	2	1	0	C_3
22	0	0	3	7	4	2	3	1	0	C_1, C_4
23	0	0	3	7	4	3	2	1	0	C_2
24	0	0	4	2	4	6	4	0	0	C_3
25	0	0	4	4	2	6	4	0	0	C_1
26	0	0	4	4	3	5	4	0	0	C_2
27	0	0	4	4	4	4	4	0	0	C_3, C_5
28	0	0	4	5	3	4	4	0	0	C_2
29	0	0	4	5	3	5	3	0	0	C_4

Continued on next page

Table C.1– continued from previous page

Distribution	W_0	W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8	Realized in code(s) equivalent to
30	0	0	4	6	2	4	4	0	0	C_1
31	0	0	4	6	3	3	3	1	0	C_4
32	0	0	4	6	4	2	3	1	0	C_2
33	0	0	4	6	4	2	4	0	0	C_3
34	0	0	4	6	4	3	3	0	0	C_2
35	0	1	0	0	10	8	0	0	1	C_2
36	0	1	0	5	5	5	3	1	0	C_3
37	0	1	1	3	7	7	0	0	1	C_2
38	0	1	1	3	8	6	0	0	1	C_1, C_4
39	0	1	1	4	6	5	2	1	0	C_2
40	0	1	1	4	6	7	0	0	1	C_3
41	0	1	1	4	7	4	2	1	0	C_1, C_4
42	0	1	1	5	5	5	2	1	0	C_3
43	0	1	1	5	6	5	1	1	0	C_5
44	0	1	2	2	7	7	0	0	1	C_1, C_4
45	0	1	2	2	8	6	0	0	1	C_2
46	0	1	2	2	9	5	0	0	1	C_3, C_5
47	0	1	2	3	4	7	3	0	0	C_2
48	0	1	2	3	5	6	3	0	0	C_1, C_4
49	0	1	2	3	6	4	3	1	0	C_2
50	0	1	2	3	6	5	2	1	0	C_1, C_4
51	0	1	2	3	6	7	0	0	1	C_2
52	0	1	2	3	7	4	2	1	0	C_2
53	0	1	2	3	8	3	2	1	0	C_3, C_5
54	0	1	2	3	8	5	0	0	1	C_2
55	0	1	2	4	3	7	3	0	0	C_3
56	0	1	2	4	4	7	2	0	0	C_5
57	0	1	2	4	5	4	3	1	0	C_1, C_4
58	0	1	2	4	5	5	2	1	0	C_2
59	0	1	2	4	5	7	0	0	1	C_5
60	0	1	2	4	6	5	2	0	0	C_1, C_4
61	0	1	2	4	7	3	2	1	0	C_2
62	0	1	2	4	7	4	1	1	0	C_1, C_4
63	0	1	2	4	7	5	0	0	1	C_1, C_4
64	0	1	2	5	4	5	2	1	0	C_5
65	0	1	2	5	5	4	2	1	0	C_2
66	0	1	2	5	5	5	1	1	0	C_3
67	0	1	2	5	6	3	2	1	0	C_1, C_4
68	0	1	2	5	6	4	1	1	0	C_2
69	0	1	3	2	4	6	4	0	0	C_2
70	0	1	3	2	4	7	3	0	0	C_1, C_4
71	0	1	3	2	5	6	3	0	0	C_2
72	0	1	3	2	6	5	3	0	0	C_3, C_5
73	0	1	3	3	3	6	4	0	0	C_4

Continued on next page

Table C.1– continued from previous page

Distribution	W_0	W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8	Realized in code(s) equivalent to
74	0	1	3	3	3	7	3	0	0	C_2
75	0	1	3	3	5	5	3	0	0	C_2
76	0	1	3	3	5	6	2	0	0	C_1, C_4
77	0	1	3	3	6	3	3	1	0	C_5
78	0	1	3	3	6	6	0	0	1	C_4
79	0	1	3	4	2	7	3	0	0	C_5
80	0	1	3	4	3	6	3	0	0	C_2
81	0	1	3	4	3	7	2	0	0	C_3
82	0	1	3	4	4	5	3	0	0	C_1, C_4
83	0	1	3	4	4	6	2	0	0	C_2
84	0	1	3	4	5	4	2	1	0	C_1, C_4
85	0	1	3	4	6	3	2	1	0	C_2
86	0	1	3	4	6	3	3	0	0	C_3
87	0	1	3	4	6	4	2	0	0	C_2
88	0	1	3	5	5	5	0	0	1	C_2
89	0	1	3	5	5	5	0	1	0	C_3
90	0	1	3	5	5	5	1	0	0	C_2
91	1	0	0	0	10	8	0	0	1	C_3
92	1	0	0	4	8	6	0	0	1	C_1
93	1	0	0	4	9	5	0	0	1	C_2
94	1	0	0	4	10	4	0	0	1	C_3, C_5
95	1	0	0	5	5	5	3	1	0	C_2
96	1	0	0	5	7	4	2	1	0	C_1, C_4
97	1	0	0	5	8	3	2	1	0	C_2
98	1	0	0	5	9	2	2	1	0	C_3, C_5
99	1	0	0	5	9	4	0	0	1	C_2
100	1	0	0	6	6	3	3	1	0	C_4
101	1	0	0	6	8	2	2	1	0	C_2
102	1	0	0	6	8	3	1	1	0	C_1, C_4
103	1	0	0	6	8	4	0	0	1	C_1
104	1	0	0	7	5	4	2	1	0	C_5
105	1	0	0	7	6	3	2	1	0	C_2
106	1	0	0	7	6	4	1	1	0	C_3
107	1	0	0	7	7	2	2	1	0	C_1, C_4
108	1	0	0	7	7	3	1	1	0	C_2
109	1	0	0	8	10	0	0	0	1	C_3
110	1	0	0	8	10	0	0	1	0	C_2

Appendix D

Hougardy.cpp

```
1 #include <ilcplex/ilocplex.h>
2 #include <string>
3
4 ILOSTLBEGIN
5
6 typedef IloArray<IloIntArray> IloIntMatrix ;
7
8 IloInt lastTime = 0;
9 IloInt timePeriod = 5;
10 int maxTotalColored = 0;
11
12
13
14
15 int bitStringDiff(IloIntArray string1, IloIntArray
     string2){
16   int count = 0;
17   for (int i = 0; i < string1.getSize(); i++){
18     if (string1[i] != string2[i]){
19       count++;
20     }
21   }
22   return count;
23 };
24
25
26 void main()
27 {
28   IloEnv env;
29
30   try {
```

```
31      IloInt nBits = 8;
32      IloInt nVertices = int(pow(2,nBits));
33      IloInt nColorClasses = int(pow(2,int(ceilf(log(
           float(nBits+1))/log(2.0)))));
34      cout << "number of bits = " << nBits << endl;
35
36      IloIntMatrix bitStrings(env,nVertices);
37      for (int v = 0; v < nVertices; v++){
38        bitStrings[v] = IloIntArray(env, nBits);
39        int remainder = v;
40        for (int b = nBits - 1; b >= 0; b--){
41          if (remainder >= pow(2,b)){
42            bitStrings[v][nBits - 1 - b] = 1;
43            remainder -= pow(2,b);
44          }
45          else{
46            bitStrings[v][nBits - 1 - b] = 0;
47          }
48        }
49      }
50
51      IloIntMatrix neighbors(env, nVertices);
52      for (int v = 0; v < nVertices; v++){
53        neighbors[v] = IloIntArray(env);
54        IloIntArray bitStringOfv = bitStrings[v];
55        for (int n = 0; n < nVertices; n++){
56          if (n != v){
57            if (bitStringDiff(bitStringOfv,bitStrings[n
                 ]) <= 2){
58              neighbors[v].add(n);
59            }
60          }
61        }
62      }
63
64      double c = 0.006;
65      IloNumArray prob_val(env, nBits+1);
66      prob_val[0] = 1.0;
67      for (int i = 1; i < prob_val.getSize(); i++){
68        prob_val[i] = prob_val[i-1] * c;
```

```
69      }
70
71      IloInt nColors = 14;
72
73
74      IloRandom random(env);
75
76      IloIntArray coloring(env,nVertices);
77      for (int i = 0; i < nVertices; i++){
78        coloring[i] = random.getInt(nColors);
79      }
80
81      IloIntArray num_col(env, nColors);
82      IloInt num_bad;
83      IloInt num_trials = 0;
84      do{
85        num_bad = nVertices;
86        for (int i = 0; i < nVertices; i++){
87          for(int j = 0; j < nColors; j++){
88            num_col[j] = 0;
89          }
90          for(int ni = 0; ni < neighbors[i].getSize();
              ni++){
91            if (neighbors[i][ni] < nVertices){
92              num_col[coloring[neighbors[i][ni]]]++;
93            }
94          }
95          if (num_col[coloring[i]] == 0){
96            num_bad--;
97          }
98          else{
99            double sum = 0.0;
100           for (int c = 0; c < nColors; c++){
101             sum += prob_val[num_col[c]];
102           }
103           double r = random.getFloat() * sum;
104           double temp = 0.0;
105           int index = -1;
106           do{
107             index++;
```

```
108             temp += prob_val[num_col[index]];
109           }
110           while (temp < r);
111           coloring[i] = index;
112         }
113       }
114       num_trials++;
115     }
116     while (num_bad > 0);
117
118     IloIntMatrix colorClasses(env, nColors);
119     for (int c = 0; c < nColors; c++){
120       colorClasses[c] = IloIntArray(env);
121       cout << "color " << c << ": ";
122       for (int v = 0; v < nVertices; v++){
123         if (coloring[v] == c){
124           cout << v << " ";
125           colorClasses[c].add(v);
126         }
127       }
128       cout << endl;
129     }
130
131     bool goodColoring = true;
132     for(int c = 0; c < nColors; c++){
133       for (int v1 = 0; v1 < colorClasses[c].getSize();
              v1++){
134         for (int v2 = v1 + 1; v2 < colorClasses[c].
              getSize(); v2++){
135           if (bitStringDiff(bitStrings[colorClasses[c
                ][v1]], bitStrings[colorClasses[c][v2]])
                <= 2){
136             goodColoring = false;
137           }
138         }
139       }
140     };
141     if (goodColoring){
142       cout << "assignments are a good coloring" <<
              endl;
```

```
143       }
144       else{
145         cout << "assignments are NOT a good coloring" <<
              endl;
146       }
147
148
149
150
151     }
152     catch (IloException& ex) {
153       cout << "Error: " << ex << endl;
154     }
155     env.end();
156 }
```

Appendix E

CPDist.cpp

```
1 #include <ilcplex/ilocplex.h>
2 #include <ilsolver/ilosolverint.h>
3 #include <string>
4 #include <fstream>
5 #include <time.h>
6 #include <stdlib.h>
7 #include <stdio.h>
8
9 ILOSTLBEGIN
10 typedef IloArray<IloIntArray> IloIntMatrix;
11 typedef IloArray<IloIntVarArray> IloIntVarMatrix;
12
13 int bitStringDiff(IloIntArray string1, IloIntArray
     string2){
14   int count = 0;
15   for (int i = 0; i < string1.getSize(); i++){
16     if (string1[i] != string2[i]){
17       count++;
18     }
19   }
20   return count;
21 }
22
23 int main () {
24
25   IloEnv env;
26
27   int nBits = 8;
28   int nVertices = int(pow(2,nBits));
29   int Dist[10000][9];
30   int feasDist[10000][9];
```

```
31  int feas[10000];
32  int solutionCounter = 0;
33  int setSize = 20;
34  try {
35    IloModel CPmodel(env);
36
37    IloIntVarArray x(env,nBits+1,0,14);
38
39    CPmodel.add(IloSum(x) == setSize);
40
41    CPmodel.add(x[0] <= 1);
42    CPmodel.add(x[1] <= 1);
43    CPmodel.add(x[2] <= 4);
44    CPmodel.add(x[3] <= 8);
45    CPmodel.add(x[4] <= 14);
46    CPmodel.add(x[5] <= 8);
47    CPmodel.add(x[6] <= 4);
48    CPmodel.add(x[7] <= 1);
49    CPmodel.add(x[8] <= 1);
50
51    CPmodel.add(x[0] + x[1] + x[2] <= 4);
52    CPmodel.add(x[6] + x[7] + x[8] <= 4);
53
54    CPmodel.add(IloIfThen(env, x[0] == 1, x[1] + x[2]
        == 0));
55    CPmodel.add(IloIfThen(env, x[8] == 1, x[6] + x[7]
        == 0));
56
57    IloSolver solver(CPmodel);
58    solver.startNewSearch();
59    while(solver.next()) {
60      solver.out() << "Dist[" << solutionCounter << "]
          = (";
61      for (int i = 0; i < nBits; i++) {
62        Dist[solutionCounter][i] = solver.getValue(x[i
            ]);
63        solver.out() << solver.getValue(x[i]) << ", ";
64      }
65      Dist[solutionCounter][nBits] = solver.getValue(x
          [nBits]);
```

```
66        solver.out() << solver.getValue(x[nBits]) << ")"
              ;
67        solver.out() << endl;
68        solutionCounter++;
69      }
70      solver.endSearch();
71      cout << endl << endl << endl << endl;
72    }
73    catch (IloException& e) {
74      cerr << "Concert exception caught: " << e << endl;
75    }
76    catch (...) {
77      cerr << "Unknown exception caught" << endl;
78    }
79
80    for (int i = 0; i < solutionCounter; i++) {
81
82      try {
83        clock_t start, finish;
84        long double duration;
85        start = clock ();
86        IloModel MIPmodel(env);
87        IloIntArray xDist(env, nBits+1);
88        for (int b = 0; b < nBits+1; b++) {
89          xDist[b] = Dist[i][b];
90        }
91
92        IloIntMatrix bitStrings(env,nVertices);
93        for (int v = 0; v < nVertices; v++){
94          bitStrings[v] = IloIntArray(env, nBits);
95          int remainder = v;
96          for (int b = nBits - 1; b >= 0; b--){
97            if (remainder >= pow(2,b)){
98              bitStrings[v][nBits - 1 - b] = 1;
99              remainder -= pow(2,b);
100           }
101           else{
102             bitStrings[v][nBits - 1 - b] = 0;
103           }
104         }
```

```
105         }
106
107         IloIntMatrix wt(env, nBits + 1);
108         for (int b = 0; b < nBits+1; b++){
109           wt[b] = IloIntArray(env, nVertices);
110           for (int v = 0; v < nVertices; v++){
111             if (bitStringDiff(bitStrings[v],bitStrings
                  [0]) == b) {
112               wt[b][v] = 1;
113             }
114             else wt[b][v] = 0;
115           }
116         }
117
118         IloIntVarArray x(env, nVertices, 0, 1);
119         MIPmodel.add(IloSum(x) == setSize);
120         for (int b = 0; b < nBits+1; b++) {
121           MIPmodel.add(IloScalProd(wt[b],x) == xDist[b])
                ;
122         }
123
124         for (int v1 = 0; v1 < nVertices; v1++) {
125           char xName[10];
126           sprintf(xName,"x_%d",v1);
127           x[v1].setName(xName);
128           for (int v2 = v1 + 1; v2 < nVertices; v2++){
129             if (bitStringDiff(bitStrings[v1],bitStrings[
                  v2]) <= 2) {
130               MIPmodel.add(x[v1] + x[v2] <= 1);
131             }
132           }
133         }
134
135         if (xDist[4] >= 1) {
136           MIPmodel.add(x[15] == 1);
137         }
138         if (xDist[4] >= 3) {
139           MIPmodel.add(x[51] == 1);
140         }
141
```

```
142       IloCplex cplex(MIPmodel);
143       cplex.setParam(IloCplex::MIPInterval,10000);
144       cplex.setParam(IloCplex::MIPEmphasis,1);
145       if (cplex.solve()) {
146         finish = clock ();
147         long double persec = CLOCKS_PER_SEC;
148         duration = (finish - start)/persec;
149         cout << "
              ********************************************
              " << endl;
150         cout << "This took " << duration << " seconds.
              " << endl;
151         feas[i] = 1;
152         for (int b = 0; b < nBits+1; b++) {
153           feasDist[i][b] = Dist[i][b];
154         }
155         cout << endl << endl << "Distribution " << i
              << " ---> [";
156         for (int b = 0; b < nBits; b++) {
157           cout << Dist[i][b] << ", ";
158         }
159         cout << Dist[i][nBits] << "]";
160         cout << ":" << endl << endl;
161         for (int v1 = 0; v1 < nVertices; v1++) {
162           if (cplex.getValue(x[v1]) > 0.5) {
163             int diff = bitStringDiff(bitStrings[v1],
                    bitStrings[0]);
164             cout << v1 << " ---- " << bitStrings[v1];
165             cout << "  " << diff;
166             cout << endl;
167           }
168         }
169       }
170       else {
171         finish = clock ();
172         duration = (finish - start)/1000.0;
173         cout << "
              ********************************************
              " << endl;
174         cout << "This took " << duration << " seconds.
```

```
            " << endl;;
175         feas[i] = 0;
176         for (int b = 0; b < nBits+1; b++) {
177           feasDist[i][b] = 0;
178         }
179         cout << "Distribution " << i << " ---> [";
180         for (int b = 0; b < nBits; b++) {
181           cout << Dist[i][b] << ", ";
182         }
183         cout << Dist[i][nBits] << "]";
184         cout<< ":" << " infeasible" << endl << endl <<
                endl;
185       }
186       cout << endl;
187       cout << "
             ************************************************
             " << endl;
188       MIPmodel.end();
189     }
190     catch (IloException& e) {
191       cerr << "Concert exception caught: " << e <<
             endl;
192     }
193     catch (...) {
194       cerr << "Unknown exception caught" << endl;
195     }
196   }
197   cout << endl << endl << endl << endl;
198   cout << "
        ********************************************"
         << endl;
199   cout << "
        ********************************************"
         << endl;
200   cout << "
        ********************************************"
         << endl;
201   cout << "Feasible Distributions:" << endl << endl;
202   int feasSolutionCounter = 0;
203   for (i = 0; i < solutionCounter; i++) {
```

```
204     if (feas[i] > 0.5) {
205       cout << "Feasible Distribution " <<
              feasSolutionCounter << " (Distribution " << i
              << ") ---> [";
206       for (int b = 0; b < nBits; b++) {
207         cout << feasDist[i][b] << ", ";
208       }
209       cout << feasDist[i][nBits] << "]" << endl;
210       feasSolutionCounter++;
211     }
212   }
213   env.end();
214 }
```