COMPSCI 396: Final Report

Vishwesh Palani

September 5, 2025

Abstract

This report is a detailed summary of the research problem and literature review performed by the student as part of the requirements for COMPSCI 396: Independent Study (Spring 2025) taken under the supervision of Dr. Mordecai Golin. Dr. Golin's prior work on the Minimum Cost Markov Chain (MCMC) problem, the Markov Chain Polytope, and their connections to *Almost Instantaneous Fixed-to-Variable* (AIFV) length codes were extensively studied. This study primarily served as preparation for HONORS 499Y (Fall 2025) thesis work on the same problem.

Note: *Throughout this paper we will use a binary encoding alphabet (i.e.* $\Sigma_{encoding} = \{0, 1\}$*). Moreover,* $\Sigma := \Sigma_{encoding}$ *throughout this document.*

Coding Theory and Huffman Coding

Coding refers to the process of mapping *atomic* source symbols to codewords, thereby transforming sequences of source symbols into encoded messages. *Source coding* is used for data compression and results in the encoded message being shorter than the original message. *Channel coding* is used for error-correction by mapping source symbols to longer codewords to add structured redundancy so that errors introduced by a noisy channel may be detected and corrected by the receiver. This report will largely discuss methods for source coding, and we will assume that the set of source symbols $\mathcal{X}$ in question are **memoryless** (i.e. each symbol emitted is statistically independent of all prior symbols and they all follow the same fixed distribution $P(\mathcal{X})$).

Lossless data compression is a class of data compression techniques that allows the original data to be reconstructed using the compressed data with no loss of information. This means that any lossless compression scheme must be invertible or **uniquely decodable**. A **prefix-free** code is a code that does not have a codeword that is a prefix of any other codeword. The set of all prefix-free codes forms a proper subset of the set of uniquely decodable codes as every prefix-free code can be decoded by reading bits off the encoding until a complete codeword is read and repeating this process until the entire encoding has been decoded. It is a *proper* subset because there exist uniquely decodable codes that are not prefix-free (a relevant example being AIFV codes which we will discuss later).

Prefix-free codes are equivalently called *instantaneous* codes as they can be decoded instantaneously as soon as the final bit of a codeword (present within the encoding) is read — one can show that a code is prefix-free if and only if it is instantaneously decodable. It is also worth noting that prefix-free codes can be equivalently represented by a **single** $|\Sigma|$-ary coding tree (since the code is prefix-free, each codeword corresponds to a leaf with the edges representing a letter from Σ), so decoding a codeword reduces to starting from the root and picking a root-to-leaf path by selecting an edge based off of the letter read from the encoded message until we arrive at a leaf — one can also show that a code is prefix-free if and only if there exists a coding tree that can be used to decode an encoding. Thus, a code is **prefix-free** $\iff$ the code is *instantaneously* decodable $\iff$ there exists a **single** coding tree that can be used to decode an encoding.

Given a set of *fixed length* source symbols $\mathcal{X}$, a *Fixed-to-Variable* (FV) length code is a mapping $f_{FV} : \mathcal{X} \to \Sigma^*$ from source symbols $a \in \mathcal{X}$ to codewords $f_{FV}(a) \in \Sigma^*$ such that the length of different codewords may vary from symbol to symbol. This definition does not assume unique decodability as such a mapping may not always be used for lossless data compression; however, in the context of lossless compression, we will assume that the mapping in question is uniquely decodable. Huffman codes, which are prefix-free FV codes, assign shorter codewords to high-frequency source symbols and are guaranteed to minimize the expected (average) codeword length among the class of prefix-free FV codes [6].

The AIFV Family and the General AIFV-m Algorithm

In [7], Yamamoto and Wei showed that relaxing the constraints of (i) using a **single** coding tree (by alternating between multiple trees) and (ii) **instantaneous** decoding (by maintaining a bounded $O(1)$ bit decoding delay) could beat Huffman coding by producing *lower* average code lengths — these were the *K-ary Almost Instantaneous FV* codes. In the original construction of *K-ary AIFV* codes, the code is a tuple of coding trees that use a K-letter encoding alphabet ($K := |\Sigma|$) and fixes the number of coding trees used based on K — when $K = 2$, 2 trees are used with a decoding delay of 2 bits, and when $K > 2$, $K - 1$ coding trees are used with a decoding delay of 1 bit.

In [8], Hu et al. introduced *Binary AIFV-m* codes, which are identical to *2-ary AIFV* codes from [7], except that the number of coding trees is m (i.e. no longer fixed to 2 or $K - 1$) and the decoding delay is relaxed to at most m bits (instead of 1 or 2 bits). [8] and [9] proved that *Binary AIFV-m* codes have a redundancy of at most $\frac{1}{m}$ bits for $m = 2, 3, 4, 5$, thereby improving upon the bound of 1 bit for Huffman coding. In fact, Huffman coding is *actually* a special case of AIFV-m codes and is contained within the AIFV family, so AIFV codes are, **at worst**, as good as Huffman codes [7].

Constructing optimal k-AIFV and Binary AIFV-m codes cannot be done using a simple $O(n \log n)$ algorithm like Huffman coding. [10] described a *general* iterative technique to construct K-ary AIFV code trees for AIFV codes with two coding trees (i.e. $K = 2, 3$), [11] then proposed a finite iterative algorithm to construct optimal *Binary* AIFV-m codes and proved its correctness for $m = 2, 3, 4, 5$, and [12] and [13] proved correctness for $m > 5$.

The focus of this study will, henceforth, be restricted to the AIFV-m problem. We will define the following terms significantly more formally in subsequent sections. For now, assume that we have a candidate AIFV-m code T (which is just an m-tuple of candidate coding trees $(T_0, T_1, \ldots, T_{m-1})$). Given our set of source symbols $\mathcal{X}$ and the corresponding probability distribution $P(\mathcal{X})$, cost(T) is simply the average codeword length (which is calculated by first computing the probability of a symbol $a \in \mathcal{X}$ being encoded by a specific tree T_i, which can be computed using the formal definition of the coding trees).

Moreover, assume that each code T in our *search space* has a property x_T (each AIFV code T has a unique x_T but several distinct codes may have the same value for x_T so $T \mapsto x_T$ is a many-to-one relationship).

Having introduced the notion of cost(T) and x_T for an arbitrary AFIV-m code T, we can introduce a general description of the algorithm proposed by [11] is as follows:

```
Algorithm: General Iterative Algorithm described by [11]
  Input: Source Symbols $\mathcal{X}$ and Distribution $P(\mathcal{X})$
  Output: Minimum Cost AIFV-$m$ Code
  Initialization: $i \leftarrow 0$, $x_{T^{(0)}} \leftarrow$ arbitrary AIFV-$m$ code;
  while $x_{T^{(i)}} \neq x_{T^{(i-1)}}$ do
      get $T^{(i)}$ using $x_{T^{(i)}}$ (Tree Construction Step);
      get $x_{T^{(i+1)}}$ using $T^{(i)}$;
      $i \leftarrow i + 1$;
  return $T^{(i)}$;
```

Just like Huffman coding, this algorithm takes in the set of source symbols $\mathcal{X}$ as well as the corresponding probability distribution $P(\mathcal{X})$. When the algorithm was first proposed, the **Tree Construction Step** involved solving an NP-Hard Integer Linear Programming (ILP) problem. This step was later changed to an $O(n^5)$ Dynamic Programming (DP) problem for $m = 2$ [14], and subsequently improved to an $O(n^3)$ DP problem (which we will look into in more detail in an upcoming section of the report) [15]. For $m > 2$, the ILPs were replaced with DPs running in $O(mn^{2m+1})$ by [11]. Computing $x_{T^{(i)}}$ can be done using some clever, yet simple linear algebra formulation and Gaussian elimination as proposed by [2] (which we will explore in detail later on).

Surprisingly, $\forall i > 0 : \text{cost}(T^{(i)}) \leq \text{cost}(T^{(i-1)}))$ even though the algorithm depends on the convergence of the property $x_{T^{(i)}}$ — we will discuss the proof of this fact in subsequent sections of this report, after we have formally defined x_T. The algorithm terminates in a **finite** number of iterations. However, there is no bound on the number of steps to convergence for *any* value of m (although this exponential in the worst case, there is speculation of a tighter bound but none have been proven as of yet so this remains an **open** problem and is, thus, a possible direction of future study).

Binary AIFV-m Codes

The problem of finding the optimal AIFV-m code is finding the m-tuple of coding trees that minimizes a certain *loss criterion*. That loss criterion will be the average codeword length, which we will define shortly. First, let us define AIFV-m trees by first exploring AIFV-2 trees.

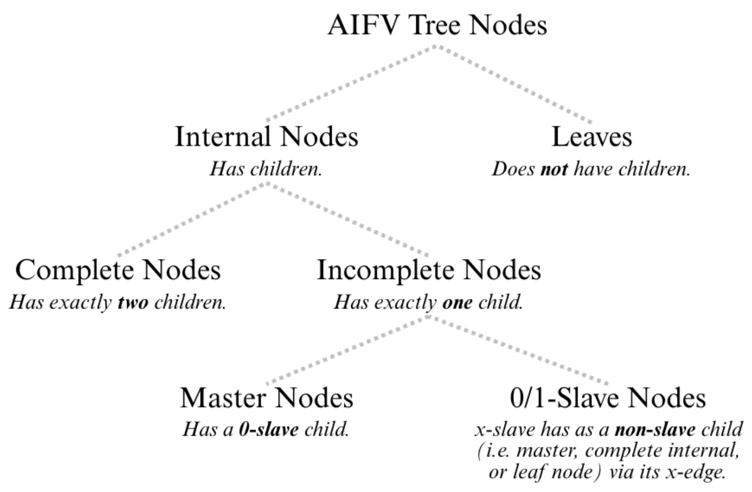


Figure 1: A classification tree for all the types of nodes present in AIFV-2 coding trees.

AIFV-2 Structure

Caution: This **entire** section has been written under the assumption that **leaf** nodes may or may not have source symbols assigned to them, so the distinction between *assigned* and *unassigned* leaves has been made all throughout the following sections. However, it has been pointed out to the student that definition has been modified to ensure that **all leaves must be assigned a source symbol**. This means that # **leaves** = # *assigned* **leaves** and # *unassigned* **leaves** = 0. Fortunately, this does not affect the following analyses.

This section will describe the properties of Binary AIFV-2 codes as per their original description provided by [7]. An AIFV-2 tree is a 2-tuple with two binary code trees $\langle T_0, T_1 \rangle$. Every left edge will read a 0 and every right edge will read a 1 (as shown in **Figure 2**).

Each tree has two types of nodes: internal nodes and leaf nodes. Internal nodes have two subtypes: complete internal nodes (With exactly 2 children that can be any type of node with no restrictions) and incomplete internal nodes (which have exactly 1 child). There are two types of incomplete internal nodes: master nodes (which must have a slave node as a child) and slave nodes (which can have all but slave nodes as children). Except for the additional classification of master and slave nodes, the rest of the nodes (e.g. complete, leaves, etc.) form the standard classes of binary tree nodes — AIFV code trees are binary trees with a few constraints described below (i.e. the set of AIFV code trees forms a proper subset of binary trees). A complete classification tree describing all types of nodes present in AIFV coding trees can be found above in **Figure 1**.

Note: Some authors refer to slave nodes as *intermediate* nodes instead.

There are a few constraints for AIFV code trees as follows:

1. Every source symbol $a \in \mathcal{X}$ must be assigned to *unique* node in T_0 and a *unique* node in T_1. These nodes must either be a leaf or a master node.

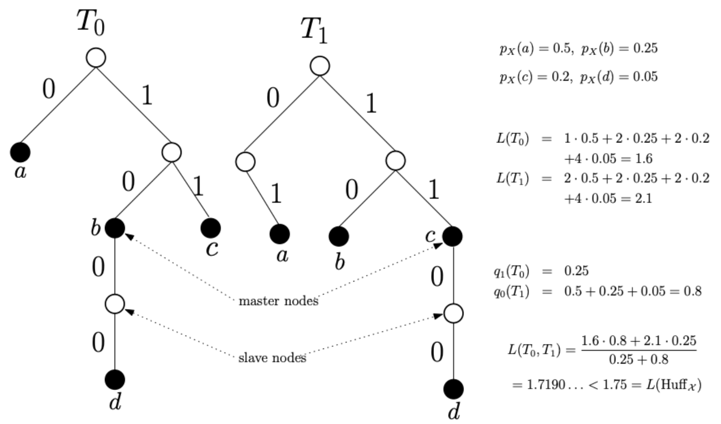
Corollary 1: #*assigned* **leaves** + #**masters** = $|\mathcal{X}|$
~~**Corollary 2:** $|\mathcal{X}| \leq$ #(leaves / master nodes)~~

2. *Every* master node must have a symbol assigned to it as per [1]. This is not necessarily the case for leaves.
   Corollary: #(master nodes) $\leq |\mathcal{X}|$.

3. Every incomplete node (slave and master) is connected to its only child via a left edge.
   Corollary: Every master node always has a 00 grandchild (since its child must be a slave node) but no 01 grandchild, as noted by [1].

4. The root of T_1 is **complete** and its left child is a slave node with its only child connected via a **right** edge—this slave node is the only exception to the previous constraint. The importance of this constraint will become apparent when we discuss the decoding procedure.
   Corollary: The root of T_1 has a 01 grandchild (and, thus, cannot be a master node as per the previous corollary).

Apart from these constraints, there are no other restrictions. Because of constraints 3 and 4, the slave nodes in AIFV-2 codes can be split into two types —
(i) 1-**slave** which is connected to its child via a 1-edge (there is exactly one these nodes in the *entire* AIFV-2 code i.e. the 0-child of the root of T_1 as per constraint 4), and
(ii) 0-**slave** which is connected to its child via a 0-edge (every other slave node apart from the one referenced in constraint 4 is a 0-slave).
The codeword for each source symbol $a \in \mathcal{X}$ are described by the sequence of edges (i.e. left-right-left-left $\longrightarrow$ 0100, just like in a Huffman coding tree) that form the path from the root to the unique node corresponding to a for that specific coding tree. It is important to note that our definition of master/slave nodes and the constraints permit the alternation of masters and slaves (i.e. we can have chains of master $\rightarrow$ slave $\rightarrow$ master $\rightarrow \ldots$) as shown in **Figure 2** below.



$$p_X(a) = 0.5,\ p_X(b) = 0.25$$
$$p_X(c) = 0.2,\ p_X(d) = 0.05$$

$$\begin{aligned} L(T_0) &= 1\cdot 0.5 + 2\cdot 0.25 + 2\cdot 0.2 \\ &\quad +4\cdot 0.05 = 1.6 \\ L(T_1) &= 2\cdot 0.5 + 2\cdot 0.25 + 2\cdot 0.2 \\ &\quad +4\cdot 0.05 = 2.1 \end{aligned}$$

$$\begin{aligned} q_1(T_0) &= 0.25 \\ q_0(T_1) &= 0.5 + 0.25 + 0.05 = 0.8 \end{aligned}$$

$$L(T_0, T_1) = \frac{1.6\cdot 0.8 + 2.1\cdot 0.25}{0.25 + 0.8}$$
$$= 1.7190\ldots < 1.75 = L(\mathrm{Huff}_{\mathcal{X}})$$

Figure 2: This figure was obtained from [1]. Above is a suboptimal binary AIFV-2 code $\langle T_0, T_1 \rangle$ for $\mathcal{X} = \{a, b, c, d\}$ with $P(\mathcal{X}) = \{0.5, 0.25, 0.2, 0.05\}$ as shown in the top right corner of the image. $L(\mathrm{Huff}_{\mathcal{X}})$ is the cost of the Huffman code for the same distribution demonstrating that even suboptimal AIFV-2 codes can beat the Huffman code.

Encoding with AIFV-2 codes

Assuming that we have a 2-tuple of coding trees (i.e. $\langle T_0, T_1 \rangle$), we can encode a sequence of source symbols as follows:

```
Algorithm: AIFV-2 Encoding Algorithm described by [1]
  Input: Sequence of source symbols $x_1 x_2 \dots x_n$
  Output: Encoded message $t$
  Initialization: $j \leftarrow 1$, $s_1 \leftarrow 0$, $t \leftarrow \epsilon$ (empty string);
  while $j \leq n$ do
      $t \leftarrow t + c_{s_j}(x_j)$ (append codeword to $t$ as per tree $T_{s_j}$);
      if $c_{s_j}(x_j)$ is a leaf in $T_{s_j}$ then
          $s_{j+1} \leftarrow 0$;
      else
          $s_{j+1} \leftarrow 1$;
      $j \leftarrow j + 1$;
  return $t$;
```

We know that symbols are assigned to either leaves or master nodes only so if a source symbol x_j is encoded by a leaf then the following symbol x_{j+1} will be encoded by T_0 and if x_j is encoded by a master node then x_{j+1} is encoded by T_1. Since s_1 is initialized to 0, the first symbol of any message is always encoded using T_0.

As an example, we will demonstrate the encoding process using the AIFV code in **Figure 2** and the sequence *cbadc* (as mentioned in **Figure 2**, $\mathcal{X} = \{a, b, c, d\}$. Set $t = \epsilon$ (empty string). The first symbol c must be encoded using T_0 so we will add 11 to t ($t = 11$). c is a leaf in T_0 so the next symbol b must be encoded using T_0 again. The remaining steps occur as follows:

i	x_i	s_i	$c_{s_i}(x_i)$	Leaf?	s_{i+1}
1	c	0	11	Yes	0
2	b	0	10	No	1
3	a	1	01	Yes	0
4	d	0	1000	Yes	0
5	c	0	11	Yes	N/A

s_i is just the identity of the encoding tree as presented in the original encoding algorithm and $c_{s_i}(x_i)$ is the codeword produced by the encoding tree for the source symbol x_i. s_1 is set to 0 by definition of the algorithm, but $\forall i > 0 : s_i$ is the same as s_{i+1} on the previous row (otherwise there would be a contradiction). The final codeword is 111001100011 which can be obtained by concatenating all the individual codewords $c_{s_i}(x_i) \forall i$.

Decoding with AIFV-2 codes

The decoding algorithm follows quite literally from the encoding algorithm. We will show how to decode a sequence of codewords by continuing from the previous example 111001100011.

Note that we don't know where each component code word ends, we have to deduce that on our own.

We know that the first symbol x_1 was encoded using T_0 by the definition of the encoding algorithm, we can go ahead and read bits off of the encoded message and trace a path through T_0 starting from the root. Starting from the root of T_0 after reading two 1s we end up at the leaf for c so $x_1 = c$ as we cannot read further symbols off of the encoded message from here onward as leaf nodes have no edges.

Since we know that $x_1 = c$ and was encoded using a leaf node, we know that x_2 is also encoded by T_0. So we repeat the process starting from the root of T_0 (remember, we have to continue reading bits from the encoded message starting from where we left of for x_1). After reading 10 from the encoding, we get to the master node for b. There are two possibilities: (i) $x_2 = b$ and the bits appearing after 10 in the encoding are part of the codeword for x_3, or (ii) x_2 is instead some symbol that appears further down the tree (we observed earlier that masters and slaves can alternate, so the codewords of some symbols can prefix the codewords of other symbols in valid AIFV coding trees).

To distinguish between cases (i) and (ii) we just need to read *two* extra bits. If we were in case (ii) and our source symbol was further down the tree we would have to read a 00 immediately (as per the corollary of constraint 3). Note that constraint 4 forces the root of T_1 to have a 01 grandchild, which means there *cannot* be a 00 grandchild as the root's 0 child is a slave node (only one child) — thus, it cannot be the case that $x_2 = b$ and the subsequent codeword from T_1 (since b is assigned to a master node in T_0) has a 00 prefix. So if a 00 is read we can *only* be in case (ii) and we are otherwise in case (i). Note that for case (ii) cannot instantaneously decode every codeword after reading the final bit of the codeword, instead we require a 2 bit or $O(1)$ bit delay (as mentioned earlier when we defined AIFV codes).

We can repeat this process until all the codewords have been exhausted. Here is the algorithm presented in formal terms:

Algorithm: AIFV-2 Decoding Algorithm described by [1]

```
Input: Binary encoding $y_1 y_2 \ldots y_n$
Output: Original sequence of source symbols $x_1 x_2 \ldots x_m$
Initialization: $i, j \leftarrow 1$, $s_1 \leftarrow 0$, $cur \leftarrow$ root of $T_0$, $cw, t \leftarrow \epsilon$ (empty string);
while $i \leq n$ do
    $cur \leftarrow y_i$-child of $cur$;
    $cw \leftarrow cw + y_i$ (append $y_i$ to $cw$ to build up the codeword);
    if $cur$ is an unassigned leaf in $T_{s_j}$ then
        throw error(”INVALID ENCODING”);
    if $cur$ is an assigned leaf in $T_{s_j}$ then
        $x_j \leftarrow$ symbol assigned to $cur$;
        $t \leftarrow t + x_j$ (append $x_j$ to the sequence of source symbols);
        $j \leftarrow j + 1$;
        $s_j \leftarrow \mathbf{0}$;
        $cur \leftarrow$ root of $T_{s_j}$;
        $cw \leftarrow \epsilon$ (reset $cw$ to empty string);
    if ($cur$ is a master in $T_{s_j}$) $\wedge$ ($y_{i+1} y_{i+2} \neq 00$) then
        $x_j \leftarrow$ symbol assigned to $cur$;
        $t \leftarrow t + x_j$ (append $x_j$ to the sequence of source symbols);
        $j \leftarrow j + 1$;
        $s_j \leftarrow \mathbf{1}$;
        $cur \leftarrow$ root of $T_{s_j}$;
        $cw \leftarrow \epsilon$ (reset $cw$ to empty string);
    $i \leftarrow i + 1$
if $cw = \epsilon$ then
    return $t$;
else
    throw error(”INVALID ENCODING”);
```

Note: The **if** clause for *unassigned* **leaves** is no longer relevant as per the *new* definition of AIFV-m codes that forces all leaves to be *assigned* a source symbol.

Argument for Exponential Search Space

We mentioned earlier that the search space for AIFV coding trees is exponential in $|\mathcal{X}|$; now we will argue this fact. Because there is no requirement that every leaf must have a symbol assigned to it, technically there can be *infinitely* many structurally distinct candidate coding trees (and therefore AIFV codes) for any set of source symbols $\mathcal{X}$ as we could select an existing AIFV coding tree with a leaf that has no symbol assigned to it, and replace that leaf with a complete binary tree of any size (using only complete internal nodes and unassigned leaves at the end) thereby producing infinitely many trees. That said, we can take any such tree and prune sub-trees that contain only unassigned nodes in time that is polynomial in the size of the AIFV code (this can be done inefficiently with repeated DFS/BFS traversal). This will output an AIFV coding tree with at most $|\mathcal{X}|$ leaves — these are candidate coding trees. Since the number of binary trees with exactly n leaves is the Catalan number $C_{n-1} = \Theta(\frac{4^n}{n^{3/2}})$, we know that the number of viable AIFV coding trees is $O(|\mathcal{X}| \cdot C_{|\mathcal{X}|-1})$ which is exponential in $|\mathcal{X}|$.

Relationship between type-k coding trees

Note: A type-k coding tree is one that may be used as the kth coding tree for any AIFV-m code.

It is worth noting that for $m = 2$, a type-1 coding tree can be made up of type-0 coding trees as per the following diagram.

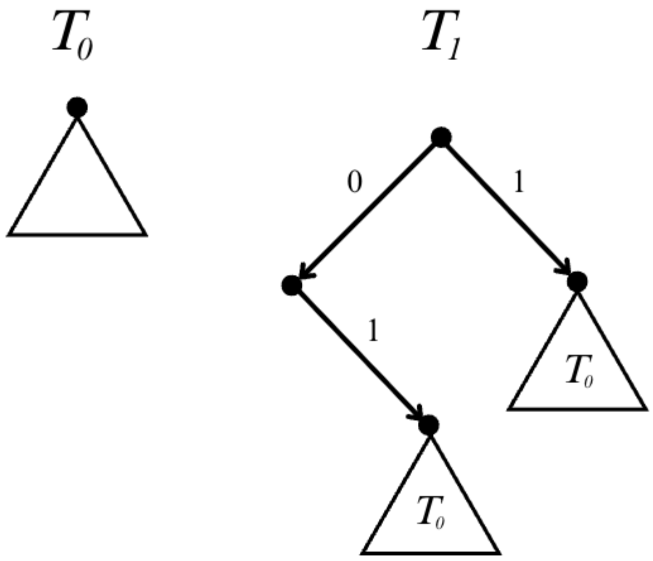


Figure 3: Type-0 trees make up *every* type-1 tree.

This is a very useful relationship to leverage in proofs regarding the coding trees and is used in a proof provided by Dr. Golin in a following subsection as well as in [1]. This relationship can also be used to describe the relationship between sets of AIFV-m coding trees (all of them are made up of type-0 coding trees). Thus, the orders of the sets of all the type-k coding trees depends on the order of type-0 trees.

More information regarding AIFV-m trees would be required to draw further conclusions.

Reduction to the Minimum Cost Markov Chain (MCMC) Problem

To search for the optimal AIFV code throughout this exponential search space, we will require a loss criterion (as mentioned earlier), which we will quote from [1]. Suppose we are given an AIFV code $\langle T_0, T_1 \rangle$. The average codeword length for T_s is $L(T_s) = \sum_{a \in \mathcal{X}} |c_s(a)| \cdot p_{\mathcal{X}}(a)$ where $s \in \{0, 1\}$. This is a loss measure for *individual* coding trees but we are yet to define a loss measure for the *entire* AIFV code. To do so, we will first model the encoding process using a 2-state Markov chain.

Suppose we are currently encoding using T_j. We know that according to the source distribution $P(\mathcal{X})$, the probability of $a \in \mathcal{X}$ occurring is $p_{\mathcal{X}}(a)$. Let $q_i(T_j)$ be the probability of switching from $T_j \to T_i$ (since there are only two trees, $i, j \in \{0, 1\}$). Given an arbitrary T_j, $q_0(T_j) = \sum_{a \in \textit{assigned}\ \textbf{leaves}\text{ of } T_j} p_{\mathcal{X}}(a)$ and $q_1(T_j) = \sum_{a \in \textbf{masters}\text{ of } T_j} p_{\mathcal{X}}(a)$ (we have essentially created a probability distribution using $P(\mathcal{X})$ since $\forall j : q_0(T_j) + q_1(T_j) = 1$).

Now, if we create a 2-state chain with state for T_0 and T_1 respectively and designate the tree-switching probabilities as state transitions, we have a well-defined Markov chain modeling the encoding process (with the current state being the current encoding tree) as follows:

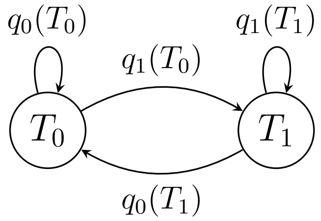

$$\begin{tikzcd} T_0 \arrow[loop above, "q_0(T_0)"] \arrow[r, bend left, "q_1(T_0)"] & T_1 \arrow[loop above, "q_1(T_1)"] \arrow[l, bend left, "q_0(T_1)"] \end{tikzcd}$$

Figure 4: Two-state Markov chain with transition probabilities $q_i(T_j)$.

We can assign the average codeword length of T_s, or $L(T_s)$, as the cost of state T_s in the Markov chain. This results in the cost of the Markov chain, and by extension the AIFV code, as follows:

$$L_{AIFV}(T_0, T_1) = \pi_0 \cdot L(T_0) + \pi_1 \cdot L(T_1)$$

where π_s is the probability of being in state T_s according to the stationary distribution of the Markov chain. We can derive the stationary distribution with some simple algebra as follows:

$$\pi_0 = \pi_0 \cdot q_0(T_0) + \pi_1 \cdot q_0(T_1)$$

$$\pi_1 = \pi_0 \cdot q_1(T_0) + \pi_1 \cdot q_1(T_1)$$

$$\pi_0 + \pi_1 = 1 \to \pi_1 = 1 - \pi_0$$

$$\pi_0 = \pi_0 \cdot q_0(T_0) + (1 - \pi_0) \cdot q_0(T_1) = \pi_0 \cdot (q_0(T_0) - q_0(T_1)) + q_0(T_1)$$

$$\implies \pi_0 \cdot (1 - q_0(T_0) + q_0(T_1)) = q_0(T_1)$$

$$\implies \pi_0 \cdot (q_1(T_0) + q_0(T_1)) = q_0(T_1)$$

$$\implies \boxed{\pi_0 = \frac{q_0(T_1)}{q_1(T_0) + q_0(T_1)}}$$

$$\implies \boxed{\pi_1} = 1 - \pi_0 = 1 - \frac{q_0(T_1)}{q_1(T_0) + q_0(T_1)} = \boxed{\frac{q_1(T_0)}{q_1(T_0) + q_0(T_1)}}$$

This is the formula for the stationary distribution as presented in [1] (there are some *edge cases* to be aware of but those are clearly outlined in the same paper).

This can be plugged into the previous formula for $L_{AIFV}(T_0, T_1)$ which can be used as our loss criterion to search through the exponential AIFV-2 code possibilities and find the **minimum cost AIFV-2 code** for a given set of source symbols $\mathcal{X}$ and its distribution $P(\mathcal{X})$.

Moreover, since we are relying on the Markov model for the AIFV-2 code, it is safe to state that the problem of finding the optimal AIFV-2 code reduces to finding the minimum cost 2-state Markov chain (within the search space of permissible Markov chains given the AIFV constraints) — this reduction treats structurally distinct AIFV coding trees with the same cost as equivalent, which is a valid assumption as we are searching for the minimum-cost AIFV code (if there are multiple such codes, we can arbitrarily pick any of them and we would still have a valid solution to our problem). However, finding the minimum cost AIFV-2 code (or finding the minimum cost Markov chain, given the AIFV constraints) is a hard problem and is not as simple as finding the optimal Huffman code.

AIFV-m Structure

Note: In this section, when we refer to a tree T_i as having a 01 node, for example, it means that there exists a path from the root of T_i such that the path $root \to 0 \to 1 \to$ will lead to the node in question.

The problem of finding the optimal AIFV-m code can be solved for any m by solving the equivalent minimum cost Markov chain problem (which we will explore in much more detail in the next section, and is *significantly* simpler to define). That said, it is worth briefly discussing the structure of AIFV-m codes independently.

The structure of AIFV-m trees can be extended from the discussion of AIFV-2 codes.

- AIFV-m codes have m coding trees by definition with m types of master nodes of degree k where $0 < k < m$.
- A degree-0 master node is just a *leaf* (as in the AIFV-2 case).
- For $k > 0$, a degree-k master node's only descendants for the next k levels are via the edges 0^k. All of these nodes are intermediate nodes with a zero child.
- Furthermore, in each T_k tree ($0 < k < m$), the node 0^k exists and is an intermediate node, while 0^k1 exists and is *not* an intermediate node. These rules have been designed to permit unique decodability.

Careful observation will reveal that the previously discussed AIFV-2 definition agrees with this generalized AIFV-m definition. The AIFV-2 codes have a master node as well as a leaf node (we can consider our leaf as a degree-0 master node). Likewise, master nodes have must have a slave child via a 0-edge (i.e. since $k = 1$, for 0^1 levels there are only intermediate nodes).

According to the provided definition of AIFV-m trees, for $m = 2$, T_1 must have a $0^1 = 0$-slave node (and since the child of an AIFV-2 slave cannot be a slave) the 01-node is not a slave either. Recall, that this was observed to ensure $O(1)$-bit distinguishability during the decoding process. Not only does this show that AIFV-2 codes fit into the generalized AIFV-m code family (as the case of $m = 2$), but it also provides intuition for how that constraint ensures unique decodability for all m.

The encoding and decoding process for AIFV-m codes is a little more complicated and irrelevant to the problem of finding the optimal AIFV-m code so we won't discuss it here.

Equivalence classes of type-k AIFV-m coding trees

Recall that for the AIFV-2 case, reducing to the MCMC problem involved transforming the AIFV-2 code into a 2-state Markov chain and the type-k coding tree T_i into a Markov chain state using $q_0(T_i), q_1(T_i)$, and $\text{cost}(T_i)$. For the general case, we would need to reduce the AIFV-m code into an m-state. The Markov chain state corresponding to the kth coding tree will depend on $m + 1$ values, namely m transition probabilities to the other m states (which is just the tree-switching probabilities to all m coding trees, including itself) and the cost of state (which is the coding tree's cost or average codeword length).

As noted earlier, these $m + 1$ values are the only data that is required to model each state and solve the AIFV-m problem as a MCMC problem, since the loss criterion L_{AIFV} only depends on the stationary distribution π and the individual costs of each coding tree. Moreover, the type-k trees are independent of each other in any AIFV-m code, as transition probabilities to other trees in the code do not depend on the trees themselves (which was outlined earlier as well).

Thus, it follows that structurally-distinct type-k coding trees with the same tree-switching probabilities and cost are considered equivalent when trying to find the *optimal* AIFV-m code as their contributions to L_{AIFV} (via the stationary distribution π and the individual tree cost) will be identical for *any* AIFV-m code.

As a result the $m + 1$ values produced by each of these coding trees (required to define the Markov chain state) will be the same, so they will all define the exact same state. Thus, we can consider this group of trees as an equivalence class represented by a *unique* Markov chain state.

Huffman Codes are AIFV-1 codes

AIFV-m codes are only defined for $m \geq 2$ (so technically, the heading for this subsection does not make sense). However, if we take the formal definition for AIFV-m codes defined above and observe the rules for $m = 1$, something interesting happens. As per our definition AIFV-1 codes have a single coding tree and a single type of master node (the degree-0 node i.e. *leaves*).

Note that for a single coding tree full of leaves, the encoding and decoding algorithms become identical to the Huffman coding algorithms. And since the updated definition of AIFV-1 codes requires that all leaves must be *assigned*, such codes are just Huffman codes. So the AIFV-1 coding problem is just the Huffman coding problem and the optimal code can be computed using the classic $O(n \log n)$ algorithm.

With the designation of being an AIFV-1 code, however, comes the guarantee that the coding delay is bounded by $\frac{1}{1} = 1$ bit since an AIFV code is *Almost Instantaneous*. However, since we are dealing with Huffman codes, we know that these are actually *instantaneous*.

Thus, while designating Huffman codes as AIFV-1 codes is valid, it defeats the AIFV objective of sacrificing instantaneity for a lower average cost (this is no longer being done when we recognize the AIFV-1 problem as the Huffman coding problem).

Nevertheless, Huffman coding is still recognized as a special case within the family of AIFV codes as identified by [1] — just maybe not explicitly as AIFV-1 codes.

MCMC Problem Overview

In the previous section, we discussed how to abstract away the details of AIFV-2 codes so that we can solve the same combinatorial optimization problem using the simpler Markov chain model. In this section, we will define this formally for AIFV-m codes and discuss some geometric interpretations.

Caution: Remember, the *old* definition of AIFV-m codes described above did not include the condition that $\forall k \in [m] : q_0(T_k) > 0$. However, the definition has since been modified and the *new* definition includes this condition (this was pointed out in the subsection describing the *old* definition). This means that codes with at least *one coding tree that violates this rule* are no longer valid AIFV-m codes. This is the only distinction between the *old* and *new* definitions for AIFV-m codes. Moving forward, every reference to an AIFV-m code will refer to the *new* definition, **unless specified**.

Significance of the MCMC Problem

We already know that there is a useful reduction from the problem of finding the optimal AIFV-m coding tree to the problem of finding the optimal minimum cost Markov chain. However, there are many more problems such as finding better parsing trees [16], lossless codes for finite channel coding [17], and AIFV codes for unequal bit-cost coding [18] that can be solved via reduction to the MCMC problem. As noted in [2], each of these problems has an input size n, with a set of n associated probabilities but the search space had size exponential in n.

It is very simple to see that the argument for exponentially many AIFV coding trees results in an exponentially many distinct-cost Markov chains through which we would have to search through to solve the corresponding minimum cost Markov chain problem. Previous iterative algorithms for each problem moved from candidate to candidate (such that the cost was non-increasing) and these ran in time that was exponential in n where each iteration required solving a local optimization problem that ran in time polynomial in n. In the AIFV-2 case, this is the $O(n^3)$ DP that is solved to construct trees that was mentioned earlier (this will be discussed in much more detail later on).

Markov Chain Prerequisites

Note: This section assumes that the basic definition of a Markov chain is known to the user.

A **recurrent** class (of a Markov chain) is a set of states that, once entered, the chain cannot leave. Moreover, recurrent classes are **communicating** which means that every pair of states is mutually-reachable via some finite sequence of states within the recurrent class. A chain could have multiple recurrent classes. A **transient** state is a state that once left can never be re-entered (this means that the probability of being in this state in the stationary distribution is 0).

A Markov chain is **irreducible** if all its states form a *single* recurrent class and the Markov chain does *not* have any transient states. The process of reducing a Markov chain is relatively simple: (1) identify the transient states and delete them (since they do not contribute to long-term behavior), (2)identify and break the chain into its recurrent classes (which are closed, maximal states sets that once entered, are never left). This process will produce the irreducible components that make up the original chain. Note that sometimes, Markov chains with a single recurrent class are also called *irreducible* because in the long-run the stationary distribution is that of the recurrent class (the transient states have a probability of 0, since they will be left eventually).

There is a lot more theory concerning the *existence* of a *unique* stationary distribution as well as whether or not there is convergence but that is not relevant to the rest of the paper.

Defining the Problem

The following definitions come from [2]. Fix $m > 1$ (m is the number of states). A state S in a Markov chain is defined by m transition probabilities $\{q_j(S) \mid j \in [m]\}$ as well as the cost/reward/net. value incurred when visiting that state $L(S)$ — so a state can be thought of as a vector belonging to $\mathbb{R}^{m+1}$ (with some constraints on its entries). Note that the set of m transition probabilities for S forms a probability distribution so it must be the case that $\forall j : q_j(S) \geq 0$ **and** $\sum_{j\in[m]} q_j(S) = 1$.

Define $\mathbb{S}_k$ for $k \in [m]$ to be the type-k state set containing a type-k state for every possible type-k coding tree in *any* possible AIFV-m code. A type-k state can be constructed for a type-k coding tree using the reduction described earlier. The type-k state set $\mathbb{S}_k$ will differ from problem to problem, so we will work with $\mathbb{S}_k$ generally (without much reference to the AIFV-m problem).

Note: An instance of the MCMC problem *always* works with a constraint on $\mathbb{S}_k$ (for example in the AIFV-m case, a type-k state S_k exists in $\mathbb{S}_k$ if and only if there exists at least one type-k coding tree that can produce that state (by producing the required $m + 1$ values). Otherwise that state cannot exist.

If there was no constraint, then in any instance of the problem, we could solve the problem by constructing a zero-cost Markov chain by ensuring all of its states have an individual cost of 0. This would make the problem meaningless. So the constraints on $\mathbb{S}_k \mid \forall k$ give the MCMC problem meaning and practical relevance.

Define $\mathbb{S} = \times_{k=0}^{m-1} \mathbb{S}_k = \{(S_0, \ldots, S_{m-1}) \mid \forall k \in [m], S_k \in \mathbb{S}_k\}$ to be the set of *permissible* Markov chains — since the states represent valid coding trees, *permissible* Markov chains represent valid AIFV-m codes. Each m-state Markov chain is constructed by picking the k-th state from the type-k state set.

There is one more constraint on each type-k state set: $\forall k \in [m] : \forall S_k \in \mathbb{S}_k : q_0(S_k) >$

0. This ensures that there is only one *aperiodic recurrent* class in the chain — the one that contains S_0. Whenever there are any two recurrent classes A, B, adding even a single transition between a state in A to a state in B causes A to no longer be a recurrent class — this is because there is now a way for the Markov chain to leave the set A, it is no longer **closed** (as a result all of the states in A become transient since they will eventually go to B via the $A \to B$ transition and never return). Adding a non-zero probability of transitioning to S_0 from any state is equivalent to adding an edge from all of the recurrent classes in the chain to the recurrent class containing S_0, and as a result, all but one recurrent class disappear. Now, we must argue that the recurrent class containing S_0 is *aperiodic.* For a class of states to be aperiodic, all states must have a period of 1. The **period** of a state is defined as the greatest common divisor of the lengths of all paths from a state to *itself.* Since S_0 has a self-loop, its period is 1 (since there exists a 1-step path $S_0 \to S_0$ via the self-loop). A fundamental property of Markov chains is that all states within the same communicating class share the same period — so all states in the recurrent class containing S_0 have a period of 1 and are, therefore, aperiodic. This completes the proof. Note that these conditions imply that the m-state Markov chain is an *ergodic unichain* as noted by [2].

The MCMC problem is formally defined as finding $S^* \in \mathbb{S}$ such that $\text{cost}(S^*) = \min_{S \in \mathbb{S}} \text{cost}(S)$. It is worth noting that since the search space for coding trees is exponential in n, $|\mathbb{S}_k| = O(e^n) \implies |\mathbb{S}| = O(e^{mn})$ which is still exponential in n as m is problem-specific and is independent of the input [2].

A Potential Relaxation

Caution: The following discussion was provided by the student and is entirely based on the *old* definition that does not require that $\forall k : q_0(T_k) > 0$. Later on in the report, we provide a proof for $q_0(T_0^*) > 0$ for the optimal AIFV-2 code (using the *old* definition), so the entire argument has been invalidated. However, it does highlight a subtlety regarding aperiodicity and convergence in Markov chains, so it may be worth reading.

$\forall k : q_0(T_k^*) > 0$ follows trivially from the *new* definition for AIFV-m codes and is introduced later as well (however, this fact is not useful to *this* subsection)

The student *speculates* that it may **not** be the case that $q_0(S_k) > 0$ needs to hold for all k, but rather for all k such that $k > 0$. This is because while it is true that $q_0(T_1) > 0$ is true for the *optimal* AIFV-2 code, there is no constraint that $q_0(T_0) > 0$ (i.e. no requirement that T_0 must have an *assigned* leaf) in the AIFV-2 case — and there does not seem to be a straightforward reason for this fact either.

In [2], it is stated that the original requirement that $\forall k : q_0(S_k) > 0$ must hold for two reasons: (i) this will result in an *ergodic unichain* which has a *unique* stationary distribution, and (ii) some proofs relied on this constraint. Careful observation reveals that no proofs of theorems/lemmas in [2] and [3] concerning the m-state chains relied on $q_0(S_0) = 0$, instead they relied on $\forall k > 0 : q_0(S_k) > 0$ — so (ii) is met even if we relax the constraint to allow $q_0(S_0) = 0$.

Relaxing the constraint may result in the m-state chain no longer being an ergodic unichain; however, we show that despite this fact, the chain will still be guaranteed to have a *unique* stationary distribution.

Allowing $q_0(S_0) = 0$ does not change the fact that there is a single recurrent class. If S_0 communicates with other states, those states will also communicate with S_0 since $\forall k > 0 : q_0(S_k) > 0$ (none of the finite-state paths between any two states in recurrent class are broken by removing the S_0 self-loop, so the class is still *communicating*). So the recurrent class containing S_0 is preserved. However, it may no longer be *ergodic* which raises the possibility of no longer having a *unique* stationary distribution without which the cost of the Markov chain and by extension L_{AIFV} of the corresponding AIFV-m code would no longer be well-defined.

In a finite-state Markov chain, there will always *exist* a stationary distribution π; however, it may not be unique. Uniqueness is guaranteed when the chain has a single irreducible, positive-recurrent communicating class. All states in finite Markov chains are positive recurrent and we already argued that the single pre-existing recurrent class containing S_0 is preserved after removing the S_0 self-loop thereby satisfying all criteria for the existence and uniqueness of the stationary distribution π. Thus, there will still exist a unique stationary distribution π so the cost of the Markov chain and, therefore L_{AIFV}, is still well-defined.

When we allow $q_0(S_0) = 0$, the resulting chain is no longer ergodic because it may not be aperiodic (as our proof of aperiodicity relied on $q_0(S_0) > 0$) which is a condition for ergodicity. However, we showed that there will always exist a unique recurrent class. The only additional property that aperiodicity, and therefore ergodicity, imparts is convergence to the stationary distribution π (by the definition of ergodicity), however, that is not relevant to the MCMC problem.

Note: It has since been pointed out to the student that while the above argument is correct from the perspective of *theorem-proving*, convergence is implicitly assumed by the definition of L_{AIFV} as we assume that the probability of encoding an arbitrary source symbol using a specific tree T_i follows the same distribution as π, which would require convergence in order for the encoding process to assume π asymptotically. Thus, convergence, and therefore aperiodicity, is required for the Markov chain.

Questioning the validity of the reduction

Note: The only distinction between the *old* and *new* definitions of AIFV-m codes is the fact that the latter includes the constraint that *all* leaves must have a codeword assigned to them (i.e. all leaves are *assigned* leaves, and there are no *unassigned* leaves present). This was outlined earlier but is especially important in this subsection, hence the reminder. **The following discussion, claim, and proof use the *old* definition and, thus, deal with *unassigned* leaves.**

It is quite easy to see that the number of AIFV-m codes $\geq |\mathbb{S}|$ (the size of *permissible* Markov chains)—a simple argument for this is that by definition there is *exactly* one type-k state for each equivalence class of type-k coding trees and no other type-k states as outlined earlier. All *permissible* m-state Markov chains follow the condition that $\forall k \in [m] : q_0(S_k) > 0$ to ensure that loss is well-defined.

Claim: For the *optimal* Markov chain to correspond to the *optimal* AIFV-m code, $\forall k \in [m] : q_0(T_k) > 0$ must be true for the *optimal* AIFV-m code (or that equivalence class of AIFV-m codes).
Proof: Suppose the optimal AIFV-m code has at least one coding tree T_k for which $q_0(T_k) = 0$ (it cannot be lower than zero) and therefore violates the constraint (assume that there is only one optimal AIFV-m code for this instance, *uniqueness* is proved later on by [3]). Although we can technically construct a state via the rules of transformation, this state will not be a part of the type-k state set $\mathbb{S}_k$ since $\forall S_k \in \mathbb{S}_k : q_0(S_k) > 0$ and this state has a zero probability of transitioning to S_0. Therefore, the m-state Markov chain that corresponds to the AIFV-m code is not a *permissible* Markov chain and is not even a part of the search space in the instance of the MCMC problem. It follows that the AIFV-m code corresponding to the solution of the MCMC instance cannot be the optimal code (since there exists a code with even lower cost). So the optimal AIFV-m code must abide by the condition $\forall k : q_0(T_k) > 0$ to validate the reduction for *all* instances of the AIFV-m coding problem (this ensures that the Markov chain of the *optimal* AIFV-m code is *permissible* and, therefore, in the search space of the MCMC instance). $\square$

We noted earlier that $q_0(T_1) > 0$ for the optimal AIFV-2 code (as proved in [1]). At first glance it is not obvious that $q_0(T_0) > 0$ for the optimal AIFV-2 code so it seems as if the optimal solution to the MCMC problem may *not* correspond to the optimal AIFV-2 code. However, we can prove that $q_0(T_0) > 0$ with a simple argument provided by Dr. Golin. Assume that the optimal AIFV-2 code $T^* = \langle T_0^*, T_1^* \rangle$ and that we are following the **old** definition (so there can exist *unassigned* leaves).

Claim: $q_0(T_0^*) > 0$
Proof: Assume that we have an arbitrary AIFV-2 code T with $q_0(T_0) = 0$. This means that $q_1(T_0) = 1$, and, thus, $\pi_0 = 0, \pi_1 = 1$. It follows that $L_{AIFV}(T) = \pi_0 \cdot L(T_0) + \pi_1 \cdot L(T_1) = L(T_1)$ so T_0 does not contribute to the cost of T in anyway, regardless of how big or small it may be. The following algebra reveals an interesting fact:

$$\pi_0 = \frac{q_0(T_1)}{q_1(T_0) + q_0(T_1)} = 0 \Longrightarrow \boxed{q_0(T_1) = 0}$$

An alternate proof is as follows:

$$\pi_1 = \frac{q_1(T_0)}{q_1(T_0) + q_0(T_1)} = 1 \Longrightarrow q_1(T_0) = q_1(T_0) + q_0(T_1) \Longrightarrow \boxed{q_0(T_1) = 0}$$

So $q_0(T_1) = 0$, which means T_1 has no *assigned* **leaves**, all the leaves present in T_1 are *unassigned.*

We will now construct a new AIFV-2 code T' using just T_1. Take the sub-tree rooted at the 01-grandchild of root(T_1). Now find the *last* (or lowest depth-wise) master node in the sub-tree, delete the remaining nodes in the sub-tree rooted at that master node (this sub-tree must consist of only internal nodes and *unassigned* leaves, so it does not include codewords).

Now replace the *last* master node with a single *assigned* leaf (so that the original codeword for the source symbol assigned to the *last* master node remains intact). This will be T_1'. Note that by introducing an *assigned* leaf, we have increased $q_0(T_1)$ to no longer be non-zero, so the stationary distribution (we will denote the stationary distribution of T' as π') becomes $\pi_0' > 0, \pi_1' < 1$. That said, all the codewords of T_1 are preserved by T_1' (since the deleted sub-tree contained no codewords) so $L(T_1') = L(T_1)$.

Now take the modified subtree of T_1' rooted at the 01-grandchild of root(T_1'). Delete the 0-child of T_1' and take *its* 1-child (i.e. the 01-grandchild of root(T_1') or the root of the subtree in question) and shift it in place of the deleted 0-child. This is T_0' (replace the old T_0). The average codeword length of T_0', or $L(T_0')$, has to be strictly less than $L(T_1')$ because we just decreased the lengths of all the codewords, coded by the 0-subtree of T_1', by 1 (when we deleted a node and shifted the remaining sub-tree upwards). The entire process of constructing $T' = \langle T_0', T_1' \rangle$ is illustrated below in Figure 5:

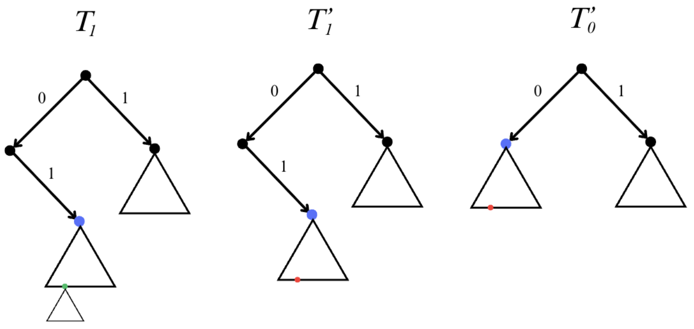


Figure 5: The **green** node in T_1 represents the *last* master node whose sub-tree has no codewords. This sub-tree is deleted and the master node is turned into the **red** leaf in T_1'. The entire sub-tree rooted at the **blue** 01-grandchild of root(T_1') is shifted up by deleting the 0-child of root(T_1') to produce T_0'.

We know that $L(T_1') = L(T_1)$ and $L(T_0') < L(T_1)$ since we deleted the 01-grandchild of root(T_1') to produce T_0' which resulted in all codewords coded by the left sub-tree of T_0 to be shorter by 1 bit. Since T_0' is otherwise identical to T_1', $L(T_0') < L(T_1') = L(T_1)$. Since $L_{AIFV}(T) = L(T_1)$, some algebra reveals the following:

$$L_{AIFV}(T') = \pi_0' \cdot L(T_0') + \pi_1' \cdot L(T_1') < \pi_0' \cdot L(T_1) + \pi_1' \cdot L(T_1') = \pi_0' \cdot L(T_1) + \pi_1' \cdot L(T_1) = L(T_1) = L_{AIFV}(T)$$

so $L_{AIFV}(T') < L_{AIFV}(T)$.

There is one edge case to consider—what if the 01-subtree of root(T_1) has *no* master nodes or assigned leaves (i.e. no source symbols are assigned to codewords prefixed by 01 in the T_1 coding tree)? This would mean that none of the modifications made to $L(T)$ and $L(T_1)$ affect the cost as none of the codewords are in the left sub-tree (all of them are in the right) which means $L(T_0') = L(T_1') = L(T_1)$ so $L_{AIFV}(T') = L_{AIFV}(T)$.

To show that there exists a better code for this case, we can perform a similar procedure. As in the previous case, T_1 has only master nodes since $q_0(T_1) = 0$ (the previous analysis will apply here). Take the sub-tree rooted at the 1-child of root(T_1) and find the *last* master node and replace it (and its sub-tree) with an *assigned* leaf. This tree is now T_1'. Take the sub-tree rooted at the 1-child of root(T_1') and make this T_0'. This construction process is illustrated by Figure 6.

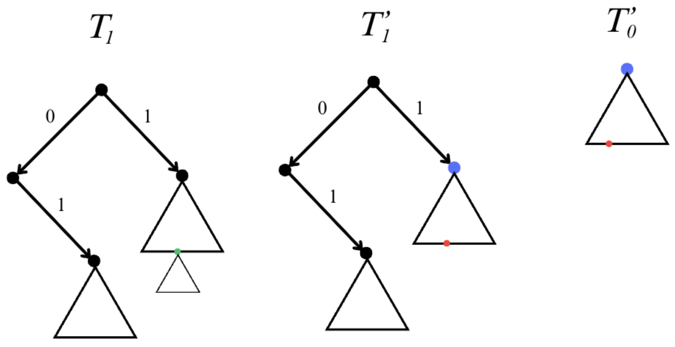


Figure 6: The **green** node in T_1 represents the *last* master node whose sub-tree has no codewords. This sub-tree is deleted and the master node is turned into the **red** leaf in T_1'. The entire sub-tree rooted at the **blue** 1-child of root(T_1') is made T_0'.

We know that in this edge-case $L_{AIFV}(T) = L(T_1)$ as in the previous case as the stationary distribution is $\pi_0 = 0, \pi_1 = 1$. Our changes have resulted in $L(T_1) = L(T_1')$ and $L(T_0') < L(T_1)$ since we remove the 1-edge from root(T_1') making all its codewords shorter by 1 bit. So, $L_{AIFV}(T') < L_{AIFV}(T)$. So for both cases, we have shown that when $q_0(T_0) = 0$, the T cannot be an optimal AIFV-2 code as we can always construct a better code. □

Since $q_0(T_0^*) > 0$ it follows that $\forall i : q_0(T_i^*) > 0$ so the optimal AIFV-2 code will have a 2-state Markov chain that corresponds to it. Thus, an optimal solution to the MCMC problem will present a valid and optimal AIFV-2 code (as per the *old* definition of AIFV-m codes). So for $m = 2$, the reduction is valid.

However, we cannot make the same argument for AIFV-m for $m > 2$ generally, in which case the reduction **may not be valid**. This is where the *new* definition for AIFV-m codes comes in. Any *finite* binary must have leaves. And the new constraint of *each* leaf being assigned a source symbol forces every coding tree in an AIFV-m code to have at least one symbol coded by a leaf, which controls the probability of going back to S_0 or T_0. Thus, $\forall k \in [m] : q_0(T_k) > 0$. Note that this proves that $\forall k \in [m] : q_0(T_k) > 0$ for *any* AIFV-m tree (since the argument is made for any finite AIFV-m tree), and not just the optimal tree which is a *significantly stronger* condition than is required to validate the reduction.

Note: Since every AIFV-m code follows the *new* definition, the Markov chain corresponding to *any* AIFV-m code will be *permissible*. The number of equivalence classes of AIFV-m codes is precisely the number of *permissible* Markov chains (i.e. $|\mathbb{S}|$).

There can no longer be an equivalence class of AIFV-m codes whose representative m-state Markov chain is *impermissible.*

Evidently, the new definition not only allows the validity of the reduction to generalize well (for all m) but the proof is also much simpler. The previous proof using the *old* definition for the AIFV-2 case was not necessary, but the student thought it might be worth putting down in writing to help build intuition. **Moreover, the previous discussions highlight some of the issues in the old definition that complicated the analyses.**

Other problems that have been noted to reduce to the MCMC problem (e.g. AIVF coding, finite state channel coding, etc.) also obey the $\forall k : q_0(S_k) > 0$ constraint. The reduction, of any problem, to the MCMC problem is only valid if *at least one* optimal solution to *every* instance of the original problem obeys the constraint $\forall k : q_0(S_k) > 0$—otherwise, the MCMC solution may not correspond to the *optimal* solution for the original problem.

Geometric Tools: Associated Hyperplanes and Polytopes

Note: A *polygon* is a 2-dimensional shape defined by straight lines (like a square, rectangle, pentagon, etc.). A *polyhedron* is a 3-dimensional shape defined by planes (like cubes, prisms, etc.). A *polytope* is the n-dimensional generalization of polygons and polyhedra — it is a geometric object with *flat* sides (defined by $(n-1)$-dimensional hyperplanes). The flat sides will be be $(n-1)$-polytopes (think about it, polyhedral objects have polygons as faces; for example, rectangular prisms have rectangles as faces).

A *convex* polytope has the additional property that it the polytope, and the points inside it, form a convex set contained in n-dimensional Euclidean space $\mathbb{R}^n$. Knowing these terms will be useful when visualizing the convex polytopes introduced in this section.

In [2], type-k states are mapped into type-k **hyperplanes** in $\mathbb{R}^m$ using the $m+1$ values that define the state (i.e. the loss $L(S_i)$ and the m transition probabilities from the state $\forall k \in [m] : q_k(S_i)$). This gives rise to some very interesting and useful geometric interpretations of the problem and its components.

Define the type-k hyperplanes $f_k : \mathbb{R}^{m-1} \times \mathbb{S}_k \to \mathbb{R}$ as follows:

$$f_0(\mathbf{x}, S_0) = L(S_0) + \sum_{j=1}^{m-1} q_j(S_0) \cdot x_j$$

$$\forall k > 0 : f_k(\mathbf{x}, S_k) = L(S_k) + \sum_{j=1}^{m-1} q_j(S_k) \cdot x_j - x_k$$

A reformulation of f_k for all $k > 0$ that the student found useful for developing *minor* intuition is as follows:

$$\forall k > 0 : f_k(\mathbf{x}, S_k) = L(S_k) + (\sum_{j \neq k, j \neq 0} q_j(S_k) \cdot x_j) + q_k(S_k) \cdot x_k - x_k$$

$$\forall k > 0 : f_k(\mathbf{x}, S_k) = L(S_k) + (\sum_{j \neq k, j \neq 0} q_j(S_k) \cdot x_j) + (q_k(S_k) - 1) \cdot x_k$$

Since $\sum_{i \in [m]} q_i(S_k) = 1$, it follows that

$$1 - q_k(S_k) = (\sum_{i \in [m]} q_i(S_k)) - q_k(S_k) = q_{\text{any state other than } k}(S_k)$$

and plugging that into the previous hyperplane equation gives us the following:

$$\boxed{\forall k > 0 : f_k(\mathbf{x}, S_k) = L(S_k) + (\sum_{j \neq k, j \neq 0} q_j(S_k) \cdot x_j) - q_{\text{any state other than } k}(S_k) \cdot x_k}$$

For all $\mathbf{x} \in \mathbb{R}^{m-1}$, define $g_k : \mathbb{R}^{m-1} \to \mathbb{R}$ and $S_k : \mathbb{R}^{m-1} \to \mathbb{S}_k$.

$$g_k(\mathbf{x}) = \min_{S_k \in \mathbb{S}_k} f_k(\mathbf{x}, S_k)$$

$$S_k(\mathbf{x}) = \arg \min_{S_k \in \mathbb{S}_k} f_k(\mathbf{x}, S_k)$$

and Markov chain

$$\mathbf{S}(\mathbf{x}) = (S_0(\mathbf{x}), S_1(\mathbf{x}), \ldots, S_{m-1}(\mathbf{x}))$$

Note: Based on the definitions so far, $\mathbf{S}(\mathbf{x})$ refers to the m-state Markov chain with each state S_i corresponding to the hyperplane that produces the lowest value when evaluated at $\mathbf{x}$ (i.e. $g_i(\mathbf{x})$, so by definition $S_i \leftarrow S_i(x)$ as defined above. This is possible, because each state is *self-contained* because the transition probabilities and loss value that define the state are independent of other states so we can select all kinds of combinations of states from each type-k state set $\mathbb{S}_k$ to produce all sorts of *permissible* m-state Markov chains.

By definition, it follows that:

$$\forall \mathbf{x} \in \mathbb{R}^{m-1} : \forall k \in [m] : g_k(\mathbf{x}) = f_k(\mathbf{x}, S_k(\mathbf{x}))$$

and for all $\mathbf{x} \in \mathbb{R}^{m-1}$:

$$h(\mathbf{x}) = \min_{k \in [m]} g_k(\mathbf{x}) = \min_{k \in [m]} \min_{S_k \in \mathbb{S}_k} f_k(\mathbf{x}, S_k)$$

The hyperplanes f_k, g_k were introduced in [2]. f_k maps states to hyperplanes in $\mathbb{R}^m$ as stated earlier. g_k is the *lower envelope* of all the type-k hyperplanes f_k. S_k produces the type-k state whose hyperplane evaluates to the lowest value at $\mathbf{x} \in \mathbb{R}^{m-1}$, and, therefore, $\mathbf{S}(\mathbf{x})$ maps a point $\mathbf{x} \in \mathbb{R}^{m-1}$ to a Markov chain $\mathbf{S}(\mathbf{x}) \in \mathbb{S}$. Just like how g_k is the lower envelope of all f_k, $h(\mathbf{x})$ also forms a lower envelope over all g_k (across all values of $k \in [m]$). Since $g_k(\mathbf{x})$

and $h(\mathbf{x})$ are lower envelopes of hyperplanes, they are the upper surface of polytopes in $\mathbb{R}^m$. Now we can define the Markov Chain Polytope in $\mathbb{R}^m$ as follows:

$$\mathbb{H} = \{(\mathbf{x}, y) \in \mathbb{R}^{m-1} \times \mathbb{R} \mid 0 \leq y \leq h(\mathbf{x})\}$$

The student would like to define k more polytopes to aid with future explanations:

$$\forall k \in [m] : \mathbb{G}_k = \{(\mathbf{x}, y) \in \mathbb{R}^{m-1} \times \mathbb{R} \mid 0 \leq y \leq g_k(\mathbf{x})\}$$

These are the polytopes whose upper surfaces are determined by g_k. This gives us an alternate way to define $\mathbb{H}$ as follows:

$$\mathbb{H} = \bigcap_{k \in [m]} \mathbb{G}_k$$

Note: Note that since the definitions of h and g_k depend on the minimization function which means that all of these functions are *concave*. So by definition, $\mathbb{H}$ and $\mathbb{G}_k$ are convex as the region contained by a concave function is *always* a convex set. Thus, it follows that each of these polytopes are *convex polytopes*.

Each of these convex polytopes (e.g. $\mathbb{H}/\mathbb{G}_k$) are, as the student likes to call them, *mountain-like* with flat bases (since the polytope does not extend below the $y = 0$ hyperplane). These sketches were immensely useful to the student when attempting to visualize the geometry of the problem as well as how the *iterative* algorithm worked. They may be of help to the reader and has, thus, been included in this report.

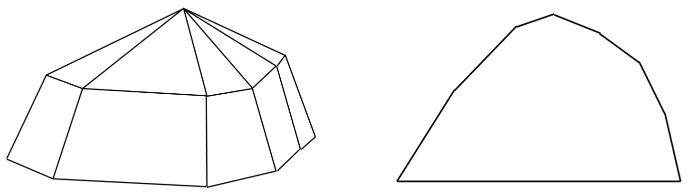


Figure 7: Example of $\mathbb{H}/\mathbb{G}_k$ for $m = 3$ (left) and $m = 2$ (right)

Caution: While none of the claims regarding $\mathbb{G}_k$ are inaccurate it is worth noting that in some cases, like for $m = 2$, some sides of the polytope are not as smooth as above. Here is an example:

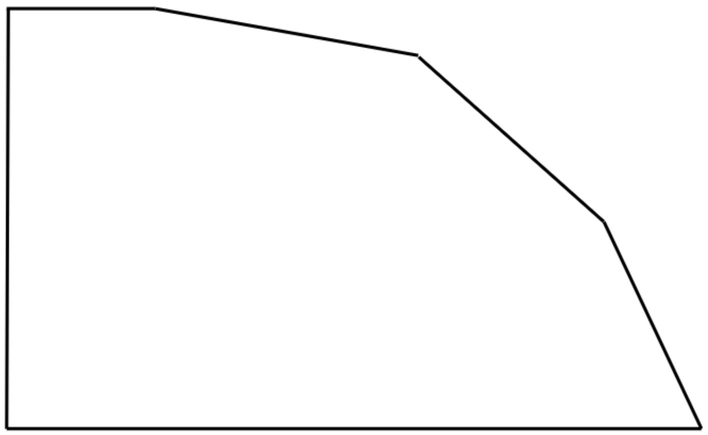
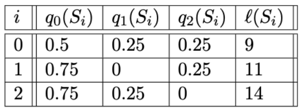


Figure 8: Example of $\mathbb{G}_1$ for $m = 2$

Here are some images from [2] follow to help visualize the polytopes for real-world instances.



i	$q_0(S_i)$	$q_1(S_i)$	$q_2(S_i)$	$\ell(S_i)$
0	0.5	0.25	0.25	9
1	0.75	0	0.25	11
2	0.75	0.25	0	14

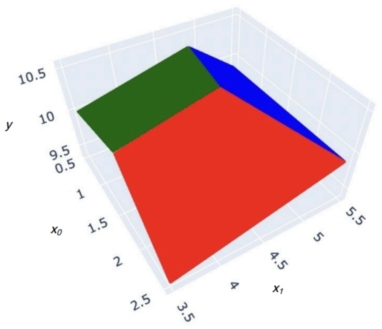


Figure 9: Example of the corresponding type-k hyperplanes for a 3-state Markov chain from [2]

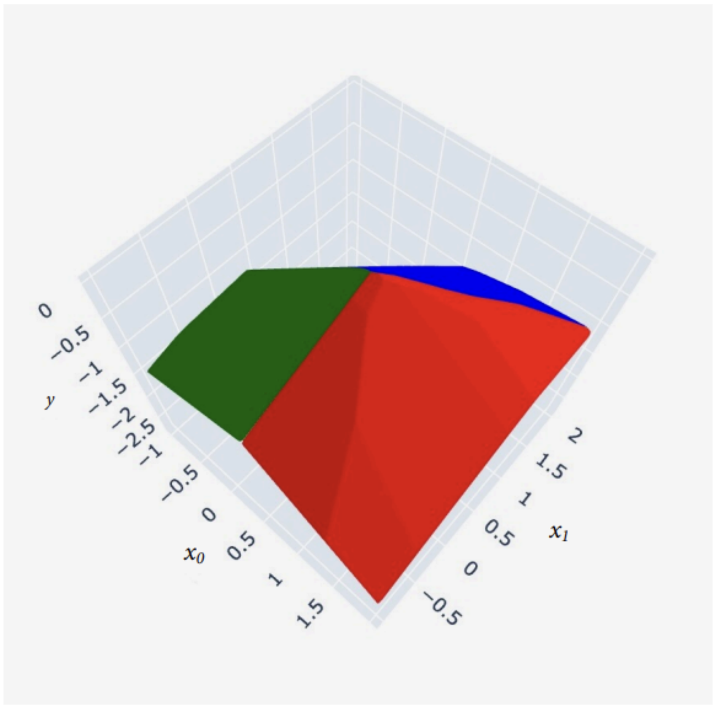


Figure 10: Example of the corresponding type-k *envelopes* g_k to form $\mathbb{H}$ for $m = 3$ from [2]

While the following construction is unnecessary, it helps formalize intuition (slightly). The type-k hyperplanes are defined for each state S_k (since they accept the state as an argument to f_k). Let $\mathbb{F}_k$ denote the set of type-k hyperplanes.

Every type-k state S_k has a *unique* type-k hyperplane (the map f_k is injective), and every type-k hyperplane must have a *unique* state $S_k \in \mathbb{S}_k$ associated with it which is determined by the coefficients of the hyperplane (and, therefore, f_k surjective). This means that there exists a bijection between the type-k states and type-k hyperplanes:

$$\forall k \in [m] : \mathbb{S}_k \iff \mathbb{F}_k$$

An alternate explanation for this is by investigating the **invertibility** of f_k—this is easy to see as we can simply take the coefficients of the polynomial to compute the loss and transition probabilities of S_k.

Recall that type-k states represent equivalence classes of type-k AIFV-m coding trees as per the reduction. Since there is a one-to-one correspondence between type-k hyperplanes and type-k states, we can consider type-k hyperplanes as representatives for these equivalence classes as well. Similar analogies exist for other problems that reduce to the MCMC problem.

Note: The following sections will briefly outline the primary arguments of each of the following papers. Proofs of any newly-introduced theorems/lemmas will *not* be included in the report. We will detail the general *iterative algorithm* to find optimal AIFV-m codes (first proposed by [11]) starting with a formulation specific for $m = 2$ and generalizing shortly afterwards. We will also formally associate properties of the AIFV-m codes with terms that were loosely-defined earlier in the report such as the *property* x_T of the AIFV code T.

Binary Search for AIFV-2 Codes (2023)

Note: In this section, we will specify the details of the *iterative algorithm* for $m = 2$.

The Iterative Algorithm for $m = 2$

Algorithm: Iterative Algorithm for $m = 2$ described by [8], [10]

```
Input: Source Symbols $\mathcal{X}$ and Distribution $P(\mathcal{X})$
Output: Minimum Cost AIFV-2 Code
Initialization: $t \leftarrow 0$, $C^{(0)} \leftarrow 2 - \log_2(3), C^{(-1)} \leftarrow 1$;
while $C^{(t)} \neq C^{(t-1)}$ do
    $t \leftarrow t + 1$;
    $T_0^{(t)} \leftarrow \arg\min_{T_0 \in \mathcal{T}_0(n)} \{L(T_0) + C^{(t-1)} \cdot q_1(T_0)\}$;
    $T_1^{(t)} \leftarrow \arg\min_{T_1 \in \mathcal{T}_1(n)} \{L(T_1) - C^{(t-1)} \cdot q_0(T_1)\}$;
    $C^{(t)} \leftarrow \frac{L(T_1^{(t)}) - L(T_0^{(t)})}{q_1(T_0^{(t)}) + q_0(T_1^{(t)})}$
Set $C^* \leftarrow C^{(t)}$;
return $\langle T_0^{(t)}, T_1^{(t)} \rangle$;
```

In each iteration, the local optimization step tries to find the trees that minimize $L(T_0) + C^{(t-1)} \cdot q_1(T_0)$ and $L(T_1) - C^{(t-1)} \cdot q_0(T_1)$. Yamamoto et al. referred to these quantities as *costs* but the student finds this interpretation confusing (this is just a side node and does not affect the analysis); however, they will not be referred to as *costs* in this report.

Evidently, the algorithm starts with an arbitrary value for $C^{(0)}$, increments t by 1, solves the local optimization problem of finding $T_0^{(t+1)}, T_1^{(t+1)}$, computes $C^{(t+1)}$ and repeats this process until $C^{(t)} = C^{(t+1)}$ for some t (i.e. until that term converges) at which point the most recent pair of trees is returned as the optimal coding pair. Some of these steps seem a little bit arbitrary, so this paper seeks to provide a geometric interpretation of the algorithm which is easier to understand.

Note, that the correctness and termination (in a finite, but unbounded) number of steps was proved already. This paper also provides a new algorithm that has a *weakly* polynomial number of steps.

Reinterpreting the Algorithm Geometrically

For now, the value $C^{(t)}$ can be thought of as the property x_T of the code T, which we will define in the *next* main section of the report, (the exact meaning intended by the authors for $C^{(t)}$ is irrelevant to our discussion) — we will formally explain $C^{(t)}$ shortly.

Note that the update step that calculates $C^{(t)}$ looks very much like a line intersection between the following two lines, namely:

$$f_{T_0} = L(T_0) + x \cdot q_1(T_0)$$

$$g_{T_1} = L(T_1) - x \cdot q_0(T_1)$$

which forces $C^{(t)}$ to become the x-coordinate of the line intersection. The meaning of the lines, and by extension the quantities being minimized in the original algorithm, are still up for debate. But, a closer look reveals that these lines are the type-k hyperplanes for $m = 2$.

This means that the local optimization step (during the ith iteration of the while loop) selects the type-k coding tree whose corresponding hyperplane evaluates to the *lowest* value at $x = C^{(i)}$ — so the optimization step is minimizing over hyperplanes (sort of). [1] defines the following envelopes:

$$E_0(x) = \min_{T_0 \in \mathcal{T}_0(n)} f_{T_0}(x) = \min_{T_0 \in \mathcal{T}_0} \{L(T_0) + x \cdot q_1(T_0)\}$$

$$E_1(x) = \min_{T_1 \in \mathcal{T}_1(n)} g_{T_1}(x) = \min_{T_1 \in \mathcal{T}_1} \{L(T_1) - x \cdot q_0(T_1)\}$$

where $\mathcal{T}_k(n)$ is the set of *all* possible type-k coding trees. It is not hard to see that $E_0(x) = g_0(x)$ and $E_1(x) = g_1(x)$ where g_0, g_1 were the envelopes defined in the previous section. This means that the images below correspond to the polytopes $\mathbb{G}_0$ and $\mathbb{G}_1$.

As mentioned earlier, the convex polytopes $\mathbb{G}_0, \mathbb{G}_1$ for any AIFV-2 coding problem always have vertical sides on one side (as illustrated in Figure 8). Moreover, E_0 is *non-decreasing*, since $q_1(T_0) \in [0, 1]$, and E_1 is *non-increasing*, since $-q_0(T_1) \in [-1, 0)$.

Note: $-q_0(T_1) \in [-1, 0)$ because of the *new* definition of AIFV-m codes. This requires that all leaves, for *any* type-k coding tree, must be *assigned* a source symbol forcing $\forall k \in [m] : q_0(T_k) > 0$ for *any* AIFV-m code. This means that $q_0(T_k) = 0$ is **impossible** limiting the slope range of type-1 hyperplanes (lines) to $-q_0(T_k) \in [-1, 0)$. This is not the case for type-0 hyperplanes.

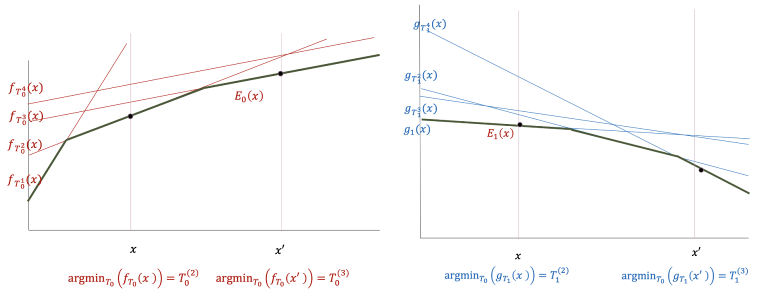


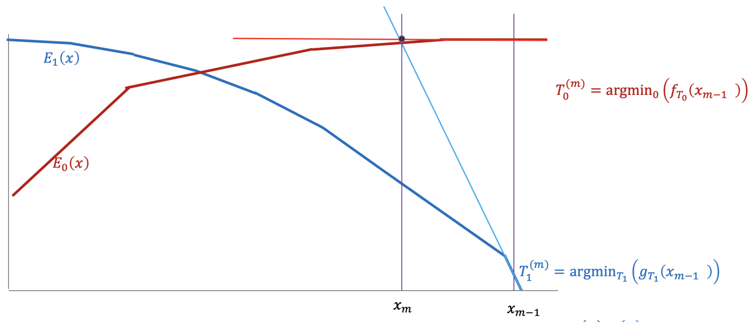
Figure 11: E_0 (left) and E_1 (right). Image taken from a presentation of [1] by its authors

The optimization step of *minimizing over hyperplanes* is equivalent to plotting all hyperplanes and then finding the hyperplane that intersects the vertical bar $x = C^{(i)}$ first. This is equivalent to evaluating $E_k(C^{(i)})$ and finding the tree whose hyperplane produces this value (there has to exist one by definition of E_k). This is equivalent to computing the following:

$$T_0(x) = \arg \min_{T_0 \in \mathcal{T}_0(n)} \{f_{T_0}(x)\}$$

$$T_1(x) = \arg \min_{T_1 \in \mathcal{T}_1(n)} \{g_{T_1}(x)\}$$

Once the corresponding type-0 and type-1 coding trees $(T_0^{(i)}, T_1^{(i)})$ are found (via their hyperplanes), we compute the x-coordinate of the line intersection to produce $C^{(i+1)}$ and this process is repeated until convergence. An image illustrating this process has been provided below:



$E_1(x)$
$E_0(x)$
$T_0^{(m)} = \text{argmin}_0 \left(f_{T_0}(x_{m-1}\ )\right)$
$T_1^{(m)} = \text{argmin}_{T_1} \left(g_{T_1}(x_{m-1}\ )\right)$
x_m
x_{m-1}

Figure 12: One iteration of the original algorithm. First, $g_0(x_{m-1})$ and $g_1(x_{m-1})$ are evaluated to find the type-0/type-1 hyperplanes, then we compute the line intersection to produce x_m and get ready for the next iteration. Image taken from a presentation of [1] by its authors.

Note: The process of evaluating $T_k(C^{(i)}) \mid \forall k \in \{0,1\}$ is done by running the $O(n^3)$ DPs proposed by [4] for tree construction (same procedure used to solve the local optimization step in the *general iterative algorithm* as before).

Unique Intersections and the Property x_T

At the end of the previous subsection, re-framing the *optimization step* of the algorithm geometrically revealed that the property $x_{T^{(i)}}$ or $C^{(i)}$ is the x-coordinate of the line intersection. This is because we used the x-coordinate to produce a new AIFV-2 code (by evaluating $E_k(C^{(i)}) \mid \forall k \in \{0,1\}$) and then computing the x-coordinate of the new intersection and repeating the process—this is exactly how general algorithm works as described in the first section.

That said, there is no guarantee that every (type-0, type-1) line pair must have a *unique* intersection, which is a property of x_T that we noted earlier (every code T has a unique x_T). Here is a simple proof below:

Claim: The lines f_{T_0}, g_{T_1} for *any* valid AIFV-2 code, we will always have a *unique* intersection (x,y).
Proof: We know that we can mix and match any two trees (one type-0 and one type-1 tree) to produce any valid AIFV-2 code. The type-k hyperplanes are 1-dimensional lines for $m = 2$. So we need to argue that any combination of (type-0, type-1) line pairs will result in a *unique* intersection.

Assume that there exists a (type-0, type-1) line pair that does not have a *unique* intersection. So the lines either have (i) no intersection or (ii) infinite intersections—in both cases this means the lines are parallel. We know that parallel lines must have the same slopes.

However, we also noted earlier that the slope range for type-0 lines $q_1(T_0) \in [0, 1]$ and the slope range for type-1 lines $-q_0(T_1) \in [-1, 0)$. Since $[0, 1] \cap [-1, 0) = \emptyset$, there exists no (type-0, type-1) line pair such that both lines have a common slope and are parallel. This means that every such line pair must have a *unique* intersection. □

This proof shows that every AIFV-2 code T has a *unique* intersection (and thus, a unique x-coordinate to rely on for x_T when the algorithm requires it). Notice that while one code may have a *unique* intersection, multiple codes may have the same x-coordinate for their intersections highlighting the many-to-one relationship described earlier.

Note: The *many-to-one* relationship was not prescribed by the original creators of the algorithm. In fact, the property x_T was not mentioned either—this was a construct created by the student for clarity (although this may be more confusing than intended for some edge cases). The original algorithm relied on $C^{(i)}$ and it relied on a *unique* value for $C^{(i)}$, which the authors of [1] later identified as the x-coordinate of the line intersection.

However, the *old* definition of AIFV-2 codes did not always guarantee a *unique* intersection. Recall that we proved that the *optimal* code follows $q_0(T_0) > 0$, so not *all* codes may follow that rule. Thus, in theory, if the algorithm landed on a code with parallel lines (non-intersecting or infinitely intersecting) $C^{(i)}$ may have been undefined. The only way to know for sure is to refer to those papers as the original papers were not a part of the literature review.

Key Lemmas regarding the AIFV-2 Problem

Here are a few key results proved in [1] (the results follow a slightly different ordering in the original paper):

1. E_0 and E_0 must intersect at a *unique* point (x^*, y^*).
   Note: This fact cannot be assumed, even though it was in analyses of the algorithm in some previous papers.

2. The y-coordinate, of the line intersection of f_{T_0} and g_{T_1} is the average code length $L_{AIFV}(T_0, T_1)$.

3. Let (x^*, y^*) be the *unique* intersection of E_0, E_1. Then $\langle T_0(x^*), T_1(x^*)\rangle$ is an *optimal* AIFV-2 code.
   Note: By the previous lemma, the cost of the optimal AIFV-2 code is y^*. This means that the costs of all AIFV-2 codes for this instance must be greater than y^*.

4. $0 \leq x^* \leq 1$

5. $q_0(T_1(x^*)) > 0$
   Note: This proof was done using the *old* definition. We showed that $q_0(T_0(x^*))$ in a previous section using the *old* definition as well. Remember, the *new* definition forces this to be true for *all* type-k coding trees, not just the optimal pair.

6. If x_1, x_2 be two critical points on the piecewise function E_k where $k \in \{0,1\}$, then $|x_1 - x_2| \geq 2^{-2b}$

Proofs for these lemmas are not provided here (as stated earlier); interested readers may seek them out in [1]. That said, the student would like to include one proof that they enjoyed reading for its simplicity and elegance—this is the proof for the second result listed above:

Claim: $L_{AIFV}(T_0, T_1) = f_{T_0}(x') = g_{T_1}(x')$ at the intersection (x', y')
Proof: Assume that f_{T_0} and g_{T_1} intersect at (x', y').

$$L(T_0) + x' \cdot q_1(T_0) = f_{T_0}(x') = g_{T_1}(x') = L(T_1) - x' \cdot q_0(T_1)$$

$$\implies x' = \frac{L(T_1) - L(T_0)}{q_1(T_0) + q_0(T_1)}$$

$$\implies y' = f_{T_0}(x') = L(T_0) + q_1(T_0) \cdot \frac{L(T_1) - L(T_0)}{q_1(T_0) + q_0(T_1)}$$

$$= \frac{L(T_0) \cdot (q_1(T_0) + q_0(T_1)) + q_1(T_0) \cdot (L(T_1) - L(T_0))}{q_1(T_0) + q_0(T_1)}$$

$$= L(T_0) \cdot \frac{q_0(T_1)}{q_1(T_0) + q_0(T_1)} + L(T_1) \cdot \frac{q_1(T_0)}{q_1(T_0) + q_0(T_1)}$$

$$= L(T_0) \cdot \pi_0 + L(T_1) \cdot \pi_1 = L_{AIFV}(T_0, T_1)$$

Note: The intersection is *unique* if and only if $0 < q_1(T_0) + q_0(T_1)$ $\square$

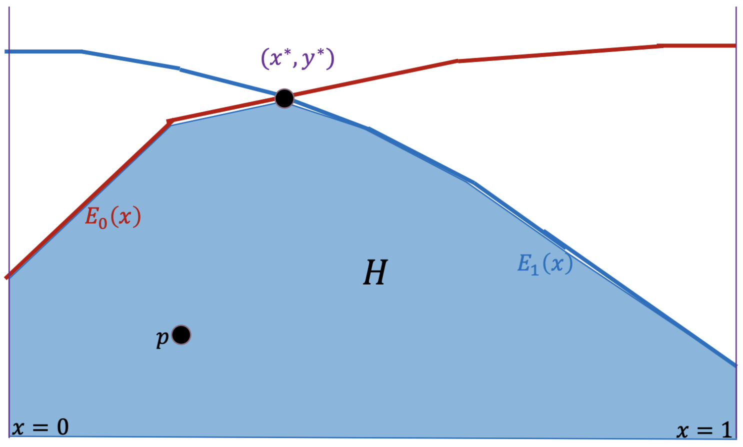


Figure 13: H corresponds to $\mathbb{H}$. Image taken from a presentation of [1] by its authors
.

The Binary Search Algorithm

```
Algorithm: Binary Search Algorithm for $m = 2$ described by [1]
  Initialization: $l, r \leftarrow 0, 1$, $\epsilon_0 \leftarrow 2^{-2b}$;
  while $r - l \geq \epsilon_0$ do
      mid $\leftarrow \frac{l+r}{2}$;
      $e_0 = E_0(\text{mid}) = f_{T_0(\text{mid})}(\text{mid})$;
      $e_1 = E_1(\text{mid}) = g_{T_1(\text{mid})}(\text{mid})$;
      if $e_0 < e_1$ then
          $l =$ mid;
      else
          $r =$ mid;
  // Now find $x^* \in [l, r]$;
  Construct trees $T_0(l), T_1(l), T_0(r), T_1(r)$ and their corresponding lines.;
  In $O(1)$ further time find $x^* \in [l, r]$, the intersection point of $E_0(x)$ and $E_1(x)$;
  return $\langle T_0(x^*), T_1(x^*) \rangle$;
```

Above is the *binary search* algorithm proposed for the AIFV-2 problem based on the lemmas proved above. The main idea is to find the intersection of the envelopes E_0, E_1 (i.e. x^*) since we know that it exists, is unique, and that $T_0(x^*), T_1(x^*)$ is an *optimal* code. We know that it must be in the $[0, 1]$-window, so the algorithm sets the initial search window to $[0, 1]$ using the pointers l, r. Each iteration halves the search space, so after $\log_2 \frac{1}{e_0} = 2b$ steps (where b is the *maximum* number of bits to encode any one input probability from $P(\mathcal{X})$), $r - l = e_0$.

The value e_0 is not arbitrary—the authors of [1] proved that the distance between two critical points on either envelope is bounded from above by 2^{-2b} (this is a lemma listed above). This means that when we exit the while loop, $r - l = 2^{-2b}$ (not just $\leq$) so it contains at most 2 critical points—one of which is the intersection (x^*, y^*). The case where we have 2 critical points occurs exactly when both critical points are at $x = r$ and $x = l$ (i.e. the boundaries of the windows). Otherwise, the intersection is the only critical point in the window. In any case, we can find the intersection x^* in $O(1)$ steps since there a constant number of cases to consider.

We know that computing $E_k(x)$ is an $O(n^3)$ DP. Moreover, we know there are exactly $2b$ iterations (refer to the previous paragraphs) so $O(b) \cdot O(n^3) = O(n^3 b)$. While the local optimization problem runs in the same time as the *general iterative algorithm*, there is also a bound on the number of iterations for the binary search algorithm (this is not the case for the general algorithm; we know that it terminates in a finite number of steps but we do not know anything about the number of steps).

Markov Chain Polytope (2024)

We know from earlier that [2] proposed the general hyperplane definitions for m-state Markov chains. However, [2] proved some interesting properties about the polytope $\mathbb{H}$ as well.

Lemma 3.1

Lemma 3.1 in [2] was one of the main results and unveiled several interesting geometric properties of $\mathbb{H}$ through various sub-lemmas. Let $S = (S_0, \ldots, S_{m-1}) \in \mathbb{S}$ be a *permissible* Markov chain (i.e. $\forall k : S_k \in \mathbb{S}_k$).

Note: $(\mathbf{x}, y) \in \mathbb{R}^m$ is a shorthand denoting that $\mathbf{x} \in \mathbb{R}^{m-1}$ and $y \in \mathbb{R}$.

1. The m $(m-1)$-dimensional hyperplanes $y = f_k(\mathbf{x}, S_k),\ k \in [m]$, intersect at a *unique* point $(\mathbf{x}^{int}, y^{int}) \in \mathbb{R}^m$. Such an *intersection* is called a **distinctly-typed intersection point**.
2. $\forall \mathbf{x} \in \mathbb{R}^{m-1},\ y^{int} = \mathrm{cost}(\mathbf{S}) = \sum_{k=0}^{m-1} f_k(\mathbf{x}, S_k) \cdot \pi_k(\mathbf{S})$.
3. $\forall \mathbf{x} \in \mathbb{R}^{m-1},\ y^{int} \geq \min_{k\in[m]} f_k(\mathbf{x}, S_k)$.
4. $\forall \mathbf{x} \in \mathbb{R}^{m-1},\ y^{int} \geq h(\mathbf{x})$

Some of the components of Lemma 3.1 are very surprising. While it was simple and intuitive to see and prove that there is a unique intersection for every AIFV-2/2-state Markov chain (as shown in the last section), that is not the case for $m > 2$ and yet its true for every *permissible* Markov chain. The proof of this lemma is very interesting but long and is, thus, left out of the report.

Tip: It is worth visualizing the components of Lemma 3.1 for the Binary Case.

Some Corollaries of Lemma 3.1

This section will add the same numbers as mentioned in [2] for clarity and reference:

Corollary 3.2: Let $S = (S_0, \ldots, S_{m-1}) \in \mathbb{S}$.

$$\min_{S\in\mathbb{S}} \mathrm{cost}(S) \geq \max_{\mathbf{x}\in\mathbb{R}^{m-1}} h(\mathbf{x})$$

This follows from (2) and (4) of Lemma 3.1. Corollary 3.2 leads to the following lemma:

Lemma 3.3: If $g_0(\mathbf{x}^*) = g_1(\mathbf{x}^*) = \cdots = g_{m-1}(\mathbf{x}^*) = y^*$ for some $\mathbf{x}^* \in \mathbb{R}^{m-1}$ and $y^* \in \mathbb{R}$, then

$$y^* = h(\mathbf{x}^*) = \mathrm{cost}(S(\mathbf{x}^*))$$

and

$$\min_{S\in\mathbb{S}} \mathrm{cost}(S) = \mathrm{cost}(S(\mathbf{x}^*)) = h(\mathbf{x}^*) = \max_{\mathbf{x}\in\mathbb{R}^{m-1}} h(\mathbf{x}) = \max\{y \mid (\mathbf{x}, y) \in \mathbb{H}\}$$

Proof: By definition of $h(\mathbf{x}),\ h(\mathbf{x}^*) = y^*$. Since $\forall k \in [m] : g_k(\mathbf{x}) = f_k(\mathbf{x}, S_k(\mathbf{x}))$, we can infer that all m hyperplanes $f_k(\mathbf{x}, S_k(\mathbf{x}^*))$ intersect at $(\mathbf{x}^*, y^*) = (\mathbf{x}^*, h(\mathbf{x}^*))$. Therefore, by (1) and (2) of Lemma 3.1, y^* is also the cost of $S(\mathbf{x}^*)$ (remember, this Markov chain is defined by $g_k(\mathbf{x}^*) \mid \forall k \in [m]$). This completes the proof of the first statement.

To prove the second statement we can combine the first statement with Corollary 3.2.

$$h(\mathbf{x}^*) = \text{cost}(S(\mathbf{x}^*)) \geq \min_{S \in \mathbb{S}} \text{cost}(S) \geq \max_{\mathbf{x} \in \mathbb{R}^{m-1}} h(\mathbf{x}) \geq h(\mathbf{x}^*)$$

This is a very nice prove that *squeezes* all the intermediate inequalities into equalities (since the leftmost and rightmost terms are identical). □

Note, that Lemma 3.3 hinges on a crucial **if** conditional—m g_k hyperplanes must intersect at a point $\mathbf{x}^*$. The last paper this report discusses proves that the general iterative algorithm **always** terminates at such a point thereby implying its *existence*. This can be formally written as follows:

Corollary 3.4: There exists $\mathbf{x}^* \in \mathbb{R}^{m-1}$ satisfying the **if** conditional of Lemma 3.3. This $\mathbf{x}^*$ satisfies $(\mathbf{x}^*, h(\mathbf{x}^*)) \in \mathbb{H}$ and $h(\mathbf{x}^*) = \max\{y \mid (\mathbf{x}, y) \in \mathbb{H}\}$.

Linear Programming and Pruning

The results outlined in the previous subsections show that there is a distinctly-typed intersection point on the highest surface of $\mathbb{H}$. Naturally, this gives rise to the following approach to solve the minimum-cost Markov chain problem:

1. Use linear programming to find **a** highest point $(\hat{\mathbf{x}}, \hat{y}) \in \mathbb{H}$.
2. Starting from $\hat{\mathbf{x}}$, find a distinctly-typed intersection point $(\mathbf{x}^*, y^*) \in \mathbb{H}$.
3. Return $S(\mathbf{x}^*)$

The first step of procedure is using linear programming to find *a* highest point in the polytope $\mathbb{H}$. Notice how we are not excluding the possibility that there may be multiple maximal points—**there may very well be a high dimensional *flat* at the top of $\mathbb{H}$ (i.e. a flat surface) instead of a single *vertex***. The linear programming method used in [2] to get to a starting point ($\hat{\mathbf{x}}$,$\hat{y}$) on the *flat* is the famous **Ellipsoid** method.

The ellipsoid algorithm, when given a (problem-specific) *polynomial*-time separation oracle as input, provides an approximate optimal solution to a convex optimization problem in *polynomial time*—the details of the algorithm and its application to this problem have been omitted as the algorithm is described at great length in [2].

Definition: Let $K \subseteq \mathbb{R}^m$ be a closed convex set. A *separation oracle* for K is a procedure that takes input $\mathbf{x} \in \mathbb{R}^m$ and either reports that $\mathbf{x} \in K$ or, if $\mathbf{x} \notin K$ returns a hyperplane that separates $\mathbf{x}$ from K. In other words, the oracle will return $\mathbf{a} \in \mathbb{R}^m$ such that $\forall z \in K : \mathbf{a}^T\mathbf{x} > \mathbf{a}^T\mathbf{z}$.

[2] outlines the fact that $S(\mathbf{x})$ serves as a separation oracle for $\mathbb{H}$. This is proved as Lemma 4.3 in the paper.

Lemma 4.3: Let m be fixed and $\mathbb{H}$ be the Markov chain polytope. Let $\mathbf{z} = (\mathbf{x}, y) \in \mathbb{R}^m$. Then knowing $S(\mathbf{x}) = (S_0(\mathbf{x}), \ldots, S_{m-1}(\mathbf{x}))$ provides a $O(m^2)$ time algorithm for either reporting that $\mathbf{z} \in \mathbb{H}$ or returning a hyperplane that separates $\mathbf{z}$ from $\mathbb{H}$.
Proof: $\mathbf{z} \in \mathbb{H}$ if and only if

$$y \leq h(\mathbf{x}) = \min_{k \in [m]} (\min_{S_k \in \mathbb{S}_k} f_k(\mathbf{x}, S_k)) = \min_{k \in [m]} g_k(\mathbf{x}) = \min_{k \in [m]} f_k(\mathbf{x}, S_k(\mathbf{x}))$$

Thus, knowing $S(\mathbf{x})$ immediately determines whether $\mathbf{z} \in \mathbb{H}$ or not. Furthermore, if $\mathbf{z} \notin \mathbb{H}$ (i.e. $y > h(\mathbf{x})$) let $k' \in [m]$ be an index satisfying $f_{k'}(\mathbf{x}, S_{k'}(\mathbf{x})) = g_{k'}(\mathbf{x}) = h(\mathbf{x})$. Then the hyperplane $y = g_{k'}(\mathbf{x})$ separates $\mathbf{z}$ and $\mathbb{H}$ as it is a *supporting* hyperplane of $\mathbb{H}$ (makes up one of the sides of $\mathbb{H}$) at point $(\mathbf{x}, h(\mathbf{x}))$. □
We mentioned that polynomial-time DPs to compute $S_k(\mathbf{x})$ existed for all $m > 2$, so the definition of the poly-time separation oracle is complete.

After the poly-time ellipsoid algorithm has returned a point $(\hat{\mathbf{x}}, \hat{y})$ that is maximal (i.e. $\forall (\mathbf{x}, y) \in \mathbb{H} : \hat{y} \geq y$), all we have to do is find *a* distinctly-typed intersection and that will be a minimum-cost Markov chain as per Lemma 3.1 and Corollary 3.2.

Note: There may be multiple distinctly-typed intersections (i.e. Markov chains) on the maximal *flat* of $\mathbb{H}$. We need any one of them to solve the MCMC problem since they all have the same cost: $\hat{y}$.

To find a distinctly-typed intersection, [2] introduced the concept of *pruning*. This resulted in "restricted"" versions of the envelopes g_k and more, which we will not detail here. This permitted the introduction of an $O(m)$ *step* algorithm (with poly-time solvable local optimization problems at each step) to produce the minimum-cost Markov chain.

Note: The *pruning* algorithm is only required if there *is* a high-dimensional flat at the top of $\mathbb{H}$. The next paper shows that not only is there a *unique* distinctly-typed intersection at the top of $\mathbb{H}$, but that this intersection is also a *vertex*—so there is only one *highest* point on $\mathbb{H}$.

Proof of Convergence and Uniqueness (2025)

The general iterative algorithm for AIFV-m codes can be re-framed to solve the MCMC problem as follows. Note that $x(S)$ is the x-coordinate of the unique intersection of S's associated hyperplanes, where $S \in \mathbb{S}$.

Iterative Algorithm for MCMC

```
Algorithm: Iterative Algorithm for MCMC
  Initialization: Let $S^{(0)} \in \mathbb{S}$ be an arbitrary permissible Markov chain;
  $p_0 \leftarrow x(S)$; $S^{(1)} \leftarrow S(p_0)$; $p_1 \leftarrow x(S^{(1)})$;
  $i \leftarrow 1$;
  while $p_i \neq p_{i-1}$ do
      $i \leftarrow i + 1$;
      $S^{(i)} \leftarrow S(p_{i-1})$;
      $p_i \leftarrow x(S^{(i)})$;
  return $S(p_i)$;
```

It is an identical algorithm based on a different "coordinate system," as noted by prior authors.

Key Results

The proofs provided in [3] are rather long so they have been left out of the report. The key results have been summarized below:

1. The iterative algorithm **terminates** in a finite number of unbounded steps and will *always* produce a minimum-cost Markov chain upon termination.
2. There exists only one *maximal* distinctly-typed intersection in $\mathbb{H}$. Moreover, this intersection is a *vertex*.

1. was already proved by earlier authors. [3] provides a simpler proof of correctness that covers all cases and does not rely on assumptions, as previous proofs did. Careful observation reveals that the proof relies on the fact that $\forall i > 0 : q_0(S_i)$ (and, thus, the proof allows $q_0(S_0) = 0$, although this has other consequences as outlined earlier). Moreover, it is also revealed that each iteration of the iterative algorithm deals with a Markov chain $S^{(i)}$ such that the cost is *non-increasing*—this is formally stated as $\forall i > 0 : c_{i-1} \geq c_i$.

The proof of 2. is quite simple but relies on some definitions that were defined for the proof of 1. and has, thus, been omitted from the report as well.

Future Directions

The student was made aware of the evolving structure of the AIFV-m code trees while producing this report — thus, he is interested in understanding those changes, as well as the reasoning behind them, in greater detail. This will act as a *minor* extension to this literature review.

There are several open problems to explore:

1. What is the *exact* bound on the number of steps in the iterative algorithm?
2. Will the distinctly-typed intersection lie in the $[0, 1]^{m-1}$ hypercube for all m?
3. Can the progress on AIFV-m coding problems be used to solve problems regarding finite-state channel coding?

As discussed earlier, the iterative algorithm has been proven to terminate in finitely many steps but there is no better bound than the exponential one when the iterative algorithm performs brute force search throughout the code space. There is speculation that a polynomial bound exists, so attempting to find and prove such a bound would be one direction to pursue. Moreover, while it was proved for $m = 2$ that $x^* \in [0, 1]$, this has not been proved for $m > 2$, and is worth exploring.

Another direction is to explore the parallels between finite state channel coding and AIFV-m coding to see if the former can be solved using techniques used to solve the latter, including but not limited to the reduction from the finite state channel coding problem to the MCMC problem as proven by [17].

References

[1] M. Golin and E. Harb. A polynomial time algorithm for constructing optimal binary aifv-2 codes. IEEE Transactions on Information Theory, 69(10):6269–6278, 2023. doi: 10.1109/TIT.2023.3287587.

[2] M. J. Golin and A. J. L. Patupat. The markov-chain polytope with applications. arXiv:2401.11622, 2024

[3] R. H. Dolatabadi, M. J. Golin, and A. Zamani. Constructing minimum-cost Markov chains with the iterative algorithm. [INCOMPLETE]

[4] M. J. Golin and E. Harb. Speeding up the AIFV-2 dynamic programs by two orders of magnitude using range minimum queries. Theor. Comput. Science., 865: 99–118, 2021. doi: 10.1016/j.tcs.2021.02.040.

[5] H. Yamamoto and X. Wei, "Almost instantaneous FV codes," in Proc. IEEE Int. Symp. Inf. Theory, Jul. 2013, pp. 1759–1763.

[6] D. A. Huffman, "A method for the construction of minimum-redundancy codes," Proc. IRE, vol. 40, no. 9, pp. 1098–1101, Sep. 1952.

[7] H. Yamamoto and X. Wei, "Almost instantaneous FV codes," in Proc. IEEE Int. Symp. Inf. Theory, Jul. 2013, pp. 1759–1763.

[8] W. Hu, H. Yamamoto, and J. Honda, "Worst-case redundancy of optimal binary AIFV codes and their extended codes," IEEE Trans. Inf. Theory, vol. 63, no. 8, pp. 5074–5086, Aug. 2017.

[9] R. Fujita, K.-I. Iwata, and H. Yamamoto, "On a redundancy of AIFV-m codes for m=3,5," in Proc. IEEE Int. Symp. Inf. Theory (ISIT), Jun. 2020, pp. 2355–2359.

[10] H. Yamamoto, M. Tsuchihashi, and J. Honda, "Almost instantaneous fixed-to-variable length codes," IEEE Trans. Inf. Theory, vol. 61, no. 12, pp. 6432–6443, Dec. 2015.

[11] H. Yamamoto and K.-I. Iwata, "An iterative algorithm to construct optimal binary AIFV-m codes," in Proc. IEEE Inf. Theory Workshop (ITW), Nov. 2017, pp. 519–523

[12] R. Fujita, K.-I. Iwata, and H. Yamamoto, "An optimality proof of the iterative algorithm for AIFV-m codes," in Proc. IEEE Int. Symp. Inf. Theory (ISIT), Jun. 2018, pp. 2187–2191.

[13] R. Fujita, K.-I. Iwata, and H. Yamamoto, "An iterative algorithm to optimize the average performance of Markov chains with finite states," in Proc. IEEE Int. Symp. Inf. Theory (ISIT), Jul. 2019, pp. 1902–1906.

[14] K.-I. Iwata and H. Yamamoto, "A dynamic programming algorithm to construct optimal code trees of AIFV codes," in Proc. Int. Symp. Inf. Theory Appl. (ISITA), Oct. 2016, pp. 641–645.

[15] M. Golin and E. Harb, "Speeding up the AIFV-2 dynamic programs by two orders of magnitude using range minimum queries," Theor. Comput. Sci., vol. 865, pp. 99–118, Apr. 2021.

[16] Ken-ichi Iwata and Hirosuke Yamamoto. Aivf codes based on iterative algorithm and dynamic programming. In 2021 IEEE International Symposium on Information Theory (ISIT), pages 2018–2023. IEEE, 2021.

[17] Ken-Ichi Iwata and Hirosuke Yamamoto. Joint coding for discrete sources and finite-state noiseless channels. In 2022 IEEE International Symposium on Information Theory (ISIT), pages 3327–3332. IEEE, 2022.

[18] Ken-Ichi Iwata, , Kengo Hashimoto, Takahiro Wakayama, and Hirosuke Yamamoto. AIFV codes allowing 2-bit decoding delays for unequal bit cost. In 2024 IEEE International Symposium on Information Theory (ISIT'24), 2024.