Lecture Summary

EE568
KUMAR
Sp '02

Lecture 25 – GDL VI – the BCJR algorithm (this time, from a GDL viewpoint)

- in this lecture we assume that we have a set of local domains and local kernels and that the local domains have been organized into a junction tree

- the MPF problem can be solved by “message-passing” and “objective function” computation – we explain
  - it is easier to think of the message along an edge as representing the state of the edge; we distinguish between edge (i,j) and edge (j,i);
  - by message-passing, we simply mean that the state of the edge is updated
  - initially the state of every message is set equal to 1
  - initially the objective function of each node is set equal to its local kernel
  - page 2 gives the message update rule
  - page 3 gives the objective fn update rule

- there are many message-passing schedules, that can be used;

- the important thing to keep in mind is simply that a given node i is ready to compute its objective fn only when it has (directly or indirectly) received a message from every other node in the junction tree;

- a message-trellis (page 5 of lecture notes, see also GDL paper) is a simple means of keeping track of this;

- set own the MPF problem corresponding to ML code symbol decoding of a convolutional code;
  - we identified the local domains, the local kernels and set up the junction tree;
  - message passing on this junction tree will accomplish what the BCJR algorithm does;
  - however, it is easier to see the connection with the BCJR algorithm if a “reduced: junction tree is drawn

Lecture 24 – GDL V

- how to go about testing whether or not it is possible to organize the local domains into a junction tree
- weight of a node
- node-weight of a graph
- weight of an edge
- edge-weight of a graph
- condition that holds for any tree: edge wt ≤ node weight - n
- equality holds when the given tree is in fact a junction tree:
  - thus a junction tree maximizes edge-weight;
  - a tree that has maximum edge-wt can be constructed via Prim’s greedy algorithm
- example

- how message passing in the junction tree solves the MPF (marginalize a product function “problem

Lecture 23 – GDL IV

- a note on computing the \( \min^* \) operation
- the general problem that the GDL algorithm solves is called the marginalize a product function problem (MPF)
- we introduce some terminology that will help us state the MPF problem, including local domains, local kernels, the global kernel, objective functions
- outlined steps needed to solve the MPF problem without going into the details
- provided an example of the MPF problem and its solution – ML code symbol decoding of the \([7, 4, 2]\) code
- now begin the details of how the MPF problem is solved
- defn of a tree and of a junction tree, example;

Lecture 2 – GDL III

- example – the max product semi-ring
- showing how the distributive law can be applied in the max-product semi-ring
- it is more convenient to work with the logarithms of probabilities rather than with the probabilities themselves; the ML codeword decoding problem can also be stated in terms of the logarithms of the probabilities; this results in the max product semi-ring being replaced by the min-sum semi-ring;
thus the ML codeword decoding problem can be carried out either by working directly with probabilities and in the max product semi-ring or else with the logarithms of the probabilities and in the min sum semi-ring;

similarly, the ML code symbol decoding problem can be carried out either by working directly with probabilities and in the sum product semi-ring or else with the logarithms of the probabilities and in the min* sum semi-ring;

the min*sum semi-ring is a semi-ring in which a new operation, the min* operation is introduced;

began formulation a general problem that is solved by the GDL;

Lecture 21 – GDL II

we present here an example in which it is shown how the problem of ML code symbol decoding of a block code is also an instance where the distributive law can step in and streamline the computation;

we do the same with ML codeword decoding of a block code; however, this time, our two operations in place of the familiar addition and multiplication are now the “max” operation and multiplication (the product operation);

with this as motivation, we now study an abstract object called a semi-ring where there are two operations that satisfy certain axioms, including that of the distributive law; the advantage of this admittedly abstract approach is that it allows us to identify and efficiently solve a great many problems as being different examples of the same fundamental problem;

it is widely held in the information, coding and communication theory communities, that the GDL (or other equivalent algorithm) has become pretty much a “must have: tool for today’s engineers

Lecture 20 – Generalized (application of the) Distributive Law (GDL)

the bit multiplicities of the turbo and convolutional code are compared

an example where the distributive law helps in reducing the number of operations required to evaluate a function is provided

Lecture 19 – Spectral Thinning

the bit error probability of any particular turbo code is difficult to analyze; however, the average bit error probability of the ensemble of turbo codes can be analyzed by using the union bound;
this requires knowledge of the average input-output weight enumerator; but it is shown how this can be computed;

Lecture 18 – Explaining turbo code performance

- rewriting in full, the BCJR recursions
- towards explaining turbo code performance, we show how to compute the IOWE for parallel concatenated codes

Lecture 17 – the BCJR algorithm

- a quick overview of decoding of the turbo code
- the BCJR algorithm
- notes from an earlier lecture were unintentionally appended to this year's lecture notes (from page 11 of LEC 17 onwards, you may disregard these)

Lecture 16 – Bit Error Probability

- Shannon’s expression for the capacity of a channel
- this discussion attempts to explain that you need a signal to noise ratio $E_b/N_0$ of at least 0 dB to be able to communicate 1/2 bit of information per channel message symbol reliably
- we begin discussing turbo codes; one of the ingredients of turbo codes are recursive systematic convolutional encoders (RSE); RSE, and their encoding are discussed here;
- a second ingredient of turbo codes is that in place of the Viterbi algorithm for ML codeword decoding of a convolutional code, we now have an algorithm that carries out optimal message bit decisions; this algorithm is called the BCJR algorithm;

Lecture 15 – $P_{be}$ (continued)

- bit error probability of a convolutional code over an AWGN channel
- geometric uniformity which explain why in analyzing the bit error probability of a convolutional code, we can assume that the transmitted codeword is the all-zero codeword;

Lecture 14 – Bit Error Probability

- pairwise error probability of codewords in case of a BSC
- input-output weight enumerator (IOWE) $\{A_{w,d}\}_{w=0,d=0}^{k,d=n}$ of a convolutional code (cc)
• relation of IOWE to the probability of bit error
• derivation of bit error probability of a cc

Lecture 13 – Generating Function (continued)

• generating function technique for finding the path enumerator \( A_F(L, D, I) \) of a convolutional code
  - modified state diagram
  - power series in several variables

• union bound on codeword error probability in terms of weight distribution

Lecture 12 – Generating Function (title is a misnomer!)

• MLD of a convolutional code over an AWGN channel

Lecture 11 – Trellis of a convolutional code

• condition on \( G(D) \) to avoid CEP in the general rate \( k/n \) case
• state diagram of a convolutional code
• trellis diagram of a convolutional code
• “true” rate of a convolutional code
• decoding of a convolutional code over a BSC via the Viterbi decoding algorithm

Lecture 10 – Catastrophic Error Propagation (CEP)

• two notes regarding arithmetic in formal power series
• the Extended Euclidean division algorithm for finding the gcd of two integers
• using the same algorithm to find the gcd of two polynomials;
• making the connection with CEP;
• explaining why it is sufficient for the gcd to be a power of \( D; \)
• statement of the necessary and sufficient condition for a convolutional code of rate \((1/n)\) to not have CEP; we did not prove the necessary condition part in class;
• an example of a rate \( k/n \) convolutional code and how one can test for CEP there;

Lecture 9 – Convolutional Codes
• difference between block and tree codes
• two example convolutional codes
• the general convolutional input/output relationship
• the input-output relation in terms of power series
• the polynomial generator matrix (PGM), examples;
• the memory of the encoder
• an example of catastrophic-error propagation
• the field of formal power series in D
• arithmetic in the field of formal power series

Lecture 8 – Performance Analysis

• weight distribution of a linear block code and \( P_{ue} \)
• probability \( P_{b} \) of bit error from the standard array table;
• an overview of some of the commonly used block codes;
• defn of a cyclic code; examples; parameters of BCH codes, Reed-Muller codes;
• McWilliams identities

Lecture 7 – Standard Array Decoding

• cosets of a linear block code, coset leader
• each coset has a unique syndrome
• bounded distance decoding (BDD)
• performance analysis of a linear block code
• \( P_{uwe}, P_{ue}, P_{b} \)

Lecture 6 – Decoding of Block Codes

• asymptotic bounds
• decoding of block codes - Maximum Likelihood Decoding (MLD)
• MLD = minimum Hamming distance decoding (MDD) over a BSC
• Standard Array Decoding (SAD) – a means of accomplishing MDD
• why long codes are of interest

14 Lecture 5 – Bounds on the size of a Code
• \(d_{\text{min}} = s + 1\)
• Singleton bound and MDS codes
• Hamming bound, perfect codes and the Golay code
• Gilbert-Varshamov (G-V) bound
• why long codes are of interest

Lecture 4 – Linear Block Codes (continued)
• \(GH^t = [0]\) test for parity-check matrix
• \(H\) for some example codes
• a general Hamming code
• systematic generator matrix and equivalent codes
• \(d_{\text{min}}\) of a linear block code, \(w_{\text{min}} = d_{\text{min}}\)
• \(d_{\text{min}}\) from \(H\)

Lecture 3 – Generator and parity-check matrices
• defined subspace (to complete notes of Lec 2)
• defined dimension of a code
• generator matrix \(G\) of a linear code; the encoding equation;
• \(G\) for the Hamming code;
• the repetition and parity-check codes and their generator matrices;
• parity-check (pc) matrix \(H\) via the dual code;
  – orthogonal vectors; the dual code; example; relating parameters of the code and of its dual;
  – the dual code is the null-space of \(G\);
  – pc mx \(H\) as \(G\) for the dual; examples;
Lecture 2 - Minimum distance and error-correction

- \( d_{\text{min}} \) and error-correction; proof;
- binary linear codes can be viewed as either groups or else vector spaces;
- group axioms and example; defn of a subgroup; test for a subgroup of \( F_2^n \);
- vector space axioms and example; subspaces; test for subspace of \( F_2^n \);

Lecture 1 - Introduction

- binary arithmetic
- an example code, the hamming code of length 7
- the Hamming weight function and properties;
- the Hamming distance function and properties
- minimum distance of an error-correcting code (ecc)