

A Polynomial Algorithm for Codes Based on Directed Graphs

A.V. Kelarev

School of Computing
University of Tasmania, Private Bag 100
Hobart, Tasmania 7001, Australia
Email: Andrei.Kelarev@utas.edu.au
www.comp.utas.edu.au/users/kelarev

Abstract

A complete description and proof of correctness are given for a new polynomial time algorithm for a class of codes based on directed graphs and involving construction well known in system theory. Our construction has already been considered in the literature in relation to other questions. The investigation of codes in this graph-based construction is inspired by analogy with classical cyclic codes that are defined in a similar way in polynomial rings. We show that all cyclic codes can be embedded in this construction. For each graph, the algorithm computes the largest number of errors which can be corrected by codes defined with this graph. In addition, it finds a generator of a code with this optimum value.

Keywords: algorithms, coding, directed graphs.

1 Introduction

It has been established by Downey, Fellows, Whittle, and Vardy that several fundamental problems concerning linear codes are NP-complete and W[1]-hard, see (Downey & Fellows 1999, Downey & Fellows 1999b, Downey, Fellows, Whittle & Vardy 2001). The aim of this paper is to develop a polynomial time algorithm for a class of codes inspired by analogy with classical cyclic codes. We are applying directed graphs to define a class of error-correcting codes and develop a polynomial algorithm for computing the number of errors these codes can correct. Our first main theorem proves the correctness and evaluates the running time of the algorithm.

As a guide we are motivated by analogy with the fact that all cyclic codes, including various efficient codes used in practice, are specified in polynomial rings. We are going to use another important construction defined in terms of directed graphs and considered by many authors, see (Kelarev 2002) for references. It is shown that all cyclic codes can be embedded in it. This construction enables us to define a class of error-correcting codes specified in terms of directed graphs.

It is natural to investigate how properties of the code depend on the properties of the graph that defines it. We develop a polynomial algorithm for computing the number of errors which can be corrected by codes defined with any given graph. In addition,

Copyright ©2006, Australian Computer Society, Inc. This paper appeared at Computing: The Australasian Theory Symposium (CATS2006), Hobart, Australia. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 51. Barry Jay and Joachim Gudmundsson, Eds. Reproduction for academic, not-for profit purposes permitted provided this text is included.

This research has been supported by ARC Discovery grant DP0449469.

our algorithm finds a generator of a code with this optimum value.

We use standard concepts concerning codes following (Lidl & Niederreiter 1994), (Lidl & Niederreiter 1997), (Lidl & Wiesenbauer 1980), (Moffat & Turpin 2002), (Pless, Huffman & Brualdi 1998) and include all the necessary information for convenience of the readers, see also (Asano, Wada & Masuzawa 2003), (Cormen, Leiserson, Rivest & Stein 2001), (Kelarev 2001), (Kelarev 2002), (Kelarev 2003), (Kelarev 2004), (Kelarev & Sokratova 2001), (Kelarev & Solé 2000), (Cazaran, Kelarev, Quinn & Vertigan 2006). As it is customary in the literature, by a polynomial algorithm we mean an algorithm with polynomial running time. It is worth mentioning that in fact our algorithm solves a special case of the more general Problem 10.1 recorded in (Kelarev 2002).

There are many known cyclic and linear codes used in practice. For example, the BCH codes are used in CDs, DVDs, mobile phones and digital television, see (Stallings 2002). The second main theorem of this paper demonstrates that our construction is general enough to incorporate all cyclic codes providing additional structure for them.

It is important to emphasise that the research on algorithmic aspects of coding theory brings practical benefits as a result of cumulative combined effect of many incremental steps in the investigation conducted by many researchers throughout the world. For example, it took decades from the invention of the BCH codes to their practical uses in CDs, DVDs, mobile phones and digital television.

The author is grateful to Mike Fellows for permission to use his idea of simplifying the main algorithm that has substantially reduced its complexity. It has led to a much simpler polynomial algorithm and was suggested to the author during Mike's visit to the School of Computing at the University of Tasmania.

2 Main results

Throughout the word 'graph' will mean a directed graph without multiple edges but possibly with loops. Let $D = (V, E)$ be a graph with the set $V = \{v_1, \dots, v_n\}$ of vertices and a set $E \subseteq V \times V$ of edges. Following standard conventions of coding theory, let us denote by $F = F_q$ an encoding alphabet regarded as a finite field. The incidence ring of D is denoted by $I_D(F)$ and is defined as the set of all formal finite sums of edges in E with coefficients in F . It is endowed with multiplication defined by the distributive laws and the rule

$$(x, y) \cdot (z, w) = \begin{cases} (x, w) & \text{if } y = z, (x, w) \in E, \\ 0 & \text{otherwise,} \end{cases} \quad (1)$$

for $x, y, z, w \in V$, see, for example, (Kelarev 2002), §3.15. The graph D is said to be *balanced* if, for

all $x_1, x_2, x_3, x_4 \in V$ with $(x_1, x_2), (x_2, x_3), (x_3, x_4), (x_1, x_4) \in E, (x_1, x_3) \in E \Leftrightarrow (x_2, x_4) \in E$. It is well known and fairly easy to verify that the above definition correctly defines multiplication as an associative operation on $I_D(F)$ if and only if D is balanced.

Thus, every element x of $I_D(F)$ is uniquely represented as a sum of the form $x = \sum_{(u,v) \in E} x_{(u,v)}(u, v)$, where $x_{(u,v)} \in F$ and only a finite number of the coefficients $f_{(u,v)}$ are nonzero.

As a motivating example, let us recall that every cyclic code can be defined with a generator polynomial as follows. Recall that the polynomial quotient ring

$$Q_n = F[x]/(1 - x^n) \quad (2)$$

consists of all polynomials with one variable x and degree $\leq n$, where addition of polynomials is defined as usual, and multiplication is performed modulo $1 - x^n$. This means that the product of two polynomials in $F[x]/(1 - x^n)$ is equal to the remainder of their product in $F[x]$ upon division by $1 - x^n$. For any $g(x) \in F[x]/(1 - x^n)$, the cyclic code $(g(x))$ generated by $g(x)$ is the set of all multiples of $g(x)$, i.e., all elements of the form

$$(f_0 + f_1x + \dots + f_{n-1}x^{n-1})g(x),$$

where $f_0, f_1, \dots, f_{n-1} \in F$. Similarly, we say that the error-correcting code generated by the elements g_1, \dots, g_k in $I_D(F)$ is the set

$$\begin{aligned} C_D(g_1, \dots, g_k) &= \\ &= \{(f_1 + h_1)g_1 + \dots + (f_k + h_k)g_k \mid \text{where} \\ &\quad f_1, \dots, f_k \in F, h_1, \dots, h_k \in I_D(F)\}. \end{aligned} \quad (3)$$

Theorem 1 *For each balanced directed graph $D = (V, E)$, Algorithm 1 finds the number of errors that error-correcting codes $C_D(g_1, \dots, g_k)$ can correct and returns a generator g of the optimal code $C_D(g)$ which achieves this error-correcting capability. The running time of the algorithm is $O(n^3)$.*

A nice and unexpected conclusion is that it turns out possible to generate an optimal code with just one generator. Thus, a perfect analogy with cyclic codes occurs, despite the fact that not all codes can be generated by one element in $I_D(F)$.

It is worth noting that a brute force algorithm for solving the same task would be exponential, because it is fairly easy to show that the number of codes of the form $C_D(g_1, \dots, g_k)$, and even the number of pairwise incomparable with respect to inclusion codes of the form $C_D(g)$, intricately depends on the graph D and can grow exponentially with n . Together with NP-completeness and W[1] hardness results due to (Downey et al. 2001), see also (Downey & Fellows 1999, Downey & Fellows 1999b), it shows that devising a polynomial algorithm in this situation was not easy.

Theorem 2 *Every cyclic code C can be embedded in $I_D(F)$ for some D and F so that C is generated by one element.*

Theorem 3 *Every linear code can be embedded in $I_D(F)$ for some D and F .*

3 Main algorithm

A pseudocode with concise description of the main algorithm is given in Figure 1. This section supplies additional intuitive explanation of the steps of the algorithm. Algorithm 1 utilizes only properties of the

Algorithm 1 *Given a balanced directed graph $D = (V, E)$, returns the largest number of errors that codes in $I_D(F)$ can correct and an element generating an optimal code of this sort.*

```

1. int i, j, b = 0, c = n*n;
2. L = a set of triplets represented as
   a red-black tree, initially empty;
3. for ( i = 1; i <= n; i++ )
4.   find In(v_i) = {x in V | (x, v_i) in E};
5. for ( j = 1; j <= n; j++ ) {
6.   for ( v in In(v_j) ) {
7.     S = In(v) ∩ In(v_j);
8.     if ( ∃T : (v, S, T) ∈ L )
9.       { t++; insert v_j in T; }
10.    else
11.      insert (v, S, {v_j}) in L;
12.   }
13. }
14. Find (v', S', T') ∈ L with maximum |T'| = b;
15. g_b = ∑_{y ∈ T'} (v', y); g_c = ∑_{w ∈ E} w;
16. for ( i = 1; i <= n; i++ ) {
17.   for ( j = 1; j <= n; j++ ) {
18.     for ( t = 1; t <= n; t++ ) {
19.       if ( (t, v_i), (t, v_j) ∈ E ) {
20.         c--;
21.         g_c = g_c - (v_i, v_j);
22.         break;
23.       }
24.     }
25.   }
26. }
27. if ( b ≥ c )
28.   return [(b - 1)/2], g_b;
29. else
30.   return [(c - 1)/2], g_c;

```

Figure 1: Main Algorithm

graphs. The notation used in the algorithm is illustrated in Figure 2. It is well known that

$$Q_n = F[x]/(1 - x^n)$$

always contains F , but $I_D(F)$ does not have to contain F . Besides, every code can be generated with just one polynomial in Q_n , whereas in $I_D(F)$ several generators can define larger classes of codes compared to just one generator.

The algorithm calculates and returns the optimal value which can be defined as follows. Let $D = (V, E)$ be a transitive graph. Recall that the *in-degree* and *out-degree* of a vertex $v \in V$ are defined by

$$\text{indeg}(v) = |\{w \in V \mid (w, v) \in E\}|, \quad (4)$$

$$\text{outdeg}(v) = |\{w \in V \mid (v, w) \in E\}|. \quad (5)$$

A vertex of D is called a *source* (*sink*) if $\text{indeg}(v) = 0$ and $\text{outdeg}(v) > 0$ (respectively, $\text{indeg}(v) > 0$, $\text{outdeg}(v) = 0$). Denote by $\text{sources}(D)$ and $\text{in}(D)$ the sets of all sources and sinks of D , respectively. For each vertex $v \in V$, put

$$\text{sources}(v) = \{u \in \text{out}(D) \mid (u, v) \in E\}, \quad (6)$$

$$\text{sinks}(v) = \{u \in \text{sinks}(D) \mid (u, v) \in E\}. \quad (7)$$

For every transitive graph it is easy to see that $\text{sources}(v)$ is equal to the set of all vertices u in $\text{sources}(D)$ such that there exists a directed path from u to v . Similarly, $\text{sinks}(v)$ coincides with the set of all vertices u in $\text{sinks}(D)$ such that there exists a directed path from v to u .

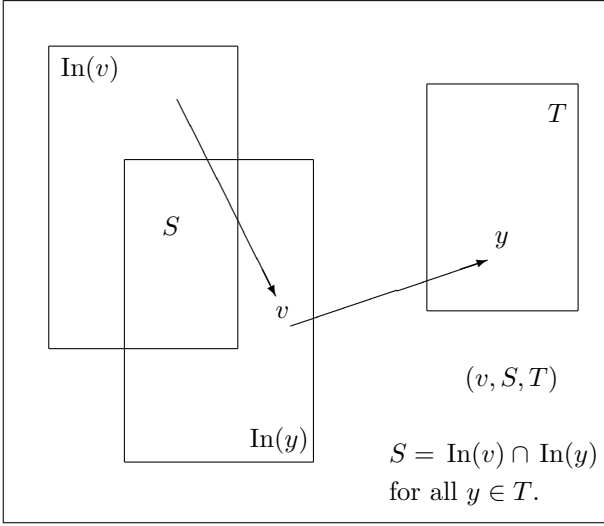


Figure 2: Steps of Algorithm 1

As we can see from Figure 1, the two key values computed by the algorithm can be defined as follows. Take a vertex $v \in V$. For each vertex $v \in V$, we introduce two special sets of vertices:

$$\text{In}(v) = \{x \in V \mid (x, v) \in E\},$$

$$\text{Out}(v) = \{x \in V \mid (v, x) \in E\}.$$

The first value b is found as the largest number of elements y in $\text{Out}(v)$ such that

$$\text{In}(y) \cap \text{In}(v) = S$$

that can be achieved for all v in V and all subsets S of $\text{In}(v)$. The second value c is the number of all edges (x, y) such that there does not exist any vertex z satisfying $(z, x), (z, y) \in E$.

Both of these values are quite sophisticated, and this is what makes the proof of our main theorem given in the next section rather nontrivial. Note that a brute force exhaustive search through all the elements v and all subsets S of $\text{In}(v)$ above requires exponential time.

4 Proofs of the main theorems

Proof of Theorem 1. In the first part of the proof we are going to show that the code $C_D(g_b)$ can correct $\lfloor (b-1)/2 \rfloor$ errors and the code $C_D(g_c)$ can correct $\lfloor (c-1)/2 \rfloor$ errors.

Minimal prerequisites on coding theory are involved in a few steps of the proof of correctness of our algorithm. Recall that the *weight* $\text{wt}_H(x)$ of the element x in $I_D(F)$ is the number of edges that occur with nonzero coefficients in x . The minimum distance of a code is the minimum weight of a difference of two distinct elements in the code. The *weight* $\text{wt}_H(C)$ of an error-correcting code C is the minimum weight of a nonzero element in C . If a code is linear, then its weight is equal to its minimum distance. A code with minimum distance d can correct $\lfloor (d-1)/2 \rfloor$ errors. Conversely, a code correcting e errors has minimum distance at least $2e+1$. Therefore, all we have to verify in the first part is that $C_D(g_b)$ has minimum distance b and $C_D(g_c)$ has minimum distance c .

First, consider the code $C_D(g_b)$. It is easily seen from line 14 in Figure 1 that Algorithm 1 ensures that

$$g_b = \sum_{y \in T'} (v', y),$$

where $(v', S', T') \in L$ and $b = |T'|$. It follows from the definition (3) that

$$C_D(g_b) = \{fg_b + hg_b \mid f \in F, h \in I_D(F)\}. \quad (8)$$

Choose a nonzero element

$$c_{\min} = fg_b + hg_b$$

with minimum weight in $C_D(g_b)$. Since the element h belongs to $I_D(F)$, the definition of $I_D(F)$ given above implies that this element can be represented in the form

$$h = \sum_{(u,v) \in E} h_{(u,v)}(u, v).$$

In view of (1) all edges (u, v) with $v \neq v'$ produce zero products in the summand hg_b of c_{\min} . Hence we can rewrite c_{\min} as

$$c_{\min} = fg_b + \sum_{(u,v') \in E} h_{(u,v')}(u, v') \sum_{y \in T'} (v', y), \quad (9)$$

where $f, h_{(u,v')} \in F$. Further, consider two possible cases.

Case 1. There exists $u \neq v'$ such that $h_{(u,v')} \neq 0$ and $(u, y) \in E$. In this case the first term

$$f \sum_{y \in T'} (v', y)$$

of c_{\min} in (9) has no summands with edges which begin in u . Therefore, if we look at the sum of the edges in c_{\min} which begin in u , then it follows from (1) that we get the expression

$$h_{(u,v')}(u, v') \sum_{y \in T'} (v', y), \quad (10)$$

which is a part of c_{\min} and does not cancel with the remaining summands of c_{\min} .

For any vertex $v \in V$, define the set

$$\text{In}(v) = \{x \in V \mid (x, v) \in E\}.$$

Notice that Algorithm 1 ensures that

$$S = \text{In}(v) \cap \text{In}(y)$$

in line 7 before it inserts y in T for $(v, S, T) \in L$ in line 9. It follows that our triple (v', S', T') satisfies $|T'| = b$ and

$$(\forall y \in T') \quad S' = \text{In}(v') \cap \text{In}(y) \quad (11)$$

Comparing (10) with (11) we see that the following two subcases are possible.

Subcase 1.1. $u \notin S'$. Then $(u, v')(v', y) = 0$ for all $y \in T'$. In this case (10) could have been eliminated from (9) which would only make c_{\min} expressed in a simpler form in (9). Since eliminations like this cannot continue indefinitely, we may assume that this subcase does not occur.

Subcase 1.2. $u \in S'$. Then $(u, v')(v', y) = (u, y)$ for all $y \in T'$, and (10) becomes

$$h_{(u,v')} \sum_{y \in T'} (u, y).$$

In this case the expression (10) contributes $|T'| = b$ to the weight of c_{\min} . Hence $w_H(c_{\min}) \geq b$.

Case 2. There does not exist $u \neq v'$ such that $h_{(u,v')} \neq 0$ and $(u, y) \in E$. In this case it follows from (1) that

$$c_{\min} = f \sum_{y \in T'} (v', y) + h_{(v',v')} \sum_{y \in T'} (v', y). \quad (12)$$

If $(v', v') \notin E$, then (10) implies

$$c_{\min} = f \sum_{y \in T'} (v', y)$$

and we get

$$w_h(c_{\min}) = |T'|.$$

If, however, $(v', v') \in E$, then (10) yields

$$c_{\min} = (f + h_{(v',v')}) \sum_{y \in T'} (v', y). \quad (13)$$

Given that c_{\min} is nonzero, we get

$$f + h_{(v',v')} \neq 0.$$

Therefore $w_h(c_{\min}) = |T'| = b$, again.

Thus, in all cases we have $w_H(c_{\min}) \geq b$. Clearly, $C_D(g_b)$ contains elements with weight b . By the choice of c_{\min} , it follows that the minimum distance of $C_D(g_b)$ is b .

Second, consider the code $C_D(g_c)$. Lines 15 to 24 of Algorithm 1 guarantee that $g_c =$

$$= \sum \{(u, v) \in E \mid (\forall t \in V)(t, u) \notin E \text{ or } (t, v) \notin E\}. \quad (14)$$

It follows from (1) that $h_{g_c} = 0$ for all $h \in I_D(F)$. Therefore (3) implies that

$$C_D(g_c) = \{fg_b \mid f \in F\}. \quad (15)$$

Hence the minimum distance of $C_D(g_c)$ is c indeed.

It remains to show that $I_D(F)$ never has codes of the form $C_D(g_1, \dots, g_k)$ which can correct more errors than the best of the codes $C_D(g_b)$ and $C_D(g_c)$. (Note that some of the codes $C_D(g_1, \dots, g_k)$ may have larger information rates.) Let us consider an arbitrary code $C = C_D(g_1, \dots, g_k)$. We have to verify that the number of errors it can correct does not exceed

$$\max\{\lfloor (b-1)/2 \rfloor, \lfloor (c-1)/2 \rfloor\}.$$

To this end we'll show that the minimum distance of C is at most $\max\{b, c\}$.

Let us start with a nonzero element $x = \sum_{i=1}^m f_i(x_i, y_i)$ which has minimum weight d in C , where $(x_i, y_i) \in E$, $f_i \in F$ and $f_i \neq 0$ for all i . We are to check that $\text{wt}_H(x) \leq \max\{b, c\}$. Consider several possible cases.

Case 1. There exists $u \in E$ such that $(u, x_i), (u, y_i) \in E$ for some i . Put $v = x_i$. Then it follows from (1) that $(u, v)x \neq 0$ and that all edges occurring in $(u, v)x$ begin in u . By the minimality of $w_H(x)$, we see that $w_H((u, v)x) = w_H(x)$, and so $(u, v)x = x$. Therefore, by (1) we have $x_1 = \dots = x_m = v$ and

$$x = \sum_{i=1}^m f_i(v, y_i), \quad (16)$$

where $(u, y_i) \in E$ for all i .

Suppose that there exist $1 \leq i, j \leq m$ such that

$$\text{In}(v) \cap \text{In}(y_i) \neq \text{In}(v) \cap \text{In}(y_j).$$

Without loss of generality we may assume that

$$\text{In}(v) \cap \text{In}(y_i) \subset \text{In}(v) \cap \text{In}(y_j).$$

Choose any

$$w \in \text{In}(v) \cap \text{In}(y_j) \setminus \text{In}(v) \cap \text{In}(y_i).$$

By the choice of w and (1), we get

$$(w, v)(v, y_i) = 0, \quad (17)$$

$$(w, v)(v, y_j) = (w, y_j), \quad (18)$$

because $(w, y_i) \notin E$ and $(w, y_j) \in E$. Since x is in $C_D(g_1, \dots, g_k)$, it follows from (3) that $(w, v)x$ belongs to $C_D(g_1, \dots, g_k)$, too. Besides, (18) implies that $(w, v)x \neq 0$. However, (17) shows us that $w_H((w, v)x) < w_H(x)$. This contradicts the minimality of $w_H(x)$, and demonstrates that, for all $1 \leq i, j \leq m$,

$$\text{In}(v) \cap \text{In}(y_i) = \text{In}(v) \cap \text{In}(y_j) = S. \quad (19)$$

Here we have denoted the intersection which occurs in (19) by S .

Let i be the smallest positive integer such that $1 \leq i \leq n$ and $\text{In}(v) \cap \text{In}(y_i) = S$. (Obviously, $i \leq b$.) It is straightforward that Algorithm 1 inserts $(v, S, \{v_i\})$ in L in line 11. Hence, when L is complete and b is being found, there will always exist T and t such that (v, S, T) belongs to L .

It follows from (19) that $y_1, \dots, y_m \in T$. Therefore $m \leq |T| = t \leq b$ by the choice of b in line 14 of the algorithm. However, $d = m$. Thus $d \leq b$ in this case.

Case 2. For all $u \in V$ and all $1 \leq i \leq m$, if $(u, x_i) \in E$ then $(u, y_i) \notin E$. Then the containment condition in line 19 of Algorithm 1 never holds true for $v = x_i, y = y_i$. Therefore line 21 is never executed for edges (x_i, y_i) which occur in x . Hence all of these edges remain in g_c , and in this case we get

$$w_H(x) \leq w_H(g_c).$$

This demonstrates that Algorithm 1 indeed returns the correct largest number of errors that a code of the form $C_D(g_1, \dots, g_k)$ can correct.

Let us now evaluate the running time of Algorithm 1. It is clear that lines 3, 4 execute in $O(n^2)$ time.

The loops in lines 5, 6 have n^2 iterations. Each iteration requires $O(n)$ to find the intersection in line 7. Given the upper bound on $|L|$, the containment condition in line 8 can be verified in $O(\lg(n^2)) = O(\lg(n))$ time, see (Asano et al. 2003) and (Cormen et al. 2001), Chapter 13. Each insertion in T in line 9 takes $O(n)$, and could even be reduced to $O(\lg(n))$, but this is not necessary. Similarly, each insertion in L in line 11 can be done in $O(\lg(n))$. Therefore the nested loops in lines 5 to 11 run in $O(n^3)$ time.

Line 14 executes in $O(\lg(n))$, and line 15 takes $O(n^2)$ to compute. The running time of lines 16 to 24 is $O(n^3)$. Therefore, the total running time of Algorithm 1 is $O(n^3)$. \square

Proof of Theorem 2. Suppose that the cyclic code C consists of codewords

$$(a_0, \dots, a_{n-1})$$

over the finite field F as encoding alphabet, where

$$a_0, \dots, a_{n-1} \in F.$$

If we identify each codeword (a_0, \dots, a_{n-1}) with the polynomial

$$a_0 + a_1x + \dots + a_{n-1}x^{n-1} \in Q_n,$$

then the code C becomes embedded in Q_n . It is well known that then there exists one polynomial $g(x)$ in Q_n such that C coincides with the set of all multiples of $g(x)$ in Q_n , i.e.,

$$C = \{f(x) \cdot g(x) \mid f(x) \in Q_n\}. \quad (20)$$

Let $K_n = (V_n, E_n)$ be the complete graph with the set

$$V_n = \{v_1, \dots, v_n\}$$

of vertices and E containing all edges including loops. For any subgraph $D = (V, E)$ of K_n , denote by $A = A_D$ the following element of $I_{K_n}(F_q)$

$$A_D = \sum_{(i,j) \in E} (v_i, v_j) \in I_{K_n}(F_q).$$

For example, if

$$L_n = (V, \{(1,1), \dots, (n,n)\})$$

is the set of all loops, then A_L is the identity element I_n of $I_{K_n}(F_q)$. Denote by C_n the cycle

$$C_n = (V, \{(v_1, v_2), \dots, (v_{n-1}, v_n), (v_n, v_1)\})$$

regarded as a subgraph of $I_{K_n}(F_q)$, and let $y = A_{C_n}$. The modulo operator gives the remainder of m on division by n and is denoted by $m \% n = m \bmod n$. It is easy to verify that the following equalities hold in $I_{K_n}(F)$, for all nonnegative integers k, m and all powers of y ,

$$y^k = \sum_{i=1}^n e_{i, (i+k) \bmod n}, \quad (21)$$

$$\begin{aligned} y^k y^m &= \sum_{i=1}^n e_{i, (i+k+m) \bmod n} \\ &= y^{(k+m) \bmod n}. \end{aligned} \quad (22)$$

The edge sets of the graphs of

$$y, y^2, \dots, y^{n-1}$$

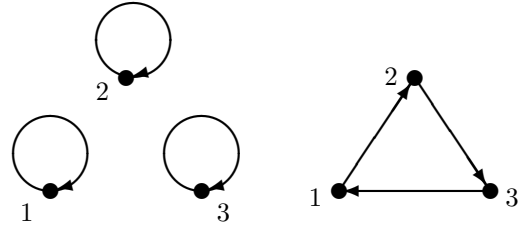
partition the set of edges of complete graph K_n . These graphs and their adjacency matrices are illustrated in Figure 3.

The equalities (22) for $1, y, \dots, y^{n-1}$ above are precisely those that are satisfied for $1, x, \dots, x^{n-1}$ according to the definition of multiplication in the quotient ring Q_n . Therefore the linear space $F[Y]$ spanned by the set

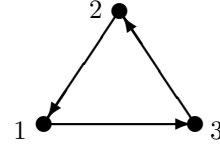
$$Y = \{1, y, y^2, \dots, y^{n-1}\} \quad (23)$$

in $I_{K_n}(F_q)$ is isomorphic to Q_n , i.e., it has the same elements and operations up to notation used for elements. If we identify the elements x^k of Q_n with elements y^k , then the parity-check code turns into the set generated in $F[Y]$ as the set of all multiples of the element

$$1 - y = \begin{bmatrix} 1 & -1 & 0 & \dots & 0 \\ 0 & 1 & -1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -1 & 0 & 0 & \dots & 1 \end{bmatrix}$$



$$1 = y^0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad y = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$



$$y^2 = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Figure 3: Adjacency matrices $1, y, y^2 \in M_3[F]$.

This example illustrates the fact that every cyclic code can be defined as a set of all multiples of one element g of the ring $I_{K_n}(F)$ multiplied by all elements of a subring of the form

$$F[Y] = Fy^0 + Fy + Fy^2 + \dots + Fy^{n-1},$$

where Y is given by (23). \square

Proof of Theorem 3. Let C be a linear (n, m) code over a finite field F , and let D be the graph with the set of vertices $V = \{1, \dots, n+1\}$ and the set of edges $\{(1, 2), (1, 3), \dots, (1, n+1)\}$. It is easily seen that the set of all elements of the form

$$c_0(1, 2) + c_1(1, 3) + \dots + c_n(1, n+1)$$

such that

$$(c_0, c_1, \dots, c_n) \in C$$

is equivalent to C regarded as a linear code. It is also easily seen that this set is closed with respect to the multiplication by arbitrary elements of $I_D(F)$. \square

5 Examples

Example 1 Let D be the graph in Figure 4. Then it is not hard to check that all sets T have cardinality at most one, for all $(v, S, T) \in L$, and so g_b generates the code which cannot correct any errors. However, the maximum number of errors which can be corrected is n with the optimal code generated by

$$g_c = (x_n, y_n) + (v, y_1) + \sum_{i=1}^n (x_i, v) + \sum_{i=1}^{n-1} (x_i, y_{i+1})$$

Example 2 Let D be the graph in Figure 5. Then the maximum number of errors which can be corrected by codes $C_D(g_1, \dots, g_k)$ is $\lfloor (n-1)/2 \rfloor$ with the optimal code generated by

$$g_b = \sum_{i=1}^n (v, y_i)$$

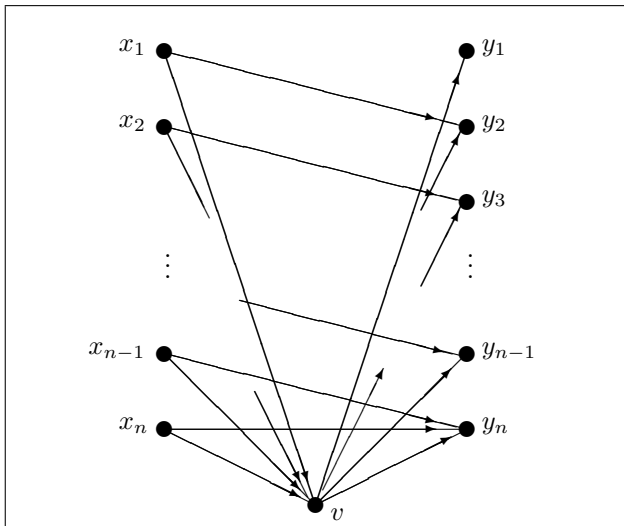


Figure 4: Graph in Example 1

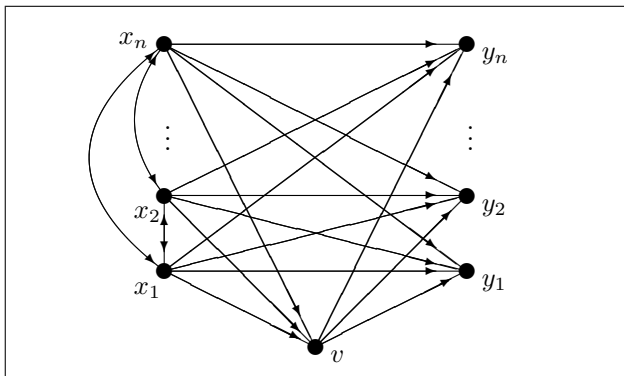


Figure 5: Graph in Example 2

Example 3 Let D be the graph in Figure 6. Then the maximum number of errors which can be corrected by codes $C_D(g_1, \dots, g_k)$ is $\lfloor n^2 - 1/2 \rfloor$ with the optimal code generated by

$$g_c = \sum_{i=1}^n \sum_{j=1}^n (x_i, y_j)$$

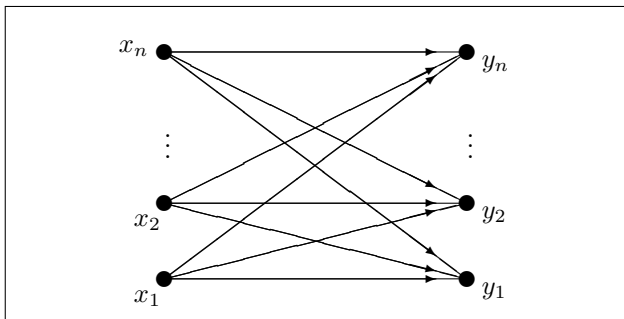


Figure 6: Graph in Example 3

The author is grateful to three referees for detailed corrections and comments that have helped to improve the exposition.

References

- Asano, T., Wada, K. & Masuzawa, T. (2003), *Theory of Algorithms*, Ohm Publishing.
- Cazaran, J., Kelarev, A.V., Quinn, S.J. Vertigan, D. (2006), An algorithm for computing the minimum distances of extensions of BCH codes, in preparation.
- Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C. (2001), *Introduction to Algorithms*, The MIT Press, Cambridge.
- Downey, R. G. & Fellows, M. R. (1999), *Parameterized Complexity*, Monographs in Computer Science, Springer, New York.
- Downey, R. G. & Fellows, M. R. (1999), Parameterized complexity after (almost) ten years: review and open questions, in 'Combinatorics, Computation & Logic '99' (Auckland), *Aust. Comput. Sci. Commun.* **21**(3), 1–33.
- Downey, R., Fellows, M. R., Whittle, G. & Vardy, A. (1999), 'The parametrized complexity of some fundamental problems in coding theory', *SIAM J. Comput.* **29**(2), 545–570.
- Kelarev, A. V. (2001), On a theorem of Cohen and Montgomery for graded rings, *Proc. Royal Soc. Edinburgh A*, **131**, 1163–1166.
- Kelarev, A. V. (2002), *Ring Constructions and Applications*, World Scientific.
- Kelarev, A. V. (2003), *Graph Algebras and Automata*, Marcel Dekker, New York.
- Kelarev, A. V. (2004), Minimum distances and information rates for matrix extensions of BCH codes, in 'The 3rd Workshop on the Internet, Telecommunications and Signal Processing', WITSP 2004, (Adelaide, 20–22 December 2004), pp. 1–6.
- Kelarev, A. V. & Sokratova, O. V. (2001), Information rates and weights of codes in structural matrix rings, *Lecture Notes in Computer Science* **2227**, 151–158.
- Kelarev, A. V. & Solé, P. (2000), Error-correcting codes as ideals in group rings, in 'Proc. Internat. Conf. AGRAM 2000', (Perth, Australia, September 2000), pp. 11–18.
- Lidl, R. & Niederreiter, H. (1994), *Introductions to Finite Fields and Their Applications*, Cambridge University Press, Cambridge.
- Lidl, R. & Niederreiter, H. (1997), *Finite Fields*, Cambridge University Press, Cambridge.
- Lidl, R. & Wiesenbauer, J. (1980), *Ring Theory and Applications*, Wiesbaden (in German).
- Moffat, A. & Turpin, A. (2002), *Compression and Coding Algorithms*, Kluwer.
- Pless, V. S., Huffman, W. C. & Brualdi, R. A., eds., (1998), *Handbook of Coding Theory*, Elsevier.
- Stallings, W. (2002), *Wireless Communications and Networking*, London, Prentice Hall.