2D Puzzle Visualizations of Boolean Formulae

Eric Holloway

DOI: 10.33014/issn.2640-5652.3.2.holloway.1

Background

In the comparison between human and computational intelligence, often times the comparison is not straightforward because humans can possess domain knowledge inaccessible to the program they are competing with. To provide a level playing field, it is helpful to have humans and computers compete in a domain where both start with equal domain knowledge, and the domain is well understood.

One such domain is boolean formulae. These formulae are sets of boolean variables that are combined with the AND ($\wedge$), OR ($\vee$), and NOT ($\neg$) logical operators. Each boolean formula maps to a specific truth table. A truth table is the result of trying all the possible permutations of value assignments applied to the formula and the resulting truth values from the formula. These formulae are well understood, and can be selected so the human and algorithm have no special advantage over the other in terms of domain knowledge, or even the algorithm has an advantage in order to make the success of the human even more significant.

The reason why the domain of boolean formulae is a good domain for comparing human and computational intelligence is because there are many problems dealing with these formulae that are NP complete and harder. Two such examples are:

1. determining if a particular formula can evaluate to true, an NP complete problem
2. inverting a truth table to derive the shortest boolean formula that represents the table, an NP hard problem

If humans can outperform computers when faced with an NP complete problem, then insofar as NP is not the same as P, we can determine that humans at least outperform deterministic Turing machines, and consequently human intelligence must be on the level of nondeterministic Turing machines.

However, the problem with this domain as far as humans are concerned is that truth tables are hard to visually process. We want to come up with a test that humans don't need a PhD to understand. Something visual is ideal, as visual representations can be understood by a human with a lower level of expertise. In this paper, I present a 2D visualization of boolean formulae that allow humans to visually comprehend truth tables of up to 12 boolean variables.

I begin by explaining how to represent bitstrings (series of 1s and 0s) with a boolean formulae, due to the straightforward observation that every bitstring can be represented by a truth table.

Bitstrings as Boolean Formulae

Bitstrings b can be expressed as logic expressions with $\lceil \log_2(|b|) \rceil$ binary variables. For example, with four logic variables x_1,x_2,x_3, and x_4 we can describe all bitstrings of length 16 with logic expressions.

To generate a bitstring from a particular logic expression, we evaluate the expression for all variable assignments, and order the results lexicographically.

Here is an example with 2 variables, x_1 and x_2, and the boolean formula $x_1 \vee \neg x_2$.

1. Evaluate the expression for all assignments.

x1	x2	$x_1 \vee \neg x_2$
0	0	1
0	1	0
1	0	1
1	1	1

2. Order the results lexicographically.

   The lexicographic order of the assignments is:

   (a) 0 0
   (b) 0 1
   (c) 1 0
   (d) 1 1

   So, taking the results of the boolean formula in this order, the bitstring is: 1, 0, 1, 1.

If we imagine the above process as a generating function, we can label the generating function as G. If we apply G to a boolean formula, then it generates the corresponding bitstring. From our previous example, $G(x_1 \vee \neg x_2) = 1011$.

A four variable example,

$$\begin{aligned} &G(\\ &\quad (x_1 \wedge x_2 \wedge \neg x_3) \\ &\quad \vee (x_1 \wedge x_3 \wedge \neg x_4) \\ &\quad \vee (\neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge x_4) \\ &\ ) = 0001010110010000. \end{aligned}$$

Next, I talk about how to generate a boolean formula randomly. This is important, because per our original quest to find a problem domain to compare human and computer performance, we need a problem domain that is larger than can fit within a physical computer. With 12 boolean variables, there are $2^{2^{12}}$ possible unique formulae, which is more than can fit within the physical universe. That being said, we will only deal with a subset of these formulae, since most are random, and are not useful for the sort of puzzle that can distinguish whether human intelligence transcends computational intelligence.

Randomly Sampling Disjunctive Normal Form (DNF) Formulae

A DNF formula is an OR of AND terms, such as $(v_1 \wedge \neg v_2) \vee (\neg v_1 \wedge v_2)$ to represent $v_1 \oplus v_2$.

The puzzles are generated by randomly sampling a DNF formula f from formulae with a set number of variables n, terms t, and maximum term size k. The random sampling of DNF formulae is close to uniform, with the exception that false terms are filtered out. Here is how it works.

1. The sampling process is given the number of variables n, maximum literals per term k and the number of terms t.
2. Each term is randomly filled with literals.
3. A term is refilled if its literals result in a contradiction.

Then, a bitstring is generated from f using $G(f)$, as explained earlier.

Noise can be added by randomly flipping a certain number of bits m. The effect of adding m bits of noise is to increase the space of formulae by $2^{|m|}$.

Turning a Bitstring Into a Puzzle

However, a bitstring by itself is not a puzzle to be solved. We would like a puzzle with an easy to represent solution. One such puzzle is to fill in missing bits in the bitstring. To create this sort of puzzle, a set of bits s are removed to be guessed.

Converting the Formula to 2D Image

To make the visualization for the human, the bitstring generated by $G(f)$ is displayed as a 2D image.

The logic variables are split into two groups, and each group's binary values are converted into integers to form the coordinates.

For instance, with four variables we have two groups for the x and y coordinates. I.e. $x = \{v_1, v_2\}$ and $y = \{v_3, v_4\}$. If we assign the variables as: $v_1 = 0, v_2 = 1, v_3 = 1, v_4 = 0$, we see that x and y groups can represent binary numbers. I.e. $x = \{0, 1\}$ and $y = \{1, 0\}$. These binary numbers are then converted to integers, $x = 2$ and $y = 1$.

Each point in the boolean space described by the four variables has an assignment of 0 or 1 from f. Converting the boolean space to a 2D graph as previously described, and using f assignments to fill in the graph where 0 is black and 1 is white, turns the truth table into an image.

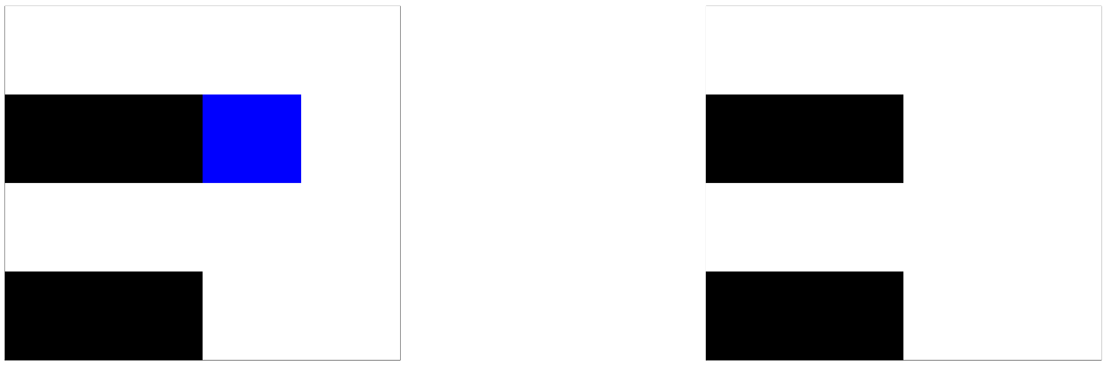
As an example, set $f = v_2 \vee v_3$. The logic table and coordinates are the following.

v_1	v_2	x	v_3	v_4	y	$v_2 \vee v_3$
0	0	0	0	0	0	0
1	0	1	0	0	0	0
0	1	2	0	0	0	1
1	1	3	0	0	0	1
0	0	0	1	0	1	1
1	0	1	1	0	1	1
0	1	2	1	0	1	1
1	1	3	1	0	1	1
0	0	0	0	1	2	0
1	0	1	0	1	2	0
0	1	2	0	1	2	1
1	1	3	0	1	2	1
0	0	0	1	1	3	1
1	0	1	1	1	3	1
0	1	2	1	1	3	1
1	1	3	1	1	3	1

Figure 1a and Figure 1b are a realization of this logic table converted into a 2D image. In the left pane, one bit has been removed and replaced with blue, and this is the bit to be guessed. The right pane shows the answer.

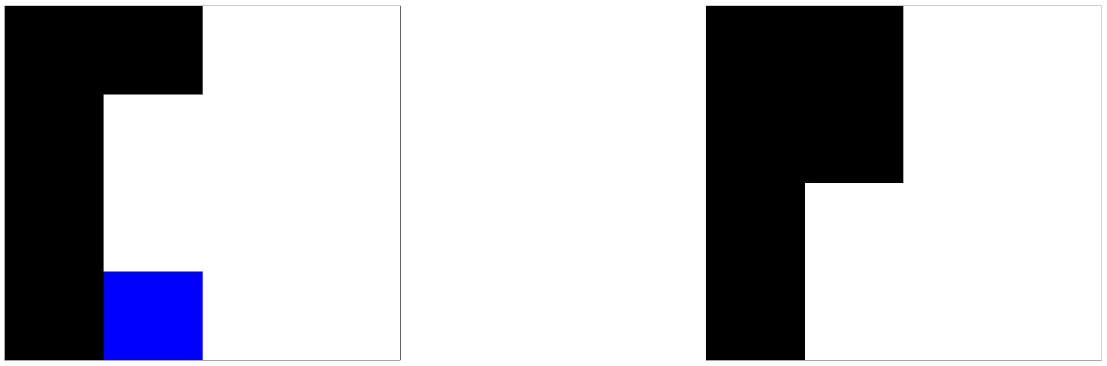
Figure 2a and Figure 2b are another puzzle of the same size, but this time with one bit of noise addded. The generation function for this puzzle is $G(v_1 \vee v_2)$.

Figure 3a, Figure 3b, Figure 4a, and Figure 4b show the prediction puzzles generated by more complex DNF formulae, with and without noise added.



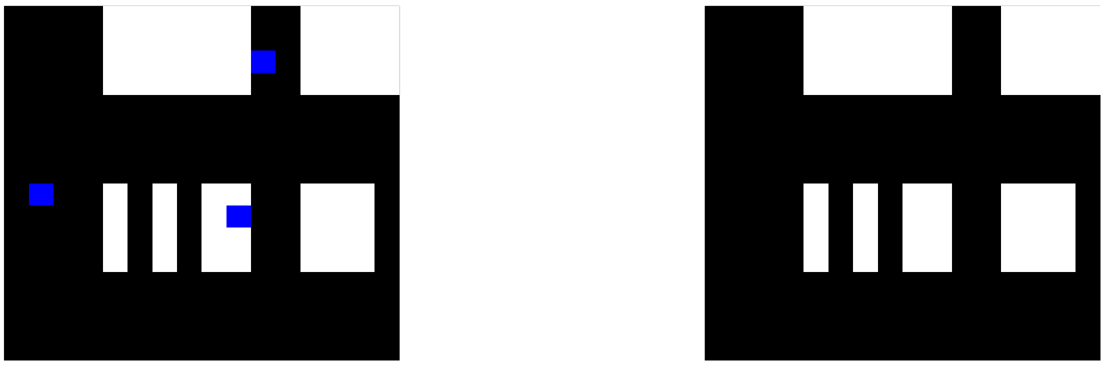
(a) Guess the blue bit. (b) Answer to puzzle.

Figure 1: Puzzle with 4 variables.



(a) Guess the blue bit. (b) Answer to puzzle, no noise.

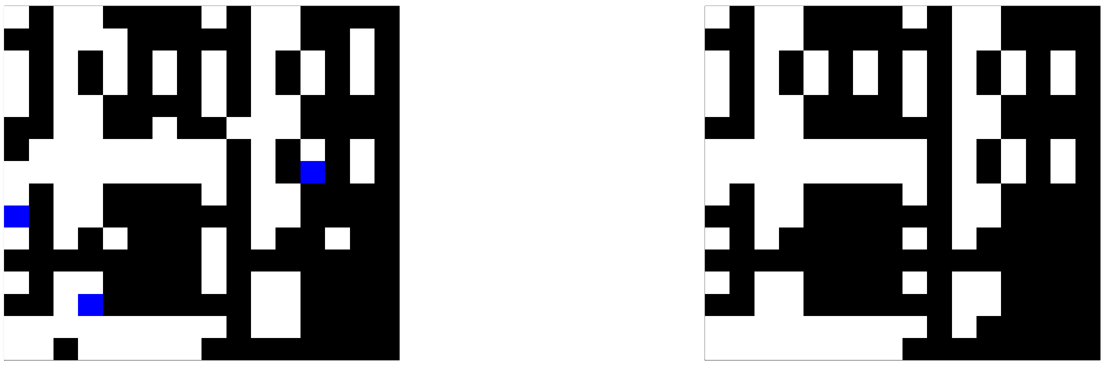
Figure 2: Puzzle with 4 variables and 1 bit of noise.



(a) Guess the blue bits.

(b) Answer to puzzle.

Figure 3: Puzzle with 8 variables.



(a) Guess the blue bits.

(b) Answer to puzzle, no noise.

Figure 4: Puzzle with 8 variables and 10 bits of noise.