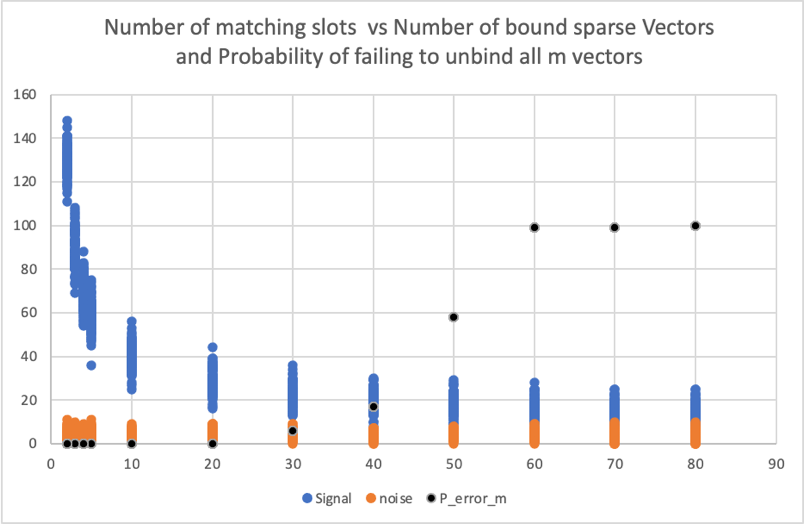
Sparse Vector Binding

# Binding Scheme









# Theory

B is the number of bit positions

m is the number of sparse vectors to be added (including the target vector)

D is a parameter that defines if we want to determine the probability of a winning pattern (D=0) or a pattern where there is the probability of a draw D=1)

We define the Pattern as arrays of elements *E* and counts *C* of the sum before adding the target pattern.

:

If D = 0, W = 1

If D = 1 and , W =

If

Here are some examples:

Example 1

Pattern = 2 0 1 0 0 3 0 2 0 1 0 1 0 0 1 0

Target = 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Sum = 3 0 1 0 0 3 0 2 0 1 0 1 0 0 1 0

m = 12

B = 16

*E* = [0,1,2,3]

C = [1,4,2,1]

Example 2

Pattern = 2 1 2 1 1 2

Target = 1 0 0 0 0 0

Sum = 3 1 2 1 1 2

m = 10

B = 6

E = [1,2]

C = [3,3]

Number of possible patterns combinations

Number of arrangements

If D = 0,

If D = 1,

Denote to indicate this and then:

Number of repeats

Where is the number of combinations of b in a. Alternatively we can write:

Pattern Probability

or





Code in Patterns6\_1.py

from \_\_future\_\_ import generators

import numpy as np

import math

from scipy.special import factorial

from scipy.misc import comb

from scipy.special import perm

def calc\_prob(elements,counts,B,D):

E = np.array(elements)

C = np.array(counts)

l = len(C) - 1

EC = E\*C

Pn = num\_patterns(E,C)

if D == 0:

C[l] -= 1

W = 1

else:

C[l-1] -= 1

W = counts[l] + 1

An = num\_arrangements(C)

Rn = num\_repeats(B,C)

P = Pn\*An\*Rn/(W \* np.float\_power(B,m-1))

return P

def num\_patterns(E,C):

F = factorial(E)

Fp = np.power(F,C)

Pn = factorial(m-1)/np.product(Fp)

return Pn

def num\_arrangements(C):

F = factorial(C)

An = factorial(sum(C))/np.product(F)

return An

def num\_repeats(B,C):

Rn = comb(B-1,sum(C))

return Rn

def accel\_asc(n):

a = [0 for i in range(n + 1)]

k = 1

y = n - 1

while k != 0:

x = a[k - 1] + 1

k -= 1

while 2 \* x <= y:

a[k] = x

y -= x

k += 1

l = k + 1

while x <= y:

a[k] = x

a[l] = y

yield a[:k + 2]

x += 1

y -= 1

a[k] = x + y

y = x + y - 1

yield a[:k + 1]

def get\_result(m,B,M):

cnt=0

p\_cnt=0

w\_cnt=0

d\_cnt=0

P\_sum = 0

T = 1.0

for value in accel\_asc(m-1):

a = np.array(value)[::-1]

if len(a) <=B:

p\_cnt +=1

elements, counts = np.asarray(np.unique(a,return\_counts=True))

E = [0]

C = [0]

for i in range(0,len(elements)):

E.append(elements[i])

C.append(counts[i])

D = 0

P = calc\_prob(E,C,B,D)

P\_sum += P

w\_cnt += 1

if P>=T:

print (a,P,P\_sum,'W')

#If can be shared win divide by i\_a

l = len(E) - 1

if l>=1 and E[l] - E[l-1] ==1:

p\_cnt += 1

D = 1

if E[l] == 1:

C[0] = 1

P = calc\_prob(E,C,B,D)

P\_sum += P

d\_cnt += 1

if P>=T:

print (a,P,P\_sum,'D')

print(P\_sum\*M)

'''

print(m,B,M,'sum=',P\_sum,P\_sum\*M)

print('Pattern Count=',p\_cnt)

print('win count=',w\_cnt)

print('draw\_cnt=', d\_cnt)

'''

B = 512

M = 256

print("Num\_slots=",M,"Num\_bits=",B)

for m in range(2,82,2):

get\_result(m,B,M)

# Results











