# Generalizations of the Blahut-Arimoto Algorithm *

**Sagnik Bhattacharya, Priyanka Kaswan, Adway Patra**
Department of Electrical and Computer Engineering, UMD
{sagnikb, pkaswan, apatra}@umd.edu

December 2020

**Abstract**

We study the Blahut-Arimoto algorithm, an alternating minimization algorithm useful for various convex optimization problems, and its applications in classical and quantum information theory. We present a convergence proof of the classical Blahut-Arimoto algorithm due to Csiszar and Tusnady, and then implement generalizations of the classical algorithm to finite state channels [Kav01, VKAL08] and quantum channels [RISB20]. The implementations of the quantum Blahut-Arimoto algorithms in MATLAB and Python have been made open-source and uploaded to Github.

## 1 Introduction

The Kullback-Liebler divergence $D(p \mid\mid q)$ between two pmf's $p$ and $q$ over the alphabet $\mathcal{X}$ is given by

$$D(p \parallel q) \coloneqq \sum_{x \in \mathcal{X}} p(x) \log \left( \frac{p(x)}{q(x)} \right).$$

If there is a symbol $x \in \mathcal{X}$ such that $q(x) = 0$ but $p(x) > 0$, then $D(p \parallel q) = \infty$. We also follow the convention that $0 \log(0/0) = 0$, and the log is taken with base 2.

Several fundamental questions in information theory, like the computation of channel capacities and rate-distortion functions can be phrased in the following form - given two convex compact subsets $\mathcal{P}$ and $\mathcal{Q}$ of the probability simplex, find

$$d_{\min} \coloneqq \inf_{p \in \mathcal{P}, q \in \mathcal{Q}} D(p \parallel q)$$

Since $D(p \parallel q)$ is convex in the pair $(p, q)$ and the sets $\mathcal{P}$ and $\mathcal{Q}$ are convex, this is a convex optimization problem. However these problems do not have an analytic solution except in the simplest of cases and we need algorithms to efficiently find an approximate numerical solution. The Blahut-Arimoto algorithm [Bla72, Ari72] is one such algorithm. Using the convexity, it is easy to show that for each $q \in \mathcal{Q}$, there exists $p^*(q) \in \mathcal{P}$ such that $p^*(q) = \min_{p \in \mathcal{P}} D(p \parallel q)$. Likewise, for each $p \in \mathcal{P}$, there exists $q^*(p) \in \mathcal{Q}$ such that $q^*(p) = \min_{q \in \mathcal{Q}} D(p \parallel q)$. The algorithm itself is the following simple iterative alternative minimization procedure

$$\begin{aligned} p_n &= p^*(q_{n-1}) \\ q_n &= q^*(p_n) \end{aligned} \qquad n = 1, 2, \ldots \tag{1}$$

*Report for ENEE662 (Convex Optimization), Fall 2020

The classical BAA has been widely used to optimize a discrete memoryless source (DMS) at the input of a discrete memoryless channel (DMC). The problem, as shown in a series of landmark papers by Shannon [Sha48], entails finding the optimal input distribution of a DMS over a finite alphabet that maximizes the mutual information between the input and output of a DMC that is characterized by a prespecified transition probability matrix. However the classical algorithm cannot handle more general channel models like channels with memory and quantum channels, and so we need more general versions of the basic algorithm.

### Structure of the report

In this report we first (section 2) present a proof of convergence for the Blahut-Arimoto algorithm, due to [CT84]. We then look at two important generalizations of the basic Blahut-Arimoto algorithm. In section 3 we see how it can be modified to handle finite-state channels, which are the simplest models for channels with memory, due to [Kav01, VKAL08]. In section 4 we see a generalization that can handle various quantities of interest in quantum information theory, due to [RISB20].

### Implementations

We implement a simple version of the finite state channel algorithm due to [VKAL08] in MATLAB, and all the algorithms in [RISB20]. The implementations of the quantum Blahut-Arimoto algorithms are in both MATLAB and Python and have been uploaded to Github repositories at `https://github.com/priyankakaswan18/Quantum-Blahut-Arimoto-Algorithm` and `https://github.com/sagnikb/quantum-blahut-arimoto`

## 2 The convergence proof

In this section we shall show that the algorithm in (1) converges to $d_{\min}$, and this is the content of theorem 5. To do this, we need to define an auxiliary function $\delta(p, p')$ as follows

$$\delta(p, p') = \sum_{x \in \mathcal{X}} \left[ p(x) \log \left( \frac{p(x)}{p'(x)} \right) - (p(x) - p'(x)) \log(e) \right]$$

and the proof works for any distance function $f(s, t)$ associated auxiliary function $\delta_f(s, s')$ that satisfies the following propositions. Here we prove that these propositions hold for the KL divergence, but the same proof works for other algorithms of a similar nature, like the expectation-maximization algorithm used in statistical modelling to find MAP and ML estimates.

We begin by noting some basic properties of the KL divergence $D(p \parallel q)$ - it is non-negative, convex in the pair $(p, q)$, and is zero if and only if $p = q$. It is not, however, symmetric and is therefore not a true metric. Now we see some properties of the function $\delta$ that are used later on.

**Proposition 1.** *The function $\delta(p, p')$ is non-negative and $\delta(p, p') = 0$ iff $p = p'$.*

*Proof.* We have the inequality $\ln(t) \leq t - 1 \Rightarrow \log(t) \leq \log(e)(t - 1)$, with equality iff $t = 1$. Now let $t = p'(x)/p(x)$; we obtain

$$\log \left( \frac{p'(x)}{p(x)} \right) \leq \log(e) \left( \frac{p'(x)}{p(x)} - 1 \right); \tag{2}$$

multiplying through by $p(x)$ and summing over $x \in \mathcal{X}$ we obtain the result. □

**Proposition 2** (Three Points Property)**.** *For $p \in \mathcal{P}$ and $q \in \mathcal{Q}$, we have that*

$$D(p^*(q) \parallel q) + \delta(p, p^*(q)) \leq D(p \parallel q)$$

*Proof.* We need to show that

$$\sum_{x \in \mathcal{X}} \left[ p^*(q)(x) \log \frac{p^*(q)(x)}{q(x)} + p(x) \log \frac{p(x)}{p^*(q)(x)} - (p(x) + p^*(q)(x)) \log e \right] \leq \sum_{x \in \mathcal{X}} p(x) \log \frac{p(x)}{q(x)} \tag{3}$$

$$\Leftrightarrow \sum_{x \in \mathcal{X}} \left[ p(x) \log \frac{p^*(q)(x)}{q(x)} - p^*(q)(x) \log \frac{p^*(q)(x)}{q(x)} - (p(x) - p^*(q)(x)) \log e \right] \geq 0 \tag{4}$$

Fix $p \in \mathcal{P}$ and $q \in \mathcal{Q}$. For any $0 \leq \alpha \leq 1$ let $p_\alpha = (1-\alpha)p^*(q) + \alpha p$. The convexity of $\mathcal{P}$ implies that $p_\alpha \in \mathcal{P}$. Using the properties of information projections (projections under the geometry induced by the KL divergence) we have, for some $0 < \tilde{\alpha} < \alpha$,

$$0 \leq \frac{1}{\alpha} \left[ D(p_\alpha \parallel q) - D(p^*(q) \parallel q) \right] = \frac{d}{d\alpha} D(p_\alpha \parallel p) \Big|_{\alpha = \tilde{\alpha}}$$

Also, $p_\alpha \big|_{\alpha=0} = p^*(q)$. Taking $\alpha \downarrow 0$, we get

$$0 \leq \lim_{\tilde{\alpha} \downarrow 0} \left( \frac{d}{d\alpha} D(p_\alpha \parallel p) \Big|_{\alpha = \tilde{\alpha}} \right) \tag{5}$$

$$= \lim_{\tilde{\alpha} \downarrow 0} \sum_{x \in \mathcal{X}} \left[ (p(x) - p^*(q)(x)) \log \frac{p_{\tilde{\alpha}}(x)}{q(x)} + (p(x) - p^*(q)(x)) \log e \right] \tag{6}$$

$$\Rightarrow 0 \leq \sum_{x \in \mathcal{X}} \left[ p(x) \log \frac{p^*(q)(x)}{q(x)} - p^*(q)(x) \log \frac{p^*(q)(x)}{q(x)} + (p(x) - p^*(q)(x)) \log e \right] \tag{7}$$

which is what we wanted to show. □

**Proposition 3** (Four points property)**.** *For $p \in \mathcal{P}$ with $\min_{q \in \mathcal{Q}} D(p \parallel q) < \infty$ and for all $p \in \mathcal{P}$, $q \in \mathcal{Q}$, we have*

$$D(p' \parallel q') + \delta(p', p) \geq D(p' \parallel q^*(p))$$

*Proof.* We need to show that

$$\sum_{x \in \mathcal{X}} \left[ p'(x) \log \frac{q^*(p)(x)}{q'(x)} + p'(x) \log \frac{p'(x)}{p(x)} - (p'(x) - p(x)) \log e \right] \geq 0 \tag{8}$$

$$\Leftrightarrow \sum_{x \in \mathcal{X}} \left[ p'(x) \log \frac{p'(x) q^*(p)(x)}{q'(x) p(x)} - (p'(x) - p(x)) \log e \right] \geq 0 \tag{9}$$

which we get after writing out $D$ and $\delta$ using the definitions. A standard inequality in information theory states that

$$\ln(1/t) \leq 1/t - 1 \Rightarrow \log t \geq (1 - 1/t) \log e.$$

Using this we get that

$$\sum_{x \in \mathcal{X}} \left[ p'(x) \log \frac{p'(x) q^*(p)(x)}{q'(x) p(x)} - (p'(x) - p(x)) \log e \right] \tag{10}$$

$$\geq \sum_{x\in\mathcal{X}} \left[ p'(x) \left( 1 - \frac{q'(x)p(x)}{p'(x)q^*(p)(x)} \right) \log e - (p'(x) - p(x)) \log e \right] \tag{11}$$

$$\geq \sum_{x\in\mathcal{X}} \left[ \left( p'(x) - \frac{q'(x)p(x)}{q^*(p)(x)} \right) \right] \log e - \sum_{x\in X} \left[ (p'(x) - p(x)) \right] \log e \tag{12}$$

$$= \sum_{x\in\mathcal{X}} p(x) \frac{q^*(p)(x) - q'(x)}{q^*(p)(x)} \log e \tag{13}$$

and we need to show that this is $\geq 0$. Now, for an arbitrary $q' \in \mathcal{Q}$ and for $0 \leq \alpha \leq 1$, let $q_\alpha = (1-\alpha)q^*(p) + \alpha q'$. The convexity of $\mathcal{Q}$ implies that $q_\alpha \in \mathcal{Q}$. Again using the properties of information projections we have, for some $0 < \tilde{\alpha} < \alpha$,

$$0 \leq \frac{1}{\alpha} \left[ D(p \parallel q_\alpha) - D(p \parallel q^*(p)) \right] = \frac{d}{d\alpha} D(p \parallel q_\alpha) \Big|_{\alpha=\tilde{\alpha}}$$

We know that $q_\alpha\big|_{\alpha=0} = q^*(p)$. Taking $\alpha \downarrow 0$ we get that

$$0 \leq \lim_{\tilde{\alpha}\downarrow 0} \frac{d}{d\alpha} D(p \parallel q_\alpha) \Big|_{\alpha=\tilde{\alpha}} = \lim_{\tilde{\alpha}\downarrow 0} \sum_{x\in\mathcal{X}} p(x) \frac{q^*(p)(x) - q'(x)}{(1-\tilde{\alpha})q^*(p)(x) + \tilde{\alpha}q'(x)} \log e \tag{14}$$

$$\Rightarrow 0 \leq \sum_{x\in\mathcal{X}} p(x) \left[ \frac{q^*(p)(x) - q'(x)}{q^*(p)(x)} \right] \log e \tag{15}$$

which shows that the quantity in (13) is non-negative, completing the proof. □

**Proposition 4.** *For $q \in \mathcal{Q}$ with $\min_{p\in\mathcal{P}} D(p \parallel q) = D(p^*(q) \parallel q) < \infty$, we have $\delta(p^*(q), p_1) < \infty$, where $p_1 = p^*(q_0)$.*

*Proof.* First, $D(p^*(q) \parallel q) < \infty$ implies that $\text{supp}(p^*(q)) \subseteq \text{supp}(q)$. Next, in proposition 2, picking $q = q_0$ and $p = p^*(q)$ gives

$$D(p^*(q_0) \parallel q_0) + \delta(p^*(q), p^*(q_0)) \leq D(p^*(q) \parallel q_0)$$

This means that $D(p^*(q) \parallel q_0)$ being finite will imply the requirement of the proposition, and this happens if $\text{supp}(p^*(q)) \subseteq \text{supp}(q_0)$. So if we have $\text{supp}(q) \subseteq \text{supp}(q_0)$, then we will have $\text{supp}(p^*(q) \subseteq \text{supp}(q) \subseteq \text{supp}(q_0)$. This condition is always met when $\text{supp}(q_0) = \text{supp}(\mathcal{Q})$. □

**Theorem 5** (Main convergence result)**.** *The iterative procedure* (1)*, with initial point $q_0$ satisfying* $\text{supp}(q_0) = \text{supp}(\mathcal{Q})$ *produces a sequence $\{p_n, q_n\}$ such that*

$$\lim_n D(p_n \parallel q_n) = \inf_{p\in\mathcal{P}, q\in\mathcal{Q}} D(p \parallel q) = d_{\min} \tag{16}$$

*Proof.* By proposition 2 we have $D(p_{n+1} \parallel q_n) + \delta(p, p_{n+1}) \leq D(p \parallel q_n)$ and by proposition 3 we have $D(p \parallel q_n) \leq D(p \parallel q) + \delta(p, p_n)$ for any $p \in \mathcal{P}$ and $q \in \mathcal{Q}$. Adding the two together, we get that $D(q_{n+1} \parallel p_n) + \delta(q, q_{n+1}) \leq D(p \parallel q) + \delta(p, p_n)$ or

$$\delta(p, p_{n+1}) \leq D(p \parallel q) - D(p_{n+1} \parallel q_n) + \delta(p, p_n) \tag{17}$$

for all $p \in \mathcal{P}$ and $q \in \mathcal{Q}$. Now, from the iteration in (1) we get that

$$D(p_n \parallel q_n) \geq D(p_{n+1} \parallel q_n) \geq D(p_{n+1} \parallel q_{n+1}) \geq D(p_{n+2} \parallel q_{n+1})$$

Assume that the limit in (16) does not exist. Then there exists $q \in \mathcal{Q}$ and $\epsilon > 0$ such that

$$D(p_{n+1} \parallel q_n) > D(p^*(q) \parallel q) + \epsilon \qquad n = 1, 2, \ldots$$

Applying (17) with this choice of $p^*(q)$ and $q$ we get that

$$\delta(p^*(q), p_{n+1}) \leq D(p^*(q) \parallel q) - D(p_{n+1} \parallel q_n) + \delta(p^*(q), p_n) \tag{18}$$

$$\Rightarrow \delta(p^*(q), p_{n+1}) \leq \delta(p^*(q), p_n) - \epsilon \qquad n = 1, 2, \ldots \tag{19}$$

which contradicts proposition 4 and the non-negativity of $\delta$ established in proposition 1, because by proposition 4 the starting value of $\delta$ is finite and decreases by $\epsilon$ after every iteration, and eventually becomes negative. □

# 3 Blahut-Arimoto Algorithms for channel capacity

In this section, we begin by describing an equivalent form of the classical Blahut-Arimoto algorithm that generalizes well to finite state channels, and show how it is used to calculate the capacity of a discrete memoryless channel (DMC). We eventually show how this algorithm generalizes to finite state channels which have memory.

## 3.1 Classical BAA

We consider a DMC with input alphabet $\mathcal{X}$, output alphabet $\mathcal{Y}$ and transition probability matrix

$$W(y|x) = P_{Y|X}(y|x) \quad \forall x \in \mathcal{X}, y \in \mathcal{Y} \tag{20}$$

An input probability distribution $Q(x)$ over $\mathcal{X}$ induces an ouput distribution $R(y)$ over $\mathcal{Y}$ given by $R(y) = (QW)(y) = \sum_x Q(x)W(y|x)$. The mutual information between the channel input and output random variables $X$ and $Y$ is given by

$$I(Q; W) = I(X; Y) = H(X) - H(X|Y) = \sum_x \sum_y Q(x)W(y \,|\, x) \log \frac{V(x \,|\, y)}{Q(x)} \tag{21}$$

where $V(x|y) = \frac{Q(x)W(y|x)}{R(y)}$ is the aposteriori probability of $X = x$ given $Y = y$ is observed. The capacity $C$ of the DMC is found by maximizing this mutual information expression over all possible input pmfs, i.e.,

$$C = \max_{Q \in \mathcal{Q}} I(Q; W) \tag{22}$$

where $\mathcal{Q} = \{Q : \mathcal{X} \to \mathbb{R} \,|\, Q(x) \geq 0 \text{ for all } x \in \mathcal{X}, \sum_x Q(x) = 1\}$ is the set of valid probability distributions. The problem is simplified by the fact that $I(Q, W)$ is a concave function of $Q$ and hence has a unique maximum, although there might be multiple input pmfs that reach this maxima.

**Proposition 6.** *For a fixed transition probability matrix $W$, $I(Q; W)$ is a concave function of $Q$.*

The classical BAA can be formulated as an iterative algorithm with some initial starting point $Q^{<0>}$. Assuming the algorithm has run upto an iteration $r$, with the quantity $I(Q^{<r>}, W)$ calculated, in the next iteration a new pmf $Q^{<r+1>}$ needs to be found such that $I(Q^{<r+1>}, W) \geq I(Q^{<r>}, W)$. This is acccomplished by introducing a surrogate function $\Psi(Q^{<r>}, Q, W)$ which has the following properties

1. At $Q = Q^{<r>}$, we have $\Psi(Q^{<r>}, Q^{<r>}, W) = I(Q^{<r>}, W)$.

2. For all $Q \in \mathcal{Q}$, $\Psi(Q^{<r>}, Q, W) \leq I(Q; W)$.

3. The maximization of $\Psi(Q^{<r>}, Q, W)$ is easy such that we can easily find

$$Q^{<r+1>} = \arg\max_{Q \in \mathcal{Q}} \Psi(Q^{<r>}, Q, W) \tag{23}$$

To find such a surrogate function, we introduce the following quantity.

**Definition 1.** *For a fixed DMC $W(y|x)$, given that the input pmf is $Q(x)$, define*

$$T_{\tilde{Q}}(x) = \sum_y W(y|x) \log \frac{\tilde{Q}(x)W(y|x)}{\sum_x \tilde{Q}(x)W(y|x)} = \sum_y W(y|x) \log \tilde{V}(x|y) \tag{24}$$

With this definition, we write

$$I(Q; W) = \sum_x Q(x) \left[ \log \left( \frac{1}{Q(x)} \right) + T(x) \right] \tag{25}$$

and let the surrogate function be

$$\Psi(\tilde{Q}, Q, W) = \sum_x Q(x) \left[ \log \left( \frac{1}{Q(x)} \right) + T_{\tilde{Q}}(x) \right] \tag{26}$$

It can be easily checked that the surrogate function satisfies all the three properties listed above. We now give the classical Blahut-Arimoto algorthm for DMCs.

**Algorithm 1** Blahut-Arimoto algorithm for DMC

1: Inputs:
- Input alphabet $\mathcal{X}$, Output alphabet $\mathcal{Y}$
- Initial guess $Q^{<0>} \in \mathcal{Q}$
- Channel Transition matrix $W(\cdot|\cdot)$
- Number of iteration steps $n$

2: **for** $r \in \{1, 2, \ldots, n\}$ **do**

3: For each $x \in \mathcal{X}$, calculate $T_{\tilde{Q}}(x) = \sum_y W(y|x) \log \frac{\tilde{Q}(x)W(y|x)}{\sum_x \tilde{Q}(x)W(y|x)}$ with $\tilde{Q} = Q^{<r-1>}$

4: Calculate $Q^{<r>} = \arg\max_{Q \in \mathcal{Q}} \Psi(\tilde{Q}, Q, W)$

5: **end for**

6: Outputs: Maximizing input distribution $Q^{<n>}(x)$, channel capacity $I(Q^{<n>}, W) = \Psi(Q^{<n>}, Q^{<n>}, W) = 0$

**Theorem 7.** *For each $r \in \{1, 2, \cdots, n\}$ the sequence of input probability distributions $Q^{<r>}$ produced by the classical BAA fulfills*

$$I(Q^{<r>}, W) \geq I(Q^{<r-1>}, W) \tag{27}$$

*Furthermore, $Q^{<r>}$ converges to a capacity achieving input distribution as $r \to \infty$.*

*Proof.* The proof is the classical result from [Ari72],[Bla72]. □

### 3.2 BAA for Finite State Machine Channels

Although DMCs provide a basic understanding of the fundamental problem of noisy information transmission, most real life scenarios are much more complicated because they do not abide by the strong independence assumption of a DMC. In this regard, finite state machine channels (FSMC) provide the much needed generalization by introducing memory into the model. However, the problem of calculating the capacity of FSMCs is found to be much more challenging and closed form expressions are, more often than not, elusive. In [VKAL08], the classical BAA was extended to the case of FSMCs to calculate the capacities in a numerical way.

**Definition 2.** *A time-invariant (discrete-time) Finite State Machine Source (FSMS) has a state sequence $\cdots, S_{-1}, S_0, S_1, \cdots$ and an output sequence $\cdots, X_{-1}, X_0, X_1, \cdots$ where $S_l \in \mathcal{S}$ and $X_l \in \mathcal{X}$ for all $l \in \mathbb{Z}$. The sets $\mathcal{S}$ and $\mathcal{X}$ are assumed finite. For any $N > 0$, the joint probability decomposes as*

$$P_{S^N_{-N+1}, X^N_{-N+1}|S_{-N}}(\boldsymbol{s}^N_{-N+1}, \boldsymbol{x}^N_{-N+1}|s_{-N}) = \prod_{l=-N+1}^{N} P_{S_l, X_l|S_{l-1}}(s_l, x_l|s_{l-1}) \tag{28}$$

*where $P_{S_l, X_l|S_{l-1}}(\cdot,\cdot|\cdot)$ is independent of $l$.*

**Example 1.** *A Bernoulli(p) source with $p \in (0,1)$ can be expressed as an FSMS source with $\mathcal{S} = \{0,1\}$ and $X_l = S_l$ for all $l \in \mathbb{Z}$ with transition probabilities $Pr(S_l = 1|S_{l-1} = s_{l-1}) = p = 1 - Pr(S_l = 0|S_{l-1} = s_{l-1})$ for all $s_{l-1} \in \mathcal{S}$.*

**Example 2.** *A $(d,k)$ Run-Length-Limited (RLL) sequence (with $d \leq k$) is defined as a binary sequence where the length of any subsequnce of 0's between any two consecutive 1's is between $d$ and $k$. A source which outputs only RLL $(1,\infty)$ sequences can be expressed as an FSMS by $\mathcal{S} = \{0,1\}$ and $X_l = S_l$ with transition probabilities $P(S_l = 0|S_{l-1} = 1) = 1$.*

**Definition 3.** *A time-invariant (discrete-time) FSMC has an input process $\cdots, X_{-1}, X_0, X_1, \cdots$, an output process $\cdots, Y_{-1}, Y_0, Y_1, \cdots$, and a state process $\cdots, S'_{-1}, S'_0, S'_1, \cdots$, where $X_l \in \mathcal{X}, Y_l \in \mathcal{Y}, S'_l \in \mathcal{S}'$ for all $l \in \mathbb{Z}$. The sets $\mathcal{X}, \mathcal{Y}, \mathcal{S}'$ are assumed finite and for any $N > 0$ the joint pmf decomposes as*

$$P_{S^N_{-N+1}, Y^N_{-N+1}|X^N_{-N+1}S_{-N}}(\boldsymbol{s}^N_{-N+1}, \boldsymbol{y}^N_{-N+1}|\boldsymbol{x}^N_{-N+1}, s_{-N}) = \prod_{l=-N+1}^{N} P_{S_l, Y_l|X_l, S_{l-1}}(s_l, y_l|x_l, s_{l-1}) \tag{29}$$

*where $P_{S_l, Y_l|X_l, S_{l-1}}(\cdot,\cdot|\cdot,\cdot)$ is independent of $l$.*

**Example 3.** *The famous Gilbert-Elliot channel can be described by a two state Markov process with $\mathcal{S}' = \{'b', 'g'\}$ and $\mathcal{X} = \mathcal{Y} = \{0,1\}$ with*

$$P_{S_l, Y_l|X_l, S_{l-1}}(s_l, y_l|x_l, s_{l-1}) = P_{S_l|S_{l-1}}(s_l|s_{l-1}) P_{Y_l|X_l, S_{l-1}}(y_l|x_l, s_{l-1}) \tag{30}$$

*The states vary according to a Markov transition matrix $\begin{bmatrix} 1-p_g & p_g \\ p_b & 1-p_b \end{bmatrix}$ independent of the input and in each state the channel acts as a $BSC(\epsilon)$ where $\epsilon$ depends on the state $s_{l-1}$.*

We intoduce some additional notation for the remainder of the section. Denote by $b_l = (s_{l-1}, s_l)$, $b'_l = (s'_{l-1}, s'_l)$ to be the $l$-th branch in the trellis diagram of the source and the channel states respectively and $b''_l = (b_l, b'_l)$. We shall assume that for any $l$, $b_l$ specifies $x_l$. Additionally, we assume that the channel is indecomposable, i.e., roughly speaking the influence

of the initial state fades out with time for every possible channel input sequence. Additionally, we denote by $Q(s_l, s_{l-1}) \equiv Q(b_l) = Pr(S_l = s_l, S_{l-1} = s_{l-1})$ and $P_{S_l,Y_l|X_l,S_{l-1}}(s_l, y_l|x_l, s_{l-1}) = W(s_l, y_l|x_l, s_{l-1})$ (Notice how $Q$ is defined differently than the previous case). With slight abuse of notation we define the following joint probabilities

$$\begin{aligned}
\mu_i &= \sum_j Q_{ij} && (31)\\
Q(\mathbf{b}) &= \frac{\prod_l Q_{s_l,s_{l-1}}}{\prod_l \mu_{s_l}} && (32)\\
W(\mathbf{y}|\mathbf{b}) &= \sum_{\mathbf{s}'} W(\mathbf{y}, \mathbf{s}'|\mathbf{b} && (33)\\
R(\mathbf{y}) &= (QW)(\mathbf{y}) = \sum_{\mathbf{b}} Q(\mathbf{b})W(\mathbf{y}|\mathbf{b}) && (34)\\
V(\mathbf{b}|\mathbf{y}) &= \frac{Q(\mathbf{b})W(\mathbf{y}|\mathbf{b})}{(QW)(\mathbf{y})} && (35)
\end{aligned}$$

Finally define the manifold $\mathcal{Q} = \{Q : \sum_{i,j} Q_{ij} = 1, Q_{ij} \geq 0, \sum_i Q_{ij} = \sum_i Q_{ji} \ \ \forall\, j\}$.

**Definition 4.** *The $\mathcal{Q}$-constrained capacity of an FSMC is given by*

$$C(\mathcal{Q}, W) = \max_{Q \in \mathcal{Q}} I(Q; W) \tag{36}$$

Our goal will be to find the numerical value of the above expression using the generalized Blahut-Arimoto algorithm. Using the same ideas as the classical case the target is to find a surrogate function that approximates the behavior of the mutual information and satisfies the three properties stated before and then do iterative maximization. It was shown in [VKAL08] that the surrogate function takes the form similar to the classical BAA case, i.e.,

$$\Psi(\tilde{Q}, Q, W) = \sum_{i,j} Q_{ij} \left[ \log\left(\frac{\mu_i}{Q_{ij}}\right) + \tilde{T}_{ij} \right] \tag{37}$$

where $\tilde{T}_{ij} = T_{ij}(\tilde{Q}, W)$ is given by the following relationships:

$$\begin{aligned}
T_{ij} &= \hat{T}_{ij} - \overline{T}_i && (38)\\
\hat{T}_{ij} &= \lim_{N\to\infty} \frac{1}{2N} \sum_{l=-N+1}^{N} \hat{T}_{ij}^N(l) && (39)\\
\overline{T}_i &= \lim_{N\to\infty} \frac{1}{2N} \sum_{l=-N+1}^{N} \overline{T}_i^N && (40)\\
\hat{T}_{ij}^N(l) &= \sum_{\mathbf{b}:b_l=(i,j)} Q(\mathbf{b}|b_l) \sum_{\mathbf{b}''} \sum_{\mathbf{y}} W(\mathbf{b}'', \mathbf{y}|\mathbf{b}) \log \frac{V(b_l''|\mathbf{y})}{V(b_l''|\mathbf{b}'', \mathbf{y})} && (41)\\
\hat{T}_{ij}^N(l) &= \sum_{\mathbf{b}:s_l=i} Q(\mathbf{b}|s_l) \sum_{\mathbf{b}''} \sum_{\mathbf{y}} W(\mathbf{b}'', \mathbf{y}|\mathbf{b}) \log \frac{V(s_l''|\mathbf{y})}{V(s_l''|\mathbf{b}'', \mathbf{y})} && (42)
\end{aligned}$$

*Remark:* Although, the calculations of the above parameters seem cumbersome, a computationally efficient way to calulate these parameters has been given in [VKAL08]. The idea uses large values of $N$ to approximate the parameters by using law of large numbers. A similar idea was used in [Kav01] to calculate the capacity of a BSC under RLL source.

**Algorithm 2** Blahut-Arimoto algorithm for FSMC

1: Inputs:
- Input alphabet $\mathcal{X}$, Output alphabet $\mathcal{Y}$
- Initial guess $Q^{<0>} \in \mathcal{Q}$
- Channel Transition matrix $W$
- Number of iteration steps $n$

2: **for** $r \in \{1, 2, \ldots, n\}$ **do**
3: For each pair $(i, j), i, j \in \mathcal{S}$, calculate $T_{ij}^{<r-1>} = T_{ij}(Q^{<r-1>}, W)$ using the above equations.
4: Calculate $Q^{<r>} = \arg\max_{Q \in \mathcal{Q}} \Psi(\tilde{Q}, Q, W)$
5: **end for**
6: Outputs: Maximizing input distribution $Q^{<n>}$, channel capacity $I(Q^{<n>}, W) = \Psi(Q^{<n>}, Q^{<n>}, W) = 0$

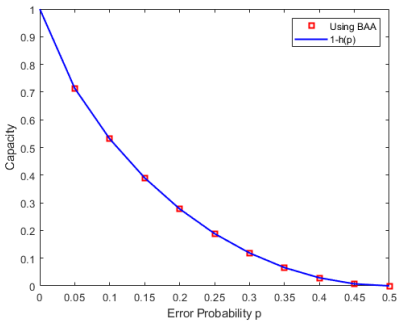


Figure 1: Capacity of Binary Symmetric Channel using classical BAA

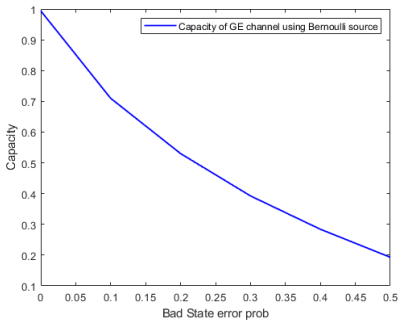


Figure 2: Capacity of Gilbert-Elliot Channel using generalized BAA

### 3.3 Simulations

We simulate both the classical BAA for DMC and the generalized BAA for FSMC using MATLAB. The classical BAA was simulated for the binary symmetric channel (BSC($p$)). For the FSMC, we simulated the Gibert-Elliot channel with the Markov chain state transition probability matrix given by

$$\begin{bmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{bmatrix} \tag{43}$$

and fixed the "good" state transition probability at $\epsilon_g = 0.001$ and vary the "bad" state transition probability. Additionally, we simulate the RLL $(1, \infty)$ binary source at the input of a binary symmetric channel. For this simulation, we use the much simpler algorithm of [Kav01] specialized for FSMS over DMC.

## 4 Quantum Blahut-Arimoto Algorithms

In [RISB20], the authors present algorithms to compute four quantities - the mutual information of quantum channels, the thermodynamic capacity of quantum channels, the coherent information of less noisy quantum channels, and the Holevo quantity of classical-quantum channels. The algorithms proposed are based on the quantum relative entropy which generalizes the KL

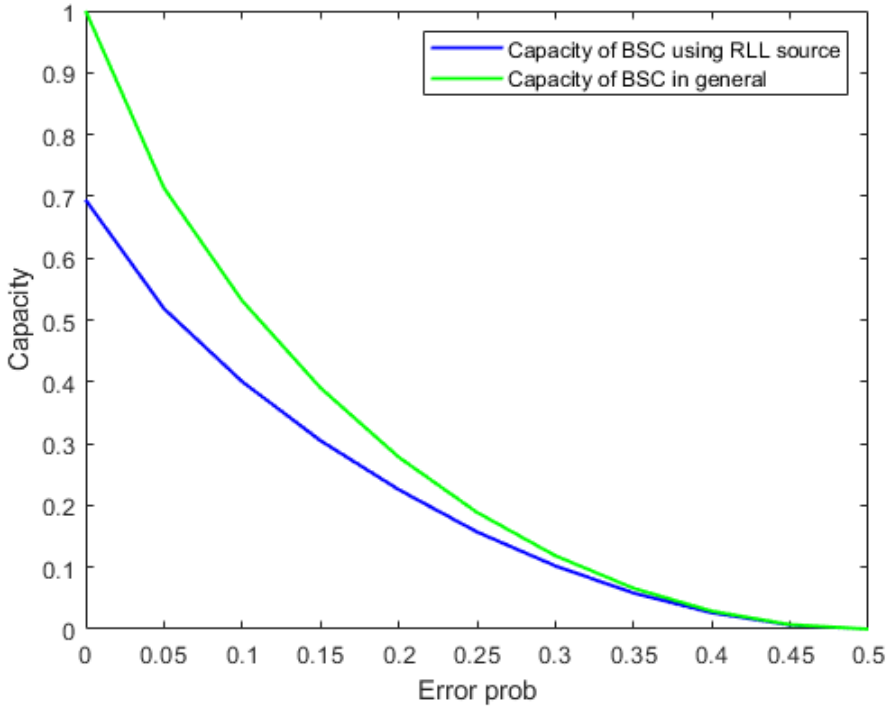


Figure 3: Capacity of Binary Symmetric Channel using classical BAA with RLL $(1, \infty)$ source

divergence and is defined as

$$D(\rho\|\sigma) = \begin{cases} \mathrm{Tr}[\rho(\log\rho - \log\sigma)] & \text{if } \sigma \gg \rho \\ \infty & \text{otherwise} \end{cases} \tag{44}$$

where $\rho$ and $\sigma$ are positive semi-definite matrices and the notation $\sigma \gg \rho$ denotes that the kernel of $\sigma$ is a subset of the kernel of $\rho$ and the (matrix) logarithm is taken on the support of the argument. The relative entropy satisfies certain inequalities under the action of channels, which are crucial for convergence proofs. One such important inequality is the data processing inequality, which generalizes the data processing inequality from classical information theory and which states that for all $\rho, \sigma$ we have $D(\mathcal{E}(\rho)\|\mathcal{E}(\sigma)) \leq D(\rho\|\sigma)$.

## 4.1 Preliminaries and notation

A qubit, also called as quantum bit, is the quantum-mechanical counterpart of the classical bit. Classical bits can store information by taking an value of with zero or one. Qubits are used for storing information in quantum computing. A qubit is a two-level quantum system with two basis qubit states written as $|0\rangle$ and $|1\rangle$, which correspond to following vectors.

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{45}$$

A qubit can be in state $|0\rangle$, $|1\rangle$ or (unlike a classical bit) in a linear combination of these basis states. We call $|k\rangle$ a 'ket', which is a column vector. The adjoint of this vector would be the row

vector $\langle k|$, which is called 'bra'. The scalar $\langle j|k\rangle$ represents the inner product between vectors $|j\rangle$ and $|k\rangle$, while the matrix $|j\rangle\langle k|$ represents the outer product between the two vectors.

- We consider finite dimensional complex Hilbert spaces only, denote them by capital letters $A, B$, etc. and denote their respective dimensions as $|A|$, $|B|$ etc.
- The set of density operators on a system $A$ is the set of all positive semi-definite matrices $\rho_A$ with trace 1, and this set is denoted by $\mathcal{D}(A)$. Each density matrix is associated with a quantum state and vice-versa.
- A quantum channel from system $A$ to system $B$, denoted by $\mathcal{E}_{A\to B} : \mathcal{D}(A) \to \mathcal{D}(B)$ is a linear completely positive trace-preserving (CPTP) map. Associated with every such channel is a set of Kraus operators $\{K_i\}$, with the property that $\sum_i K_i^\dagger K_i = I$, and the action of the channel on some $\rho_A \in \mathcal{D}(A)$ can be written as $\sum_i B_i \rho B_i^\dagger \in \mathcal{D}(B)$.
- Associated with every channel $\mathcal{E}_{A\to B}$ is a complementary channel $\mathcal{E}^c_{A\to E}$. For ease of exposition, $\mathcal{E}$ and $\mathcal{E}_c$ are used instead of $\mathcal{E}_{A\to B}$ and $\mathcal{E}^c_{A\to E}$.
- The Von Neumann entropy generalizes the Shannon entropy and is defined as

$$S(\rho) = -\operatorname{Tr}[\rho \log \rho].$$

- Discrete probability distributions can be expressed as vectors $\lambda = [\lambda_1, \ldots, \lambda_m]$ with $\sum_i \lambda_i = 1$ or (as for inputs to the cq channel) as diagonal matrices with entries $\lambda_1, \ldots, \lambda_m$.

### 4.1.1 Complementary Channel

We show how to obtain Kraus operators for the complementary channel from the Kraus operators $\{A_k : 1 \le k \le n\}$ of the channel[1]. We fix an $n$ dimensional basis for the 'environment' $E$, and define an operator

$$D = \sum_{k=1}^{n} A_k \otimes |k\rangle \tag{46}$$

which is a linear operator mapping $A$ to $B \otimes E$, where $\otimes$ represents tensor product between the matrices. The action of the complementary channel on density matrices can then be represented as (called the Steinspring representation)

$$\mathcal{E}_c(\rho) = \operatorname{Tr}_B \left( D \rho D^\dagger \right) \tag{47}$$

We can simplify this expression by observing that

$$D\rho D^\dagger = \sum_{j=1}^{n} \sum_{k=1}^{n} A_j \rho A_k^\dagger \otimes |j\rangle\langle k| \tag{48}$$

so that

$$\mathcal{E}_c(\rho) = \sum_{j=1}^{n} \sum_{k=1}^{n} \operatorname{Tr}\left( A_j \rho A_k^\dagger \right) |j\rangle\langle k| \tag{49}$$

[1]Method given in `https://quantumcomputing.stackexchange.com/a/5797`

We can use this to obtain the Choi matrix corresponding to $\mathcal{E}_c$, which is defined by

$$\text{Choi}(\mathcal{E}_c) = \sum_{j=1}^{n}\sum_{k=1}^{n} \mathcal{E}_c(|j\rangle\langle k|) \otimes |j\rangle\langle k| \tag{50}$$

Let the eigendecomposition of $\text{Choi}(\mathcal{E}_c)$ be given by $\left\{(\lambda_i, v_i) : 1 \leq i \leq n^2\right\}$. The Kraus operators for the complementary channel can be recovered from the eigendecomposition by turning the $v_i$'s into $n \times n$ matrices $M_i$ and multiplying with the square root of the respective eigenvalue $\sqrt{\lambda_i}$.

### 4.1.2 Adjoint channel

The adjoint of a quantum channel $\mathcal{E}_{A\to B}$, defined by $\mathcal{E}^\dagger_{B\to A}$ satisfies

$$\text{tr}(X\mathcal{E}(\rho)) = \text{tr}\left(\mathcal{E}^\dagger(X)\rho\right) \tag{51}$$

and, for given Kraus operators $\{A_i\}$ of the quantum channel, the adjoint channel's action can be written as

$$\mathcal{E}^\dagger(X) = \sum_i A_i^\dagger X A_i \tag{52}$$

### 4.1.3 Adjoint channel of complementary channel

The action of the adjoint channel of the complementary channel can be found by first finding the Kraus operators of the complementary channel using the method described in section 4.1.1 and then applying the definition given in section 4.1.2.

### 4.1.4 Examples of Channels

**Amplitude damping channel** The amplitude damping channel for $0 \leq p \leq 1$, denoted by $\mathcal{E}_p^{AD}$ acts on qubit systems and has Kraus operators given by

$$A_0 = \begin{pmatrix} 1 & 0 \\ 0 & \sqrt{1-p} \end{pmatrix}, \quad A_1 = \begin{pmatrix} 0 & \sqrt{p} \\ 0 & 0 \end{pmatrix} \tag{53}$$

These satisfy $A_0^\dagger A_0 + A_1^\dagger A_1 = 1$ and are therefore valid Kraus operators. The channel action on a general qubit density matrix would be:

$$\rho = \begin{pmatrix} r & q \\ q^* & 1-r \end{pmatrix} \to \rho' = \begin{pmatrix} p + r(1-p) & q\sqrt{1-p} \\ q^*\sqrt{1-p} & (1-p)(1-r) \end{pmatrix} \tag{54}$$

If $p = 0$ the channel gives the input density matrix as the output without any change, $\rho' = \rho$. Conversely, if $p = 1$ then

$$\rho \to \rho' = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \tag{55}$$

The channel action tries to push the system towards $|0\rangle$, by supressing coherences (the off-diagonal terms), $q \to q\sqrt{1-p}$ and by changing the populations (the diagoinal terms), $r \to p + r(1-p)$. Hence it is called amplitude damping channel. The effect is stronger for larger value of $p$. This is a purely quantum channel with no classical counterpart.

**cq channel** A classical quantum channel can be represented by a set of input-output pairs $\{(x, \tau_x)\}_{x\in\{1,2,\ldots,N\}}$, with $x \in \{1, 2, \ldots, N\}$ as a (classical) input and the quantum states $\tau_x \in \mathcal{D}(B)$ as outputs. For an input distribution vector $\lambda$, the output corresponds to

$$\mathcal{E}(\rho_\lambda) = \sum_x \lambda_x \mathcal{E}(|x\rangle\langle x|) = \sum_x \lambda_x \tau_x \tag{56}$$

where $\lambda_i$ denotes the $i$ -th component of the probability vector $\lambda$ and $\rho_\lambda = \sum_k \lambda_k |k\rangle\langle k|$. This is a generalization of classical discrete memoryless channels, because the channel behaves classically when the $\tau_x$ are all diagonal matrices.

## 4.2 Quantum Blahut-Arimoto Algorithm

For a quantum measure given as a convex optimization problem over input states, we write a two-variable extension function $J$ and then iteratively perform alternate maximization over both variables to numerically compute the original quantity of interest. Algorithm 3 shows the updating structure of Blahut-Arimoto algorithm, given the following conditions on $J, \mathcal{F}_1$ and $\mathcal{F}_2$ are satisfied: For $\gamma > 0$ and density operators $\sigma \gg \rho$

$$J_\gamma(\rho, \sigma) = -\gamma D(\rho\|\sigma) + \mathrm{Tr}[\rho\mathcal{F}(\sigma)] \in \mathbb{R} \tag{57}$$

where $\mathcal{F}$ is a Hermitian matrix which acts as an outer operator to density operators in a way that $\mathrm{Tr}[\rho\mathcal{F}(\sigma)]$ is continuous in $\sigma$ for $\sigma \gg \rho$. The update rules are

$$\mathcal{F}_1(\rho) = \underset{\sigma \text{ with } \sigma\gg\rho}{\arg\max}\ J_\gamma(\rho, \sigma), \quad \mathcal{F}_2(\sigma) = \underset{\rho \text{ with } \sigma\gg\rho}{\arg\max}\ J_\gamma(\rho, \sigma) \tag{58}$$

When quantum Blahut-Arimoto algorithms satisfy the property

$$\mathrm{Tr}[\rho\{\mathcal{F}(\sigma) - \mathcal{F}(\rho)\}] \le \gamma D(\rho\|\sigma) \tag{59}$$

for all density operators $\sigma \gg \rho$, optimizers in (58) have the expressions

$$\mathcal{F}_1(\rho) = \rho \tag{60}$$

$$\mathcal{F}_2(\sigma) = \frac{1}{Z(\sigma)} \exp\left(\log\sigma + \frac{1}{\gamma}\mathcal{F}(\sigma)\right) \tag{61}$$

with normalizing factor $Z(\sigma) = \mathrm{Tr}\left[\exp\left(\log\sigma + \frac{1}{\gamma}\mathcal{F}(\sigma)\right)\right]$.

### 4.2.1 Convergence

Given a strictly positive definite initial state $\rho^{(1)} > 0$ on Hilbert space $A$ and

$$0 \le \mathrm{Tr}[\rho\{\mathcal{F}(\sigma) - \mathcal{F}(\rho)\}] \le \gamma D(\rho\|\sigma) \tag{62}$$

for density operators $\sigma \gg \rho$, we have that $C(n)$ of Algorithm 3 is monotonically increasing and converges for $n \to \infty$ to

$$C^\star = \max_{\rho,\sigma} \mathrm{mith}_{\sigma\gg\rho}\, J_\gamma(\rho, \sigma) \tag{63}$$

with the following bound on approximation error

$$|C^\star - C(n)| \le \frac{\gamma D\left(\rho^\star\|\rho^{(1)}\right)}{n} \tag{64}$$

where $\rho^\star$ is the optimizer (possibly not unique) that achieves the capacity $C^\star$. If $\rho^{(1)}$ is picked to be the maximally mixed state, the error is limited as $|C^\star - C(n)| \le \frac{\gamma \log|A|}{n}$.

**Algorithm 3** Blahut-Arimoto algorithm: Iterative double optimization over density operators

1: Inputs:
- Initial guess $\rho_A^{(1)} \in \mathcal{D}(A)$ with full support, i.e., $\rho_A^{(1)} > 0$
- Function $J_\gamma : \mathcal{D}(A) \times \mathcal{D}(B) \mapsto \mathbb{R}$ with a coefficient $\gamma > 0$
- Update relations $\mathcal{F}_1 : \mathcal{D}(A) \mapsto \mathcal{D}(B)$ and $\mathcal{F}_2 : \mathcal{D}(B) \mapsto \mathcal{D}(A)$
- Number of iteration steps $n$

2: **for** $t \in \{1, 2, \ldots, n\}$ **do**

3: $\quad \sigma_B^{(t)} = \mathcal{F}_1\left(\rho_A^{(t)}\right)$

4: $\quad \rho_A^{(t+1)} = \mathcal{F}_2\left(\sigma_B^{(t)}\right)$

5: **end for**

6: Outputs: $\rho_A^{(n+1)}, C(n) = J_\gamma\left(\rho_A^{(n+1)}, \sigma_B^{(n)}\right)$, where $C(n)$ should approximate $C^\star = \max_{\rho_A, \sigma_B} J_\gamma(\rho_A, \sigma_B)$ for $n \to \infty$ =0

## 4.3 Coherent Information of Less Noisy Channels

For a quantum channel $\mathcal{E}_{A\to B}$ with complementary channel $(\mathcal{E}_c)_{A\to E}$, the coherent information, $I_{\text{coh}}(\mathcal{E})$ is the maximum of the coherent information $I_{\text{coh}}(\rho, \mathcal{E}) = S(\mathcal{E}(\rho)) - S(\mathcal{E}_c(\rho))$ over input states $\rho$.

$$I_{\text{coh}}(\mathcal{E}) = \max_\rho \underbrace{S(\mathcal{E}(\rho)) - S(\mathcal{E}_c(\rho))}_{=I_{\text{coh}}(\rho,\mathcal{E})}. \tag{65}$$

We limit ourselves to only consider less noisy channels. A channel $\mathcal{E}$ is called less noisy when the private capacity of its complementary channel $\mathcal{E}_c$ is zero. This implies that for all density operators $\rho$ and $\sigma$, we have

$$D(\mathcal{E}(\rho)\|\mathcal{E}(\sigma)) \geq D(\mathcal{E}_c(\rho)\|\mathcal{E}_c(\sigma)) \tag{66}$$

### 4.3.1 Blahut-Arimoto algorithm

The coherent information can be estimated using a Blahut-Arimoto algorithm. We define the following two variable extension of $I_{\text{coh}}(\rho, \mathcal{E})$ for $\sigma \gg \rho$.

$$J_\gamma(\rho, \sigma, \mathcal{E}) = I_{\text{coh}}(\rho, \mathcal{E}) + D(\mathcal{E}(\rho)\|\mathcal{E}(\sigma)) - D(\mathcal{E}_c(\rho)\|\mathcal{E}_c(\sigma)) - \gamma D(\rho\|\sigma) \tag{67}$$

After some simplication, $J_\gamma$ can be brought into the following form

$$J_\gamma(\rho, \sigma, \mathcal{E}) = -\gamma \operatorname{Tr}[\rho \log \rho] + \operatorname{Tr}[\rho\{\gamma \log \sigma + \mathcal{F}(\sigma)\}] \tag{68}$$

where $\mathcal{F}(\sigma) = \mathcal{E}_c^\dagger \log \mathcal{E}_c(\sigma) - \mathcal{E}^\dagger \log \mathcal{E}(\sigma)$, and $\gamma = 1$ corresponds to the standard algorithm. Further,

$$\begin{aligned} &\operatorname{Tr}[\rho\{\mathcal{F}(\sigma) - \mathcal{F}(\rho)\}] = D(\mathcal{E}(\rho)\|\mathcal{E}(\sigma)) - D(\mathcal{E}_c(\rho)\|\mathcal{E}_c(\sigma)) \\ &\implies 0 \leq \operatorname{Tr}[\rho\{\mathcal{F}(\sigma) - \mathcal{F}(\rho)\}] \leq \gamma D(\rho\|\sigma) \end{aligned} \tag{69}$$

which satisfies (59). Hence, a double optimization form of the coherent information $I_{\text{coh}}(\mathcal{E})$ becomes

$$\max_{\rho,\sigma \text{ with } \sigma\gg\rho} J_\gamma(\rho, \sigma, \mathcal{E}) = I_{\text{coh}}(\mathcal{E}) \tag{70}$$

Performing the two maximizations in $\max_{\rho,\sigma} J_\gamma(\rho, \sigma, \mathcal{E})$ iteratively, leads to Algorithm 2 (see (60) and (61) for the update rules).

**Algorithm 4** Blahut-Arimoto type algorithm for the coherent information

Inputs: Quantum channel $\mathcal{E}_{A\to B}$, its complementary channel $\mathcal{E}_c$ and the respective adjoint channels $\mathcal{E}^\dagger_{B\to A}$ and $\mathcal{E}^\dagger_c$ (all given as lookup tables whose $(i,j)$ -th entry is given by the action of the channel on $|i\rangle\langle j|$ ), acceleration coefficient $\gamma$ and additive error $\varepsilon > 0$

Choose $\rho^{(1)} = \frac{1_A}{|A|}$

**for** $t \in \{1, 2, \ldots, n = \lceil \gamma \log |A|/\varepsilon \rceil\}$ **do**

$$\rho^{(t+1)} = \frac{1}{Z^{(t+1)}} \exp\left(\log \rho^{(t)} + \frac{1}{\gamma}\mathcal{F}\left(\rho^{(t)}\right)\right), \text{ where}$$

$$\mathcal{F}(\sigma) = \mathcal{E}_c^\dagger \log \mathcal{E}_c(\sigma) - \mathcal{E}^\dagger \log \mathcal{E}(\sigma) \text{ and}$$

$$Z^{(t+1)} = \mathrm{Tr}\left[\exp\left(\log \rho^{(t)} + \frac{1}{\gamma}\mathcal{F}\left(\rho^{(t)}\right)\right)\right] \text{ normalizes the state.}$$

**end for**

Outputs: $\rho^{(n+1)}$, $\quad I_{\mathrm{coh}}(n) = J_\gamma\left(\rho^{(n+1)}, \rho^{(n)}, \mathcal{E}\right)$ with $|I_{\mathrm{coh}}(\mathcal{E}) - I_{\mathrm{coh}}(n)| \leq \varepsilon$ =0

### 4.3.2 Simulations

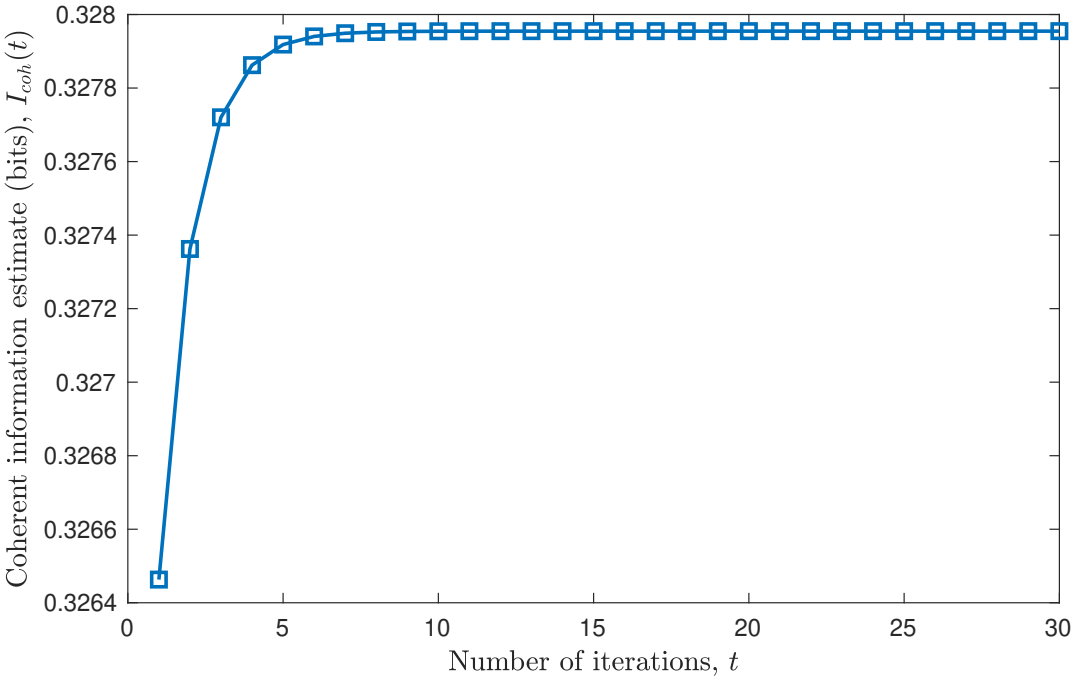


Figure 4: Convergence of the Blahut-Arimoto algorithm to the coherent information of the amplitude damping channel $\mathcal{E}^{AD}_{0.3}$.

We consider the amplitude damping channel $\mathcal{E}^{AD}_p$ with decay probability $p = 0.3$. We choose an additive error threshold of $\varepsilon = 10^{-6}$. Figure 4 shows the improvement obtained in the coherent information estimate with each iteration. The figure shows the lower bound on the coherent information in each iteration step $t$ until we terminate when $|C^\star - C(t)| \leq 10^{-6}$, by which we achieve an estimate with additive error smaller than $\varepsilon$. The standard Blahut-Arimoto algorithm takes $\gamma = 1$.

Similar to coherent information, it is shown in [RISB20] that we may also estimate other entropic optimization problems in quantum information by bringing them into the standard form (68) of Blahut-Arimoto algorithms. For all the cases discussed in the following subsections, the form of $\mathcal{F}(\sigma)$ satisfies (4.2.1) and this allows us to prove the convergence.

## 4.4 Mutual Information of Quantum Channels

The entanglement-assisted classical capacity represents the maximum rate at which one can reliably send a classical message through a quantum channel while using shared entanglement. For a channel $\mathcal{E}_{A\to B}$, it is given by the mutual information $I(\mathcal{E})$ defined as

$$I(\mathcal{E}) = \max_{\rho} S(\rho) + S(\mathcal{E}(\rho)) - S\left(\mathcal{E}_c(\rho)\right) \tag{71}$$

In this case the function $F(\sigma)$ is given by

$$\mathcal{F}(\sigma) = \mathcal{E}_c^\dagger \log \mathcal{E}_c(\sigma) - \log(\sigma) - \mathcal{E}^\dagger \log \mathcal{E}(\sigma) \tag{72}$$

### 4.4.1 Simulations

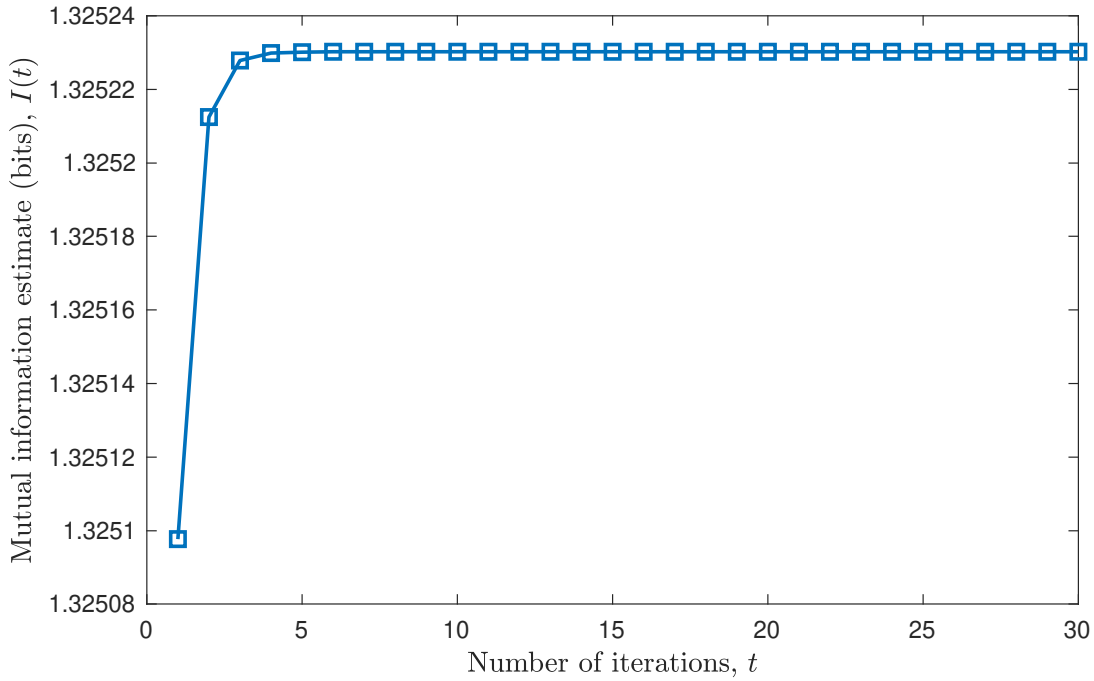


Figure 5: Convergence of the Blahut-Arimoto algorithm to the mutual information of the amplitude damping channel $\mathcal{E}_{0.3}^{AD}$.

We consider the amplitude damping channel $\mathcal{E}_p^{AD}$ with decay probability $p = 0.3$. We choose an additive error threshold of $\varepsilon = 10^{-6}$. Figure 5 shows the improvement obtained in the mutual information estimate with each iteration. The figure shows the lower bound on the mutual

information in each iteration step $t$ until we terminate when $|C^\star - C(t)| \leq 10^{-6}$, by which we achieve an estimate with additive error smaller than $\varepsilon$. The standard Blahut-Arimoto algorithm takes $\gamma = 1$.

## 4.5 Thermodynamic Capacity of Quantum Channels

The thermodynamic capacity quantifies the informationtheoretic power of quantum channels in the presence of physical restrictions imposed by thermodynamics. For a quantum channel $\mathcal{E}_{A\to B}$, it can be written as

$$T(\mathcal{E}) = \max_{\rho} S(\rho) - S(\mathcal{E}(\rho)) \tag{73}$$

In this case the function $F(\sigma)$ is given by

$$\mathcal{F}(\sigma) = \mathcal{E}^\dagger(\log \mathcal{E}(\sigma)) - \log \sigma \tag{74}$$

### 4.5.1 Simulations

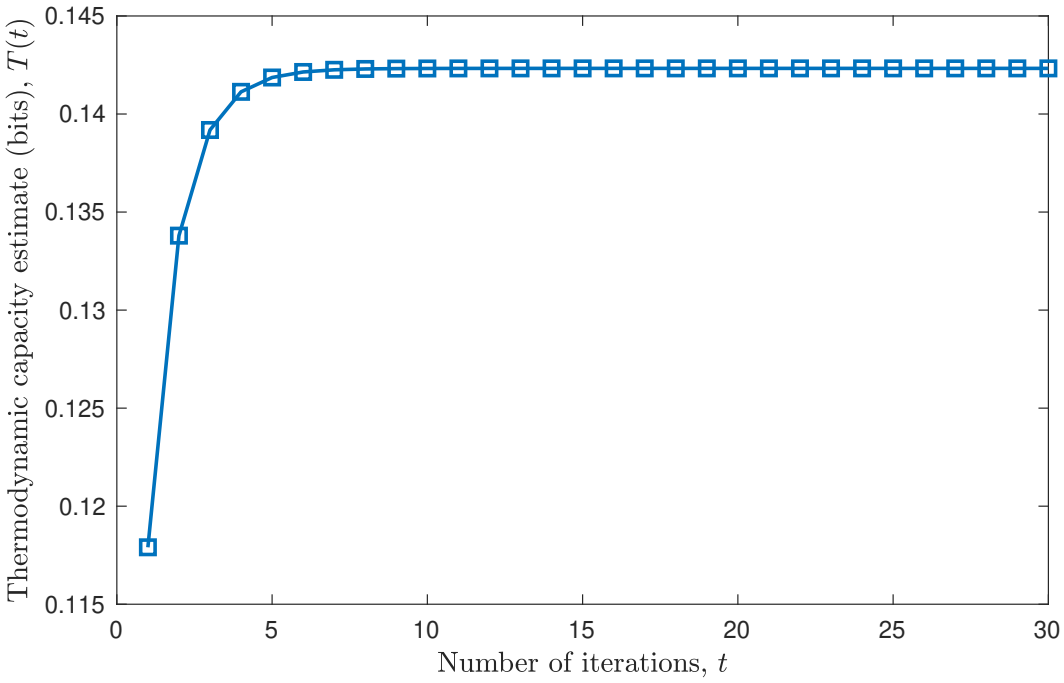


Figure 6: Convergence of the Blahut-Arimoto algorithm to the thermodynamic capacity of the amplitude damping channel $\mathcal{E}_{0.3}^{AD}$.

We consider the amplitude damping channel $\mathcal{E}_p^{AD}$ with decay probability $p = 0.3$. We choose an additive error threshold of $\varepsilon = 10^{-6}$. Figure 6 shows the improvement obtained in the thermodynamic capacity estimate with each iteration. The figure shows the lower bound on the thermodynamic capacity in each iteration step $t$ until we terminate when $|C^\star - C(t)| \leq 10^{-6}$, by

which we achieve an estimate with additive error smaller than $\varepsilon$. The standard Blahut-Arimoto algorithm takes $\gamma = 1$.

## 4.6 Holevo Quantity of Classical Quantum Channels

The Holevo quantity that quantifies the classical channel capacity of a cq channel is defined as

$$\chi(\mathcal{E}) = \max_{\lambda} \sum_{i} \lambda_i \operatorname{Tr}\left[\tau_{\mathcal{E},i} \left\{\log \tau_{\mathcal{E},i} - \log \mathcal{E}\left(\rho_\lambda\right)\right\}\right] \tag{75}$$

with $\tau_{\mathcal{E},i} = \mathcal{E}(|i\rangle\langle i|)$. In this case the function $F(\sigma)$ is given by

$$\mathcal{F}(\sigma) = \sum_{i} |i\rangle\langle i| \operatorname{Tr}[\mathcal{E}(|i\rangle\langle i|)(\log \mathcal{E}(|i\rangle\langle i|) - \log \mathcal{E}(\sigma))] \tag{76}$$

### 4.6.1 Simulations

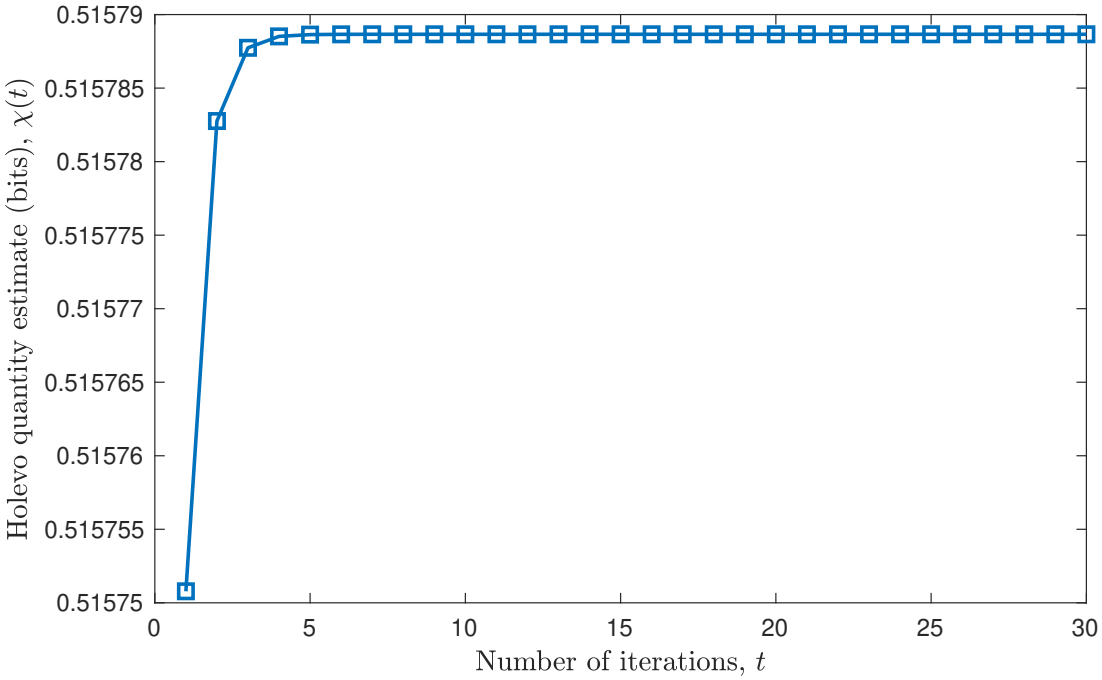


Figure 7: Convergence of the Blahut-Arimoto algorithm to the holevo quantity of the random classical quantum channel with input alphabet of size 2 and output dimension 2.

We consider a random classical quantum channel with input alphabet of size 2 and output dimension 2. The ensemble of output density operators were chosen randomly, conditioned on the fact that they satisfy the properties of density operators. The particular density operators for the simulation are

$$\tau_0 = \begin{pmatrix} 0.1022 + 0.0000i & 0.0164 - 0.2362i \\ 0.0164 + 0.2362i & 0.8978 + 0.0000i \end{pmatrix} \tag{77}$$

$$\tau_1 = \begin{pmatrix} 0.8261 + 0.0000i & 0.1732 - 0.2255i \\ 0.1732 + 0.2255i & 0.1739 + 0.0000i \end{pmatrix} \tag{78}$$

We choose an additive error threshold of $\varepsilon = 10^{-6}$. Figure 7 shows the improvement obtained in the Holevo quantity estimate with each iteration. The figure shows the lower bound on the Holevo quantity in each iteration step $t$ until we terminate when $|C^\star - C(t)| \leq 10^{-6}$, by which we achieve an estimate with additive error smaller than $\varepsilon$. The standard Blahut-Arimoto algorithm takes $\gamma = 1$.

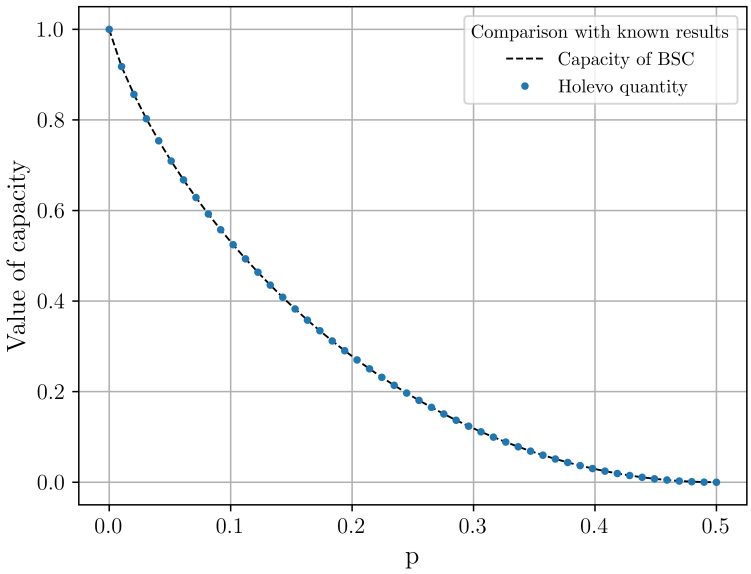


Figure 8: Comparison between the Holevo quantity when the matrices are cq channel are chosen to be same as the BSC($p$) and using the quantum Blahut-Arimoto algorithm to obtain the Holevo quantity, with the known $1 - h(p)$ for the BSC($p$).

The binary symmetric channel BSC($p$) is obtained in the cq channel formalism by choosing the input states to be $\{1, 2\}$ and the output density matrices to be

$$\rho_1 = \begin{pmatrix} p & 0 \\ 0 & 1-p \end{pmatrix} \qquad \rho_2 = \begin{pmatrix} 1-p & 0 \\ 0 & p \end{pmatrix} \tag{79}$$

We simulated the Holevo quantity for $0 \leq p \leq 1$ and compared it with the capacity of the BSC($p$), which has capacity $1 - h(p)$ where $h(p)$ is the binary entropy function $h(p) = -p\log(p) - (1-p)\log(p)$. The results are shown in fig. 8.

# MATLAB Codes for capacity calculations using Blahut-Arimoto Algorithm

This section contains all MATLAB files used for simulations in Section 3.3.

---

```
     breaklines
function [] = baa_dmc_cap_sim()

%% Calculates BSC capacity using classical BAA
eps=0:0.05:0.5;

channel_cap = zeros(1,length(eps));
entr_fun = zeros(1,length(eps));
for i=1:length(eps)

    tran_mat = [1-eps(1,i) eps(1,i);eps(1,i) 1-eps(1,i)];
    [cap, in_pmf] = calculate_cap_dmc(tran_mat);
    channel_cap(1,i) = cap;
    entr_fun(1,i) = 1-log2_entropy(eps(1,i),1/eps(1,i))-log2_entropy(1-eps(1,i),1/(1-eps(1,i)));

end

plot(eps,channel_cap,'sr',eps,entr_fun,'b','LineWidth',1.5);
legend('Using BAA','1-h(p)');
xlabel('Error Probability p');
ylabel('Capacity');
```

---

```
     breaklines
function [T_N_by] = calculate_T_params_fsmc(Q,p_g,p_b,eps_g,eps_b)

%% The function calculates the T parameters for the given GE channel parameters

N=20000;
N_bar = 2*N+1; %% Length of sequence
y_states = {'0','1'};
j_states = {'0g','1g','0b','1b'};

%%calculating V(s''_1|y_hat)
mu = sum(Q);
    P=zeros(2,2);
    for i=1:2
        for j=1:2
            P(i,j) = Q(i,j)/mu(1,i);  %% Source state transition matrix
        end
    end
T=[(1-p_b)*P(1,1) (1-p_b)*P(1,2) p_b*P(1,1) p_b*P(1,2); ...
    (1-p_b)*P(2,1) (1-p_b)*P(2,2) p_b*P(2,1) p_b*P(2,2); ...
    p_g*P(1,1) p_g*P(1,2) (1-p_g)*P(1,1) (1-p_g)*P(1,2); ...
    p_g*P(2,1) p_g*P(2,2) (1-p_g)*P(2,1) (1-p_g)*P(2,2)];
E = [1-eps_g eps_g; eps_g 1-eps_g; 1-eps_b eps_b; eps_b 1-eps_b];

[out_seq, int_j_states] = hmmgenerate(N_bar,T,E,'Symbols',...
    y_states,'Statenames',j_states);
%disp(out_seq);
%disp(int_j_states);
int_s_states = cell(1,2*N);
int_c_states = cell(1,2*N);
b = cell(1,2*N);
b_p = cell(1,2*N);
b_pp = cell(1,2*N);
for i=1:N_bar
    temp = int_j_states{i};
    int_s_states(1,i) = cellstr(temp(1));
    int_c_states(1,i) = cellstr(temp(2));
end

for i=1:2*N
    temp1 = int_s_states{i};
    temp2 = int_s_states{i+1};
    b(1,i) = cellstr(strcat(temp1,temp2));
    temp1 = int_c_states{i};
    temp2 = int_c_states{i+1};
    b_p(1,i) = cellstr(strcat(temp1,temp2));
    temp1 = int_j_states{i};
    temp2 = int_j_states{i+1};
    b_pp(1,i) = cellstr(strcat(temp1,temp2));
end

V_s_pp_y = hmmdecode(out_seq,T,E,'Symbols',{'0','1'});

%%calculating V(s''_1|b_hat,y_hat)
%% T remains same, output is now (b,y)
E1 = [P(1,1)*(1-eps_g) 0 P(2,1)*(1-eps_g)*mu(1,2)/mu(1,1) 0 ...
    P(1,1)*eps_g 0 P(2,1)*eps_g*mu(1,2)/mu(1,1) 0; ...
    0 eps_g*P(1,2)*mu(1,1)/mu(1,2) 0 P(2,2)*eps_g ...
    0 (1-eps_g)*P(1,2)*mu(1,1)/mu(1,2) 0 P(2,2)*(1-eps_g); ...
    P(1,1)*(1-eps_b) 0 (1-eps_b)*P(2,1)*mu(1,2)/mu(1,1) 0 ...
    P(1,1)*eps_b 0 eps_b*P(2,1)*mu(1,2)/mu(1,1) 0; ...
```

```
    0 eps_b*P(1,2)*mu(1,1)/mu(1,2) 0 P(2,2)*eps_b ...
    0 (1-eps_b)*P(1,2)*mu(1,1)/mu(1,2) 0 P(2,2)*(1-eps_b)];

out_seq_by = cell(1,2*N);
for i=1:2*N
    out_seq_by(1,i) = cellstr(strcat(b(i),out_seq(i+1)));
end
by_states = {'000','010','100','110','001','011','101','111'};
V_s_pp_by = hmmdecode(out_seq_by,T,E1,'Symbols',by_states);

%%Calculating V(b''_1|y_hat)
%% Both transition and emission matrices will change
b_pp_states = cell(1,16);
for i=1:4
    for j=1:4
        b_pp_states(1,4*(i-1)+j) = cellstr(strcat(j_states(i),j_states(j)));
    end
end
temp = blkdiag(T(1,:),T(2,:),T(3,:),T(4,:));
T_new = repmat(temp,4,1);
E_new = repmat(E,4,1);

V_b_pp_y = hmmdecode(out_seq(1,2:N_bar),T_new,E_new,'Symbols',y_states);
%disp(V_b_pp_y);

%%Calculating V(b''_1|b_hat,y_hat)
%% T remains same
temp1 = [1-eps_g 0 0 0 eps_g 0 0 0];
temp2 = [1-eps_b 0 0 0 eps_b 0 0 0];
temp3 = [temp1;circshift(temp1,5); temp2; circshift(temp2,5);];
temp4 = [temp3; circshift(temp3,2,2)];
E_nn = repmat(temp4,2,1);

V_b_pp_by = hmmdecode(out_seq_by,T_new,E_nn,'Symbols',by_states);

%%Calculating T_N_by matrices

T_N_by_2 = zeros(1,2); %% the second term in the equation
mu = sum(Q);

T_N_by_2(1,1) = sum((log2_entropy(V_s_pp_y(1,:),V_s_pp_y(1,:))...
    +log2_entropy(V_s_pp_y(3,:),V_s_pp_y(3,:))));
T_N_by_2(1,1) = T_N_by_2(1,1) - sum((log2_entropy(V_s_pp_by(1,:),V_s_pp_by(1,:))...
    +log2_entropy(V_s_pp_by(3,:),V_s_pp_by(3,:))));
T_N_by_2(1,2) = sum((log2_entropy(V_s_pp_y(2,:),V_s_pp_y(2,:))...
    +log2_entropy(V_s_pp_y(4,:),V_s_pp_y(4,:))));
T_N_by_2(1,2) = T_N_by_2(1,2) - sum((log2_entropy(V_s_pp_by(2,:),V_s_pp_by(2,:))...
    +log2_entropy(V_s_pp_by(4,:),V_s_pp_by(4,:))));
T_N_by_2 = T_N_by_2./mu;
T_N_by_2 = T_N_by_2/(2*N);

T_N_by_1 = zeros(2,2); %% the first term in the equation

T_N_by_1(1,1) = sum(log2_entropy(V_b_pp_y(1,:),V_b_pp_y(1,:))...
    +log2_entropy(V_b_pp_y(3,:),V_b_pp_y(3,:))...
    +log2_entropy(V_b_pp_y(9,:),V_b_pp_y(9,:))...
    +log2_entropy(V_b_pp_y(11,:),V_b_pp_y(11,:)));
T_N_by_1(1,1) = T_N_by_1(1,1) - sum(log2_entropy(V_b_pp_by(1,:),V_b_pp_by(1,:))...
    +log2_entropy(V_b_pp_by(3,:),V_b_pp_by(3,:))...
    +log2_entropy(V_b_pp_by(9,:),V_b_pp_by(9,:))...
    +log2_entropy(V_b_pp_by(11,:),V_b_pp_by(11,:)));

T_N_by_1(1,2) = sum(log2_entropy(V_b_pp_y(2,:),V_b_pp_y(2,:))...
    +log2_entropy(V_b_pp_y(4,:),V_b_pp_y(4,:))...
    +log2_entropy(V_b_pp_y(10,:),V_b_pp_y(10,:))...
    +log2_entropy(V_b_pp_y(12,:),V_b_pp_y(12,:)));
T_N_by_1(1,2) = T_N_by_1(1,2) - sum(log2_entropy(V_b_pp_by(2,:),V_b_pp_by(2,:))...
    +log2_entropy(V_b_pp_by(4,:),V_b_pp_by(4,:))...
    +log2_entropy(V_b_pp_by(10,:),V_b_pp_by(10,:))...
    +log2_entropy(V_b_pp_by(12,:),V_b_pp_by(12,:)));

T_N_by_1(2,1) = sum(log2_entropy(V_b_pp_y(5,:),V_b_pp_y(5,:))...
    +log2_entropy(V_b_pp_y(7,:),V_b_pp_y(7,:))...
    +log2_entropy(V_b_pp_y(13,:),V_b_pp_y(13,:))...
    +log2_entropy(V_b_pp_y(15,:),V_b_pp_y(15,:)));
T_N_by_1(2,1) = T_N_by_1(2,1) - sum(log2_entropy(V_b_pp_by(5,:),V_b_pp_by(5,:))...
    +log2_entropy(V_b_pp_by(7,:),V_b_pp_by(7,:))...
    +log2_entropy(V_b_pp_by(13,:),V_b_pp_by(13,:))...
    +log2_entropy(V_b_pp_by(15,:),V_b_pp_by(15,:)));

T_N_by_1(2,2) = sum(log2_entropy(V_b_pp_y(6,:),V_b_pp_y(6,:))...
    +log2_entropy(V_b_pp_y(8,:),V_b_pp_y(8,:))...
    +log2_entropy(V_b_pp_y(14,:),V_b_pp_y(14,:))...
    +log2_entropy(V_b_pp_y(16,:),V_b_pp_y(16,:)));
T_N_by_1(2,2) = T_N_by_1(2,2) - sum(log2_entropy(V_b_pp_by(6,:),V_b_pp_by(6,:))...
    +log2_entropy(V_b_pp_by(8,:),V_b_pp_by(8,:))...
    +log2_entropy(V_b_pp_by(14,:),V_b_pp_by(14,:))...
    +log2_entropy(V_b_pp_by(16,:),V_b_pp_by(16,:)));

T_N_by_1=T_N_by_1./Q;
T_N_by_1 = T_N_by_1/(2*N);

%%Final T_N_by matrix
T_N_by = zeros(2,2);

for i=1:2
    for j=1:2
```

```
158             T_N_by(i,j) = T_N_by_1(i,j)-T_N_by_2(1,i);
159         end
160     end
161
162     end
```

---

```
    breaklines
1   function [] = ge_capacity_bern_source()
2
3       %% Source is Bernoulli
4       %% Channel is Gilbert-Elliot Channel
5       r = rand(1,3);
6       q=[r(1,1) r(1,2);r(1,2) r(1,3)]/(sum(r)+r(1,2)); %% Initialization
7
8       p_g=0.3; p_b=0.3;
9       eps_g=0.001; eps_b=0:0.1:0.5; %% GE channel parameters
10      delta=0.001; %%stopping criteria threshold
11
12      cap_results = zeros(1,length(eps_b));
13
14      for iter = 1:length(eps_b)
15          q_r=q;
16          cap = 0;
17          prev_cap = 1;
18          disp(eps_b(1,iter));
19          while(abs(cap-prev_cap)>delta || cap<0)
20
21              T=calculate_T_params_fsmc(q_r,p_g,p_b,eps_g,eps_b(1,iter));
22
23              A = 2.^(T);
24
25              [M,L] = eig(A);
26              [m,index] = max(diag(L));
27              eig_vec = M(:,index);
28              P=zeros(2,2);
29              for i=1:2
30                  for j=1:2
31                    P(i,j) = (eig_vec(j,1)/eig_vec(i,1))*(A(i,j)/m);
32                  end
33              end
34              mu = [P(2,1)/(P(2,1)+P(1,2)) P(1,2)/(P(2,1)+P(1,2))];
35
36              for i=1:2
37                  for j=1:2
38                    q_r(i,j) = P(i,j)*mu(1,i);
39                  end
40              end
41              prev_cap = cap;
42              cap = 0;
43              for i=1:2
44                  for j=1:2
45                     cap = cap + q_r(i,j)*(log2(1/P(i,j))+T(i,j));
46                  end
47              end
48              disp("The capacity at this stage is : "+cap);
49              cap_results(1,iter)=cap;
50          end
51      end
52
53      plot(eps_b,cap_results,'b','LineWidth',1.4);
54      legend('Capacity of GE channel using Bernoulli source');
55      xlabel('Bad State error prob');
56      ylabel('Capacity');
57  end
```

---

```
    breaklines
1   function [T_hat] = calculate_T_params_dmc(Q,eps)
2
3       N=100000;
4       s_states = {'0','1'};
5       y_states = {'0','1'};
6
7       mu = sum(Q);
8       T=zeros(2,2);
9       for i=1:2
10          for j=1:2
11              T(i,j) = Q(i,j)/mu(1,i);  %% Source state transition matrix
12          end
13      end
14
15      Emis = 0.5*[(1-eps)+(1-eps) ...
16          (eps)+(eps); ...
17          (eps)+(eps) ...
18          (1-eps)+(1-eps)];
19
20      [out_seq, int_s_states] = hmmgenerate(N+1,T,Emis,'Symbols',...
21          y_states,'Statenames',s_states);
22
23      b = cell(1,N);
24      for i=1:N
25          temp1 = int_s_states{i};
26          temp2 = int_s_states{i+1};
```

```
        b(1,i) = cellstr(strcat(temp1,temp2));
    end

    V_s_y = hmmdecode(out_seq(2:end),T,Emis,'Symbols',{'0','1'});

    T_branch = [T(1,:) 0 0; 0 0 T(2,:);T(1,:) 0 0; 0 0 T(2,:)];

    Emis_branch = [Emis;Emis];

    V_b_y = hmmdecode(out_seq(2:end),T_branch,Emis_branch,...
        'Symbols',{'0','1'});

    T_hat = zeros(2,2);

    T_hat(1,1) = (sum(log2_entropy(V_b_y(1,:),V_b_y(1,:)))/Q(1,1) ...
        - sum(log2_entropy(V_s_y(1,:),V_s_y(1,:)))/mu(1,1))/N;
    T_hat(1,2) = (sum(log2_entropy(V_b_y(2,:),V_b_y(2,:)))/Q(1,2) ...
        - sum(log2_entropy(V_s_y(1,:),V_s_y(1,:)))/mu(1,1))/N;
    T_hat(2,1) = (sum(log2_entropy(V_b_y(3,:),V_b_y(3,:)))/Q(2,1) ...
        - sum(log2_entropy(V_s_y(2,:),V_s_y(2,:)))/mu(1,2))/N;
    T_hat(2,2) = (sum(log2_entropy(V_b_y(4,:),V_b_y(4,:)))/Q(2,2) ...
        - sum(log2_entropy(V_s_y(2,:),V_s_y(2,:)))/mu(1,2))/N;


end
```

---

```
    breaklines
function [] = bsc_capacity_rll_source()

    %% Source can not output more than one consecutive 0
    %% Channel is BSC
    r = rand(1,2);
    q=[0 r(1,1);r(1,1) r(1,2)]/(sum(r)+r(1,1)); %% Initialization


    eps=0:0.05:0.5;
    delta=0.001; %%stopping criteria threshold

    q_r=q;

    cap_results = zeros(1,length(eps));
    entr_fun = zeros(1,length(eps));
    for iter = 1:length(eps)
        disp(eps(1,iter));
        cap = 0;
        prev_cap = 1;
        while(abs(cap-prev_cap)>delta|| cap<0)

            T=calculate_T_params_dmc(q_r,eps(1,iter));
            T(isnan(T))=0;
            A = 2.^(T);
            A(1,1)=0;
            [M,L] = eig(A);
            [m,index] = max(diag(L));
            eig_vec = M(:,index);
            P=zeros(2,2);
            for i=1:2
                for j=1:2
                  P(i,j) = (eig_vec(j,1)/eig_vec(i,1))*(A(i,j)/m);
                end
            end
            mu = [P(2,1)/(P(2,1)+P(1,2)) P(1,2)/(P(2,1)+P(1,2))];

            for i=1:2
                for j=1:2
                  q_r(i,j) = P(i,j)*mu(1,i);
                end
            end
            prev_cap = cap;
            cap = sum(sum(log2_entropy(q_r,1./P)+q_r.*T));

            disp("The capacity at this stage is : "+cap);
            cap_results(1,iter)=cap;
        end

        entr_fun(1,iter) = 1-log2_entropy(eps(1,iter),1/eps(1,iter))-log2_entropy(1-eps(1,iter),1/(1-eps(1,iter)));
    end

    plot(eps,cap_results,'b','LineWidth',1.4);
    hold on;
    plot(eps,entr_fun,'g','LineWidth',1.4);
    legend('Capacity of BSC using RLL source','Capacity of BSC in general');
    xlabel('Error prob');
    ylabel('Capacity');
end
```

---

# Matlab Codes for Quantum Blahut-Arimoto Algorithm

This section contains all MATLAB files used for coding Quantum Blahut-Arimoto Algorithm. The following functions have not been included in this report. The code is available on Github at `https://github.com/priyankakaswan18/Quantum-Blahut-Arimoto-Algorithm`

- RandomDensityMatrix.m : Generates a random density matrix. (This function has been used form QETLAB: `http://www.qetlab.com/RandomDensityMatrix`)
- RandomUnitary.m : Generates a random unitary or orthogonal matrix. (This function has been used form QETLAB: `http://www.qetlab.com/RandomUnitary`)
- optargs.m : Handles optional input arguments for functions. (This function has been used form QETLAB: `http://www.qetlab.com/Opt_args`)
- tensor.m : computes the tensor product. (This function is being used from Toby Cubitt's webpage: `http://www.dr-qubit.org/matlab.html`)

---

```
    breaklines
% ket  Transforms a vector into column vector.
function w=ket(v)
   [~,x]=size(v);
   if x>1
     w=v.';
   else
     w=v;
   end
```

---

```
    breaklines
% bra  Transforms a vector into a normalized row vector.
function w=bra(v)
   [y,x]=size(v);
   if x>1
     w=conj(v);
   else
     w=v';
   end %if
   % normalization
   w=w/sqrt(w*w');
```

---

```
    breaklines
% ketbra  Dirac's bra-ket
%    ketbra(phi1,phi2) denotes the outer product of phi1 and phi2.
function k=ketbra(v1,v2)
k=ket(v1)*bra(v2);

if trace(k)~=0
   k=k/trace(k);
end
```

---

```
    breaklines
% braket  Dirac's bra-ket
%    braket(phi1,phi2) denotes the scalar(inner) product of phi1 and phi2.
function b=braket(phi1,phi2)
       b=bra(phi1)*ket(phi2);
```

---

```
    breaklines
function output = E(rho)

% this function gives action of a quantum channel with Kraus operators A0
% and A1 on a density matrix rho

global A0 A1
output=A0*rho*A0'+A1*rho*A1';
```

---

```
    breaklines
function output = Ec(rho)

% this function gives action of the complementary channel of a quantum channel with Kraus operators A0
% and A1 on a density matrix rho

global A0 A1 zero one

output= trace(A0*rho*A0')*ketbra(zero,zero)+ trace(A0*rho*A1')*ketbra(zero,one) + trace(A1*rho*A0')*ketbra(one,zero)+ trace(A1*rho*A1')*ketbra(one,one);
```

---

```
    breaklines
function output = Eadjoint(rho)

% this function gives action of the adjoint channel of a quantum channel with Kraus operators A0
% and A1 on a density matrix rho

global A0 A1

output = A0'*rho*A0+A1'*rho*A1;
```

---

```
    breaklines
function output = Ecadjoint(rho)

% this function gives action of the adjoint channel of the complementary channel
% of a quantum channel with Kraus operators A0 and A1 on a density matrix rho.

global zero one

% apply the Choi map to the complementary channel to obtain the corresponding Choi matrix
choi=tensor( Ec(ketbra(zero,zero)) , ketbra(zero,zero) )...
    +tensor( Ec(ketbra(zero,one)) , ketbra(zero,one) )...
    +tensor( Ec(ketbra(one,zero)) , ketbra(one,zero) )...
    +tensor( Ec(ketbra(one,one)) , ketbra(one,one) );


% compute the spectral decomposition of choi matrix
[V,D] = eig(choi);

% initializing Kraus operators of adjoint channel of complementary channel
% of the quantum channel

B={};
B{1}=zeros(2,2);
B{2}=zeros(2,2);
B{3}=zeros(2,2);
B{4}=zeros(2,2);

% Kraus operators will be the eigenvectors rearranged into a matrix
% and the weight of each Kraus operator will be the corresponding eigenvalue.

for i=1:4
    v1=V(:,i); % ith eigenvector
    d=D(i,i); % ith eigenvalue
    if d>10^(-6) % only updating a Kraus operator in case of corresponding non-zero eigenvalue
        % creation of elements of ith matrix (i.e., ith Kraus operator)
        % from ith eigenvector and ith eigenvalue
        B{i}(1,1)=v1(1)*sqrt(d);
        B{i}(1,2)=v1(2)*sqrt(d);
        B{i}(2,1)=v1(3)*sqrt(d);
        B{i}(2,2)=v1(4)*sqrt(d);

    end

end

% resultant action on density operator
output=B{4}'*rho*B{4}+B{1}'*rho*B{1}+B{2}'*rho*B{2}+B{3}'*rho*B{3};
```

---

```
    breaklines
function output = ERand(rho)

% this function gives action of a random classical quantum channel.
% Output density operators randomly in mainfile.m


global tau1 tau2


output=rho(1,1)*tau1+rho(2,2)*tau2;
```

---

```
    breaklines
function output = F(sigma,quantity)

% F is a Hermitian matrix valued super-operator on density operators used in algorithm,
% which is different for different quantities of interest
% quantity- the quantity of interest to be estimated, eg- Coherent information, Holevo quantity, etc
```

```
6
7   global zero one
8
9   if strcmp(quantity,'Thermodynamic_capacity')
10      output=Eadjoint(logm(E(sigma))./ log(2))-logm(sigma)./ log(2);
11  end
12
13  if strcmp(quantity,'Holevo_quantity')
14      output=ketbra(zero,zero)*trace( ERand(ketbra(zero,zero))*( logm(ERand( eye(2)*10^(-6) + ketbra(zero,zero)))./ log(2)-logm(ERand(sigma))./ log(2) ) )...
15          +ketbra(one,one)*trace( ERand(ketbra(one,one))*( logm(ERand( eye(2)*10^(-6) +ketbra(one,one)))./ log(2)-logm(ERand(sigma))./ log(2) ) );
16  end
17
18  if strcmp(quantity,'Quantum_mutual_information')
19      output=Ecadjoint(logm(Ec(sigma))./ log(2)) -Eadjoint(logm(E(sigma))./ log(2))  -logm(sigma)./ log(2);
20  end
21
22  if strcmp(quantity,'Coherent_information')
23      output=Ecadjoint(logm(Ec(sigma))./ log(2)) -Eadjoint(logm(E(sigma))./ log(2));
24  end
```

---

```
    breaklines
1   function output = J(rho,sigma,gamma,quantity)
2
3   % Writing the two variable extension J for estimation using Blahut-Arimoto algorithm.
4   % rho,sigma - density operators
5   % gamma- accerleration parameter for algorithm
6   % quantity- the quantity of interest to be estimated, eg- Coherent information, Holevo quantity, etc
7
8   output= -gamma * trace( rho*(logm(rho)./log(2)) )...
9       +trace( rho* ( gamma*logm(sigma)./log(2) + F(sigma,quantity) ) );
```

---

```
    breaklines
1   clear;
2   clc;
3
4   gamma=1; % acceleration parameter (=1 for standard algorithm)
5   epsilon=10^(-6); % additive error threshold
6   modA=2; % dimension of hilbert space A
7
8
9   %% Orthonormal basis states for qubit
10
11  global A0 A1 zero one p
12  zero=ket([1 0]);
13  one=ket([0 1]);
14
15  %% Kraus operators for Amplitude Damping Channel with decay probability p
16
17  p=0.3;
18  A0=ketbra(zero,zero)+sqrt(1-p)*ketbra(one,one);
19  A1=sqrt(p)*ketbra(zero,one);
20
21  %% Creating output density operators for random classical quantum channel
22
23  global tau1 tau2
24
25  tau1=RandomDensityMatrix(2);
26  tau2=RandomDensityMatrix(2);
27
28  disp(tau1);
29  disp(tau2);
30
31  %% Default settings for plots
32
33  width = 5;     % Width in inches
34  height = 3;    % Height in inches
35  alw = 0.75;    % AxesLineWidth
36  fsz = 11;      % Fontsize
37  lw = 1.5;      % LineWidth
38  msz = 8;       % MarkerSize
39  label_font=12;
40
41  %% Thermodynamic_capacity
42
43
44  % no_of_iterations=ceil(gamma*log(modA)/epsilon);
45
46  no_of_iterations=30; % number of iterations
47  quantity=zeros(1,no_of_iterations); % the quantity of interest, eg- Thermodynamic_capacity
48  rhot=eye(2)/modA; % intial density operator is chosen to be the maximally mixed state
49
50  for t=1:no_of_iterations % iterations of Blahut-Arimoto
51      Z1=trace(expm(log(2) .* (logm(rhot)./ log(2)+(1/gamma)*F(rhot,'Thermodynamic_capacity')) ));
52      rhotp1=(1/Z1)*expm(log(2) .*( logm(rhot)./ log(2)+(1/gamma)*F(rhot,'Thermodynamic_capacity')));
53      quantity(t)=J(rhotp1,rhotp1,gamma,'Thermodynamic_capacity');
54      rhot=rhotp1;
55  end
56
57  figure
58  pos = get(gcf, 'Position');
59  set(gcf, 'Position', [pos(1) pos(2) width*100, height*100]); %<- Set size
60  set(gca, 'FontSize', fsz, 'LineWidth', alw); %<- Set properties
```

```
plot(quantity,'-s','LineWidth',lw,'MarkerSize',msz); %<- Specify plot properites
ylabel('Thermodynamic capacity estimate (bits), $T(t)$','Interpreter','latex','FontSize',label_font)
xlabel('Number of iterations, $t$','Interpreter','latex','FontSize',label_font)


%% Holevo_quantity


no_of_iterations=30; % number of iterations
quantity=zeros(1,no_of_iterations); % the quantity of interest, eg- Holevo_quantity
rhot=eye(2)/modA; % intial density operator is chosen to be the maximally mixed state

for t=1:no_of_iterations % iterations of Blahut-Arimoto
    Z1=trace(expm(log(2) .*( logm(rhot)./ log(2)+(1/gamma)*F(rhot,'Holevo_quantity'))));
    rhotp1=(1/Z1)*expm(log(2) .*( logm(rhot)./ log(2)+(1/gamma)*F(rhot,'Holevo_quantity')));
    quantity(t)=J(rhotp1,rhotp1,gamma,'Holevo_quantity');
    rhot=rhotp1;
end

figure
pos = get(gcf, 'Position');
set(gcf, 'Position', [pos(1) pos(2) width*100, height*100]); %<- Set size
set(gca, 'FontSize', fsz, 'LineWidth', alw); %<- Set properties
plot(quantity,'-s','LineWidth',lw,'MarkerSize',msz); %<- Specify plot properites
ylabel('Holevo quantity estimate (bits), $\chi(t)$','Interpreter','latex','FontSize',label_font)
xlabel('Number of iterations, $t$','Interpreter','latex','FontSize',label_font)

%% Quantum_mutual_information


no_of_iterations=30; % number of iterations
quantity=zeros(1,no_of_iterations);  % the quantity of interest, eg- Quantum_mutual_information
rhot=eye(2)/modA; % intial density operator is chosen to be the maximally mixed state

for t=1:no_of_iterations % iterations of Blahut-Arimoto
    Z1=trace(expm(log(2) .*( logm(rhot)./ log(2)+(1/gamma)*F(rhot,'Quantum_mutual_information'))));
    rhotp1=(1/Z1)*expm(log(2) .*( logm(rhot)./ log(2)+(1/gamma)*F(rhot,'Quantum_mutual_information')));
    quantity(t)=J(rhotp1,rhotp1,gamma,'Quantum_mutual_information');
    rhot=rhotp1;
end

figure
pos = get(gcf, 'Position');
set(gcf, 'Position', [pos(1) pos(2) width*100, height*100]); %<- Set size
set(gca, 'FontSize', fsz, 'LineWidth', alw); %<- Set properties
plot(quantity,'-s','LineWidth',lw,'MarkerSize',msz); %<- Specify plot properites
ylabel('Mutual information estimate (bits), $I(t)$','Interpreter','latex','FontSize',label_font)
xlabel('Number of iterations, $t$','Interpreter','latex','FontSize',label_font)

%% Coherent_information


no_of_iterations=30; % number of iterations
quantity=zeros(1,no_of_iterations);  % the quantity of interest, eg- Coherent_information
rhot=eye(2)/modA; % intial density operator is chosen to be the maximally mixed state

for t=1:no_of_iterations % iterations of Blahut-Arimoto
    Z1=trace(expm(log(2) .*( logm(rhot)./ log(2)+(1/gamma)*F(rhot,'Coherent_information'))));
    rhotp1=(1/Z1)*expm(log(2) .*( logm(rhot)./ log(2)+(1/gamma)*F(rhot,'Coherent_information')));
    quantity(t)=J(rhotp1,rhotp1,gamma,'Coherent_information');
    rhot=rhotp1;
end

figure
pos = get(gcf, 'Position');
set(gcf, 'Position', [pos(1) pos(2) width*100, height*100]); %<- Set size
set(gca, 'FontSize', fsz, 'LineWidth', alw); %<- Set properties
plot(quantity,'-s','LineWidth',lw,'MarkerSize',msz); %<- Specify plot properites
ylabel('Coherent information estimate (bits), $I_{coh}(t)$','Interpreter','latex','FontSize',label_font)
xlabel('Number of iterations, $t$','Interpreter','latex','FontSize',label_font)
```

# Python Codes for Quantum Blahut-Arimoto Algorithms

This section contains the Python file with implementations for all the Blahut-Arimoto algorithms in [RISB20]. This is also available on Github at `https://github.com/sagnikb/quantum-blahut-arimoto`, which also has documentation on how to use the code.

```
     breaklines
import numpy as np
import scipy.linalg as linalg
import random
import matplotlib.pyplot as plt

def D(rho, sigma):
    '''
    Returns the quantum relative entropy between two density matrices rho and sigma.
    Does not check for ker(sigma) subseteq ker(rho) (in which case this value is inf)
    '''
    return(np.trace(rho @ (linalg.logm(rho) - linalg.logm(sigma)))/(np.log(2)))

def randpsd(n):
    '''
    Returns a random real psd matrix of dimension n x n, by first creating a random
    square matrix M of dimension n and then returning M @ M^T, which is always psd
    after making the trace 1
    '''
    M = np.zeros((n,n))
    for i in range(n):
        for j in range(n):
            M[i,j] = random.random()
    M = M @ (M.T)
    return (1/(np.trace(M))) * M

def create_cq_channel(dim, n):
    '''
    Creates a random cq-channel with input alphabet size n and output dimension dim
    Uses randpsd
    '''
    channel = []
    for i in range(n):
        channel.append(randpsd(dim))
    return channel

def create_basis(dim):
    '''
    Creates the standard basis for C^dim
    '''
    basis = []
    for i in range(dim):
        basis_vector = np.zeros((1, dim))
        basis_vector[0, i] = 1
        basis.append(basis_vector)
    return basis

def create_amplitude_damping_channel(p):
    '''
    Returns Kraus operators for 2x2 amplitude damping channel with parameter p
    '''
    kraus_operators = []
    M = np.zeros((2,2)); M[0,0] = 1; M[1,1] = np.sqrt(1-p)
    kraus_operators.append(M)
    M = np.zeros((2,2)); M[0,1] = np.sqrt(p)
    kraus_operators.append(M)
    return(kraus_operators)

def adjoint_channel(kraus_operators):
    '''
    Given a set of Kraus operators for a channel, returns the Kraus operators for
    the adjoint channel
    '''
    adjoint_kraus_operators = []
    for matrix in kraus_operators:
        adjoint_kraus_operators.append(matrix.conj().T)
    return adjoint_kraus_operators

def complementary_channel(kraus_operators):
    '''
    Given a set of Kraus operators for a channel, returns the Kraus operators for
    the complementary channel. First computes the Choi matrix for the Kraus operators,
    then computes eigenvalues and eigenvectors for the Choi matrix and then 'folds them'
    to create Kraus operators for the complementary channel
    (https://quantumcomputing.stackexchange.com/a/5797)
    '''
    n = len(kraus_operators)
    zbasis = create_basis(n)
    choi = np.zeros((np.square(n), np.square(n)))
    for j in range(n):
        for k in range(n):
            for l in range(n):
                for m in range(n):
                    choi = choi + np.trace(kraus_operators[m].conj().T @ kraus_operators[l] @ np.outer(zbasis[j], zbasis[k])) * \
                    np.kron(np.outer(zbasis[l], zbasis[m]), np.outer(zbasis[j], zbasis[k]))
    w, v = linalg.eigh(choi) #Choi matrix is symmetric, and eigh is more accurate
    v = v.T # the columns of V are the eigenvectors
```

```
    channel = []
    for i in range(len(w)):
        channel.append(np.sqrt(w[i])*np.resize(v[i], (n,n))) # folding to get the Kraus operators
    return channel

def act_channel(kraus_operators, density_matrix):
    '''
    Given a channel as a list of Kraus operators and an input density matrix,
    computes the output density matrix.
    '''
    l = len(kraus_operators)
    output_matrix = np.zeros(np.shape(density_matrix))
    for i in range(l):
        output_matrix = output_matrix + kraus_operators[i] @ density_matrix @ (kraus_operators[i].conj().T)
    return output_matrix

def J(quantity, rho, sigma, gamma, basis, channel, adjoint_channel, complementary_channel, adj_complementary_channel):
    '''
    Computes the function J from https://arxiv.org/abs/1905.01286 for the given quantity (which can be 'h',
    'tc', 'coh' or 'qmi') taking as input the channel and the associated adj, complementary and adjoint
    complementary channels
    '''
    return -1*gamma*np.trace(rho @ (linalg.logm(rho)/np.log(2))) + np.trace(rho @ (gamma * (linalg.logm(sigma)/np.log(2)) +
    F(quantity, sigma, basis, channel, adjoint_channel, complementary_channel, adj_complementary_channel)))

def F(quantity, sigma, basis, channel, adjoint_channel, complementary_channel, adj_complementary_channel):
    '''
    Computes the function J from https://arxiv.org/abs/1905.01286 for the given quantity (which can be 'h',
    'tc', 'coh' or 'qmi') taking as input the channel and the associated adj, complementary and adjoint
    complementary channels
    '''
    if quantity == 'h':
        s =  np.shape(basis[0])
        output_matrix = np.zeros((s[1], s[1]))
        Esigma = np.zeros((np.shape(channel[0])[0], np.shape(channel[0])[0]))
        for i in range(len(channel)):
            Esigma = Esigma + sigma[i,i] * channel[i]
        for i in range(len(channel)):
            output_matrix = output_matrix + np.outer(basis[i], basis[i]) * np.trace(channel[i] @ (linalg.logm(channel[i])/np.log(2) -
            linalg.logm(Esigma)/np.log(2)))
        return output_matrix
    elif quantity == 'tc':
        return -1*linalg.logm(sigma)/np.log(2) + act_channel(adjoint_channel, linalg.logm(act_channel(channel, sigma))/np.log(2))
    elif quantity == 'coh':
        return act_channel(adj_complementary_channel, linalg.logm(act_channel(complementary_channel, sigma))/np.log(2)) - \
        act_channel(adjoint_channel, linalg.logm(act_channel(channel, sigma))/np.log(2))
    elif quantity == 'qmi':
        return -1*linalg.logm(sigma)/np.log(2) + \
        act_channel(adj_complementary_channel, linalg.logm(act_channel(complementary_channel, sigma))/np.log(2)) - \
        act_channel(adjoint_channel, linalg.logm(act_channel(channel, sigma))/np.log(2))
    else:
        print('quantity not found')
        return 1

def capacity(quantity, channel, gamma, dim, basis, eps, **kwargs):
    '''
    Runs the Blahut-Arimoto algorithm to compute the capacity given by 'quantity' (which can be 'h', 'tc',
    'coh' or 'qmi' taking the channel, gamma, dim, basis and tolerance (eps) as inputs)
    With the optional keyword arguments 'plot' (Boolean), it outputs a plot showing how the calculated value
    changes with the number of iterations.
    With the optional keyword arguments 'latexplot' (Boolean), the plot uses latex in the labels
    '''
    if quantity != 'h': #holevo quantity doesn't need the other channels
        Adjoint_channel = adjoint_channel(channel)
        Complementary_channel = complementary_channel(channel)
        Adj_Complementary_channel = adjoint_channel(complementary_channel(channel))
    else:
        Adjoint_channel = channel; Complementary_channel = channel; Adj_Complementary_channel = channel
    #to store the calculated values
    itern = []
    value = []
    #initialization
    rhoa = np.diag((1/dim)*np.ones((1,dim))[0])
    #Blahut-Arimoto algorithm iteration
    for t in range(int(gamma*np.log2(dim)/eps)):
        itern.append(t)
        sigmab = rhoa
        rhoa = linalg.expm(np.log(2)*(linalg.logm(sigmab)/np.log(2) + \
        (1/gamma)*F(quantity, sigmab, basis, channel, Adjoint_channel, Complementary_channel, Adj_Complementary_channel)))
        rhoa = rhoa/np.trace(rhoa)
        value.append(J(quantity, rhoa, rhoa, gamma, basis, channel, Adjoint_channel, Complementary_channel, Adj_Complementary_channel))
    #Plotting
    if kwargs['plot'] == True:
        if kwargs['latexplot'] == True:
            plt.rc('text', usetex=True)
            plt.rc('font', family='serif')
        fig, ax = plt.subplots()
        plt.plot(itern, value, marker = '.', markersize='7', label = r'Capacity value vs iteration')
        plt.xlabel(r'Number of iterations', fontsize = '14')
        plt.ylabel(r'Value of capacity', fontsize = '14')
        plt.xticks(fontsize = '8')
        plt.yticks(fontsize = '8')
        plt.grid(True)
        plt.show()
    return J(quantity, rhoa, rhoa, gamma, basis, channel, Adjoint_channel, Complementary_channel, Adj_Complementary_channel)
```

# References


[Ari72] S. Arimoto. An algorithm for computing the capacity of arbitrary discrete memoryless channels. *IEEE Transactions on Information Theory*, 18(1):14–20, 1972. 1, 6

[Bla72] R. Blahut. Computation of channel capacity and rate-distortion functions. *IEEE Transactions on Information Theory*, 18(4):460–473, 1972. 1, 6

[CT84] I Csiszár and G Tusnády. Information geometry and alternating minimization problems. *Statistics & Decision, Supplement Issue No*, 1, 1984. 2

[Kav01] A. Kavcic. On the capacity of markov sources over noisy channels. In *GLOBECOM'01. IEEE Global Telecommunications Conference (Cat. No.01CH37270)*, volume 5, pages 2997–3001 vol.5, 2001. 1, 2, 8, 9

[RISB20] N. Ramakrishnan, R. Iten, V. Scholz, and M. Berta. Quantum blahut-arimoto algorithms. In *ISIT*, pages 1909–1914, 2020. 1, 2, 9, 16, 28

[Sha48] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948. 2

[VKAL08] P. O. Vontobel, A. Kavcic, D. M. Arnold, and H. Loeliger. A generalization of the blahut–arimoto algorithm to finite-state channels. *IEEE Transactions on Information Theory*, 54(5):1887–1918, 2008. 1, 2, 7, 8