Course 67800 Spring 2021

Probabilistic Methods in Artificial Intelligence

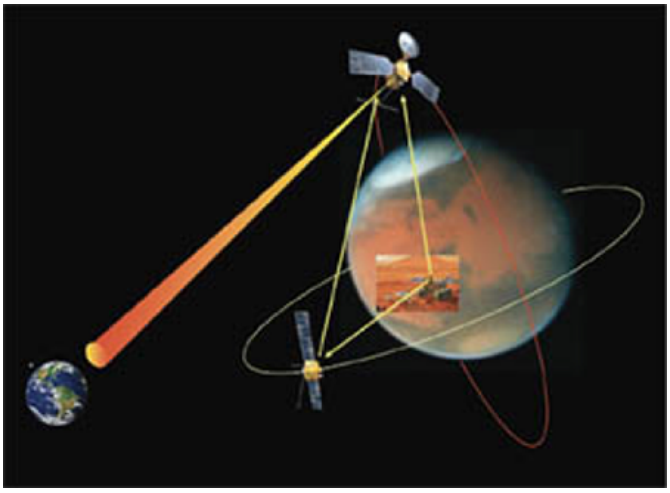
Programming Assignment 2: Approximate Inference

Deadline: Thursday 27/5/21, 23:59

Prof. Gal Elidan *TA: Hagai Rappeport*

1 Background

In this programming assignment [1], you will implement an algorithm for reliable communication in the presence of noise, a technique used by NASA to communicate from deep space.



The Mars Rover is trying to communicate with mission headquarters on Earth. The message the Mars Rover wants to transmit is a binary sequence $X \in \mathbf{B}^N$ of N bits, where $\mathbf{B} = \{0, 1\}$. To deal with transmission noise, the Mars Rover appends N redundant bits to each message to allow for error correction. The redundant bits are chosen using a clever scheme: the message is encoded as $Y = GX$ by a special matrix $G \in \mathbf{B}^{2N \times N}$ – a *generator matrix*. G is chosen such that the first N bits of Y are equal to X, while the remaining N bits are redundant "parity checks" (here GX is computed modulo 2, so that $Y \in \mathbf{B}^{2N}$).

The encoded message $Y \in \mathbf{B}^{2N}$ obtained this way is special because it is a *codeword*: the Mars Rover and mission headquarters have agreed that all valid messages or codewords are such that $HY = \mathbf{0}$, where $H \in \mathbf{B}^{N \times 2N}$ is a pre-specified *Low Density Parity Check* (LDPC) matrix. The matrices G and H are paired, and chosen so that multiplying by G creates valid messages (codewords), and H can be used to check for errors in a received message (on Earth). Mathematically, $HG = 0$, so that $HY = HGX = \mathbf{0}$ for any input message X. This codeword $Y \in \mathbf{B}^{2N}$ is then transmitted through deep space back to the mission control on Earth who then receives it as the noisy $\widetilde{Y}$. The decoding process refers to the procedure of recovering the ground truth codeword Y (and thus also X, the first N bits of Y) from the noisy version $\widetilde{Y}$. You will implement this process using *Loopy Belief Propagation*. We'll represent the problem as a *Markov Network*, and specifically as a *Factor Graph*. We define two types of factors: Unary factors $\phi_{Y_i}(Y_i)$ associating Y_i with $\widetilde{Y}_i$ (which are treated as constants) – so messages that are similar to the one received are more likely. In addition, we define *parity-check factors* $\phi_{P_j}(\boldsymbol{Y}_j)$, each defined over a set of variables $\boldsymbol{Y}_j \subseteq \boldsymbol{Y}$ (defined by rows of H) and assigning zero probability to messages that are not valid codewords.

The columns of H correspond to codeword bits, and rows to parity check constraints. We define $H_{ij} = 1$ if parity check P_i depends on codeword bit Y_j, and $H_{ij} = 0$ otherwise. Valid codewords are those for which the sum of the bits connected to each parity check, as indicated by H, equals zero in modulo-2 addition (i.e., the number of "active" bits must be even). As illustrated in Fig. 1, we can visualize these parity check constraints via

[1] Assignment adapted from Brown University CS242 instructed by Erik Sudderth

a corresponding *factor graph*, a cluster graph which contains a node for each variable in the MN and a node for each factor, with edges connecting each factor to the variables in its scope. As clusters, each factor node contains the variables in its scope and each single-variable node contains the single variable. Each sepset (edge) is always between a single-variable node and a factor node, so it contains just the single variable. All the messages are as we defined in loopy-BP over cluster graphs. The single-variable nodes don't own any factor, so they just multiply incoming messages. They also don't need to sum anything since the sepset is equal to the variable.

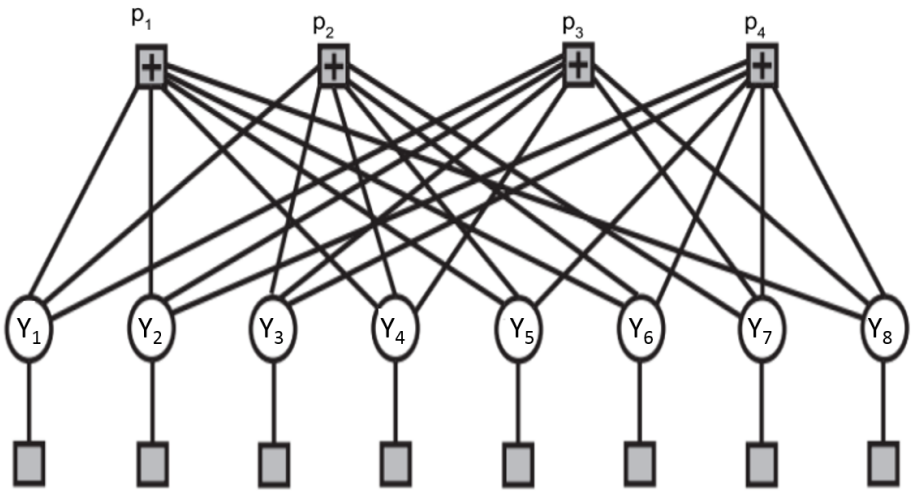


Figure 1: An example for a *Factor Graph* representing the LDPC error correction. Notice that each parity-check factor ϕ_{P_j} is connected to a different subset of codewords $\{Y_j\}$

2 The Tasks

The ultimate goal of this assignment is to decode the original message from the noisy one. You are given an implementation of a Factor class, implementing different operations (e.g. factor product). You are also given a *partial* implementation of a FactorGraph class, which implements different inference operations. You need to complete the implementation.

- It is recommended to read the instructions for all sections before starting to implement.
- Implement your code in factor_graph.py and do_pa2.py (see comments there). factors.py can be left unchanged (unless you really want to change it).
- The problems may require substantial computing time, so start early.

1. [20 points] **Warmup**
   (a) What is the general expression for the joint probability defined by our MN (like the one in Fig. 1)?
   (b) What are the different factors and their parameters definition?
   (c) Formally (as a mathematical expression) define the message decoding task.
   (d) If the probability for an error in **each bit** is 5%, what is the chance of receiving a correct 128-bit message $\widetilde{Y}$?
2. [10 points] Implement code that, given an arbitrary parity check matrix H and a noisy codeword $\widetilde{Y}$, constructs a corresponding factor graph (FactorGraph object). The parity check factors should evaluate to 1 if an even number of adjacent bits are active (equal 1), and 0 otherwise. For a given H matrix, define a small test case, anpad verify that your graphical model assigns zero probability to invalid codewords. What values did you get?

3. Implement *Loopy Belief Propagation* for the factor graphs you generate in (b). Loopy BP on a factor graph is similar to the algorithm we saw for cluster graphs. The algorithm runs on the factor graph itself (no need to construct another graph), including both node types. Use parallel message update schedule (send all messages in each iteration using the BP equations, based on the messages from the **previous** iteration). Initialize by setting all variable-to-factor messages to be constant.

4. [20 points] Load the $N = 128$-bit LDPC code provided in ***ldpc36-128.mat***. To evaluate decoding performance, we assume that the all-zeros codeword Y is sent, which always satisfies any set of parity checks. Using the **rand** method, simulate the output $\widetilde{Y}$ of a binary symmetric channel: each transmitted bit is flipped to its complement with error probability $\epsilon = 0.05$. Define unary factors for each variable node Y_i which equal $1 - \epsilon$ if that bit equals the "received" bit at the channel output, and ϵ otherwise. Run your loopy belief propagation code for 50 iterations. After the final iteration, plot the estimated posterior probability (conditioned on the received, noisy message) that each codeword bit **equals one**. If we decode by setting each bit to the maximum of its corresponding marginal, would we find the right codeword?

5. [20 points] Repeat the experiment from part (c) for 10 random channel noise realizations with error probability $\epsilon = 0.06$. For each trial, run loopy-BP for 50 iterations. After each iteration, estimate the codeword by taking the maximum of each bit's marginal distribution, and evaluate the Hamming distance (number of differing bits) between the estimated and true (all-zeros) codeword. On a single plot, display 10 curves showing Hamming distance versus iteration for each trial. Is BP a reliable decoding algorithm?

6. [10 points] Repeat part (d) with two higher error probabilities, $\epsilon = 0.08$ and $\epsilon = 0.10$. Discuss any qualitative differences in the behavior of the loopy BP decoder.

7. [20 points] Load the $N = 1600$-bit LDPC code provided in ***ldpc36-1600.mat***. Start by converting a 40×40 binary image to a 1600-bit message vector; you may use the logo image we provide, or create your own. Encode the message using the provided generator matrix G, and add noise with error probability $\epsilon = 0.06$ (flip each bit with that probability). Plot first the noisy input image and then images showing the output of the sum-product decoder after 0, 1, 2, 3, 5, 10, 20, and 30 iterations.