Coding, Sudoku, Sinkhorn . . . and Back?

Todd K. Moon

Electrical and Computer Engineering Dept.
Utah State University
Communication occurs in the presence of signal distortions: *noise*.

How to make communications more reliable without increasing energy?
Communication occurs in the presence of signal distortions: \textit{noise}.

How to make communications more reliable without increasing energy?

Error Correction Coding

Message $m$  
Encoder  
Codeword $c$  
Channel  
Noise $n$  
Received data $r$  
Decoder  
Message estimate $\hat{m}$
ECC Example

- ECC works by adding some kind of redundancy to a code.
- Classic (and almost easiest example): **Hamming codes**
  
  ▶ Generator matrix: $G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$
  
  ▶ Message: $m = \begin{bmatrix} m_0 \\ m_1 \\ m_2 \\ m_3 \end{bmatrix}$
  
  ▶ Encoding: $c = mG$
  
  ▶ Example:

  $m = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$.

  $c = mG = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$
Decoding? More complicated, but depends upon the Parity check matrix

\[ H = \begin{bmatrix}
  1 & 0 & 1 & 1 & 1 & 0 & 0 \\
  1 & 1 & 1 & 0 & 0 & 1 & 0 \\
  0 & 1 & 1 & 1 & 0 & 0 & 1
\end{bmatrix} \]

Note that \( GH^T = 0 \).

We have \( rH^T = 0 \) if and only if \( r \) is a codeword.

All of the “parities” must check.

\[
\begin{align*}
  r_0 + r_2 + r_3 + r_4 &= 0 \\
  r_0 + r_1 + r_2 + r_5 &= 0 \\
  r_1 + r_2 + r_3 + r_6 &= 0
\end{align*}
\]
Another Representation of a Code

- Form a graph using the parity check matrix as the incidence matrix of the graph.

\[ H = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \]

This is called the Tanner graph of the code.
- Send information to checks, then back to bits, iteratively.
- The information sent is actually *probability vectors* sent in a belief propagation algorithm.
Belief Propagation

- Message from variable node to function node:
  \[ \mu_{x \rightarrow \alpha}(x) = \prod_{\gamma \in \text{Nb}(x) \setminus \alpha} \mu_{\gamma \rightarrow x}(x) \]

- Message from function node to variable node:
  \[ \mu_{\alpha \rightarrow x}(x) = \sum_{\sim \{x\}} \left( \alpha(x, \text{Nb}(\alpha)) \prod_{y \in \text{Nb}(\alpha) \setminus x} \mu_{y \rightarrow \alpha}(x) \right) \]
Low Density Parity Check Codes

- Defined by a parity check matrix which is *sparse*.
- Decoded by a message passing, or belief propagation (BP) algorithm.
- Very strong codes. The best codes in the world; designed to within 0.00045 dB of the theoretical best.
Some Questions

- Q: How long does the decoding algorithm take?
  - A: It could be hundreds of iterations.
- Q: Can we speed it up?
  - A: Hmm ...
- Q: Can we use this same BP algorithm on other codes that do not have a sparse parity check matrix?
  - A: No! Unfortunately, there are too many cycles in the graph. This leads to biases in the bit estimates which destroy the utility of the decoder.
Illustration of General Linear Block Decoding with BP

![Graph of Probability of bit error vs. $E_b/N_0$ (dB)](image)

- **Iterative decoded**
- **Uncoded**
- **Hard-decision decoded**

![Graph of Probability of bit error vs. Matrix weight](image)
Why do we care?

- There are many other codes.
- Some public key cryptosystems are built upon error correction codes (McEliece)
- If we could do a better job with BP decoding, we might make progress against these.
- Q: Can we find a decoding algorithm that would be
  - Faster
  - Stronger (able to work with denser matrices)

Hold that thought ...
Sudoku

Fill in the blanks, with unique digits 0 — 9 on each row, column, and $3 \times 3$ block:

```
  4 9 6 7
  7 6 9 3
  1 7 4 8
  2 1 6 1
  1 4 3 2
  6 2 9 4
```

- Discrete constraint satisfaction problem (typical of other hard problems)
- NP-complete
Solving Sudoku — Conventional

- Eliminate obvious things
- Get to a point where a decision tree may be necessary: If this is true, what does this mean about other choices ...
- Backtracking may be necessary.
- Can be solved using a computer programmed to search on trees.
Tanner Graph for Sudoku

Constraint functions: each connected cell $S_i$ must be unique.
Belief Propagation for Sudoku

- Represent the contents of each cell as a **probability vector**:

  \[ p_1 = [P(S_1 = 1) \ P(S_1 = 2) \ \cdots \ P(S_1 = 9)] , \text{ etc.} \]

  For example,

  \[ p_4 = [0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0] \]

  For unknown cells, initialize with uniform over possible values:

  \[ p_1 = \frac{1}{3} [0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0] \]

- Send probability vectors around the Tanner graph using the same general belief propagation rule (with suitably modified constraint functions).
Let $C_m$ denote the event that the $m$th constraint is satisfied. The message that constraint node $C_m$ sends to cell $S_n$ is $r_{mn}(x) = P(C_m|S_n = x)$. The message that cell node $S_n$ sends to constraint node $C_m$ is

\[
q_{mn}(x) = P(S_n = x|\text{all constraints involving } S_n \text{ except } C_m \text{ satisfied})
\]

\[= P(S_n = x|\{C_{m'}, m' \in \mathcal{M}_{n,m}\}).\]

Then:

- $r_{mn}(x) = \sum_{\{x_{n'}, n' \in N_{m,n}\} : \prod_{l \in N_{m,n}} q_{ml}(x_l)} \prod_{l \in N_{m,n}} q_{ml}(x_l)$.
- $\{x, x_{n'}\}$ all unique

- $q_{mn}(x) = \alpha_{mn} P(S_n = x) \prod_{m' \in \mathcal{M}_{n,m}} r_{m'n}(x)$. 
Does it Work?

- Somewhat slow: (Has to sum over permutations)
- Doesn’t solve all puzzles: Too many short cycles in Tanner graph!

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Puzzles</td>
<td>10</td>
<td>10</td>
<td>25</td>
<td>23</td>
</tr>
<tr>
<td>No. BP solved</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Av. No. BP Iterations</td>
<td>5.8</td>
<td>7</td>
<td>6.3</td>
<td>7.5</td>
</tr>
</tbody>
</table>
Sinkhorn Balancing

- Definition: A positive matrix whose columns each sum to 1 and whose rows each sum to 1 is called a **doubly stochastic matrix**.
- Problem: Given a matrix $A$ with nonnegative elements, find a “related” doubly stochastic matrix $\tilde{A}$.
- Solution:

---

**Sinkhorn Balancing**

**Repeat until Convergence:**
- Normalize each column of $A$ by its column sum
- Normalize each row of $A$ by its row sum
Sinkhorn Balancing

An instance of **Projection on Convex Sets**

Set of Row Stochastic Matrices

Set of Column Stochastic Matrices

$\tilde{A}$
Sinkhorn and Sudoku

- Take one of the constraints of the Sudoku matrix, e.g., the first row.
- Represent each cell in the constraint as a $1 \times 9$ probability vector.
- Stack up these probability vectors to produce a $9 \times 9$ matrix $M$.

$$M = \begin{bmatrix}
p_1 \\
p_2 \\
p_3 \\
\vdots \\
p_9
\end{bmatrix}$$

Call this the **constraint probability matrix**.

- Contemplate the nature of this matrix if the probability vectors represented the true solution:
  - Each row would sum to 1 (probability vectors)
  - Each column would sum to 1 (there would only be one 1 in each column if all digits are unique).

At the solution, $M$ is doubly stochastic!
Sinkhorn and Sudoku

- Initially, the constraint probability matrix may not be doubly stochastic.
- So, we do a little “Math by wish fulfillment”: we make it be what we want it to be.
- Do this for each constraint in turn.

**Sinkhorn Sudoku Solution (SSS)**

**Initialization:** Set initial probability vectors \( p_1, \ldots, p_{81} \)

**Repeat:**
- For each constraint \( m \in \{1, 2, \ldots, 27\} \)
  - Form the constraint probability matrix \( M_m \)
  - Sinkhorn balance to obtain \( \tilde{M}_m \).
  - Extract the probability vectors \( p_i \) from \( \tilde{M}_m \)
- End for \( m \)
- Determine the most probable contents \( S_i \) from \( p_i \)
- If all constraints are satisfied, **break** with success
- If too many iterations, **break** with failure

End Repeat
Does it Work?

Yes!

- For every difficult puzzle in a puzzle book, it converges.
- May take many iterations.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Puzzles</td>
<td>10</td>
<td>10</td>
<td>25</td>
<td>23</td>
</tr>
<tr>
<td>No. SSS solved</td>
<td>10</td>
<td>10</td>
<td>25</td>
<td>23</td>
</tr>
<tr>
<td>Av. No. SSS Iterations</td>
<td>35.5</td>
<td>56.9</td>
<td>169.4</td>
<td>177.4</td>
</tr>
<tr>
<td>No. BP solved</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Av. No. BP Iterations</td>
<td>5.8</td>
<td>7</td>
<td>6.3</td>
<td>7.5</td>
</tr>
</tbody>
</table>

- May exhibit nonuniform convergence.
- Tends to have long plateaus.
Convergence Examples

Light & Easy

Moderate

Demanding

Very Challenging
Random Sudoku

SSS works for this too!
Some Results on Convergence

- Kullback-Leibler: \( D(P \parallel Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)} \).

- As applied to the puzzle: \( P \) is a probability set associated with a puzzle. \( D(P \parallel Q^k) = \sum_{m=1}^{3N} D(P_m \parallel Q^k_m) \).
A Convergence result

- Theorem: Let $\mathcal{P}$ denote the probability constraint matrices for the solution of a puzzle with a unique solution. Then:

$$D(\mathcal{P}||\mathcal{Q}^{[0]}) < \infty$$

$$D(\mathcal{P}||\mathcal{Q}^{[k+1]}) \leq D(\mathcal{P}||\mathcal{Q}^{[k]})$$

That is, every iteration is no farther from the solution.

- Proof: Essentially an application of the “information inequality” $\log x \leq x - 1$. 
Getting Stuck

\[
\begin{array}{|c|c|c|}
\hline
3 & 2 & 9 \\
\hline
5 & 6 & 1 \\
\hline
8 & 5 & 1 \\
\hline
9 & 4 & 8 \\
\hline
7 & 2 & 6 \\
\hline
\end{array}
\]
Other Information-Theoretic Connections

Goal: Preserve as much information from one step to the other (while maintaining doubly-stochastic nature):

\[
\begin{align*}
\text{minimize} & \quad \sum_{i,j} q_{ij}^{[k+1]} \log \frac{q_{ij}^{[k+1]}}{q_{ij}^{[k]}} \\
\text{subject to} & \quad \sum_j q_{ij}^{[k+1]} = 1, \forall i = 1, 2, \ldots, N \\
& \quad \sum_i q_{ij}^{[k+1]} = 1, \forall j = 1, 2, \ldots, N \\
& \quad q_{ij}^{[k+1]} \geq 0, \forall i, j = 1, 2, \ldots, N.
\end{align*}
\]

It can be shown that the solution of this is obtained by Sinkhorn balancing.
This, SSS is an optimal information-preserving algorithm.
It can be shown furthermore (using the log-sum inequality) that the solution found minimizes the KL distance from the original probability matrices.
What does it mean?

- No search of trees — suspended judgements are not represented by nodes in a tree, but by probabilities
- As evidence accumulates about a decision, the probabilities coalesce toward the correct answer.
- Proof of convergence?
- Proof of universality?
Belief propagation works well for LDPC codes
But not too well on Sudoku (too many cycles).
Sinkhorn works well on Sudoku (even though it has lots of cycles)
Can we develop a Sinkhorn-like decoder for LDPC which:
  - Works well in the presence of lots of cycles
  - Converges faster?
Other Applications

- Can we use Sinkhorn balancing on other discrete constraint satisfaction problems?
- What are practical limits on problem sizes that can be dealt with?
- Can the Sinkhorn balancing computation be accelerated?