# Trellis Representation and its Complexity

(Draft Version)

MuTsing

Apr. 8th 2025 $\sim$ Some Day. 2025

## 1 Preliminaries of Coding Theory

### 1.1 A Simple Model for Message Sending

We have some message, for example "The password for my credit card is 123456", want to share with a receiver. If we send the message directly, everone who stole this message will have the information. We can certainly choose some special method to encode it like "My password is ABCDEF" and provided the receriver know that $1 \leftrightarrow A, ..., 6 \leftrightarrow F$.

These procedures are named as encoding, decoding. However, in real-world, there is another problem in sending message.

$$Sender \longrightarrow Encoding \longrightarrow Message \longrightarrow Decoding \longrightarrow Receiver$$

$$123456 \longrightarrow ABCDEF \longrightarrow ABCDEF \longrightarrow ABCDEF \longrightarrow 123456$$

We introduce neccessary definitions about the above concepts.

**Definition 1** (Alphabet)**.** The possible symbols in our encoded message form an **alphabet**.

For example, we can use binary digits to represent information in computers; here, the digits 0 and 1 are the elements of the alphabet.

**Definition 2** (Sentence)**.** A finite length tuple is named the sentence (string) over the alphabets, i.e. $s := A^n$ for a length n string.

### 1.2 Message Sending with Errors

Assume we want to sent the encoded binary sentence 1111, in the real-world, errors may occur as the sending relies on some physical device. For example, we have probability 0.9 to send a digit correctly, i.e. $1 \to 1, 0 \to 1$. Then, the probability of the sending to be perfectly correct is: $0.9^4 = 06561$, which is some kinda of unstable already.

So we want to make it more stable. A good intuitive idea is we send the message twice, or send an extended version message like (1111|1111) with a copy. But we cannot identity which location is wrong. So we cannot copy them directly[1]

[1] Or other reasons to introduce the additive strategy.

**Definition 3** (Hamming Distance)**.** We can introduce the similiarity of two strings. We defien the **hamming distance** of two same length string as the number of their different digits at the same coordinate.

Assume the origin sentence without redundancy is $(x_1, x_2, x_n)$, what we want is to introduce $(y_1, y_2, ..., y_m)$ such that when error occurs less than a number $r$, we can always find the previous message. We actually lift a point in a lower dimension space into a higher dimension and provided it a Hamming distance structure; then, we claim that the every sentence in this space can try to find the nearest target (codeword) to find the corrected sentence.

Figure 1: The visual of lifting a string into a codeword with error-correcting.

The following figure shows this idea.

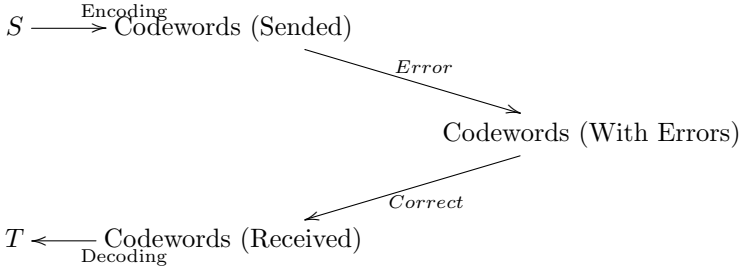



Figure 2: The procedure of message sending with error and correcting.

We can also learn more about the perfect code, the hamming code, and the minimum distance etc., to have more intuition and understanding. But I think they are not so important for today's talk.

# 2 Matrix Representation of Code

In this section, I use the following logic to introduce the content:

$$\text{Codewords} \to \text{Parity-Check Matrix} \to \text{Generate Matrix}$$

We do not care about the encoding and decoding procedures.

**Remark 1.** Identity a codeword We want our codewords to be solid enough. We already know that we can try: $(x_1, x_2, x_3, x_1 + x_2, x_1 + x_3)$ to make a triple information solid with error tolerence 2. We observe the truth that the way we designed in this code can be represented as the constraints:

$$\begin{cases} x_4 = x_1 + x_2 \\ x_5 = x_1 + x_3 \end{cases}$$

which means the code with length 5 and digits 3 is determined identitily by the constrints, also a matrix $2 \times 5$:

$$H := \begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{pmatrix}_{(2,5)}$$

We have the truth that every codewords $\mathbf{x}$ with our design must satisfies $H\mathbf{x} = 0$

**Definition 4** (Parity-Check Matrix)**.** The above matrix is defined as the **Parity-Check Matrix** of our code.

**Property 1.** A vector in $\mathbb{F}^n$ belongs to our codewords iff $Hx = 0$ where $H$ is the Parity-Check Matrix.

**Definition 5** (Generate Matrix)**.** As every codeword in our code is in the nullspace of $H$, we can also find a matrix whose range is our codewords, which is defined as the **Generate Matrix**.

**Property 2.** Assume the Parity-Check Matrix is $H = (A_{(k,n-k)} \mid I_k)$, we have a form for the Generate Matrix reprected to the Parity-Check Matrix:

$$G^T = (I_{n-k} \mid -A^T)$$

*Proof.* Every vector in the range of $G$ has the representation as:

$$x = G^T u, u \in \mathbb{F}^{n-k}$$

As a result, every vector in the range of $G$ must has the property that: $Hx = HG^T u = 0$, which derives a form of $G$ with some observation in blocked matrix.

Sometimes, we first write the $G$ then the $H$; in this case, the $k, n-k$ is reversed. Besides, in binary code, $-A = A$.

# 3 Searching the best as Correcting

As we shared before, we have lift our information to a higher dimension space, with the property that the information will not loss with error tolerence $d$. I want to illustrate that in the correcting procedure, it's trivial to find the nearest codeword as the sent code.

But how can we implement the searching? Assume there is a data structure restored all the codewords, we can certainly check them one by one. As the nullspace of $H$ has dimension $n-k$, we will have $q^{n-k}$ states to search, it is some kinda of expensive.

But we start from here.

# 4 Some Graph Theory

## 4.1 Complexity of graph

In this section, I provide some intuitive idea without more proof for some theorems (maybe) in graph complexity. It has no influence in our topic, i.e. the Trellis Representation. If this section has some faults, the result only be influenced in the optimal property. One can find this property in the origin paper we're reading.

- State Space Profile (Cross-Section).
- Branch Factor Profile.
- State Complexity. We define the maximum of state space profile as **state complexity**.

- Branch Complexity.

A trivial observation is that if we require all the paths are isomorphic to legal codewords, i.e. we build the graph provided the path to be corresponded to a codeword[1]. We can only try to decrease the state space profile.

**Theorem 1.** (Build a graph with path-isomorphic Holding Property)
If we build a tree (graph) and provide the induction property of hoding path-isomorphic, the graph (tree) itself is path-isomorphic.

**Remark 2.** Even though this theorem is trivial in observation like we always try to build a graph by find all the next layer by legal actions. But we need to prove that for every state in layer $i$, there are same number of legal actions to layer $k+1$, which is maybe not so trivial. So I just come up with this theorem and without proof.

I give an idea for the proof that actually, we cannot identity the actions. If we denote them as $1, 2, ..., n$, every action is equivalent. As a result, from layer 0, all the nodes in next layer generated by the transitive map is equivalent.

I guess the following theorem is also true. As I'm not sure some name in Lattice Theory, maybe there is a named Lattice structure for this. I name it as "the fundamentals of Trellis Representation".

**Theorem 2.** (Fundamentals of Trellis Representation) If we provide the above graph representation of a code with next actions with only the legal actions, it's also state complexity minimized. Most importantly, the number of actions at any code in the same layer is same.

*Proof.* It's not very difficult, I will prove it in the next section with the building of Trellis Representation.

**Definition 6** (Precise Complexity)**.** We can use $2|E| - |V| + 1$ to give a precise number for the graph's complexity.

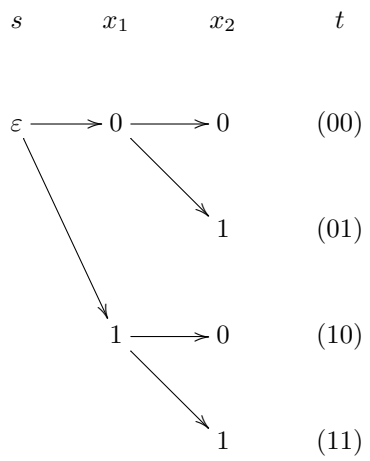
# 5 Graph Representation of Codeword

In this section, we introduce an intuitive idea about how to represent a code, i.e. find a isomorphism of all the codewords, by tree (graph) structure. We use the binary code as example, assume $x$ is length $n$ and every digit is one of $\mathbb{Z}_2 := \{0, 1\}$.

**Theorem 3.** Natural Graph Representation We can represent a codeword by the path in a graph. This representation is bijection.

[1] A comparison is to build a complete tree then delete illegal path (subtree).

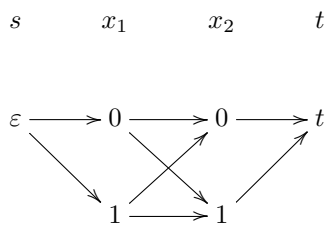
*Proof.* Here is a simple draft for a length 2 code without any constraints.




To build the bijection between the complete tree to our code. The key is to delete some illegal edge from the graph.

**Property 3.** For the above representation, we have every path from the strat to a node is exactly the sentence. And for the last coordinate layer, the edge number is just the total number of codewords. Thus, it's one-to-one. Besides, in this graph, every edge means "append" and every state node is also a sentence.

**Theorem 4.** A code can has another thinner graph representation:

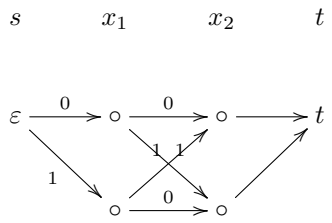



**Property 4.** In the representation, the graph node is not the state anymore. Instead, a path is isomorphic to a state.

**Remark 3.** Why thinner (less state nodes)? When we design algorithm or do real computation, a thiner tree(graph) means smaller storage we used. The transitive relationship is some kinda always needed thus restored[1] in searching. So we trying to find a smaller graph.

[1] If you're familiar with dynamic programming (dp), it's trivial because we always use the paths as a isomorphism of states.

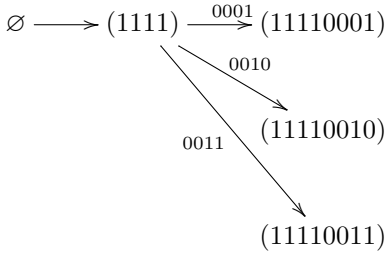
As we have seen that the node in the above graph has no meaning of codewords anymore, the label can be deleted, instead, we use the action as the edge labels for the transtive map.



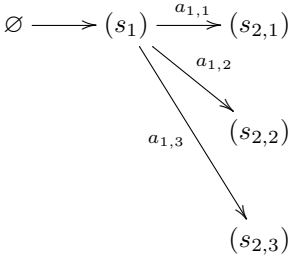

## 5.1 Automata for Searching

If neccessary, I need this subsection to let the audience to be familiar with the dp-like searching and building technical.

**Definition 7** (Prefix-Tree Match)**.** We can use automata wot build a high-effiency searching Finite-State Machine (FSM). For example, if we want to identity $11110001, 11110010, 11110011$, we certainly know there are only three strings and we can use:




**Property 5.** (Rename of the action) If we rename the paths above with an isomorphism, i.e. a bijection, it's same. What's more, if we has an isomorphism about the state and the path, it's also isomorphism to the code itself.




# 6 Trellis Representation

The previous section we care about the easiest case for a representation of our code, the unconstrained case. In this section, we need to try to find the graph isomorphic to our code with constraints. I think one should try to be familiar with DP, and the dimension theorem (isomorphism theorem) of a finite dimension vector space.

## 6.1 Trellis Representation for Code

In this subsection, I will give an introduction to the Trellis Representation for Code. Review some thing from before, I want to show that the trellis representation with its construction is optimal in the complexity.

**Definition 8** (Trellis Representation of Code)**.** Every code has its trellis graph, i.e. a graph with every path from $s$ to $t$ or layer $n+1$ is isomorphic with a codeword. We can build the skeleton of Trellis Representation in this way.

- The graph has $0, 1, ..., n, n+1$ layers;
- The initial layer $L_0$ has only one root node, represent the empty string (codeword);
- In every $L_k, k \in [1..n]$, we have $q$ nodes, where the $q$ is the number of all the posisble legal characters in the field.

**Property 6.** Certainly, in building the graph, there could be less than $q$ . One can think the path parrellel as choose the $0-th$ elements, the first right-down-like path as choose the $1-th$, etc.

- The paths from a node $s_k$ in $L_k$ to $L_{k+1}$ dependents on the node itself, denote the legal characters from $s_k$ to next layer states as $c[s_k]$. We will see that for the nodes in the same layer, the number is same; thus denoted as $c_k$;
- The number of $c_k$ can be computed based on one of isomorphism theorems in Linear Algebra (or Group Theory).

**Definition 9** (Minimal Trellis Graph)**.** The minimal trellis graph is defined as the trellis graph with the smallest Complexity of graph.

The following lemmas we may need to build a Minimal Trellis Graph.

**Lemma 1.** (Identity of codeword) A vector is a codeword iff $Hx = 0$.

**Definition 10** (Past code)**.** For a length $n$ codeword, we define the past subcode as $P_r := s[1..r]000$.

**Definition 11** (Future code)**.** For a lenght $n$ codeword, we define the future subcode as $F_r := 000s[(r+1)..n]$

**Lemma 2.** (Subcode-check of codeword) A passcode can be cut off at any position $r \in \{0, 1, ..., n\}$. As a result, we have the cut-off check for the subcode with the cut-off matrix $H_r, \bar{H}_r$.

**Lemma 3.** (Same number of possible characters at i-th layer) The space of $P_r$ is actually a first-r projection of our code. As a reulst, it's a subspace (subgroup) of our code.

*Proof.* This can be proved by the cut-off matrix product make a subspace. And our matrix and vector rely on the field, which forms an abelian additive group. One can check the group structure is perserved.

**Lemma 4.** (Markov Property of the Trellis Graph) Use a node to represent a codeword (state), and possible characters as action at layer $i$, the graph has the **markov property**.

**Lemma 5.** (A map represent of state space) Denote the $\mathcal{S}_r$ as number of possible states at time $r$, we have $s_r : \mathcal{C} \to \mathcal{S}_r, s_r(x) := H_r x$. Besides, we have $G^T : \mathbb{F}^k \to \mathcal{C}$ and $\mathsf{Range}(G^T) = \mathsf{Ker}(H)$. As a result, we have: $s_r(u) = HG^T u$ of all the possible states.

**Remark 4.** One can consider this construction reversely. Firstly, we have the target vector, i.e. the subcode of codeword, which is determined by the property $H_r x = 0$. It means if we have $s_r : \mathcal{C} \to \mathcal{S}$, which is just an map form a larger space to smaller, the vector we want is determined by $\mathsf{Ker}(S)$.

**Lemma 6.** (Backward checking) As the state space at $r$ is fixed, if we use the backward checking for our code, it must be same, i.e. $s_r(x) = \bar{H}_r \bar{G}_r^T u$.

**Remark 5.** When I write this, it seems that I partially understand why we require the $\mathsf{range}(K_n) = \mathsf{Ker}(K_{n+1})$ in short-five-lemma of **Exact Sequence**. This property provide us the ability to use chain-product to represent the objects in the final space from the start.

Exact Sequence technical seems to be important and the theoretical fundamentals of dp. One can also understand this by the truth that the state space of the i-th layer is fixed for a trellis graph and the posisble choice is same for every node.

**Lemma 7.** (Isomorphism Theorem) Use the isomorphism theorem, we have the following truth:

- The possible states for every node in layer $i$ is exactly $q^{k-(p_i+f_i)}$;
- The possible paths from $i$ to $i+1$ layer is $q^{k-(p_i+f_{i+1})}$.