Information Theory HW3

許博翔

November 10, 2023

1 General Convex Optimization Solver

I used cvxpy as the general convex optimization solver.

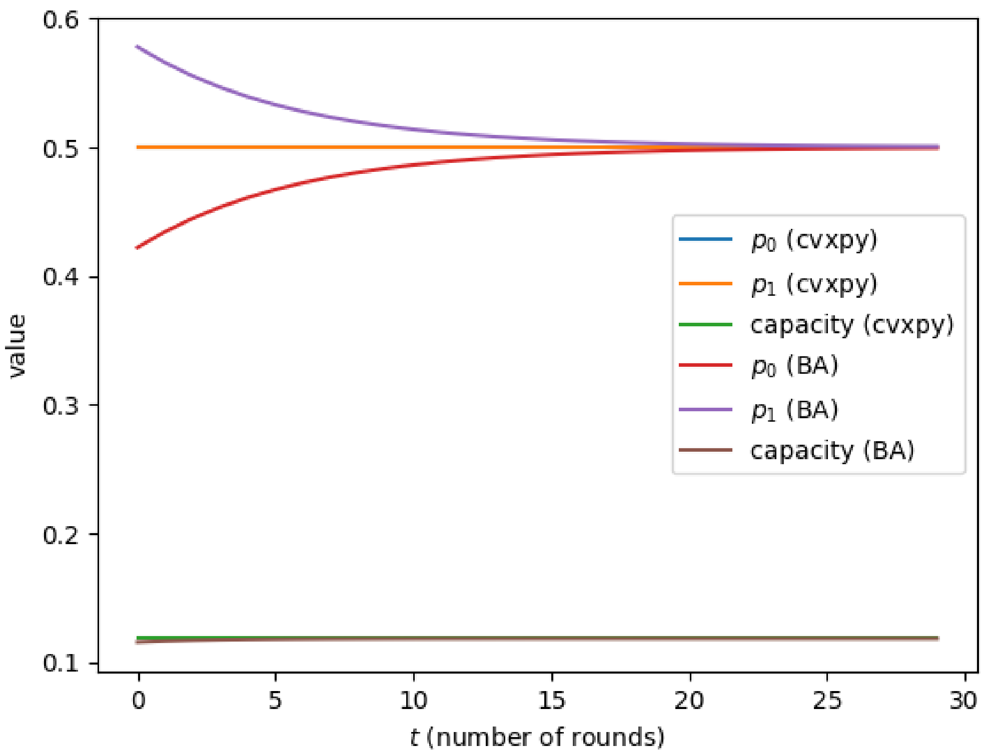
2 Result

Note: the result of cvxpy on the graph is that counted by cvxpy directly, which is independent of the number of rounds implemented in the Blahut-Arimoto algorithm.

2.1 Symmetric Channel

2.1.1 Binary

$P_{Y|X} = \begin{pmatrix} 1-p & p \\ p & 1-p \end{pmatrix}$ where $p = 0.3$.

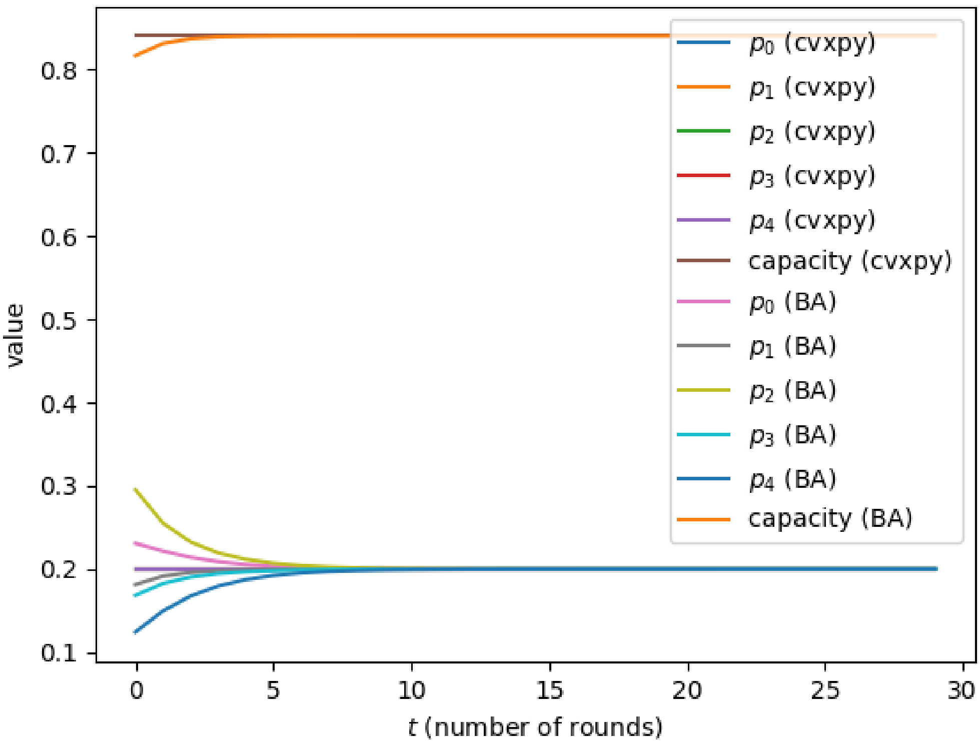


Result: uniform distribution.

Capacity: ≈ 0.119.

2.1.2 Complicated

$$P_{Y|X} = \begin{pmatrix} 1-p & p/4 & p/4 & p/4 & p/4 \\ p/4 & 1-p & p/4 & p/4 & p/4 \\ p/4 & p/4 & 1-p & p/4 & p/4 \\ p/4 & p/4 & p/4 & 1-p & p/4 \\ p/4 & p/4 & p/4 & p/4 & 1-p \end{pmatrix} \text{ where } p = 0.3.$$



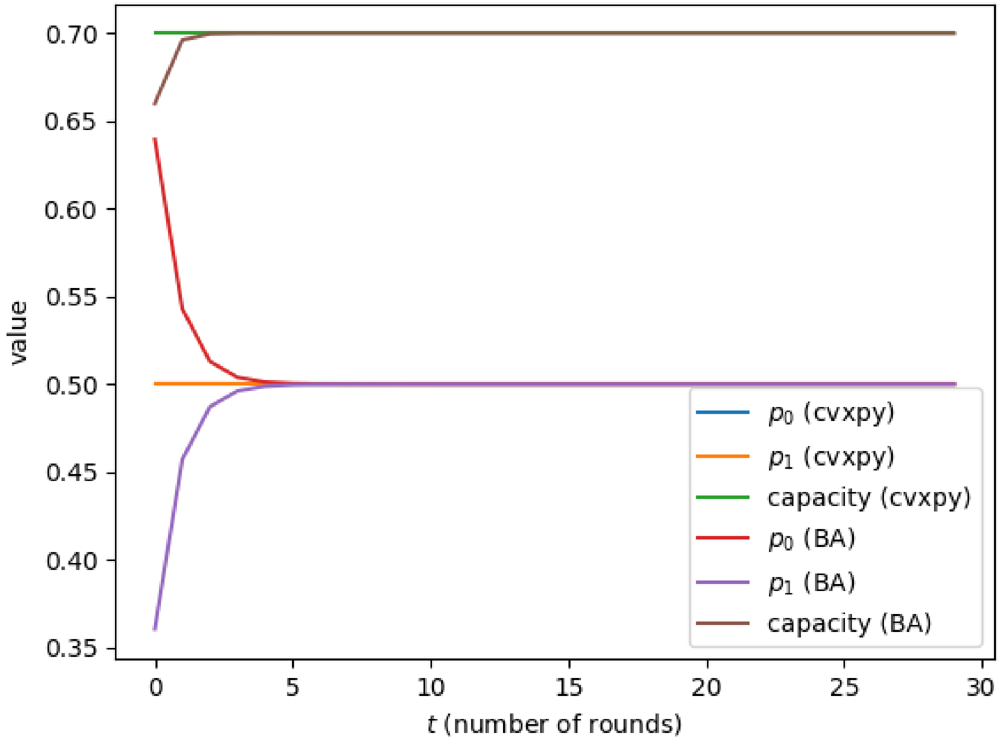
Result: uniform distribution.

Capacity: ≈ 0.841.

2.2 Erasure Channel

2.2.1 Binary

$P_{Y|X} = \begin{pmatrix} 1-p & p & 0 \\ 0 & p & 1-p \end{pmatrix}$ where $p = 0.3$.

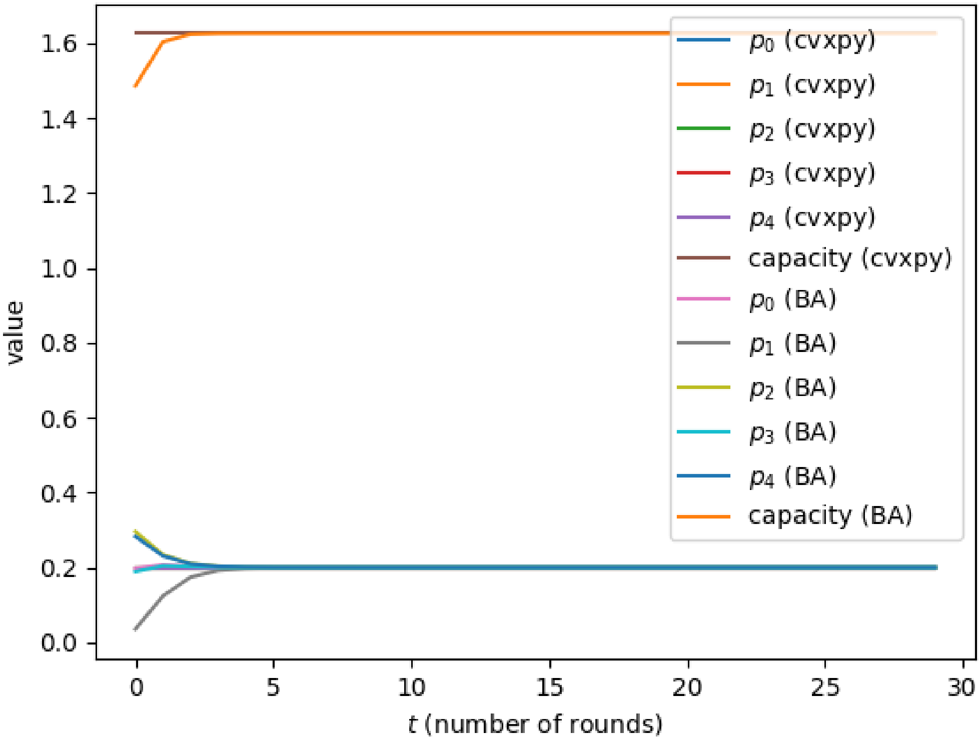


Result: uniform distribution.

Capacity: 0.7.

2.2.2 Complicated

$$P_{Y|X} = \begin{pmatrix} 1-p & 0 & 0 & 0 & 0 & p \\ 0 & 1-p & 0 & 0 & 0 & p \\ 0 & 0 & 1-p & 0 & 0 & p \\ 0 & 0 & 0 & 1-p & 0 & p \\ 0 & 0 & 0 & 0 & 1-p & p \end{pmatrix} \text{ where } p = 0.3.$$



Result: uniform distribution.

Capacity: ≈ 1.625.

3 Source Code

The following is my source code, where the `BA()` function is to compute the capacity by the Blahut-Arimoto algorithm, while the `general()` function is to compute using cvxpy.

```
import numpy as np
```

```
import cvxpy as cp
from random import *
from scipy.special import xlogy
from numpy import log
import matplotlib.pyplot as plt

T=30

def BA(P):
        n, m=len(P), len(P[0])
        p, q=[], []
        xx, yy=[], [[] for i in range(n+1)]
        for t in range(T):
                pd=[]
                if t==0:
                        pd=[random() for i in range(n)]
                else:
                        pd=[1]*n
                        for i in range(n):
                                for j in range(m):
                                        pd[i]*=q[j][i]**P[i
                                            ][j]
                sm=sum(pd)
                p=[i/sm for i in pd]
                xx.append(t)
                for i in range(n):
                        yy[i].append(p[i])
                q=[]
                for j in range(m):
                        pd=[p[k]*P[k][j] for k in range(n)]
                        sm=sum(pd)
```

```
            q.append([i/sm for i in pd])
        C=sum([sum([xlogy(p[i]*P[i][j]/log(2), q[j][i]/p[i]) for j in range(m)]) for i in range(n)])
        yy[n].append(C)
    print(p)
    print(C)
    for i in range(n):
        plt.plot(xx, yy[i], label='$p_'+str(i)+'$ (BA)')
    plt.plot(xx, yy[n], label='capacity (BA)')
    plt.xlabel('$t$ (number of rounds)')
    plt.ylabel('value')

def general(P):
    n, m=len(P), len(P[0])
    p=cp.Variable(shape=n)
    q=P@p
    C=cp.sum(cp.entr(q)/log(2))+cp.sum([p[i]*sum([xlogy(P[i][j]/log(2), P[i][j]) for j in range(m)]) for i in range(n)])
    prob=cp.Problem(cp.Maximize(C), [cp.sum(p)==1, p>=0])
    prob.solve()
    print(p.value)
    print(prob.value)
    xx, yy=[], [[] for i in range(n+1)]
    for t in range(T):
        xx.append(t)
        for i in range(n):
            yy[i].append(p.value[i])
```

```
                yy[n].append(prob.value)
        for i in range(n):
                plt.plot(xx, yy[i], label='$p_'+str(i)+'$ (
                   cvxpy)')
        plt.plot(xx, yy[n], label='capacity (cvxpy)')

seed(77777144949)

p=0.3
P=[[1-p, p], [p, 1-p]]
general(P)
BA(P)
plt.legend(loc='best')
plt.savefig('symmetric.png')
plt.show()

n, m=5, 5
p=0.3
P=[[1-p if i==j else p/(n-1) for j in range(m)] for i in
   range(n)]
print(np.array(P))
general(P)
BA(P)
plt.legend(loc='best')
plt.savefig('symmetric2.png')
plt.show()

p=0.3
P=[[1-p, p, 0], [0, p, 1-p]]
general(P)
BA(P)
```

```
plt.legend(loc='best')
plt.savefig('erasure.png')
plt.show()

n, m=5, 6
p=0.3
P=[[1-p if j==i else p if j==m-1 else 0 for j in range(m)]
   for i in range(n)]
print(np.array(P))
general(P)
BA(P)
plt.legend(loc='best')
plt.savefig('erasure2.png')
plt.show()
```