

3F4: Data Transmission

Handout 3: Trellis Based Algorithms

Jossy Sayir

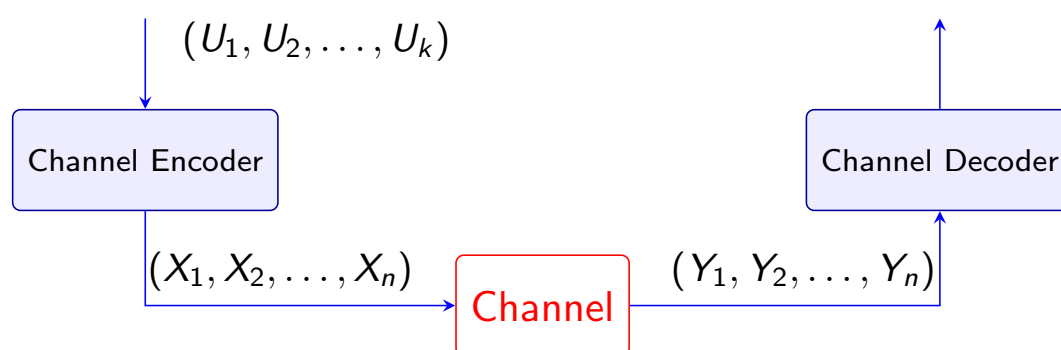
Probability, Systems, Information and Inference Lab Ψ^2
Department of Engineering
js851@cam.ac.uk

Note: slides 14, 16 and 17 are not self-explanatory. They will be annotated during the lecture to explain how the decoder works. If self-studying this part, you can only make sense of those slides by watching the lecture recording.

Lent Term 2026

1 / 17

Reminder: optimal decoding



An unknown codeword $(X_1, \dots, X_n) \in \mathcal{C}$ from a linear block code \mathcal{C} is transmitted and an observation vector $(Y_1, \dots, Y_n) = (y_1, \dots, y_n)$ is received.

2 / 17

Reminder: decoder implementation

	Computation	Decision
Block optimal	$P(\underline{x} \underline{y})$	$\arg \max_{\underline{x}} P(\underline{x} \underline{y})$
Bit optimal	$P(x_\ell \underline{y})$ or $P(u_\ell \underline{y})$	$\arg \max_{x \in \{0,1\}} P(X_\ell = x \underline{y})$

- Bayes' rule:

$$P(\underline{x}|\underline{y}) = \frac{P(\underline{x})P(\underline{y}|\underline{x})}{P(\underline{y})} = \frac{P(\underline{x}) \prod_{\ell=1}^n P(y_\ell|x_\ell)}{P(\underline{y})}$$

- Block decision:

$$\arg \max_{\underline{x} \in \mathcal{C}} P(\underline{x}|\underline{y}) = \arg \max_{\underline{x} \in \mathcal{C}} P(\underline{x}) \prod_{\ell=1}^n P(y_\ell|x_\ell)$$

- Bit computation:

$$P(X_\ell = x|\underline{y}) = \sum_{\underline{x} \in \mathcal{C}, x_\ell = x} P(\underline{x}|\underline{y}) \text{ (or equivalent for } U_\ell)$$

- We need to compute the sum or max over 2^k codewords!

3 / 17

Objective

We need efficient methods to compute sums or max over 2^K terms, where typically $K > 500$.

4 / 17

A motivational example

- Compute $\max_{\underline{u}} f(\underline{u})$ or $\sum_{\underline{u}} f(\underline{u})$ (full max/sum)
- Compute $\max_{\underline{u}: u_\ell = u} f(\underline{u})$ or $\sum_{\underline{u}: u_\ell = u} f(\underline{u})$ (partial max/sum)
- For example, $\underline{u} = u_1 u_2 u_3$ binary can take on values 000,001,010,011,100,101,110,111. Sum or max over 8 terms.
- Assume function factorises $f(u_1 u_2 u_3) = f_1(u_1) f_2(u_2) f_3(u_3)$
- Full sum directly:

$$f(\underline{u}) = f_1(0)f_2(0)f_3(0) + f_1(0)f_2(0)f_3(1) + f_1(0)f_2(1)f_3(0) \\ + f_1(0)f_2(1)f_3(1) + f_1(1)f_2(0)f_3(0) + f_1(1)f_2(0)f_3(1) \\ + f_1(1)f_2(1)f_3(0) + f_1(1)f_2(1)f_3(1)$$

7 sums, 16 multiplications

- Full sum can be computed by factorisation:

$$\sum_{\underline{u}} f(\underline{u}) = (f_1(0) + f_1(1)) (f_2(0) + f_2(1)) (f_3(0) + f_3(1))$$

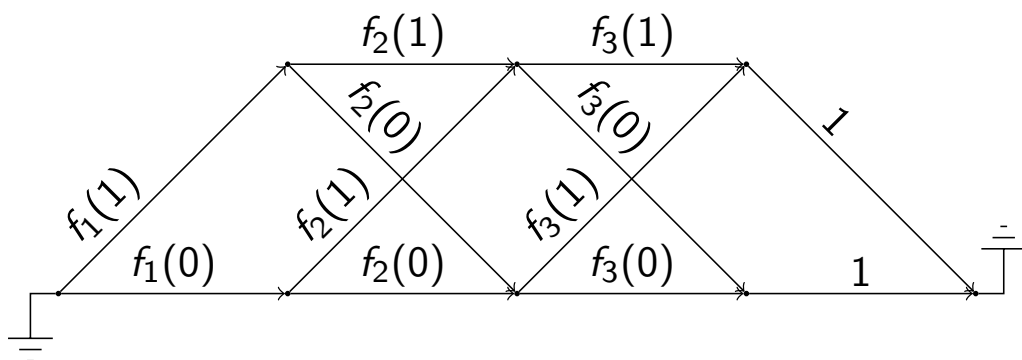
3 sums, 2 multiplications

- Similar approach can obtain the overall max.

5 / 17

Factorisation on a trellis

- The full and all partial sums can be computed by expressing the problem on a trellis



- The branch labels are called the γ values
- At every node j , we compute the α_j as $\sum_i \gamma_{ij} \alpha_i$ where the sum is over all the branches **ending** in node j (**forward iteration**)
- At every node j , we compute the β_j as $\sum_i \gamma_{ij} \beta_i$ where the sum is over all the branches **starting** in node j (**backward iteration**)

6 / 17

Factorisation on a trellis (continued)

- $\alpha_{\text{toor}} = \beta_{\text{root}} = \sum_{\underline{u}} f(\underline{u})$ is the full sum
- For any branch, $\alpha_i \gamma_{ij} \beta_j$ is the sum of the function over all paths that pass through the branch
- To obtain the sum over all \underline{u} whose ℓ -th symbol is 0, sum $\alpha_i \gamma_{ij} \beta_j$ over all branches corresponding to a 0 in the ℓ -th trellis stage
- Any marginalised value of interest can be computed by summing partial results in the trellis similarly
- Replace sums by max in the computation of α and β to solve partial max problems
- A trellis is a **factorisation tool**. It allows us to optimally re-use partial results in all the computations we require.

7 / 17

Formal definitions

- A trellis for the function f must satisfy the following conditions:
 - it is a cycle-free directed graph with a unique root and toor
 - every path \underline{u} through the trellis is valid in the sense that the function f of interest is defined for \underline{u}
 - every valid argument \underline{u} of f corresponds to a path in the trellis
 - for any vertex v in the trellis, the function f of interest factorises as

$$f(\text{path through } v) = f_1(\text{path to } v) f_2(\text{path from } v)$$

for all paths through v

8 / 17

History

- The multiply&add algorithm is called the “Forward Backward” algorithm in the inference, machine learning, and speech processing literature. It is typically applied to general Hidden Markov Models (HMMs) and was originally proposed by Baum & Petrie in 1966
- In communications, it is often called the Bahl Cocke Jelinek Raviv (BCJR) algorithm after a 1974 paper that proposed it for decoding convolutional codes
- The Baum Welch algorithm puts the forward backward on its head, using it to estimate unknown model parameters
- The multiply&max algorithm is almost universally called the Viterbi algorithm after his 1967 paper applying it to convolutional codes. It is an instance of dynamic programming, developed by Bellman in the 1950s
- For DNA alignment, it is known as Needleman Wunsch or Smith Waterman (slightly different flavours of the same algorithm)

9 / 17

Numerical Stability

- Viterbi algorithm often turned from multiply&max algorithm to sum&max or sum&min by taking log or $-\log$
- For BCJR, sums cannot be evaluated in the log domain. Still, products of probabilities are challenging without taking logs.

Many implementations use the so called “log max” trick

$$\ln(e^x + e^y) = \ln \left[e^{\max(x,y)} \left(1 + e^{\min(x,y) - \max(x,y)} \right) \right]$$

$$= \max(x, y) + \ln(1 + e^{-\Delta})$$

$$\text{where } \Delta = \max(x, y) - \min(x, y) \geq 0$$

and $\ln(1 + e^{-\Delta})$ is often tabulated for fast evaluation. Python has a function `numpy.logaddexp()` to evaluate this¹.

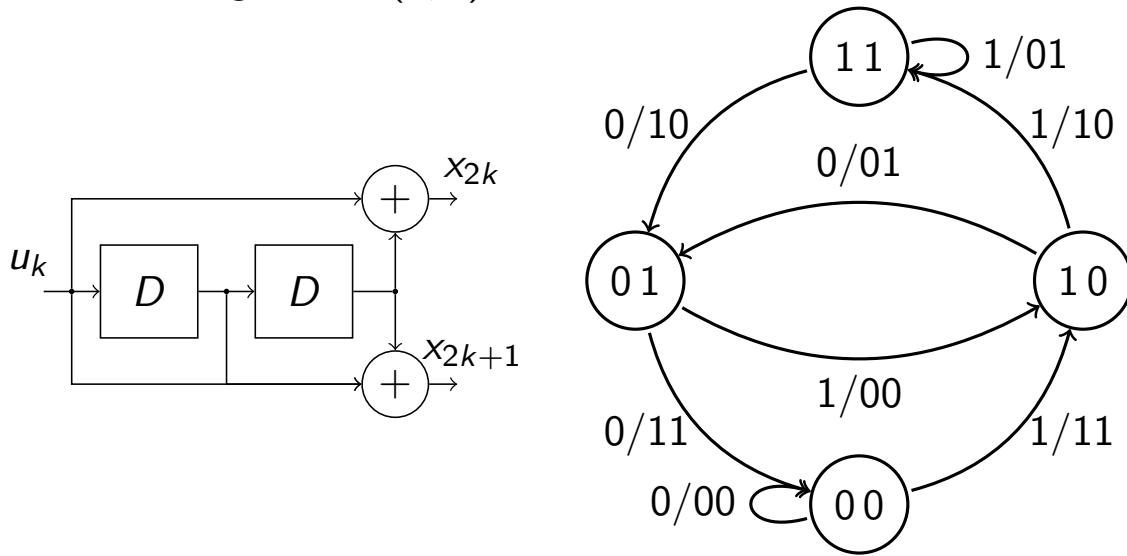
- It is not well approximated by a single polynomial but piecewise linear or quadratic approximation e.g., around pre-computed $\ln \kappa$ for $\kappa = 1, 2, 3 \dots$ follows from

$$\ln(1 + e^{-\Delta}) = \ln \frac{\kappa + 1}{\kappa} - \frac{\Delta - \ln \kappa}{\kappa + 1} + \frac{\kappa(\Delta - \ln \kappa)^2}{2(\kappa + 1)^2} + o((\Delta - \ln \kappa)^3)$$

¹Thanks to Ben Domb 2024 for pointing this out

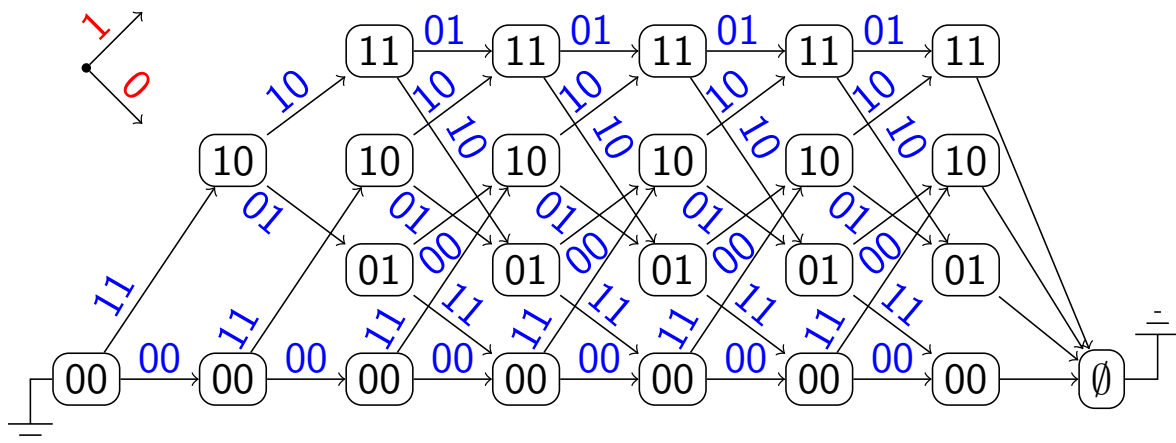
Trellis of a convolutional code

We consider again the $(5, 7)_8$ forward encoder



11 / 17

Trellis of a convolutional encoder (continued)



- This time we consider the 6 input *non-terminated* encoding.
- In order to satisfy the trellis conditions, we add a termination state \emptyset . The edges into \emptyset aren't labeled as they aren't associated with inputs or outputs
- It's easy to verify that all trellis conditions are fulfilled

12 / 17

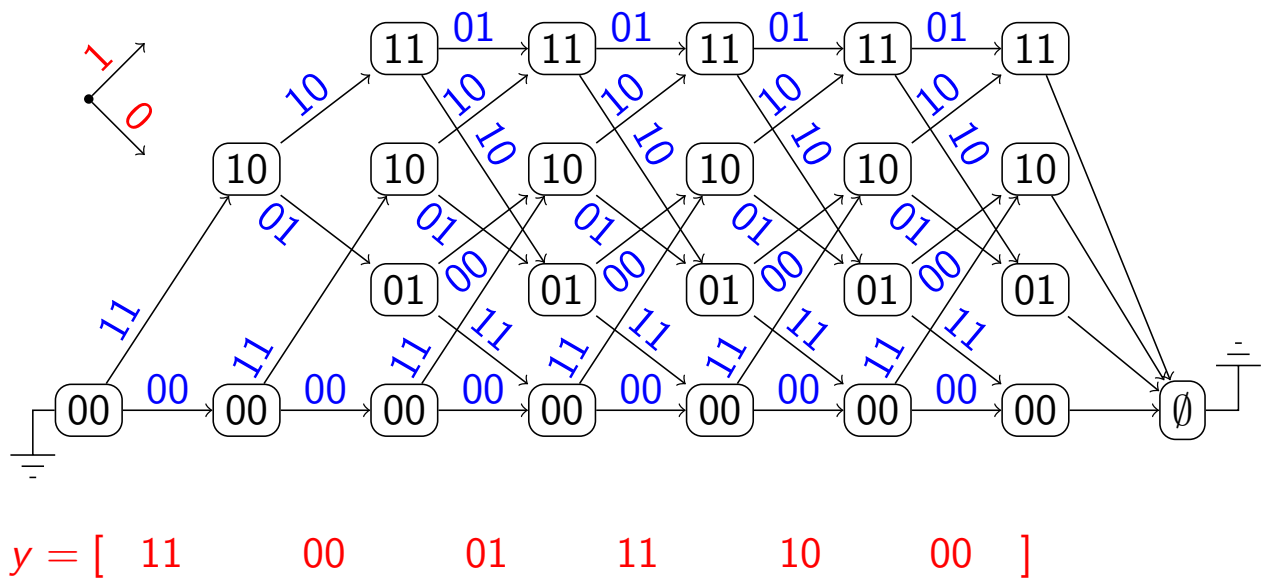
Transmission over a binary symmetric channel (BSC) with crossover probability $p < 0.5$

$$\begin{aligned}
 \arg \max_{\underline{x}} P(\underline{y}|\underline{x}) &= \arg \min_{\underline{x}} \left[-\log \prod_{k=1}^N P(y_k|x_k) \right] \\
 &= \arg \min_{\underline{x}} \left[-\log \left(p^{d(\underline{x},\underline{y})} (1-p)^{N-d(\underline{x},\underline{y})} \right) \right] \\
 &= \arg \min_{\underline{x}} \left[-\log \left((1-p)^N \left(\frac{p}{1-p} \right)^{d(\underline{x},\underline{y})} \right) \right] \\
 &= \arg \min_{\underline{x}} \left[-N \log(1-p) + d(\underline{x},\underline{y}) \log \frac{1-p}{p} \right] \\
 &= \arg \min_{\underline{x}} d(\underline{x},\underline{y})
 \end{aligned}$$

so the maximum likelihood decoder for this channel is a minimum Hamming distance decoder.

13 / 17

Viterbi Algorithm for the BSC



14 / 17

Transmission over a Binary Input Additive White Gaussian Noise channel (BiAWGN)

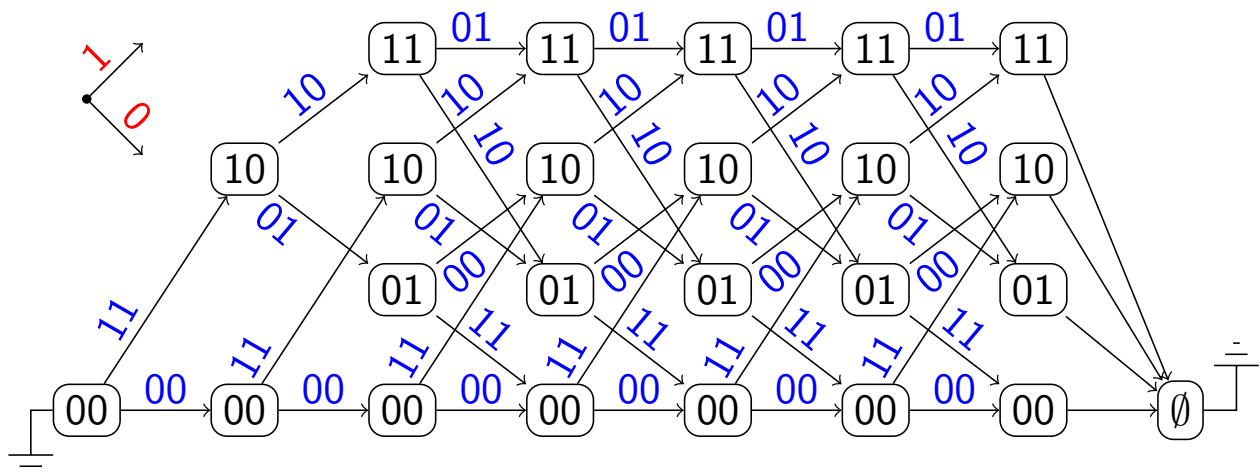
The x_k are now interpreted as +1 (for 0) or -1 (for 1). The outputs are $Y = X + Z$ where $Z \sim \mathcal{N}(0, \sigma^2)$.

$$\begin{aligned} \arg \max_{\underline{x}} f(\underline{y}|\underline{x}) &= \arg \min_{\underline{x}} \left[-\log \prod_{k=1}^N f(y_k|x_k) \right] \\ &= \arg \min_{\underline{x}} \left[-\log \prod_{k=1}^N \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y_k-x_k)^2}{2\sigma^2}} \right] \\ &= \arg \min_{\underline{x}} \sum_{k=1}^N (y_k - x_k)^2 \end{aligned}$$

so the maximum likelihood decoder for this channel is a minimum Euclidian distance decoder.

15 / 17

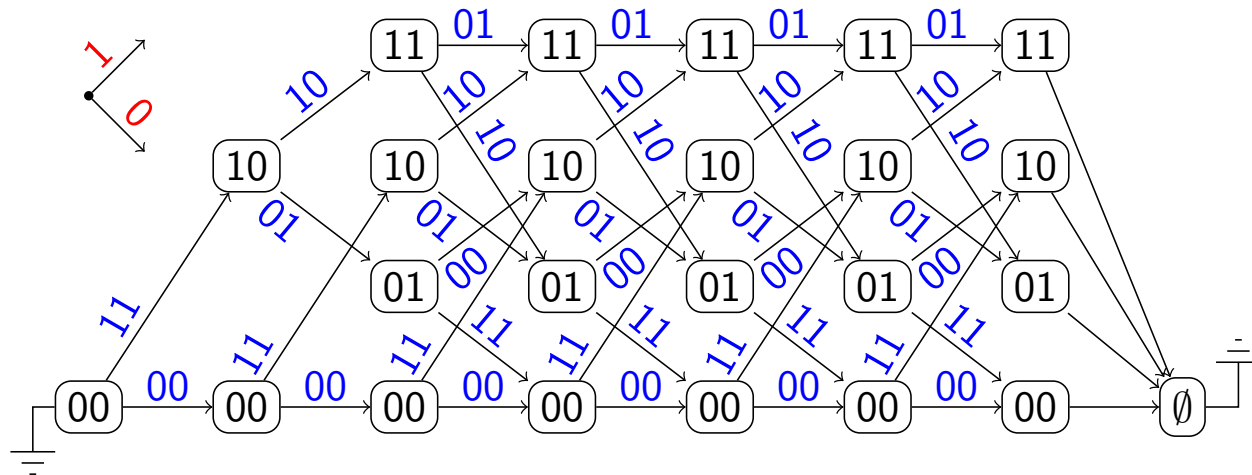
Viterbi Algorithm for the BiAWGN



$$\underline{y} = [-1.2, 0.7, 0.1, -0.9, 1.1, -0.8, 0.4, -0.2, 0.1, -0.8, 0.8, 0.5]$$

16 / 17

BCJR for the BSC



$$\underline{y} = [11 \quad 00 \quad 01 \quad 11 \quad 10 \quad 00]$$