$(\neg f_1 \vee f_3 \vee f_4 \vee f_5) \wedge (\neg f_3 \vee f_4 \vee f_5) \wedge (\neg f_1 \vee f_3 \vee \neg f_4) \wedge (f_1 \vee f_2)$
$\wedge (f_1 \vee \neg f_2) \wedge (\neg f_1 \vee \neg f_5) \wedge (\neg f_3 \vee \neg f_4 \vee f_5)$

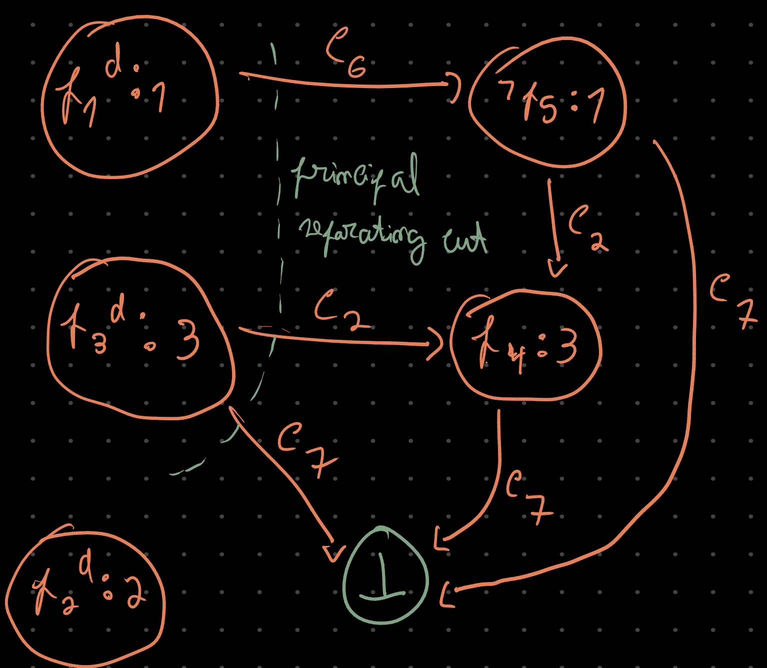
$\downarrow$ Set notation

$$X = \{\underbrace{\{\neg f_1, f_3, f_4, f_5\}}_{c_1}, \underbrace{\{\neg f_3, f_4, f_5\}}_{c_2}, \underbrace{\{\neg f_1, f_3, \neg f_4\}}_{c_3}, \underbrace{\{f_1, f_2\}}_{c_4}, \underbrace{\{f_1, \neg f_2\}}_{c_5}, \underbrace{\{\neg f_1, \neg f_5\}}_{c_6}, \underbrace{\{\neg f_3, \neg f_4, f_5\}}_{c_7}\}$$

1. [ ]
2. $[f_1^d]$ by rule decide
3. $[f_1^d, \neg f_5]$ by rule unit-prop(c_6)
4. $[f_1^d, \neg f_5, f_2^d]$ by rule decide
5. $[f_1^d, \neg f_5, f_2^d, f_3^d]$ by rule decide
6. $[f_1^d, \neg f_5, f_2^d, f_3^d, f_4]$ by rule unit-prop(c_2)

Conflict with c_7

$$C = (\underset{d=1}{\neg f_1} \vee \underset{d=3}{\neg f_3}) \rightarrow c_8$$



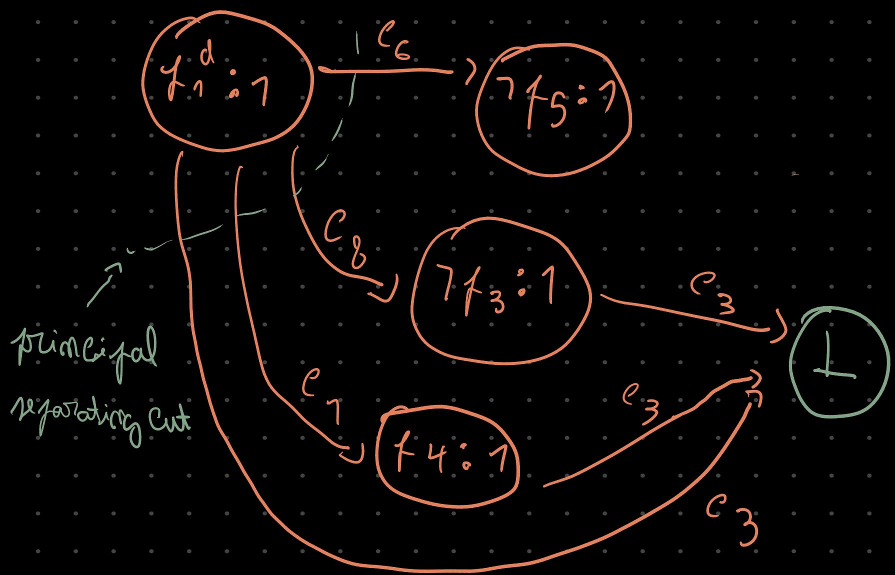
The second greatest decision level in C is 1, so we delete from w every literal with a decision level greater than 1.

7. $[p_1^d, \neg p_5]$ by rule backjump

8. $[p_1^d, \neg p_5, \neg p_3]$ by rule unit-prop (c_8)

9. $[p_1^d, \neg p_5, \neg p_3, p_4]$ by rule unit-prop (c_7)

Conflict with c_3



$d = 1$

$C = \neg p_1 \to c_9$

The second greatest decision level is 0, so we delete all literals from W.

10. [ ] by rule backjump

11. $[\neg p_1]$ by rule unit-prop (c_9)

12. $[\neg p_1, p_2]$ by rule unit-prop (c_4)

Conflict with c_5

13. fail by rule fail

Because there are no more decision literals inside the partial valuation, the formula is UNSAT

Report project 1 LVM 2025/2026

José Marques (105673)
José Lopes (103938)

18 October 2025

1 Encoding - exercise 2

1.1 Variables

Let n be the length of both the secret code and the guesses on the board. for $i \in \{1, 2, ..., n\}$ and $k \in \{0, 1, ..., 9\}$, we define the propositional symbols

p_{ik} - entry at position i has number k

1.2 Constraints

Each position must have at least 1 number

$$\bigvee_{i=0}^{n} \bigvee_{k=1}^{9} p_{ik} \tag{1}$$

Each position must have at most 1 number

$$\bigwedge_{i=0}^{n} \bigwedge_{j=0}^{8} \bigwedge_{k=j+1}^{9} (\neg p_{ij} \vee \neg p_{ik}) \tag{2}$$

1.2.1 Almost Mastermind - exercise 2.1

If there are no black pegs, we negate the numbers at the current guess's position.

Let a_i be the number at the position i of a given guess, where $i = 1, ..., n$

$$\bigwedge_{i=1}^{n} \neg p_{ia_i} \tag{3}$$

Else, let b be the number of black pegs for a given guess. We need to consider all possible groups of b size, without repetition. We get a total of
$B = \frac{n!}{b!(n-b)!}$ sets of b size.

These sets have, at most, n elements, if $b = n$. Each of these sets will contain the possible positions that have the correct numbers. let P_l be such a set, with $l = 1, ..., b$ and NP_l the complement of P_l. e.g. if $n = 4$, $b = 3$ and $P_1 = \{1, 2, 3\}$, then $NP_1 = \{4\}$.

$$\bigvee_{l=1}^{B} \bigwedge_{\substack{c \in P_l \\ q \in NP_l}} p_{ca_c} \wedge \neg p_{qa_c} \tag{4}$$

With this, we are saying, at least one of the possible b positions has the correct numbers in them.

1.2.2 Full Mastermind - exercise 2.2

Let w be the number of white pegs for a given guess. For each combination (i.e. the set P_l with the positions we consider to have the correct number), the number of sets that contain the possible white positions on the guess
$W = \frac{(n-b)!}{w!((n-b)-w)!}$

This is because there's no need to consider positions that we are assuming to be correct for the current combination (stored in P_l)

Let $WP_m, m = 1, ..., W$ be sets of positions on the guess, where the size of each $WP_m = w$ (WP for wrong position).

And so, for each black combination, and for each white combination, we need to consider all permutations of positions ,which can be positions present in WP_m. for example, we can have a 2 in position 1 and a 3 in position 2 that are not in the code, but if those positions are white (that is, in WP_m) 3 can be at position 1 and 2 at position 2.

Then, the number of possible permutations is

$Perm = (n - b)!/((n - b) - w)!$,

and again, let $V_z, z = 1, ..., Perm$ be the sets that have all possible permutations of possible positions where the number indexed by WP_m can be, to match the secret code

And finally we can write the encoding as such:

$$\bigvee_{l=1}^{B} \bigvee_{m=1}^{W} \bigvee_{z=1}^{Perm} (\bigwedge_{\substack{c \in P_l \\ q \in NP_l}} p_{ca_c} \wedge \neg p_{qa_c} \bigwedge_{d \in WP_m} \bigwedge_{e \in V_z} \neg p_{da_d} \wedge p_{ea_d})$$

(5)

for $e \neq d$.

2 Small experiments

We tested independently two variables, number of guesses and size of a guess (which we sometimes call board length). We measure formula building time, time to find a SAT solution a time to find all solutions to a Board.

2.1 Increasing guess number

Increasing the number of guesses increases the formula building time somewhat linearly, as seen in figures (1) (2) and (3). The spikes in time are, as expected, related to the bigger proportion of white pegs in relation to black pegs.

We see the same pattern for boards with length of 9 (figure (4)), but for the same number of guesses, each building is much much bigger.

Figure 1: SAT times of boards with a secret code of length 6

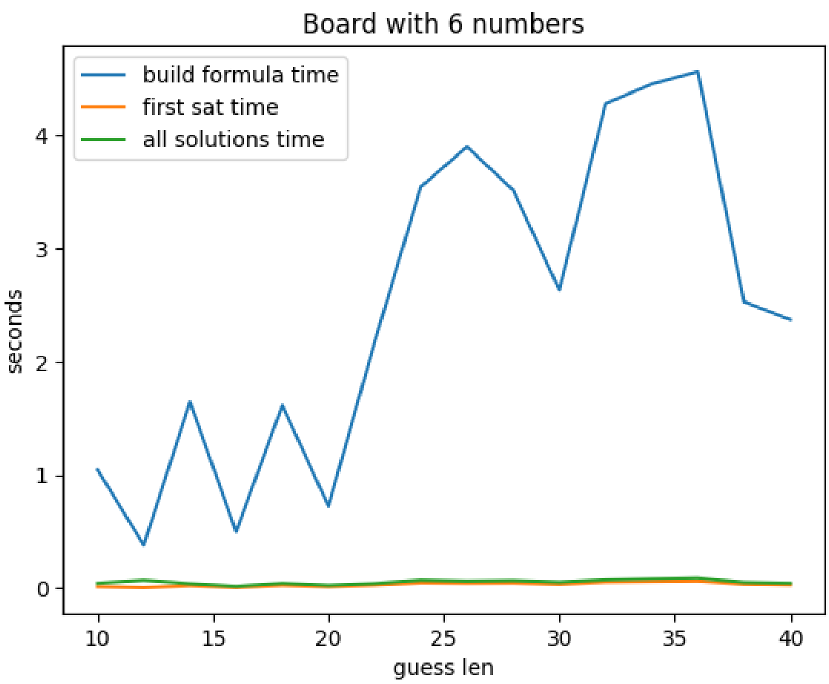


Figure 2: SAT times of boards with a secret code of length 7

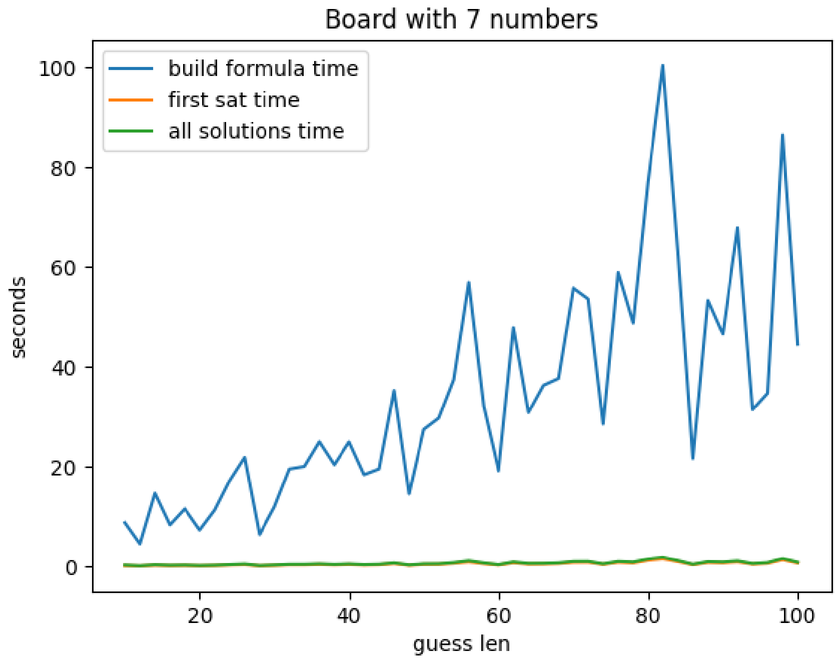
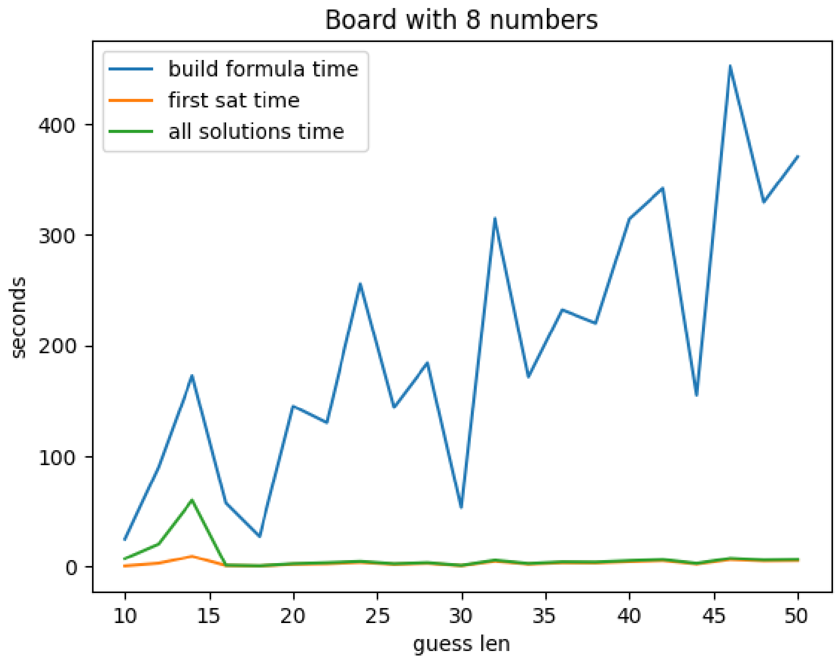


Figure 3: SAT times of boards with a secret code of length 8



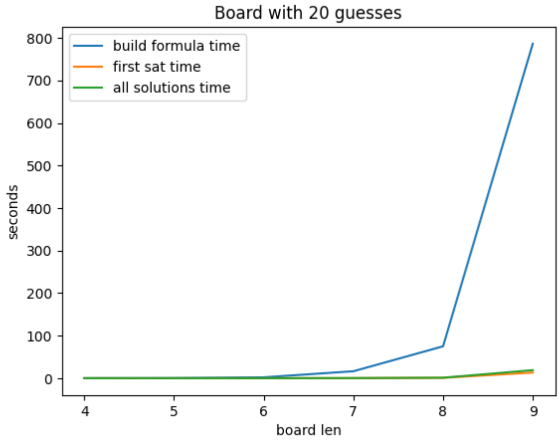
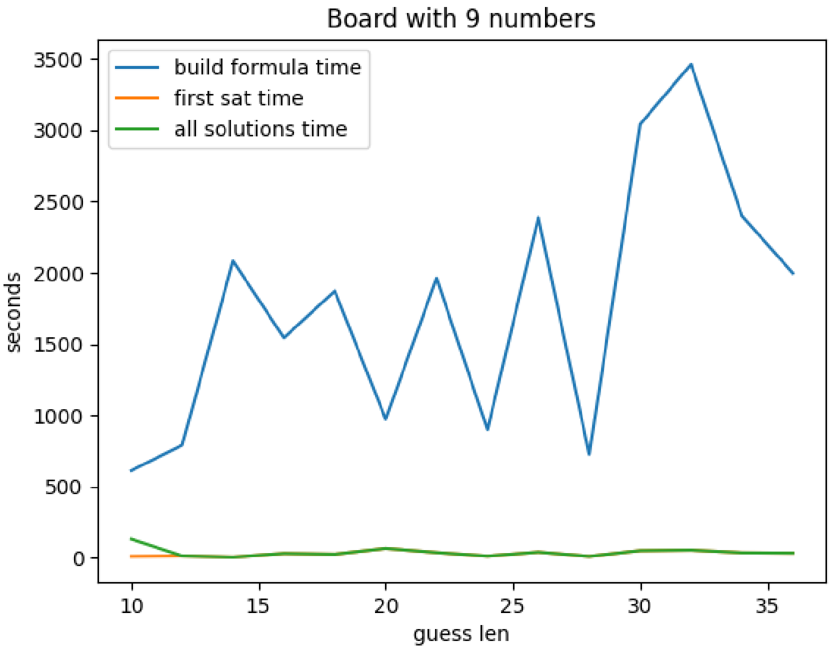


Figure 5: SAT times of boards with increasing code length

Figure 4: SAT times of boards with a secret code of length 9



2.2 Increasing secret code length

If we now increase the length of the secret code, but with the same guess number, we see a sharp rise in the time it takes to construct a formula (figure (5))

This is because we consider all permutations for possible places for the white pegs, for each combination of black pegs. This corresponds to the complexity of our encoding being factorial.

Very interestingly, the SAT solving times are very similar across our experiments, which speaks to the power of the advancements on SAT solvers, and the importance of encoding a problem in the most efficient manner, to speed up the formula building time.