Appendix A

Python program for integer $(CT_lB)_b$ codes

```
import itertools
#parameters
b=8
l=3
q=2**b-1
ebl_u = []
C = range(0,q)
R = []
# Generating the set epsilonbl
# Function for e_bl^i
def ebl(i):
    t = 1
    A = [0,1]
    A_copy_l = []
    coeff = []
    # List of l-1 number of A's
    while t<l:
        A_copy_l.append(A)
        t = t+1
# Cartesian product of l-1 number of A's
    for element in itertools.product(*A_copy_l):
```

```
        coeff.append(element)
# Calculate and store ebl(i)'s
    ebli = []
# Iteration over the list of coefficients
    c = 0
    while c<len(coeff):
        s = 2**(i-1)
        # Iteration over the coefficients in an element of coeff
           ↪ list
        p = 0
        while p<(l-1):
            s = s + coeff[c][p] * 2**(i+p)
            p = p + 1

        ebli.append(s%q)
        c = c + 1
    return(ebli)
# Function for e{bar}_bl^i i.e. Ebl(i)
def Ebl(i):
    t = 1
    A = [0,1]
    A_copy = []
    coeff = []
# List of b-i number of A's
    while t<b-i+1:
        A_copy.append(A)
        t = t+1
# Cartesian product of b-i number of A's
    for element in itertools.product(*A_copy):
        coeff.append(element)
```

```
# Calculate and store ebl(i)'s
    Ebli = []
# Iteration over the list of coefficients
    c = 0
    while c<len(coeff):
        s = 2**(i-1)
# Iteration over the coefficients in an element of coeff list
        p = 0
        while p<(b-i):
            s = s + coeff[c][p] * 2**(i+p)
            p = p + 1

        Ebli.append(s%q)
        c = c + 1
    return(Ebli)
# Unions
ebl_u = []
for i in range(1,b-l+2):
    ebl_u = list(set(ebl_u).union(set(ebl(i))))
Ebl_u = []
for i in range(b-l+2,b+1):
    Ebl_u = list(set(Ebl_u).union(set(Ebl(i))))
ebl_u = list(set(ebl_u).union(set(Ebl_u)))
#print(len(sorted(ebl_u)))
# Function to calculate CiEb
def mult(i, ebl_u):
    return [(-C[i]*j)%q for j in ebl_u]
    # Trying to find Ci's
def CE_append(i):
    flag = 0
    E = mult(i, ebl_u)
```

```
    if (len(E)!=len(set(E))):
        flag = 1
    for S in CE:
        F = list(set(S) & set(E))
        if (F!=[]):
            flag = 1
    if flag == 0:
        CE.append(E)
        R.append(i)
# Main work
CE = [ebl_u]
#range may be restricted as per the required code rate
for i in range(2,q):
    CE_append(i)
#print(CE)
#ready coefficients
print (R)
```

Appendix B

Python program for integer $LACTB_{(d/\boldsymbol{l},b)C}$ codes

```
import itertools
#parameters
b=8
l=6
d=3
q=2**b-1
ebld_u = []
C = range(0,q)
R = []
# Generating the set epsilonbld
def ebld(i):
    t = 1
    A = [0,1]
    A_copy_l = []
    coeff = []
    # List of l-1 number of A's
    while t<l:
        A_copy_l.append(A)
        t = t+1
# Cartesian product of l-1 number of A's
```

```
    for element in itertools.product(*A_copy_l):
        coeff.append(element)
# Restricting coeff
    reject = []
    for c in coeff:
        s = 0
        for p in c:
            s = s + p
        if s>d-1:
            reject.append(c)

    for r in reject:
        coeff.remove(r)
# Calculate and store ebl(i)'s
    ebli = []

    # Iteration over the list of coefficients
    c = 0
    while c<len(coeff):
        s = 2**(i-1)
# Iteration over the coefficients in an element of coeff list
        p = 0
        while p<(l-1):
            s = s + coeff[c][p] * 2**(i+p)
            p = p + 1

        ebli.append(s%q)
        c = c + 1
    return(ebli)
#print(sorted(ebld(1)))
# Union
```

```
ebld_u = []
for i in range(1,b-l+2):
    ebld_u = list(set(ebld_u).union(set(ebld(i))))


print(sorted(ebld_u))
# Function to calculate CiEb
def mult(i, ebld_u):
    return [(-C[i]*j)%q for j in ebld_u]
# Trying to find Ci's
def CE_append(i):
    flag = 0
    E = mult(i, ebld_u)
    if (len(E)!=len(set(E))):
        flag = 1
    for S in CE:
        F = list(set(S) & set(E))
        if (F!=[]):
            flag = 1
    if flag == 0:
        CE.append(E)
        R.append(i)
# Main work


CE = [ebld_u]
#range may be restricted as per the required code rate
for i in range(2,q):
    CE_append(i)
#print(CE)
#ready coefficients
print (R)
```

Appendix C

Python program for integer $HACTB_{(h/\boldsymbol{l},b)C}$ codes

```
import itertools
#parameters
b=8
l=4
h=2
q = 2**b-1
epblh = []
C = range(0,q)
R = []
# Generating the set epsilon blh
def eblh(i):
    t = 1
    A = [0,1]
    A_copy_l = []
    coeff = []
    # List of l-1 number of A's
    while t<l:
        A_copy_l.append(A)
        t = t+1
# Cartesian product of l-1 number of A's
```

```
    for element in itertools.product(*A_copy_l):
        coeff.append(element)
# Restricting coeff
    reject = []
    for c in coeff:
        s = 0
        for p in c:
            s = s + p
        if s<h-1:
            reject.append(c)
        for r in reject:
        coeff.remove(r)
# Calculate and store ebl(i)'s
    ebli = []
# Iteration over the list of coefficients
    c = 0
    while c<len(coeff):
        s = 2**(i-1)
# Iteration over the coefficients in an element of coeff list
        p = 0
        while p<(l-1):
            s = s + coeff[c][p] * 2**(i+p)
            p = p + 1
        ebli.append(s%q)
        c = c + 1
    return(ebli)
# Unions
eblh_u = []
for i in range(1,b-l+2):
    eblh_u = list(set(eblh_u).union(set(eblh(i))))
print(sorted(eblh_u))
```

```
# Function to calculate CiEb
def mult(i, eblh_u):
    return [(-C[i]*j)%q for j in eblh_u]
# Trying to find Ci's
def CE_append(i):
    flag = 0
    E = mult(i, eblh_u)
    if (len(E)!=len(set(E))):
        flag = 1
    for S in CE:
        F = list(set(S) & set(E))
        if (F!=[]):
            flag = 1
    if flag == 0:
        CE.append(E)
        R.append(i)
# Main work

CE = [eblh_u]
# the range may be managed as per the required code rate
for i in range(2,q):
    CE_append(i)
#print(CE)
print (R)
```

Appendix D

Python program for integer $(U_l SB)_b$ codes

```
import itertools
b=8
q=2**b-1
l=3
C = range(0,q)
R = []
L1=[]
L=[]
for t in range(1,l+1):
    for i in range(0,b-t+1):
        L1.append((2**i)*(2**t-1)%q)
        L1.append(-(2**i)*(2**t-1)%q)
#print(L1)
for i in L1:
    if i not in L:
        L.append(i)
#print (str(L))
# Function to calculate CiEb
def mult(i, L):
    return [(-C[i]*j)%q for j in L]
# Trying to find Ci's
```

```
# CE-set of sets, i-index for CE, j-how long have we come towards q
def CE_append(i):
    flag = 0
    E = mult(i, L)
    if (len(E)!=len(set(E))):
        flag = 1
    for S in CE:
        F = list(set(S) & set(E))
        if (F!=[]):
            flag = 1
    if flag == 0:
        CE.append(E)
        R.append(i)
# Main work
CE = [L]
for i in range(2,255):
    CE_append(i)
#print(sorted(CE))
print (R)
```

Appendix E

Python program for integer $(A_lSB)_b$ codes

```
import numpy as np
from itertools import product
# Define error sets
def e(b, l, q):
    e_b_l_power = np.power(2, np.arange(0,b+1-l))%q
    e_b_l = (e_b_l_power*(2**l-1))%q
    return np.unique(e_b_l)

def E(b,l, q):
    E_b_l = np.array([], dtype=np.int64)

    for i in range(0, l+1):
        E_b_l = np.append(E_b_l, e(b, i, q))

    return np.delete(np.unique(E_b_l), [0])

def U(r, b, q):
  u_element = np.array([-1])

  if r > 0:
```

```
#Generating possible values of U_r matrix
     u_set = np.array(list(product(u_element, repeat=r-1)))
     power_vector = np.power(2, np.arange(b-r+1, b))


     U_results = u_set.dot(power_vector) - np.power(2, b-r)


     return (U_results%q).item()


  else:
     raise ValueError("r␣must␣be␣greater␣than␣0")


def V(s, b, q):
  v_element = np.array([-1])


  if s > 0:
#Generating possible values of V_s matrix
    v_set = np.array(list(product(v_element, repeat=s-1)))


    power_vector = np.power(2, np.arange(s-1))


    V_results = v_set.dot(power_vector) - np.power(2, s-1)


    return (V_results%q).item()


   else:
    raise ValueError("s␣must␣be␣greater␣than␣0")
def C_i_C_j_generate (C, b, l, q):
    epsilon = (E(b,l, q))%q
    C_epsilon = np.empty((0,epsilon.size), np.int64)
    C_allowed = np.array([], np.int64)
```

```
    C_epsilon = np.append(C_epsilon, np.array([epsilon]), axis=0)
    C_allowed = np.append(C_allowed, -1)

    for i in C:
        C_epsilon_i = (-epsilon*i)%q
        non_unique_set = [np.intersect1d(C_i_epsilon, C_epsilon_i).
            ↪ size != 0 \
                        for C_i_epsilon in C_epsilon]

        is_set = (C_epsilon_i.size == np.unique(C_epsilon_i).size)
        is_unique = sum(non_unique_set) == 0

        if is_set & is_unique:
            C_epsilon = np.append(C_epsilon, np.array([C_epsilon_i]),
                ↪ axis=0)
            C_allowed = np.append(C_allowed, i)
    if C_allowed.size >1:
      C_epsilon = np.append(C_epsilon[1:], np.array([C_epsilon[0]]),
          ↪ axis=0)
      C_allowed = np.append(C_allowed[1:], C_allowed[0])
    return C_allowed, C_epsilon
def U_V_C_i_C_i1(C_i, C_i1, epsilon_bl, l, b, q):
  C_iP_C_i1Q = np.array([], np.int64)

  C_i_epsilon = (-C_i*epsilon_bl)%q
  C_i1_epsilon = (-C_i1*epsilon_bl)%q

  for r in np.arange(1, l):
    for s in np.arange(1, l+1-r):
      C_i_U_rs = C_i*U(r,b,q)
      C_j_V_ss = C_i1*V(s, b, q)
```

```
    C_i_C_j_add = (C_i_U_rs + C_j_V_ss)%q

    if (C_i_C_j_add in C_i_epsilon) or C_i_C_j_add in C_i1_epsilon:
      return False, C_iP_C_i1Q
    else:
      C_iP_C_i1Q = np.append(C_iP_C_i1Q, C_i_C_j_add)

  return True, C_iP_C_i1Q

def C_i_generate (C, b, l, q):
  U_V_C_i_union_set = np.array([], np.int64)

  epsilon_bl = E(b,l, q)%q
  Cij_probable, Cij_epsilon = C_i_C_j_generate(C, b, l, q)
  C_test_array = epsilon_bl.copy()

  Ci_not_allowed = np.array([], np.int64)
  C_i_index, C_j_index = Cij_probable.size - 2, Cij_probable.size - 1

  while C_i_index >= 0:
    C_i = Cij_probable[C_i_index]
    C_j = Cij_probable[C_j_index]

    U_V_C_i_test, U_V_C_i_array = U_V_C_i_C_i1(C_i, C_j, epsilon_bl,
      ↪ l, b, q)

    if (not U_V_C_i_test) or (np.intersect1d(C_test_array,
      ↪ U_V_C_i_array).size != 0):
      Ci_not_allowed = np.append(Ci_not_allowed, C_i)
      C_i_index = C_i_index - 1
```

```
    elif (np.intersect1d(C_test_array, Cij_epsilon[C_i_index]).size
        ↪ != 0):
      Ci_not_allowed = np.append(Ci_not_allowed, C_i)
      C_i_index = C_i_index - 1

    else:
      #print("\n", C_i, C_test_array, U_V_C_i_array, np.sort(
          ↪ Cij_epsilon[C_i_index]), "\n")
      C_test_array = np.union1d(C_test_array, U_V_C_i_array)
      C_test_array = np.union1d(C_test_array, Cij_epsilon[C_i_index])
      #print("\n", C_i, C_test_array, U_V_C_i_array, np.sort(
          ↪ Cij_epsilon[C_i_index]), "\n")
      C_j_index = C_i_index
      C_i_index = C_i_index - 1

  Cij_probable = np.setdiff1d(Cij_probable, Ci_not_allowed)
  print(Cij_probable)
  return Cij_probable
  return Ci_not_allowed

b = 9
l = 3
q = 2**b-1
C = range(2,q)
C_i_generate (C, b, l, q)
```

Appendix F

Python program for integer $(B_l AEC)_b$ codes

```
import numpy as np
from itertools import product
# Define error sets
def e(b, l, q):
    e_b_l_power = np.power(2, np.arange(0,b+1-l))%q
    e_b_l = np.array([], dtype=np.int64)
    for i in np.arange(1, 2**l, 2):
     e_b_l = np.append(e_b_l, ((e_b_l_power*i)%q))

    return np.unique(e_b_l)
def E(b,l, q):
    E_b_l = np.array([], dtype=np.int64)

    for i in range(0, l+1):
      #print(e(b, i, q))
      E_b_l = np.append(E_b_l, e(b, i, q))

    return np.unique(E_b_l)
#print(E(8,2,255))

def P(r, b, q):
```

```
    p_element = np.array([-1, 0])

    if r > 0:
#Generating possible values of U_r matrix
        p_set = np.array(list(product(p_element, repeat=r-1)))
#Generating Vector for 2^0 to 2^{r-2}
        power_vector = np.power(2, np.arange(r-1))

        P_results = p_set.dot(power_vector) - np.power(2, r-1)

        return np.array(np.unique(P_results%q),int)

    else:
        raise ValueError("r must be greater than 0")
#print(P(1,8,255))
def Q(s, b, q):
    q_element = np.array([-1, 0])

    if s > 0:
        #Generating possible values of V_r matrix
        q_set = np.array(list(product(q_element, repeat=s-1)))
        #Generating Vector for 2^(b-s+1) to 2^{b}
        power_vector = np.power(2, np.arange(b-s+1, b))

        Q_results = q_set.dot(power_vector) - np.power(2, b-s)

        return np.array(np.unique(Q_results%q), int)

    else:
        raise ValueError("s must be greater than 0")
#print(Q(1,8,255))
```

```
def C_i_C_j_generate (C, b, l, q):
    epsilon = (E(b,l, q))%q
    C_epsilon = np.empty((0,epsilon.size), np.int64)
    C_allowed = np.array([], np.int64)

    C_epsilon = np.append(C_epsilon, np.array([epsilon]), axis=0)
    C_allowed = np.append(C_allowed, -1)

    for i in C:
        C_epsilon_i = (-epsilon*i)%q
        non_unique_set = [np.intersect1d(C_i_epsilon, C_epsilon_i).
            ↪ size != 0 \
                        for C_i_epsilon in C_epsilon]

        is_set = (C_epsilon_i.size == np.unique(C_epsilon_i).size)
        is_unique = sum(non_unique_set) == 0

        if is_set & is_unique:
            C_epsilon = np.append(C_epsilon, np.array([C_epsilon_i]),
                ↪ axis=0)
            C_allowed = np.append(C_allowed, i)

    if C_allowed.size >1:
      C_epsilon = np.append(C_epsilon[1:], np.array([C_epsilon[0]]),
          ↪ axis=0)
      C_allowed = np.append(C_allowed[1:], C_allowed[0])
    return C_allowed, C_epsilon
def all_out_combo_addition(C_i, C_i1, P, Q):
  C_i_P = C_i*P
  C_j_Q = C_i1*Q
```

```
  cartesian = np.array(np.meshgrid(C_i_P, C_j_Q)).T.reshape(-1, 2)
  collapse = []
  for l in cartesian:
    collapse.append(l[0]+l[1])
  return np.array(collapse)

def P_Q_C_i_C_i1(C_i, C_i1, epsilon_bl, l, b, q):
  C_iP_C_i1Q = np.array([], np.int64)

  C_i_epsilon = (-C_i*epsilon_bl)%q
  C_i1_epsilon = (-C_i1*epsilon_bl)%q

  for r in np.arange(1, l):
    for s in np.arange(1, l+1-r):
      C_i_C_j_add = (all_out_combo_addition(C_i, C_i1, P(r,b,q), Q(s,
          ↪ b,q)))%q

      if (C_i_C_j_add in C_i_epsilon) or (C_i_C_j_add in C_i1_epsilon
          ↪ ):
        return False, C_iP_C_i1Q
      else:
        C_iP_C_i1Q = np.append(C_iP_C_i1Q, C_i_C_j_add)

  return True, C_iP_C_i1Q
def C_i_generate (C, b, l, q):
  P_Q_C_i_union_set = np.array([], np.int64)

  epsilon_bl = E(b,l, q)%q
  Cij_probable, Cij_epsilon = C_i_C_j_generate(C, b, l, q)
  C_test_array = epsilon_bl.copy()
```

```
Ci_not_allowed = np.array([], np.int64)
C_i_index, C_j_index = Cij_probable.size - 2, Cij_probable.size - 1

while C_i_index >= 0:
 C_i = Cij_probable[C_i_index]
 C_j = Cij_probable[C_j_index]

 P_Q_C_i_test, P_Q_C_i_array = P_Q_C_i_C_i1(C_i, C_j, epsilon_bl, l, b, q)



 if (not P_Q_C_i_test) or (np.intersect1d(C_test_array, P_Q_C_i_array).size != 0):
   Ci_not_allowed = np.append(Ci_not_allowed, C_i)
   C_i_index = C_i_index - 1

 elif (np.intersect1d(C_test_array, Cij_epsilon[C_i_index]).size != 0):
  Ci_not_allowed = np.append(Ci_not_allowed, C_i)
  C_i_index = C_i_index - 1

 else:
  #print("\n", C_i, C_test_array, U_V_C_i_array, np.sort(Cij_epsilon[C_i_index]), "\n")
  C_test_array = np.union1d(C_test_array, P_Q_C_i_array)
  C_test_array = np.union1d(C_test_array, Cij_epsilon[C_i_index])
  #print("\n", C_i, C_test_array, U_V_C_i_array, np.sort(Cij_epsilon[C_i_index]), "\n")
  C_j_index = C_i_index
  C_i_index = C_i_index - 1
```

```
  Cij_probable = np.setdiff1d(Cij_probable, Ci_not_allowed)
  print(Cij_probable)
  return Cij_probable
  return Ci_not_allowed
b = 8
l = 2
q = 2**b-1
C = range(2,q)
C_i_generate (C, b, l, q)
```

Bibliography

[1] Abramson, N. and Elspas, B. Double-error-correcting encoders and decoders for non-independent binary errors. In *International Conference on Information Processing, Paris*, 1959.

[2] Abramson, N. M. *Information Theory and Coding.* McGraw-Hill Book Company Inc., New York, 1963.

[3] Alexander, A., Gryb, R., and Nast, D. Capabilities of the telephone network for data transmission. *Bell System Technical Journal*, 39(3):431–476, 1960.

[4] Anderson, D. A. and Metze, G. Design of totally self-checking check circuits for m-out-of-n codes. *IEEE Transactions on Computers*, 100(3):263–269, 1973.

[5] Arlat, J. and Carter, W. C. Implementation and evaluation of a (b, k)-adjacent error-correcting/detecting scheme for supercomputer systems. *IBM Journal of Research and Development*, 28(2):159–169, 1984.

[6] Assmus (Jr), E. and Mattson (Jr), H. Coding and combinatorics. *Siam Review*, 16(3):349–388, 1974.

[7] Assmus (Jr), E., Mattson (Jr), H., and Turyn, R. Research to develop the algebraic theory of codes. Technical report, Sylvania Electronic Systems-East Waltham Ma Applied Research Lab, 1967.

[8] Baumann, R. Soft errors in advanced computer systems. *IEEE Design & Test of Computers*, 22(3):258–266, 2005.

[9] Baylis, J. *Error-Correcting Codes; A Mathematical Introduction.* Chapman and Hall, London, 1998.

[10] Belov, B. I. A conjecture on the Griesmer bound. *Optimization Methods and Their Applications*, 182:100–106, 1972.

[11] Berardi, L., Dass, B. K., and Sobha, G. High–density burst error-correcting linear codes. *Tamsui Oxford Journal of Mathematical Sciences*, 19(2):177–187, 2003.

[12] Berlekamp, E. R. Nonbinary BCH decoding. *IEEE International Symposuim on Information Theory*, San Remo, Italy, 1967.

[13] Berlekamp, E. R. *Algebraic Coding Theory.* McGraw Hill Book Company, New York, 1968.

[14] Berlekamp, E. R., Peile, R. E., and Pope, S. P. The application of error control to communications. *IEEE Communications Magazine*, 25(4):44–57, 1987.

[15] Blake, I. F. Codes over certain rings. *Information and Control*, 20(4):396–404, 1972.

[16] Blake, I. F. *Algebraic Coding Theory, History and Development.* Dowden, Hustchinson and Ross, Stroudsburg, Pa, 1973.

[17] Blake, I. F. Codes over integer residue rings. *Information and Control*, 29(4): 295–300, 1975.

[18] Blake, I. F. and Mullin, R. C. *The Mathematical Theory of Coding.* Academic Press, New York, 1975.

[19] Bose, B. and Al-Bassam, S. Byte unidirectional error correcting and detecting codes. *IEEE Transactions on Computers*, 41(12):1601–1606, 1992.

[20] Bose, R. C. *Information Theory, Coding and Cryptography.* The McGraw-Hill, India, 2016.

[21] Bose, R. C. and Ray-Chaudhuri, D. K. On a class of error correcting binary group codes. *Information and Control*, 3(1):68–79, 1960.

[22] Buccimazza, B., Dass, B. K., and Jain, S. High-density-burst error detection. *Journal of Discrete Mathematical Sciences and Cryptography*, 7(1):5–21, 2004.

[23] Burton, D. *Elementary Number Theory.* McGraw Hill, 2010.

[24] Campopiano, C. N. Bounds on burst-error-correcting codes. *IRE Transactions on Information Theory*, 8(3):257–259, 1962.

[25] Chien, R. T. and Tang, D. T. On definitions of a burst. *IBM Journal of Research and Development*, 9(4):292–293, 1965.

[26] Clark, G. C. and Cain, J. B. *Error-Correction Coding for Digital Communications*. Plenum Press, New York, 1981.

[27] Das, P. K. A class of solid burst error correcting codes derived from a reversible code. *University Politechnica of Bucharest Sceintific Bulletin-Series A-Applied Mathematics and Physics*, 80(4):153–162, 2018.

[28] Dass, B. K. On a burst-error correcting code. *Journal of Information and Optimization Sciences*, 1(3):291–295, 1980.

[29] Dass, B. K. and Das, P. K. On perfect-like binary and non-binary linear codes-a brief survey. *Bulletin of the Malaysian Mathematical Sciences Society*, 32 (2):187–210, 2009.

[30] Dass, B. K. and Tyagi, V. Bounds on blockwise burst error correcting linear codes. *Information Sciences*, 20(3):157–164, 1980.

[31] Dass, B. K., Eugeni, F., and Innamorati, S. Low-density burst error-locating/correcting linear codes. *Journal of Information and Optimization Sciences*, 16(3):533–548, 1995.

[32] Dass, B. K., Sobha, G., and Zannetti, M. High-density CT-burst error-locating linear codes. *Journal of Interdisciplinary Mathematics*, 5(3):243–249, 2002.

[33] Elias, P. Coding for noisy channels. *IRE Convention Record*, 3:37–47, 1955.

[34] Fire, P. A class of multiple-error-correction binary codes for non-independent errors. *Sylvania Report RSL-E-2, Sylvania Reconnaissance Systems Laboratory, Mountain View, California*, 1959.

[35] Forney, G. D. *Concatenated Codes*. Cambridge, Massachusetts. The M.I.T. Press, 1966.

[36] Gallagher, R. G. *Information Theory and Reliable Communication.* Wiley, New York, 1968.

[37] Gilbert, E. N. A comparison of signalling alphabets. *Bell System Technical Journal*, 31(3):504–522, 1952.

[38] Golay, M. J. E. Notes on digital coding. *Proceedings of the IRE*, 37(1):657, 1949.

[39] Griesmer, J. H. A bound for error-correcting codes. *IBM Journal of Research and Development*, 4(5):532–542, 1960.

[40] Hamming, R. W. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160, 1950.

[41] Hartley, R. V. L. Transmission of information. *The Bell System Technical Journal*, 7(3):535–563, 1928.

[42] Hill, R. *A First Course in Coding Theory.* Oxford University Press, 1986.

[43] Hocquenghem, A. Codes correcteurs d'erreurs. *Chiffres*, 2:147–156, 1959.

[44] Huang, Z., Wang, X., Chen, X., and Kan, H. Network coding with interleaving. In *2007 International Conference on Parallel Processing Workshops (ICPPW 2007)*. IEEE, 2007.

[45] Huffman, W. C. and Pless, V. *Fundamentals of Error-Correcting Codes.* Cambridge University Press, Cambridge, 2003.

[46] Jensen, D. Block code to efficiently correct adjacent data and/or check bit errors, Patent No: US 6604222 B1, Date of Patent- Aug 5, 2003.

[47] Johnson, S. A new upper bound for error-correcting codes. *IRE Transactions on Information Theory*, 8(3):203–207, 1962.

[48] Johnsson, L. Introduction to HPC architecture. *Department of Computer Sciences, University of Houston, Houston, TX, USA*, 2014.

[49] Justesen, J. and Hoholdt, T. *A course in error-correcting codes.* EMS Textbooks in Mathematics. European Mathematical Society (EMS), Zurich, 2004.

[50] Kasami, T., Lin, S., and Peterson, W. W. On the minimum weight of BCH codes. *Journal of The Institute of Electronics and Communication Engineers of Japan*, 50:1617–1622, 1967.

[51] Kautz, W. and Levitt, K. A survey of progress in coding theory in the soviet union. *IEEE Transactions on Information Theory*, 15(1):197–244, 1969.

[52] Kim, Y., Daly, R., Kim, J., Fallin, C., Lee, J. H., Lee, D., Wilkerson, C., Lai, K., and Mutlu, O. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. *ACM SIGARCH Computer Architecture News*, 42(3):361–372, 2014.

[53] Klove, T. *Codes for Error Detection, Series on Coding Theory and Cryptology.* World Scientific Publishing Company Private Limited, 2007.

[54] Kostadinov, H., Morita, H., and Manev, N. Integer codes correcting single errors of specific types $(\pm e_1, \pm e_2, \ldots, \pm e_s)$. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 86(7):1843–1849, 2003.

[55] Kostadinov, H., Morita, H., and Manev, N. On (± 1) error correctable integer codes. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 93A(12):2758–2761, 2010.

[56] Lee, L. H. C. *Error-Control Block Codes for Communication Engineers.* Artech House, USA. 2000.

[57] Levenshtein, V. I. and Vinck, A. H. Perfect (d, k)-codes capable of correcting single peak-shifts. *IEEE Transactions on Information Theory*, 39(2):656–662, 1993.

[58] Lin, S. and Costello, D. J. *Error Control Coding-Fundamentals and Applications.* Prentice-Hall of India Private Limited, New Delhi, 1983.

[59] Ling, S. and Xing, C. *Coding Theory: A First Course.* Cambridge University Press, 2004.

[60] MacWilliams, F. J. and Sloane, N. J. A. *The Theory of Error-Correcting Codes.* Elsevier-North-Holland, Amsterdam, 1983.

[61] McEliece, R. J. *The Theory of Information and Coding.* Addison-Wesley Publishing Co., Reading Mass. London, Amsterdam, 1977.

[62] Meggit, J. E. Error correcting codes and their implementation for data transmission systems. *IRE Transactions on Information Theory*, 7(4):234–244, 1961.

[63] Mehlhorn, K. and Sanders, P. *Algorithms and Data Structures: The Basic Toolbox.* Springer, 2008.

[64] Morelos-Zaragoza, R. H. *The Art of Error-Correcting Coding.* John Wiley and Sons Limited, 2002.

[65] Muller, D. E. Application of boolean algebra to switching circuit design and to error detection. *Transactions of the I.R.E. Professional Group on Electronic Computers*, EC-3(3):6–12, 1954.

[66] Neubauer, A., Freudenberger, J., and Kuhn, V. *Coding Theory: Algorithms, Architectures and Applications.* John Wiley & Sons Ltd, England, 2007.

[67] Nyquist, H. Certain factors affecting telegraph speed. *The Bell System Technical Journal*, 3(2):324–346, 1924.

[68] Park, S. and Bose, B. Burst asymmetric/unidirectional error correcting/detecting codes. In *Digest of Papers. Fault-Tolerant Computing: 20th International Symposium.* IEEE Computer Society, 1990.

[69] Peterson, W. W. *Error-Correcting Codes.* First Edition. M.I.T. Press, Cambridge, Massachusetts, 1961.

[70] Peterson, W. W. and Weldon(Jr), E. J. *Error-Correcting Codes.* 2nd Edition. M.I.T. Press, Cambridge, Massachusetts, 1972.

[71] Pless, V. *Introduction to the Theory of Error-Correcting Codes.* 3rd Edition. John Wiley and Sons, 1998.

[72] Plotkin, M. Binary codes with specified minimum distance. *IRE Transactions on Information Theory*, IT-6(4):445–450, 1960.

[73] Poli, A., Huguet, L., and Craig, I. *Error Correcting Codes: Theory and Applications.* Prentice Hall, 1992.

[74] Pradhan, D. K. and Stiffler, J. J. Error-correcting codes and self-checking circuits. *IEEE Computer*, 13(03):27–37, 1980.

[75] Prange, E. *Cyclic error-correcting codes in two symbols.* Air force Cambridge research center, 1957.

[76] Radaelli, D., Puchner, H., Wong, S., and Daniel, S. Investigation of multi-bit upsets in a 150 nm technology sram device. *IEEE Transactions on Nuclear Science*, 52(6):2433–2437, 2005.

[77] Radonjic, A. (Perfect) integer codes correcting single errors. *IEEE Communications Letters*, 22(1):17–20, 2017.

[78] Radonjic, A. Integer codes correcting double errors and triple-adjacent errors within a byte. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 28(8):1901–1908, 2020.

[79] Radonjic, A. Integer codes correcting single errors and detecting burst errors within a byte. *IEEE Transactions on Device and Materials Reliability*, 20(4): 748–753, 2020.

[80] Radonjic, A. Integer codes correcting single asymmetric errors. *Annals of Telecommunications*, 76(1):109–113, 2021.

[81] Radonjic, A. and Vujicic, V. Integer codes correcting burst errors within a byte. *IEEE Transactions on Computers*, 62(2):411–415, 2013.

[82] Radonjic, A. and Vujicic, V. Integer codes correcting spotty byte asymmetric errors. *IEEE Communications Letters*, 20(12):2338–2341, 2016.

[83] Radonjic, A. and Vujicic, V. Integer codes correcting high-density byte asymmetric errors. *IEEE Communications Letters*, 21(4):694–697, 2016.

[84] Radonjic, A. and Vujicic, V. Integer codes correcting single errors and burst asymmetric errors within a byte. *Information Processing Letters*, 121:45–50, 2017.

[85] Radonjic, A. and Vujicic, V. Integer codes correcting burst and random asymmetric errors within a byte. *Journal of the Franklin Institute*, 355(2):981–996, 2018.

[86] Radonjic, A. and Vujicic, V. Integer codes correcting sparse byte errors. *Cryptography and Communications*, 11(5):1069–1077, 2019.

[87] Radonjic, A. and Vujicic, V. Integer codes correcting burst asymmetric within a byte and double asymmetric errors. *Cryptography and Communications*, 12 (2):221–230, 2020.

[88] Radonjic, A. and Vujicic, V. Integer codes correcting burst asymmetric errors within a byte. *IETE Journal of Research*, 68(1):176–182, 2022.

[89] Radonjic, A., Bala, K., and Vujicic, V. Integer codes correcting double asymmetric errors. *IET Communications*, 10(14):1691–1696, 2016.

[90] Ramaswami, R., Sivarajan, K., and Sasaki, G. *Optical Networks:A Practical Perspective*. Morgan Kaufmann, 2009.

[91] Reed, I. S. A class of multiple-error-correcting codes and the decoding scheme. *Transactions of the IRE Professional Group on Information Theory*, 4(4):38–49, 1954.

[92] Reed, I. S. and Solomon, G. Polynomial codes over certain finite fields. *SIAM Journal on Applied Mathematics*, 8(2):300–304, 1960.

[93] Reiger, S. Codes for correction of clustered errors. *IRE Transactions on Information Theory*, 6(1):16–21, 1960.

[94] Rhee, M. Y. *Error-Correcting Coding Theory.* McGraw-Hill. 1989.

[95] Sacks, G. E. Multiple error correction by means of parity-checks. *IRE Transactions on Information Theory*, 4(4):145–147, 1958.

[96] Saitoh, Y. and Imai, H. Constructions of codes correcting burst asymmetric errors. In *International Symposium on Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes*. Springer, 1990.

[97] Saitoh, Y. and Imai, H. Some classes of burst asymmetric or undirectional error correcting codes. *Electronics Letters*, 5(26):286–287, 1990.

[98] Shannon, C. E. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[99] Sharma, B. D. and Dass, B. K. Adjacent-error correcting binary perfect codes. *Cybernetics and System*, 7(1-2):9–13, 1977.

[100] Shiva, S. and Sheng, C. Multiple solid burst-error-correcting binary codes (Corresp.). *IEEE Transactions on Information Theory*, 15(1):188–189, 1969.

[101] Shui, Y., Meng, L., Wanchun, D., Xiting, L., and Sanming, Z. Networking for big data: A survey. *IEEE Communications Surveys & Tutorials*, 19(1): 531–549, 2017.

[102] Singleton, R. C. Maximum distance q-ary codes. *IEEE Transactions on Information Theory*, 10(2):116–118, 1964.

[103] Slepian, D. A class of binary signaling alphabets. *The Bell System Technical Journal*, 35(1):203–234, 1956.

[104] Solomon, G. and Stiffler, J. J. Algebraically punctured cyclic codes. *Information and Control*, 8(2):170–179, 1965.

[105] Spiegel, E. Codes over $\mathbb{Z}_m$. *Information and Control*, 35(1):48–51, 1977.

[106] Strocks, R. Number theoretical codes. Technical report, University of Essen, 1997.

[107] Valenti, M. C. The evolution of error control coding. *AuthorâĂŹs Ph. D. dissertation: Iterative detection and decoding for wireless communications,*

Bradley Department of Electrical & Computer Engineering, Virginia Tech, 1999.

[108] Van Lint, J. H. *Coding Theory*. Springer, Berlin, Germany, 1971.

[109] Van Lint, J. H. A survey of perfect codes. *The Rocky Mountain Journal of Mathematics*, 5(2):199–224, 1975.

[110] Van Lint, J. H. *Introduction to Coding Theory*. 3rd Edition. Springer-Verlag, Heidelberg, Second Indian Reprint, 2005.

[111] Vanstone, S. A. and Oorschot, P. C. V. *An Introduction to Error Correcting Codes with Applications*. The Springer International Series in Engineering and Computer Science, 1989.

[112] Varshamov, R. One asymmetrical error-correcting codes. *Avtomatika i Telemehanika*, 26(2):288–292, 1965.

[113] Varshamov, R. P. Estimate of the number of signals in error-correcting codes. *Doklady Akademii Nauk*, 117:739–741, 1957.

[114] Vermani, L. R. *Elements of Algebraic Coding*. Chapman and Hall, 1996.

[115] Vinck, A. J. H. and Morita, H. Codes over the ring of integers modulo m. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 81(10):2013–2018, 1998.

[116] Wolf, J. A survey of coding theory. *IEEE Transactions on Information Theory*, 19(4):1967–1972, 1973.

[117] Wyner, A. Low-density-burst-correcting codes (Corresp.). *IEEE Transactions on Information Theory*, 9(2):124–124, 1963.

⋆ ⋆ ⋆ ⋆ ⋆

List of Publications and Conferences

Published/Accepted

1. Pokhrel, N.K. and Das, P.K. Unidirectional solid burst correcting integer codes. *Journal of Applied Mathematics and Computing*. doi:10.1007/s12190-021-01662-2, 2021. (SCI Expanded, UGC-CARE List, Group II).

2. Pokhrel, N.K. and Das, P.K. Low-density and high-density asymmetric CT-burst correcting integer codes. *Advances in Mathematics of Communications*. doi:10.3934/amc.2022030, 2022. (SCI Expanded, UGC-CARE List, Group II).

3. Pokhrel, N.K., Das, P.K. and Radonjic, A. Integer codes capable of correcting burst asymmetric errors. *Journal of Applied Mathematics and Computing*. doi:10.1007/s12190-022-01770-7, 2022. (SCI Expanded, UGC-CARE List, Group II).

4. Pokhrel, N.K. and Das, P.K. Probability of erroneous decoding for integer codes correcting burst asymmetric/unidirectional/symmetric errors within a byte and up to double asymmetric errors between two bytes. *Kuwait Journal of Science*. doi: 10.48129/kjs.online, 2022. (SCI Expanded, UGC-CARE List, Group II).

5. Das, P.K. and Pokhrel, N.K. Asymmetric CT-burst correcting integer codes. In 5^{th} *International Conference on Information Systems and Computer Networks (ISCON), IEEE*. doi:10.1109/ISCON52037.2021.9702506, 2021. (Scopus, UGC-CARE List, Group II).

Submitted

1. Das, P. K. and Pokhrel, N. K. Asymmetric solid burst correcting integer codes. *Submitted for publication*.

Presentation at Conferences

1. Pokhrel, N. K. Integer codes correcting simultaneous bursts. In *International Conference on Advanced Mathematical Analysis and its Applications*. Department of Mathematics, Berhampur University, Berhampur, Odhisa, India, February, 2020.

2. Pokhrel, N. K. Integer codes correcting asymmetric bursts connected between two adjoining bytes. In 1^{st} *National Conference on Application of Mathematical Tools in Social Sciences and Sciences*. Department of Mathematics, Zakir Husain Delhi College, University of Delhi, Delhi, October, 2020.