A Generalization of the Football Pool Problem

K.P.F. Verstraten

February 28, 2014

Abstract

In this paper, we generalize the Football Pool problem to a new, wider class of covering problems of spaces equipped with the Hamming distance. We refer to problems in this class as the *Inverse Football Pool Problem* and the *Hamming Distance Covering Problem*. We find several relations between these problems and the Football Pool Problem, with some especially interesting correspondences for binary alphabets. We use these relations to generate combinatorial bounds for the minimal covering cardinalities of the new instances. For open instances, we use a genetic algorithm to improve the upper bounds and we use symmetry-reducing techniques and the symmetry-shrunken Sherali-Adams relaxation to improve the lower bounds.

Thesis submitted in partial fulfillment of the requirements for the degree of
Master of Science in Operations Research and Management Science

Tilburg School of Economics and Management
Tilburg University

Contents

1 Acknowledgements

The author wishes to thank several people. Thanks go out to my supervisor Edwin van Dam, who supported my writing a thesis on what is commonly referred to as 'a game for mathematicians', for which I am very grateful. I would also like to thank Juan Vera Lizcano, who is not only my second reader, but also the one who introduced me to the Football Pool Problem in the first place. Thanks also go out to James Ostrowski, who provided me with helpful comments on the Sherali-Adams relaxation from overseas.

Many thanks to my family for their support during my studies. I would like to thank everyone at the EOR department for making the last years a very educational experience, and I would also like to thank everyone who made the last years very valuable in many other ways.

2 Introduction

The Football Pool Problem is a very famous subject in combinatorics, which was first studied in the Finnish sports magazine *Veikkaaja*[1] in the 1940s. In football pools, one can buy tickets to bet on the outcome of a given number n of matches. Usually, the payoff of such a bet depends on the number of correct guesses. In the foolball pool problem, strategies are studied that ensure a certain payoff. If each game has q different outcomes, the minimum number of tickets that has to be bought to ensure that there is always one ticket where all guesses are correct is q^n. For example, in football, each match has $q = 3$ possible outcomes (home win, draw, away win), so the number of tickets one has to buy to ensure he guesses all $n = 9$ matches in a Dutch Eredivisie competition round correctly is $3^9 = 19683$.

This is quite a lot, so one may also be interested in ensuring a lower number of correct guesses. While the number of tickets that need to be bought for n correct guesses is trivial, the number of tickets one has to buy to ensure $n - 1$ correct guesses cannot be expressed by a closed formula. In fact, this is a very hard problem, which led to the discussion in *Veikkaaja.* For relatively small parameters $q = 3$ and $n = 6$, the exact solution to this problem is not known. This makes the football pool problem not only a problem of interest for football pool fanatics, but also for mathematicians and combinatorialists. Moreover, the problem can be seen as the construction of an efficient error correcting code in the field of information theory, which is also of interest for mathematicians. It is also related to the graph coloring and graph packing problems, and even to mathematical games such as sudoku.

In the 40's, *Veikkaaja* contributors and information theorists did not know the correspondence between their research. For example, the perfect ternary Golay code for the case $q = 3$, $n = 13$ and $n - 2$ correct predictions was published in the *Veikkaja* by Juhani Virtakallio in 1947 [1], while information theorist Marcel Golay independently discovered the code in 1949 [7]. In the 50's, the term football pool problem was more often used in mathematical articles. Since, in football games, the number of different outcomes is $q = 3$, this is the problem that was studied the most. However, many variations to this problem have been studied since then. For $q = 2$, the problem is called the binary covering problem, which is discussed, among others, in [4],[14] and [33]. The variation $q > 3$ was, amongst others, discussed in [14] and [32].

Combinations between different alphabet sizes in codes have also been considered. For example, a code consisting of a letter and single digit number has alphabet sizes 26 and 10. Especially the combinations between binary and

[1]While the Veikkaaja is a sports magazine, it is an important source in the history of covering codes. Many upper bounds for covering codes have been published in this magazine that have not been improved until this date. Exactly how the contributors constructed these bounds without modern computer technology remains somewhat of a mystery today.

ternary alphabets have been considered, amongst others in [12], [17] and [24].

Another interesting variation is that of multiple coverings. Suppose that multiple, say m, football pool betters want to work together and implement a strategy that will ensure at least m prizes. That is, there are at least m tickets bought that have at least a certain amount of right guesses. Working together can actually decrease the amount of tickets per person needed, so it can allow for more efficient strategies. This variation has been studied in [3], [10] and [34]. Some more variations are discussed in [11].

The variation that we want to generalize in this paper is that which is called the *inverse football pool problem.* This is the problem of placing a minimum number of bets, such that there is always one *completely wrong.* This problem was actually discussed in the *Veikkaaja* editions 52 / 60, but had gone unnoticed by the mathematical community for a long time. In 2002, Taneli Riihonen rediscovered the problem [29], which was then published by Östergård and Riihonen a year later [27]. In 2011, the problem was studied by David Brink, who calculated better lower bounds for the cases with 7 or more matches, and introduced bounds for the cases with $q > 3$ and the name inverse football pool problem [2].

The problem is named the inverse football pool problem because we require a minimum number of *wrong* guesses, instead of a minimum number of right guesses. However, only the case where the number of incorrect guesses equals n is discussed in these papers. This means that the inverse football pool problem can be generalized to incorporate other minimum numbers of incorrect guesses. Furthermore, since n incorrect guesses is a boundary case, it also means that we require *exactly* n incorrect guesses. This problem can be generalized to other exact numbers of correct or incorrect guesses as well.

The construction of codes for these strategies may not be very useful in actual football pools. However, the problem does relate to the covering problem of interesting symmetric graphs. Furthermore, the constructions may be useful as space-filling designs in discrete spaces.

In Section 3, we give the general problem statement of the football pool problem and the variations we will discuss. Then, in Section 4, we will discuss the football pool problem with a maximum number of correct guesses. We provide combinatorial bounds for the general cases of this problem. In Section 5 we will discuss bounds for the football pool problem with an exact number of correct guesses. To the writer's best knowledge, this problem has not been studied in this context before. In Section 6, we give some notions on symmetry in the football pool problem, and in Section 7, we explain the symmetry-shrunken Sherali-Adams relaxation, a technique that we can use to improve our lower bounds. In Section 8, we use a genetic algorithm to improve our upper bounds, and in Section 9, we explain our implementation of the Sherali-Adams relaxation to our open cases in order to improve our lower bounds. A short summary of this paper and its conclusions can be found in Section 10, along with some recommendations for further research.

3 Problem definition

Consider the set $F_q^n := \{0, 1, \cdots, q-1\}^n$, q, $n \in \mathbb{N}$, an n-dimensional space with q letters in each dimension that is equipped with the Hamming distance: $d^H(w, v) := |\{i : w_i \neq v_i\}|$. We refer to elements in F_q^n as *words*, because they could represent words of length n in an alphabet of size q. In the football pool problem, we search for a minimum cardinality covering subset C of F_q^n. This is defined as follows:

Definition 3.1. In the football pool problem (FPP), let $C \subseteq F_q^n$. For any given parameters q, $n \in \mathbb{N}$ and any radius R, $0 \leq R \leq n$, we call C a *covering* of F_q^n with radius R if $\forall\, w \in F_q^n, \exists\, v \in C : d^H(w, v) \leq R$.

We denote an instance of the football pool problem with parameters q, n and R by $FPP_q(n, R)$.

In this paper, we study variations on FPP, where words cover all other words at Hamming distance at least R and at Hamming distance exactly R, respectively. We refer to these problems as the inverse football pool problem and the Hamming distance covering problem, respectively. In these problems, a covering subset is defined as follows:

Definition 3.2. In the inverse football pool problem ($IFPP$), let $C \subseteq F$ be a subset of F. For any given parameters q, $n \in \mathbb{N}$ and any radius R, $0 \leq R \leq n$, we call C a *covering* of F with radius R if $\forall\, w \in F, \exists\, v \in C : d^H(w, v) \geq R$.

Definition 3.3. In the Hamming distance covering problem (HDC), let $C \subseteq F$ be a subset of F. For any given parameters q, $n \in \mathbb{N}$ and any radius R, $0 \leq R \leq n$, we call C a *covering* of F with radius R if $\forall\, w \in F, \exists\, v \in C : d^H(w, v) = R$.

We denote an instance of the inverse football pool problem and the Hamming distance covering problem with parameters q, n and R by $IFPP_q(n, R)$ and $HDC_q(n, R)$, respectively.

We denote the cardinality of the smallest possible covering subset of F in FPP by $K_q(n, R)$. Many of the values for this problem are tabulated online at `http://www.sztaki.hu/~keri/codes/`. For $IFPP$ and HDC, we denote the cardinality of the smallest possible covering subsets by $T_q(n, R)$ and $E_q(n, R)$, respectively. We will illustrate the construction and verification of smallest possible covering subsets for these problems in the next example:

Example 3.1. Consider the space F_q^n for $q = 2$, $n = 3$, $F = \{0,1\}^3$. If we enumerate the words in this space as vectors, we get the following matrix:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

We can express the Hamming distances between these words, which can be readily checked, by the following distance matrix:

$$D^H = \begin{bmatrix} 0 & 1 & 1 & 2 & 1 & 2 & 2 & 3 \\ 1 & 0 & 2 & 1 & 2 & 1 & 3 & 2 \\ 1 & 2 & 0 & 1 & 2 & 3 & 1 & 2 \\ 2 & 1 & 1 & 0 & 3 & 2 & 2 & 1 \\ 1 & 2 & 2 & 3 & 0 & 1 & 1 & 2 \\ 2 & 1 & 3 & 2 & 1 & 0 & 2 & 1 \\ 2 & 3 & 1 & 2 & 1 & 2 & 0 & 1 \\ 3 & 2 & 2 & 1 & 2 & 1 & 1 & 0 \end{bmatrix}$$

For a given radius R, we can then define different covering matrices $A^{K,R}$, $A^{T,R}$ and $A^{E,R}$ as follows:

$$A^{K,R}_{(i,j)} = \begin{cases} 1 & \text{if } D^H_{(i,j)} \leq R \\ 0 & \text{otherwise} \end{cases}$$

$$A^{T,R}_{(i,j)} = \begin{cases} 1 & \text{if } D^H_{(i,j)} \geq R \\ 0 & \text{otherwise} \end{cases}$$

$$A^{E,R}_{(i,j)} = \begin{cases} 1 & \text{if } D^H_{(i,j)} = R \\ 0 & \text{otherwise} \end{cases}$$

In these matrices, each column represents the set covered by the corresponding word for FPP, $IFPP$ or HDC with radius R. Using these matrices, we can then define the integer linear programming formulation of each of the problems:

$$\min_x e^T x$$

$$\text{s.t. } Ax \geq e$$
$$x_i \in \{0,1\} \forall i \in \{1,2,\cdots,8\}.$$

Where $A = A^{K,R}, A^{T,R}$ or $A^{E,R}$ for FPP, $IFPP$ or HDC, respectively. Here, e denotes the vector of ones of length 8, or length q^n in the general case.

Solving these integer linear programming problems, we attain the following values:

$$\begin{array}{llll} K_2(3,0)=8 & K_2(3,1)=2 & K_2(3,2)=2 & K_2(3,3)=1 \\ T_2(3,0)=1 & T_2(3,1)=2 & T_2(3,2)=2 & T_2(3,3)=8 \\ E_2(3,0)=8 & E_2(3,1)=4 & E_2(3,2)=4 & E_2(3,3)=8 \end{array}$$

Note that here, the values $K_2(3,0), K_2(3,1), K_2(3,2)$ and $K_2(3,3)$ are solutions to instances of the original football pool problem, and the values $T_2(3,3)$ and $E_2(3,3)$ correspond to the solution to Brink's inverse football pool problem for $q = 2, n = 3$ [2].

The ILP formulation of this problem is a special case of the set cover problem (SCP), which is a well-known problem in combinatorics. The corresponding set cover decision problem, which is to establish whether a covering code with a given cardinality $|C|$ exists for an instance, is one of Karp's 21 NP-complete problems [16]. The problem of finding a minimum cardinality covering code for an instance is an NP-hard problem [18].

Because the problems are NP-hard, polynomial time algorithms cannot produce guaranteed optimal results, unless $\mathbf{P} = \mathbf{NP}$. In fact, polynomial time algorithms cannot produce a tighter approximation than $(1 - o(1)) \ln n$ times the optimal value for general SCP, an approximation bound close to the guaranteed bound $0.72 \ln n$, that is reached by a greedy algorithm [6]. This bound is not very tight. For example, a straighforward greedy algorithm for $FPP_3(6,1)$ finds $19 \leq K_3(6,1) \leq 90$, while its actual cardinality is known to be between 71 and 73.

Even though the problems are NP-hard, for instances of this size, an integer linear programming formulation can easily be solved. However, for larger instances, this is not so easy. This is because of the symmetric nature of the problem. Because of this symmetry, a branch-and-bound decomposition of the problem will have many symmetric nodes, which results in similar calculations being made a lot of times before a node can be killed. Because of this symmetry, the value of $K_3(6,1)$ is still unknown, even though non-symmetric ILP problems of the same size (729 nodes) are solved on a daily basis. The complexity of a symmetric problem also greatly increases when we consider larger problem sizes.

Since solving the ILP formulation is too time-consuming to compute the smallest covering set cardinalities for larger instances, combinatorial constructions have proven to be useful in solving the football pool problem and other symmetric problems. In Section 4 and 5, we consider the same instance as in Example 3.1, and show that the given values are indeed correct for this instance using combinatorial arguments. These bounds will be generalized for other instances of $IFPP$ and HDC. We will not consider new or improved bounds for FPP in these sections, but some relations between the different problems will be discussed.

For cases where combinatorial bounds are not sufficient to find an optimal value for an instance, we will need other methods to improve the lower and upper bounds for this value. For upper bounds, local search algorithms can be used to approximate the optimal bound. Most of the tightest bounds known today have been constructed by use of such local search methods, such as tabu search [35], simulated annealing [19] and genetic algorithms [8]. We use a genetic algorithm to improve the upper bounds for our open instances. The construction of these upper bounds for $IFPP$ and HDC is decribed in Section 8.

Lower bounds for FPP are often constructed by combinatorial arguments [15][22]. A less often used method to construct lower bounds is by relaxations of the ILP formulation. However, Ostrowski recently showed that the use of ILP relaxations can be used to provide good lower bounds [28]. We can use a simple LP relaxation, but more sophisticated relaxations can be used to generate tighter bounds, such as the Lagrangian relaxation [5], the Sherali-Adams relaxation [30], the Lovász-Schrijver relaxation [25] and the Lasserre relaxation [20]. The improvement of lower bounds for open instances by relaxations will be discussed in Section 9.

4 General bounds for the Inverse Football Pool Problem

An important general bound for any set covering problem is the so called *sphere-packing bound* or *volume bound*. Since the problem is symmetric, each word covers the same amount of words. Since we know the total amount of words in F_q^n, we can derive a lower bound by dividing it by the number of words covered by each word:

Theorem 4.1. For any three values $q, n, R \in \mathbb{N}$, $T_q(n, R) \geq q^n / \sum_{r=R}^{n}((q-1)^r * \binom{n}{r})$.

Proof. For any word $w \in F_q^n$ and a radius $r, 0 \leq r \leq n$, consider the set $S^H(w, r) = \{v \in F_q^n : d^H(w, v) = r\}$ for $0 \leq r \leq n$. Any $v \in S^H(w, r)$ differs from w in exactly r coordinates. Since w has got n coordinates, we can choose these r coordinates in $\binom{n}{r}$ ways. For each of these combinations, each coordinate can attain $q-1$ values, since we have q different values in each dimension, and the value must be different than that in w. Therefore, for each choice of coordinates, there are $(q-1)^r$ words at distance r from w. Therefore, $|S^H(w,r)| = (q-1)^r * \binom{n}{r}$.

Since a word w covers another word v if $d^H(w,v) \geq R$, we add over all $r \geq R$, so each word covers $\sum_{r=R}^{n}(q-1)^r * \binom{n}{r}$ other words.

We know that $|F_q^n| = q^n$. Since each word in a code C covers $\sum_{r=R}^{n}(q-1)^r * \binom{n}{i}$ words, C cannot be a covering code for F_q^n if $|C| * \sum_{r=R}^{n}(q-1)^r * \binom{n}{r} \leq q^n$.

Therefore, since $T_q(n,R)$ corresponds to the cardinality of a minimum size covering code, we have that $q^n / \sum_{r=R}^{n}((q-1)^r * \binom{n}{r}) \leq T_q(n,R)$. □

The value of the sphere-packing bound is actually equal to the lower bound found by solving the linear relaxation of the ILP stated in Section 3. Note that, since $T_q(n,R)$ has an integer value, we can round this lower bound up to $\lceil q^n / \sum_{r=R}^{n}((q-1)^r * \binom{n}{r}) \rceil$.

For the values in Example 3.1, we thus have the following lower bounds:

$$\begin{array}{ll} T_2(3,0) \geq \lceil 8/8 \rceil = 1 & T_2(3,1) \geq \lceil 8/7 \rceil = 2 \\ T_2(3,2) \geq \lceil 8/4 \rceil = 2 & T_2(3,3) \geq \lceil 8/1 \rceil = 8 \end{array}$$

In fact, we can see that the optimal value found in Example 3.1 is equal to the sphere-packing lower bound for the cases $R = 0, 2, 3$. This gives rise to the following definition:

Definition 4.1. A covering code C is called a *perfect code* if its cardinality is equal to its sphere-packing lower bound. That is, if each word is covered exactly once.

Note that we do not mean the rounded-up sphere-packing bound, but the value $q^n / \sum_{r=R}^{n}((q-1)^r * \binom{n}{r})$. Indeed, this implies that each word is covered exactly once. For some of the cases from Example 3.1, this is not surprising. For the case $R = 0$, the number of covered words per word can be rewritten as follows:

$$\sum_{r=0}^{n}((q-1)^r * \binom{n}{r}) = \sum_{r=0}^{n}(a^{(n-r)} b^r \binom{n}{r}) \text{ with } a = 1 \text{ and } b = q-1.$$

Then, by the binomial theorem, we have that $\sum_{r=0}^{n}(a^{(n-r)} b^r \binom{n}{r}) = (a+b)^n = q^n = |F_q^n|$, so any one word covers F_q^n. Obviously, this is a perfect code with $T_q(n,0) = 1$. Note that this is the same problem as the trivial maximum-distance football pool problem for $R = n$, which leads to $K_q(n,n) = 1$.

The perfect code for the case $q = 2, n = 3, R = 3$ can be generalized as well, to a class of perfect codes for $IFPP_2(n, n)$. For these cases, each word covers only its exact opposite (e.g. $[1, 0, 1]^\top$ covers $[0, 1, 0]^\top$), so a covering code needs to include every word's opposite, which is unique, since $q = 2$. Therefore, $T_2(n, n) = 2^n$ (II). Even though it also holds that $T_2(n, n) = K_2(n, 0)$, this case is not equal to $FPP_2(n, 0)$, since every word covers only itself in that problem. For this reason, when we take $q > 2$, we still have $K_q(n, 0) = q^n$, while $T_q(n, n)$ not trivial [2]. However, for $q = 2$, there is a remarkable correspondence between FPP and $IFPP$:

Theorem 4.2. For any two values $n, R \in \mathbb{N}$, we have that $T_2(n, R) = K_2(n, n-R)$.

Proof. Let $q = 2$ and $n, R \in \mathbb{N}$. For all $w \in F_q^n$, we denote by w' the unique word in $S^H(w, n)$. Since $q = 2$, for any word $v \in F_q^n$, either $v_i = w_i$ or $v_i = w_i'$. Therefore, $d^H(v, w) + d^H(v, w') = n$. This implies that the set $S^H(w, r)$ is equal to the set $S^H(w', n-r)$ for all $0 \leq r \leq n$. It follows that, if a code C is covering for $IFPP_2(n, R)$. Then the code $C' = \{w' : \exists\, w \in C : d^H(w, w') = n\}$ is covering for $FPP_2(n, n - R)$. Since the problems are equivalent, it holds that their minimal cardinality solutions $T_2(n, R)$ and $K_2(n, n-R)$ must be equal. □

The case $IFPP_2(3, 2) = 2$ also has a perfect code. This code is less trivial, and is part of an infinite length class of perfect codes, which is stated in the following theorem:

Theorem 4.3. For any odd $n \in \mathbb{N}$ and a radius $R = \frac{n+1}{2}$, there is a perfect code for $IFPP_2(n, R)$ with cardinality $T_2(n, R) = 2$.

Proof. Let $q = 2$, take an odd $n \in \mathbb{N}$ and $R = \frac{n+1}{2}$. Take two words $w, w' \in F_q^n$, such that $w' \in S^H(w, n)$. According to Theorem 4.1, each of these words covers $\sum_{r=R}^{n}((q-1)^r * \binom{n}{r}) = \sum_{r=R}^{n} \binom{n}{r} = \frac{1}{2} * 2^n$ words. Here, the last equality follows from the formula $\sum_{r=0}^{n} \binom{n}{r} = 2^n$ and the fact that $R = \frac{n+1}{2}$.

Similar to Theorem 4.2, we can show that $S^H(w, r) = S^H(w', n - r)\, \forall\, 0 \leq r \leq n$. This means that if $d^H(v, w) \geq R$, then $d^H(v, w') \leq n - R = n - \frac{n+1}{2} < n - \frac{n}{2} = \frac{n}{2} < \frac{n+1}{2} = R$ and vice-versa. Therefore, a word is always covered by either w or w', and never by both.

This means that $2 * \frac{1}{2} * 2^n = 2^n = |F_q^n|$ unique words are covered by w and w' together. Therefore, they form a perfect covering code with $T_2(n, R) = 2$. □

The code for $IFPP_2(3, 1)$ is not perfect. In fact, nontrivial perfect codes are rather scarce. By Theorem 4.2, however, we have that all perfect codes for $FPP_2(n, R)$ have an equivalent perfect code for $IFPP_2(n, n - R)$. This means that for all n of the form $n = 2^j - 1, j \in \mathbb{N}$, we have $T_2(n, n-1) = 2^{n-j}$ [34]. Furthermore, for $IFPP_2(23, 20)$, we have an equivalent to the perfect binary Golay code [23], with $T_2(23, 20) = 4096$. The ternary Golay code, which is perfect for $FPP_3(11, 2)$ does not have an equivalent in $IFPP$. We can show this more generally:

Theorem 4.4. For any three values $q, n, R \in \mathbb{N}$, with $1 < R < n$, $q > 2$, $IFPP_q(n, R)$ does not have a perfect code.

Proof. Take any two words $w, v \in F_q^n$. Since $q > 2$, we can then construct a word $u \in F_q^n$, such that $w_i \neq u_i \neq v_i$, $i \in \{1, \dots, R\}$. Therefore, $d_H(w, u) \geq R$, $d_H(v, u) \geq R$, so both w and v cover u. This means that any code containing at least two words is not perfect. Since we have that $R > 1$, we know that at least two words are needed to cover F_q^n, so any covering code is not perfect. $\square$

For instances without a perfect code, the sphere-packing lower bound is usually not a very tight bound. For some instances of $IFPP$ the following lower bound is very useful:

Theorem 4.5. For any three values $q, n, R \in \mathbb{N}$, $T_q(n, R) \geq \frac{n}{n-R+1}$.

Proof. Consider $IFPP_q(n, R)$. Now, for a word *not* to be covered by another word, they need to have at least $n - R + 1$ coordinates in common. If a words is not covered by any word in a code C, that means that it has at least $n - R + 1$ coordinates in common with every word in the code. If we can construct a code C, such that for all $w, v \in C$, it holds that $d^H(w, v) = n$, this means an uncovered word must have at least $|C| * (n - R + 1)$ coordinates. We know the number of coordinates in a word to be n, so a code cannot be covering if $|C| * (n - R + 1) \leq n$. This implies that for any covering code C, we must have that $|C| > \frac{n}{n-R+1}$ and thus $T_q(n, R) > \frac{n}{n-R+1}$ or $T_q(n, R) \geq \lfloor \frac{n}{n-R+1} + 1 \rfloor$. $\square$

If $q > \frac{n}{n-R+1}$, we can actually construct a code C such that $d^H(w, v) = n \, \forall \, w, v \in C$, namely:

$$\{(0, 0, \dots, 0), (1, 1, \dots, 1), \cdots, (\lfloor \frac{n}{n-R+1} \rfloor, \lfloor \frac{n}{(n-R+1)} \rfloor, \dots, \lfloor \frac{n}{n-R+1} \rfloor)\}.$$

This is a so-called *repetition code* For these cases, we have a strict bound:

Corollary 4.5.1. For any three values $q, n, R \in \mathbb{N}$, such that $q \geq \lfloor \frac{n}{n-R+1} + 1 \rfloor$, there is a smallest covering code with $T_q(n, R) = \lfloor \frac{n}{n-R+1} + 1 \rfloor$.

On the other hand, if $q < \lfloor \frac{n}{n-R+1} + 1 \rfloor$, there is no smallest covering code with $T_q(n, R) = \lfloor \frac{n}{n-R+1} + 1 \rfloor$. Therefore, $T_q(n, R) > \lfloor \frac{n}{n-R+1} + 1 \rfloor$.

From this corollary, we can see that the $T_q(n,R)$ decreases as q increases. This is true in general for $IFPP_q(n,R)$: Let $q,n,R \in \mathbb{N}$ and let C be a covering code for $F_q^n = \{0,1,\cdots,q-1\}^n$, with $|C| = T_q(n,R)$. Then, for any $w' \in F_{q+1}^n = \{0,1,\cdots,q\}^n$, define w as follows:

$$w_i = \begin{cases} 0 & \text{if } w_i' = q \\ w_i' & \text{otherwise} \end{cases}$$

Since no coordinate attains the value q in w, it is also in F_q^n, hence it is covered by at least one $v \in C$. Since no coordinate in v attains the value q, we have that $R \leq d^H(w,v) \leq d^H(w',v)$, so w' is also covered by v. Therefore, C is a covering code for F_{q+1}^n with cardinality $T_q(n,R)$. It follows that $T_q(n,R) \geq T_{q+1}(n,R)$.

The value $T_q(n,R)$ also decreases when n increases. This can be proven by a special case of *direct multiplication* of covering codes:

Theorem 4.6. For any five values q,n,n',R and $R' \in \mathbb{N}$, it holds that $T_q(n+n',R+R') \leq T_q(n,R) * T_q(n',R')$.

Proof. Let C and C' be a covering code for $F_q^n = \{0,1,\cdots,q-1\}^n$ and $F_q^{n'} = \{0,1,\cdots,q-1\}^{n'}$, respectively. Define the set $C'' := \{w'' : \exists\, w \in C, w' \in C' : w_i'' = w_i \,\forall\, i \leq n, w_i'' = w_i' \,\forall\, n < i \leq n+n'\}$. Note that $|C''| = |C| * |C'|$.

Now, for any word $v \in F_q^{n+n'} = \{0,1,\cdots,q-1\}^{n+n'}$, there is a word $w \in C$ such that $|\{i \leq n : w_i \neq v_i\}| \geq R$ and a word $x' \in C'$ such that $|\{n < i \leq n' : w_i' \neq v_i\}| \geq R$. Therefore, there is a word $w'' = [w,w']^\top \in C''$, for which $d^H(w'',v) = |\{i \leq n : w_i \neq v_i\}| + |\{n < i \leq n' : w_i' \neq v_i\}| \geq R + R'$. Hence, C'' is a radius $R+R'$ covering code for $F_q^{n+n'}$. $\square$

The case $T_q(n+1,R) \leq T_q(n,R) * T_q(1,0) = T_q(n,R)$ is a specific case of this direct multiplication. However, when n and R increase simultaneously, $T_q(n,R)$ increases, which we will show in the following theorem:

Theorem 4.7. For any three values $q,n,R \in \mathbb{N}$, we have that $T_q(n+1,R+1) \geq T_q(n,R)$.

Proof. Let $q,n,R \in \mathbb{N}$ and let C' be a covering code for $F_q^{n+1} = \{0,1,\cdots,q-1\}^{n+1}$ with radius $R+1$. Now, we define the set $C \subseteq F_q^n = \{0,1,\cdots,q-1\}^n$ as follows: $C = \{w : \exists\, w' \in C' : w_i' = w_i \,\forall\, i \leq n\}$. Note that $|C| \leq |C'|$.

Now, for any $v \in F_q^n$ and $w \in C$ we have that $d^H(w,v) = |\{i : w_i \neq v_i\}| = |\{i < n+1 : w_i' \neq v_i'\}| \geq |\{i : w_i' \neq v_i'\}| - 1 \geq (R+1) - 1 = R$. Therefore, C is covering for F_q^n with radius R. It follows that, $T_q(n,R) \leq T_q(n+1,R+1)$. $\square$

Even with these bounds, there are still a lot of open values for this problem. We will focus on the (traditional) cases $q = 3, n \leq 13$. The values for $T_3(n, n-1)$ are open for $n \geq 6$, the values for $T_3(n, n-2)$ are open for $n \geq 9$ and the values for $T_3(n, n-3)$ are still open for $n \geq 12$. These problems need to be optimized for relatively large sets F_q^n, so we need a method to deal with the symmetry of this problem.

5 General bounds for the Hamming Distance Covering Problem

For HDC, the border cases $HDC_q(n, 0)$ and $HDC_q(n, n)$ are equal to $FPP_q(n, 0)$ and $IFPP_q(n, n)$, respectively. We have that $K_q(n, 0) = E_q(n, 0)$ and $T_q(n, n) = E_q(n, n)$. Furthermore, instances of HDC_2 have an interesting property:

Theorem 5.1. For any $n, R \in \mathbb{N}$ with $0 \leq R \leq n$, we have that any code is a covering for $HDC_2(n, R)$ if and only if it is covering for $HDC_2(n, n - R)$.

Proof. From Theorem 4.2, we know that for any word $w \in F_2^n$, there is exactly one word $w' \in S_H(w, n)$ and that for any word $v \in F_2^n$, we have that $d_H(v, w) + d_H(v, w') = n$. Therefore, if $d_H(v, w) = R$, then $d_H(v, w') = n - R$. Suppose a code is covering for $HDC_2(n, R)$. Then for all $w \in F_2^n$, there is a word v in the code such that $d_H(v, w) = R$. Then we also have for any word $w' \in F_2^n$, there is a word v in the code such that $d_H(v, w') = n - R$, hence the code is covering for $HDC_2(n, n - R)$. The proof for the reverse is similar. $\square$

We therefore have $E_q(n, R) = E_q(n, n - R)$, so it holds that $K_2(n, 0) = E_2(n, 0) = E_2(n, n) = T_2(n, n)$, a special case of Theorem 4.2. However, for all values $0 < R < n$, $HDC_2(n, R)$ is not equivalent to $FPP_q(n, R)$ or $IFPP_q(n, R)$, even for $q = 2$.

Note that in general the set covered by a word in $HDC_q(n, R)$ is included in the set covered by that same word in $FPP_q(n, R)$ and $IFPP_q(n, R)$, so we have $E_q(n, R) \geq K_q(n, R)$ and $E_q(n, R) \geq T_q(n, R)$ for all $q, n, R \in \mathbb{N}$.

Similar to $IFPP$, we have a sphere-packing bound for HDC. Since each word covers $(q-1)^R * \binom{n}{R}$ other words, this sphere packing bound equals $q^n/((q-1)^R * \binom{n}{R})$. For this problem, we also have an infinite length class of instances where we can attain this bound, and thus have perfect codes. To show this, we first introduce some notions from graph theory.

Any covering matrix A corresponds to an undirected graph (V, E), with V the set of q^n vertices corresponding to words in F_q^n and E the set of edges, such that for all $v, w \in V, (v, w) \in E$ if $A_{vw} = 1$. A vertex v then covers a vertex w if

$(v, w) \in E$. The graph of the instance $HDC_2(n, 1)$, for example, is a hypercube of dimension n. Note that, if a vertex v covers itself, we have that $(v, v) \in E$. We introduce the notions of connectivity and bipartitivity for a graph:

Definition 5.1. In a graph (V, E), a pair of vertices v, w are called *connected*, if (V, E) contains a path of edges from v to w. A graph is said to be connected if all pairs $v, w \in V$ are connected.

Definition 5.2. A graph (V, E) is called *bipartite* if we can divide V into two disjoint subsets U, W, such that every edge in $e \in E$ connects a vertex $u \in U$ to a vertex $w \in W$.

If a graph is not connected, this means we do not have one covering problem, but multiple separate covering problems. For many instances of HDC, this actually holds:

Lemma 5.1. The graph (V, E), corresponding to an instance $HDC_2(n, R)$ with $R > 0$ even, is not connected.

Proof. Consider a word $w \in F_2^n$, we denote the number of zeros in w by $z(w)$. Since $q = 2$, we can generate a word $v \in S^H(w, R)$ by changing r_1 ones in w to zeros and r_2 zeros in w to ones, such that $r_1 + r_2 = R$. This means that $z(v) = z(w) + r_1 - r_2$. Since R is even, either r_1 and r_2 are both odd or both even. So if $z(w)$ is odd, this means that $z(v)$ is odd. If $z(w)$ is even, $z(v)$ is even, for all $v \in S^H(w, R)$. This means that in the corresponding graph (V, E), there exists no edge $(w, v) \in E$ if $z(w)$ is odd and $z(v)$ is even, for all $w, v \in V$. Therefore, vertices $w \in V$ with $z(w)$ odd are not connected to vertices $v \in V$ with $z(v)$ even. $\square$

We can, however, see that all vertices v, w with $z(v), z(w)$ odd are connected, and all vertices v, w with $z(v), z(w)$ even are connected by choosing different r_1, r_2. This means that all graphs corresponding to an instance $HDC_2(n, R)$ with R even consist of two connected subgraphs. Note that in both subgraphs, each vertex covers vertices at Hamming distance R. Furthermore, the number of odd vertices and the number of even vertices are equal by properties of the binomial coefficients. This means that the two connected subgraphs are isomorphic.

For R odd, the graph corresponding to $HDC_2(n, R)$ has the property of bipartitivity:

Lemma 5.2. The graph (V, E), corresponding to an instance $HDC_2(n, R)$ with R odd, is bipartite.

Proof. We again denote the number of zeros in a word $w \in F_2^n$ by $z(w)$. Since R is odd, two words $w, v \in F_2^n$ only cover each other if $z(w)$ is even and $z(v)$ is odd or vice-versa. For the graph (V, E) corresponding to $HDC_2(n, R)$, we can therefore define two sets $U, W \subset V$, such that $z(u)$ is odd for all $u \in U$, $z(w)$ is even for all $w \in W$. These sets are disjoint and we have that $U \cup W = V$. Then, by Definition 5.2, the graph (V, E) is bipartite. □

We define a *duo* of words, w^0 and w^1, such that there is a word $w \in F_2^{n-1}$ for which $w^0 = [0\, w]^T$ and $w^1 = [1\, w]^T$. Note that in a duo w^0, w^1, we always have that $z(w^0)$ is odd and $z(w^1)$ is even, or vice-versa. We can therefore see them as corresponding words in the two isomorphic subgraphs of an instance with R even. We can therefore find a minimum cardinality code for one of the connected subgraphs, and generate the optimal code for the other graph by selecting the duo words of those in the subcode.

The construction of these duo codes leads to an interesting equivalence. We therefore first define a more general case of HDC:

Definition 5.3. We define $\mathfrak{R}$ to be the collection of all subsets of $\{0, \ldots, n\}$. We then denote by $HDC_q(n, S), S \in \mathfrak{R}$ the covering problem, such that any code $C \subseteq F$ is a *covering* of F if $\forall\, w \in F, \exists\, v \in C, r \in S : d_H(w, v) = r$.

Note that any instance of FPP or $IFPP$ can be written as a special case of HDC in this way. We use this definition in the next theorem:

Theorem 5.2. For $R > 0$, $HDC_2(n, R)$ is equivalent to $HDC_2(n-1, \{R, R-1\})$.

Proof. First we show the case for R even. We then know that the corresponding graph is not connected. Instead, we have two equivalent connected subgraphs.

We prove that the subgraph with all words w with $z(w)$ is equivalent to the graph corresponding to $HDC_2(n-1, \{R, R-1\})$. The case for the graph with $z(w)$ odd is similar.

We can relabel the vertices in a connected subgraph by removing the first coordinate of the word corresponding to each vertex. We then have a vertex for each word in F_2^{n-1}. We prove that two words cover each other if they are at Hamming distance R or $R-1$. Since we have that $\binom{n}{R} = \binom{n-1}{R} + \binom{n-1}{R-1}$, it follows that all words at other Hamming distance do not cover each other.

Take two words $w, v \in F_q^{n-1}$, such that $d_H(w, v) = R - 1$. Since R is even, this means that either $z(w)$ is even and $z(v)$ is odd or vice-versa. This means that the deleted coordinates for these words were unequal, hence their Hamming distance in the original labeling was R. Therefore, they cover each other.

Now take two words, $w, v \in F_q^{n-1}$, such that $d_H(w, v) = R$. Since R is even, this means that $z(w)$ and $z(v)$ are either both even or both odd. Therefore, their deleted coordinates are equal, and their Hamming distance in the original labeling was also R. Therefore, they cover each other.

This means that the minimal cardinality solution to each of the subgraphs is equal to $E_2(n-1, \{R, R-1\})$. Hence, the minimal cardinality covering of the full graph is equal to $2 * E_2(n-1, \{R, R-1\})$, and the problems are equivalent.

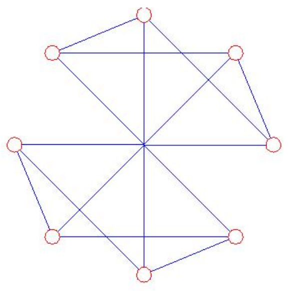
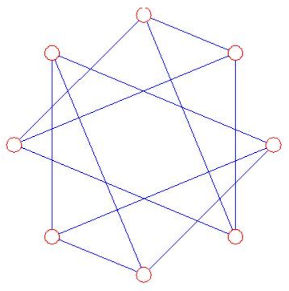
For R odd, we can do something similar. Since the graph corresponding to $HDC_2(n, R)$ is bipartite, we can define U and W to be the sets of all vertices v with $z(v)$ odd or even, respectively. Now consider the subgraph consisting only of the vertices in U. The case for W is similar.

We now have a subgraph $(U, \emptyset)$ that is edgeless. Now, each vertex $u \in U$ has a corresponding duo word in W. Then, for any two vertices $u, v \in U$, we add the edge (u, v) if $d_H(v, w) = R$, where w is the duo word corresponding to u.

Note that the problem of covering this graph is equivalent to the problem of covering all vertices $u \in U$ in the original graph, since all the covering sets are the same.

We relabel the vertices in the subgraph by removing the first coordinate from all words corresponding to the vertices. Note that we now have vertices corresponding to F_2^{n-1}. By similar reasoning as for the subgraph with R even, we can show that all words now cover other words at Hamming distance R or $R-1$. We can create the same problem by creating a subgraph consisting of all vertices $w \in W$. Since the problems of covering the vertices $u \in U$ and the vertices $w \in W$ are equivalent, we therefore also have that $HDC_2(n, R)$ is equivalent to $HDC_2(n-1, \{R, R-1\})$ when R is odd. $\square$

This proof gives an interesting relation between a class of bipartite graphs and graphs with isomorphic disconnected subgraphs that can correspond to each other such that their minimal covering cardinalities are equal.

Left: $HDC_2(3, 1)$ (bipartite). Right: $HDC_2(3, 2)$ (disconnected).

Note that Theorem 5.2 implies $E_2(n, R) \geq 2 * K_2(n-1, R)$. This gives us useful lower bounds even for $R > 1$. For $R = 1$, we can derive the following:

Corollary 5.2.1. For any instance $HDC_2(n, 1)$ with $n > 1$, there is an optimal code with $E_2(n, 1) = 2 * K_2(n-1, 1)$.

Note that this does not means that each of these problems is closed, since $K_2(n, 1)$ is open for most cases with $n \geq 6$. For n of the form $n = 2^j - 1, j \in \mathbb{N}$, however, we have cases of $FPP_2(n, 1)$ with a perfect code. This also means that there are corresponding perfect codes for $HDC_2(n+1, 1)$:

Theorem 5.3. For all $j \in \mathbb{N}$, there is a perfect code for $HDC_2(2^j, 1)$ and $HDC_2(2^j, 2^j - 1)$ with $E_2(n, 1) = E_2(n, n-1) = 2^{n-j}$.

Proof. For these instances, the sphere-covering bound equals $\frac{2^n}{(2-1)^1 * \binom{n}{1}} = 2^n/n$. Since n is of the form $2^j, j \in \mathbb{N}$, this bound is equal to $2^n/2^j = 2^{n-j}$.

We know that $K_2(n-1, 1) = 2^{(n-1)-j}$ [26]. Since $E_2(n, 1) = 2 * K_2(n-1, 1) = 2^{n-j}$, we have a perfect code. □

We can generalize the notion of duos for $q > 2$ and $R > 1$ in the following way. We form a code for $HDC_q(n, R)$ from the union of two subcodes: $C = \{[C^R \, C^{n-R}]^T : C^R \subseteq F_q^R, C^{n-R} \subseteq F_q^{n-R}\}$. Choose a subradius $r \in \mathbb{N}, 0 \leq r \leq R$. We then have that C is covering for $HDC_q(n, R)$ if C^{n-R} is covering for $FPP_q(n-R, R-r)$ and C^R is covering for $HDC_q(R, R), HDC_q(R, R-1), \cdots, HDC_q(R, r)$. Note that if we take $q = 2$, $R = 1$, $r = 0$, we get the construction of duos from Theorem 5.3 and if we take $R = 0$, we get the equivalence of $FPP_q(n, 0)$ and $HDC_q(n, 0)$.

We denote the minimal cardinality for a such a covering code C^R by $D_q(R, r)$. For $q = 2$, we have that C^R is a covering code for $HDC_2(R, R)$, which has optimal solution $E_2(R, R) = 2^R = |F_2^R|$. Therefore, $D_q(R, r) = 2^R$, which does not depend on the choice of r. Since we have that $K_q(n, R)$ decreases as R increases, taking $r = 0$ leads to the best upper bound of this kind. For general q, we have that $D_q(R, R) = T_q(R, R)$ and $D_q(R, 0) = q^R$. Finding $D_q(R, r)$ for $q > 2$, $0 < r < R$ is not trivial. For small instances of this problem, $D_3(R, r)$ is tabulated below:

D_3	$r=0$	$r=1$	$r=2$	$r=3$	$r=4$	$r=5$	$r=6$
$R=1$	3	2	–	–	–	–	–
$R=2$	9	4	3	–	–	–	–
$R=3$	27	6	6	5	–	–	–
$R=4$	81	15	9	9	8	–	–
$R=5$	243	$27-35$	$15-17$	15	15	12	–
$R=6$	729	$71-98$	$18-31$	$18-28$	$18-27$	$18-27$	18

Note that direct multiplication of duo codes can be used to generate upper bounds for larger instances. For any instance of HDC, we have that $E_q(n, R) \leq D_q(R, r) * K_q(n-R, R-r)$ for all $r \leq R \leq n$ for which $n-R \geq R-r$. This form of multiplication often leads to a tighter upper bound than direct multiplication, which is, similar to Theorem 4.6, also possible for HDC.

Like in the inverse football pool problem, we focus on variations of the traditional FPP cases, $q = 2, 3$ and $n \leq 13$. After straightforward optimization, we still have open instances for $HDC_2(n, R)$ with $n > 8$ and for $HDC_3(n, R)$ with $n > 5$. For instances $n > 4$, we still have open problems for $D_3(R, r)$. We will improve the bounds for these open instances and the open instances for $IFPP$ in Sections 8 and 9.

6 Symmetry in the football pool problem

We saw in Example 3.1 that we can express an instance of $FPP, IFPP$ or HDC as the integer linear program $\min_{x \in \{0,1\}^{q^n}} \{e^T x | Ax \geq e\}$. We denote the feasible region of such an instance by $\mathcal{F}$. A *permutation* π is a bijection from the set of indices of x, denoted by I^{q^n}, to itself. We denote the set of all possible permutations on I^{q^n} by Π^{q^n}. For each $\pi \in \Pi^{q^n}$, we can create the matrix P^π such that

$$P^\pi_{ij} = \begin{cases} 1 & \text{if } \pi(j) = i \\ 0 & \text{otherwise} \end{cases}$$

We can then define the vector of variables $\pi(x) := P^\pi x$ and $\pi(x_i) := x_{\pi(i)}$. Note that, if we permute the variables in this problem, we do not need to permute the objective function vector, since it is a vector of ones. This means that the objective function value of a solution is always retained when we permute this solution. We use the notion of permutations for the following definitions:

Definition 6.1. We say that a *permutation group* $\Gamma \in \Pi^{q^n}$ *acts on* a set of indices S if

i) the identity permutation ($\pi(x) = x$) is in Γ,
ii) for all permutations $\pi \in \Gamma$, we have that their inverse π^{-1} is also in Γ,
iii) for all $i \in S$, we have that $\pi(i) \in S$ for all $\pi \in \Gamma$.

Note that the permutation group Π^{q^n} acts on I^{q^n}. However, a permutation group Γ can be a strict subset of Π^{q^n}. We consider the symmetric group $\mathcal{G}$:

Definition 6.2. For any instance of $FPP, IFPP$ or HDC, let $\mathcal{G} \subseteq \Pi^{q^n}$ denote the *symmetric group* of the instance, defined by $\mathcal{G} := \{\pi \in \Pi^{q^n} | \pi(x) \in \mathcal{F} \, \forall \, x \in \mathcal{F}\}$. Two feasible solutions x and $\pi(x)$ are called *isomorphic* if $\pi \in \mathcal{G}$.

Note that the identity permutation maps a feasible solution to itself, so it is in $\mathcal{G}$. Furthermore, if a permutation $\pi \in \mathcal{G}$ maps one feasible solution to another feasible solution with the same objective value, the inverse is also true, so $\pi^{-1} \in \mathcal{G}$. Finally, $\mathcal{G}$ is defined for I^{q^n}, so we have that for all $i \in I^{q^n}, \pi(i) \in I^{q^n}$. Therefore, all requirements for a permutation group are met by the group $\mathcal{G}$.

So, for any $x \in \mathcal{F}$, there is another feasible solution for every $\pi \in \mathcal{G}$. This also means that there can be many optimal solutions. We demonstrate this in the following example:

Example 6.1. Consider the instance $IFPP_3(1,1)$. We can write this problem as the following ILP formulation:

$$\begin{aligned} &\min_{x \in \{0,1\}^3} x_1 + x_2 + x_3 \\ &\text{s.t. } x_1 + x_2 \geq 1 \\ &\quad\;\;\, x_1 + x_3 \geq 1 \\ &\quad\;\;\, x_2 + x_3 \geq 1. \end{aligned}$$

where $x_1 = [0]$, $x_2 = [1]$ and $x_3 = [2]$.

It can be easily seen that the feasible solutions to this problem are $[0,1,1]^\top$, $[1,0,1]^\top$, $[1,1,0]^\top$ and $[1,1,1]^\top$ with objective values $2,2,2$ and 3, respectively.

Now, consider the permutations (123), (132), (213), (231), (312) and (321). If we permute the indices of the variables in x according to any of these permutations, each of the feasible solutions will remain feasible with the same objective value. Therefore, all these permutations belong to the symmetric group $\mathcal{G}$ of this instance. Note that $\mathcal{G} = \Pi^{q^n}$ here, but not in general.

The existence of many isomorphic solutions in an ILP can really confound the optimization process, since we do not know if a solution is optimal until we have considered all solutions. Computing the symmetric group $\mathcal{G}$ is an NP-hard problem in general, and often harder than solving the actual instance [28]. A group that is easier to calculate is the *formulation group*. To define this group, we first define permutations on the constraints.

Consider the constraint matrix $A \in \mathbb{R}^{m \times q^n}$. A *permutation* σ is a bijection from the set of row indices of A, denoted by I^m, to itself. Note that for instances of $FPP, IFPP$ or HDC, we have that $m = q^n$, so we have that $\sigma \in \Pi^{q^n}$. For each $\sigma \in \Pi^{q^n}$, we can create the matrix P^σ such that $\sigma(A) := P^\sigma A$. Note that, if we permute the rows of A, we do not need to apply the same permutation to the vector b, since it is a vector of ones e. We can use the notion of constraint permutations to define the formulation group $\tilde{\mathcal{G}}$.

Definition 6.3. For any instance of $FPP, IFPP$ or HDC, let $\tilde{\mathcal{G}} \subseteq \Pi^{q^n}$ denote the *formulation group* of the instance, defined by $\tilde{\mathcal{G}} := \{\pi \in \Pi^{q^n} \,|\, \exists\, \sigma \in \Pi^{q^n}$, such that $P^\sigma A P^\pi = A\}$.

We also define the *constraint formulation group* $\mathcal{C} := \{\sigma \in \Pi^{q^n} \,|\, \exists \pi \in \Pi^{q^n}$, such that $P^\sigma A P^\pi = A\}$.

We can see here that for the identity permutation, π, the identity permutation σ gives $IAI = A$, so the identity permutation π is in $\tilde{\mathcal{G}}$. Furthermore, it follows directly from $P^\sigma A P^\pi = A$ that $P^{\sigma^{-1}} A P^{\pi^{-1}} = A$. Therefore, if $\pi \in \tilde{\mathcal{G}}, \pi^{-1} \in \tilde{\mathcal{G}}$. Thirdly, $\tilde{\mathcal{G}}$ is defined on the full set I^{q^n}, so it is a permutation group. The reasoning for $\mathcal{C}$ being a permutation group is similar.

The formulation group is in general easier to calculate than the symmetric group, and can be generated by software such as Nauty. We can see that the formulation group is a supergroup of the symmetric group in the following way: Let $x \in \mathcal{F}$, and $\pi \in \tilde{\mathcal{G}}$. Then by definition of $\tilde{\mathcal{G}}$, there exists a permutation $\sigma \in \Pi^{q^n}$, such that $P^\sigma A P^\pi = A$. Since $x \in \mathcal{F}$, we have that $Ax \geq e$ and $P^\sigma A P^\pi x \geq e \rightarrow A(P^\pi x) \geq e$, so $\pi(x) \in \mathcal{F}$, which implies $\pi \in \mathcal{G}$.

Given a permutation group, we can define what variables or constraints can be mapped to what other variables or constraints. We therefore define the orbit:

Definition 6.4. The *orbit* of a variable x_i under the permutation group $\Gamma \subseteq \Pi^{q^n}$ acting on x is denoted by $\operatorname{orb}(x_i, \Gamma) := \{x_j \,|\, \exists\, \pi \in \Gamma : \pi(x_i) = j\}$.

The *orbit* of a constraint a_i under the permutation group $\Gamma \subseteq \Pi^{q^n}$ acting on A is denoted by $\operatorname{orb}(a_i, \Gamma) := \{a_j \,|\, \exists\, \sigma \in \Gamma : \sigma(a_i) = j\}$.

For variables, we have that for any permutation group Γ, if $x_j \in \operatorname{orb}(x_i, \Gamma)$, then $x_i \in \operatorname{orb}(x_j, \Gamma)$, since the inverse of a permutation $\pi \in \Gamma$ is also in Γ. Similarly this holds for different constraints a_i and a_j. This means that we can partition the set of all variables or constraints according to the different orbits the variables or constraints are in. We denote these partitions by $G(A, \Gamma)$ and $C(A, \Gamma)$.

In particular, we consider the orbital partitions for the permutation groups $\mathcal{G}$ and $\mathcal{C}$. For instances of $FPP, IFPP$, and HDC, this partition is actually pretty straightforward:

Theorem 6.1. Given an instance of $FPP, IFPP$ or HDC, we have that $x_j \in \operatorname{orb}(x_i, \mathcal{G}) \,\forall\, i, j \in I^{q^n}$ and $a_j \in \operatorname{orb}(a_i, \mathcal{C}) \,\forall\, i, j \in I^{q^n}$.

Proof. We have that the $|S^H(w, R)|$ is equal for all $w \in F_q^n$. Therefore, a permutation exists such that if $v \in S^H(w, r), \pi(v) \in S^H(\pi(w), r) \,\forall\, v, w \in F_q^n$. Since all distances between the words are preserved, this means that the sets that are covered by each word also stay the same. This means that there is a $\sigma \in \Pi^{q^n}$ such that $P^\sigma A P^\pi = A$, and we can construct a permutation $\pi \in \mathcal{G}$ such that w is mapped to any word in F_q^n. $\square$

This theorem implies that $|G(A, \mathcal{G})| = |C(A, \mathcal{C})| = 1$. We will use this proof to formulate the symmetry-shrunken LP relaxation of our problem in Section 7.

7 The symmetry-shrunken Sherali-Adams relaxation

In order to create lower bounds for our open instances, we can use relaxations to the ILP formulation in Section 3. A simple LP relaxation gives us the value of the sphere-packing bound. You can easily check that the non integer variable vector $[q^n / \sum_{r=R}^{n}((q-1)^r * \binom{n}{r}), \ldots, q^n / \sum_{r=R}^{n}((q-1)^r * \binom{n}{r})]^T$ is covering for the LP relaxation. Therefore, this relaxation does not improve the bounds we already know.

We can, however, relax the ILP formulation in ways that give tighter bounds. Some of these relaxations, such as the Lovász-Schrijver (LS) relaxation and the Lasserre relaxation, require semi-definite programming, while the Sherali-Adams (SA) relaxation does not [21]. The LS, Lasserre and SA relaxation generate a lot of extra variables, so semi-definite programming can still require a lot of computation time, while a linear program usually still finds the optimal value very fast. We therefore use the SA relaxation to provide lower bounds in this paper.

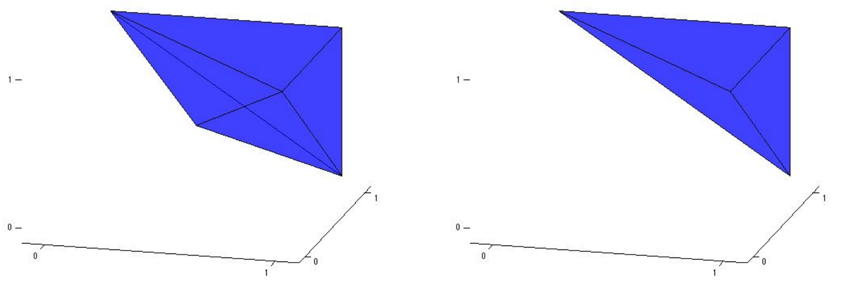
We denote the polytope that is the convex hull of the points in $\mathcal{F}$, the feasible region, by $P := \text{conv}(\mathcal{F})$. Since this is a bounded region, we know from the maximum principle that the optimal solution is one of the vertices of the convex hull, and thus is an integer solution, since $\mathcal{F} \subset \{0,1\}^{q^n}$. Since P is also convex, we can find its optimum by a linear program, which can be solved efficiently.

Unfortunately, finding the linear programming formulation of P can be very time consuming. If we consider the linear programming relaxation of our problem, we get the following polytope:

$$P^0 := \{x \in [0,1]^{q^n} | Ax \geq e\}.$$

This polyhedron P^0 is generally not equal to P. We will show this in the next example:

Example 7.1. We consider again the instance $IFPP_3(1,1)$. We know that $P = \text{conv}([0,1,1]^\top, [1,0,1]^\top, [1,1,0]^T, [1,1,1]^\top)$. However, this polyhedron is not equal to P^0, the polyhedron generated by the linear relaxation of $IFPP_3(1,1)$.



Left: P_0. Right: P.

In order to remove P^0/P from the polytope, we can use Gomory cuts [9], which replace a constraint $\sum_{i=1}^{q^n} a_i x_i \geq b$ with all a_i integer, that is valid for P^0, by the constraint $\sum_{i=1}^{q^n} a_i x_i \geq \lceil b \rceil$ to form a new polyhedron P^1. In Example 7.1, we can replace the constraint $x_1 + x_2 + x_3 \geq 1.5$ with $x_1 + x_2 + x_3 \geq 2$ to obtain $P^1 = P$ from P^0. The problem is that, when an ILP problem is very symmetric, we may need a large amount of such cuts to find P from P_0. If we perform a Gomory cut $\sum_{i=1}^{q^n} a_i x_i \geq \lceil b \rceil$, there may be many $\pi \in \mathcal{G}$, such that $\sum_{i=1}^{q^n} a_i \pi(x_i) < \lceil b \rceil$, which means the solutions $\pi(x)$ are not eliminated by the cut. Since x and $\pi(x)$ have the same objective function value, this means the cut does not increase our lower bound. We may therefore need to perform such a cut many times. Is it often better to consider the isomorphism groups before cutting, for example by orbitopal fixing [13].

There are other ways to decrease the size of a symmetric problem. Because the $FPP, IFPP$ and HDC instances have a lot of symmetry, we can actually reduce the LP formulation. We call this the symmetry-shrunken LP, which is due to [28]:

Theorem 7.1. Given an LP formulation where $G(A, \mathcal{G})$ is the orbital partition of the variables with respect to the symmetric group of variables, and $C(A, \mathcal{C})$ is the orbital partition of the constraints with respect to the symmetric group acting on the constraints, one can construct an LP formulation containing $|G(A, \mathcal{G})|$ (the number of non-isomorphic variables) many variables and $|C(A, \mathcal{C})|$ (the number of non-isomorphic constraints) many constraints that has the same optimal objective value.

From this theorem and the result from Theorem 6.1, we know the following:

Corollary 7.1.1. For an instance of $FPP, IFPP$ or HDC, we can construct an LP problem with one variable and one constraint, that has the same objective value as the regular LP relaxation of the problem.

We can create the symmetry-shrunken LP from an LP by replacing all x_j terms by terms of a new variable y_i, where $i = \min \arg(\text{orb}(x_j, \mathcal{G}))$, the minimal index of all variables in the orbit of x_j. This results in an LP where all variables x_j that can be permuted to each other while retaining the optimal solution are substituted by the same variable. We can do the same for the constraints in A, and permutation group $\mathcal{C}$. We will show this in the next example:

Example 7.2. Consider again the instance $IFPP_3(1,1)$. The constraint matrix for the LP relaxation of this problem looks as follows:

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

We have seen in Example 6.1 that $\mathcal{G} = \Pi^{q^n}$. We also have that $\mathcal{C} = \Pi^{q^n}$, since $P^\sigma A P^\pi = A$ for all $\sigma = \pi, \pi \in \Pi^{q^n}$ for this instance. If we replace all variables in the same orbit, we get the following LP problem:

$$\min_{y \in [0,1]} 3y_1$$

$$\text{s.t. } 2y_i \geq 1.$$

Note that the optimal solution to this relaxation is 1.5, which is indeed equal to the optimal value of the original LP relaxation. We can map the solution of the symmetry-shrunken LP to that of the original LP relaxation by setting x_j equal to y_i for all $x_j \in \text{orb}(x_i, \mathcal{G})$, which gives us the variable vector $[\frac{1}{2}, \frac{1}{2}, \frac{1}{2}]^\top$.

We can create such a symmetry-shrunken LP for the Sherali-Adams relaxation. This is a so-called lift and project method, that consists of a hierarchy of relaxation polytopes P^t, such that $P^0 \supseteq P^1 \supseteq \cdots \supseteq P^{q^n-1} \supseteq P^{q^n} = P$. For proofs of these relations, see [30]. Of course, unless $\mathbf{P} = \mathbf{NP}$, generating the level q^n relaxation takes exponential time. However, we may be able to generate intermediate relaxation levels to find bounds that are tighter than those found so far. We can construct and solve the LP relaxation corresponding to P^t as follows:

Step 1:
We take as P^0 the homogeneous linear relaxation polytope that is symmetry-shrunken with respect to its constraints: $\{x \in [0,1]^{q^n} \mid -1 + a_1 \geq 0\}$.

```
Step 2:
For every $J \subseteq I^{q^n}$, with $|J| = t$ and $J \preceq \pi(J) \, \forall \pi \in \text{stab}(a_1, \mathcal{G})$
    For every partition $(J_0, J_1)$ of $J$, such that $J_0 \preceq \pi(J_0) \, \forall \pi \in \text{stab}(a_1, \mathcal{G})$
    and $J_1 \preceq \pi(J_1) \, \forall \pi \in \text{stab}(J_0, \mathcal{G}) \cup \text{stab}(a_1, \mathcal{G})$.
        Generate the constraint $(-y_0 + cy_i)(\prod_{i \in J_0} x_i)(\prod_{j \in J_1} 1 - x_j)$. (lift)

Step 3:
For every $S \subseteq I^{q^n}$ which has a term $\prod_{i \in S} x_i$ in one of the constraints, replace
$\prod_{i \in S} x_i$ by $y_{S^L}$, with $S^L$ such that $S^L \preceq \pi(S^L) \, \forall \pi \in \mathcal{G}$, $\exists \pi \in \mathcal{G} : \pi(S^L) = S$.
(project)

Step 4:
Solve the linear relaxation over $P^t$, and then set $x_i$ to $y_1 \, \forall i \in I^{q^n}$.
```

Here the permutation group $\text{stab}(a_1, \mathcal{G})$ denotes the set of permutations in $\mathcal{G}$ that map all variables to a variable with the same coefficient in a_1. The permutation group $\text{stab}(J_1, \mathcal{G})$ is the set of permutations in $\mathcal{G}$ that map each element in J_1 to another element in J_1. The symbol $\preceq$ denotes a *lexicographical ordering*. We have that $S \preceq T$, with the elements of S and T ordered from small to large, if there is a number k, such that the first k elements in T are as least as large as the first k elements in S, and there are no more than k first elements of S that are as least as large as their corresponding elements in T. We then say that S is *lexicographically smaller* than T. E.g. the sets $\{1, 3, 6\}, \{1, 4, 5\}, \{2, 3, 4\}$ are lexicographic in this order: $\{1, 3, 6\} \preceq \{1, 4, 5\} \preceq \{2, 3, 4\}$.

This method is called a lift an project method because we *lift* our constraints to a higher dimensional space by multiplying them with variables. This results in non-linear constraints, which are NP-hard to solve. Therefore, we *project* this space back onto a linear space by replacing the monomials $\prod_{i \in S} x_i$ with y_{S^L}. This means that the linear program we solve is a simulation of a non-linear program. However, we can solve it in polynomial time, since we do not enforce $y_{S^L} = \prod_{i \in S} x_i$.

We do not actually know the symmetric group $\mathcal{G}$. However, we do know the formulation group $\tilde{\mathcal{G}}$, and we that $\mathcal{G} \subseteq \tilde{\mathcal{G}}$. Therefore, $\text{stab}(a_1, \mathcal{G}) \subseteq \text{stab}(a_1, \tilde{\mathcal{G}})$, so the partitions J we find using the formulation group is a subset of those we could find using the symmetric group.

We refer to the relaxation using polytope P^t as the *level t SA relaxation*. The generation of an SA relaxation takes longer than solving it. Since the level t SA relaxation has $\binom{q^n}{t}$ extra variables and $2 * \binom{q^n}{t}$ constraints, generating high level relaxations is usually not computationally tractable. That is why the symmetry-shrunken LP is a useful method for this relaxation. We will show how the relaxation works in the next example:

Example 7.3. Consider again the instance $IFPP_3(1, 1)$. We will construct and solve the level 1 SA relaxation for this problem.

Step 1:
We take the LP formulation Example 7.2 and remove all constraints but the first to get the polytope P_0.

Step 2:
Note that $\mathcal{G} = \Pi^{q^n}$. We have that $\text{stab}(a_1, \mathcal{G}) = \text{stab}(\{2,3\}, \mathcal{G}) = \{(123), (132)\}$. Since $\{2\} \preceq \{3\}$, we have two options for J: $\{1\}$ and $\{2\}$. This then leads to four different partition possibilities:

$$\begin{array}{ll} J_0 = \{1\} & J_1 = \emptyset \\ J_0 = \emptyset & J_1 = \{1\} \\ J_0 = \{2\} & J_1 = \emptyset \\ J_0 = \emptyset & J_1 = \{2\}. \end{array}$$

We then get the following constraints:

$$\begin{gathered} (x_2 + x_3 - 1)x_1 \geq 0 \\ (x_2 + x_3 - 1)(1 - x_1) \geq 0 \\ (x_2 + x_3 - 1)x_2 \geq 0 \\ (x_2 + x_3 - 1)(1 - x_2) \geq 0. \end{gathered}$$

Step 3:
We have that $\{1\} \preceq \{2\} \preceq \{3\}$ and $\{1,2\} \preceq \{1,3\} \preceq \{2,3\}$. We can therefore replace the nonlinear terms in the constraints to get the following linear system:

$$\begin{array}{l} \min_{y \in [0,1]^2} 3y_1 \\ \text{s.t. } -y_1 + 2y_{12} \geq 0 \\ \quad\quad 3y_1 - 2y_{12} \geq 1 \\ \quad\quad y_{12} \geq 0 \\ \quad\quad 2y_1 - y_{12} \geq 1. \end{array}$$

Step 4:
The optimal solutions for the problem in Step 3 is $y_1 = \frac{2}{3}, y_{12} = \frac{1}{3}$. We can project this solution onto the space $[0,1]^3$ by setting $x_1 = x_2 = x_3 = y_1$ to get the solution $[\frac{2}{3}, \frac{2}{3}, \frac{2}{3}]^\top$ with the optimal solution value 2.

Note that the optimal solution to the level 1 relaxation gives a tighter bound for this instance than the regular LP-relaxation. In fact, the relaxation gives us the optimal value for the problem. We can also see that this formulation has got only 2 variables and 4 constraints, where the regular level 1 SA relaxation would have 6 variables and 18 constraints. In Section 9, we will use this method to improve the bounds for some open cases of $IFPP$ and HDC.

8 Improving upper bounds for open cases

Main results: See Appendices A, B and C.

In this section, we do not look at the open cases for $IFPP_2$, since these are equivalent to FPP_2, the binary covering problem. We do improve bounds for the open cases of HDC_2, with $n > 8$. For both $IFPP_3$ and HDC_3, we have open cases for $n > 5$. In order to generate tight upper bounds for these instances, we use a genetic algorithm, which simulates natural evolution by creating genetic representation of codes, determining their *fitness* and generating offspring between two such codes if they are fit. We repeat this procedure for a number of iterations, called *generations* in this context.

The genetic representation of a code C is simply a vector of length q^n, with binary variables corresponding to the lexicographically ordered words in F_q^n, where a value of 1 indicates that the corresponding word is in the code. We call the collection of all codes in the program the *population*, which we denote by $\mathbf{P}$.

Since we are looking for minimum cardinality codes, we can choose the cardinality of a code as its fitness. However, if we allow only covering codes in the population, it is hard to improve on existing solution. Therefore, we slightly modify the definition of fitness in the program:

For each code $C \in \mathbf{P}$, we define $S(C) \subseteq F_q^n$ to be the set of all words covered by C. The fitness, $f(C)$ of each code is then defined as $f(C) := |C| + |F_q^n| - |S(C)|$. A fitness implies that we can use the code C to construct a covering code of cardinality f, since we can add a word to the code for each of the $|F_q^n| - |S(C)|$ uncovered words in order to cover all of F_q^n. This means that a code C corresponds to a covering code with cardinality $f(C)$.

Formally, the algorithm goes as follows:

Step 1:
We initialize the time and set the generation number to 0. We then generate a population $\mathbf{P}$ by a randomized greedy algorithm. Furthermore, for each code $C \in \mathbf{P}$ and each word $w \in F_q^n$, we define the *score* $\Delta_w f(C)$ of each word to be the change in fitness of C if we were to add or remove w from C.

Step 2:
We select the fittest codes from $\mathbf{P}$, with exception of the 'escape probability' ϵ of selecting a code with a higher fitness to avoid getting stuck in a local optimum too fast. On the selected codes, we perform the following operations:

Reproduction
We select two of the fittest codes, *parent codes* C_1, C_2 and take a threshold

ω. Then, for each of the codes, we select the words w with $|\Delta_w f(C_1)| > \omega$ or $|\Delta_w f(C_2)| > \omega$. These words either have a large negative score, which means that either they can cover a lot of uncovered words, or they have a large positive score, which means they are the unique covering word for a lot of words in F_q^n. These words represent strong genes in the parent codes. We construct a new code, the *offspring* C^*, consisting of the strong genes from both parents. We then perform a randomized greedy algorithm to improve this new code to a local optimum. We calculate $f(C^*)$ and $\Delta_w f(C^*)$ and then C^* is added to the population.

Mutation
Since all initial codes in $\mathbf{P}$ and all codes from reproduction are local optima, we perform some mutations in the codes in order to break free from these optima. We take a fit code, and add or remove the word with the lowest score. If this does not directly lead to an improvement, we add or remove another word from the code to make sure we do not reverse the operation in the next mutation. The new code is then added to the population.

Step 3:
We select the codes that move on to the next generation through tournament selection. This works as follows: we randomly pair all codes in the population. For each pair of codes, we remove the code with the highest fitness from the population. In case of a tie, a winner is randomly decided. The best code in the population therefore always stays in the population. However, words with higher fitness still have a possibility of survival is they are matches against another weaker code. This way, we avoid getting stuck in local optima too soon.

Step 4:
We increment the generation number and check if there is any code in the new population that has an improved upper bound compared to the best upper bound found so far. If this value is equal to the best lower bound known, or if the time limit or generation limit has passed, we terminate the algorithm. Otherwise, we go to Step 2.

This model has got a few parameters that can be decided by the user, such as maximum population size, the time limit and maximum number of generations, the number of reproductions and mutations in each generation and the parameters ϵ and ω. The default settings for the program are in the source code, which can be found in Appendix D, but they can be tinkered with to produce better approximations for certain instances.

Using this algorithm, we are able to significantly improve the best known upper bounds. For each instance, we let the program run for a maximum of 3000 seconds on a 2.3 GHz Intel Core i5 MacBook. However, most of the best solutions so far were found in considerably less time. The running times for non-optimal solutions found are tabulated below:

	upper bound	time elapsed (s)
$HDC_2(10,2)$	40	22.77
$HDC_2(11,2)$	64	$146,52$
$HDC_2(11,3)$	28	0.70
$HDC_2(12,2)$	106	2961.94
$HDC_2(12,3)$	44	121.37
$HDC_2(12,4)$	16	32.25
$HDC_2(12,5)$	14	531.40
$HDC_2(12,6)$	12	11.67
$HDC_2(13,3)$	76	607.38
$HDC_2(13,4)$	26	6.27
$HDC_2(13,5)$	20	5.98
$HDC_2(13,6)$	16	5.95
$HDC_3(6,2)$	23	0.91
$HDC_3(6,3)$	12	0.37
$HDC_3(6,5)$	10	217.35
$HDC_3(7,2)$	45	5.37
$HDC_3(7,4)$	13	1.82
$HDC_3(7,5)$	10	$18,10$
$HDC_3(7,6)$	12	$74,18$
$HDC_3(8,2)$	146	$13,22$
$HDC_3(8,4)$	21	$21,96$
$HDC_3(8,5)$	14	$12,49$
$HDC_3(8,6)$	15	$12,32$
$HDC_3(8,7)$	21	$12,31$

Table: Best found upper bounds using the genetic algorithm.

Since the algorithm is random, these results are not guaranteed within the time given in the table. On the other hand, new runs of the same length may improve the best upper bound found. From the table, one can easily see in which instances the upper bound found by the greedy algorithm was improved and in which instances it was not. The time to create the adjacency matrix is not included in the time given in the table, but can be calculated in reasonable time for all instances in the table.

9 Improving lower bounds for open cases

In order to establish better lower bounds, we can use techniques that reduce the high level of symmetry in the problem, which is described in Section 6. We use two different approaches. For small cardinality codes, we enumerate the number of unique codes up to isomorphism, and then check whether one is covering for the instance. If not, we can improve the lower bound. This is particularly useful for the different instances of HDC_2, since we can split the corresponding graphs into subgraphs with smaller lower bounds. For larger codes, enumerating all these codes is not computationally tractable, so we use the Sherali-Adams relaxation described in Section 7.

In order to strengthen the relaxation, we can add some extra valid constraints. We can set integrality constraints for some of the values of y_{S^L} by introducing the variables $Y_{S^L} = y_{S^L} * |orb(S^L, \mathcal{G})|$ for all $S^L : |S| = 2$. Since orbit sizes may be large, setting this equality may cause the problem to become numerically unstable. We therefore use the `CPLEX` solver, which has numerical verification, and set the program to focus on numerical stability.

We can add extra constraints that bound the orbit representatives of size 2 from below by using the binomial relation between the variables. For example, if we have that $y_{\{1\}} = m$, then this means that $\sum_{i=1}^{q^n} x_i \geq m$, which in turn implies $\sum_{i=2}^{q^n} \sum_{j=1}^{i-1} x_i x_j \geq \binom{m}{2}$. Since for each term $x_i x_j$, we have a corresponding y_S, we can add the following constraint to the relaxation:

$$\sum_{S^L \in 2^S} Y_{S^L} \geq \binom{y_1 * |orb(\{1\}, \mathcal{G})}{2}.$$

More generally, for orbit representatives of size k, we have that

$$\sum_{S^L \subseteq V, |S^L| = k} Y_{S^L} \geq \binom{y_1 * |orb(\{1\}, \mathcal{G})}{k}.$$

In order to keep computation of the orbits tractable for large instances, we apply the level 1 Sherali-Adams relaxation to our open problems. This means that we only need the first constraint for our relaxation, since cross terms of more than two variables do not occur in the level 1 relaxation. However, this constraint is nonlinear. We can linearize it by using dummy constraints d_i, such that:

$$y_1 * |orb(\{1\}, \mathcal{G}) = \sum_{i=1}^{ub} d_i,$$

where ub is the best known upper bound for the instance. We order the dummy variables, such that $d_1 \geq d_2 \geq \ldots \geq d_{ub}$. We can then replace the nonlinear constraint by:

$$\sum_{S^L \in 2^S} Y_{S^L} = \sum_{i=1}^{ub} (i-1) * d_i.$$

Since we have that $\sum_{i=1}^{\ell}(i-1) = \binom{\ell}{2}$, the constraints are equivalent. This type of constraints has been shown to be effective in the Sherali-Adams relaxation [28].

For the level 1 relaxation, the different orbit representatives are very easy to calculate. If we take $a_1 = A_1$, then the different orbits under $\text{stab}(a_1, \mathcal{G})$ can be represented by the lexicographically minimal words with weight $z(w) = i$, for all $i \in \{0, \dots, n\}$.

The orbit representatives for all sets $S = \{s_1, s_2\}$ can simply be defined by the distance between s_1 and s_2 in the graph. For example, for the graph corresponding to $HDC_q(n, 1)$, this distance is equal to the Hamming distance $d_H(s_1, s_2)$.

In order to calculate the different possible sets J, we use Nauty. We calculate the orbits of the sets of cross terms using Sage. The code used for MATLAB can be found in Appendix D. The results for this relaxation, the time needed to calculate the level 1 Sherali-Adams bound and the comparison to the sphere-packing bound is shown in the table below:

	LP	SA	time elapsed (s)
$HDC_3(6,1)$	61	62	0.07
$HDC_3(7,1)$	157	158	0.11
$HDC_3(8,1)$	411	412	0.32
$HDC_3(8,2)$	59	60	0.10

Table: Linear and Sherali-Adams relaxation bounds.

We can see that the level 1 Sherali-Adams bound is a slight improvement compared to the linear relaxation bound. However, for HDC_3, we do not have any recursive bounds between different instances, so the level 1 Sherali-Adams bound is the best bound found so far. Furthermore, the bound can be calculated very quickly, even for large instances. This means that we can slightly improve the bounds for instances of sizes that are hard to improve otherwise.

We note that the Sherali-Adams relaxation does not improve the bounds for cases with $R > 2$. Since orbits of sets of size 2 can be defined by their inner distance, a higher radius R implies that there are fewer orbits, which may explain why the relaxation is weaker for these cases. This also means that the Sherali-Adams relaxation is more useful for instances of HDC than $IFPP$, since vertices in graphs corresponding to $IFPP$ with $q > 2$ never have a distance higher than 2.

10 Conclusion and recommendations

We generalize the notion of the *Inverse Football Pool problem* by Brink [2], to a wider class of problems, which we call the Inverse Football Pool Problem ($IFPP$) and the Hamming Distance Covering problem (HDC). There are many relations between the different problems and the original Football Pool Problem (FPP), which we use to determine combinatorial lower and upper bounds for the new instances. We give a few properties of the symmetric aspects of FPP, $IFPP$ and HDC, and use this symmetry to reduce the Sherali-Adams relaxation formulation of the problems. We use this relaxation to improve the lower bounds on hard cases. For the upper bounds of these cases, we use a genetic algorithm.

This thesis contributes to the existing literature in the following ways:

- We generalize the football pool problem to a more general class of covering problems corresponding to graphs equipped with the Hamming distance, which is defined for all sets of covering radii $S \subseteq \{0, \ldots, n\}$, according to Definition 5.3. In particular, we consider the sets $S := \{r \in \{0, \ldots, n\} | r \geq R\} \, \forall \, R \in \{0, \ldots, n\}$ and the sets S with $|S| = 1$, which we label the Inverse Football Pool Problem and the Hamming Distance Covering Problem, respectively. However, other sets S can be studied in future research. From the perspective of discrete optimization, we can check how the minimum covering and maximum packing cardinalities corresponding to these graphs relate to each other. Maybe these cardinalities can be ordered corresponding to the vertex degrees of their graphs, which may help in finding useful recursive lower bounds for HDC_q with $q > 2$. Note that we can generalize the definition of duo codes similarly to the generalization in Definition 5.3, which leads to another class of instances with highly nontrivial minimum covering cardinalities. From the perspective of graph theory, the properties of this class of graphs can be studied, and sufficient conditions to define a graph corresponding to an alphabet equipped with the Hamming distance can be formulated.

- Several relations between FPP, $IFPP$ and HDC are defined, which allows for general lower and upper bounds for a lot of instances. In particular, we find equivalence between the instances $HDC_2(n, R)$ and $HDC_2(n-1, \{R, R-1\})$, which allows us to study smaller subgraphs of some large instances, which makes them easier to solve. The result shows an interesting relation between isomorphic disconnected graphs and bipartite graphs.

- We introduce the use of duo codes, that provide better combinatorial bounds than direct multiplication of codes. This method may be applicable to more general Hamming distance covering problems, and the bounds for relatively small instances are still not closed, which could be improved in the future.

- This thesis provides a basic introduction to the symmetry in a graph equipped with the Hamming distance. Some straightforward orbits are explained, and the symmetry-shrunken LP for these graphs is explained. Since this class of graphs is highly symmetric, some more research could be done on the precise structures of the underlying symmetric groups and formulation groups, and their relations to the group corresponding to other graphs in the class.

- The symmetry-shrunken level 1 Sherali-Adams relaxation is tested for new problem instances. An easy formulation of the level 1 relaxation is given, and the formulation is even smaller than that in [28] since we bound the number of dummy variables. The results show that the bound improves more when there is a large number of orbits of sets of cardinality 2, which corresponds to a small covering radius R. Due to storage memory issues, the writer was unable to generate any larger adjacency matrices, but lower bounds of larger instances will very likely be improved by using the level 1 Sherali-Adams relaxation as well. Of course, the bounds may also be improved by using higher levels of the Sherali Adams relaxation. The Sherali-Adams relaxation is not formulated for lower bounds of duo code coverings, but this could be studied in future research.

The source code for the genetic algorithm and the Sherali-Adams relaxation, as well as the source codes used to create adjacency matrices and to generate greedy solutions, is in Appendix D, and can be used to provide bounds for other instances or, with minor alterations, for other problems.

11 Open source software used

`Nauty`: McKay, B.D. and Piperno, A., 2013. *Practical Graph Isomorphism, II.* Journal of Symbolic Computation. `http://cs.anu.edu.au/~bdm/nauty/`.

`Sage`: Stein, W.A. et al., 2014. *Sage Mathematics Software* (Version 6.0), The Sage Development Team. `http://www.sagemath.org`.

`Matgraph`: Scheinerman, E.R., *Matgraph: A MATLAB Toolbox for Graph Theory.* `http://www.ams.jhu.edu/~ers/matgraph/`

12 References

[1] Barg, A., 1993. *At the dawn of the theory of codes.* The Mathematical Intelligencer 15 (1), 20-6.

[2] Brink, D., 2011. *The Inverse Football Pool Problem.* Journal of Integer Sequences 14.

[3] Clayton, R.F., 1987. *Multiple packings and coverings in algebraic coding theory.* Doctoral dissertation, University of California, Los Angeles.

[4] Cohen, G.D., Lobstein, A.C. and Sloane, N.J.A., 1986. *Further Results on the Covering Radius of Codes.* IEEE Transactions on Information Theory 32 (5), 680-94.

[5] Everett, H., 1963. *Generalized Lagrange multiplier method for solving problems of optimum allocation of resources.* Operations Research 11 (3), 399-417.

[6] Feige, U., 1998. *A threshold of ln n for approximating set cover.* Journal of the ACM 45.4, 643-52.

[7] Golay, M.J.E., 1949. *Notes on digital coding.* Proceedings of the I.R.E. 37, 657.

[8] Gommard, G. and Plagne, A., 2003. $K_5(7,3) \leq 100$. Journal of Combinatorial Theory Series A 104 (2), 365-70.

[9] Gomory, R.E., 1958. *Outline of an algorithm for integer solutions to linear programs.* Bulletin of the American Mathematical Society 64 (5), 275-8.

[10] Hämäläinen, H.O., Honkala, I.S., Kaikkonen, M.K. and Litsyn, S.N., 1993. *Bounds for binary multiple covering codes.* Designs, Codes and Cryptography 3 (3), 251-75.

[11] Hämäläinen, H.O., Honkala, I.S., Litsyn, S.N., Östergård, P.R.J., 1995. *Football pools - A game for mathematicians.* The American Mathematical Monthly, 102 (7), 579-88.

[12] Hämäläinen, H.O. and Rankinen, S., 1991. *Upper bounds for football pool problems and mixed covering codes.* Journal of Combinatorial Theory, Series A, 56 (1) 84-95.

[13] Kaibel, V., Peinhardt, M., Pfetsch, M.E., 2007. *Orbitopal fixing* 74-88.

[14] Kalbfleisch, J.G. and Stanton, R.G., 1969. *A combinatorial problem in matching.* Journal of the London Mathematical Society 1.1, 60-4.

[15] Kamps, H.J.L, Van Lint, J.H., 1967. *The football pool problem for 5 matches.* Journal of Combinatorial Theory 3 (4), 315-25.

[16] Karp, R.M., 1972. *Reducibility Among Combinatorial Problems* Complexity of Computer Computations, 85-103

[17] Kolev, E. and Landgev, I., 1994. *On some mixed covering codes of small length.* Algebraic Coding, 38-50.

[18] Korte, B. and Vygen, J., 2012. *Combinatorial Optimization: Theory and Algorithms* 5th ed.

[19] Van Laarhoven, P.J.M., Aarts, E.H.L, Van Lint, J.H., Wille, L.T., 1989. *New upper bounds for the football pool problem for 6, 7 and 8 matches.* Journal of Combinatorial Theory, Series A 52 (2), 304-12.

[20] Lasserre, J.B., 2001. *An explicit exact SDP relaxation for nonlinear 0-1 programs.* Lecture Notes in Computer Science 2081, 293-303.

[21] Laurent, M., 2003. *A comparison of the Sherali-Adams, Lovász-Schrijver and Lasserre relaxations for 0-1 programming.* Mathematics of Operations Research 28 (3), 470-96.

[22] Linderoth, J., Margot, F., Thain, G., 2009. *Improving bounds on the football pool problem by integer programming and high-throughput computing.* INFORMS Journal on Computing 21 (3), 445-57.

[23] Van Lint, J.H., 1975. *A survey of perfect codes.* Journal of Mathematics 5 (2).

[24] Van Lint, J.H. and Van Wee, G.J., 1991. *Generalized bounds on binary/ternary mixed packing and covering codes.* Journal of Combinatorial Theory, Series A, 57 (1), 130-43.

[25] Lovász, L. and Schrijver, A., 1991. *Cones of matrices and set-functions and 0-1 optimization.* SIAM Journal on Optimization 1, 166-90.

[26] Östergård, P.R.J. and Pottonen, O., 2009. *The perfect binary one-error-correcting codes of length 15: Part I. Classification.* IEEE Transactions on Information Theory 55, 4657-60.

[27] Östergård, P.R.J., and Riihonen, T., 2003. *A Covering Problem for Tori.* Annals of Combinatorics 7, 357-63.

[28] Ostrowski, J., 2012. *Using symmetry to optimize over the Sherali-Adams relaxation.* Combinatorial Optimization 59-70.

[29] Riihonen, T., 2002. *How to Gamble 0 Correct in Football Pools.* Special Project in Communications, Helsinki University of Technology.

[30] Sherali, H.D. and Adams, W.P., 1990. *A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems.* SIAM Journal on Discrete Mathematics 3, 411-30.

[31] Sloane, N.J.A., *The On-Line Encyclopedia of Integer Sequences*, published at http://oeis.org/ .

[32] Stanton, R.G., Horton, J.D. and Kalbfleisch, J.G., 1969. *Covering theorems for vectors with special reference to the case of four and five components.* Journal of the London Mathematical Society 2.1, 493-9.

[33] Taussky, O. and Todd, J., 1948. *Covering theorems for groups.* Annales de la Société Polonaise de Mathématique 21, 303-5.

[34] Van Wee, G.J., Cohen, G.D., and Litsyn, S.N., 1991. *A note on perfect multiple coverings of the Hamming space.* IEEE Transactions on Information Theory 37 (3), 678-82.

[35] Wille, L.T., 1987. *The football pool problem on six matches.* Journal of Combinatorial Theory, Series A 45, 171-7.

13 Appendix A: Tables for IFPP

Tables for $T_2(n, R)$:

	$R = 0$	$R = 1$	$R = 2$	$R = 3$	$R = 4$	$R = 5$	$R = 6$
$n = 1$	**1**^{a}	**2**^{b}	–	–	–	–	–
$n = 2$	**1**^{a}	2	**4**^{b}	–	–	–	–
$n = 3$	**1**^{a}	2	**2**^{c}	**8**^{b}	–	–	–
$n = 4$	**1**^{a}	2	2	4	**16**^{b}	–	–
$n = 5$	**1**^{a}	2	2	**2**^{c}	7	**32**^{b}	–
$n = 6$	**1**^{a}	2	2	2	4	12	**64**^{b}
$n = 7$	**1**^{a}	2	2	2	**2**^{c}	7	16
$n = 8$	**1**^{a}	2	2	2	2	4	12
$n = 9$	**1**^{a}	2	2	2	2	**2**^{c}	7
$n = 10$	**1**^{a}	2	2	2	2	2	4
$n = 11$	**1**^{a}	2	2	2	2	2	**2**^{c}

	$R = 7$	$R = 8$	$R = 9$	$R = 10$	$R = 11$	$R = 12$	$R = 13$
$n = 7$	**128**^{b}	–	–	–	–	–	–
$n = 8$	32	**256**^{b}	–	–	–	–	–
$n = 9$	16	62	**512**^{b}	–	–	–	–
$n = 10$	12	24-30	107-120	**1024**^{b}	–	–	–
$n = 11$	7	15-16	37-44	180-192	**2048**^{b}	–	–
$n = 12$	4	11-12	18-28	62-78	342-380	**4096**^{b}	–
$n = 13$	**2**^{c}	7	12-16	28-42	97-128	598-704	**8192**^{b}

Perfect codes are printed in bold.

a : $T_q(n, 0) = 1$.
b : $T_2(n, n) = 2^n$.
c : Theorem 4.3.

All values without superscript follow directly from Theorem 4.2.

Tables for $T_3(n, R)$:

	$R = 0$	$R = 1$	$R = 2$	$R = 3$	$R = 4$	$R = 5$	$R = 6$
$n = 1$	$\mathbf{1}^a$	2^b	–	–	–	–	–
$n = 2$	$\mathbf{1}^a$	2	3^b	–	–	–	–
$n = 3$	$\mathbf{1}^a$	2	2	5^b	–	–	–
$n = 4$	$\mathbf{1}^a$	2	2	3	8^b	–	–
$n = 5$	$\mathbf{1}^a$	2	2	2	3	12^b	–
$n = 6$	$\mathbf{1}^a$	2	2	2	3	$^*6^e$	18^b
$n = 7$	$\mathbf{1}^a$	2	2	2	2	3	$^*7\text{-}9^e$
$n = 8$	$\mathbf{1}^a$	2	2	2	2	3	3
$n = 9$	$\mathbf{1}^a$	2	2	2	2	2	3
$n = 10$	$\mathbf{1}^a$	2	2	2	2	2	3
$n = 11$	$\mathbf{1}^a$	2	2	2	2	2	2

	$R = 7$	$R = 8$	$R = 9$	$R = 10$	$R = 11$	$R = 12$	$R = 13$
$n = 7$	29^b	–	–	–	–	–	–
$n = 8$	$^d7\text{-}15^e$	44^b	–	–	–	–	–
$n = 9$	$^c4\text{-}6^e$	$^c7\text{-}24^e$	$66\text{-}68^b$	–	–	–	–
$n = 10$	3	$^c4\text{-}9^e$	$^c10\text{-}36^e$	$99\text{-}104^b$	–	–	–
$n = 11$	3	3	$^c5\text{-}15^e$	$^c14\text{-}54^e$	$149\text{-}172^b$	–	–
$n = 12$	3	3	$^c4\text{-}6^e$	$^c6\text{-}24^e$	$^c19\text{-}72^e$	$224\text{-}264^b$	–
$n = 13$	2	3	3	$^c4\text{-}9^e$	$^c8\text{-}36^e$	$^c26\text{-}132^e$	$336\text{-}408^b$

Perfect codes are printed in bold. Nontrivial optimization / relaxation results are marked with an asterisk (*).

a : $T_q(n, 0) = 1$.
b : D. Brink: *The Inverse Football Pool Problem* [2].
c : Theorem 4.1.
d : Theorem 4.7.
e : Theorem 4.6.

All values without superscript follow directly from Corollary 4.5.1.

14 Appendix B: Tables for HDC

Tables for $E_2(n,R)$, published electronically as *A230014* [31]:

	$R=0$	$R=1$	$R=2$	$R=3$	$R=4$	$R=5$	$R=6$
$n=1$	$\mathbf{2}^a$	$\mathbf{2}^a$	–	–	–	–	–
$n=2$	$\mathbf{4}^a$	$\mathbf{2}^b$	$\mathbf{4}^a$	–	–	–	–
$n=3$	$\mathbf{8}^a$	4	4	$\mathbf{8}^a$	–	–	–
$n=4$	$\mathbf{16}^a$	$\mathbf{4}^b$	4	$\mathbf{4}^b$	$\mathbf{16}^a$	–	–
$n=5$	$\mathbf{32}^a$	8	6	6	8	$\mathbf{32}^a$	–
$n=6$	$\mathbf{64}^a$	14	8	6	8	14	$\mathbf{64}^a$
$n=7$	$\mathbf{128}^a$	24	8	8	8	8	24
$n=8$	$\mathbf{256}^a$	$\mathbf{32}^b$	16	8	8	8	16
$n=9$	$\mathbf{512}^a$	${}^d64^e$	24	12	10	10	12
$n=10$	$\mathbf{1024}^a$	${}^d124^e$	34-40*	16	12	10	12
$n=11$	$\mathbf{2048}^a$	${}^d214\text{-}240^e$	${}^e48\text{-}64^g$	${}^e24\text{-}28^*$	*16*	*14*	*14*
$n=12$	$\mathbf{4096}^a$	${}^c342\text{-}384^e$	${}^e74\text{-}106^*$	${}^e30\text{-}44^*$	${}^e14\text{-}16^*$	*8-14*	*8-12*
$n=13$	$\mathbf{8192}^a$	${}^d684\text{-}760^e$	${}^e124\text{-}176^g$	${}^e36\text{-}76^*$	${}^e22\text{-}26^*$	${}^e8\text{-}20^*$	${}^f6\text{-}16^*$

	$R=7$	$R=8$	$R=9$	$R=10$	$R=11$	$R=12$	$R=13$
$n=7$	$\mathbf{128}^a$	–	–	–	–	–	–
$n=8$	$\mathbf{32}^b$	$\mathbf{256}^a$	–	–	–	–	–
$n=9$	24	${}^d64^e$	$\mathbf{512}^a$	–	–	–	–
$n=10$	16	34-40*	124^e	$\mathbf{1024}^a$	–	–	–
$n=11$	*16*	${}^e24\text{-}28^*$	${}^e48\text{-}64^g$	${}^d214\text{-}240^e$	$\mathbf{2048}^a$	–	–
$n=12$	*8-14*	${}^e14\text{-}16^*$	${}^e30\text{-}44^*$	${}^e74\text{-}106^*$	${}^c342\text{-}384^e$	$\mathbf{4096}^a$	–
$n=13$	${}^f6\text{-}16^*$	${}^e8\text{-}20^*$	${}^e22\text{-}26^*$	${}^e36\text{-}76^*$	${}^e124\text{-}176^g$	${}^d684\text{-}760^e$	$\mathbf{8192}^a$

Perfect codes are printed in bold. Nontrivial optimization / relaxation results are marked with an asterisk (*).

a : $E_2(n,0) = E_2(n,n) = 2^n$.
b : Theorem 5.3.
c : $E_q(n,R) \geq K_q(n,R)$.
d : Corollary 5.2.1.
e : Theorem 5.2.
f : Sphere-packing bound.
g : $E_q(n,R) \leq D_q(R,r) * K_q(n-R,R-r)$.

All values without superscript follow from straightforward `CPLEX` optimization.

Tables for $E_3(n, R)$, published electronically as *A238305* [31]:

	$R = 0$	$R = 1$	$R = 2$	$R = 3$	$R = 4$	$R = 5$	$R = 6$
$n = 1$	**3**^{a}	2^b	–	–	–	–	–
$n = 2$	**9**^{a}	3	3^b	–	–	–	–
$n = 3$	**27**^{a}	6	4	5^b	–	–	–
$n = 4$	**81**^{a}	14	6	5	8^b	–	–
$n = 5$	**243**^{a}	27	12	6	7	12^b	–
$n = 6$	**729**^{a}	^{c}71-81^g	^{c}15-23*	*7-12*	*7*	*8-10*	18^b
$n = 7$	**2187**^{a}	*158-219^g	^{f}27-45*	^{c}11-18^g	*6-13*	*5-10*	^{d}7-12*
$n = 8$	**6561**^{a}	*412-558^g	*60-135^e	^{f}15-48^g	^{c}9-21*	^{f}4-14*	^{f}4-15*
$n = 9$	**19683**^{a}	^{f}1094-1458^g	^{f}137-306^g	^{f}30-102^g	^{c}11-45^g	^{c}6-35^g	^{f}4-28^g
$n = 10$	**59049**^{a}	^{f}2953-3867^g	^{f}329-729^g	^{f}62-204^g	^{f}18-90^g	^{c}9-51^g	^{f}5-31^g
$n = 11$	**177147**^{a}	^{f}8053-10935^g	^{f}806-1971^g	^{f}135-486^g	^{f}34-180^g	^{f}12-102^g	^{c}9-72^e
$n = 12$	**531441**^{a}	^{f}22144-28431^g	^{f}2014-4995^g	^{f}302-1215^e	^{f}68-405^g	^{f}21-105^g	^{c}10-93^g
$n = 13$	**1594323**^{a}	^{f}61321-83106^g	^{f}5111-6561^g	^{f}697-2835^g	^{f}140-810^g	^{f}39-315^g	^{f}15-93^g

	$R = 7$	$R = 8$	$R = 9$	$R = 10$	$R = 11$	$R = 12$	$R = 13$
$n = 7$	29^b	–	–	–	–	–	–
$n = 8$	^{f}7-21*	44^b	–	–	–	–	–
$n = 9$	^{f}5-30^e	^{f}9-42^e	66-68^b	–	–	–	–
$n = 10$	^{f}4-35^e	^{f}6-45^e	^{f}12-63^e	99-104^b	–	–	–
$n = 11$	^{f}5-42^e	^{f}5-49^e	^{f}7-70^e	^{f}16-105^e	149-172^b	–	–
$n = 12$	^{f}6-78^e	^{f}5-49^e	^{f}5-70^e	^{f}8-100^e	^{f}22-168^e	224-264^b	–
$n = 13$	^{f}9-126^e	^{f}5-90^e	^{f}5-77^e	^{f}6-105^e	^{f}10-150^e	^{f}30-252^e	336-408^b

Perfect codes are printed in bold. Nontrivial optimization / relaxation results are marked with an asterisk (*).

a : $E_q(n, 0) = q^n$.
b : D. Brink: *The Inverse Football Pool Problem* [2].
c : $E_q(n, R) \geq K_q(n, R)$.
d : $E_q(n, R) \geq T_q(n, R)$.
e : Direct multiplication.
f : Sphere-packing bound.
g : $E_q(n, R) \leq D_q(R, r) * K_q(n - R, R - r)$.

All values without superscript follow from straightforward `CPLEX` optimization.

15 Appendix C: Covering codes for non-trivial instances

Codes are displayed by the numbers of their words in lexicographic ordering of F_q^n.

$HDC_2(9,2), HDC_2(9,7)$: 13, 16, 58, 83, 99, 119, 143, 185, 188, 194, 214, 230, 269, 272, 314, 339, 355, 375, 399, 441, 444, 450, 470, 486.

$HDC_2(9,3), HDC_2(9,6)$: 1, 105, 111, 113, 145, 178, 277, 367, 384, 404, 406, 463.

$HDC_2(9,4), HDC_2(9,5)$: 2, 9, 22, 154, 155, 182, 295, 329, 395, 419.

$HDC_2(10,2), HDC_2(10,8)$: 1, 5, 50, 96, 127, 147, 177, 222, 236, 240, 294, 332, 345, 349, 363, 391, 421, 436, 440, 458, 513, 517, 562, 608, 639, 659, 689, 734, 748, 752, 806, 844, 857, 861, 875, 903, 933, 948, 952, 970.

$HDC_2(10,4), HDC_2(10,6)$: 1, 2, 15, 16, 49, 50, 241, 242, 1009, 1010, 1023, 1024.

$HDC_2(10,5)$: 1, 2, 22, 97, 158, 513, 514, 534, 609, 670.

$HDC_2(11,3), HDC(11,8)$: 2, 55, 61, 164, 255, 268, 301, 303, 390, 436, 519, 525, 1010, 1020, 1058, 1068, 1495, 1501, 1602, 1617, 1753, 1760, 1808, 1900, 1999, 2016, 2022, 2033.

$HDC_2(11,4), HDC_2(11,7)$: 1, 2, 7, 8, 31, 32, 63, 64, 255, 256, 1023, 1024, 1025, 1026, 2047, 2048.

$HDC_2(11,5), HDC_2(11,6)$: 1, 19, 20, 32, 192, 768, 1025, 1043, 1044, 1056, 1216, 1792.

$HDC_2(12,2), HDC_2(12,10)$: 39, 55, 89, 94, 137, 228, 276, 279, 366, 425, 446, 452, 471, 556, 582, 646, 657, 764, 767, 783, 796, 865, 881, 950, 975, 1072, 1074, 1099, 1165, 1172, 1184, 1253, 1266, 1282, 1301, 1392, 1403, 1467, 1477, 1488, 1581, 1597, 1603, 1624, 1683, 1708, 1770, 1802, 1912, 1955, 1960, 2010, 2013, 2087, 2103, 2137, 2142, 2185, 2276, 2324, 2327, 2414, 2473, 2494, 2500, 2519, 2604, 2630, 2694, 2705, 2812, 2815, 2831, 2844, 2913, 2929, 2998, 3023, 3120, 3122, 3147, 3213, 3220, 3232, 3301, 3314, 3330, 3349, 3440, 3451, 3515, 3525, 3536, 3629, 3645, 3651, 3672, 3731, 3756, 3818, 3850, 3960, 4003, 4008, 4058, 4061.

$HDC_2(12,3), HDC_2(12,9)$: 1, 2, 17, 18, 85, 86, 171, 172, 191, 192, 255, 256, 819, 820, 841, 842, 873, 874, 919, 920, 973, 974, 3133, 3134, 3139, 3140, 3175, 3176, 3225, 3226, 3267, 3268, 3855, 3856, 3867, 3868, 3931, 3932, 4005, 4006, 4021, 4022, 4081, 4082.

$HDC_2(12,4), HDC_2(12,8)$: 52, 139, 294, 413, 1636, 1755, 1910, 1997, 2100, 2187, 2342, 2461, 3684, 3803, 3958, 4045.

$HDC_2(12,5), HDC_2(12,7)$: 800, 1249, 1645, 1648, 1664, 1769, 1888, 2848, 3297, 3693, 3696, 3712, 3817, 3936.

$HDC_2(12,6)$: 628, 1229, 1262, 1268, 1270, 1293, 2676, 3277, 3310, 3316, 3318, 3341.

$HDC_2(13,3), HDC_2(13,10)$: 123, 131, 271, 689, 829, 965, 1275, 1351, 1449, 1464, 1513, 1521, 1559, 1585, 1610, 1623, 1755, 1991, 2096, 2150, 2244, 2280, 2322, 2702, 2736, 2942, 3036, 3208, 3230, 3294, 3418, 3511, 3657, 3858, 3868, 3876, 3940, 4070, 4219, 4227, 4367, 4785, 4925, 5061, 5371, 5447, 5545, 5560, 5609, 5617, 5655, 5681, 5706, 5719, 5851, 6087, 6192, 6246, 6340, 6376, 6418, 6798, 6832, 7038, 7132, 7304, 7326, 7390, 7514, 7607, 7753, 7954, 7964, 7972, 8036, 8166.

$HDC_2(13,4), HDC_2(13,9)$: 6, 14, 22, 24, 44, 60, 4035, 4057, 4063, 4065, 4071, 4081, 4093, 4102, 4110, 4118, 4120, 4140, 4156, 8131, 8153, 8159, 8161, 8167, 8177, 8189.

$HDC_2(13,5), HDC_2(13,8)$: 1, 2, 31, 32, 33, 34, 993, 994, 4065, 4066, 4095, 4096, 4097, 4098, 4159, 4160, 8129, 8130, 8191, 8192.

$HDC_2(13,6), HDC_2(13,7)$: 1, 2, 63, 64, 963, 964, 1501, 1502, 2799, 2800, 3429, 3430, 5773, 5774, 5927, 5928.

$HDC_3(6,2)$: 67, 83, 134, 156, 165, 216, 236, 250, 254, 265, 280, 304, 307, 336, 375, 455, 559, 584, 599, 636, 666, 681, 716.

$HDC_3(6,3)$: 16, 114, 216, 421, 459, 479, 485, 486, 490, 499, 571, 600.

$HDC_3(6,4)$: 1, 41, 81, 401, 466, 588, 693.

$HDC_3(6,5)$: 20, 23, 329, 383, 384, 387, 658, 663, 685, 693.

$HDC_3(7,2)$: 61, 116, 119, 171, 186, 282, 294, 346, 467, 482, 497, 550, 605, 633, 685, 784, 796, 851, 863, 906, 1002, 1054, 1078, 1190, 1202, 1241, 1269, 1321, 1377, 1429, 1457, 1531, 1583, 1586, 1626, 1641, 1746, 1798, 1922, 1934, 1937, 1949, 2085, 2112, 2137.

$HDC_3(7,4)$: 1, 28, 310, 365, 430, 729, 744, 899, 953, 1132, 1473, 1511 , 1812.

$HDC_3(7,5)$: 114, 137, 258, 313, 551, 668, 700, 751, 782, 804.

$HDC_3(7,6)$: 191, 196, 393, 508, 512, 922, 1117, 1234, 1656, 1848, 1853, 1968.

$HDC_3(8,2)$: 1, 42, 77, 91 ,92 ,122, 167, 243, 347, 358, 429, 443, 467, 537, 621, 627, 661, 678, 686, 785, 876, 881, 936, 943, 946, 987, 990, 1012, 1087, 1137, 1262, 1400, 1429, 1444, 1488, 1490, 1606, 1705, 1718, 1732, 1789, 1803, 1860, 1863, 1919, 1990, 2014, 2063, 2082, 2087, 2131, 2203, 2292, 2380, 2410, 2412, 2415, 2426, 2456, 2521, 2571, 2629, 2690, 2726, 2796, 2800, 2834, 2843, 2943, 2984, 2992, 3027, 3030, 3083, 3153, 3199, 3233, 3244, 3272, 3282, 3402, 3472, 3493, 3567, 3656, 3666, 3681, 3755, 3770, 3787, 3853, 4062, 4092, 4095, 4130, 4162, 4189, 4197, 4214, 4297, 4347, 4404, 4414, 4477, 4673, 4674, 4687, 4727, 4796, 4827, 4843, 4864, 4877, 4959, 5014, 5016, 5080, 5206, 5210, 5309, 5328, 5419, 5441, 5492, 5518, 5598, 5631, 5676, 5751, 5785, 5819, 5885, 5922, 5933, 5974, 5981, 6031, 6063, 6128, 6202, 6288, 6330, 6333, 6368, 6427, 6503.

$HDC_3(8,4)$: 1, 2, 81, 87, 96, 291, 597, 771, 1539, 2109, 2934, 3280, 3281, 3351, 3723, 4026, 4374, 5481, 5793, 6318, 6561.

$HDC_3(8,5)$: 27, 204, 229, 1388, 1786, 1870, 1914, 2341, 3624, 3959, 4406, 4503, 5803, 6195.

$HDC_3(8,6)$: 1, 365, 463, 729, 871, 930, 1718, 2328, 3268, 3281, 3619, 4361, 4752, 4779, 5859.

$HDC_3(8,7)$: 77, 321, 807, 808, 992, 1136, 2023, 2095, 2211, 2280, 2358, 3257, 4156, 4252, 4321, 4494, 4599, 5393, 5471, 5540, 6481.

$D_3(5,1)$: 9, 10, 14, 17, 21, 36, 37, 49, 52, 56, 59, 69, 79, 85, 86, 97, 99, 102, 117, 119, 128, 129, 136, 148, 150, 161, 167, 178, 183, 198, 200, 211, 217, 231, 242.

$D_3(5,2)$: 1, 2, 14, 27, 58, 64, 110, 121, 122, 133, 134, 166, 172, 219, 231, 241, 243.

$D_3(6,3)$: 1, 26, 45, 76, 127, 150, 168, 197, 236, 276, 307, 344, 365, 381, 386, 417, 419, 457, 518, 534, 542, 583, 591, 606, 607, 659, 706, 729.

$D_3(6,2)$: 1, 4, 15, 26, 32, 34, 39, 70, 72, 74, 331, 337, 341, 347, 354, 358, 365, 376, 383, 399, 454, 616, 656, 658, 672, 681, 691, 696, 704, 715, 729.

$D_3(6,1)$: 1,2, 15, 26, 29, 32, 44, 48, 51, 54, 58, 61, 70, 76, 87, 90, 99, 103, 118, 119, 121, 146, 149, 155, 158, 161, 168, 173, 177, 184, 188, 196, 212, 213, 219, 229, 232, 235, 237, 252, 253, 261, 270, 274, 283, 287, 309, 317, 320, 325, 326, 339, 344, 354, 365, 371, 376, 379, 393, 396, 402, 415, 416, 427, 435, 446, 457, 464, 467, 491, 500, 507, 519, 530, 532, 538, 546, 552, 565, 583, 584, 588, 589, 600, 602, 622, 629, 648, 655, 669, 682, 687, 693, 698, 704, 710, 715, 729.

16 Appendix D: MATLAB codes

Below is the source code used to generate adjacency matrices A.

```
function [x, fval] = GreedyAlg(A)
tic

x = zeros(size(A,2),1);

scoreM = A;
score = sum(scoreM);

while nnz(A*x) < size(A,1)
    x(find(score == max(score),1)) = 1;
    scoreM(A*x == 1,:) = 0;
    score = sum(scoreM);
end

fval = nnz(x);

end
```

Below is the source code for the greedy algorithm. Input requires the adjacency matrix A.

```
function [x, fval] = GreedyAlg(A)
tic

x = zeros(size(A,2),1);

scoreM = A;
score = sum(scoreM);

while nnz(A*x) < size(A,1)
    x(find(score == max(score),1)) = 1;
    scoreM(A*x == 1,:) = 0;
    score = sum(scoreM);
end

fval = nnz(x);

end
```

Below is the source code for the genetic algorithm. Input requires the adjacency matrix A and the lower bound lb. Note that the code can be generalized to work for cases with $q > 3$ or $n > 13$ if the part that defines q and n is removed, and they instead become part of the input.

```
function [C,K] = GenAlg(A,lb)
%% Initialization
```

```
tic

% We determine q and n by A's size
if mod(size(A,1),2) == 0
    q = 2;
else
    q = 3;
end
for i = 1:13
    if q^i == size(A,1)
        n = i;
        break
    end
end

F = dec2base(0:q^n-1,q,n)-48; % The matrix containing all words

time = 300;
% The maximum time that we want to calculate
POPSIZE = 40-q*n;
% The maximum population size
REPNUMBER = ceil(POPSIZE/3);
% The number of reproductions that we want to perform in each generation
MUTNUMBER = floor(2*POPSIZE/3);
% The number of mutations that we want to perform in each generation
GENNUMBER = 9999999;
% The maximum number of generations that we want to run

population = zeros(q^n,POPSIZE);
fitness = zeros(1,POPSIZE);
neighbors = zeros(q^n,POPSIZE);
score = zeros(q^n,POPSIZE);

fprintf('\nInitializing population \n');
K = q^n; % initialize fitness of the empty code

% We generate an initial population
for p = 1:POPSIZE

    % Initialize the input for mutation
    incCode = population(:,p);
    incFitness = q^n;
    incNeighbors = zeros(q^n,1);
    incScore = (-sum(A(:,1))+1)*ones(q^n,1);

    % we can find a better code by adding or removing
    % the word for which the score is negative
    while min(incScore) < 0 && toc < time
        [incCode,incFitness,incNeighbors,incScore] = ...
            mutation(incCode,incNeighbors,incFitness,incScore,F,A,q,n);
        % We find a local optimum for each code in the initial population
    end

    % Store output of mutation
    population(:,p) = incCode;
    fitness(1,p) = incFitness;
    neighbors(:,p) = incNeighbors;
    score(:,p) = incScore;
end

%Display results of the initial population
[K,I] = min(fitness); x = toc;
fprintf('Smallest cardinality in initial population: %d\n', K);
fprintf('Time elapsed: %f', x);

% We save the initial best solution
C = population(:,I);
C(neighbors(:,I) == 0) = C(neighbors(:,I) == 0) + 1;
% We update the solution code to its feasible counterpart


%% Producing new generations
gen = 0;
x = toc;

while gen < GENNUMBER && x < time && min(fitness) > lb

    % Initialize new population
    newPopulation = zeros(q^n,REPNUMBER + MUTNUMBER);
    newNeighbors = zeros(q^n,REPNUMBER + MUTNUMBER);
    newFitness = zeros(1,REPNUMBER + MUTNUMBER);
    newScore = zeros(q^n,REPNUMBER + MUTNUMBER);

    % We generate REPNUMBER reproductions
    k = 1;
    while k <= REPNUMBER

    row = find(fitness == min(fitness));
    % The fittest parent will be selected

    % We build in an 'escape' probability for local optima
    if length(row) > 1 && rand < 0.8

        pick1 = row(ceil(rand*length(row)));
        pick2 = row(ceil(rand*length(row)));

        while pick2 == pick1 % Make sure that we have two different parents
            pick2 = row(ceil(rand*length(row)));
        end
    else
        if rand < 0.8
            pick1 = row(1);
            pick2 = ceil(rand*POPSIZE);
        else
            pick1 = ceil(rand*POPSIZE);
            pick2 = ceil(rand*POPSIZE);
        end

        while pick2 == pick1 % Make sure that we have two different parents
            pick2 = ceil(rand*POPSIZE);
        end
    end

    % Define the scores of the parents
    codeScore1 = score(:,pick1);
    codeScore2 = score(:,pick2);

    [repCode,repFitness,repNeighbors,repScore] = ...
        reproduction(codeScore1,codeScore2,F,A,q,n,lb);

    %insert the new codes in the newPopulation matrix
    newPopulation(:,k) = repCode;
    newNeighbors(:,k) = repNeighbors;
    newFitness(1,k) = repFitness;
    newScore(:,k) = repScore;

    k = k+1; % We move to the next reproduction
    end

    % We generate mutations
    k = 1;
    while k <= MUTNUMBER
        % We build in an 'escape' probabilty for local optima
        if rand < 0.8
            row = find(fitness == min(fitness));
            pick = row(ceil(rand*length(row)));
        else
            pick = ceil(rand*length(POPSIZE));
        end

        %Define the input for mutation
        code = population(:,pick);
        codeNeighbors = neighbors(:,pick);
        codeFitness = fitness(pick);
        codeScore = score(:,pick);

        % We mutate the chosen code
        [mutCode,mutFitness,mutNeighbors,mutScore] = ...
            mutation(code,codeNeighbors,codeFitness,codeScore,F,A,q,n);

        %insert the new code in the newPopulation matrix
        newPopulation(:,REPNUMBER+k) = mutCode;
        newNeighbors(:,REPNUMBER+k) = mutNeighbors;
        newFitness(1,REPNUMBER+k) = mutFitness;
        newScore(:,REPNUMBER+k) = mutScore;

        k = k+1; % We move to the next mutation
    end

    % We use a 2-code tournament to select population for the next
    % generation. This way, the best code always remains in the population,
    % and weak codes still have a small chance of staying in the
    % population, which can help us escape local optima
    [population,fitness,neighbors,score] = ...
        selection(population,newPopulation,fitness,newFitness,neighbors,...
        newNeighbors,score,newScore,POPSIZE,q,n);

    %Plot the results
%     fitBounds(gen + 1, :) = [mean(fitness),min(fitness)];
%     lastGens = ceil(100/(n*q));
%     lastBest = fitBounds(max(gen-lastGens,1):gen+1,:);
%     xas = max(gen−lastGens,1):gen+1; % define x−as for the zoomed plot
%     subplot(1,2,1);% plot of all generations and the cardinalities
%         plot(fitBounds)
%         title('Cardinality found in each generation');
%         xlabel('Generation number');
%         ylabel('Cardinality');
%         legend('Mean cardinality','Minimum cardinality');
%         drawnow
%     subplot(1,2,2); % the zoomed plot
%     plot(xas,lastBest)
%         title('Cardinality of the last generations');
%         xlabel('Generation number');
%         ylabel('Cardinality');
%         legend('Mean cardinality','Minimum cardinality');
%         drawnow
%
%Display results of this generation
%fprintf('\nGeneration number: %d \n',gen+1);
%fprintf('Smallest possible cardinality found so far: %d\n', min(fitness));

    gen = gen + 1; % We update the generation number
    x = toc; % We update the time passed


    if min(fitness)< K
        fprintf('\nNew smallest cardinality found: %d', min(fitness));
        fprintf('\nTime elapsed: %f', x);
        [K,I] = min(fitness);
        C = population(:,I);
        C(neighbors(:,I) == 0) = C(neighbors(:,I) == 0) + 1;
    end
    % We update the solution code to its feasible counterpart


end

%% Generate results


fprintf('\nTime elapsed: %f seconds', x);
fprintf('\nNumber of generations: %d', gen);
if round(x) >= time
    fprintf('\nStopping criterium: Time limit reached\n');
else
    if K == lb
        fprintf('\nStopping criterium: Optimum reached\n');
    else
        fprintf('\nStopping criterium: Max # of generations reached\n');
    end
end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [mutCode,mutFitness,mutNeighbors,mutScore] = ...
    mutation(code,codeNeighbors,codeFitness,codeScore,F,A,q,n)
```

```
mutCode = code;
mutNeighbors = codeNeighbors;
mutScore = codeScore;
mutFitness = codeFitness;

newWords = find(mutScore == min(mutScore));
% We find the words that reduce fitness the most when we add/subtract them
pick = newWords(ceil(rand*length(newWords)));

    mutFitness = mutFitness + codeScore(pick);

    if mutCode(pick) == 1
        % This means that the word is in the code, so we need to remove it

        mutCode(pick) = 0;
        neighborhood = A(:,pick);
        mutNeighbors=mutNeighbors-A(:,pick); % We update the neighbors

        % We update the score
        row = find(neighborhood == 1);
        for k = 1:length(row)
            if mutNeighbors(row(k)) == 0
                neighborhood2 = A(:,row(k));
                mutScore = mutScore - neighborhood2;
            end
            if mutNeighbors(row(k)) == 1
                neighborhood2 = A(:,row(k));
                mutScore = mutScore + neighborhood2.*mutCode;
            end
        end

        mutScore(pick) = -codeScore(pick);

    else % This means that the word is not in the code, so we add it

        mutCode(pick) = 1;
        neighborhood = A(:,pick);
        mutNeighbors=mutNeighbors+neighborhood;

        % We update the score
        row = find(neighborhood == 1);
        for k = 1:length(row)
            if mutNeighbors(row(k)) == 1
                neighborhood2 = A(:,row(k));
                mutScore(mutCode == 0) = mutScore(mutCode == 0) + ...
                    neighborhood2(mutCode == 0);
            end
            if mutNeighbors(row(k)) == 2
                neighborhood2 = A(:,row(k));
                mutScore = mutScore - neighborhood2.*mutCode;
            end
        end
        mutScore(pick) = -codeScore(pick);
    end

    if mutFitness >= codeFitness
```

```
        tabuScore = mutScore; tabuScore(pick) = 999999;
        newWords = find(tabuScore == min(tabuScore));
        % We find the words that will reduce fitness the most
        pick = newWords(ceil(rand*length(newWords)));

        mutFitness = mutFitness + codeScore(pick);

        if mutCode(pick) == 1
            % This means that the word is in the code, so we remove it

            mutCode(pick) = 0;
            neighborhood = A(:,pick);
            mutNeighbors=mutNeighbors-A(:,pick); % We update the neighbors

            % We update the score
            row = find(neighborhood == 1);
            for k = 1:length(row)
                if mutNeighbors(row(k)) == 0
                    neighborhood2 = A(:,row(k));
                    mutScore = mutScore - neighborhood2;
                end
                if mutNeighbors(row(k)) == 1
                    neighborhood2 = A(:,row(k));
                    mutScore = mutScore + neighborhood2.*mutCode;
                end
            end

            mutScore(pick) = -codeScore(pick);

        else % This means that the word is not in the code, so we add it

            mutCode(pick) = 1;
            neighborhood = A(:,pick);
            mutNeighbors=mutNeighbors+neighborhood;

            % We update the score
            row = find(neighborhood == 1);
            for k = 1:length(row)
                if mutNeighbors(row(k)) == 1
                    neighborhood2 = A(:,row(k));
                    mutScore(mutCode == 0) = mutScore(mutCode == 0) + ...
                        neighborhood2(mutCode == 0);
                end
                if mutNeighbors(row(k)) == 2
                    neighborhood2 = A(:,row(k));
                    mutScore = mutScore - neighborhood2.*mutCode;
                end
            end
            mutScore(pick) = -codeScore(pick);
        end
    end

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [repCode,repFitness,repNeighbors,repScore] = ...
    reproduction(codeScore1,codeScore2,F,A,q,n,lb)
```

```
repCode = zeros(q^n,1);
repFitness = q^n;
repNeighbors = zeros(q^n,1);
repScore = (-sum(A(:,1))+1)*ones(q^n,1);

genes1 = find(abs(codeScore1) > (q^n)/(2*lb));
genes2 = find(abs(codeScore2) > (q^n)/(2*lb));
genes = union(genes1,genes2);

for i = 1:length(genes)

    pick = genes(i);
    repFitness = repFitness + repScore(pick);

    if repCode(pick) == 1
        % This means that the word is in the code, so we need to remove it

        repCode(pick) = 0;
        neighborhood = A(:,pick);
        repNeighbors=repNeighbors-neighborhood; % We update the neighbors

        % We update the score
        row = find(neighborhood == 1);
        for k = 1:length(row)
            if repNeighbors(row(k)) == 0
                neighborhood2 = A(:,row(k));
                repScore = repScore - neighborhood2;
            end
            if repNeighbors(row(k)) == 1
                neighborhood2 = A(:,row(k));
                repScore = repScore + neighborhood2.*repCode;
            end
        end

        repScore(pick) = -repScore(pick);

    else % This means that the word is not in the code, so we add it

        repCode(pick) = 1;
        neighborhood = A(:,pick);
        repNeighbors=repNeighbors+neighborhood;

        % We update the score
        row = find(neighborhood == 1);
        for k = 1:length(row)
            if repNeighbors(row(k)) == 1
                neighborhood2 = A(:,row(k));
                repScore(repCode == 0) = repScore(repCode == 0) + ...
                    neighborhood2(repCode == 0);
            end
            if repNeighbors(row(k)) == 2
                neighborhood2 = A(:,row(k));
                repScore = repScore - neighborhood2.*repCode;
            end
        end
        repScore(pick) = -repScore(pick);
    end
```

```
end

while min(repScore) < 0
    [repCode,repFitness,repNeighbors,repScore] = ...
        mutation(repCode,repNeighbors,repFitness,repScore,F,A,q,n);
end

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [population,fitness,neighbors,score] = ...
    selection(population,newPopulation,fitness,newFitness,neighbors,...
    newNeighbors,score,newScore,POPSIZE,q,n)

    % We merge the original and new population into one set
    newPopulation = [population newPopulation];
    newFitness = [fitness newFitness];
    newNeighbors = [neighbors newNeighbors];
    newScore = [score newScore];

    tournamentOrder = randperm(size(newPopulation,2));
    % We randomly match different codes for the selection tournament

    % We initialize the matrices for the next generation
    population = zeros(q^n,POPSIZE);
    fitness = zeros(1,POPSIZE);
    neighbors = zeros(q^n,POPSIZE);
    score = zeros(q^n,POPSIZE);

    for p = 1:POPSIZE
        code1 = tournamentOrder(2*p-1); code2 = tournamentOrder(2*p);
        if newFitness(code1) < newFitness(code2)
            population(:,p)=newPopulation(:,code1);
            fitness(1,p)=newFitness(1,code1);
            neighbors(:,p)=newNeighbors(:,code1);
            score(:,p)=newScore(:,code1);
        else
            population(:,p)=newPopulation(:,code2);
            fitness(1,p)=newFitness(1,code2);
            neighbors(:,p)=newNeighbors(:,code2);
            score(:,p)=newScore(:,code2);
        end
    end
end
```

Below is the source code for the level 1 Sherali-Adams relaxation.

```
function [x, fval] = SARelax1(a,J,Sets,orbitSizes,ub)

% Input information:
% a is the first constraint of the adjacency constraint matrix A that
% we want to relax.
% J is the vector of possible vertices that can form J.
% Sets is the q^n x q^n matrix where Sets(i,j) gives the number of the
% orbit the set is in.
% orbitSizes gives the sizes of the orbits in Sets.
% ub gives the best known upper bound of the instance
```

```
constraintVertices = find(a);
numOrbits = max(max(Sets));

SA = zeros(2*size(J,1)+ub-1,2*(1+numOrbits)+ub);
b = zeros(2*size(J,1)+ub-1,1);
% This will be the constraint system for the SA relaxation

for j = 1:size(J,1)

    b(2*j) = -1; % This is the homogeneous result from (a-1)(1-x).
    % Note thatthere is no homogeneous term in (a-1)x

    SA(2*j-1,1) = 1; % -1*x part from (a-1)*x
    SA(2*j,1) = -size(constraintVertices,2)-1; % a*1 part from (a-1)(1-x)

    for v = 1:size(constraintVertices,2)
        if J(j) == constraintVertices(v)
            SA(2*j-1,1) = SA(2*j-1,1) - 1; % Since xi*xi = xi
            SA(2*j,1) = SA(2*j,1) + 1;
        else
            orbit = Sets(J(j),constraintVertices(v));
            SA(2*j-1,1+orbit)=SA(2*j-1,1+orbit)-1;
            SA(2*j,1+orbit)=SA(2*j,1+orbit)+1;
        end
    end
end

% We set the orbit and dummy variables in this constraint
SAeq(1:2*(numOrbits+1)+ub) = [orbitSizes(1),zeros(1,1+2*numOrbits), ...
    -ones(1,ub)]; beq = 0;
for o = 1:numOrbits+1
    SAeq(size(SAeq,1)+1,o) = -orbitSizes(o);
    SAeq(size(SAeq,1),1+numOrbits+o) = 1;
    beq(1+o,1) = 0;
end

% We set an ordering on the dummy variables
for d = 1:ub-1
    SA(size(SA,1)+1,2*(numOrbits+1)+d)=-1;
    SA(size(SA,1),2*(numOrbits+1)+d+1) = 1;
    b(size(SA,1)) = 0;
end

% We add the counting constraint
SA(size(SA,1)+1,:) = 0; b(size(SA,1)) = 0;
for o = 1:numOrbits
    SA(size(SA,1),2+numOrbits+o) = -1;
end
for d = 2:ub
    SA(size(SA,1),2*(numOrbits+1)+d)=d-1;
end

f = [orbitSizes(1),zeros(1,1+2*numOrbits+ub)]; % The objective function
% consists only of the cost of the single variable, multiplied by the
% cardinality of its orbit
```

```
options = 'emphasis.numerical'; % We use the simplex method for optimizing,
% because the (default) interior point method may be slightly inaccurate,
% which can lead to unreliable results when working with large orbits.

% We create the variable types. Note that all types are continuous except
% for the orbit representatives of size 2
ctype = [];
for c = 1:1+numOrbits
    ctype = [ctype,'C'];
end
for c = 1+numOrbits+1:2*(numOrbits+1)
    ctype = [ctype,'I'];
end
for c = 2*(numOrbits+1)+1:2*(numOrbits+1)+ub
    ctype = [ctype,'C'];
end

% We solve the linear relaxation
[x,fval] = cplexmilp(f,SA,b,SAeq,beq,[],[],[],zeros(2*(numOrbits+1)+ub,1),...
    [ones(1+numOrbits,1);ub;nchoosek(ub,2)*ones(numOrbits,1);ones(ub,1)]...
    ,ctype,[],options);

end
```