Combining the Burrows Wheeler Transform and the Context-Tree Weighting Algorithm

Frans M.J. Willems

Department of Electrical Engineering
Eindhoven University of Technology

1st IEEE Seminar on Future Directions in
& 2nd African Winter School on
Information Theory and Communications,
August 16-21, 2015, Protea Kruger Gate, South Africa

Motivation

European School on Information Theory, April 20-24, 2015, Zandvoort, The Netherlands:

Tutorial lectures were given by Young-Han Kim, Stephanie Wehner, Imre Csiszar, and Stephan ten Brink and

Motivation

Richard Durbin (Genome Informatics Group, Sanger Institute, Cambridge, UK)

Human Genome Project, now 99598 citations, h-index = 102, now co-leader of 1000 Genomes Project.

"Storage and Search of Genome Sequence Information".

Based on the **Burrows-Wheeler transform**: "A Block-sorting Lossless Data Compression Algorithm", Digital Systems Research Center Report 124, 1994, by M. Burrows, & D.J. Wheeler.

Universal Source Coding Algorithms

CLASSES of UNIVERSAL SOURCE CODING ALGORITHMS:

- **String Matching methods**
  Ziv-Lempel (1977, Sliding-Window, **IT-Soc Best Paper Award**).
  Ziv-Lempel (1978, Dictionary).
  Elias (1987, Recency rank, Move-to-Front analysis). W. (1989, Kac-connection). Wyner-Ziv (1994, LZ77 - proof).
- **Source-Modeling and Arithmetic Coding methods**
  Arithm. Cod.: Pasco (1976), Rissanen (1976), Rissanen-Langdon (1979), Rissanen-Langdon (1981)
  Source Modeling: Langdon-Rissanen (1983), Rissanen (1983, Context-algorithm), Weinberger-Rissanen-Feder (1995), W.-Shtarkov-Tjalkens (1995, **IT-Soc Best Paper Award**): Context-Tree Weighting (CTW) Method
- **Burrows-Wheeler (BW) transform, followed by Move-to-Front and Huffman Coding Methods**
  BW: 1994
  MTF: Bentley-Sleator-Tarjan-Wei (1986), Elias (1987)

Problem Description

Computer Science focusses on **minimizing Memory and Computational Complexity.**
Information Theory focusses on **minimizing Redundancy.**

BW is very efficient, allows for searching in the compressed domain, but is **SUBOPTIMAL in terms of redundancy** for tree sources.
BW does NOT achieve the Rissanen lower bound ($1/2 \log_2 N$ bits per parameter, 1984, **IT-Soc Best Paper Award**).
Effros-Visweswariah-Kulkarni-Verdu (2002): in the binary case $\log_2 N$ bits per parameter **extra redundancy** per parameter.

Fixed-depth CTW is linear in N (memory and computations) but is **REDUNDANCY OPTIMAL** in Rissanen sense ($1/2 \log_2 N$ bits per parameter) for tree sources. CTW does not allow for searching in the compressed domain however.

PROBLEM:

Improve the redundancy of BW data-compression.

Problem Description

Computer Science focusses on **minimizing Memory and Computational Complexity.**
Information Theory focusses on **minimizing Redundancy.**

BW is very efficient, allows for searching in the compressed domain, but is **SUBOPTIMAL in terms of redundancy** for tree sources.
BW does NOT achieve the Rissanen lower bound ($1/2 \log_2 N$ bits per parameter, 1984, **IT-Soc Best Paper Award**).
Effros-Visweswariah-Kulkarni-Verdu (2002): in the binary case $\log_2 N$ bits per parameter **extra redundancy** per parameter.

Fixed-depth CTW is linear in N (memory and computations) but is **REDUNDANCY OPTIMAL** in Rissanen sense ($1/2 \log_2 N$ bits per parameter) for tree sources. CTW does not allow for searching in the compressed domain however.

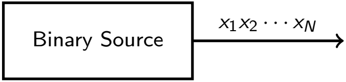
PROBLEM:

Improve the redundancy of BW data-compression.

Outline Lecture

- INTRODUCTION
- IID SOURCES, PREFIX CODES, REDUNDANCY
- ENUMERATIVE CODING
- ARITHMETIC CODING
- CONTEXT-TREE WEIGHTING
- BURROWS WHEELER
- CODING BW-SEQUENCES
- CODING B-COUNTS
- FINDING BEST TREE MODEL
- BINARY DECOMPOSITION
- CONCLUSION
- FUTURE DIRECTIONS

Binary Sources, Sequences, IID



The **binary source** produces a **sequence** $x_1^N = x_1x_2\cdots x_N$ with components $\in \{0,1\}$ with probability $P(x_1^N)$.

Definition (Binary IID Source)

For an **independent identically distributed (i.i.d.)** source with parameter θ, for $0 \le \theta \le 1$,

$$P(x_1^N) = \prod_{n=1}^{N} P(x_n),$$

where

$$P(1) = \theta, \text{ and } P(0) = 1-\theta.$$

A sequence x_1^N containing w ones (i.e., having weight w) and $N-w$ zeros has probability

$$P(x_1^N) = (1-\theta)^{N-w}\theta^w.$$

Definition (Entropy Binary IID Source)

The **ENTROPY** of this source is $h(\theta) \stackrel{\Delta}{=} (1-\theta)\log_2 \frac{1}{1-\theta} + \theta \log_2 \frac{1}{\theta}$ (bits).

Prefix Codes

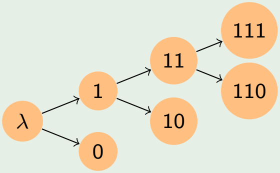
Definition (Prefix Code)

In a **prefix code** no codeword is the prefix of any other codeword.

We restrict ourselves to prefix codes. Codewords in a prefix code can be regarded as **leaves in a rooted tree**. Prefix codes lead to **instantaneous decodability**.

Example

x_1^N	$c(x_1^N)$	$L(x_1^N)$
00	0	1
01	10	2
10	110	3
11	111	3



Prefix-Codes (cont.)

Theorem (Kraft, 1949)

(a) The lengths of the codewords in a prefix code satisfy Kraft's inequality

$$\sum_{x_1^N \in \mathcal{X}^N} 2^{-L(x_1^N)} \leq 1.$$

(b) For codeword lengths satisfying Kraft's inequality there exists a prefix code with these lengths.

Prefix-Codes (cont.)

This leads to:

Theorem (Fano, 1961)

(a) Any prefix code satisfies

$$E[L(X_1^N)] \geq H(X_1^N) = Nh(\theta).$$

The minimum is achieved if and only if $L(x_1^N) = \log_2 \frac{1}{P(x_1^N)}$ *(***ideal codeword length***) for all* $x_1^N \in \mathcal{X}^N$ *with nonzero* $P(x_1^N)$.

(b) There exist prefix codes with $L(x_1^N) = \left\lceil \log_2 \frac{1}{P(x_1^N)} \right\rceil$, *hence*

$$L(x_1^N) < \log_2 \frac{1}{P(x_1^N)} + 1$$

*(***ideal codeword length*** plus 1 bit). These codes achieve*

$$E[L(X_1^N)] < H(X_1^N) + 1 = Nh(\theta) + 1 \text{ bit}.$$

Redundancy

Definition

The **individual redundancy** $\rho(x_1^N)$ of a sequence x_1^N is defined as

$$\rho(x_1^N) = L(x_1^N) - \log_2 \frac{1}{P(x_1^N)},$$

i.e., **codeword-length minus ideal codeword-length**.

Definition

The **expected redundancy** $\rho(x_1^N)$ is defined as

$$\rho = E[L(X_1^N)] - E[\log_2 \frac{1}{P(x_1^N)}] = E[L(X_1^N)] - H(X_1^N),$$

i.e., **expected codeword-length minus entropy**.

Note that there exist codes with individual redundancies ≤ 1 and consequently with expected redundancies ≤ 1.

Indexing Sequences of Fixed Weight

IDEA:

Sequences having the same number of ones (weight) have the same probability and only need to be INDEXED.

Definition (Lexicographical Ordering)

In a **lexicographical ordering** $(0 < 1)$ we say that $x_1^N < y_1^N$ if $x_n < y_n$ for the smallest index n such that $x_n \neq y_n$.
Consider a subset $\mathcal{S}$ of the set $\{0,1\}^N$. Let $i_{\mathcal{S}}(x_1^N)$ be the **lexicographical index** of $x_1^N \in \mathcal{S}$, i.e., the number of sequences $y_1^N < x_1^N$ for $y_1^N \in \mathcal{S}$.

Example

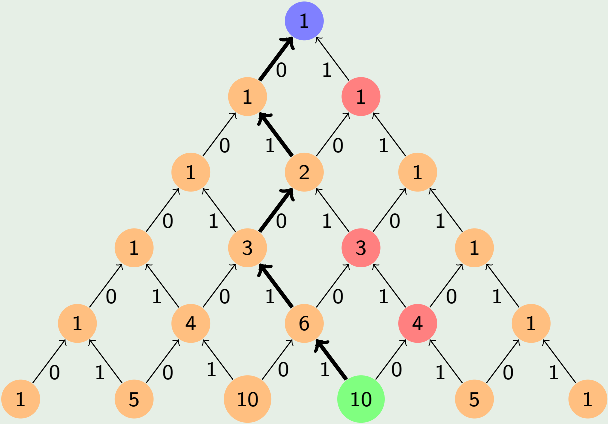
Let $N = 5$ and $\mathcal{S} = \{x_1^N : w(x_1^N) = 3\}$ where $w(x_1^N)$ is the weight of x_1^N.
Then $|\mathcal{S}| = \binom{5}{3} = 10$ and:

$$
\begin{array}{ll}
i_{\mathcal{S}}(11100) = 9 & i_{\mathcal{S}}(10011) = 4 \\
i_{\mathcal{S}}(11010) = 8 & i_{\mathcal{S}}(01110) = 3 \\
i_{\mathcal{S}}(11001) = 7 & i_{\mathcal{S}}(01101) = 2 \\
i_{\mathcal{S}}(10110) = 6 & i_{\mathcal{S}}(01011) = 1 \\
i_{\mathcal{S}}(10101) = 5 & i_{\mathcal{S}}(00111) = 0
\end{array}
$$

Pascal-Δ Method (Lynch (1966), Davisson (1966), Schalkwijk (1972))

Example (From Sequence to Index)

Let $N = 5$ and $\mathcal{S} = \{x_1^N : \sum x_n = 3\}$. Then $|\mathcal{S}| = \binom{5}{3} = 10$.

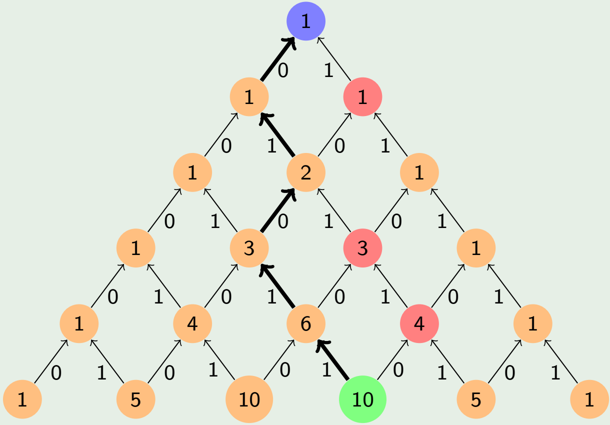


$i(11010) = 4 + 3 + 1 = 8.$

Pascal-Δ Method

Example (From Index to Sequence)

Again $N = 5$ and $\mathcal{S} = \{x_1^N : \sum x_n = 3\}$.



Index $i = 8$, now (a) $8 \geq 4$ hence $x_1 = 1$, (b) $8 \geq 4 + 3$ hence $x_2 = 1$, (c) $8 < 4 + 3 + 2$ hence $x_3 = 0$, (d) $8 \geq 4 + 3 + 1$ hence $x_4 = 1$, (e) $x_5 = 0$.

How to Code the Index?

If sequence x_1^N has weight w, the index $i(x_1^N)$ is encoded using a fixed-length code of length

$$L_I(x_1^N) = \left\lceil \log_2 \binom{N}{w} \right\rceil .$$

Example

Index $i_{\mathcal{S}}(11010) = 8$. Since $|\mathcal{S}| = 10$ the length of the fixed-length code for $i(x_1^N)$ is 4, and the corresponding codeword could be 1000.

How to Code the Weight? θ Coding

The weights that can occur are $w \in \{0, 1, \cdots, N\}$. Knowing source parameter θ we can easily compute the probability that weight w occurs

$$P_\theta(w) = \binom{N}{w}(1-\theta)^{N-w}\theta^w.$$

The weight w of sequence x_1^N can therefore be encoded with a **variable-length code** of length

$$L_\theta(w) = \left\lceil \log_2 \frac{1}{\binom{N}{w}(1-\theta)^{N-w}\theta^w.} \right\rceil.$$

Now we can write for the **total codeword length** for sequence x_1^N having weight w

$$\begin{aligned} L(x_1^N) &= L_\theta(w) + L_I(x_1^N) \\ &= \left\lceil \log_2 \frac{1}{\binom{N}{w}(1-\theta)^{N-w}\theta^w} \right\rceil + \left\lceil \log_2 \binom{N}{w} \right\rceil \\ &\leq \log_2 \frac{1}{(1-\theta)^{N-w}\theta^w} + 2. \end{aligned}$$

θ Coding: Individual and Expected Redundancy

Assume that our sequence x_1^N has weight $w = w(x_1^N)$, then for all $0 \le \theta \le 1$, we get for the **individual redundancy**

$$
\begin{aligned}
\rho(x_1^N) &= L(x_1^N) - \log_2 \frac{1}{(1-\theta)^{N-w}\theta^w} \\
&\le \log_2 \frac{1}{(1-\theta)^{N-w}\theta^w} + 2 - \log_2 \frac{1}{(1-\theta)^{N-w}\theta^w} \\
&= 2 \text{ bits.}
\end{aligned}
$$

For the **expected redundancy**, when $0 \le \theta \le 1$, we get

$$\rho = E[L(X_1^N)] - Nh(\theta) \le 2 \text{ bits.}$$

How to Code the Weight? Uniform Coding

The weights that can occur are $w \in \{0, 1, \cdots, N+1\}$, hence there are $N+1$ alternatives. Therefore we can use a **uniform code** of length

$$L_U(w) = \lceil \log_2(N+1) \rceil .$$

Now we can write for the **total codeword length** for sequence x_1^N having weight w

$$\begin{aligned} L(x_1^N) &= L_U(w) + L_I(x_1^N) \\ &= \lceil \log_2(N+1) \rceil + \left\lceil \log_2 \binom{N}{w} \right\rceil \\ &\leq \log_2(N+1)\binom{N}{w} + 2. \end{aligned}$$

Uniform Coding: Individual Redundancy

Assume that our sequence x_1^N has weight w, then for all $0 \leq \theta \leq 1$, then we get for the individual redundancy

$$\begin{aligned}\rho(x_1^N) &= L(x_1^N) - \log_2 \frac{1}{(1-\theta)^{N-w}\theta^w} \\ &\leq \log_2(N+1)\binom{N}{w} + 2 - \log_2 \frac{1}{(1-\theta)^{N-w}\theta^w}.\end{aligned}$$

If both $N - w \rightarrow \infty$ and $w \rightarrow \infty$ the **Stirling approximation** yields

$$(N+1)\binom{N}{w} \approx (N+1)\frac{\sqrt{2\pi N}}{\sqrt{2\pi(N-w)}\sqrt{2\pi w}}\left(\frac{N}{N-w}\right)^{N-w}\left(\frac{N}{w}\right)^w.$$

Moreover

$$(1-\theta)^{N-w}\theta^w \leq \left(\frac{N-w}{N}\right)^{N-w}\left(\frac{w}{N}\right)^w.$$

Combining this yields for the **individual redundancy** for $N - w \rightarrow \infty$ and $w \rightarrow \infty$ that

$$\begin{aligned}\rho(x_1^N) &= L(x_1^N) - \log_2 \frac{1}{(1-\theta)^{N-w}\theta^w} \\ &\overset{(\approx)}{\leq} \log_2 \sqrt{\frac{N}{2\pi}} - \log_2 \sqrt{\left(\frac{N-w}{N}\right)\left(\frac{w}{N}\right)} + 2 \text{ bits.}\end{aligned}$$

Uniform Coding: Expected Redundancy

For the **expected redundancy**, when $0 < \theta < 1$, for large N we obtain

$$\rho = E[L(X_1^N)] - Nh(\theta) \overset{(\approx)}{\leq} \log_2 \sqrt{\frac{N}{2\pi}} - \log_2 \sqrt{(1-\theta)\theta} + 2 \text{ bits.}$$

UNIVERSAL!

NOT VERY GOOD AT THE BORDERS ...

How to Code the Weight? Krichevsky-Trofimov

Definition (KT Probability (1981))

For all integer $a \geq 0$ and $b \geq 0$, define the KT-probability as

$$P_{kt}(a,b) \triangleq \frac{(2a)!}{2^a a!} \frac{(2b)!}{2^b b!} \frac{1}{2^{a+b}(a+b)!}.$$

Since $\sum_{w=0,N} \binom{N}{w} P_{kt}(N-w,w) = 1$, the weight w of sequence x_1^N can be encoded with a **variable-length KT-code** of length

$$L_{KT}(w) = \left\lceil \log_2 \frac{1}{\binom{N}{w} P_{kt}(N-w,w)} \right\rceil.$$

Now we can write for the **total codeword length** for sequence x_1^N having weight w

$$\begin{aligned} L(x_1^N) &= L_{KT}(w) + L_I(x_1^N) \\ &= \left\lceil \log_2 \frac{1}{\binom{N}{w} P_{kt}(N-w,w)} \right\rceil + \left\lceil \log_2 \binom{N}{w} \right\rceil \\ &\leq \log_2 \frac{1}{P_{kt}(N-w,w)} + 2. \end{aligned}$$

KT Coding: Individual Redundancy

Assume that our sequence x_1^N has weight $w = w(x_1^N)$, then for all $0 \leq \theta \leq 1$, we get

$$\begin{aligned}\rho(x_1^N) &= L(x_1^N) - \log_2 \frac{1}{(1-\theta)^{N-w}\theta^w} \\ &\leq \log_2 \frac{1}{P_{kt}(N-w,w)} + 2 - \log_2 \frac{1}{(1-\theta)^{N-w}\theta^w}\end{aligned}$$

If both $N - w \to \infty$ and $w \to \infty$ the **Stirling approximation** yields

$$\frac{1}{P_{kt}(N-w,w)} \approx \sqrt{\frac{\pi N}{2}} (\frac{N}{N-w})^{N-w} (\frac{N}{w})^w.$$

Again

$$(1-\theta)^{N-w}\theta^w \leq (\frac{N-w}{N})^{N-w}(\frac{w}{N})^w.$$

Combining this yields for the **individual redundancy** for $N - w \to \infty$ and $w \to \infty$ that

$$\rho(x_1^N) = L(x_1^N) - \log_2 \frac{1}{(1-\theta)^{N-w}\theta^w} \overset{(\approx)}{\leq} \frac{1}{2}\log_2 N + \frac{1}{2}\log_2 \frac{\pi}{2} + 2 \text{ bits.}$$

KT Coding: Expected Redundancy

For the **expected redundancy**, when $0 < \theta < 1$, for large N we get

$$\rho = E[L(X_1^N)] - Nh(\theta) \overset{(\approx)}{\leq} \frac{1}{2}\log_2 N + \frac{1}{2}\log_2 \frac{\pi}{2} + 2 \text{ bits.}$$

NOTE

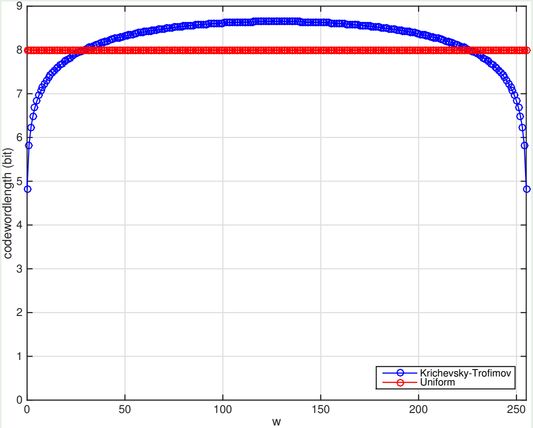
It can be shown for KT coding, for all $0 \leq \theta \leq 1$, that

$$\begin{aligned} \rho(x_1^N) &\leq \frac{1}{2}\log_2 N + 3 \text{ bits for all } x_1^N, \text{ and} \\ \rho &\leq \frac{1}{2}\log_2 N + 3 \text{ bits.} \end{aligned}$$

Weight Codewordlength

Example ($N = 255$, codewordlength as function of weight w of x_1^N)

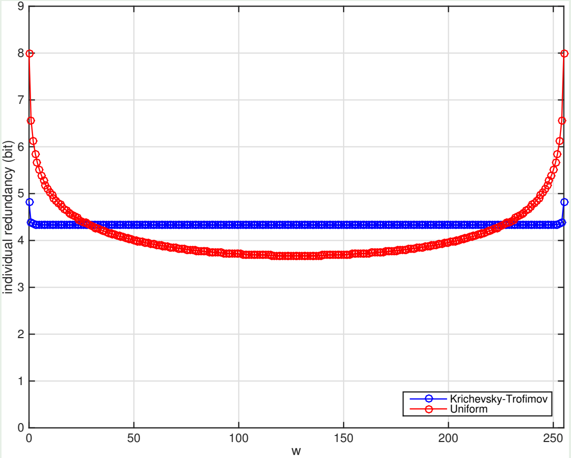
No $\lceil \cdots \rceil$-s.



Codewordlengths are roughly $\log_2 N$. Note that KT-codewordlength is smaller ($\approx \frac{1}{2} \log_2 N$) for $w = 0$ and $w = N$.

BOUND Individual Redundancy

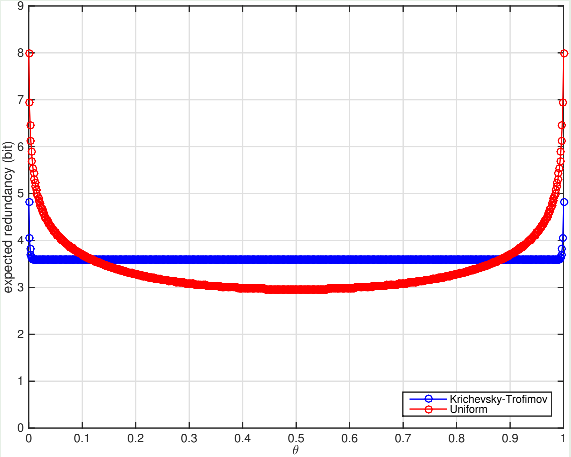
Example ($N = 255$, individual redundancy as a function of the weight w)



Individual redundancies are roughly $\frac{1}{2}\log_2 N$. Note that uniform redundancy is larger ($= \log_2(N+1)$) for $\theta = 0$ and $\theta = 1$.

Expected Redundancy

Example ($N = 255$, expected redundancy as function of θ.)



Expected redundancies are roughly $\frac{1}{2}\log_2 N$. Again the uniform redundancy is larger ($= \log_2(N+1)$) for $w = 0$ and $w = N$.

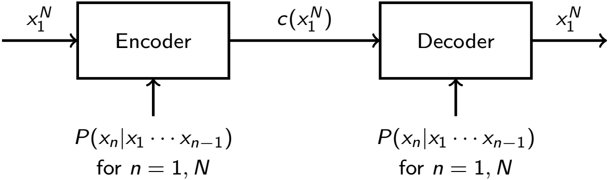
Conclusion Enumerative Source Coding

- NON-UNIVERSAL: θ-Coding yields individual redundancy not exceeding 2 bits.
- UNIVERSAL:
  - Both Uniform Coding and KT Coding achieve individual redundancy roughly $\frac{1}{2}\log_2 N$ if both $N - w \to \infty$ and $w \to \infty$.
  - KT Coding has similar behaviour at borders ($w = 0$ and $w = N$).
  - Uniform Coding is simpler.

Arihmetic Coding: Idea Elias

Elias:

Codewords can be COMPUTED SEQUENTIALLY from the source sequence using conditional PROBABILITIES of next symbol given the previous ones, and vice versa.



Arithmetic Coding: Universal Codes

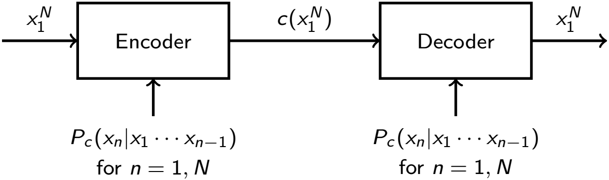
Coding Probabilities

If the **actual probabilities** $P(x_1^N)$ are **not known** arithmetic coding is still possible if instead of $P(x_1^N)$ we use **coding probabilities** $P_c(x_1^N)$ satisfying

$$\begin{aligned} P_c(x_1^N) &> 0 \text{ for all } x_1^N, \text{ and} \\ \sum_{x_1^N} P_c(x_1^N) &= 1. \end{aligned}$$

Then

$$L(x_1^N) < \log_2 \frac{1}{P_c(x_1^N)} + 2.$$



PROBLEM: How do we choose the coding probabilities $P_c(x_1^N)$?

Arithmetic Coding for a Binary IID Source, Unknown θ

Arithmetic Coding based on KT Probability

A good coding probability $P_c(x_1^N)$ for a sequence x_1^N that contains a zeroes and $b = N - a$ ones is

$$P_{kt}(a, b) \triangleq \frac{(2a)!}{2^a a!} \frac{(2b)!}{2^b b!} \frac{1}{2^{a+b}(a+b)!}.$$

Probability of a sequence with a zeroes and b ones followed by a one

$$P_{kt}(a, b+1) = \frac{b + 1/2}{a + b + 1} \cdot P_{kt}(a, b),$$

hence SEQUENTIAL COMPUTATION is possible!
Using arithmetic coding, the total individual redundancy

$$\rho(x_1^N) < \log_2 \frac{\theta^a (1-\theta)^b}{P_{kt}(a, b)} + 2 \leq \frac{1}{2} \log_2 N + 3 \text{ bits.}$$

for all θ and x_1^N with a zeroes and b ones.

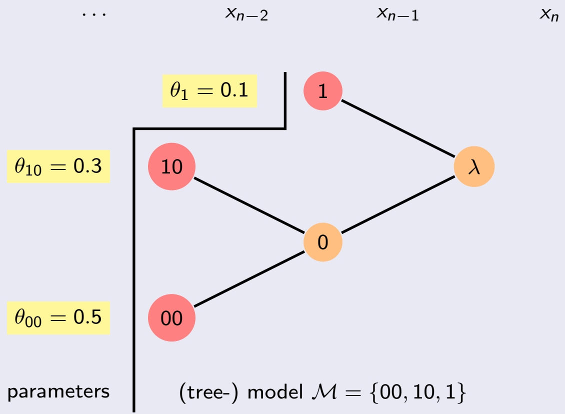
Context Tree Weighting: Universal Codes

OBJECTIVE:

Design good code probabities for sources with UNKNOWN PARAMETERS and STRUCTURE.

Context Tree Weighting: Binary Tree-Sources

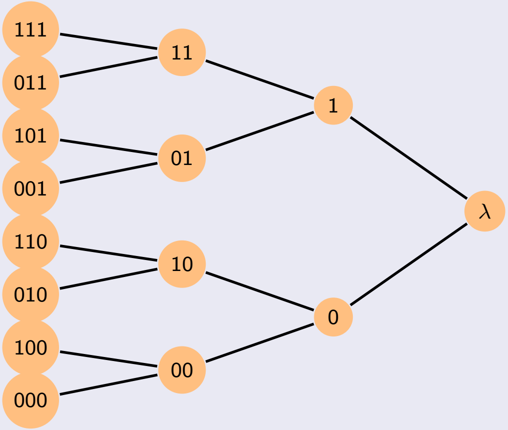
Definition



$$
\begin{aligned}
P(X_n = 1 | \cdots, X_{n-1} = 1) &= 0.1 \\
P(X_n = 1 | \cdots, X_{n-2} = 1, X_{n-1} = 0) &= 0.3 \\
P(X_n = 1 | \cdots, X_{n-2} = 0, X_{n-1} = 0) &= 0.5
\end{aligned}
$$

Context Trees

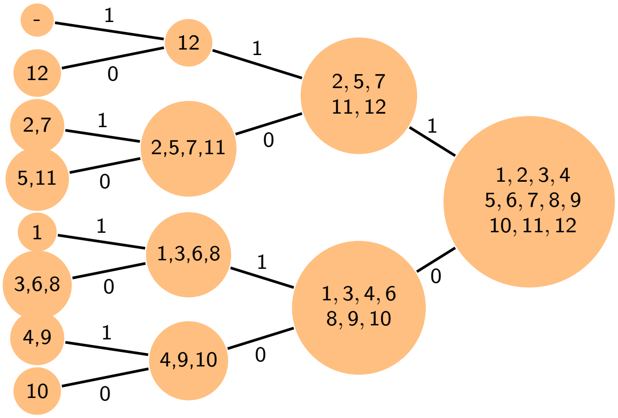
Definition (Context Tree)



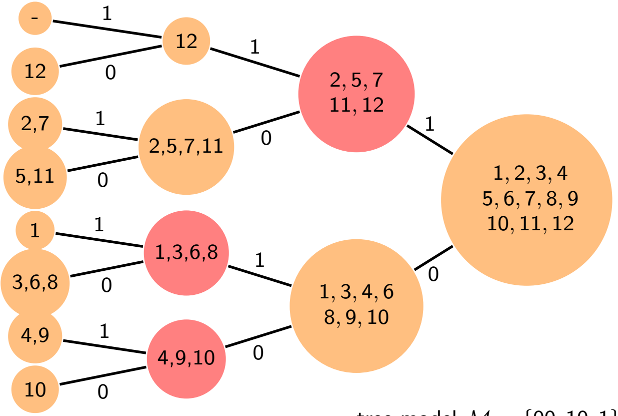
Node s contains the sequence of source symbols that have occurred following context s. Depth is D.

Context-Tree Splits Up Sequences in Subsequences

			1	2	3	4	5	6	7	8	9	10	11	12	$n \rightarrow$
1	1	0	1	0	0	1	0	1	0	0	0	1	1	0	$\cdots$
	past						x_1^N								



Context-Tree Nodes Containing IID Sequences



tree-model $\mathcal{M} = \{00, 10, 1\}$

Context-Tree Weighting Coding Probabilities

Recursive Definition: From Leaves via Internal Nodes to Root

Let a_s and b_s be the number of zeros resp. ones in the subsequence corresponding to leaf, node, or root s.

The subsequence corresponding to a leaf s of the context tree is IID. A good coding probability for this subsequence is therefore

$$P_w(s) \triangleq P_{kt}(a_s, b_s).$$

Weighting the coding probabilities corresponding to both alternatives (node is iid or needs further splitting) yields the coding probability

$$P_w(s) \triangleq \frac{P_{kt}(a_s, b_s) + P_w(0s) \cdot P_w(1s)}{2}$$

for the subsequence that corresponds to node or root s.
Recursively we find in the *root* λ of the context-tree the coding probability $P_w(\lambda)$ for the entire source sequence x_1^N.

Individual Redundancy CTW

Theorem (W., Shtarkov, and Tjalkens (1995))

In general for a tree source with $|\mathcal{M}|$ leaves (parameters):

$$\rho(x_1^N) < (2|\mathcal{M}|-1) + \sum_{l \in \mathcal{M}} \frac{1}{2} \log_2(a_l + b_l) + 2 \text{ bits.}$$

(model, parameter, and coding redundancies)

About Model Redundancies

A binary tree model can be described recursively:
Code(tree)=0 if tree is only root,
else Code(tree)=(1,Code(subtree-0),Code(subtree-1)).
This leads to $2|\mathcal{M}|-1$ bits.

Remarks Context-Tree Weighting

- CTW implements a "weighting" (Bayes mixture) over all tree-models with depth not exceeding D, i.e.,

$$P_w(\lambda) = \sum_{\mathcal{M} \leq D} P(\mathcal{M}) P_{kt}(x_1^N | \mathcal{M}),$$

  with $P_{kt}(x_1^N|\mathcal{M}) = \prod_{s \in \mathcal{M}} P_{kt}(a_s, b_s)$ and $P(\mathcal{M}) = 2^{-(2|\mathcal{M}|-1)}$.
- There is one tree-model of depth 0 (i.e. the IID model). If there are $\#_d$ models of depth not exceeding d then there are $\#_d^2 + 1$ models of depth not exceeding $d+1$. Therefore $\#_2 = 5$, $\#_3 = 26$, $\#_4 = 677$, $\#_5 = 458330$, $\#_6 = 210066388901$, $\#_7 = 4.4128 \cdot 10^{22}$, $\#_8 = 1.9473 \cdot 10^{45}$, etc.
- Straightforward analysis. No model-estimation that only gives asymptotic results as in e.g. Rissanen [1983, 1986], Weinberger, Rissanen, and Feder [1995]).
- Number of computations needed to process the source sequence x_1^N is linear in N. Same holds for the storage complexity.
- Optimal parameter redundancy behavior in Rissanen [1984] sense (i.e., $\frac{1}{2}\log_2 N$ bits/parameter).

Remarks Context-Tree Weighting

- A modified version achieves entropy not only for tree sources but for **all stationary ergodic sources**.
- A two-pass version **(context-tree maximizing)** exists that finds the minimum description length **(MDL)** model, matching to the source sequence. Now

$$P_m(s) \triangleq \frac{\max[P_{kt}(a_s, b_s), P_m(0s) \cdot P_m(1s)]}{2}.$$

  If a tree source with model generates the sequence x_1^N the maximizing method produces a model estimate which is correct with probability one as $N \to \infty$. The two-pass version achieves again

$$\rho(x_1^N) < (2|\mathcal{M}| - 1) + \sum_{l \in \mathcal{M}} \frac{1}{2} \log_2(a_l + b_l) + 2 \text{ bits}.$$

- If instead of P_{kt} we would use "uniform weight coding"

$$P_u(x_1^N) = \frac{1}{N+1} \frac{1}{\binom{N}{w}}$$

  as coding probability we get similar redundancy results with the problems mentioned before at the borders.

Burrows Wheeler Transform (1)

The Burrows Wheeler Transform is a TRANSFORM! The transformed sequence will be coded.

Let $x_1^N = 100101000110$ again. Consider x_1^N and all its cyclic left rotations.

1	1	0	0	1	0	1	0	0	0	1	1	0
2	0	0	1	0	1	0	0	0	1	1	0	1
3	0	1	0	1	0	0	0	1	1	0	1	0
4	1	0	1	0	0	0	1	1	0	1	0	0
5	0	1	0	0	0	1	1	0	1	0	0	1
6	1	0	0	0	1	1	0	1	0	0	1	0
7	0	0	0	1	1	0	1	0	0	1	0	1
8	0	0	1	1	0	1	0	0	1	0	1	0
9	0	1	1	0	1	0	0	1	0	1	0	0
10	1	1	0	1	0	0	1	0	1	0	0	0
11	1	0	1	0	0	1	0	1	0	0	0	1
12	0	1	0	0	1	0	1	0	0	0	1	1

Burrows Wheeler Transform (2)

Sort lexicographically based on the suffixes.

12	0	1	0	0	1	0	1	0	0	0	1	1
2	0	0	1	0	1	0	0	0	1	1	0	1
7	0	0	0	1	1	0	1	0	0	1	0	1
5	0	1	0	0	0	1	1	0	1	0	0	1
11	1	0	1	0	0	1	0	1	0	0	0	1
1	1	0	0	1	0	1	0	0	0	1	1	0
3	0	1	0	1	0	0	0	1	1	0	1	0
8	0	0	1	1	0	1	0	0	1	0	1	0
6	1	0	0	0	1	1	0	1	0	0	1	0
4	1	0	1	0	0	0	1	1	0	1	0	0
9	0	1	1	0	1	0	0	1	0	1	0	0
10	1	1	0	1	0	0	1	0	1	0	0	0

Burrows Wheeler Transform (3)

Now the left-most column is the BW-transform of x_1^N. Hence $BW(100101000110) = (101100110000)$.

12	0	1	0	0	1	0	1	0	0	0	1	1
2	0	0	1	0	1	0	0	0	1	1	0	1
7	0	0	0	1	1	0	1	0	0	1	0	1
5	0	1	0	0	0	1	1	0	1	0	0	1
11	1	0	1	0	0	1	0	1	0	0	0	1
1	1	0	0	1	0	1	0	0	0	1	1	0
3	0	1	0	1	0	0	0	1	1	0	1	0
8	0	0	1	1	0	1	0	0	1	0	1	0
6	1	0	0	0	1	1	0	1	0	0	1	0
4	1	0	1	0	0	0	1	1	0	1	0	0
9	0	1	1	0	1	0	0	1	0	1	0	0
10	1	1	0	1	0	0	1	0	1	0	0	0

The index $i_{bw}(x_1^N) = 6$ of x_1^N in the ordering is later needed for reconstruction.

Burrows Wheeler Reconstruction (1)

$$
\begin{array}{|c|ccccccccccc}
0 & . & . & . & . & . & . & . & . & . & . & . \\
0 & . & . & . & . & . & . & . & . & . & . & . \\
0 & . & . & . & . & . & . & . & . & . & . & . \\
0 & . & . & . & . & . & . & . & . & . & . & . \\
1 & . & . & . & . & . & . & . & . & . & . & . \\
1 & . & . & . & . & . & . & . & . & . & . & . \\
0 & . & . & . & . & . & . & . & . & . & . & . \\
0 & . & . & . & . & . & . & . & . & . & . & . \\
1 & . & . & . & . & . & . & . & . & . & . & . \\
1 & . & . & . & . & . & . & . & . & . & . & . \\
0 & . & . & . & . & . & . & . & . & . & . & . \\
1 & . & . & . & . & . & . & . & . & . & . & .
\end{array}
$$

We see 7 times 0, and 5 times 1 in the BW-transform, sorting yields the rightmost column.

Burrows Wheeler Reconstruction (2)

$$
\begin{array}{|c|cccccccccccc}
0 & . & . & . & . & . & . & . & . & . & . & 1 \\
0 & . & . & . & . & . & . & . & . & . & . & 1 \\
0 & . & . & . & . & . & . & . & . & . & . & 1 \\
0 & . & . & . & . & . & . & . & . & . & . & 1 \\
1 & . & . & . & . & . & . & . & . & . & . & 1 \\
1 & . & . & . & . & . & . & . & . & . & . & 0 \\
0 & . & . & . & . & . & . & . & . & . & . & 0 \\
0 & . & . & . & . & . & . & . & . & . & . & 0 \\
1 & . & . & . & . & . & . & . & . & . & . & 0 \\
1 & . & . & . & . & . & . & . & . & . & . & 0 \\
0 & . & . & . & . & . & . & . & . & . & . & 0 \\
1 & . & . & . & . & . & . & . & . & . & . & 0
\end{array}
$$

Observe that, in all rows, a symbol in the leftmost column (BW-transform) follows the symbol in the rightmost column.
We then see 3 times 00, 4 times 10, 4 times 01, and 1 times 11. Sorting yields the two rightmost columns.

Burrows Wheeler Reconstruction (3)

$$\begin{array}{|c|ccccccccccc}
0 & . & . & . & . & . & . & . & . & . & 1 & 1 \\
0 & . & . & . & . & . & . & . & . & . & 0 & 1 \\
0 & . & . & . & . & . & . & . & . & . & 0 & 1 \\
0 & . & . & . & . & . & . & . & . & . & 0 & 1 \\
1 & . & . & . & . & . & . & . & . & . & 0 & 1 \\
1 & . & . & . & . & . & . & . & . & . & 1 & 0 \\
0 & . & . & . & . & . & . & . & . & . & 1 & 0 \\
0 & . & . & . & . & . & . & . & . & . & 1 & 0 \\
1 & . & . & . & . & . & . & . & . & . & 1 & 0 \\
1 & . & . & . & . & . & . & . & . & . & 0 & 0 \\
0 & . & . & . & . & . & . & . & . & . & 0 & 0 \\
1 & . & . & . & . & . & . & . & . & . & 0 & 0
\end{array}$$

Again, in all rows, a symbol in the leftmost column (BW-transform) follows the corresponding symbols in the two rightmost columns.
We then see 1 times 000, 2 times 100, 3 times 010, 1 times 110, 2 times 001, 2 times 101, 1 times 011, and 0 times 000. Sorting yields the three rightmost columns.

Burrows Wheeler Reconstruction (4)

$$
\begin{array}{|c|cccccccccc}
0 & . & . & . & . & . & . & . & . & 0 & 1 & 1 \\
0 & . & . & . & . & . & . & . & . & 1 & 0 & 1 \\
0 & . & . & . & . & . & . & . & . & 1 & 0 & 1 \\
0 & . & . & . & . & . & . & . & . & 0 & 0 & 1 \\
1 & . & . & . & . & . & . & . & . & 0 & 0 & 1 \\
1 & . & . & . & . & . & . & . & . & 1 & 1 & 0 \\
0 & . & . & . & . & . & . & . & . & 0 & 1 & 0 \\
0 & . & . & . & . & . & . & . & . & 0 & 1 & 0 \\
1 & . & . & . & . & . & . & . & . & 0 & 1 & 0 \\
1 & . & . & . & . & . & . & . & . & 1 & 0 & 0 \\
0 & . & . & . & . & . & . & . & . & 1 & 0 & 0 \\
1 & . & . & . & . & . & . & . & . & 0 & 0 & 0
\end{array}
$$

Burrows Wheeler Reconstruction (5)

$$
\begin{array}{|c|cccccccccccc}
0 & . & . & . & . & . & . & . & 0 & 0 & 1 & 1 \\
0 & . & . & . & . & . & . & . & 1 & 1 & 0 & 1 \\
0 & . & . & . & . & . & . & . & 0 & 1 & 0 & 1 \\
0 & . & . & . & . & . & . & . & 1 & 0 & 0 & 1 \\
1 & . & . & . & . & . & . & . & 0 & 0 & 0 & 1 \\
1 & . & . & . & . & . & . & . & 0 & 1 & 1 & 0 \\
0 & . & . & . & . & . & . & . & 1 & 0 & 1 & 0 \\
0 & . & . & . & . & . & . & . & 1 & 0 & 1 & 0 \\
1 & . & . & . & . & . & . & . & 0 & 0 & 1 & 0 \\
1 & . & . & . & . & . & . & . & 0 & 1 & 0 & 0 \\
0 & . & . & . & . & . & . & . & 0 & 1 & 0 & 0 \\
1 & . & . & . & . & . & . & . & 1 & 0 & 0 & 0
\end{array}
$$

Burrows Wheeler Reconstruction (6)

$$\left|\begin{array}{c|cccccccccccc} 0 & . & . & . & . & . & . & 0 & 0 & 0 & 1 & 1 \\ 0 & . & . & . & . & . & . & 0 & 1 & 1 & 0 & 1 \\ 0 & . & . & . & . & . & . & 0 & 0 & 1 & 0 & 1 \\ 0 & . & . & . & . & . & . & 0 & 1 & 0 & 0 & 1 \\ 1 & . & . & . & . & . & . & 1 & 0 & 0 & 0 & 1 \\ 1 & . & . & . & . & . & . & 0 & 0 & 1 & 1 & 0 \\ 0 & . & . & . & . & . & . & 1 & 1 & 0 & 1 & 0 \\ 0 & . & . & . & . & . & . & 0 & 1 & 0 & 1 & 0 \\ 1 & . & . & . & . & . & . & 1 & 0 & 0 & 1 & 0 \\ 1 & . & . & . & . & . & . & 1 & 0 & 1 & 0 & 0 \\ 0 & . & . & . & . & . & . & 1 & 0 & 1 & 0 & 0 \\ 1 & . & . & . & . & . & . & 0 & 1 & 0 & 0 & 0 \end{array}\right.$$

Burrows Wheeler Reconstruction (7)

0	.	.	.	.	.	1	0	0	0	1	1
0	.	.	.	.	.	0	0	1	1	0	1
0	.	.	.	.	.	1	0	0	1	0	1
0	.	.	.	.	.	1	0	1	0	0	1
1	.	.	.	.	.	0	1	0	0	0	1
1	.	.	.	.	.	0	0	0	1	1	0
0	.	.	.	.	.	0	1	1	0	1	0
0	.	.	.	.	.	0	0	1	0	1	0
1	.	.	.	.	.	0	1	0	0	1	0
1	.	.	.	.	.	1	1	0	1	0	0
0	.	.	.	.	.	0	1	0	1	0	0
1	.	.	.	.	.	1	0	1	0	0	0

Burrows Wheeler Reconstruction (8)

0	.	.	.	.	0	1	0	0	0	1	1
0	.	.	.	.	0	0	0	1	1	0	1
0	.	.	.	.	0	1	0	0	1	0	1
0	.	.	.	.	1	1	0	1	0	0	1
1	.	.	.	.	1	0	1	0	0	0	1
1	.	.	.	.	1	0	0	0	1	1	0
0	.	.	.	.	0	0	1	1	0	1	0
0	.	.	.	.	1	0	0	1	0	1	0
1	.	.	.	.	1	0	1	0	0	1	0
1	.	.	.	.	0	1	1	0	1	0	0
0	.	.	.	.	0	0	1	0	1	0	0
1	.	.	.	.	0	1	0	1	0	0	0

Burrows Wheeler Reconstruction (9)

$$
\begin{array}{|c|ccccccccccc}
0 & . & . & . & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
0 & . & . & . & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\
0 & . & . & . & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\
0 & . & . & . & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\
1 & . & . & . & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\
1 & . & . & . & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & . & . & . & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
0 & . & . & . & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\
1 & . & . & . & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\
1 & . & . & . & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
0 & . & . & . & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
1 & . & . & . & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0
\end{array}
$$

Burrows Wheeler Reconstruction (10)

$$
\left|\begin{array}{c|ccccccccccc}
0 & . & . & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
0 & . & . & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\
0 & . & . & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\
0 & . & . & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\
1 & . & . & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\
1 & . & . & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & . & . & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
0 & . & . & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\
1 & . & . & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\
1 & . & . & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
0 & . & . & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
1 & . & . & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0
\end{array}\right.
$$

Burrows Wheeler Reconstruction (11)

$$
\begin{array}{|c|cccccccccccc}
0 & . & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
0 & . & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\
0 & . & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\
0 & . & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\
1 & . & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\
1 & . & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & . & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
0 & . & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\
1 & . & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\
1 & . & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
0 & . & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
1 & . & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0
\end{array}
$$

Burrows Wheeler Reconstruction (12)

$$
\left|\begin{array}{c|cccccccccccc}
0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0
\end{array}\right.
$$

Burrows Wheeler Reconstruction (13)

11	0	1	0	0	1	0	1	0	0	0	1	1
10	0	0	1	0	1	0	0	0	1	1	0	1
9	0	0	0	1	1	0	1	0	0	1	0	1
8	0	1	0	0	0	1	1	0	1	0	0	1
7	1	0	1	0	0	1	0	1	0	0	0	1
6	1	0	0	1	0	1	0	0	0	1	1	0
5	0	1	0	1	0	0	0	1	1	0	1	0
4	0	0	1	1	0	1	0	0	1	0	1	0
3	1	0	0	0	1	1	0	1	0	0	1	0
2	1	0	1	0	0	0	1	1	0	1	0	0
1	0	1	1	0	1	0	0	1	0	1	0	0
0	1	1	0	1	0	0	1	0	1	0	0	0

From the index $i_{bw}(x_1^N) = 6$ of x_1^N we get the original sequence back.

Coding the BW-transformed sequence

IDEA:

For tree sources, the transformed sequence $BW(x_1^N)$ is a concatenation of IID subsequences, as many as there are leaves in the tree model.[a]

[a] Circularity effects are ignored.

Example ($x_1^N = 100101000110$ and tree-model $\mathcal{M} = \{00, 10, 1\}$)

$BW(100101000110) = 101, 1001, 10000$. See

12	0	1	0	0	1	0	1	0	0	0	1	1
2	0	0	1	0	1	0	0	0	1	1	0	1
7	0	0	0	1	1	0	1	0	0	1	0	1
5	0	1	0	0	0	1	1	0	1	0	0	1
11	1	0	1	0	0	1	0	1	0	0	0	1
1	1	0	0	1	0	1	0	0	0	1	1	0
3	0	1	0	1	0	0	0	1	1	0	1	0
8	0	0	1	1	0	1	0	0	1	0	1	0
6	1	0	0	0	1	1	0	1	0	0	1	0
4	1	0	1	0	0	0	1	1	0	1	0	0
9	0	1	1	0	1	0	0	1	0	1	0	0
10	1	1	0	1	0	0	1	0	1	0	0	0

Coding the BW-transformed sequence

IDEA:

For tree sources, the transformed sequence $BW(x_1^N)$ is a concatenation of IID subsequences, as many as there are leaves in the tree model.[a]

[a] Circularity effects are ignored.

Example ($x_1^N = 100101000110$ and tree-model $\mathcal{M} = \{00, 10, 1\}$)

$BW(100101000110) = 101, 1001, 10000$. See

12	0	1	0	0	1	0	1	0	0	0	1	1
2	0	0	1	0	1	0	0	0	1	1	0	1
7	0	0	0	1	1	0	1	0	0	1	0	1
5	0	1	0	0	0	1	1	0	1	0	0	1
11	1	0	1	0	0	1	0	1	0	0	0	1
1	1	0	0	1	0	1	0	0	0	1	1	0
3	0	1	0	1	0	0	0	1	1	0	1	0
8	0	0	1	1	0	1	0	0	1	0	1	0
6	1	0	0	0	1	1	0	1	0	0	1	0
4	1	0	1	0	0	0	1	1	0	1	0	0
9	0	1	1	0	1	0	0	1	0	1	0	0
10	1	1	0	1	0	0	1	0	1	0	0	0

Coding the BW-transformed sequence

- Move-to-front techniques (Bentley et al. (1986), Elias (1987)) combined with Huffman codes can be used.
- Piecewise IID coding (Merhav (1993), W. (1996)) can be used. These methods demonstrate that we need to specify the transition points, which requires roughly $\log_2 N$ bits per transition (Effros et al. (2002)).
- CTW only needs $2|\mathcal{M}| - 1$ bits, which is roughly 2 bits per transition.

OBJECTIVE:

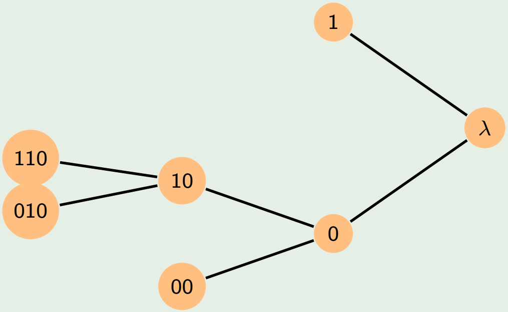
Design a procedure for coding the BW transformed sequence that needs only $2|\mathcal{M}| - 1$ bits.

FSM-Closed Tree Models

Definition

The generator of leaf $u_d u_{d-1} \cdots u_1$ at depth d is $u_d u_{d-1} \cdots u_2$ at depth $d-1$. A tree model is FSM-closed if all its leaves have a generator that is either a leaf or internal node of the tree model.

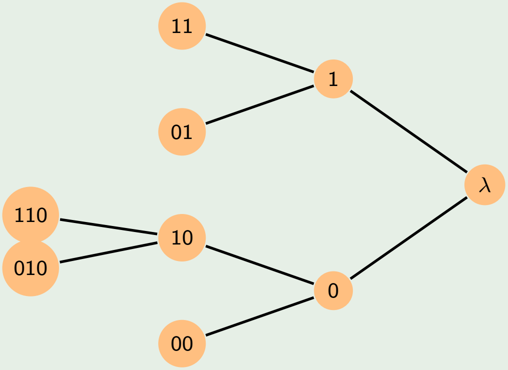
Example (Tree model $\mathcal{M} = \{00, 010, 110, 1\}$)



Leaf 00 has generator 0 which is an internal node of $\mathcal{M}$. But note that leaves 110 and 010 do not have a generator in tree model $\mathcal{M}$. Therefore $\mathcal{M}$ is not FSM-closed and we cannot describe the source as a FSM.

FSM-Closed Tree Models

Example (Tree model $\mathcal{M}' = \{00, 010, 110, 01, 11\}$)



Tree model $\mathcal{M}'$ is FSM-closed.

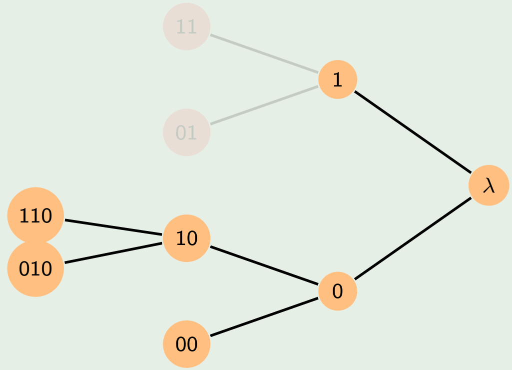
If a tree model is FSM-closed then each leaf and each node in the tree model has a generator that is a leaf or node of the tree model.

FSM-Closed Tree Models

Definition

A tree model can be made FSM-closed by adding the generators of all leaves. The resulting model is the FSM-closure of the tree model.

Example

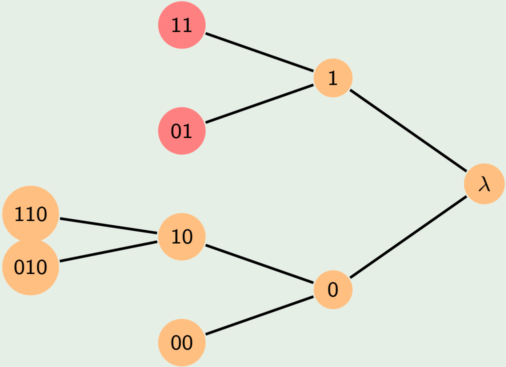


FSM-Closed Tree Models

Definition

A tree model can be made FSM-closed by adding the generators of all leaves. The resulting model is the FSM-closure of the tree model.

Example



Coding BW-sequences: PROBLEM

Suppose that $\widehat{\mathcal{M}}$ is the tree-model that matches best to the BW-sequence. More about how to find $\widehat{\mathcal{M}}$ later.

- This tree model $\widehat{\mathcal{M}}$ is included in the description ($2|\widehat{\mathcal{M}}| - 1$ bits).
- Apart from that the b-counts (weights of subsequences) of all leaves of $\widehat{\mathcal{M}}$ are added to the description.
- Finally the lexicographical indices of all the subsequences corresponding to the leaves of $\widehat{\mathcal{M}}$ are included.

Question:

Can we reconstruct the entire BW-sequence from the description using ENUMERATIVE techniques?

Definition

Let a_s the number of zeros and b_s the number of ones in the subsequence that corresponds to s, then

$$c_s = a_s + b_s$$

is length of this subsequence.

FSM-Closed Tree Model (1)

Suppose that the b-counts in the leaves of **FSM-closed tree model** $\widehat{\mathcal{M}} = \{00, 010, 110, 01, 11\}$ are given to the decoder, hence the decoder knows b_{00}, b_{010}, b_{110}, b_{01}, b_{11}, and N.

- The decoder first computes the b-counts in all the nodes of the tree model, hence b_λ, b_0, b_1, and b_{01}.
- The decoder now processes layer by layer, starting in the root (layer 0). First in the root the a-count is computed.

$$a_\lambda = N - b_\lambda.$$

  Layer 0 is processed now.
- In layer 1 there are two nodes, 0 and 1 that fit into $\widehat{\mathcal{M}}$. Since $\widehat{\mathcal{M}}$ is FSM-closed, their generator λ exists, is at level 0, and is thus processed before. Therefore we can compute the c's of these nodes.

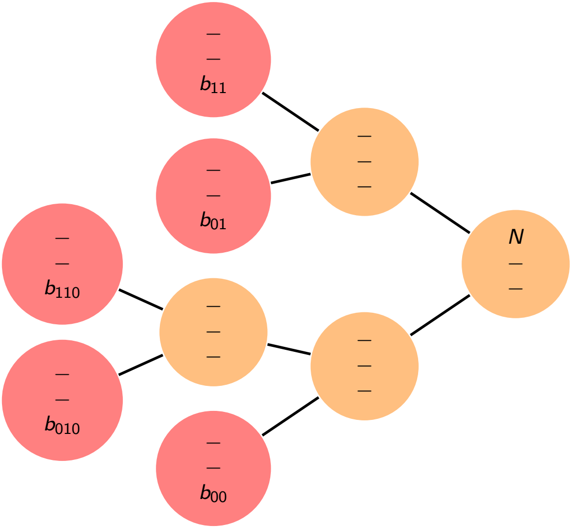
$$\begin{aligned} c_0 &= a_\lambda \\ c_1 &= b_\lambda. \end{aligned}$$

  With the available b-counts also the a-counts can be computed

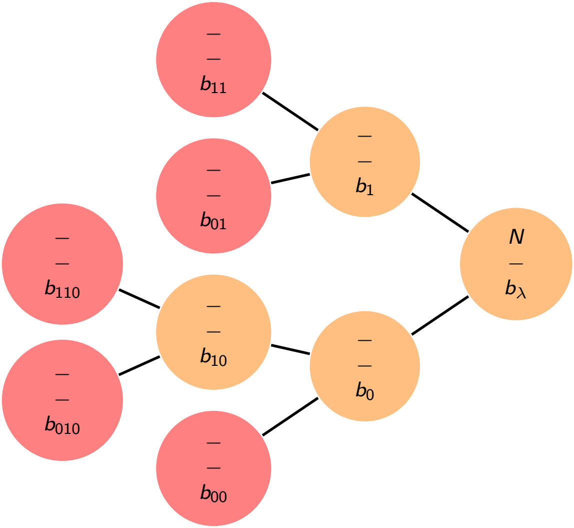
$$\begin{aligned} a_0 &= c_0 - b_0 \\ a_1 &= c_1 - b_1. \end{aligned}$$

  Layer 1 is processed now.

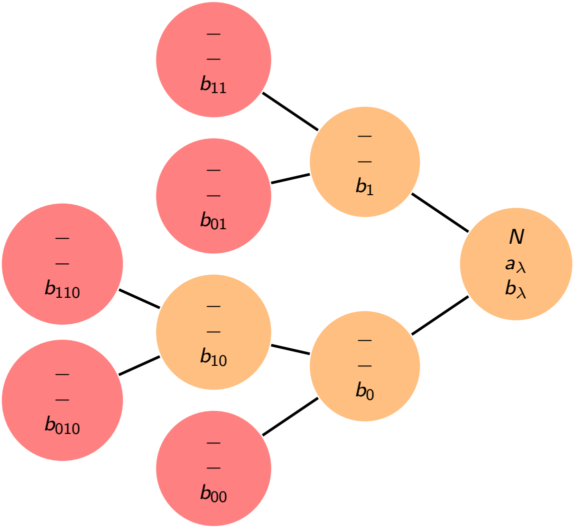
FSM-Closed Tree Model (2)



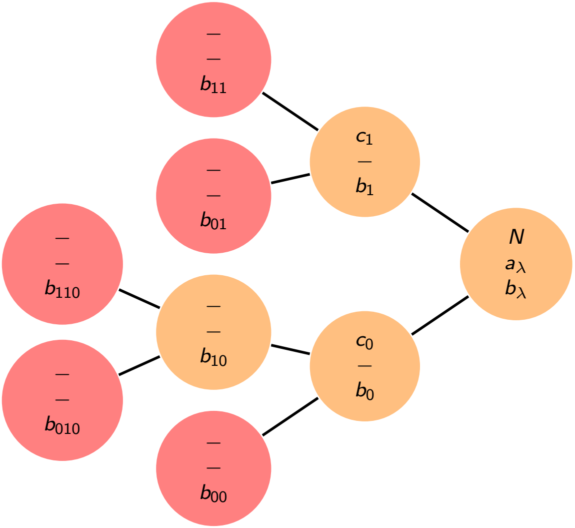
FSM-Closed Tree Model (3)



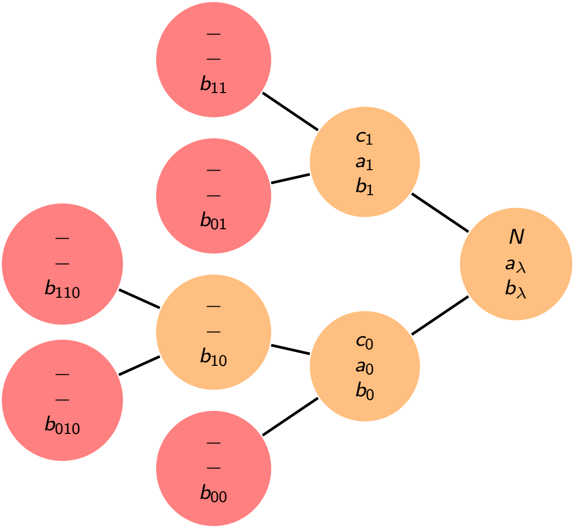
FSM-Closed Tree Model (4)



FSM-Closed Tree Model (5)



FSM-Closed Tree Model (6)



FSM-Closed Tree Model (7)

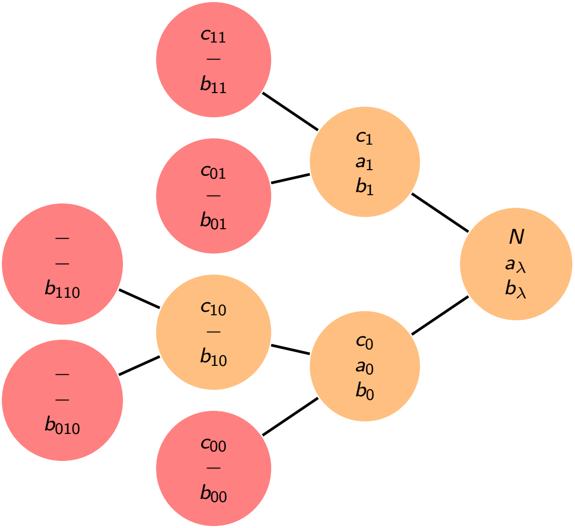
- In layer 2 there are three leaves, 00, 01, and 11, and a node 10, that fit into $\widehat{\mathcal{M}}$. Since $\widehat{\mathcal{M}}$ is FSM-closed, again their generators 0 and 1 exist, are at level 1, and are thus processed before. Now we can compute the c's of these leaves and node.

$$
\begin{aligned}
c_{00} &= a_0 \\
c_{10} &= a_1 \\
c_{01} &= b_0 \\
c_{11} &= b_1.
\end{aligned}
$$

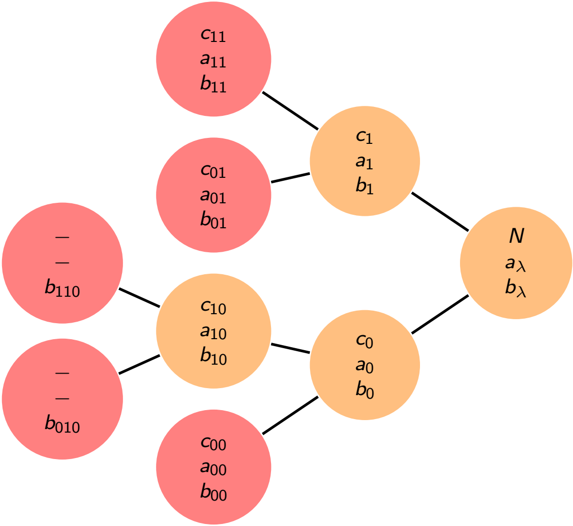
With the available b-counts also the a-counts can be computed for these leaves and node

$$
\begin{aligned}
a_{00} &= c_{00} - b_{00} \\
a_{10} &= c_{10} - b_{10} \\
a_{01} &= c_{01} - b_{01} \\
a_{11} &= c_{11} - b_{11}.
\end{aligned}
$$

FSM-Closed Tree Model (8)



FSM-Closed Tree Model (9)



FSM-Closed Tree Model (10)

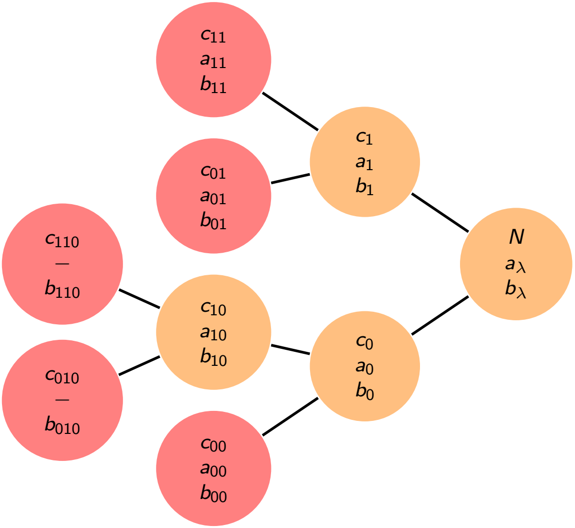
- In layer 3 there are two leaves, 010, and 110, that fit into $\widehat{\mathcal{M}}$. Since $\widehat{\mathcal{M}}$ is FSM-closed, again their generators 01 and 11 exist, are at level 2, and are thus processed before. Now we can compute the c's of the subsequences in these two leaves.

$$\begin{aligned} c_{010} &= a_{01} \\ c_{110} &= a_{11}. \end{aligned}$$

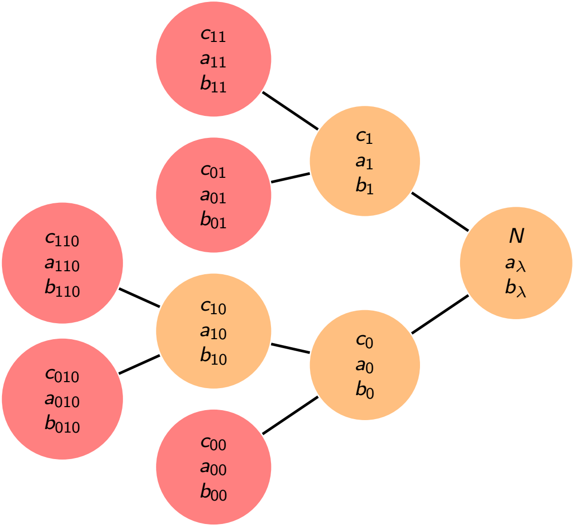
With the available b-counts also the a-counts can be computed for these leaves

$$\begin{aligned} a_{010} &= c_{010} - b_{010} \\ a_{110} &= c_{110} - b_{110}. \end{aligned}$$

FSM-Closed Tree Model (11)



FSM-Closed Tree Model (12)



FSM-Closed Tree Model (13)

- For all leaves s in $\widehat{\mathcal{M}}$ the subsequence length c_s and b-count b_s is known. Note that the b-count is the weight of the subsequence.
- Now with the indices i_{00}, i_{010}, i_{110}, i_{01}, and i_{11}, the corresponding subsequences $\underline{bw}_{00}$, $\underline{bw}_{010}$, $\underline{bw}_{110}$, $\underline{bw}_{01}$, and $\underline{bw}_{11}$, can be reconstructed.
- The BW-transformed sequence is now the concatenation of the five subsequences, hence

$$BW = \underline{bw}_{00}, \underline{bw}_{010}, \underline{bw}_{110}, \underline{bw}_{01}, \underline{bw}_{11}.$$

NOTE that this approach is outlined in Martin, Seroussi, & Weinberger (2004), with emphasis on the FSM-case, not on the BWT-case.

Tree Model, not FSM-Closed (1)

Suppose that the b-counts in the leaves of tree model $\widehat{\mathcal{M}} = \{00, 010, 110, 1\}$, which is **not FSM-closed**, are given to the decoder, hence the decoder knows b_{00}, b_{010}, b_{110}, b_1, and N.

- The decoder first computes the b-counts in all the nodes of the tree model, hence b_λ, b_0, and b_{10}.
- The decoder now processes layer by layer, starting in the root (layer 0). First in the root the a-count is computed.

$$a_\lambda = N - b_\lambda.$$

Now all the nodes in layer 0 are processed.

- In layer 1 there are two nodes, 0 and 1, that fit into $\widehat{\mathcal{M}}$. Since "all nodes in layer 0" are complete, we can compute the c's of the nodes 0 and 1.

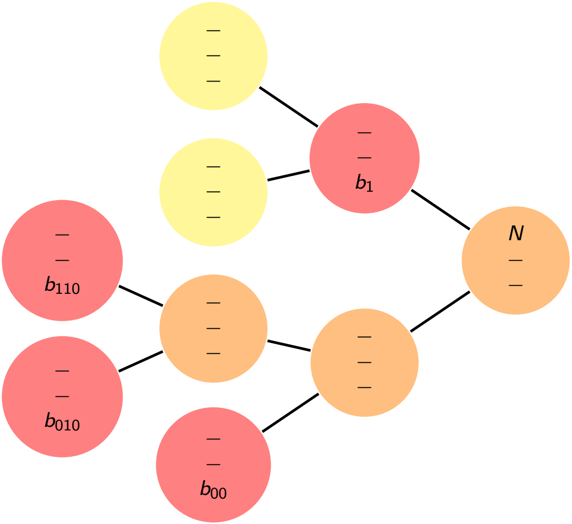
$$\begin{aligned} c_0 &= a_\lambda \\ c_1 &= b_\lambda. \end{aligned}$$

With the available b-counts also the a-counts can be computed for these nodes

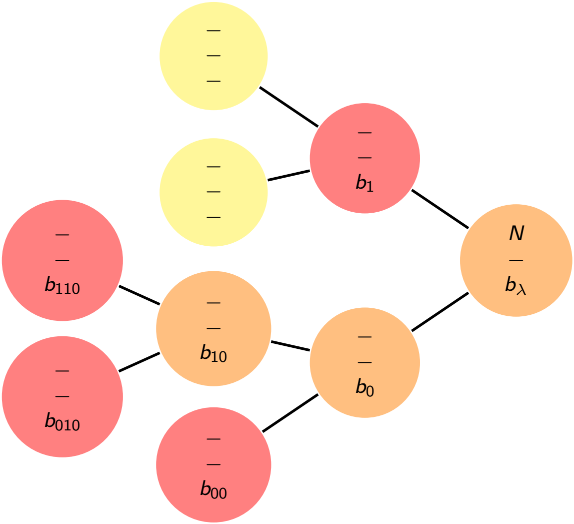
$$\begin{aligned} a_0 &= c_0 - b_0 \\ a_1 &= c_1 - b_1. \end{aligned}$$

Now all the nodes in layer 1 are processed.

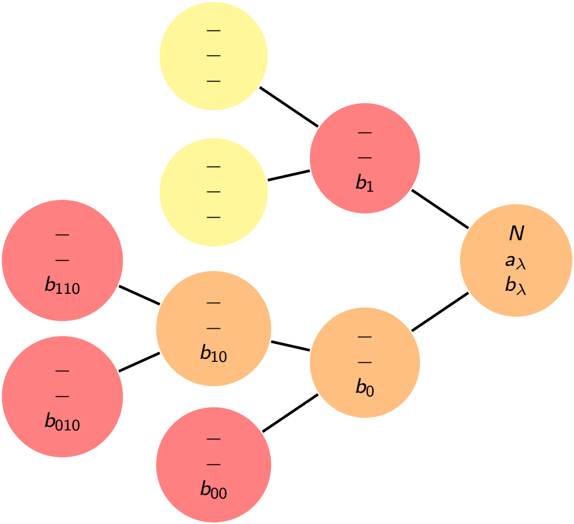
Tree Model not FSM-Closed (2)



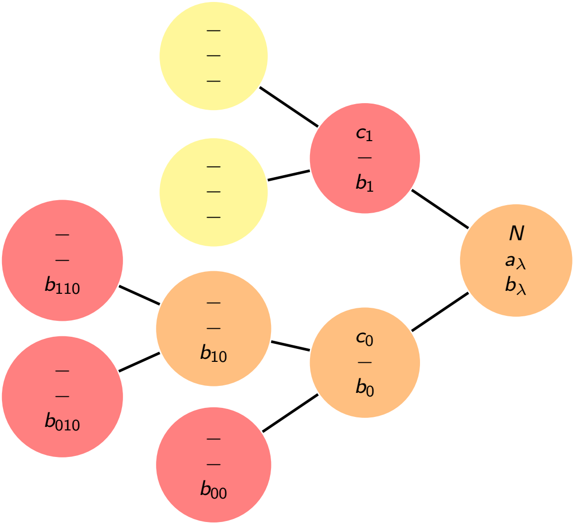
Tree Model not FSM-Closed (3)



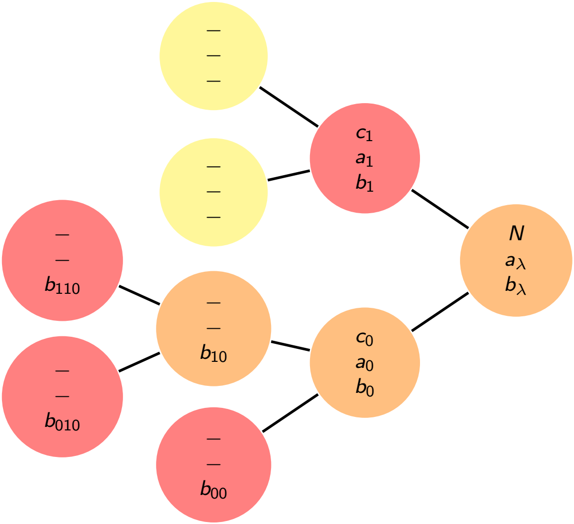
Tree Model, not FSM-Closed (4)



Tree Model, not FSM-Closed (5)



Tree Model, not FSM-Closed (6)



Tree Model, not FSM-Closed (7)

- In layer 2 there is a leaf, 00, and a node 10, that fit into $\widehat{\mathcal{M}}$. Since all nodes at layer 1 are complete, we can compute the c's of the subsequences in this leaf and node.

$$\begin{aligned} c_{00} &= a_0 \\ c_{10} &= a_1. \end{aligned}$$

  With the available b-counts for the node and leaf that fit into $\widehat{\mathcal{M}}$ the a-counts can be computed

$$\begin{aligned} a_{00} &= c_{00} - b_{00} \\ a_{10} &= c_{10} - b_{10}. \end{aligned}$$

- In layer 2 there are two nodes, 01 and 11, **that do not fit into** $\mathcal{M}$. Since all nodes at layer 1 are processed, we can also compute the c's of these two nodes.

$$\begin{aligned} c_{01} &= b_0 \\ c_{11} &= b_1. \end{aligned}$$

  We need b_{01} and b_{11} now ...

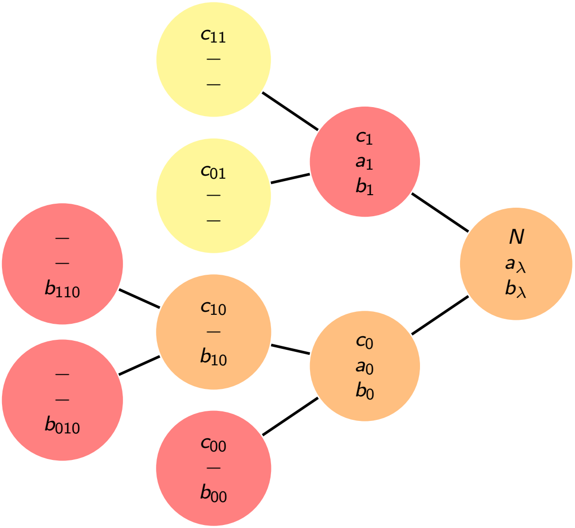
Tree Model, not FSM-Closed (8)

New Approach

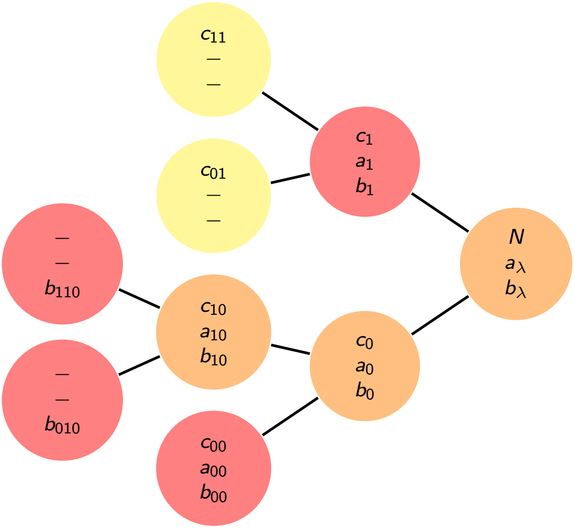
- Leaf 1 at layer 1 is complete (we know c_1 and weight b_1), and therefore the corresponding subsequence $\underline{bw}_1$ can be reconstructed from the lexicographical index i_1.
-
  - The first c_{01} digits of $\underline{bw}_1$ are digits corresponding to node 01, call this sequence $\underline{bw}_{01}$. We can now simply count the number a_{01} of 0-digits and the number b_{01} of 1-digits in $\underline{bw}_{01}$.
  - The last c_{11} digits of $\underline{bw}_1$ are digits corresponding to node 11, call this sequence $\underline{bw}_{11}$. We can now simply count the number a_{11} of 0-digits and the number b_{11} of 1-digits in $\underline{bw}_{11}$.

  Now all the nodes in layer 2 are processed.

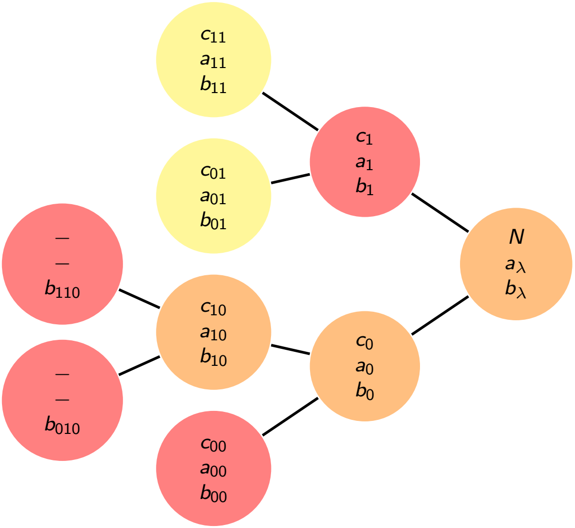
Tree Model, not FSM-Closed (9)



Tree Model, not FSM-Closed (10)



Tree Model, not FSM-Closed (11)



Tree Model, not FSM-Closed (12)

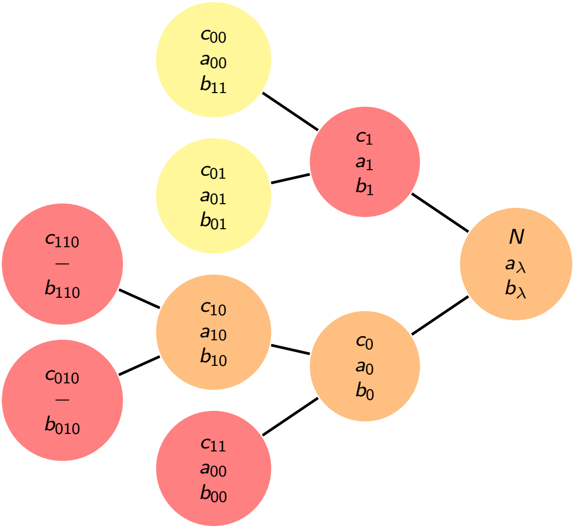
- In layer 3 there are two leaves, 010, and 110, that fit into $\widehat{\mathcal{M}}$. Since all nodes at level 2 are complete, we can compute the c's of these leaves.

$$\begin{aligned} c_{010} &= a_{01} \\ c_{110} &= a_{11}. \end{aligned}$$

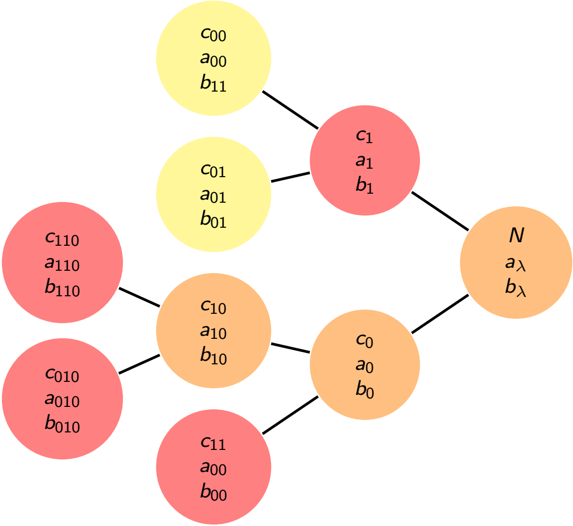
With the available b-counts also the a-counts can be computed for these leaves and node

$$\begin{aligned} a_{010} &= c_{010} - b_{010} \\ a_{110} &= c_{110} - b_{110}. \end{aligned}$$

Tree Model, not FSM-Closed (13)



Tree Model, not FSM-Closed (14)



Tree Model, not FSM-Closed (15)

- For all remaining leaves s in $\widehat{\mathcal{M}}$ the subsequence length c_s and b-count b_s (weight) is known.
- Now with the indices i_{00}, i_{010}, and i_{110}, the corresponding subsequences $\underline{bw}_{00}$, $\underline{bw}_{010}$, and $\underline{bw}_{110}$, can be reconstructed.
- Since $\underline{bw}_1$ was reconstructed before, the BW-transformed sequence is now the concatenation of the four subsequences, hence

$$BW = \underline{bw}_{00}, \underline{bw}_{010}, \underline{bw}_{110}, \underline{bw}_1.$$

NOTE that the new approach works in the non-FSM closed case, without having to form the FSM-closure of the tree model first.

Coding b-Counts: Objective

Objective

How do we code the b-counts in the leaves $\mathcal{L}(\widehat{\mathcal{M}})$? Ideally we would like to achieve total codewordlength

$$\sum_{l \in \mathcal{L}(\widehat{\mathcal{M}})} \log_2(a_l + b_l + 1).$$

This corresponds to **uniform weight coding**. Note that the c's apart from $c_\lambda = N$, are still unknown.

Coding b-Counts: Procedure

- We start in the root node λ. There we describe b_λ. To this we need
$$\lceil \log_2(a_\lambda + b_\lambda + 1) \rceil \text{ bits.}$$
- Then in all internal nodes s, starting in the root node λ, and processing level by level, we first find
$$i = \arg\min_{i=0,1} \log_2(a_{is} + b_{is} + 1),$$
and we use one bit to describe i. Then we describe b_{is} using
$$\lceil \log_2(a_{is} + b_{is} + 1) \rceil \text{ bits.}$$
Note that when b_{is} needs to be described, the corresponding $c_{0s} = a_{0s} + b_{0s}$ and $c_{1s} = a_{1s} + b_{1s}$ have already been reconstructed.

This procedure leads to

$$\lceil \log_2(a_\lambda + b_\lambda + 1) \rceil + \sum_{n \in \mathcal{N}(\widehat{\mathcal{M}})} \lceil \log_2 \min(a_{0n} + b_{0n} + 1, a_{1n} + b_{1n} + 1) \rceil \text{ bits,}$$

where $\mathcal{N}(\widehat{\mathcal{M}})$ are the internal nodes of $\widehat{\mathcal{M}}$.

Coding b-Counts: Loss per Node, Total Loss

Consider a node $n \in \mathcal{N}(\widehat{\mathcal{M}})$.

- Observe that we have described b_n using $\lceil \log_2(c_n+1) \rceil$ bits, and next we describe both b_{0n} and b_{1n} using $1 + \lceil \log_2 \min(c_{0n}+1, c_{1n}+1) \rceil$ bits.
- Directly describing b_{0n} and b_{1n} would require $\lceil \log_2(c_{0n}+1) \rceil + \lceil \log_2(c_{1n}+1) \rceil$ bits.
- The **loss in this node** is now

$$\begin{aligned} & \lceil \log_2(c_n+1) \rceil + 1 + \lceil \log_2 \min(c_{0n}+1, c_{1n}+1) \rceil \\ & \qquad - \lceil \log_2(c_{0n}+1) \rceil - \lceil \log_2(c_{1n}+1) \rceil \\ = & \quad \lceil \log_2(c_n+1) \rceil + 1 - \lceil \log_2 \max(c_{0n}+1, c_{1n}+1) \rceil \\ \leq & \quad 2 \text{ bit.} \end{aligned}$$

Total loss is therefore

$$2(|\widehat{\mathcal{M}}| - 1) \text{ bits.}$$

Finding Best Tree Model: Maximizing

A two-pass version **(context-tree maximizing)** exists that finds the best model **(MDL)** matching to the source sequence, if coding is done as described before.

Context-Tree Maximizing

Define for nodes n

$$\mu(n) \triangleq 1+\min\left[\left\lceil \log_2 \binom{c_n}{b_n} \right\rceil, 1+\lceil \log_2 \min(c_{0n}+1, c_{1n}+1)\rceil+\mu(0n)+\mu(1n)\right],$$

while for leaves l

$$\mu(l) \triangleq \left\lceil \log_2 \binom{c_l}{b_l} \right\rceil.$$

Tracking this procedure yield $\widehat{\mathcal{M}}$. Code-length is

$$L = \log_2(N+1) + \mu(\lambda).$$

This procedure can be carried out **very efficiently during the BW transform phase**.

Finding Best Tree Model: Integration with BW Transform

Fix a depth, e.g. 3, then the subsequences of the nodes at depth 3 can be easily found, see BW table below:

12	0	1	0	0	1	0	1	0	0	0	1	1
2	0	0	1	0	1	0	0	0	1	1	0	1
7	0	0	0	1	1	0	1	0	0	1	0	1
5	0	1	0	0	0	1	1	0	1	0	0	1
11	1	0	1	0	0	1	0	1	0	0	0	1
1	1	0	0	1	0	1	0	0	0	1	1	0
3	0	1	0	1	0	0	0	1	1	0	1	0
8	0	0	1	1	0	1	0	0	1	0	1	0
6	1	0	0	0	1	1	0	1	0	0	1	0
4	1	0	1	0	0	0	1	1	0	1	0	0
9	0	1	1	0	1	0	0	1	0	1	0	0
10	1	1	0	1	0	0	1	0	1	0	0	0

Now node 000 $\rightarrow \{10\} \rightarrow$ 1, node 100 $\rightarrow \{9,4\} \rightarrow$ 01, node 010 $\rightarrow \{6,8,3\} \rightarrow$ 100, node 110 $\rightarrow \{1\} \rightarrow$ 1, node 001 $\rightarrow \{5,11\} \rightarrow$ 10, node 101 $\rightarrow \{7,2\} \rightarrow$ 00, node 011 $\rightarrow \{12\} \rightarrow$ 0, and finally node 111 $\rightarrow \emptyset \rightarrow \emptyset$.

CTW: Binary Decomposition, 1.8 bit/ASCII.

- Bytes: 8 bits,
  ASCII-symbols: 7 bits,
  DNA nucleotide (A,T, G and C): 2 bits, etc.
- The **first bit of a symbol** depends on the context (a number of past symbols).
  The **second bit of a symbol** depends on the first bit of that symbol and the context.
  The **third bit of a symbol** depends on the first and second bit of that symbol and the context, etc.
- Therefore there is a **tree model for the first bit**.
  There are **two tree models for the second bit, one for first bit being 0 and a second one when the first bit is 1**.
  There are **four tree models for the third bit** etc.

Question:

Can we do a binary decomposition also in combination with a BW transform on the symbols? Does there exist a way of finding the c's based on only the b-counts?

Binary Decomposition: Example (1)

SYMBOL-SIZE: 2 bits.
Suppose that the b-counts in the leaves of the three tree models are specified. The decoder can now compute the b-counts of all inner nodes.

- The decoder first computes in tree-$\emptyset$ (corresponding to the first digit of the symbol) count $a_\lambda^\emptyset$

$$a_\lambda^\emptyset = N - b_\lambda^\emptyset.$$

  Now in tree-0 and in tree-1 (corresponding to the second digit of the symbol) the decoder can compute the c's and then the a's in the root nodes

$$\begin{aligned} c_\lambda^0 &= a_\lambda^\emptyset & \qquad a_\lambda^0 &= c_\lambda^0 - b_\lambda^0 \\ c_\lambda^1 &= b_\lambda^\emptyset & \qquad a_\lambda^1 &= c_\lambda^1 - b_\lambda^1. \end{aligned}$$

  Now **all the nodes in layer 0 of the three trees are processed**.

- The decoder now starts working on layer 1 (one symbol contexts). First in tree-$\emptyset$ the decoder computes the c's and then the a's

$$\begin{aligned} c_{00}^\emptyset &= a_\lambda^0 & \qquad a_{00}^\emptyset &= c_{00}^\emptyset - b_{00}^\emptyset \\ c_{01}^\emptyset &= b_\lambda^0 & \qquad a_{01}^\emptyset &= c_{01}^\emptyset - b_{01}^\emptyset \\ c_{10}^\emptyset &= a_\lambda^1 & \qquad a_{10}^\emptyset &= c_{10}^\emptyset - b_{10}^\emptyset \\ c_{11}^\emptyset &= b_\lambda^1 & \qquad a_{11}^\emptyset &= c_{11}^\emptyset - b_{11}^\emptyset \end{aligned}$$

Binary Decomposition: Example (2)

- Then in tree-0 the decoder computes the c's and then the a's

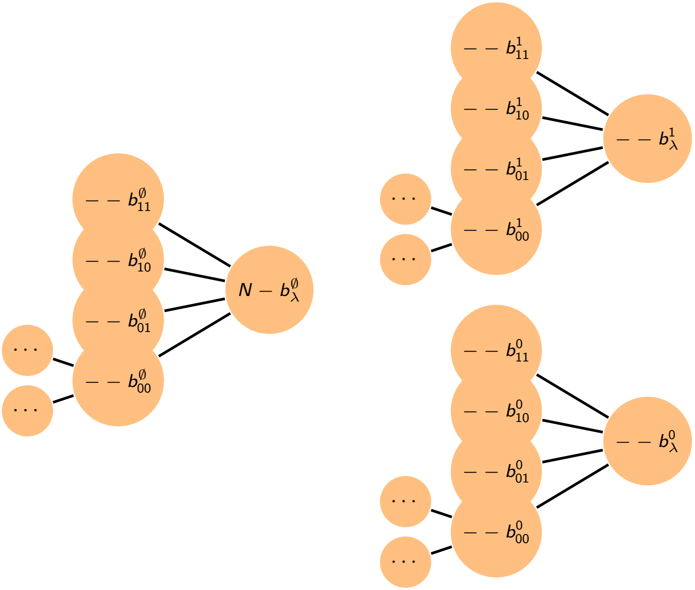
$$
\begin{aligned}
c^0_{00} &= a^{\emptyset}_{00} & \qquad a^0_{00} &= c^0_{00} - b^0_{00}\\
c^0_{01} &= a^{\emptyset}_{01} & \qquad a^0_{01} &= c^0_{01} - b^0_{01}\\
c^0_{10} &= a^{\emptyset}_{10} & \qquad a^0_{10} &= c^0_{10} - b^0_{10}\\
c^0_{11} &= a^{\emptyset}_{11} & \qquad a^0_{11} &= c^0_{11} - b^0_{11},
\end{aligned}
$$

  and in tree-1 the decoder computes the c's and then the a's

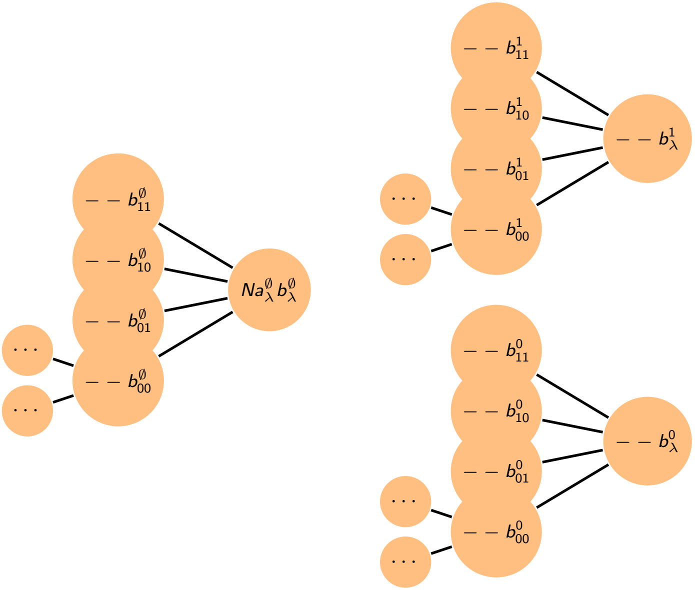
$$
\begin{aligned}
c^1_{00} &= b^{\emptyset}_{00} & \qquad a^1_{00} &= c^1_{00} - b^1_{00}\\
c^1_{01} &= b^{\emptyset}_{01} & \qquad a^1_{01} &= c^1_{01} - b^1_{01}\\
c^1_{10} &= b^{\emptyset}_{10} & \qquad a^1_{10} &= c^1_{10} - b^1_{10}\\
c^1_{11} &= b^{\emptyset}_{11} & \qquad a^1_{11} &= c^1_{11} - b^1_{11}.
\end{aligned}
$$

  Now **all the nodes in layer 1 of the three trees are processed**.
- Etc.
- If a leaf is encountered the corresponding subsequence is reconstructed using the index. From then on **a-counts and b-counts can be found by inspection of the reconstructed subsequence**.

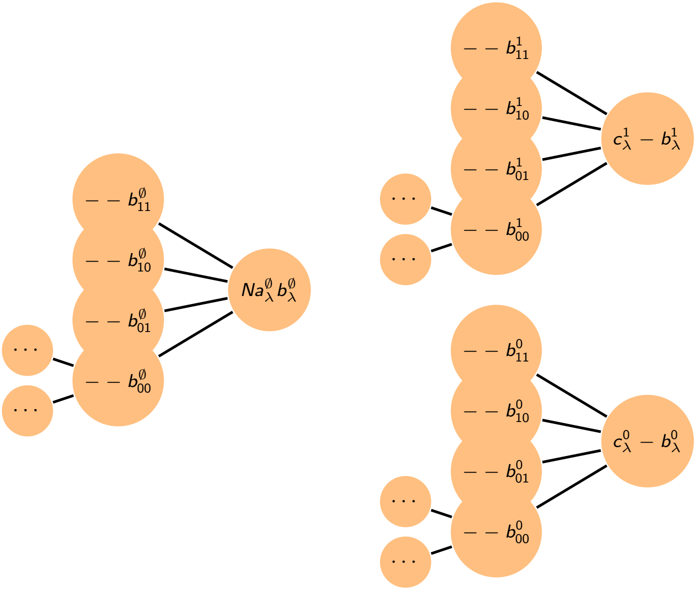
Binary Decomposition: Example (3)



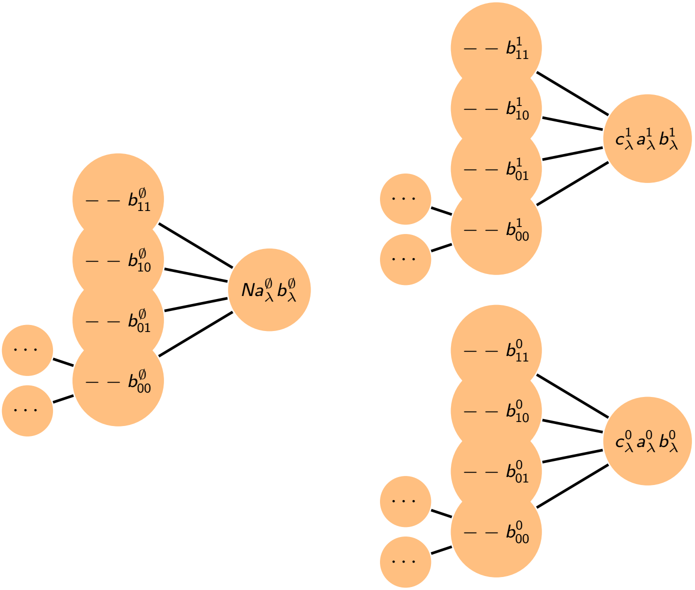
Binary Decomposition: Example (4)



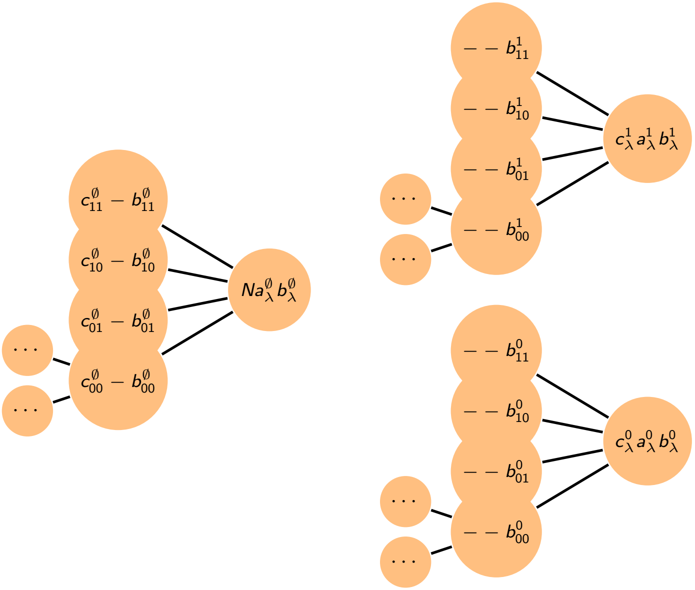
Binary Decomposition: Example (5)



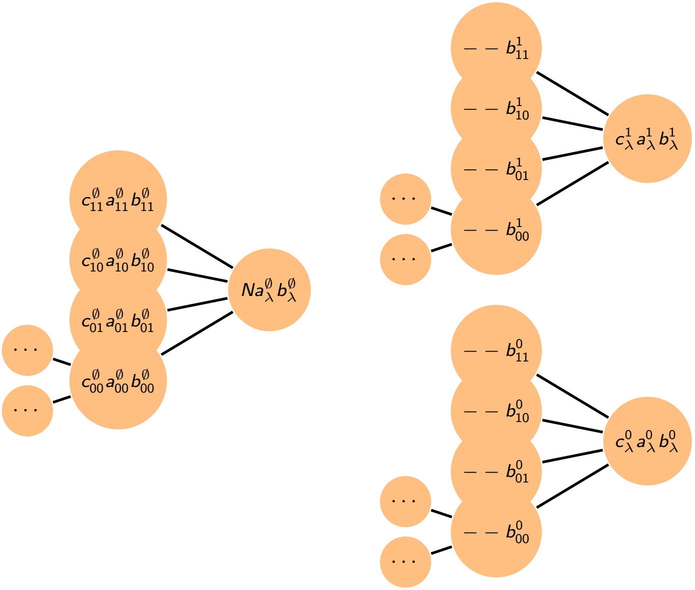
Binary Decomposition: Example (6)



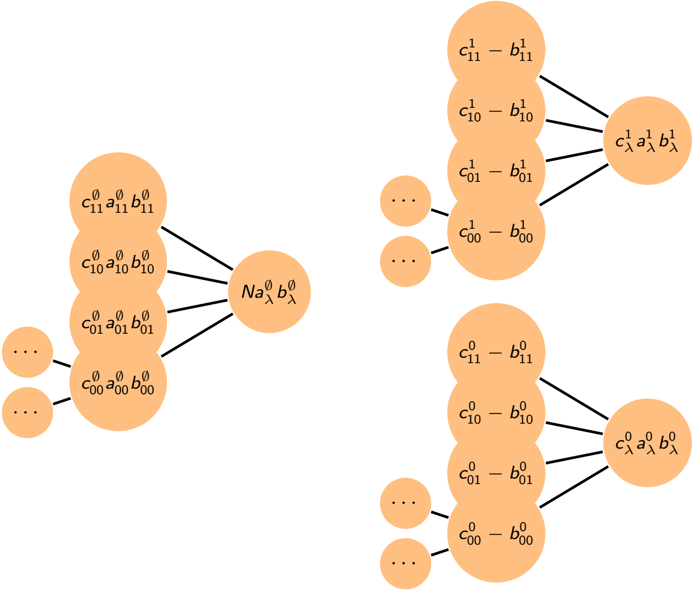
Binary Decomposition: Example (7)



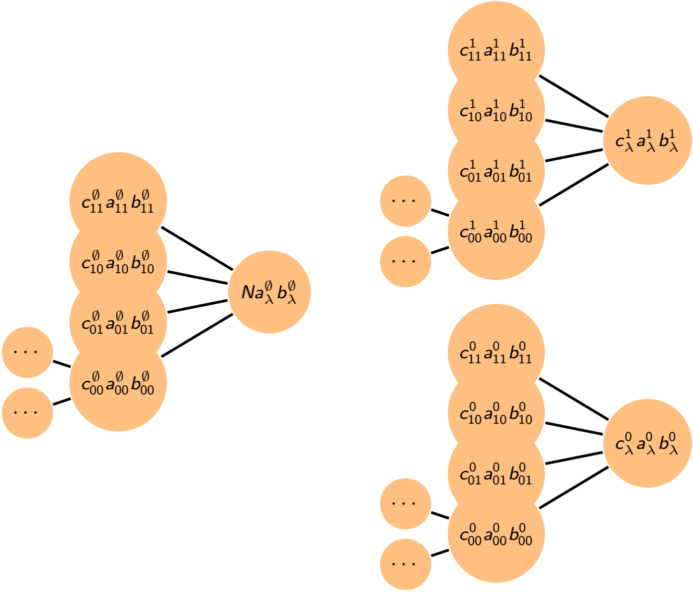
Binary Decomposition: Example (8)



Binary Decomposition: Example (9)



Binary Decomposition: Example (10)



Binary Decomposition: Coding *b*-Counts, Maximizing

- Coding the *b*-counts costs more bits now.
  It can be shown that we loose at most 6 bit per internal node (quaternary splitting can be accomplished with three binary splits).
  Fortunately this is at most 2 bit per parameter again.
- Maximizing formula exists again.

Conclusion

- The BW transform method does not need $\log_2(N)$ bits to specify a transition.
- The FSM closure is not needed. Specifying the maximizing tree model $\widehat{\mathcal{M}}$ and the b-counts in the leaves is enough.
- A loss comes from the fact that these b-counts are specified recursively. This loss is upper-bounded by 2 bit per leaf.
- Maximizing methods exist that match perfectly with the BW transform.
- There exist also binary decomposition techniques that combine with the BW transform.
- Also KT-coding of weights can be analysed.

Future Directions

- Software?
- Suppose that the data have left-right symmetry hence $P(a, b) = P(b, a)$, $P(a, b, c) = P(c, b, a)$, $P(a, b, c, d) = P(d, c, b, a)$, etc. This reduces the number of parameters. Algorithm? Important for image-compression.
- CTW can handle side-information by considering it as context (e.g. Cai, Kulkarni, and Verdu, 2005). BW-version?
- Can BW-techniques be used to achieve CT-weighting performance. Here BW-techniques are described that achieve CT-maximizing performance.
- What if the side-information is not-properly aligned? LZ is more robust now. Important for reference-based genome compression (Chern et al., 2012).
- Can also Lempel-Ziv be boosted? Context connection via Herschkovitz-Ziv (1998).