GBF_LDOI

January 27, 2024

1 G-BF decoding for group codes

We given an example considering $G = \mathcal{C}_{31} = \{1, g, \cdots, g^{30}\}$, considering a code $\mathfrak{C} \subseteq \mathbb{F}_2 G$, with orthogonal idemponent

$e_0^+ = g^3 + g^6 + g^{12} + g^{17} + g^{24}$

[1]:
```
#Auxiliary functions

#On input a vector $\mathtt{a}$, return the corresponding
#element of the group algebra
#K denotes the field, G the group, KG the group algebra
def from_vector_to_KG(K,G,KG,a):

    a_in_KG = KG(0);
    for i in range(len(a)):

        KG_term = a[i]*KG(G[i]);

        a_in_KG += KG_term;


    return a_in_KG;


##############################

#Sample vector with weight t, length n
#K is the finite field
def low_weight_vector(K,n,t):

    q = K.cardinality();
    a = vector(K,n);
    while a.list().count(0)>(n-t):

        i = randrange(n);
        if a[i] == 0:
            val = K(1+randrange(q-1));
```

```
            a[i] = val;

    return a;

##############################
```

2 Define the code

[12]:
```
n = 31; #code length
q = 2; #finite field size

#Define group and field
G = CyclicPermutationGroup(n);

K = GF(q); #finite field

KG = GroupAlgebra(G,K); #group algebra

#Define idemponent e0_plus
e0_plus_in_KG = KG(G[3]) + KG(G[6]) + KG(G[12]) + KG(G[17]) + KG(G[24])

#Doing some sanity checks
print("e0_plus*e0_plus == e0_plus ?",e0_plus_in_KG*e0_plus_in_KG ==␣
 ↪e0_plus_in_KG)

#Define M matrix for decoding
M = matrix(K,n,n);
for i in range(n):

    g_i = G[i]*e0_plus_in_KG;

    m_i = vector(K,g_i);
    for j in range(n):
        M[j,i] = m_i[j];

#Compute also adjacency matrix
A = matrix(ZZ,n,n)
for i in range(n-1):
    for j in range(i+1,n):
        a_ij = 0
        for ell in range(n):
            if (M[ell,i]==1)&(M[ell,j]==1):
                a_ij += 1
        A[i,j] = a_ij
        A[j,i] = a_ij
```

```
#print adjacency matrix
print(" ")
print("Adjacency matrix:")
print(A)
```

```
e0_plus*e0_plus == e0_plus ? True

Adjacency matrix:
[0 0 0 1 0 1 1 1 0 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 0 1 0 0]
[0 0 0 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 0 1 0]
[0 0 0 0 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 0 1]
[1 0 0 0 0 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 0]
[0 1 0 0 0 0 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1]
[1 0 1 0 0 0 0 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1]
[1 1 0 1 0 0 0 0 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1]
[1 1 1 0 1 0 0 0 0 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0]
[0 1 1 1 0 1 0 0 0 0 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1]
[1 0 1 1 1 0 1 0 0 0 0 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1]
[1 1 0 1 1 1 0 1 0 0 0 0 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 1 1 1 1]
[1 1 1 0 1 1 1 0 1 0 0 0 0 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 1 1 1]
[1 1 1 1 0 1 1 1 0 1 0 0 0 0 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 1 1]
[1 1 1 1 1 0 1 1 1 0 1 0 0 0 0 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 1]
[1 1 1 1 1 1 0 1 1 1 0 1 0 0 0 0 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0]
[0 1 1 1 1 1 1 0 1 1 1 0 1 0 0 0 0 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0]
[0 0 1 1 1 1 1 1 0 1 1 1 0 1 0 0 0 0 0 1 0 1 1 1 0 1 1 1 1 1 1 1]
[1 0 0 1 1 1 1 1 1 0 1 1 1 0 1 0 0 0 0 0 1 0 1 1 1 0 1 1 1 1 1 1]
[1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 0 1 0 0 0 0 0 1 0 1 1 1 0 1 1 1 1]
[1 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 0 1 0 0 0 0 0 1 0 1 1 1 0 1 1 1]
[1 1 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 0 1 0 0 0 0 0 1 0 1 1 1 0 1 1]
[1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 0 1 0 0 0 0 0 1 0 1 1 1 0 1]
[1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 0 1 0 0 0 0 0 1 0 1 1 1 0]
[0 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 0 1 0 0 0 0 0 1 0 1 1 1]
[1 0 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 0 1 0 0 0 0 0 1 0 1 1]
[1 1 0 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 0 1 0 0 0 0 0 1 0 1]
[1 1 1 0 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 0 1 0 0 0 0 0 1 0]
[0 1 1 1 0 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 0 1 0 0 0 0 0 1]
[1 0 1 1 1 0 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 0 1 0 0 0 0 0]
[0 1 0 1 1 1 0 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 0 1 0 0 0 0]
[0 0 1 0 1 1 1 0 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 0 1 0 0 0]
```

3 Algorithm for G-BF decoder

[17]:
```
#G-BF decoder
def GBF(K, s_in_KG, M, threshold_BF, IterMax):

    #Represent syndrome as a vector
```

```
    s = vector(s_in_KG);

    my_err_vec = vector(K,n); #error vector estimate
    num_iter = 0; #iteration counter

    while ((s.list().count(1)>0))&(num_iter < IterMax ):

        #Compute counters
        counters = matrix(s.change_ring(ZZ))*M.change_ring(ZZ);

        #Flipping bits
        for i in range(n):
            if counters[0,i]>=threshold_BF[num_iter]:
                my_err_vec[i] += K(1);
                s += vector(M[:,i].transpose());

        num_iter += 1;

    if  s.list().count(1)>0:
        return -1 #report failure
    else:
        return my_err_vec
```

4 Set decoder and simulation parameters

- number of errors
- number of iterations
- thresholds

[22]:
```
t = 2; #number of errors
threshold_BF = [4,4]; #this must be a list with length = IterMax
num_trials = 1000; #number of decoding attempts

IterMax = len(threshold_BF)
```

[23]:
```
#Test decoding for many trials
dfr = RR(0);
for num_test in range(1,num_trials+1):

    #Sample random error vector
    err_vec = low_weight_vector(K,n,t);
    err_vec_in_KG = from_vector_to_KG(K,G,KG,err_vec);

    #Computing syndrome
    s_in_KG = err_vec_in_KG*e0_plus_in_KG;

    #Run decoder
```

```
    err_vec_estimate = GBF(K, s_in_KG, M, threshold_BF, IterMax)
    if err_vec_estimate != err_vec:
        dfr += 1;

    #Print results
    if (num_test%100)==0:
        print("t = ",t,", Num test = "+str(num_test)+", DFR = "+str(N(dfr/
↪(num_test))));
```

```
t =  2 , Num test = 100, DFR = 0.000000000000000
t =  2 , Num test = 200, DFR = 0.000000000000000
t =  2 , Num test = 300, DFR = 0.000000000000000
t =  2 , Num test = 400, DFR = 0.000000000000000
t =  2 , Num test = 500, DFR = 0.000000000000000
t =  2 , Num test = 600, DFR = 0.000000000000000
t =  2 , Num test = 700, DFR = 0.000000000000000
t =  2 , Num test = 800, DFR = 0.000000000000000
t =  2 , Num test = 900, DFR = 0.000000000000000
t =  2 , Num test = 1000, DFR = 0.000000000000000
```