

Sudoku became a world-wide craze in November 2004, when the first puzzle appeared in the pages of *The Times*. Where did it come from? Here we want to explore the hypothesis that Sudoku was originally an error correcting code (used by aliens? by the Incas? by the army?)

## Definition

A **Sudoku scheme** of order  $\ell$  is a square of side  $\ell^2$  (thus including  $\ell^4$  cells) divided in  $\ell^2$  square blocks, each of side  $\ell$ . Below is an example of partially filled grill of order  $\ell = 3$ .

	1	2			9	6		
		7	3	5			9	
8				4			3	7
4		3	2				1	
	8		7	6		9		
9					3	8		4
	6				5	7		1
7		9	6		1			
	5			2			6	9

A **Sudoku solution** is an assignment of integers in  $\{1, 2, \dots, \ell^2\}$  ('numbers') to the cells in the grid that satisfies three conditions: (i) For each row, the numbers appearing in that row must be distinct; (ii) For each column, the numbers appearing in that column must be distinct; (iii) For each of the  $\ell^2$ ,  $\ell \times \ell$  squares, the numbers appearing in that square are distinct. Notice that any set of  $\ell^2$  distinct symbols could play the role of the numbers in  $\{1, \dots, \ell^2\}$ .

We shall call Sudoku solution **codeword**. The set of all solutions will be called **codebook** (sometimes, for short, the **code**) and denoted by  $\mathfrak{C}$ , or  $\mathfrak{C}(\ell)$  whenever it will be necessary to specify its order.

## Rate

Here is how Sudoku can be used to convey information. First write your information as a (very long!) binary string. Then chop the string into blocks of length  $L = \log_2 |\mathfrak{C}|$  (for simplicity, think of  $L$  as an integer.) Each of the blocks consists in a number  $z$  between 0 and  $|\mathfrak{C}| - 1 = 2^L - 1$  written in binary notation. This is encoded as the  $L$ -th Sudoku solution (say, in lexicographic order) and then transmitted. The

The **rate** of this code is the number of information bits conveyed per 'channel use.' This is a bit ambiguous. Argue that a good mathematical definition would be

$$R(\ell) = \frac{\log_2 |\mathfrak{C}(\ell)|}{\ell^4 \log_2 \ell^2}. \quad (1)$$

The number of Sudoku solutions of order 2 is  $|\mathfrak{C}(2)| = 288$  thus yielding  $R(2) \approx 0.255310$ . A counting tour de force lead recently to the result

$$|\mathfrak{C}(3)| = 6,670,903,752,021,072,936,960$$

which corresponds to  $R(3) \approx 0.282354$ .

Note that  $|\mathfrak{C}(\ell)|/(\ell^2)^{\ell^4}$  is the probability that filling the cells with iid uniformly random numbers, one gets a solution. A naive guess consists in approximating this with the product of probabilities that each constraint is satisfied. Show that this yields

$$R_{\text{naive}}(\ell) = 1 - \frac{3}{2 \log \ell} + O(\ell^{-2}). \quad (2)$$

It is reasonable to conjecture that  $R(\ell) \rightarrow 1$  as  $\ell \rightarrow \infty$ .

## Errors

A codeword transmitted through a noisy channel can be corrupted in two ways. The first consists in **erasures** and is widespread in magazines:

1	2	3	4	→	1		3	4
3	4	1	2		3	4	1	2
2	1	4	3				4	3
4	3	2	1		4	3	2	

A probabilistic model for this channel would be the following: each entry in the grid is erased independently with probability  $\epsilon$ .

A more severe type of noise would consist in ‘flipping’ the numbers.

1	2	3	4	→	1	3	3	4
3	4	1	2		3	4	1	2
2	1	4	3		2	1	4	3
4	3	2	1		4	3	2	4

Again, within a probabilistic setting we might assume that each entry is changed independently with probability  $\epsilon$  in a uniformly random element in  $\{1, \dots, \ell^2\}$ .

In what sense is the second type of noise ‘worse’ (for the same value of  $\epsilon$ ) than the first one? The formalization is provided by the notion of **physical degradation**.

## Distance

Given two Sudoku solutions  $x, x'$ , their **(Hamming) distance** is the number of positions in which they differ. The code **minimum distance** is the minimum distance between any two distinct codewords:

$$d_{\min}(\ell) = \min\{d(x, x') : x, x' \in \mathfrak{C}(\ell)\}. \quad (3)$$

Show that  $d_{\min}(\ell) = 4$  for any  $\ell$ . I must confess I know a proof only for  $\ell = 2, 3$  (look at the grid below), but am pretty sure it is true in general. Can you find a proof for  $\ell \geq 4$ ?

Suppose somebody erases entries maliciously from your Sudoku solution. What is the maximum number of erasures such that you can be sure to reconstruct the original solution? What about flips?

9	2	6	5	7	1	4	8	3
3	5	1	4	8	6	2	7	9
8	7	4	9	2	3	5	1	6
5	8	2	3	6	7	1	9	4
1	4	9	2	5	8	3	6	7
7	6	3	1			8	2	5
2	3	8	7			6	5	1
6	1	7	8	3	5	9	4	2
4	9	5	6	1	2	7	3	8

## Decoding

Any algorithm for solving Sudoku can be used as a decoder: take the output of the noisy channel, and apply the algorithm to get a complete valid solution (a codeword.) I am not a good Sudoku solver and normally use the following (pretty dumb) algorithm, that works for correcting erasures: if there is a row, a column or a square with only one empty cell, I fill it. I repeat and hope for good.

In order to formalize this, let me introduce some notations. I let  $V$  denote the set of  $\ell^4$  cells and write  $i, j, \dots \in V$  to indicate specific cells. The set of line plus rows plus squares will be indicated by  $C$  (for ‘constraints’), and I’ll write  $a, b, \dots \in C$  for its elements. We have  $|C| = 3\ell^2$ . The set of cells appearing in a given constraint  $a$  is denoted by  $\partial a$ . Vice-versa, the set of constraints that involve cell  $i$  is  $\partial i$ .

---

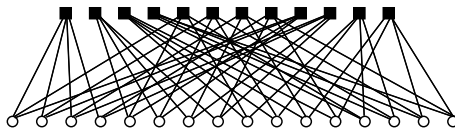
### ITERATIVE SCANNING (A partially filled grid $G$ )

---

- 1: Set **flag** = **false**
  - 2: For  $a \in C$
  - 3:     If  $\partial a$  include a unique empty cell
  - 4:         Fill it
  - 5:     Set **flag** = **true**
  - 6: End for;
  - 7: If **flag** = **true**
  - 8:     Goto 1;
  - 9: Else
  - 10:     Return current grid;
- 

Can you invent a similar algorithm to correct flips?

Notice in passing that the notation introduced above naturally leads to a bipartite graph, called the **factor graph** of the problem. This has vertices sets  $V$ , and  $C$ , and an edge  $(i, a)$  whenever  $i \in \partial a$  or (equivalently)  $a \in \partial i$ . Below is the factor graph for  $\ell = 2$  (squares correspond to constraints, circles to variables.)



## Error probability

It would be nice to have an idea of how well the iterative algorithm does. We need to define some performance measure. Two well known such measures are: (i) The **block error probability**  $P_B$ . This is the probability that the original grid is not reconstructed, or it is reconstructed incorrectly. (ii) The **symbol (or bit) error probability**  $P_b$ . This is the probability that a symbol is not reconstructed correctly, averaged over the symbol position. In both cases we assume a uniformly random codeword (Sudoku solution) is transmitted.

In order to formalize these definitions, we need some more notations. We let  $x = \{x_i : i \in V\}$  be a uniformly random Sudoku solution (this is a vector with  $\ell^4$  entries in  $[\ell^2]$ .) The output of the noisy channel will be denoted by  $y = \{y_i : i \in V\}$  (this is a vector with entries  $y_i \in \{*, 1, \dots, \ell^2\}$ .) Finally, the output of the iterative algorithm on input  $y$  will be  $\hat{x}(y) = \{\hat{x}_i(y) : i \in V\}$ . We have the definitions

$$P_B(\epsilon, \ell) \equiv \mathbb{P}\{\hat{x}(y) \neq x\}, \quad (4)$$

$$P_b(\epsilon, \ell) \equiv \frac{1}{|V|} \sum_{i \in V} \mathbb{P}\{\hat{x}_i(y) \neq x_i\}. \quad (5)$$

How does  $P_B(\epsilon, \ell)$  behave for small  $\epsilon$ ?

## WORK !!!!!

Implement the iterative decoding algorithm described above for order 3 Sudoku tables. Choose one solution and run a Monte Carlo simulation to estimate the resulting symbol and block error probability.

I expect to receive

1. The Sudoku solution you used.
2. A print-out of the code.
3. A plot of error probability versus  $\epsilon$ .