INF244, Autumn 2013, Project

October 30, 2013

The task is to write some code to implement partially-generic message-passing decoding on a factor graph, where the factor graph is specified by an input text file. Variable nodes in the factor graph could have input size, but for this project we shall assume that all variable nodes have two-state input (i.e. can be represented by vectors of size two). We shall also assume that state variable nodes have power-of-two state sizes 2,4,8, or 16 (more if you want). Function nodes shall be factors of the global function, g, and shall be functions of the variable nodes. The global function, g, should output to the real numbers but, to make life easier we just make it output to the integers. You shall input g from a text file in terms of your chosen factorisation. Those factors representing rows of a parity check matrix shall be described by indicator functions. For rooted-trees we shall assume the functions output to the integers. Those factors relating to samples from a probability distribution (and, for our project, connected to a single two-state variable) will be of the form $f(x|y)$, where the y variable is suppressed, and the inputs shall be sampled from an AWGN distribution, whose mean and standard deviation shall be specified by the text file. If, instead, the function $f(x|y)$ is a function of two two-state variables x and y, then it implements the function $x == y$, and can be represented by the binary string 1001 (i.e. $f(x, y) = x + y + 1$). In this case, the y variable is sampled from the AWGN distribution.

Specification:

- INPUT: a text file named 'MP.txt'. Format of 'MP.txt' is as follows:
  - Line 1: Number of variables nodes. - (non-negative integer, 0 - 100)
  - Line 2: Number of state variable nodes. - (non-negative integer, 0 - 10, followed by size (#states) of each state node)

- Line 3: Number, m, of function nodes. - (non-negative integer, 0 - 100)
- Line 4: Specify m functions, one per line. - (see examples below)
- Line $4+m$: Character to select decoding algorithm to perform.
  - 'A': for SPA - flooding (see pages 69–72 of FGSPA.pdf).
  - 'B': for SPA - rooted-tree - (must be consistent with previous function specs).
  - 'C': for forward/backward - (must be consistent with previous function specs).
- If 'A' on line $4+m$
  - Line $5+m$: Number of iterations to perform. - (non-negative integer)
  - Line $6+m$: Number of independent decodings to perform. - (non-negative integer, 0 - 10000)
  - Line $7+m$: R - code rate (real)
  - Line $8+m$: E_b - bit energy (real)
  - Line $9+m$: N_0 - noise (real)
    From the last three inputs, code rate, bit energy, and noise, one can model the AWGN channel. See discussion on modeling AWGN later in this note.
- If 'B' on line $4+m$
  - Line $5+m$: Non-negative integer, k, to specify variable node k at root of tree - (non-negative integer), or 'a' to compute all marginals.
- If 'C' on line $4+m$
  - Line $5+m$: Character to select message-passing variant to perform.
    - 'S': for SPA (BJCR-type)
    - 'M': for Min-Sum (Viterbi-type).
  - Line $6+m$: Number of independent decodings to perform. - (non-negative integer, 0 - 10000)
  - Line $7+m$: R - code rate (real)
  - Line $8+m$: E_b - bit energy (real)
  - Line $9+m$: N_0 - noise (real)

- RUN MESSAGE-PASSING - Assuming all input node variables are in state zero (zero codeword 'sent').
- OUTPUT:
  - If SPA - flooding chosen then
    - O/P Bit-Error Rate (BER)
    - O/P Word-Error Rate (WER)
  - If SPA - rooted-tree chosen then

    - O/P marginal values stored in variable node k or in all variables, as appropriate.
  - If forward-backward chosen then
    - O/P Bit-Error Rate (BER)
    - O/P Word-Error Rate (WER)

1 Example Textfile Format

1.1 Decoding on a Factor Graph with Cycles: for the graph on page 41 of FGSPA.pdf (number variables from zero!):

```
6                           (#variable nodes)
0                           (#state nodes)
9                           (#function nodes)
[0,2,3]10010110             (Boolean function truth tables)
[0,1,4]10010110
[1,2,5]10010110
[0]G                        (single variable functions taking sampled AWGN
[1]G                            - 'G' stands for AWGN samples)
[2]G
[3]G
[4]G
[5]G
A                           (flooding)
50                          (#flooding iterations)
100                         (#decodings)
0.5                         (code rate - note that the parity matrix has 3 rows, hence 0.5 = 3/6)
10.0                        (E_b)
1.0                         (N_0          make E_b/N_0 bigger to reduce decoding errors)
```

Then, RUN.

- Plot curves showing $10\log_{10}(\frac{E_b}{N_0})$, (x-axis), against $\log_{10}$(bit-error-rate (BER)), (y-axis), as the total number of flooding iterations varies, and do likewise for the word-error-rate (WER).

1.2 Forward-Backward Decoding on a Trellis: for the trellis on page 34 of FGSPA.pdf, using the principles outlined in pages 46–56 of FGSPA.pdf (number variables from zero!):

```
9                                           (#variable nodes)
7:1,2,4,4,4,2,1                             (#state nodes, followed by size of each node)
12                                          (#function nodes)
[0,3,9,10]10000001                          (function truth tables)
[1,4,10,11]1000000000010000000001000000100
[2,5,11,12]10000010000000000001010000000000000000000010100000000000000000001
[6,12,13]100000000001000000001000000010
[7,13,14]1000010000100001
[8,14,15]1001
[3]G                        (single variable functions taking sampled AWGN
[4]G                            - 'G' stands for AWGN samples)
[5]G
[6]G
[7]G
```

```
[8]G
C                           (BJCR)
S                           (sum-product variant)
100                         (#decodings - probably you want to make this 1 to start with)
0.5                         (code rate - note that the parity matrix has 3 rows, hence 0.5 = 3/6)
10.0                        (E_b)
1.0                         (N_0          make E_b/N_0 bigger to reduce decoding errors)
```

Then, RUN.

- Plot curves showing $10\log_{10}(\frac{E_b}{N_0})$, (x-axis), against $\log_{10}$(bit-error-rate (BER)), (y-axis), as the total number of flooding iterations varies, and do likewise for the word-error-rate (WER).

1.3 Computing All Marginals of a Rooted-Tree: For a graph almost identical to that on page 9 of FGSPA.pdf, except that x_4 is disjoint, and using the method described in pages 8 − 22 (number variables from zero!):

Assume all variables are two-state, and, for simplicity, that all functions output to the integers. For example, let $f_A(x_0) = 1 + x_0$, $f_B(x_1) = 3x_1$, $f_C(x_0, x_1, x_2) = x_0 + x_1 + x_2$, $f_D(x_2, x_3) = x_2 - x_3 + 1$, $f_E(x_4) = 2 - x_4$. Then,

```
5                           (#variable nodes)
0                           (#state nodes)
5                           (#function nodes)
[0]1,2                      (function truth tables)
[1]0,3
[0,1,2]0,1,1,2,1,2,2,3
[2,3]1,2,0,1
[4]2,1
B                           (rooted-tree)
a                           (compute all marginals)
```

Then, RUN.

Print out all marginals - for the example here, the marginal for x_0 should be $(63, 198)$ and, for x_4, $(174, 87)$.

2 Simulating an Additive White Gaussian Channel

(i.e. simulating a normal distribution, i.e. a Gaussian distribution). Let us assume Binary Phase Shift Keying (BPSK) modulation. Each codeword bit, c_i, is then modulated to a real (more generally complex) signal $t_i = (2c_i - 1)\sqrt{E_c}$, $0 \leq i < n$, where $E_c = RE_b$, R is code rate, and E_b is bit energy. Assuming the channel acts independently and additively on the bits,

the received vector r is given by $r = t + \nu$. For ν Additive White Gaussian Noise (AWGN), each element of ν is independent, identically distributed (i.i.d.), with zero mean, and variance $\sigma^2 = N_0/2$, where N_0 is bit noise.

One obtains an expression for $P(c_i = 1|r_i)$ as,

$$P(c_i = 1|r_i) = \frac{p(r_i|t_i = \sqrt{E_c})}{p(r_i|t_i = \sqrt{E_c}) + p(r_i|t_i = -\sqrt{E_c})},$$

where one has assumed that $P(c_i = 1) = P(c_i = 0) = \frac{1}{2}$. For AWGN we get that

$$p(r_i|t_i = \sqrt{E_c}) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2\sigma^2}(r_i - \sqrt{E_c})^2}.$$

Therefore,

$$P(c_i = 1|r_i) = \frac{1}{1 + e^{-2\sqrt{E_c} r_i/\sigma^2}}.$$

It follows, of course that $P(c_i = 0|r_i) = 1 - P(c_i = 1|r_i)$.

The above analysis for AWGN can be translated to the factor graph as follows. The function nodes $f(y_i|x_i)$ are two-state vectors with entries that sum up to one,

$$f(y_i|x_i) = \begin{pmatrix} P(c_i = 0|r_i) \\ P(c_i = 1|r_i) \end{pmatrix},$$

where $P(c_i = 1|r_i)$ is defined above in terms of the AWGN channel. The variable nodes, x_i, are two-state vectors whose entries are initialised to

$$x_i = \begin{pmatrix} 1/2 \\ 1/2 \end{pmatrix}.$$

It remains to sample $r_i = t_i + \nu_i$, $\forall i$, where ν_i is sampled from a Gaussian distribution. It shall be assumed, for simulation purposes, that the all-zero word is always sent, so $t_i = -\sqrt{E_c}$, $\forall i$. We can make this simplification because we are only considering linear codes, so all codewords are simple translates of each other. To sample a Gaussian distribution we shall use the Box-Muller method.

2.1 The Box-Muller Method

(see, for instance, `http://en.wikipedia.org/wiki/Box-Muller_transform`, and `http://mathworld.wolfram.com/Box-MullerTransformation.html`, for more details). Typically a software (or hardware) algorithm to output

random numbers according to a Gaussian (i.e. normal) distribution is not readily available. What one can do is generate random numbers according to some constant distribution. The Box-Muller method is a way to generate random numbers according to an approximate Gaussian distribution by making use of random numbers generated from a constant distribution. Here's what you do.

- 1. Generate random numbers from a constant distribution (e.g. by using 'rand' in c, or something like that), i.e. generate $u, v \in [-1, +1]$.
- 2. Assign $s = u^2 + v^2$.
- 3. If $s = 0$ or $s \geq 1$ then go back to step 1.
- 4. Assign $z_0 = u \cdot \sqrt{\frac{-2 \ln s}{s}}$, $z_1 = v \cdot \sqrt{\frac{-2 \ln s}{s}}$.

Then z_0 and z_1 are (approximately) independent, identically distributed (i.i.d.) normal (i.e. Gaussian) variables with zero mean and variance $\sigma^2 = 1$.

If z is a random normal variable with mean 0 and variance $\sigma^2 = 1$, then $z' = az + b$ is a random normal variable with mean b and variance $\sigma^2 = a^2$. So the Box-Muller algorithm allows to easily generate a normal sampling with any mean and variance.

For our purposes we wish to add AWGN, ν, to the modulated codeword, t, so as to generate the received word $r = t + \nu$. Thus, for each bit position, we have $r_i = t_i + \nu_i$, where ν_i is sampled from a normal distribution with mean 0 and variance $\sigma^2 = N_0/2$.

To recap. We input code rate, R, bit energy E_b, and noise energy, N_0, from which we compute $E_c = RE_b$, $\sigma^2 = N_0/2$, and SNR$= E_b/N_0$. For simulation purposes we assign $t_i = -\sqrt{E_c}$, $\forall i$, (i.e. the all-zero codeword). For each decoding instance, we generate n normal samples, ν_i, $0 \leq i < n$, where $\sigma^2 = N_0/2$. Thus we compute $r_i = t_i + \nu_i$, $\forall i$. Then, using the values for r_i, E_c and σ^2, we can compute $P(c_i = 1|r_i) = \frac{1}{1+e^{-2\sqrt{E_c} r_i/\sigma^2}}$, $\forall i$. Finally, by observing that $P(c_i = 0|r_i) = 1 - P(c_i = 1|r_i)$, we can assign the two-state vector, $f(y_i|x_i)$ as $f(y_i|x_i) = \begin{pmatrix} P(c_i = 0|r_i) \\ P(c_i = 1|r_i) \end{pmatrix}$, $\forall i$. With x_i initialised to $x_i = (\frac{1}{2}, \frac{1}{2})^T$, $\forall i$, we can now commence SPA iteration.

- By quantizing the x-axis to 0.01 steps, plot the value of z_0 (x-axis, ± 0.005) against the number of times that value is generated (y-axis),

where at least 10000 samples are taken. Hopefully you should get a Gaussian 'bell' curve.