Fountain Codes

Gauri Joshi, Joong Bum Rhim, John Sun, Da Wang

Massachusetts Institute of Technology

December 8, 2010

Outline

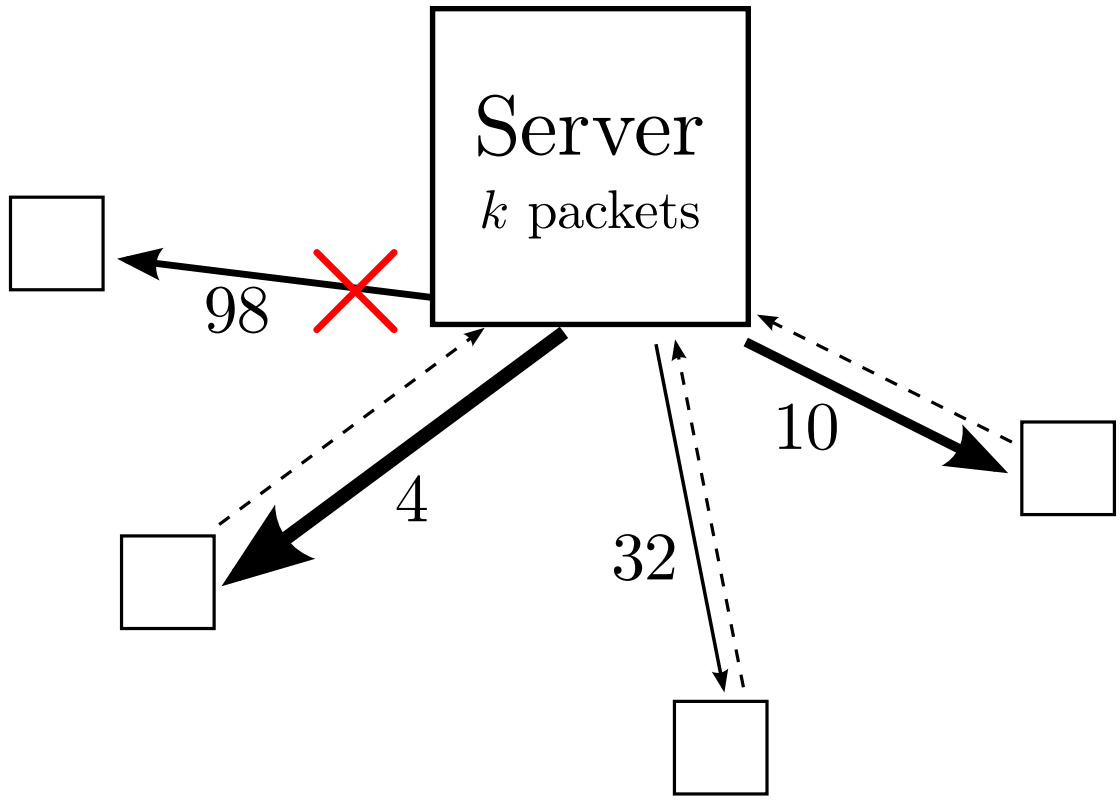
- Tornado Codes
- LT Codes
- Raptor Codes
- Extensions
- Summary

Outline

- Tornado Codes
- LT Codes
- Raptor Codes
- Extensions
- Summary

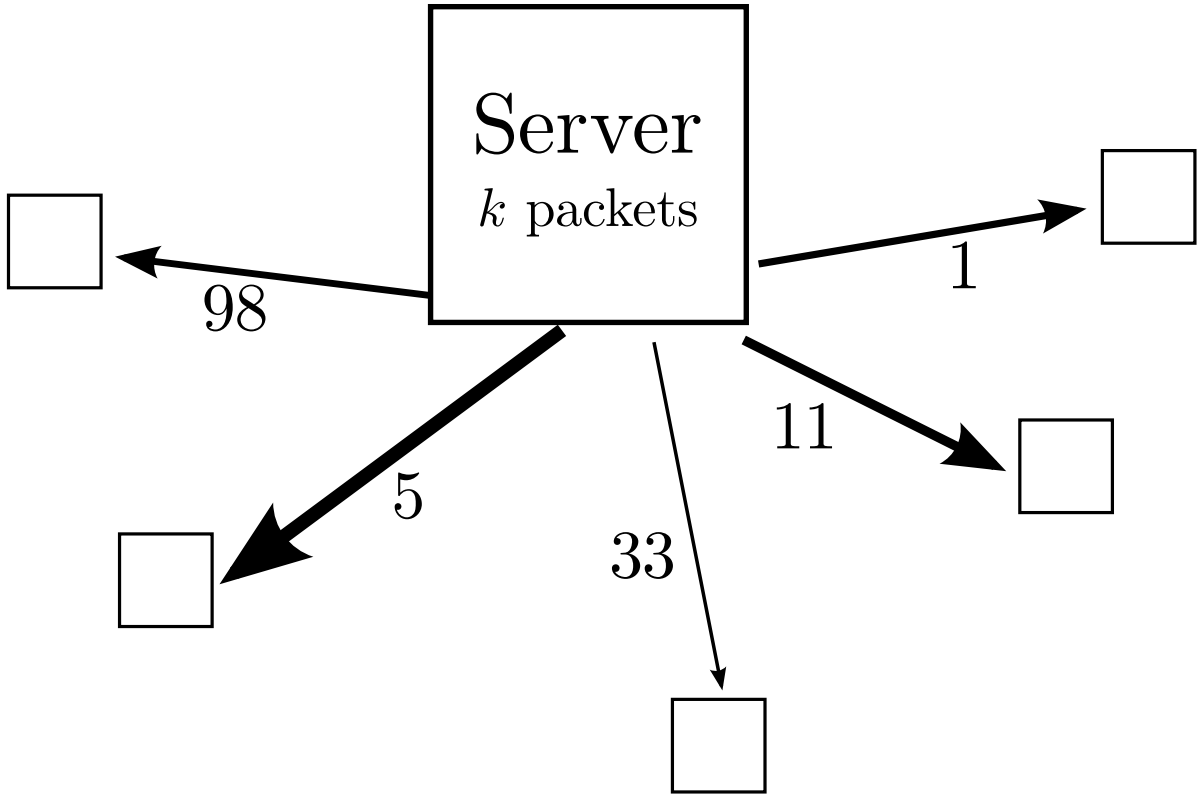
Internet Communication

- Internet is based on unicast protocols
- Can be very inefficient for dissemination of a large file



Internet Communication

- Internet is based on unicast protocols
- Can be very inefficient for dissemination of a large file



Digital Fountain Idea

- Server sends randomly generated droplets
- User collects them until a cup fills up
- Properties: Rateless, Universal.

[http://www.desktops-wallpapers.com/water.html]

Performance Measures

- Complexity:
  - Cost of encoding and decoding cost
  - Desire per-packet cost to be $O(1)$
- Overhead:
  - If number of encoding symbols necessary for decoding, then $\epsilon = (n-k)/k$
  - Optimal is 0 – no redundancy necessary
- Space:
  - Amount of storage necessary for encoding and decoding
  - Want per-packet space to be $O(1)$

Comparison across codes

	RS codes	LDPC	Tornado
Time Overhead	$O(n^2)$	$O(n^2)$	$O(n \ln(1/\epsilon))$
Reception Efficiency	1	$\frac{1}{1+\epsilon}$	$\frac{1}{1+\epsilon}$

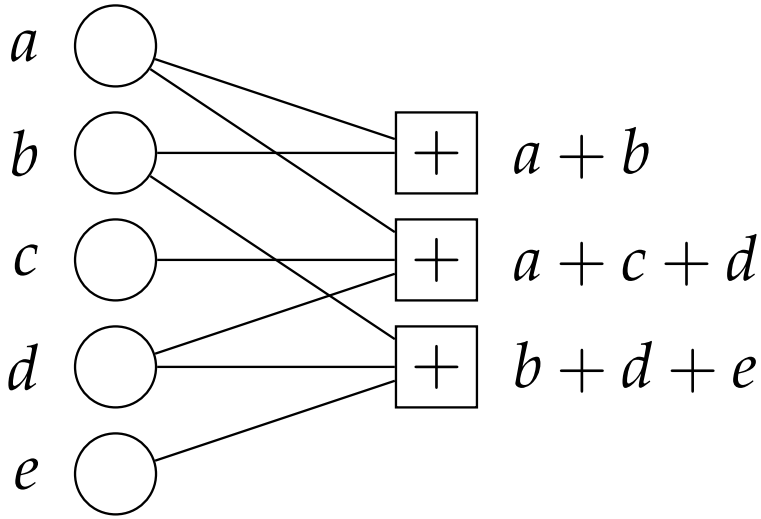
Tornado codes: Main contribution

- Unlike regular LDPC, use sparse irregular graphs.
- A simple linear time decoding algorithm
- Design and analysis of graphs for full recovery with this algorithm

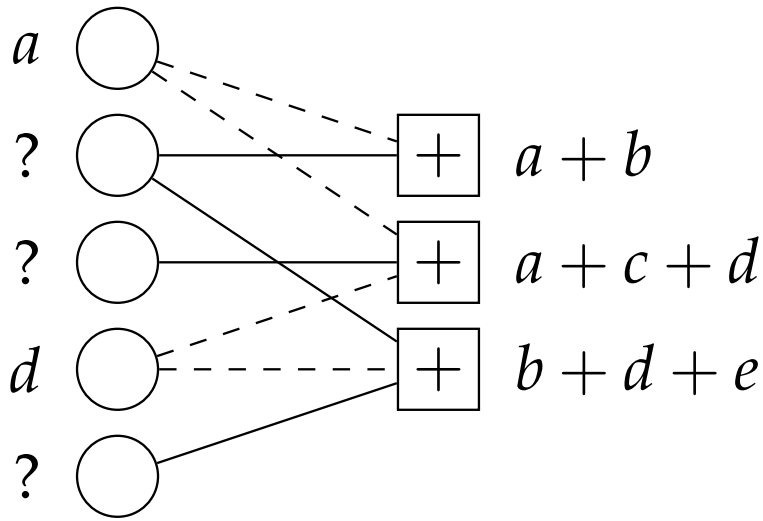
Erasure decoding [Luby (1997)]

"Given the value of a check bit and all but one of the message bits on which it depends, set the missing bit to be the XOR of the check bit and its known message bits"

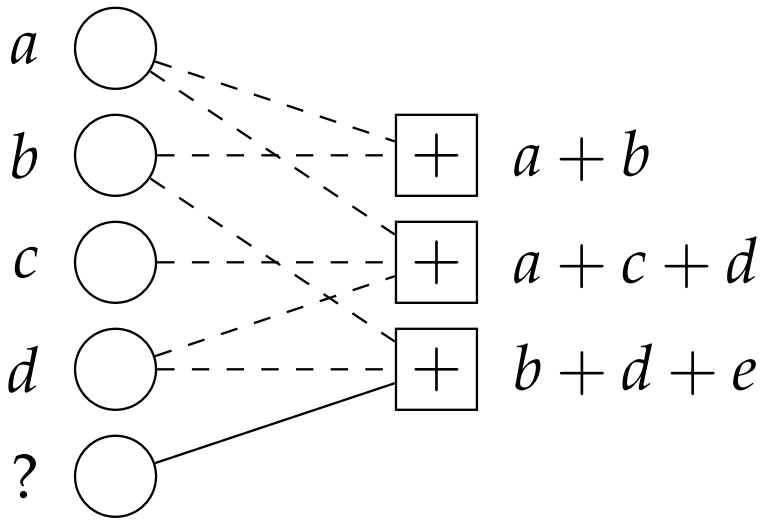
Decoding example



Decoding example

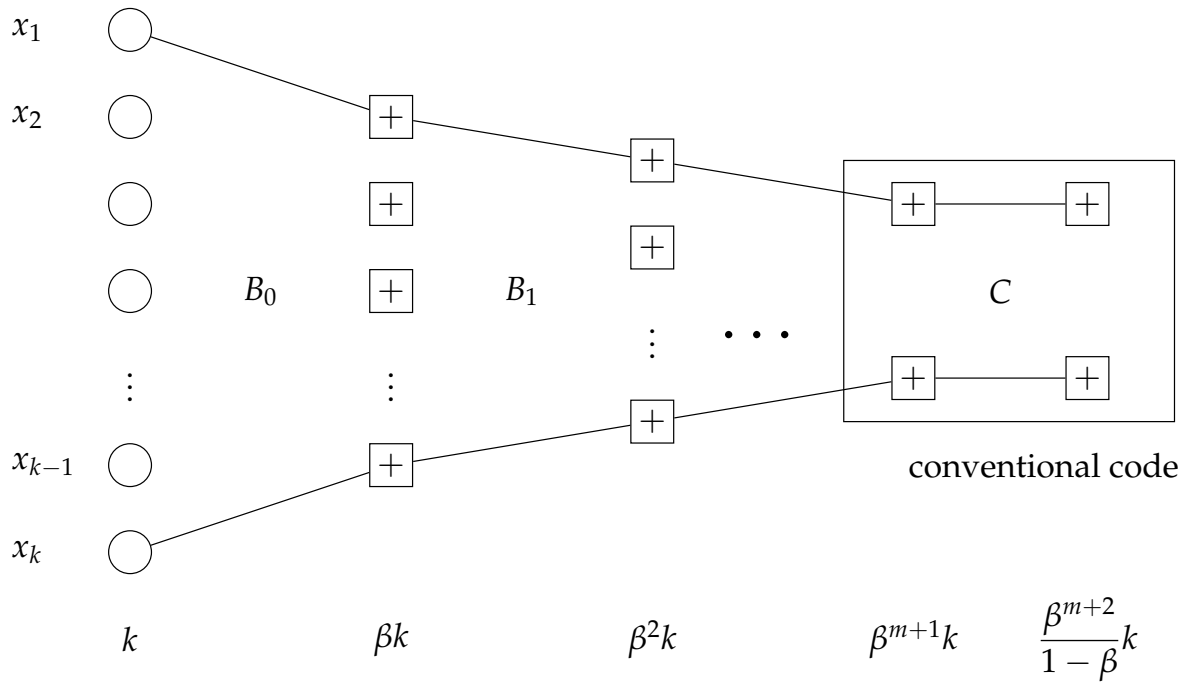


Decoding example



Tornado Codes: Graph Construction

- Cascade of $m+2$ bipartite graphs
- k information bits, $\beta^i k$ check bits at level i
- Last level - conventional rate $1-\beta$ code
- Overall rate $1-\beta$



Degree distributions

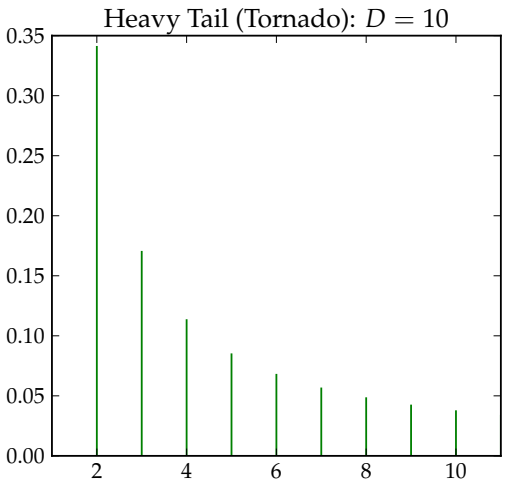
- Design Objective: There should be at least one right node with degree one at every step of decoding
- Condition: $\rho(1-\delta\lambda(x)) > 1-x \quad$ for all $x \in (0,1]$
- Optimal Distributions:

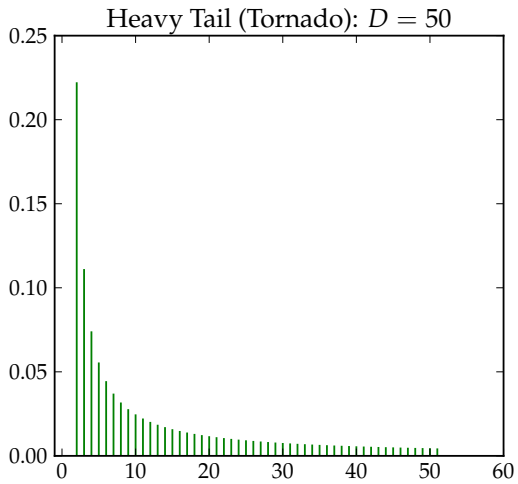
$$\lambda_i = \frac{1}{H(D)(i-1)} \quad \text{for all } i = 2,3,..,D+1 \tag{1}$$

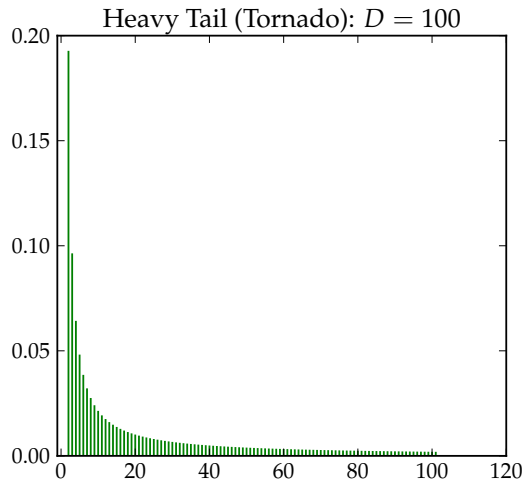
$$\rho_i = \frac{e^{-\alpha}\alpha^{i-1}}{(i-1)!}, \tag{2}$$

where, $H(D) = \sum_{i=1}^{D} \frac{1}{i} \approx ln(D)$, $d_\lambda = \beta d_\rho$.

Truncated Heavy Tail distribution







Comparison of degrees with LDPC codes

LDPC codes

- Right degree $d_\rho = 2d$, $Pr(\text{Right degree} = 1) = 1/2^{2d-1}$
- Left node has average $d/2^{2d-1}$ neighbors of degree one

Tornado Codes

- Avg. right degree $2\ln(D)$,
  $Pr(\text{Right degree} = 1) \approx 1/(D+1)$
- Left node has on an average 1 neighbors of degree one

Outline

- Tornado Codes
- LT Codes
- Raptor Codes
- Extensions
- Summary

Introduction

- The first practical rateless codes [Luby (2002)]

- The original data of k information symbols can be recovered
  - from $k + O(\sqrt{k}\ln^2(k/\delta)$ encoding symbols
    (overhead: $O(\ln^2(k/\delta)/\sqrt{k})$)
  - with probability $1 - \delta$
  - by on average $O(\ln(k/\delta))$ symbol operations per symbol.

- Low-density generator-matrix (LDGM) codes

Encoding

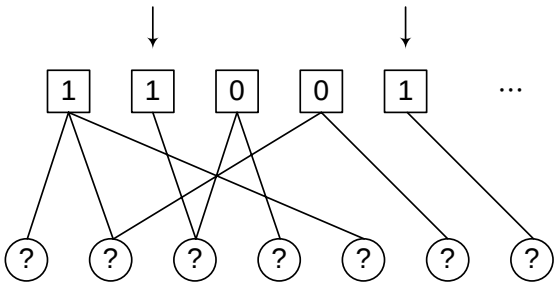
1) Determine the degree d of an encoding symbol from a given degree distribution $\rho(d)$.

2) Choose d distinct information symbols uniformly at random.

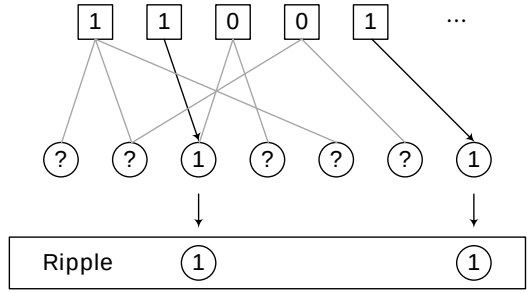
3) Assign the exclusive-or of the chosen d information symbols to the encoding symbol.

Decoding (LT Process)

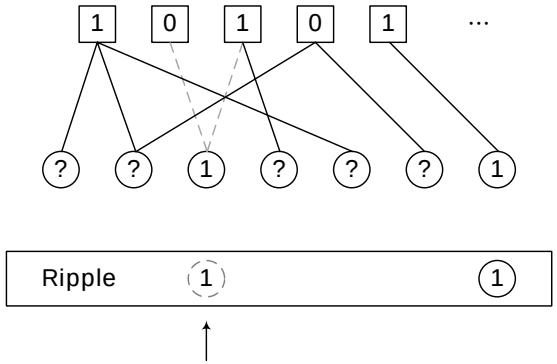
1) Release



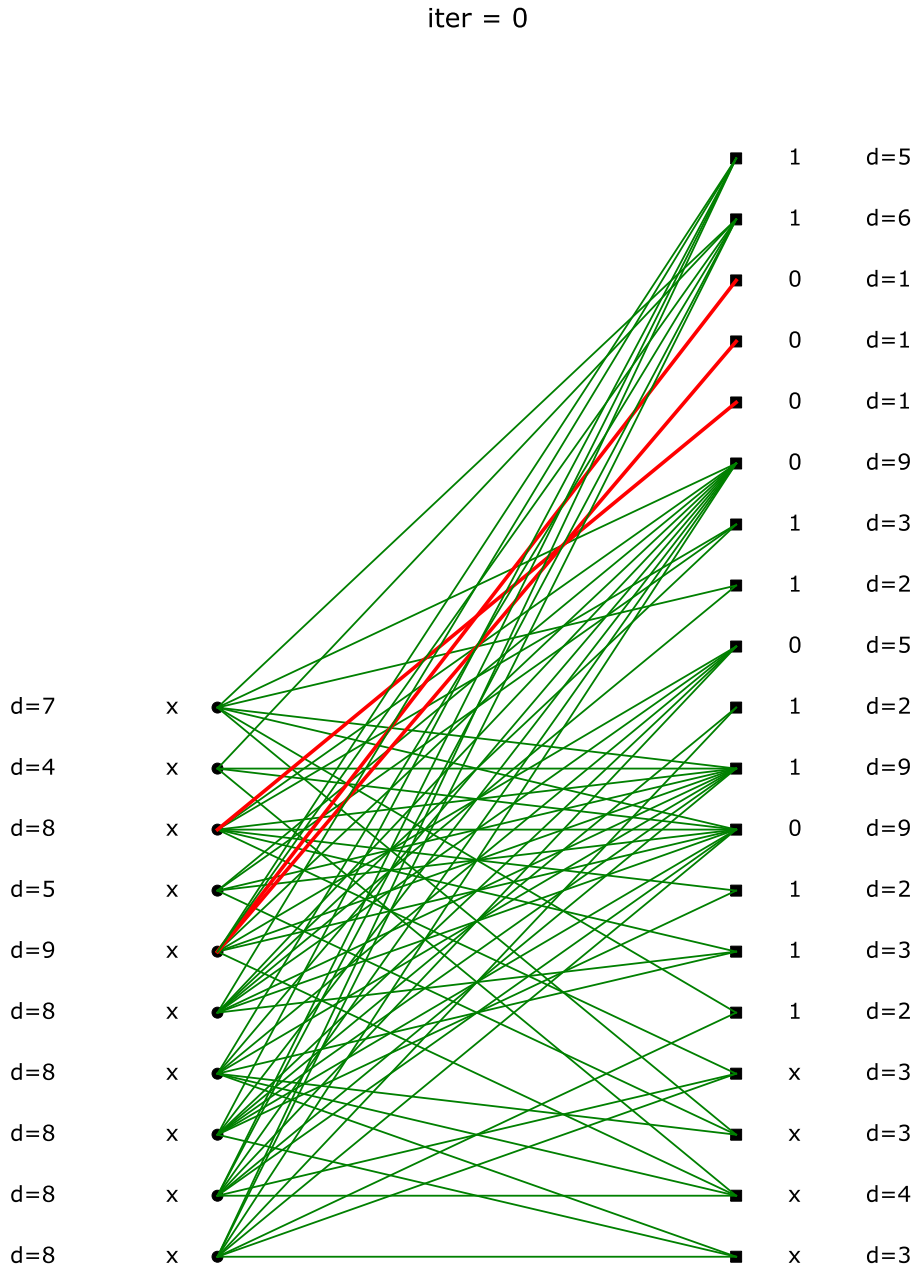
2) Cover



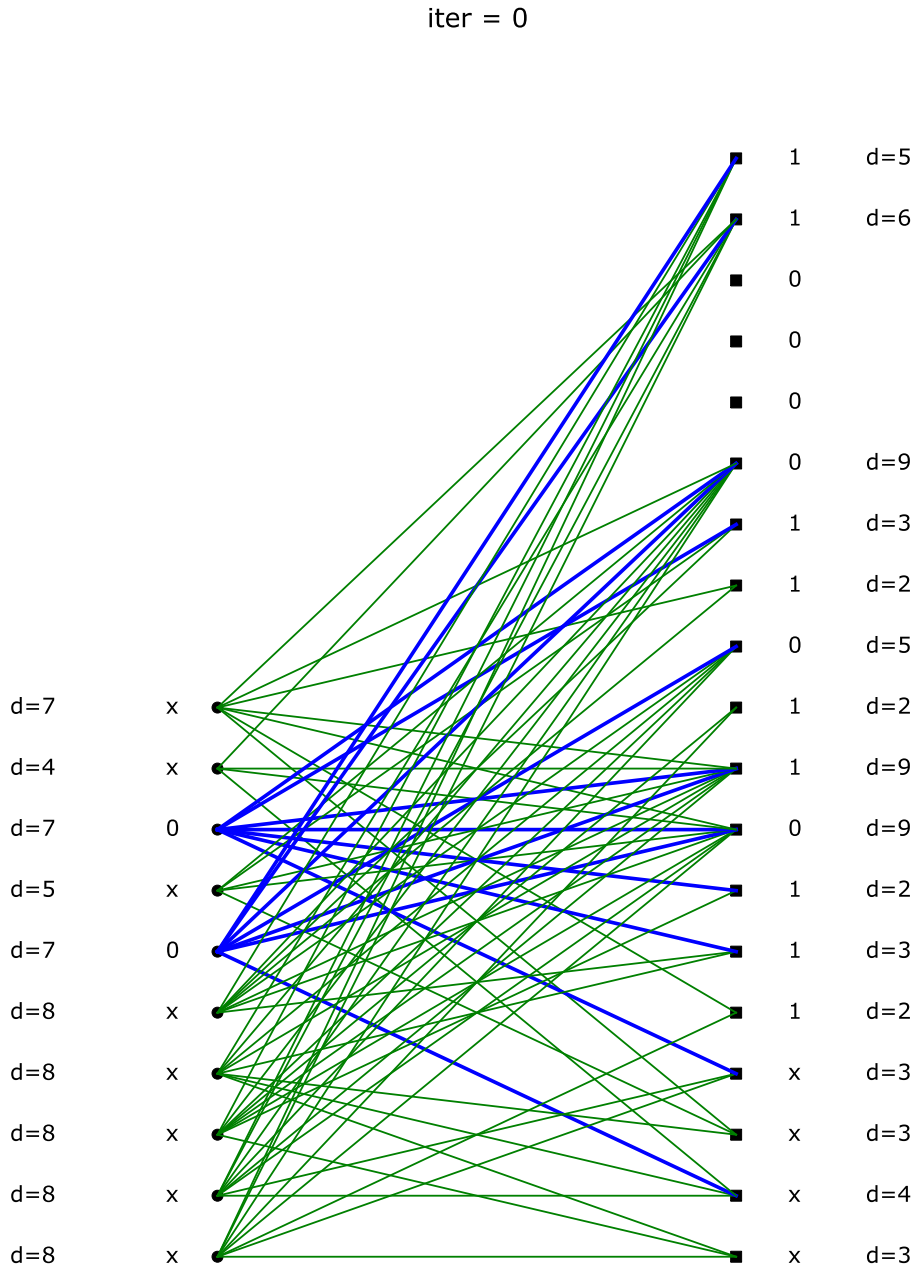
3) Process



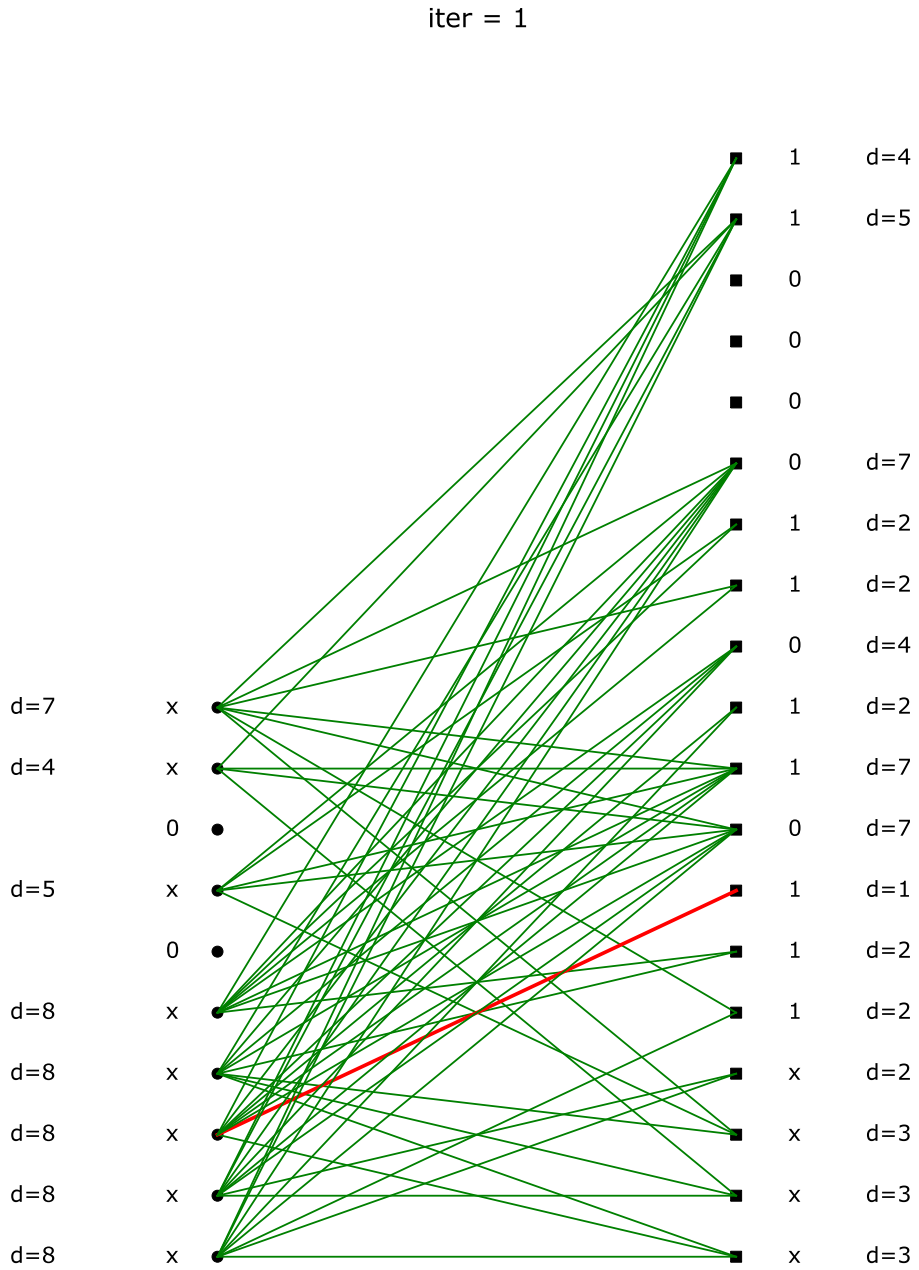
LT Decoding example



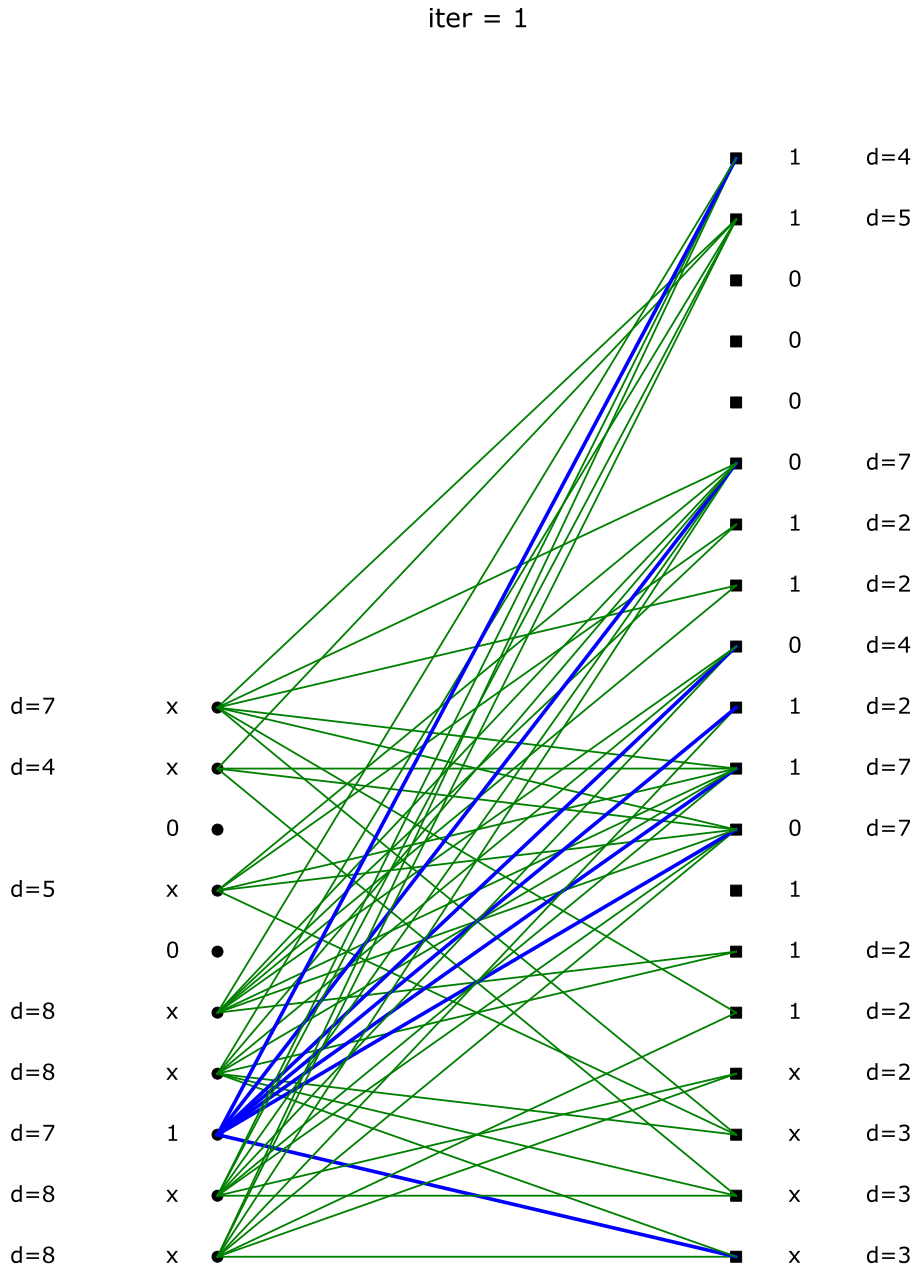
LT Decoding example



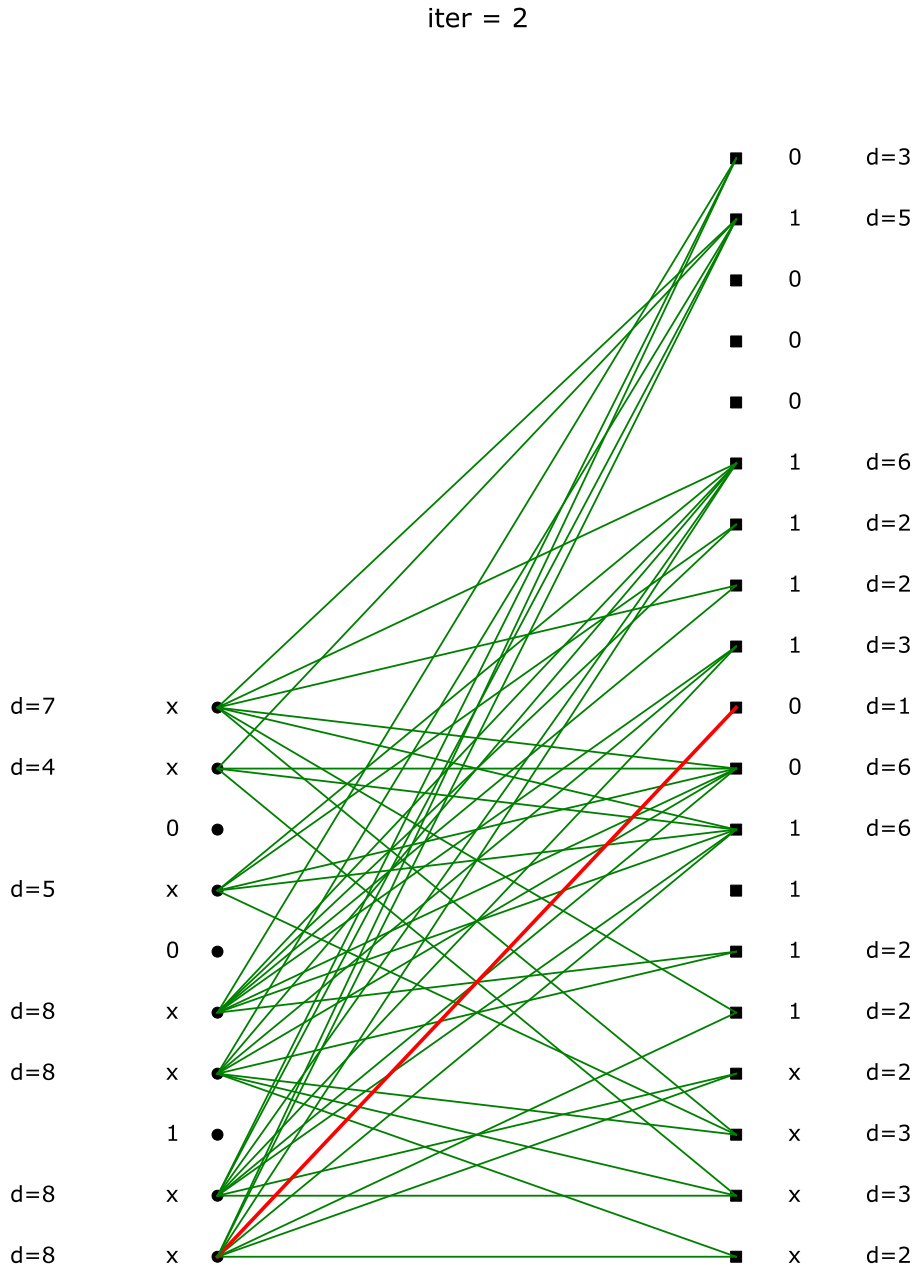
LT Decoding example



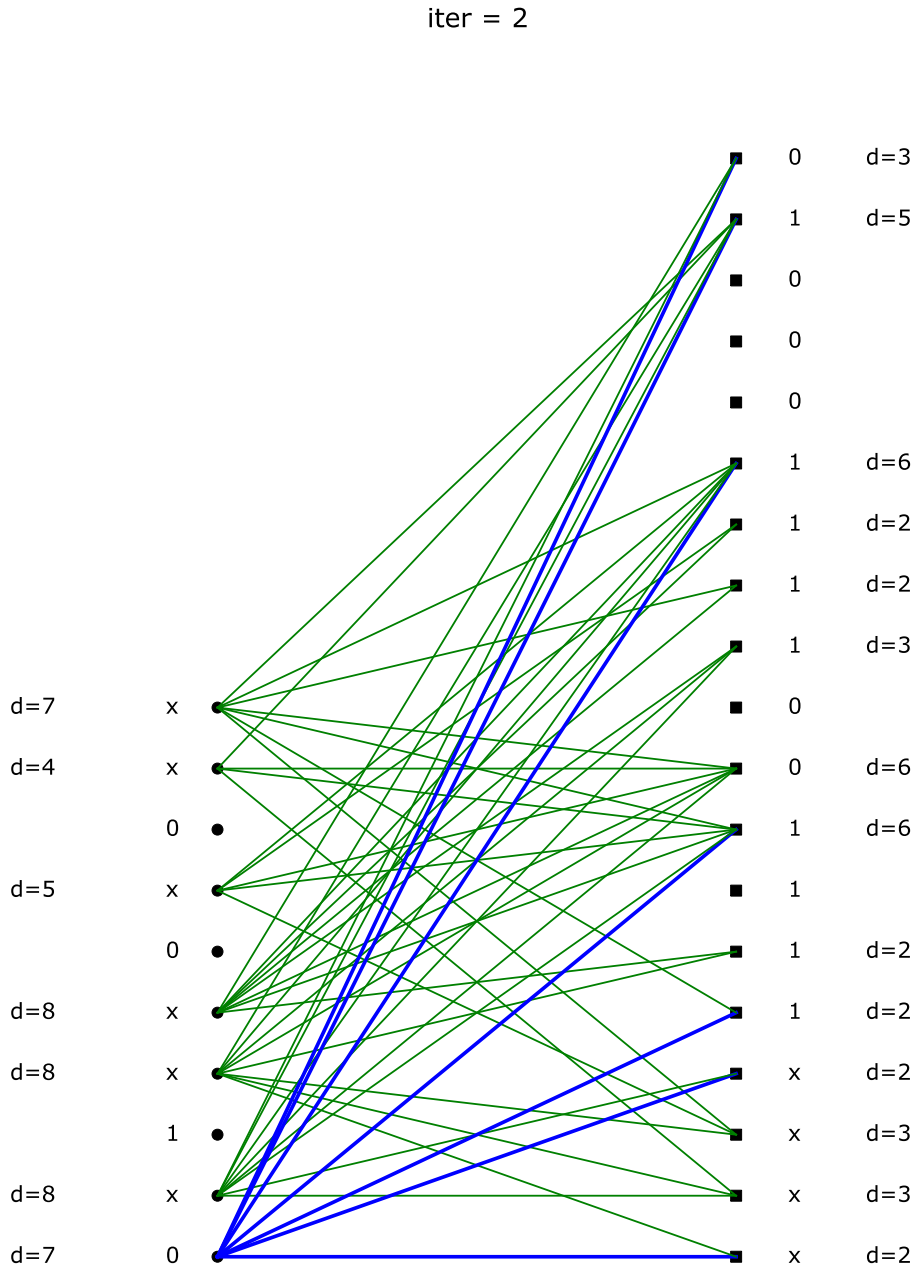
LT Decoding example



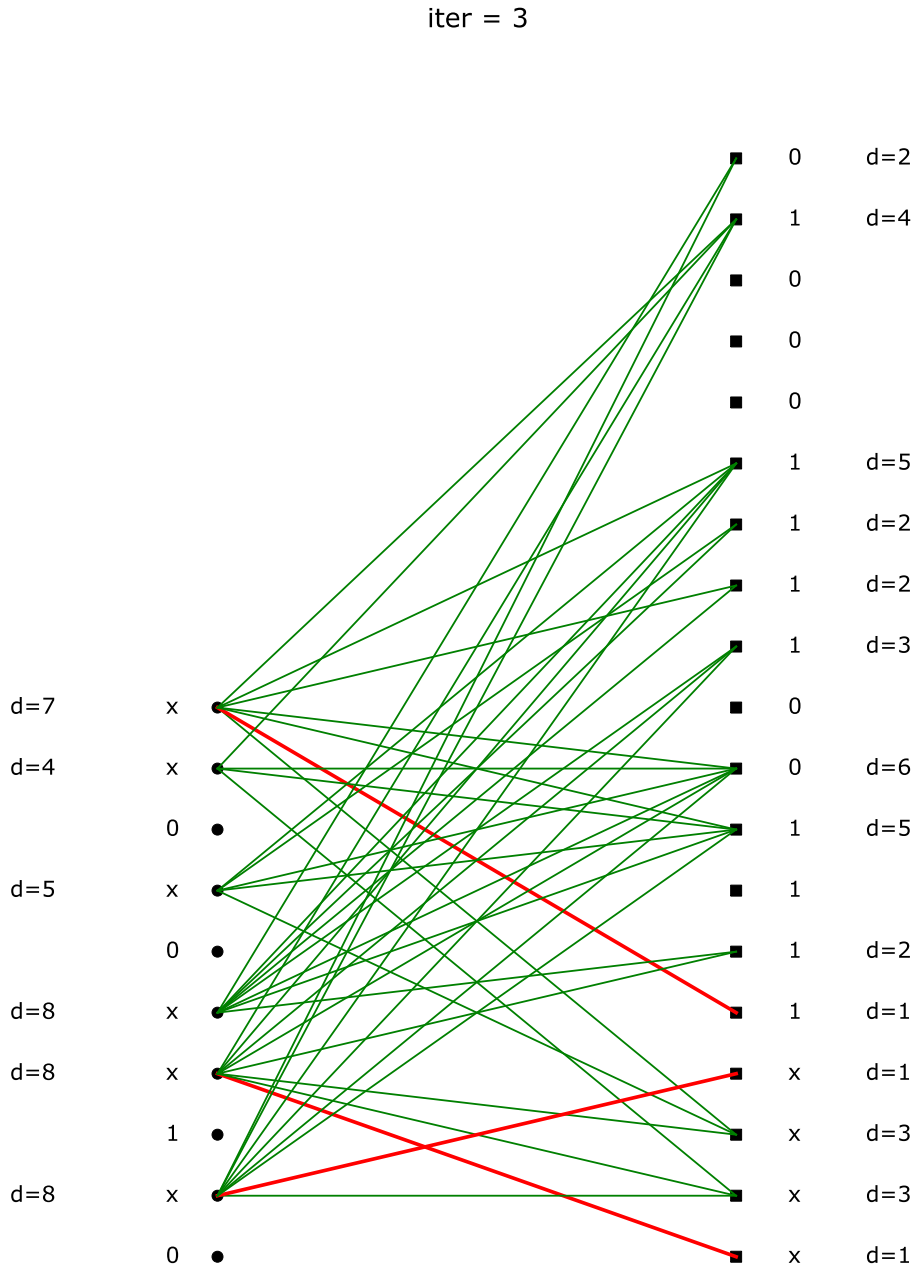
LT Decoding example



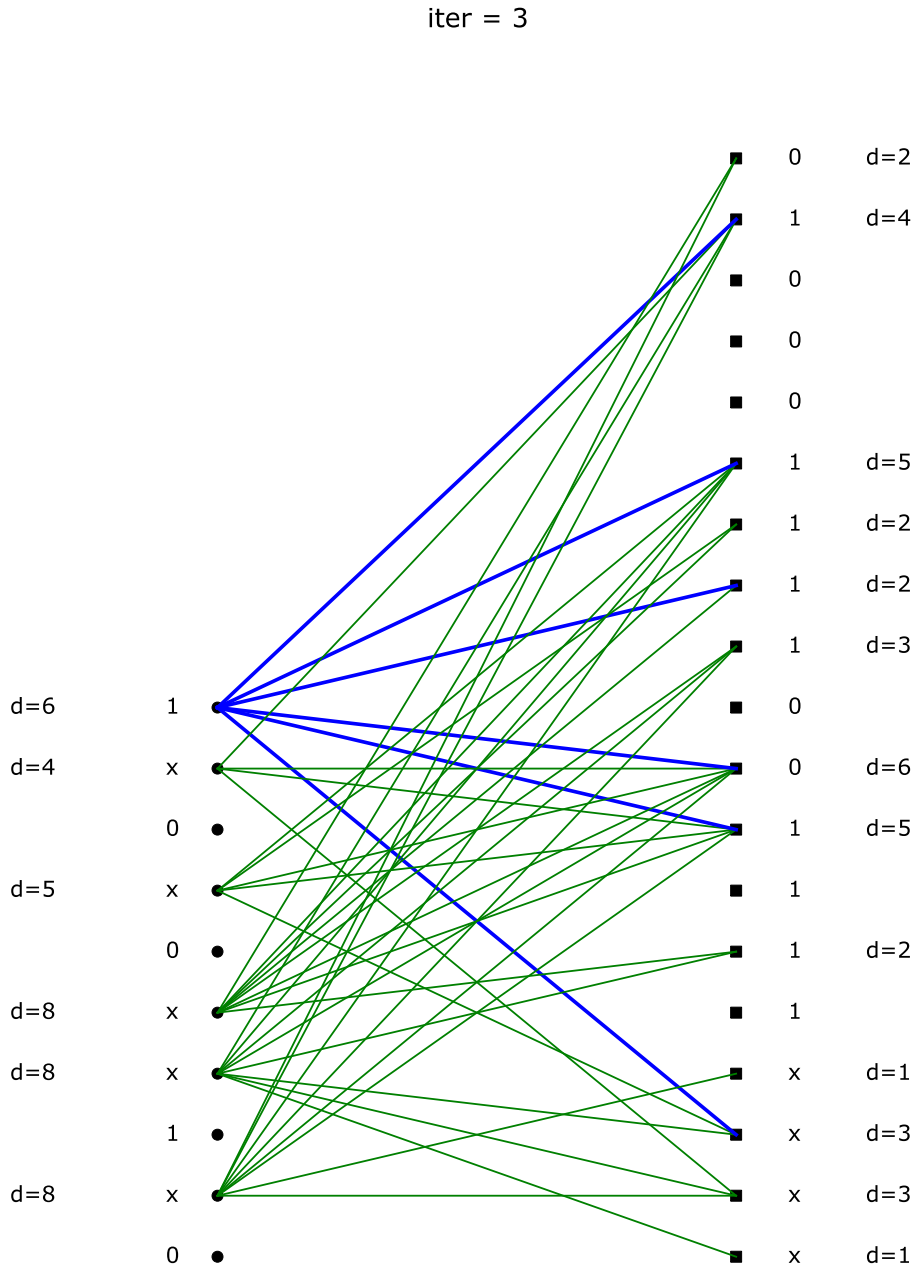
LT Decoding example



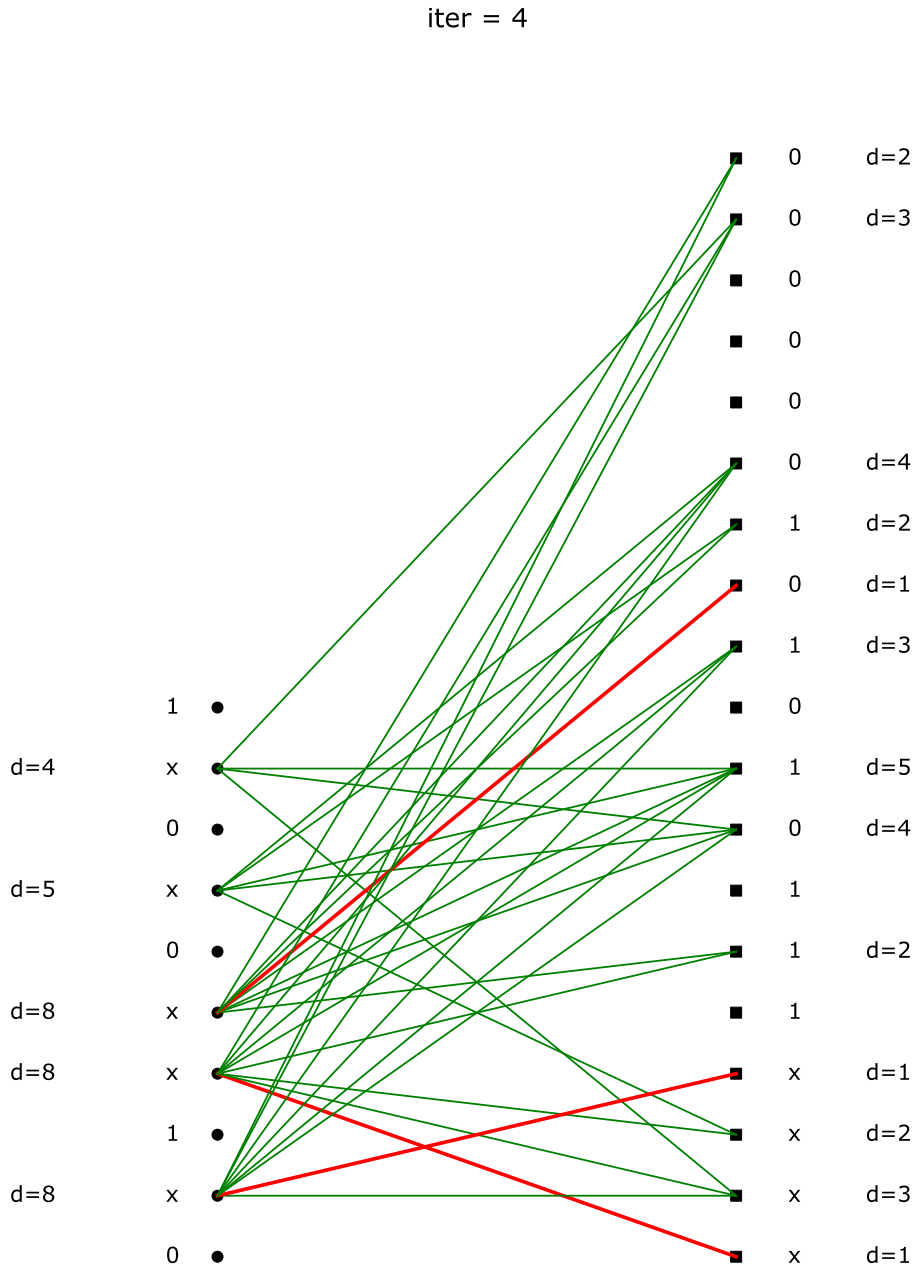
LT Decoding example



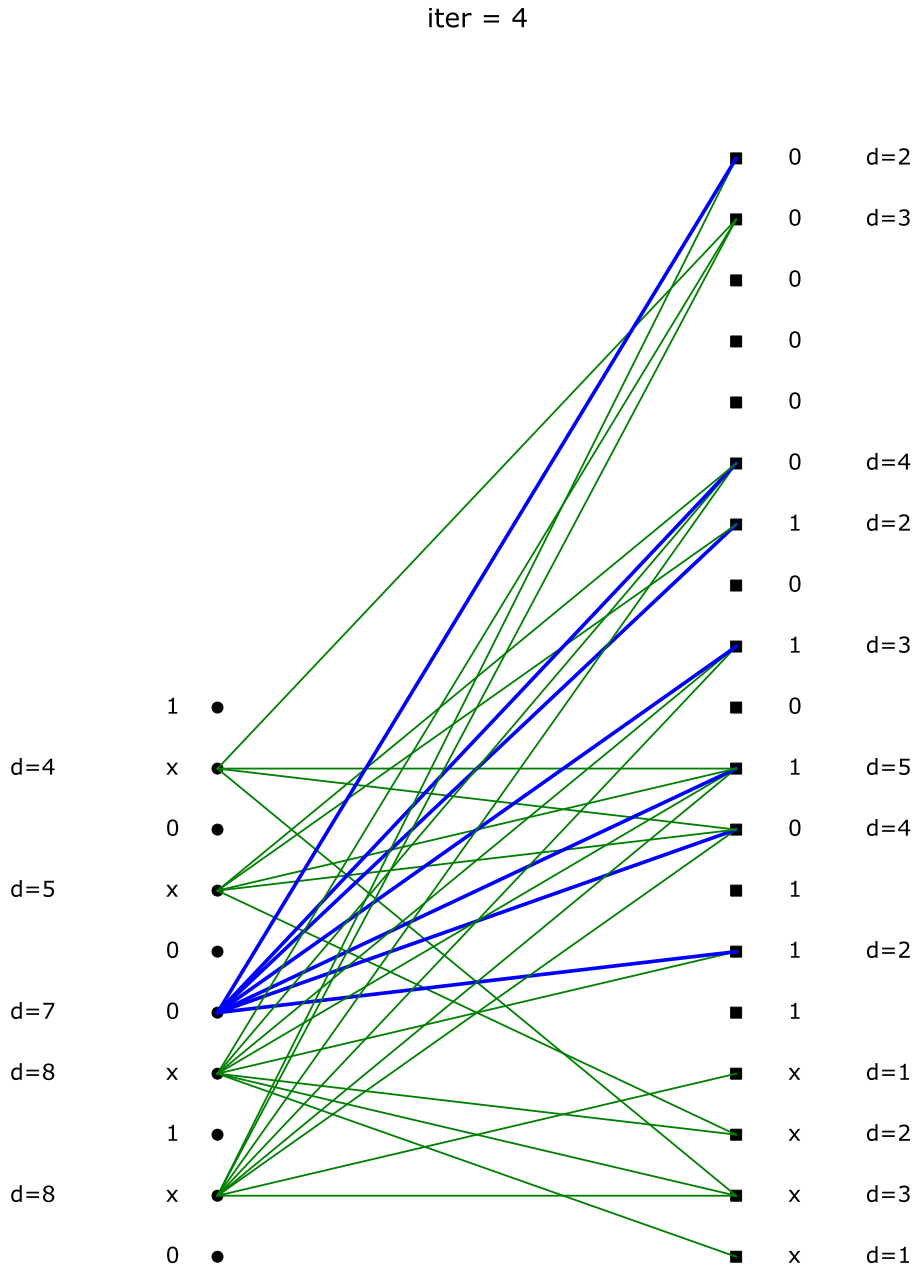
LT Decoding example



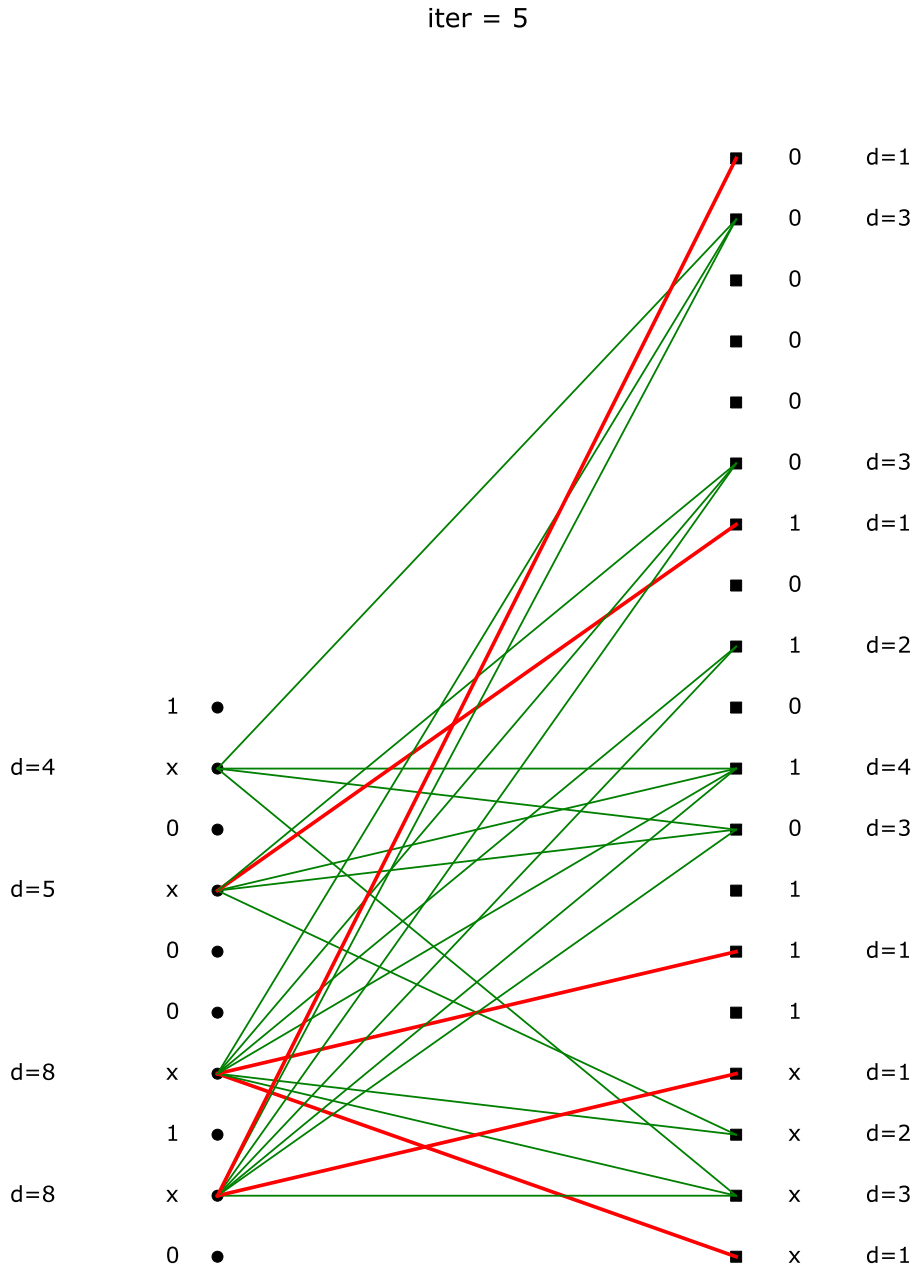
LT Decoding example



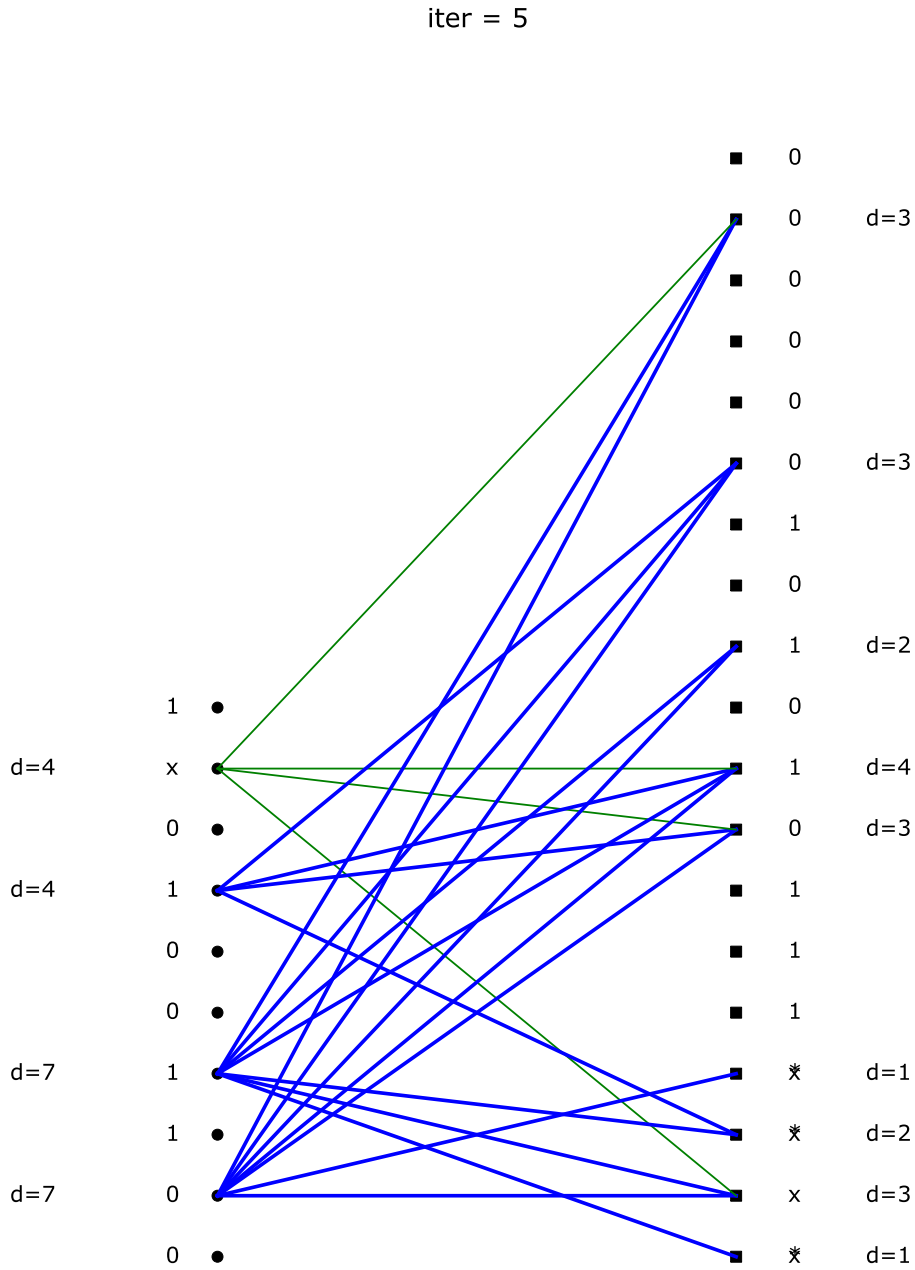
LT Decoding example



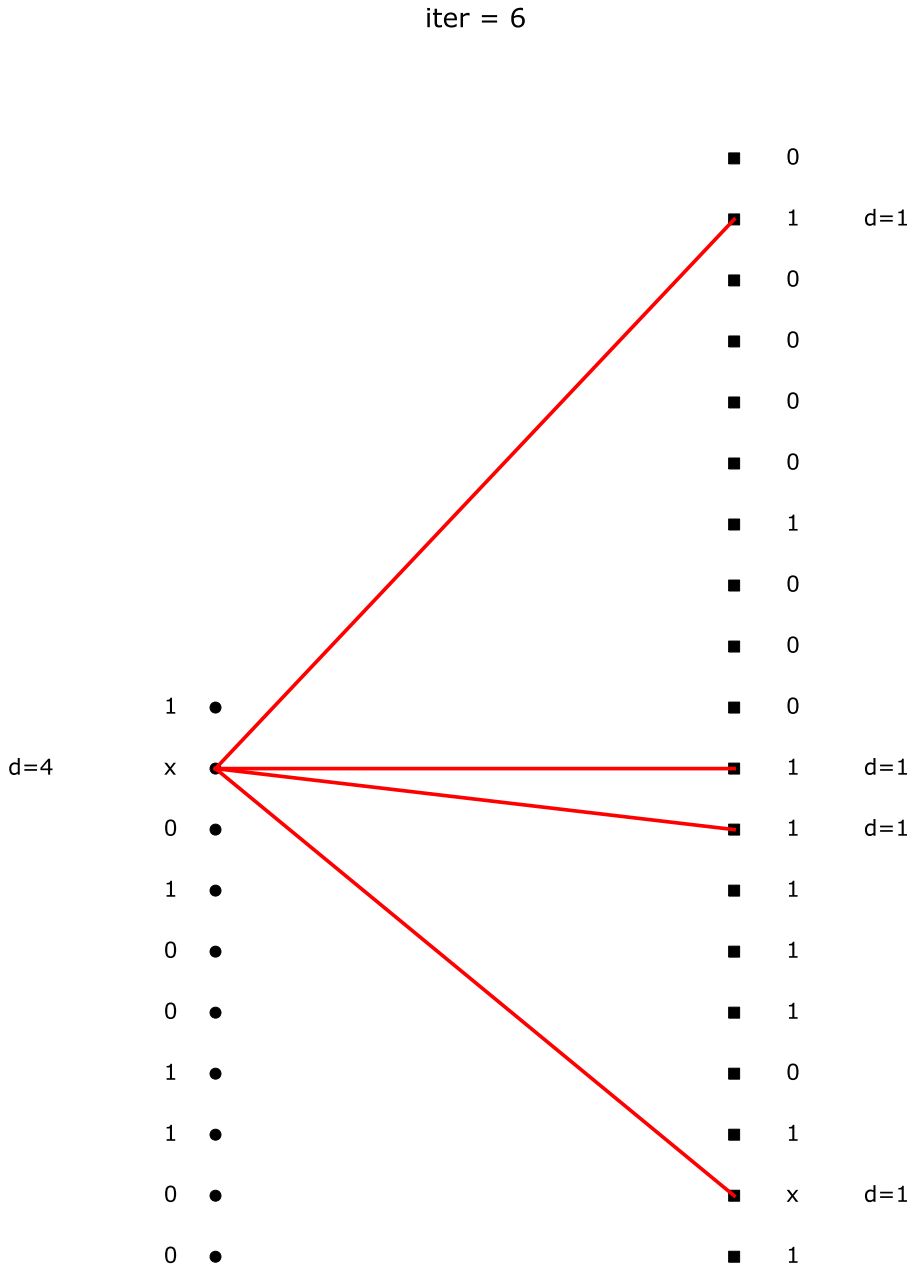
LT Decoding example



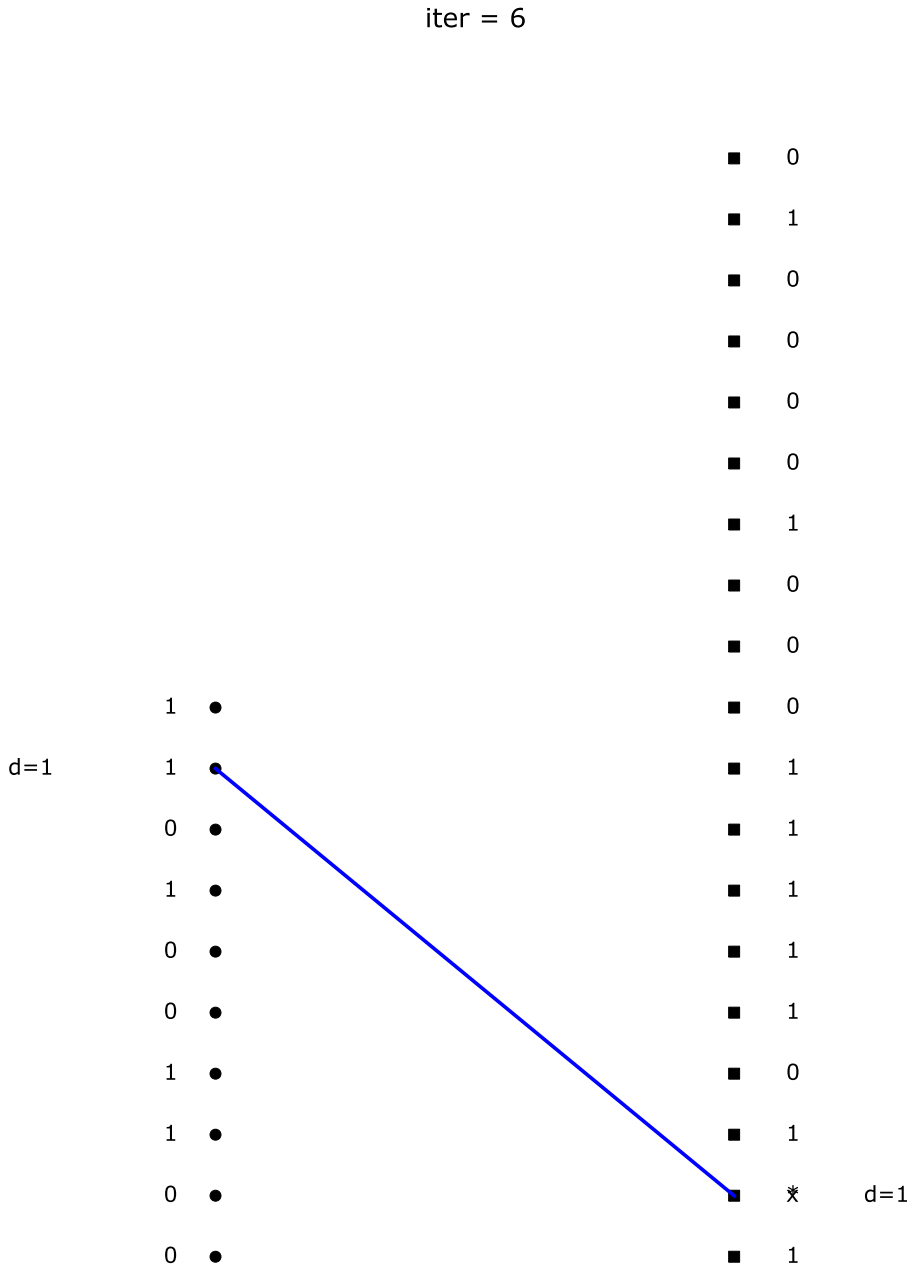
LT Decoding example



LT Decoding example

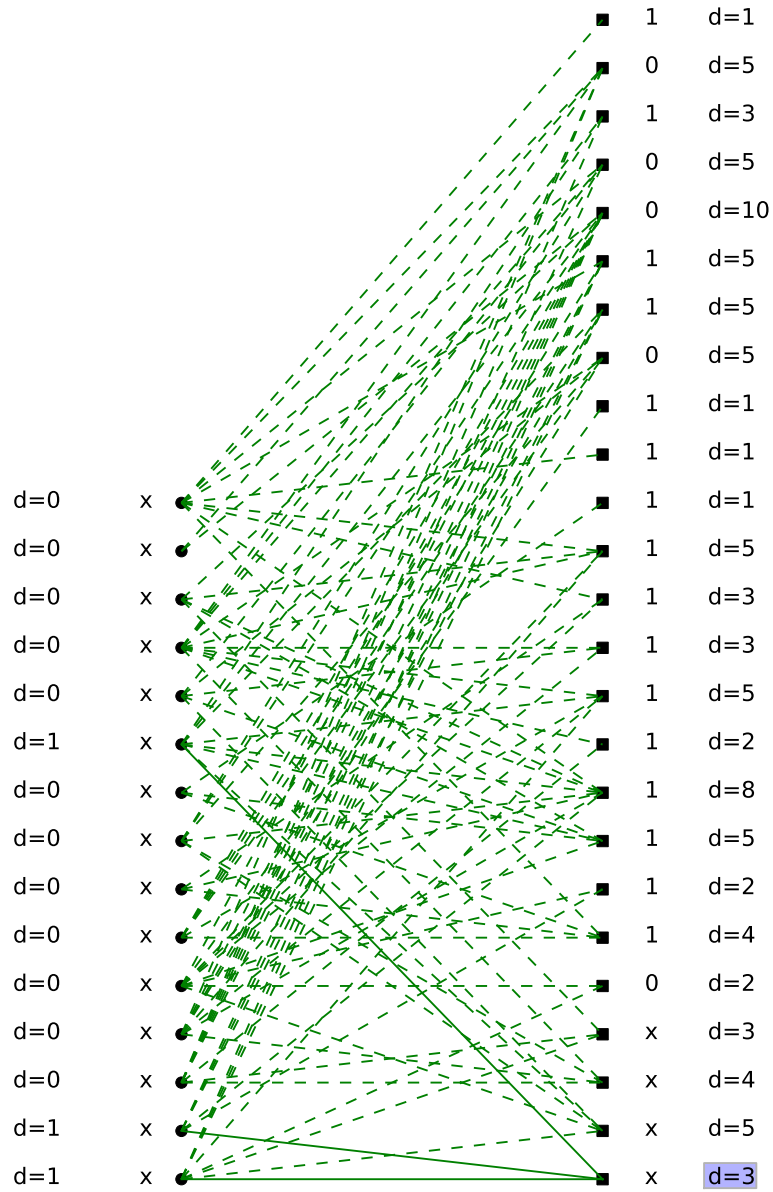


LT Decoding example



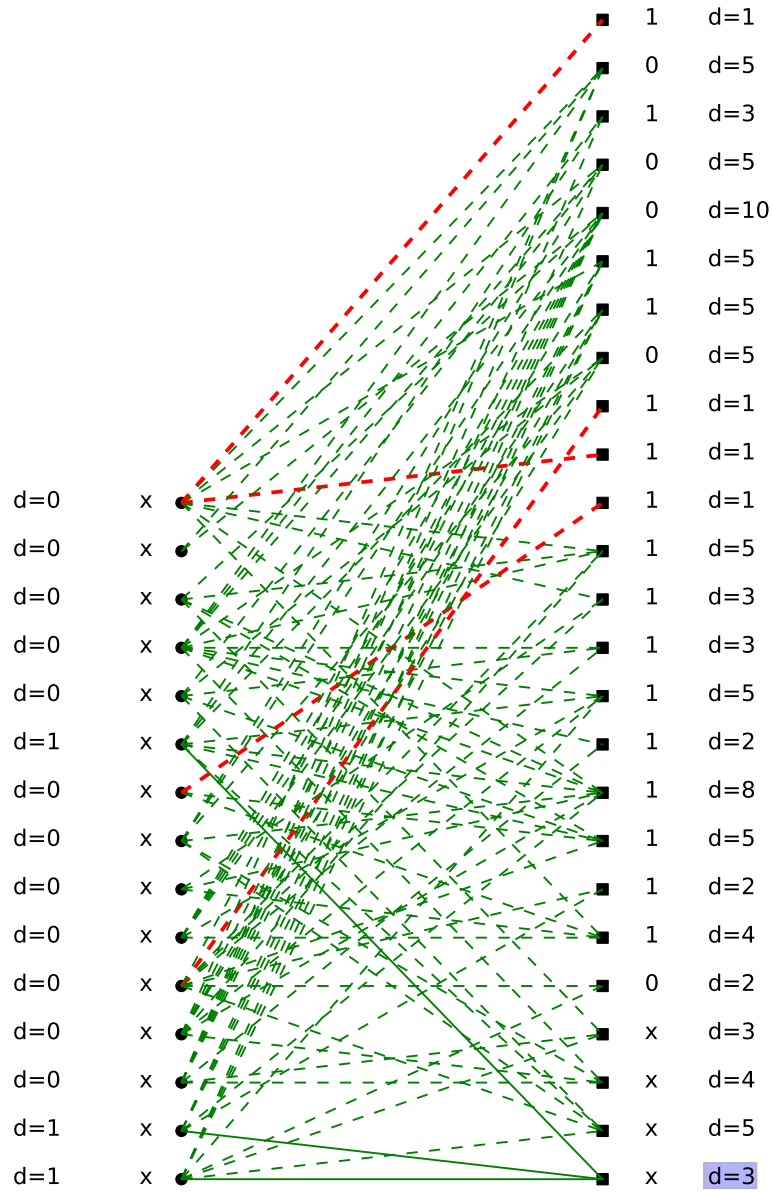
Progressive LT Decoding example

- Decode as we received symbols from the channel



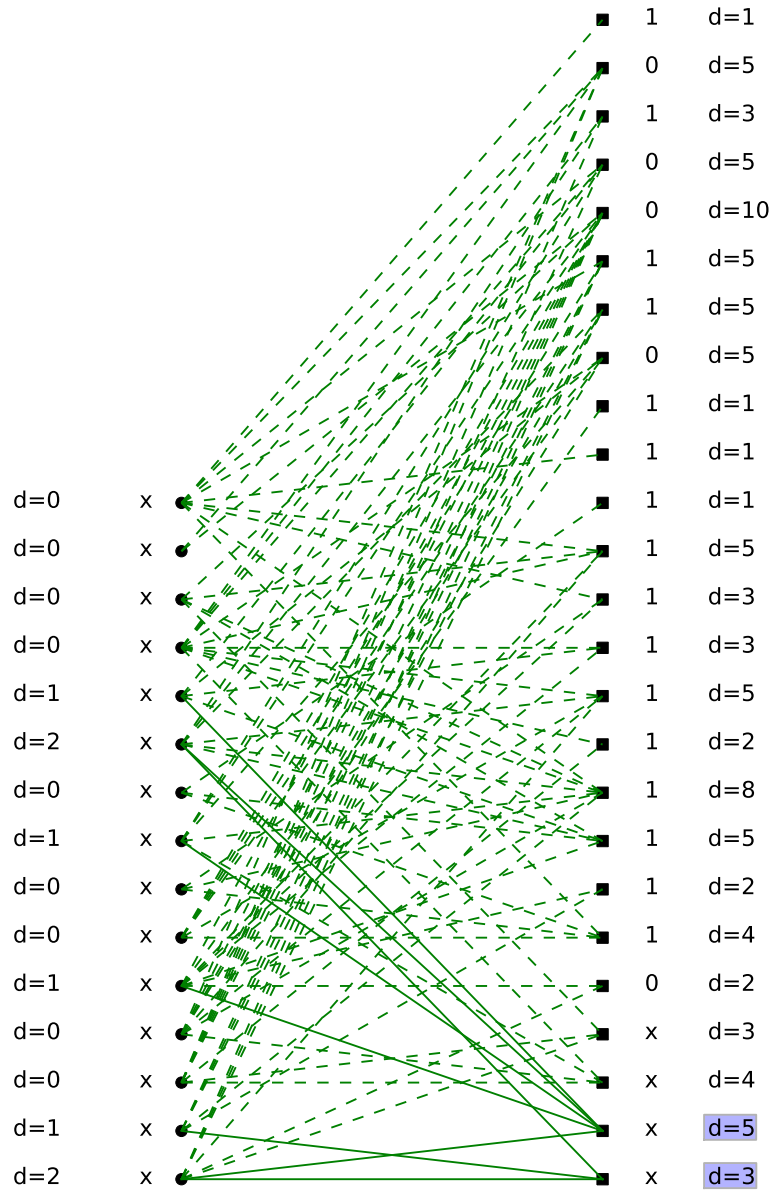
Progressive LT Decoding example

- Decode as we received symbols from the channel



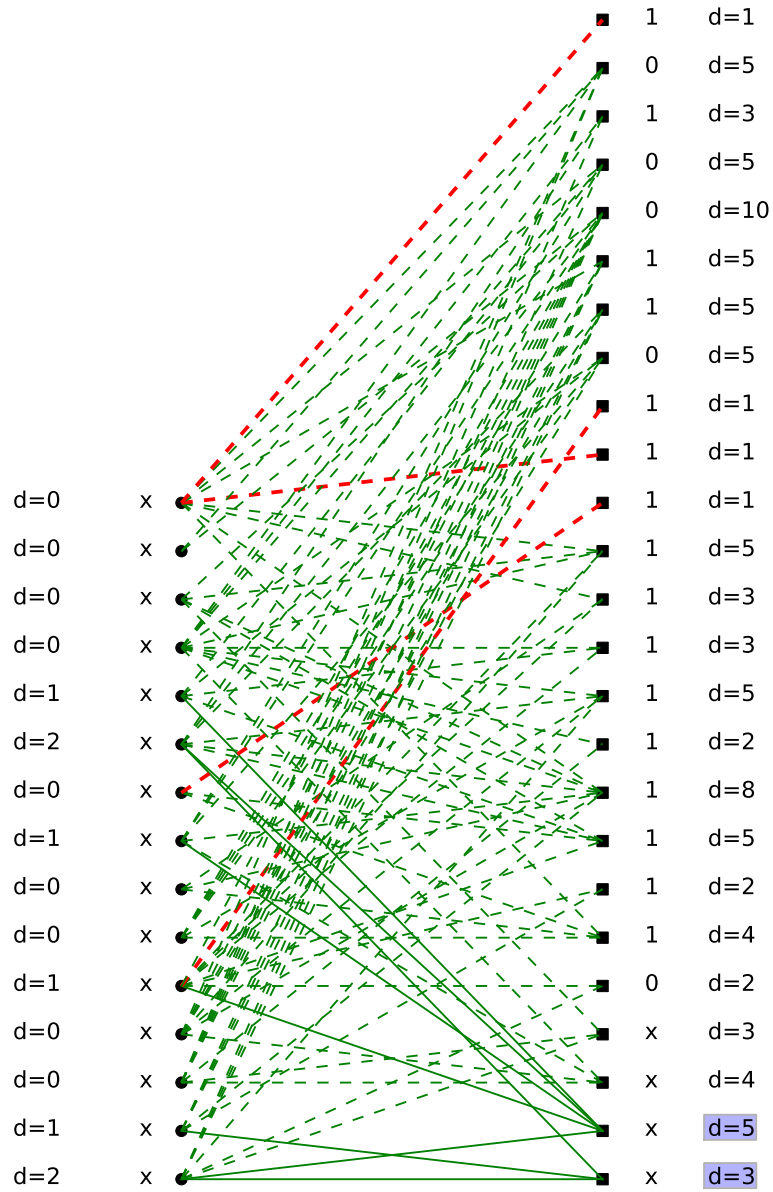
Progressive LT Decoding example

- Decode as we received symbols from the channel



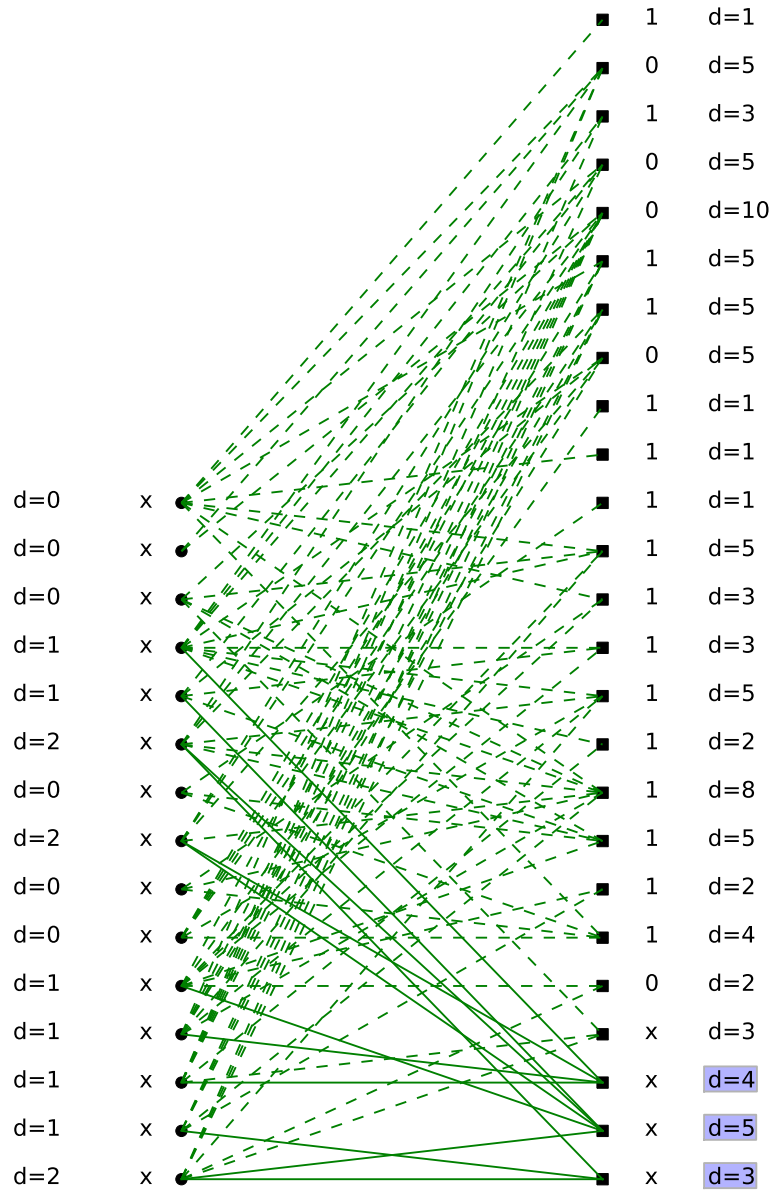
Progressive LT Decoding example

- Decode as we received symbols from the channel



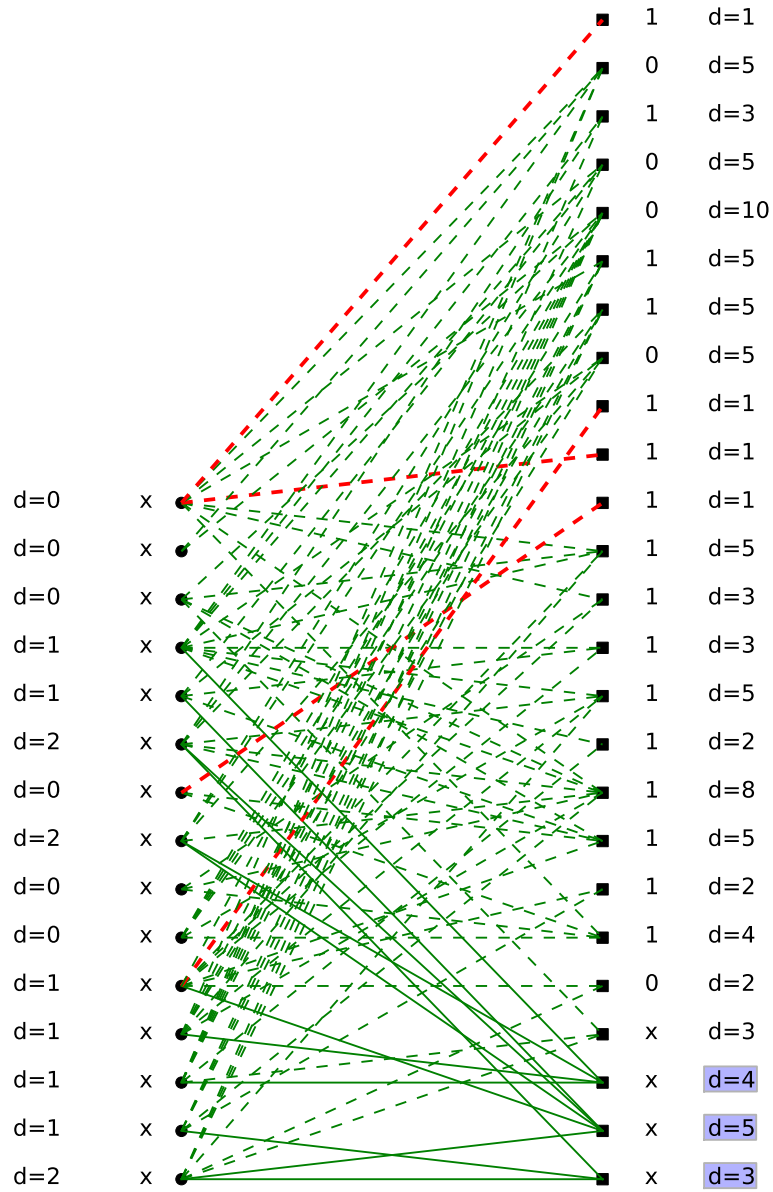
Progressive LT Decoding example

- Decode as we received symbols from the channel



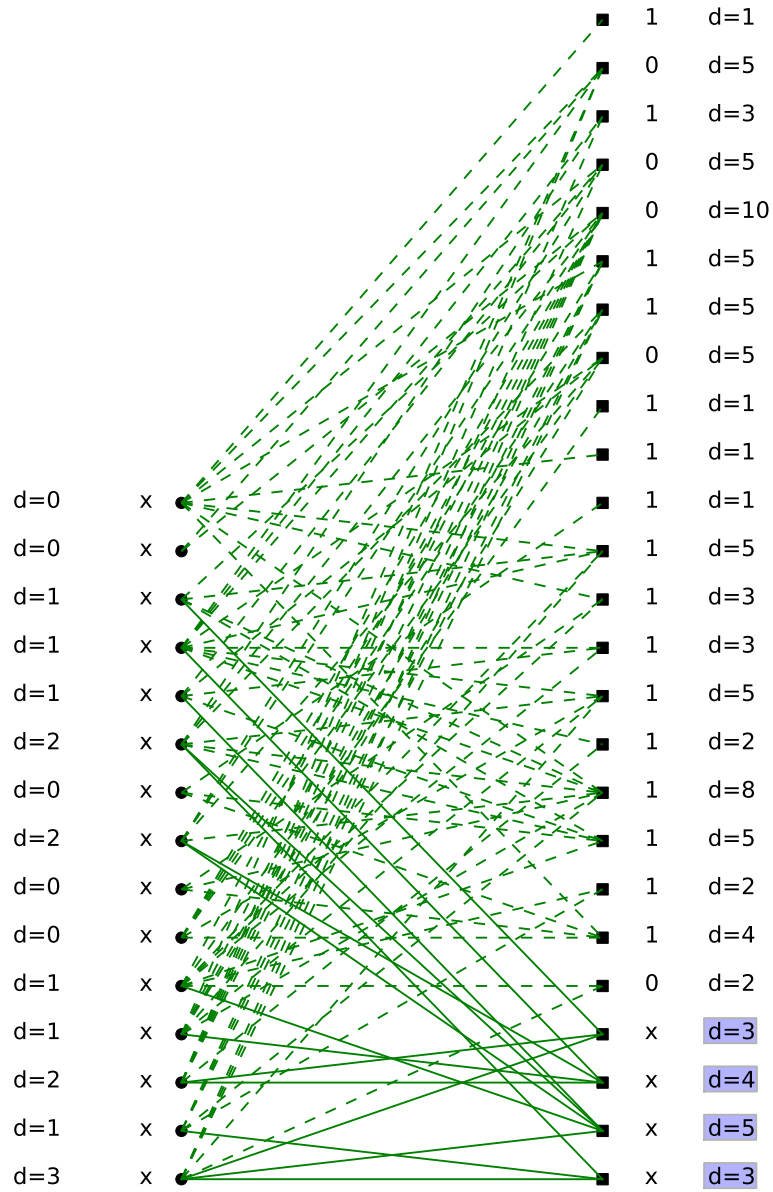
Progressive LT Decoding example

- Decode as we received symbols from the channel



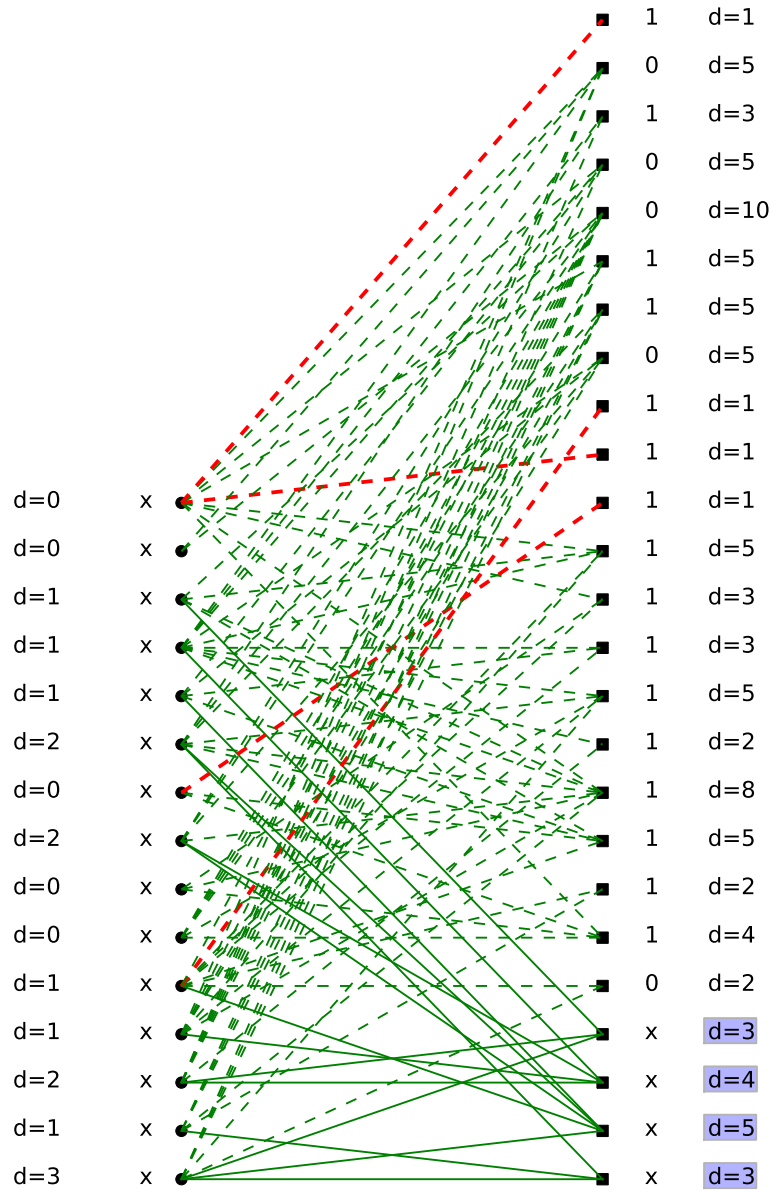
Progressive LT Decoding example

- Decode as we received symbols from the channel



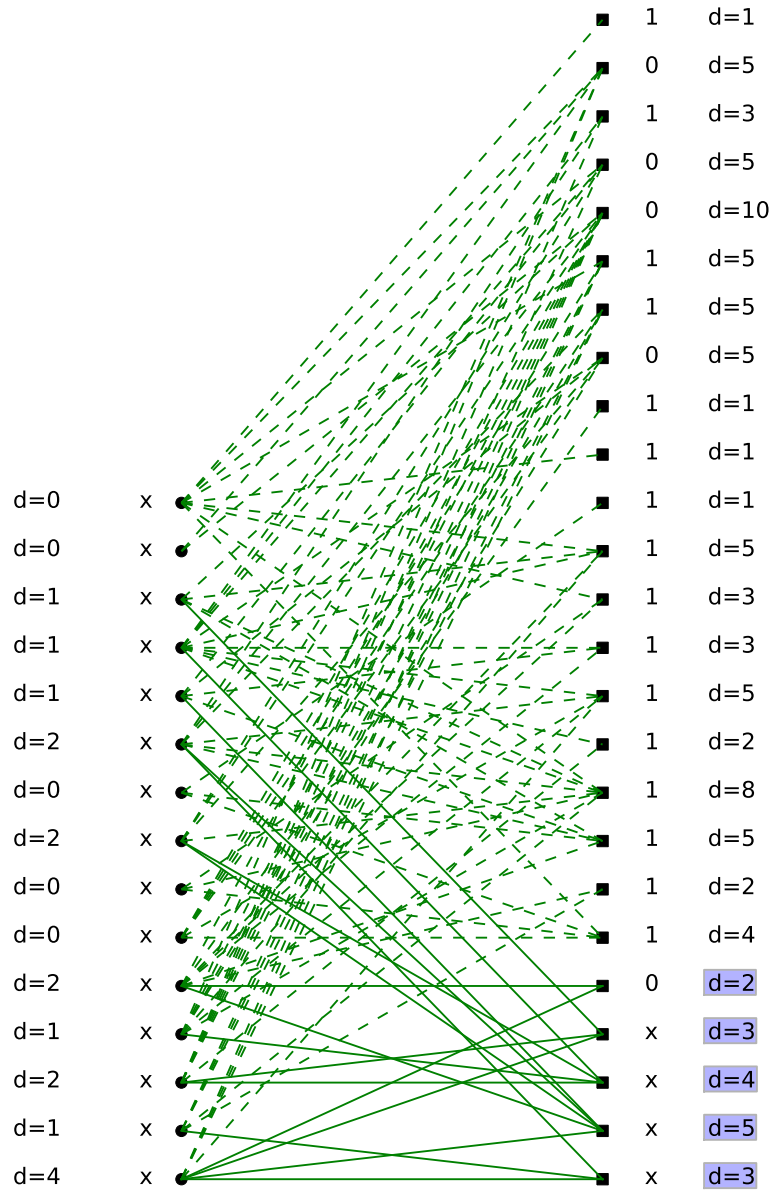
Progressive LT Decoding example

- Decode as we received symbols from the channel



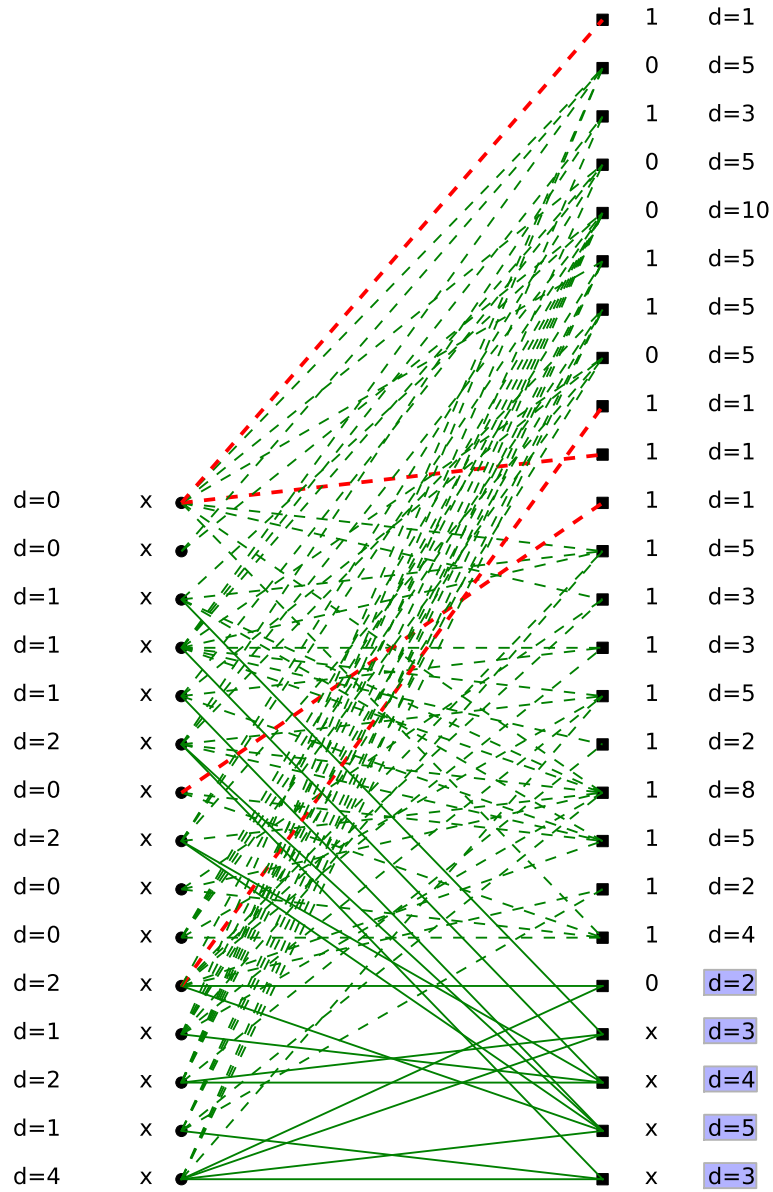
Progressive LT Decoding example

- Decode as we received symbols from the channel



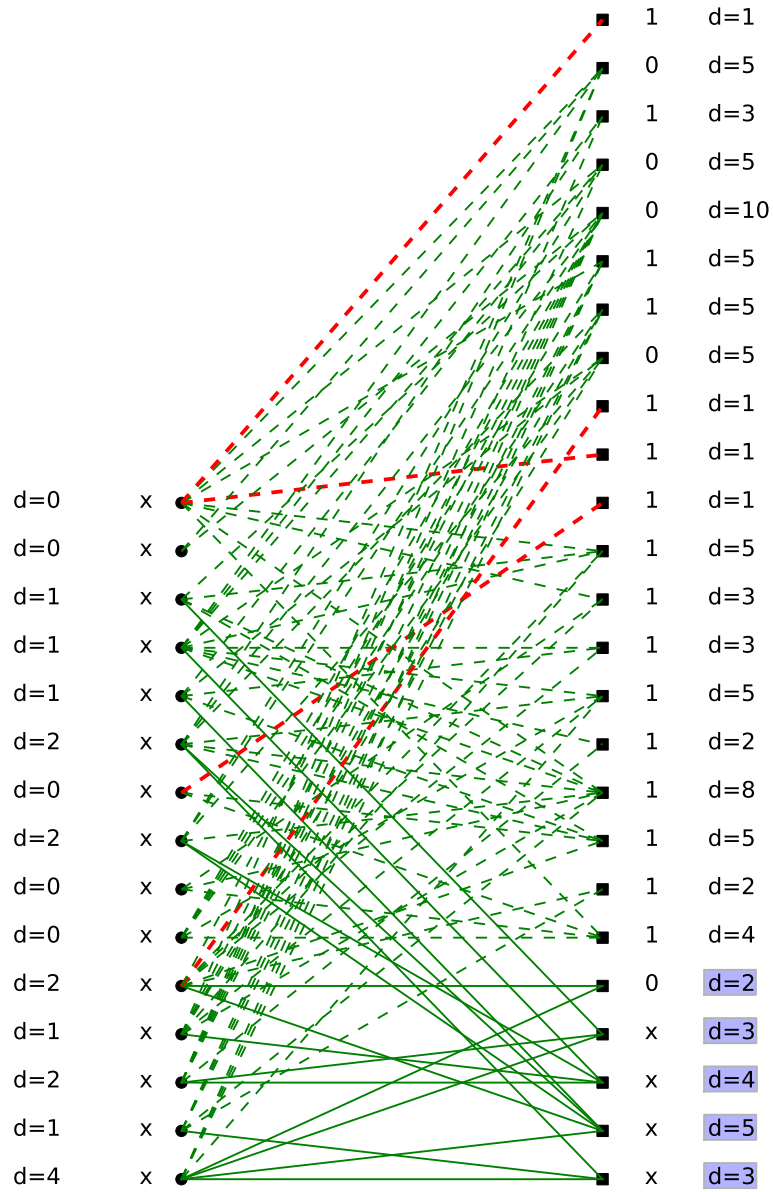
Progressive LT Decoding example

- Decode as we received symbols from the channel



Progressive LT Decoding example

- Decode as we received symbols from the channel
- See animation.

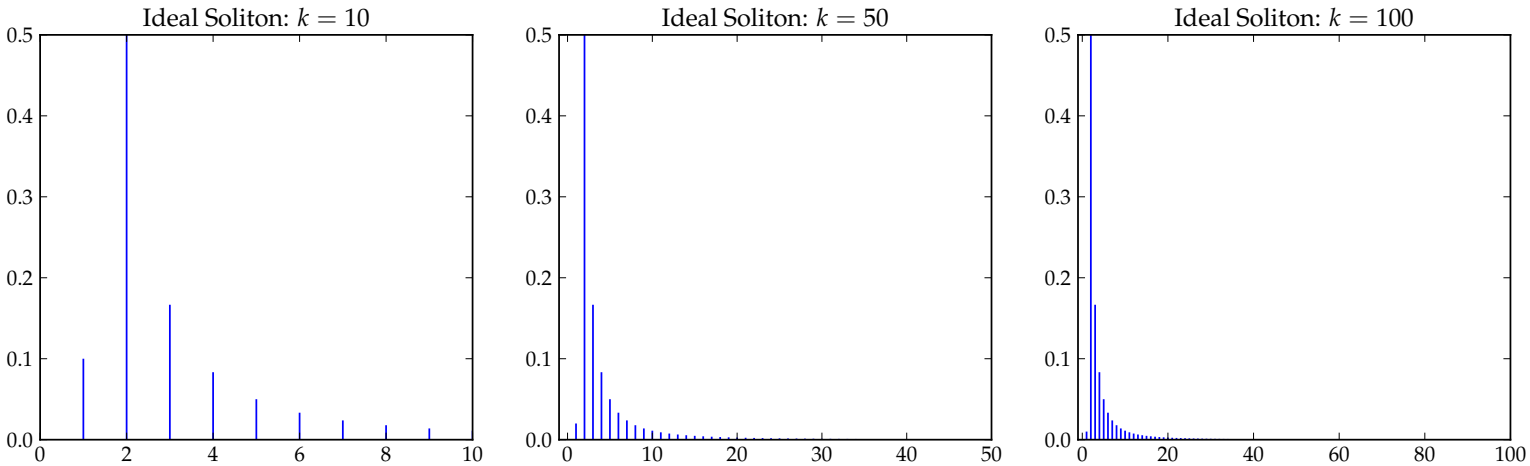


Ideal Soliton distribution

Ideal Soliton distribution $\rho(i)$ is given by

$$\rho(1) = \frac{1}{k} \quad \text{and} \quad \rho(i) = \frac{1}{i(i-1)}, i = 2, \ldots, k.$$

- k encoding symbols are sufficient to recover k information symbols.
- The expected ripple size is 1 during the whole decoding process.

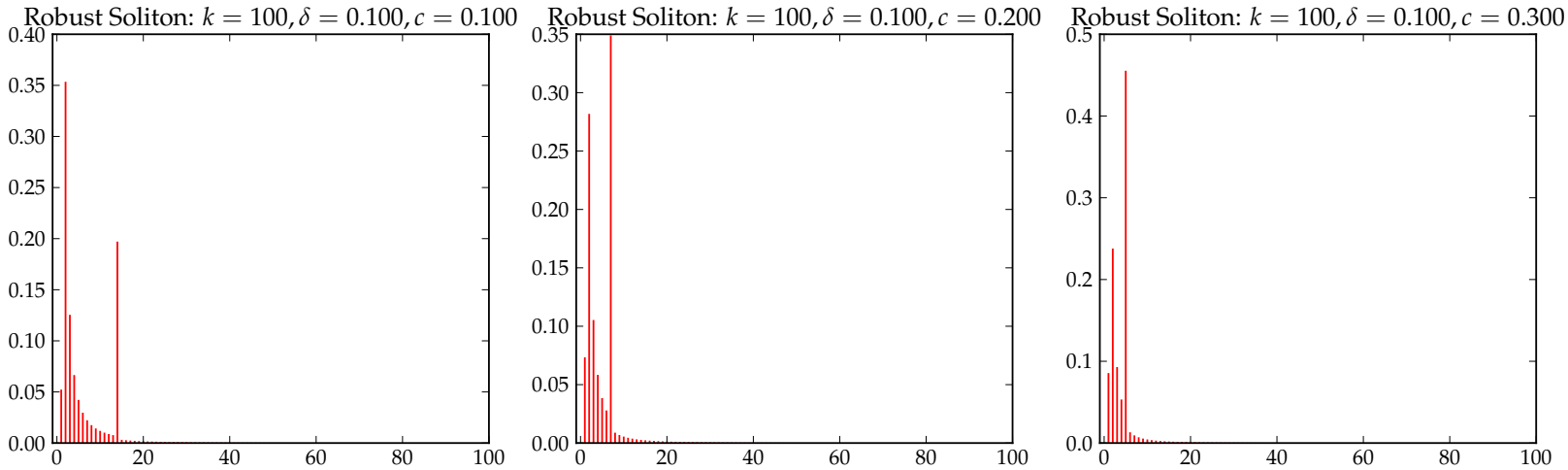


Robust Soliton distribution

Robust Soliton distribution $\mu(i) = \frac{1}{\beta}(\rho(i) + \tau(i))$, where $\rho(i)$ is the Ideal Soliton distribution and

$$\tau(i) = \begin{cases} \frac{R}{ik} & \text{for } i = 1, \ldots, k/R - 1 \\ \frac{R \ln(R/\delta)}{k} & \text{for } i = k/R \\ 0 & \text{for } i = k/R + 1, \ldots, k \end{cases} .$$

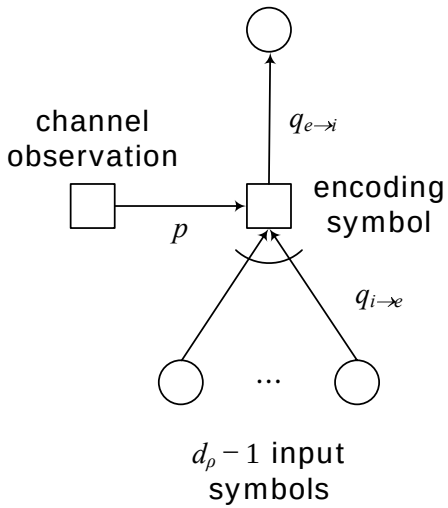
- Adding $\tau(i)$ maintains the ripple with expected size R.
- δ denotes the allowable failure probability.
- Choose R and δ - design issue.



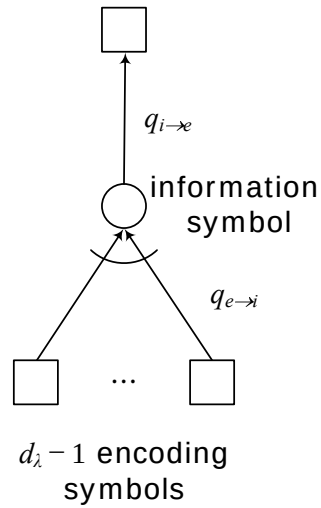
Performance of LT codes

Information symbols are uniformly chosen by each encoding symbol. Thus, asymptotically, the degree distribution of information symbols is a Poisson distribution.

- Encoding symbol node



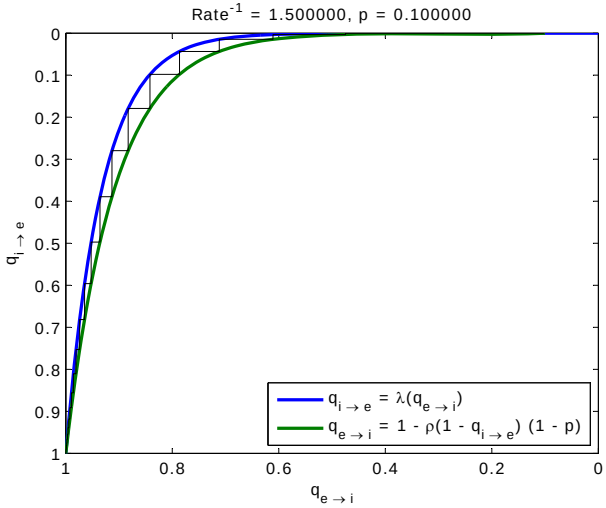
- Information symbol node

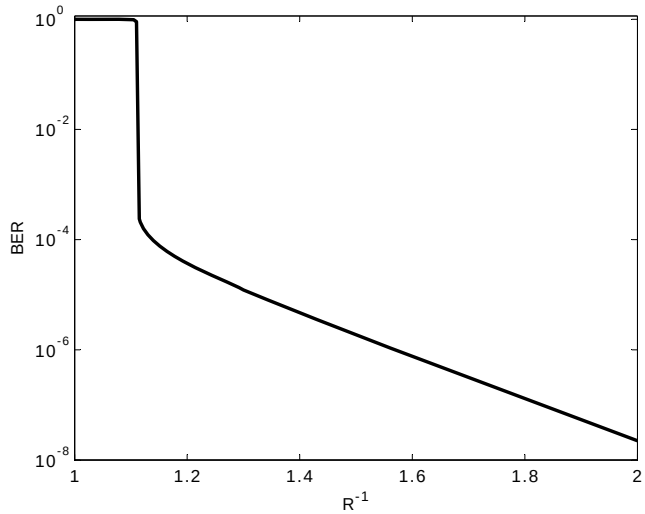


$$q_{e\to i} = 1 - \rho(1 - q_{i\to e})(1 - p)$$

$$q_{i\to e} = \lambda(q_{i\to e})$$

Error floor





- Lower bound: an information symbol is of degree 0 with probability of $L_0 = e^{-d_\rho / R}$.
- Applying precoding technique can be a solution (Raptor codes).

Outline

- Tornado Codes
- LT Codes
- Raptor Codes
- Extensions
- Summary

Another Fountain Code Idea

- LT codes demonstrated the power of rateless codes
- Requires $O(\log k)$ decoding complexity
- Issue is that all input bits must be decoded

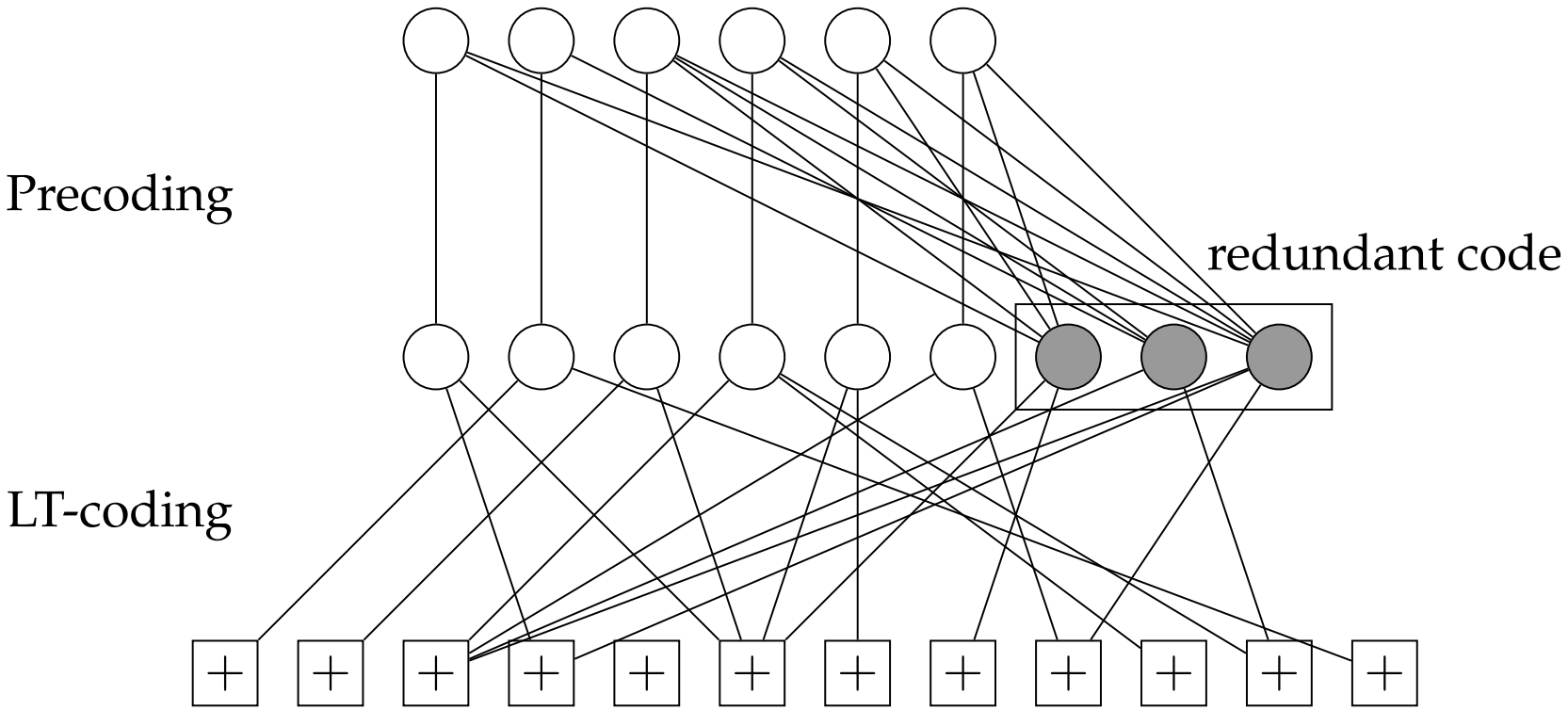
Idea: What if we relax this condition?

Raptor Code

Add pre-code before LT code:

- Standard erasure code such as Tornado, irregular RA, right-regular LDPC or truncated LT
- Allow for δ erasures on input side of LT
- Reduce complexity but require two-stage code
- Need buffer to store intermediate bits

Raptor Code



Key Result

Assume pre-code and LT code are well designed. Then:

- Overhead ϵ
- Space $(1+\epsilon)/(1+\epsilon/2)$
- Encoding complexity $O(\log(1/\epsilon))$
- Decoding complexity $O(\log(1/\epsilon))$

[Shokrollahi (2006)]

Ingredients

The pre-code should:

- be an erasure code
- have rate $R = (1+\epsilon/2)/(1+\epsilon)$
- Can decode up to $(1-R)/2$ errors

The LT code should:

- require $n = (1+\epsilon/2)m + 1$ encoding bits to recover at least $(1-\delta)m$ input bits, where $\delta = (\epsilon/4)/(1+\epsilon)$
- have node degree distribution

$$P_D(x) = \frac{1}{\mu+1}\left(\mu x + \sum_{i=2}^{D} \frac{x^i}{(i-1)i} + \frac{x^{D+1}}{D}\right),$$

where $D = \lceil 4(1+\epsilon)/\epsilon \rceil$, $\mu = (\epsilon/2) + (\epsilon/2)^2$

Practical Considerations

Finite-length codes:

- Previous results are asymptotic
- Can have bad performance in low blocklength ($\sim 10,000$)
- Use linear program for LT degree distribution
- Use LDPC pre-codes
- Use ML estimator

Practical Considerations

Systematic codes:

- Want input bits as part of fountain
- Use matrix representation of code:
  - G is generator matrix for pre-code
  - S is adjacency matrix of encoding packets
  - Output $z = xGS$
- Find full-rank submatrix R
- Preprocess input to be $\hat{x} = xR^{-1}$

Outline

- Tornado Codes
- LT Codes
- Raptor Codes
- Extensions
- Summary

Fountain Codes for Noisy Channels

- Apply fountain codes and belief propagation to other noisy channels
  - BP decoding
- Raptor code on binary-input memoryless symmetric channels (BIMSCs) [Etesami & Shokrollahi (2006)]
  - No universal Raptor codes for channels other than BEC
  - Asymptotically, Raptor codes can achieve a rate within a constant gap to capacity
  - Analysis: semi-Gaussian approximation

Distributed Fountain Codes

Use fountain codes to implement distributed network memory using unreliable sensor nodes [Dimakis et al. 2006]

- k data-generating nodes
- n unreliable storage nodes
- Approximate data collection queries: recover at least $(1-\delta)k$ original data packet after querying any $(1+\varepsilon)k$ nodes.
- Challenges:
  - bandwidth constraint: small degree in the bipartite graph
  - distributed construction: local encoding

Fountain code for Source Coding

- Linear source code can achieve the entropy rate of memoryless sources
- The parity check matrix H of a good channel code and be a good compressor
- Practical sources are not memoryless
- Solution:
  - Burrows-Wheeler transform: produce outputs that are piecewise i.i.d.
  - Detect the i.i.d. segments and use capacity achieving codes and modified BP for compression
  - Fountain code: its rateless properties handle heterogeneous sources more naturally [Caire et al. 2005]

Weighted Fountain Codes

A generalization of the fountain code construction [Rahnavard et al. 2007]

- Select the neighbors of the encoded symbols *non-uniformly* from the information symbols

Benefits:

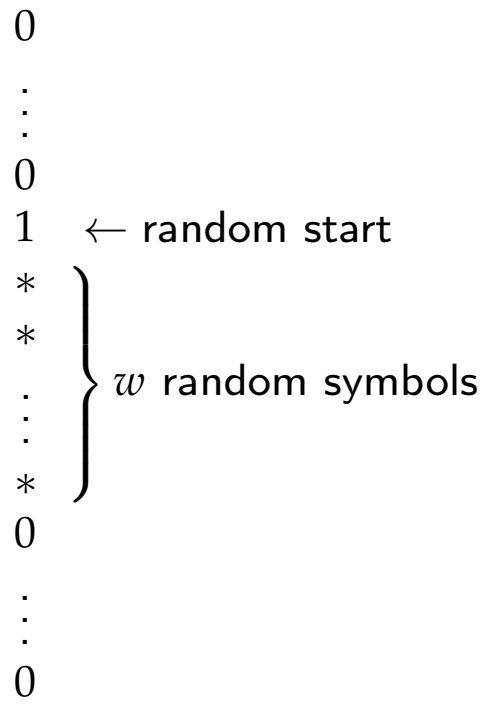
- Unequal error protection (UEP) $\approx$ unequal recovery time (URT)

Windowed Fountain Codes

Another generalization of fountain codes [Studholme & Blake, 2006].

Characteristics:

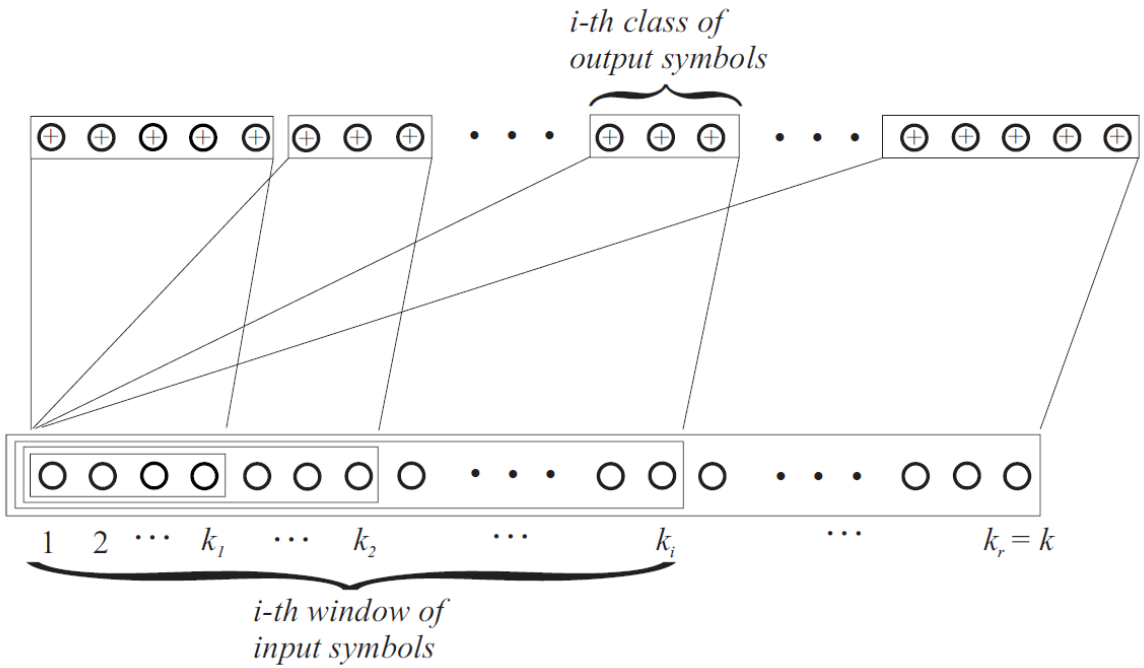
- constant overhead
- almost surely decoding
- higher decoding complexity: $O(k^{3/2})$
- effective for low number of information symbols



Expanding Window Fountain Code

Apply the windowing idea for unequal error protection [Sejdinovic et al.2009].

- choose strictly increasing subsets of the information symbols to be a sequence of windows
- the input symbols in the smallest window will have the strongest error protection
  - contained in all larger windows
  - more likely to be used when producing encoding symbols.
- Analysis: And-Or tree technique [Luby et al. 1998]



Fountain Capacity

A new definition of rate [Shamai et al. 2007]

- Standard rate definition: *ratio of the information symbols and transmitted symbols*
  - encoder's perspective: "pay-per-use"
- Fountain rate: *ratio of information symbols transmitted to channel symbols received*
  - decoder's perspective: "pay-per-view"

Example:

- a erasure broadcast channel with different channel quality
- compound channel: over-pessimistic
- fountain capacity region: $[0, C_1] \times [0, C_2] \times \cdots \times [0, C_n]$, where C_i is the point-to-point Shannon capacity.

Results in [Shamai et al. 2007] are for point-to-point case only:

- always upper bounded by the Shannon capacity.
- equal to Shannon capacity for stationary memoryless channels.

Outline

- Tornado Codes
- LT Codes
- Raptor Codes
- Extensions
- Summary

Why are Fountain Codes Good?

- rateless/universal
- capacity-achieving/approaching
- low encoding/decoding complexity
- unequal error protection

Applications:

- hybrid ARQ [Soijanin et al.2006]
- scalable video streaming [Bogino et al.2007]
- any system that involves an erasure channel...

Performance Summary of Fountain Codes

	Tornado	LT	Raptor
Rateless	No	Yes	Yes
Overhead	ϵ	$\epsilon \to 0$	$\epsilon \to 0$
Encoding Complexity per Symbol	$O(\epsilon \ln(1/\epsilon))$	$O(\ln(k))$	$O(1)$
Decoding Complexity per Symbol	$O(\epsilon \ln(1/\epsilon))$	$O(\ln(k))$	$O(1)$