JOHANNES KEPLER
UNIVERSITY LINZ

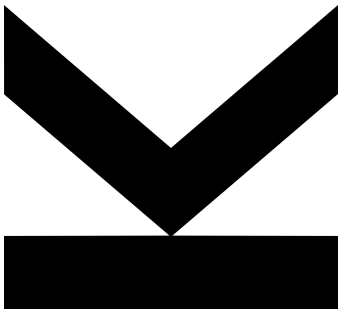
Authors / Eingereicht von
Annika Schmidthaler
David Prinz
Reza Rajabi
Jessica Om
Aleksandar Stojanović
Matriculation number / Matrikelnummer
K12411307
K12412063
K12428264
K12419136
K12411325

Institute / Department / Institut / Abteilung
Institute for Integrated Circuits and Quantum Computing

Lecture / LVA
Computational Complexity

December 23, 2025

Minesweeper



Group Project

JOHANNES KEPLER UNIVERSITY LINZ
Altenbergerstraße 69
4040 Linz, Austria
www.jku.at
DVR 0093696

JOHANNES KEPLER
UNIVERSITY LINZ

Contents

1. Introduction

1.1. Game Description

Minesweeper is a popular puzzle video game. The game features a grid of $x \times y$ tiles. Hidden throughout that field are multiple mines which the player needs to avoid. The game ends in a win if the player manages to clear all fields without detonating a single bomb. ([1])

Each tile within the grid can be either unopened, opened or flagged. Unopened tiles are blank and can be opened. Players are able to flag a cell to denote a possible mine location. Flagged cells are marked with a flag symbol on the grid and still considered as unopened. ([1])

Selecting a cell opens it. An open cell can either be a mine, which immediately ends the game and results in failure, a number that indicates the amount of mines that are horizontally, vertically or diagonally adjacent to it, or blank, in which case all non-mine cells neighbouring it will be revealed. A cell can have up to eight neighbours. ([1])

Figure 1: A game of Minesweeper
Source: [2]

A game of Minesweeper begins by opening a cell. While playing, increasingly more information about the grid becomes known to the player which further aids in deducing the next safe cell to open. Furthermore, the remaining amount of mines is given to the player. The mine count is calculated by subtracting the total number of mines by the number of flagged cells, thus also allowing the mine count to be negative. ([1])

To win at Minesweeper, the player has to clear all non-mine cells without opening a mine. There isn't a score count or time limit, however players' time to finish is being measured. Difficulty can be increased by adding more mines or starting with a larger game field.

After establishing the game rules and restrictions, we define our language **MSWP**: A field is in **MSWP** if there exists a layout of bombs such that the game field is valid, i.e. the numbers in the game field correctly correspond to the amount of neighbouring bombs.

1.2. Motivation

Choosing Minesweeper as this project's topic was appealing for a multitude of reasons. The primary ones of this paper are the wide-spread familiarity with the game and the topic being well-suited to the Computational Complexity lecture at JKU Linz.

Since Minesweeper has been bundled with numerous operating systems, millions of users have become deeply acquainted with its gameplay. Thus this paper should be more accessible to readers of a non-technical background.

To win at Minesweeper, players have to use logical reasoning to deduce which fields are safe and which are not. Such principals and strategies perfectly align with the subject matter covered in the lecture.

2. NP membership of Minesweeper

In this section of the paper we want to show that **MSWP** is in the problem class **NP**. So there exists a short certificate and a verifier that accepts that certificate in at most polynomial time. If that is the case, then our language is complete. If the verifier also rejects assignment which are not part of the language, then it is sound as well ([3]). Both of these requirements need to be fulfilled in order for our language to be part of **NP**.

For Minesweeper specifically we have to consider a placement of b on the game field G, such that every numbered cell correctly indicates the number of adjacent bombs (see section 2.4 for all rules that need to be fulfilled).

2.1. Certificate

The certificate in this case is a assignment of bombs to the game board grid, i.e. a configuration for each cell that tells us whether it's a bomb or not.

2.2. Accepting and Rejecting

To correctly verify the certificate, we need to (i) check the number of bombs of the certificate aligns with the number of bombs b of the input, (ii) for each numbered cell count the number of bombs in the neighbouring cells and verify that the count matches the value given in the cell.

MSWP fulfills the completeness criteria of **NP**, as if there is a Minesweeper game $MG \in$ **MSWP**, then a board configuration with b bombs exists, where all numbered cells are consistent with the count of neighbouring bombs. Using this board placement as a certificate, the verifier will accept since all verification rules are fulfilled. As our input consists of the game field width n, game field height m and amount of bombs b (so $(n \cdot m) + 1$), the input size is at most polynomial ($O(n \cdot m)$). In the worst case (square grid, $n = m$) our runtime is $O(n^2)$.

The soundness of **MSWP** is present as well, because we reject every certificate that violates the game field rules set by us (see section 2.4). Thus it is not possible for a false certificate to be accepted.

As we have proven **MSWP** is complete and sound, thus we conclude that **MSWP** is a member of **NP**.

2.3. Encoding Minesweeper

Because of the relative simplicity and structured rule set of Minesweeper, we can construct logical formula based on those rules.

Input Because Minesweeper is played on a two-dimensional field, we have the game board width n and height m. Furthermore we also have the number of bombs b.

Output We want to know if the given configuration is a valid assignment of bombs.

Game Board Definition To effectively encode the game board, we assign a unique label F_iX to each cell on the game board. i is a natural number refers to a specific cell on the board. We start at the cell on the left-most upper corner of the game field and start counting up. $i \in \{1, \ldots, n \cdot m\}$.

X on the other hand specifies the amount of bombs that surround the current cell F_i. X ranges from 0 to 9, however we have chosen 9 to mean that a bomb is placed on that specific position. $X \in \{\underbrace{0,1,2,3,4,5,6,7,8}_{\text{neighbouring bombs}}, \underbrace{9}_{\text{bomb}}\}$.

Neighbours As previously established, a cell can have up to $k = 8$ neighbours. We encode each neighbour of the current cell F_i by n_k where $k \in \{1, \ldots, 8\}$. Each n_k has a truth value. If n_k is true, then that neighbour is a bomb. If it is false, then the neighbour isn't a bomb. We introduce this for easier notation when iterating over the neighbours.

2.4. Game Rules

Rule: Each field has at least one value

$$\bigwedge_{i=1}^{n \cdot m} \left(\bigvee_{x=0}^{9} F_iX \right)$$

Rule: each field has at most one value

$$\bigwedge_{i=1}^{n \cdot m} \bigwedge_{0 \leq x_1 < x_2 \leq 9} \left(\overline{F_{ij}X_1} \vee \overline{F_{ij}X_2} \right)$$

Rule: across the entire game board there are at least b bombs

$$j = n \cdot m - b + 1$$
$$\binom{n \cdot m}{j} \text{ clauses}$$
$$j \text{ literals each}$$

$$\bigwedge_{1 \leq i_1 < i_2 < \ldots < ij \leq n \cdot m} (F_{i1}9 \vee F_{i2}9 \vee \ldots \vee F_{ij}9)$$

Rule: across the entire game board there are at most b bombs

$$\begin{gathered} \tbinom{n \cdot m}{b+1} \text{ clauses} \\ b+1 \text{ literals each} \end{gathered}$$

$$\bigwedge_{1 \leq i_1 < i_2 < \ldots < ib < ib+1 \leq n \cdot m} (\overline{F_{i1}9} \vee \overline{F_{i2}9} \vee \ldots \vee \overline{F_{ib}9} \vee \overline{F_{ib+1}9})$$

Rule: if a field has value 0, then no neighbours are a bomb

$$\begin{gathered} n \cdot m \cdot 8 \text{ clauses} \\ 2 \text{ literals each} \end{gathered}$$

$$\bigwedge_{i=1}^{n \cdot m} \bigwedge_{k=1}^{8} (\overline{F_i 0} \vee \overline{n_k})$$

Rule: if a field has value z, then exactly z neighbours are bombs

$$1 \leq z \leq 7$$

at least z bombs:

$$\begin{gathered} j = 8 - z + 1 \\ n \cdot m \cdot \tbinom{8}{j} \text{ clauses} \\ j+1 \text{ literals each} \end{gathered}$$

$$\bigwedge_{i=1}^{n \cdot m} \bigwedge_{1 \leq k_1 < k_2 < \ldots < kj \leq 8} (\overline{F_i X} \vee n_{k1} \vee \ldots \vee n_{kj})$$

at most z bombs:

$$j = z + 1$$
$$n \cdot m \cdot \binom{8}{j} \text{ clauses}$$
$$j + 1 \text{ literals each}$$

$$\bigwedge_{i=1}^{n \cdot m} \bigwedge_{1 \leq k_1 < k_2 < \ldots < kj \leq 8} (\overline{F_i X} \vee \overline{n_{k1}} \vee \ldots \vee \overline{n_{kj}})$$

Rule: if a field has value 8, then all neighbours are a bombs

$$n \cdot m \cdot 8 \text{ clauses}$$
$$2 \text{ literals each}$$

$$\bigwedge_{i=1}^{n \cdot m} \bigwedge_{k=1}^{8} (\overline{F_i 8} \vee n_k)$$

3. Karp Reduction from 3-SAT

3.1. SAT $\leq_p$ MSWP

To encode a **SAT** formula F with n variables and m clauses into an instance of **MSWP** M, we define structures which function as building blocks. (see section 3.4).

The basic idea is to look closer at the rows and columns. Each row of the game board represents a clause, while each column is a stream of inputs and variables where we determine the value for each variable. Via these two stream and our defined components we construct a formula in CNF that is satisfiable if and only if there exists a layout of bombs such that the Minesweeper board is in a valid configuration.

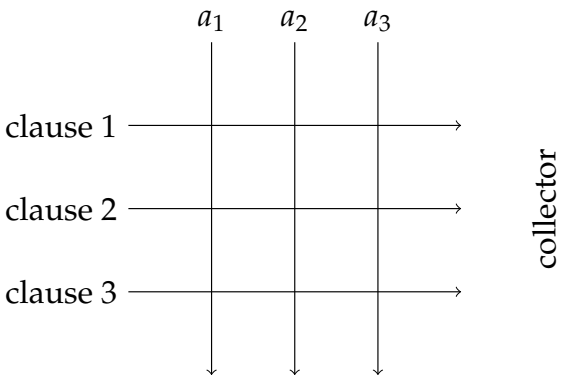


Figure 2: Concept behind **MSWP** formula construction

Symbol	Definition
n	denotes how many bombs are in the neighbourhood of cell
0	blank cell
●	predefined bombs
●	input-dependent bomb
X	input-dependent safe cell
?	value of variable (can be true: X or false: ●)
•	input-dependent cell (can become X or ● depending on variable value)

Table 1: Notation Table

3.2. Alphabet & Notation

First of all we introduce an alphabet that will help us later with the verification. The alphabet encodes all possible values a cell within the game field may have. A cell could be number n without zero (same behaviour as in Minesweeper), be blank (defined here as 0), have a predefined bomb, an input-dependent bomb, an input-dependent safe cell, a variable (unopened cell) or a cell without a value. Basically we define $\alpha = \{n, 0, \cdot, \cdot, \times, ?, \cdot\}$.

As the following chapters will frequently use the alphabet in multiple figures, we created a reference table for the readers' convenience.

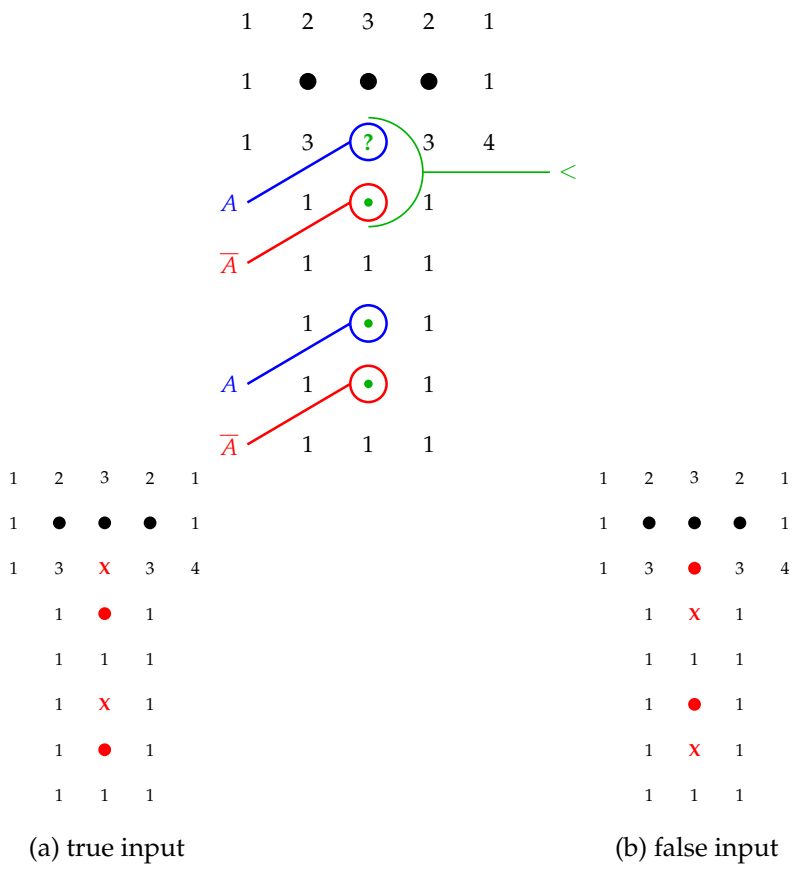
3.3. Evaluation

The method to compute our Minesweeper field into a logical formula goes as such:

- We divide the game field into rows and columns
- We form clauses via the rows, by going from left to right
- We compute variables by going from top to bottom for each column
- To aid with our computation, we use the help of multiple components which alter the input stream of our rows and columns

3.4. Components

INPUT STREAM This input stream goes from top to bottom and evaluates each variable (? and ●) and sets its according value (X or ●). (See Table 1)



(a) true input

(b) false input

Figure 3: Input Stream

CLAUSE STREAM Represent a clause within our formula. It starts out as false. When we encounter a variable we check its truth value. If it evaluates to true, then our clause becomes true. Should the variable be false, but the clause's previous state was true, then the clause still stays true. So we use a logical or on the clause itself and the variable in the input stream $\underbrace{(x \vee y \vee \ldots)}_{\text{Clause}} \vee x$.

In simple terms: The Clause Stream "carries" the state of the clause from left to right, and as soon as an input satisfies the clause, the clause becomes true for the rest of the stream.

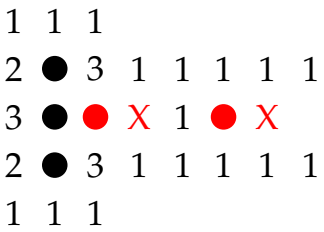


Figure 4: Clause Stream

CROSSING This is used when the Clause Stream and input stream meet, but the variable of the Input Stream is not part of the clause. This component causes for the variable in the Input Stream to be ignored by the Clause Stream, i.e. the Clause Stream will not use the variable. The Clause Stream and Input Stream remain unchanged.

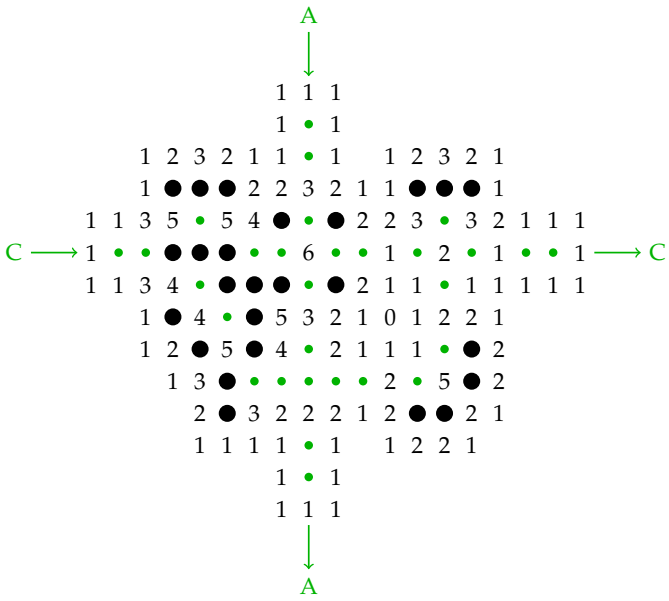


Figure 5: Crossing

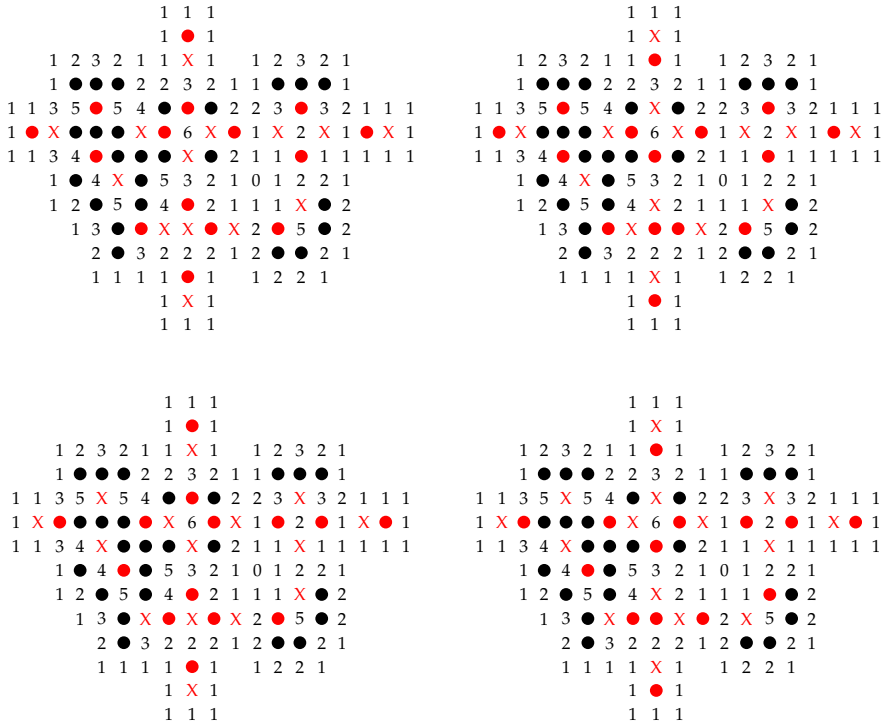


Figure 6: Every possible assignment Clause Stream and Input Stream within Crossing. From top-left to bottom right the options are (Input, Clause): (false, false), (true,false), (false, true), (true, true)

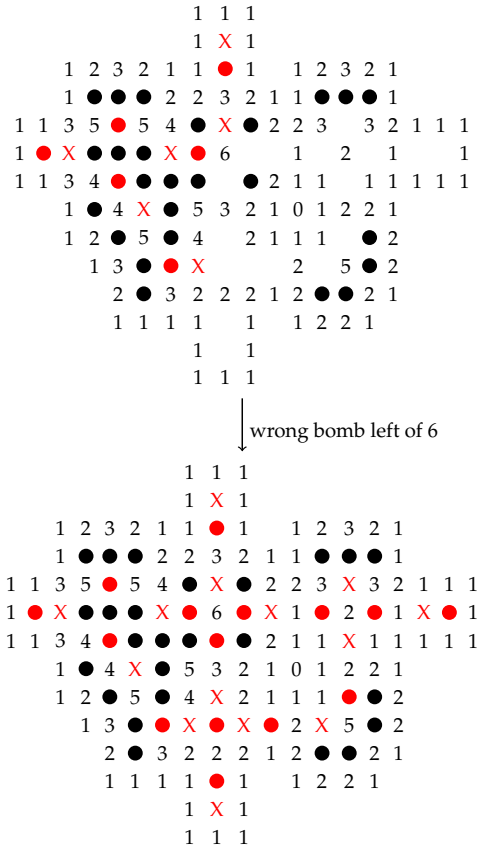


Figure 7: Crossing Example

OR A simple disjunction of the input stream of the clause and the variable input stream. The clause becomes true if it either was already true or the current literal is true. Functionally identical to a logical OR-Gate.

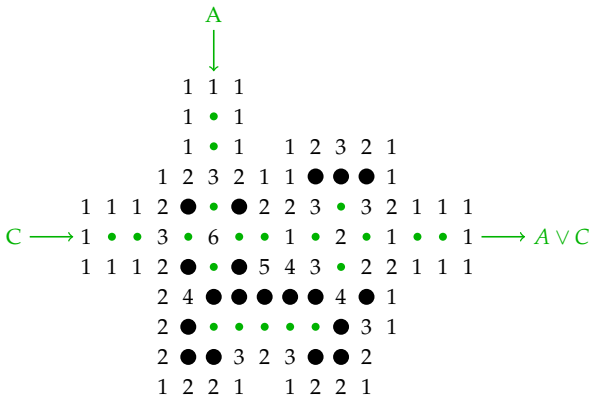


Figure 8: Or

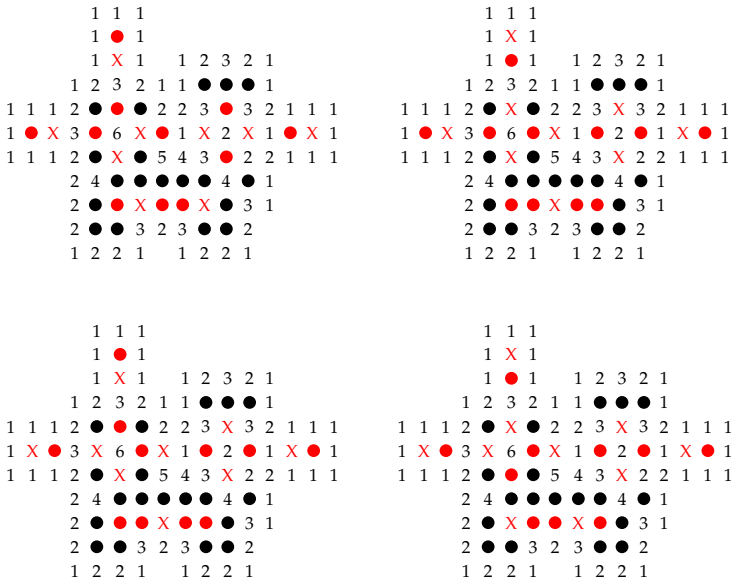
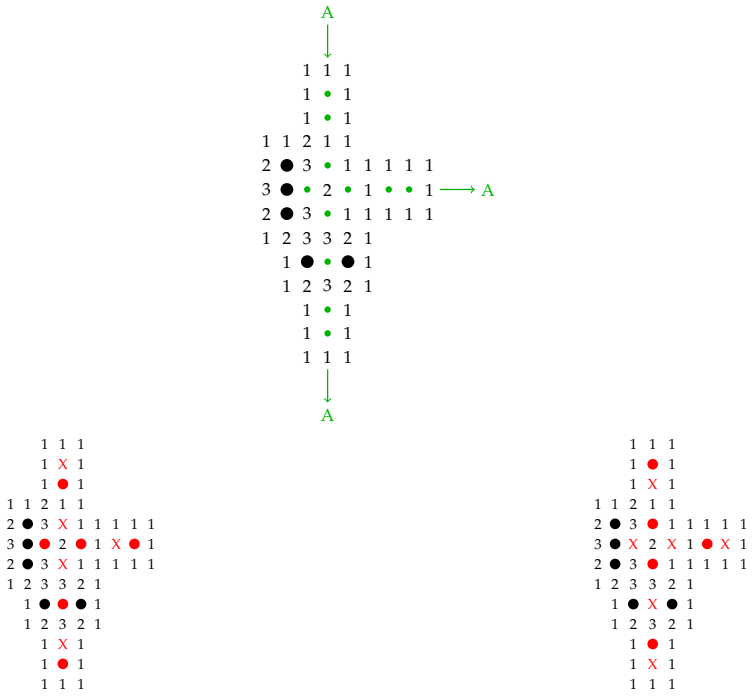


Figure 9: Every possible assignment Clause Stream and Input Stream within Or. From top-left to bottom right the options are (Input, Clause): (false, false), (true,false), (false, true), (true, true)

SPLITTER This takes a variable input stream and splits it into two outputs with the same value. This is primarily used to aid us with the **OR** component, as otherwise we'd only have the disjunction of the clause and the literal, but this way we get to compute the

variable input stream further and keep the calculation from the **OR** component. Thus we always use this before an **OR**.



(a) true input

(b) false input

Figure 10: Splitter

TURN STREAM Turns a horizontal stream into a vertical one. Needed as some components stop computing the pure input stream, but as we often still need to further process the input stream, we use this component. This module, alongside **SPLITTER**, is necessary when using the **OR** component. **TURN STREAM** main functionality is that it brings both the **CLAUSE STREAM** and **INPUT STREAM** together. Also the **CROSSING** and **OR** components expect that the Input Stream is vertical.

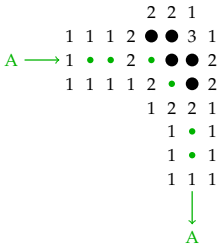
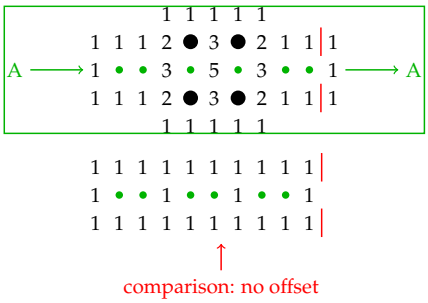
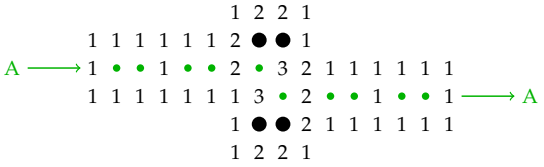


Figure 11: Turn Stream

HORIZONTAL/VERTICAL OFFSET STREAM Offsets the variable **INPUT STREAM** horizontally or vertically. We introduce this component as we often need to align the **INPUT STREAM** after certain operations and as the input stream only repeats on every third cell.



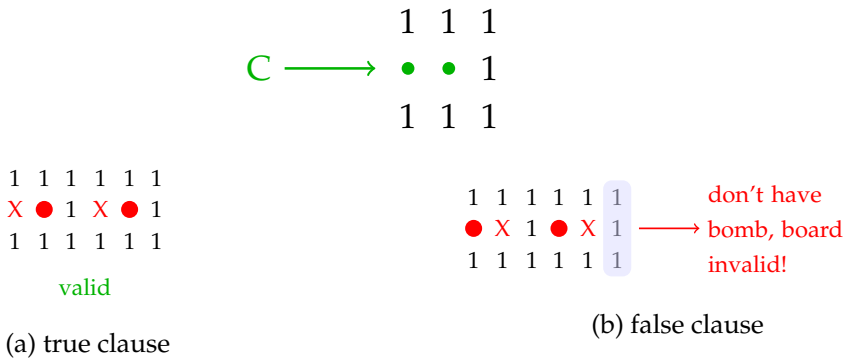
(a) horizontal offset



(b) vertical offset

Figure 12: Offset Stream

COLLECTOR This component signifies the end of a clause. All collectors have to evaluate to true, as otherwise the formula will not be satisfiable and the game field will thus also be inconsistent.

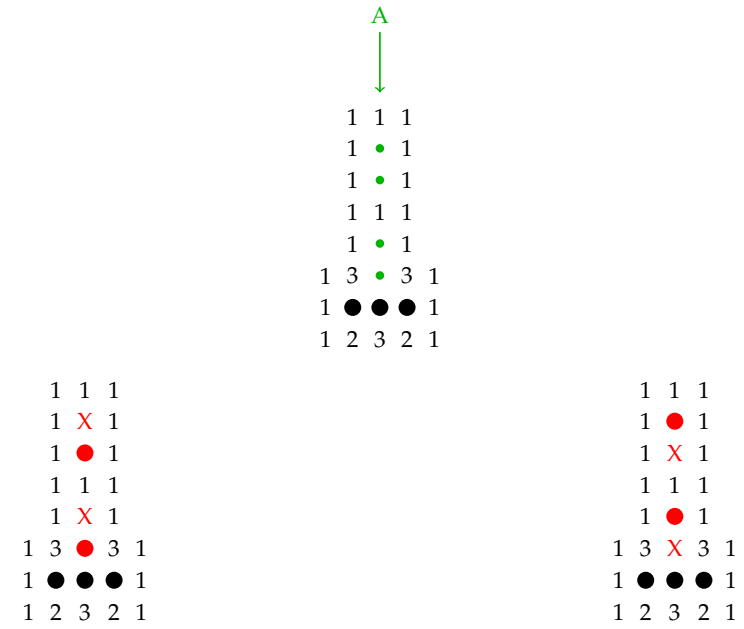


(a) true clause

(b) false clause

Figure 13: Collector

END Signifies the end of the variable **INPUT STREAM**.



(a) true clause

(b) false clause

Figure 14: End

3.5. Constructing SAT Formula

After explaining every component within our system, we want to use them to construct a CNF formula φ. We first define a function f : **SAT** $\rightarrow$ **MSWP** that maps φ to a game of Minesweeper.

To do that we first take a look at the **INPUT STREAM**. The **INPUT STREAM** receives an input-dependent cell and then, depending on the value of the variable, we can determine if the input x_i is a safe cell (true) or a bomb (false).

Next up we analyse the **CLAUSE STREAM**. As defined in section 3.4, the **CLAUSE STREAM** component determines if a clause evaluate to true or not, by analysing the next literal or by taking the clauses previous state. Clauses are initialised to *false* by default.

Finally each clause ends up in a **COLLECTOR** which enforces clauses to evaluate to true. (We achieve a similar feat to that of 8.7 of the lecture script, as we have broken a large formula up into the input, consistancy and termination parts). As the size of the game board is the number of variables and clauses, the formula can be computed in polynomial time.

As is immediately obvious, a constructed **MSWP** formula directly emulates a given **SAT** formula, from the literals to the clauses. Therefore any model of the formula in **SAT** directly corresponds to a model in **MSWP** (and vice versa) ([4]).

3.6. 3-SAT $\leq_p$ SAT

To also show the reduction from **3-SAT**, we will use the intermediate reduction to **SAT** ([4]), the language of satisfiable boolean formulas in CNF. This relation is immediately obvious, as **3-SAT** $\subset$ **SAT**. For the Karp reduction we have $f(x) = x$ and $x \in$ **3-SAT** $\iff$ $f(x) \in$ **SAT**.

3.7. 3-SAT $\leq_p$ MSWP

Due to the transitivity of Karp reductions, we can combine the previous reductions **3-SAT** $\leq_p$ **SAT** and **SAT** $\leq_p$ **MSWP** to conclude **3-SAT** $\leq_p$ **MSWP**.

3.8. NP-hard & NP-complete

As shown, an arbitrary formula in **3-SAT** can be converted to an instance of **MSWP** in polynomial time. This proves that MSWP is **NP**-hard. Together with the fact that we have proven that **MSWP** is in **NP**, we conclude that **MSWP** is **NP**-complete.

3.9. Cost Calculation Example

A **SAT** formula inside Minesweeper consists of m number of clauses with n amount of variables.

If we take a 5×4 game field as an example, we can calculate the following:

Bombs $m \cdot (3 \cdot 98 + (n-3) \cdot 47) + n \cdot (4+4) + m \cdot (4+1)$. Each clause has to check three literals, all other variables not in the clause are not checked, thus we have to subtract them, Input and Output Size, Clause Start + Collector

Height $4 + m \cdot 25 + 5$. (Input Size, checked blocks, end size)

Width $4 + n \cdot 39 + 3$. (Clause Start, checked blocks, Collector Size)

Transforming a 3-SAT Problem to a Minesweeper Problem is possible and costs are P and not exponential. Thus Minesweeper is **NP**-complete.

A. Indexes

List of Figures

List of Tables

B. Addendum

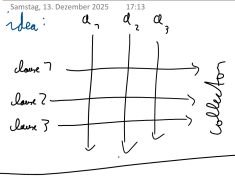
This section is for interested readers who'd like to see how we derived this paper's contents.

B.1. Component Definition

Minesweeper - Component Definition

Samstag, 13. Dezember 2025 17:13

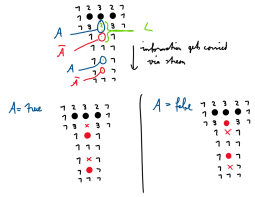
Idea: similar to MC Vaster



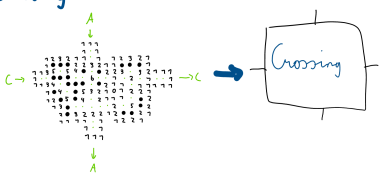
CNF formula satisfiable ⇔ There exists a layout of bombs and that the minesweeper board is valid

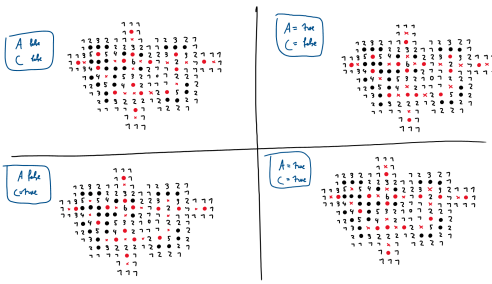
Notation: numbers have the same function they have in Minesweeper
all completely empty space is a 0
● ... predefined Bomb
● ... input-dependent Bomb
× ... input-dependent safe field (no Bomb)
? ... value of variable (true → ×, false → ●)
· ... field without value, can become × or ● (dependent on input)

Input stream for a variable:



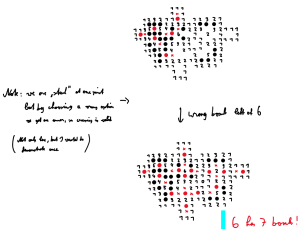
Crossing: clause does not use variable, stream do not change



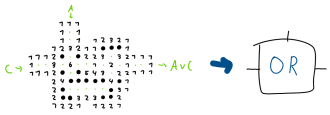


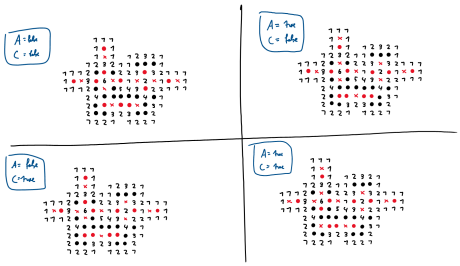
Close stream

clause variable appeared in a clause is ? if at least one literal satisfied the clause



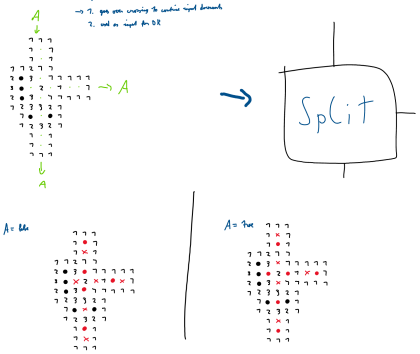
OR: clause is true if clause was already true or if current literal is true





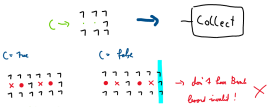
Splitter: Because OR does not keep variable input as secondary output,

use split input from OR to two equal stream
→ 1. goes over crossing to contain input bombs
2. used as input for OR

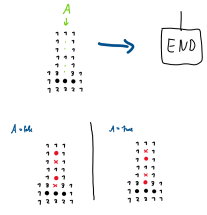


Collector: needs to receive true

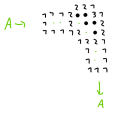
Otherwise clause is false → not satisfying assignment → not valid minesweeper configuration



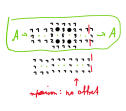
End: ends input stream



Turn stream



Offset stream (horizontally)

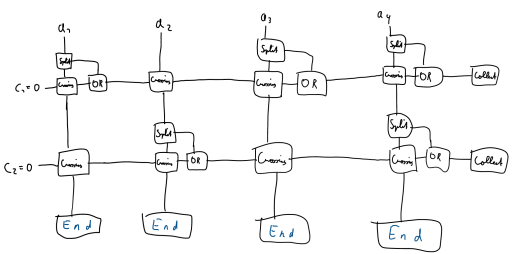


Offset stream (vertically)

needed for clause to read negation of variable (when it was negated literal)

Example $(a_1 \vee \bar{a}_3 \vee a_4) \wedge (a_2 \vee \bar{a}_4)$

c_1 c_2



+ Offsets where they are needed

B.2. Cost Calculations

Minesweeper Cost Calculations

Samstag, 13. Dezember 2025 17:16

Input

3 predetermined, 1 input dependent } 4 Bombs

width : 5

height : 4

End

3 predetermined, 1 input dependent } 4 Bombs

width : 5

height : 5

Start of clause stream

4 Bombs

width : 4

height : 5

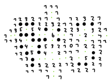
Collector

1 input dependent Bomb

width 3

height 3

Crossing

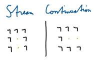


25 predetermined, max 11 input dependent } 36 Bombs

width : 18

height : 13

Stream Continuation



1 input dependent Bomb

width 3

height 3

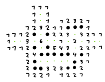
Turn

5 predetermined, 3 input dependent } 8 Bombs

width : 8

height : 7

OR

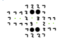


19 predetermined, max 11 input dependent } 30 Bombs

width : 15

height : 11

Offset vertically

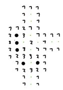


4 predetermined, 3 input dependent } 7 Bombs

width : 9

height : 6

Splitter

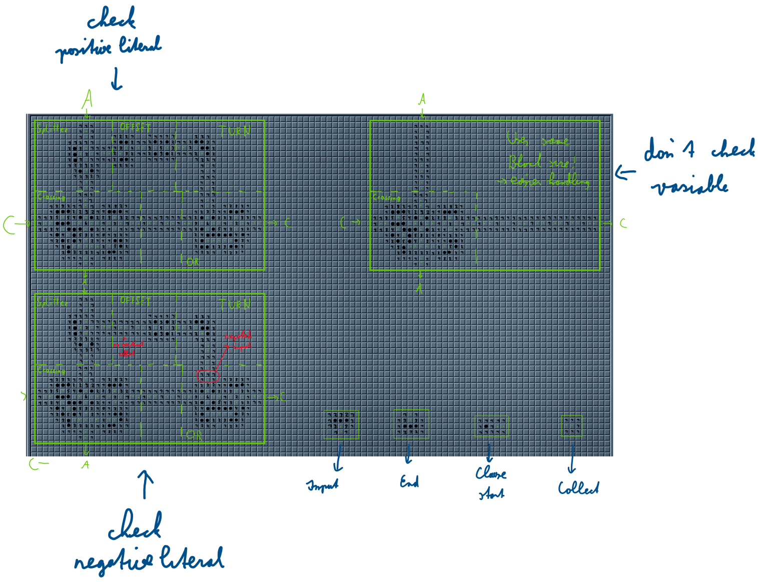


5 predetermined, 5 input dependent } 10 Bombs

width : 8

height : 12

Possible Blocks:



Check Blocks:

height : 25

width : 39

Bomb count :
- check positive : 62 predetermined, 36 dependent } 98 Bomb — has higher cost
- check negative : 58 predetermined, 36 dependent } 94 Bomb
- don't check : 25 predetermined, 22 dependent } 47 Bomb

Cost calculation:

3-SAT CNF Formular : n variables, m clauses

Bombs : $m \cdot (3 \cdot 98 + (n-3) \cdot 47) + n \cdot (4+4) + m \cdot (4+7)$

- $3 \cdot 98$: each clause has to check 3 literals (take more expensive block)
- $(n-3) \cdot 47$: all other variable are not in clause → not checked
- $(4+4)$: Input, End
- $(4+7)$: clause start, Collector

Height : $4 + m \cdot 25 + 5$ (input, check blocks, End)

Width : $4 + n \cdot 39 + 3$ (clause start, check blocks, Collector)

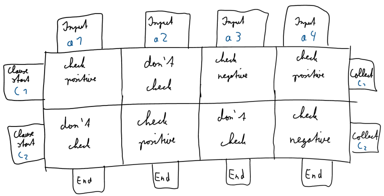
Transforming an 3-SAT Problem to a Minesweeper problem is possible and costs are in P (not exponential)

→ Minesweeper is NP complete!

Example

$(a_1 \vee \overline{a_3} \vee a_4) \wedge (a_2 \vee \overline{a_4})$

(c_1, c_2)

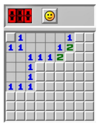


B.3. Reduction to SAT

Minesweeper

Samstag, 6. Dezember 2025 12:33

Input: n, m (dimension of game field), b (count of bombs), Given field → Eg:



$n = 9 \quad m = 9 \quad b = 10$

Known variable that are true:
$F_2 1, F_3 0, F_4 0, F_5 0, F_6 0, F_7 1$
$F_{10} 1, F_{11} 1, F_{12} 0 \ldots$

Output: Is given board solveable?
= Is there an valid assignment of bombs?

Definition / Explanation of variables

$F_i X$ → Specific field on game board with $1 < i \le n \cdot m$
With a certain value $X \in \{0,1,2,3,4,5,6,7,8,9\}$ (0–8: number, 9: Bomb)

F_1	F_2	..	F_n
F_{n+1}	F_{n+2}	...	F_{2n}
⋮	⋮	⋮	⋮
$F_{(m-1)\cdot n+1}$	...	...	F_{nm}

(n columns, m rows)

Neighbours n_K: We define the $k=8$ neighbours of F_i as $n1$ to $n8$
if n_K = true → neighbour is a bomb
if n_K = false → neighbour is not a bomb

This is just for simplicity for iteration over the neighbours and to not always having to write ‚9' to demonstrate the bomb value

F_{i-n-1}	F_{i-n}	F_{i-n+1}
F_{i-1}	F_i	F_{i+1}
F_{i+n-1}	F_{i+n}	F_{i+n+1}

→

n1	n2	n3
n4	F_{ij}	n5
n6	n7	n8

Note: all Neighbours that are outside the game board always have the value 0
→ $F_i\,0$ = true for $i < 1$ or $i > n \cdot m$
$F_i\,1 \ldots F_i\,9$ = false

Transition from all game rules to SAT

Rule: Each field has at least one value

$$\bigwedge_{i=1}^{n\cdot m}\left(\bigvee_{x=0}^{9} F_{ij}X\right)$$

$(F_{11}0 \vee F_{11}1 \vee F_{11}2 \vee \ldots) \wedge$
$(F_{12}0 \vee F_{12}1 \vee \ldots) \wedge$
⋮
$(F_{nm}0 \vee F_{nm}1 \vee \ldots)$

$n \cdot m$ clauses
10 literals each

Rule: Each field has at most one value

$$\bigwedge_{i=1}^{n\cdot m}\bigwedge_{0\le x_1<x_2\le 9}\left(\overline{F_{ij}X_1} \vee \overline{F_{ij}X_2}\right)$$

$(\overline{F_{11}0} \vee \overline{F_{11}1}) \wedge (\overline{F_{11}0} \vee \overline{F_{11}2}) \wedge \ldots$
⋮
$(\overline{F_{nm}0} \vee \overline{F_{nm}1}) \ldots (\overline{F_{nm}8} \vee \overline{F_{nm}9})$

$n \cdot m \cdot 36$ clauses
2 literals each

Rule: Across the board there are at least b bombs

$j = n \cdot m - b + 1$

$$\bigwedge_{1\le i_1<i_2\ldots<i_j\le n\cdot m}\left(F_{i_1}9 \vee F_{i_2}9 \vee \ldots \vee F_{i_j}9\right)$$

$\binom{n\cdot m}{j}$ clauses
j literals each

Rule: Across the board there are at most b bombs

$$\bigwedge_{1\le i_1<i_2<\ldots<i_b<i_{b+1}\le n\cdot m}\left(\overline{F_{i_1}9} \vee \overline{F_{i_2}9} \vee \ldots \vee \overline{F_{i_b}9} \vee \overline{F_{i_{b+1}}9}\right)$$

$\binom{n\cdot m}{b+1}$ clauses
$b+1$ literals each

Rule: if a field has value 0, no neighbour is a bomb → n_K is false

$$\bigwedge_{i=1}^{n\cdot m}\bigwedge_{k=1}^{8}\left(\overline{F_i0} \vee \overline{n_K}\right)$$

$n \cdot m \cdot 8$ clauses
2 literals each

Rule: If a Field has value 1, exactly one neighbour is a bomb → $n_K 9$ = true

at least one: $\bigwedge_{i=1}^{n\cdot m}\left(\overline{F_i1} \vee n_1 \vee n_2 \vee n_3 \vee \ldots \vee n_8\right)$ — $n \cdot m$ clauses, 9 literals each

at most one: $\bigwedge_{i=1}^{n\cdot m}\bigwedge_{1\le k_1<k_2\le 8}\left(\overline{F_i1} \vee \overline{n_{k1}} \vee \overline{n_{k2}}\right)$ — $n \cdot m \cdot 28$ clauses, 3 literals each

Rule: If a Field has value 2, exactly 2 neighbours are bombs

at least two: $j = 7$ $(8-x+1)$ — $\bigwedge_{i=1}^{n\cdot m}\bigwedge_{1\le k_1<\ldots<k_j\le 8}\left(\overline{F_i2} \vee n_{k_1} \vee \ldots \vee n_{kj}\right)$ — $n \cdot m \cdot 8$ clauses, 8 literals each

at most two: $\bigwedge_{i=1}^{n\cdot m}\bigwedge_{1\le k_1<k_2<k_3\le 8}\left(\overline{F_i2} \vee \overline{n_{k1}} \vee \overline{n_{k2}} \vee \overline{n_{k3}}\right)$ — $n \cdot m \cdot 56$ clauses, 4 literals each

Rule: If a Field has value 3, exactly 3 neighbours are bombs

at least 3: $j = 6$ — $\bigwedge_{i=1}^{n\cdot m}\bigwedge_{1\le k_1<\ldots<k_j\le 8}\left(\overline{F_i3} \vee n_{k_1} \vee \ldots \vee n_{kj}\right)$ — $n \cdot m \cdot 28$ clauses, 7 literals each

at most 3: $j = 4$ $(x+1)$ — $\bigwedge_{i=1}^{n\cdot m}\bigwedge_{1\le k_1<\ldots<k_j\le 8}\left(\overline{F_i3} \vee \overline{n_{k1}} \vee \ldots \vee \overline{n_{kj}}\right)$ — $n \cdot m \cdot 70$ clauses, 5 literals each

Rule: If a Field has value 4, exactly 4 neighbours are bombs

at least 4: $j = 5$ — $\bigwedge_{i=1}^{n\cdot m}\bigwedge_{1\le k_1<\ldots<k_j\le 8}\left(\overline{F_i4} \vee n_{k_1} \vee \ldots \vee n_{kj}\right)$ — $n \cdot m \cdot 56$ clauses, 6 literals each

at most 4: $j = 5$ — $\bigwedge_{i=1}^{n\cdot m}\bigwedge_{1\le k_1<\ldots<k_j\le 8}\left(\overline{F_i4} \vee \overline{n_{k1}} \vee \ldots \vee \overline{n_{kj}}\right)$ — $n \cdot m \cdot 56$ clauses, 6 literals each

Rule: If a Field has value 5, exactly 5 neighbours are bombs

at least 5: $j = 4$ — $\bigwedge_{i=1}^{n\cdot m}\bigwedge_{1\le k_1<\ldots<k_j\le 8}\left(\overline{F_i5} \vee n_{k_1} \vee \ldots \vee n_{kj}\right)$ — $n \cdot m \cdot 70$ clauses, 5 literals each

at most 5: $j = 6$ — $\bigwedge_{i=1}^{n\cdot m}\bigwedge_{1\le k_1<\ldots<k_j\le 8}\left(\overline{F_i5} \vee \overline{n_{k1}} \vee \ldots \vee \overline{n_{kj}}\right)$ — $n \cdot m \cdot 28$ clauses, 7 literals each

Rule: If a Field has value 6, exactly 6 neighbours are bombs

at least 6: $j = 3$ — $\bigwedge_{i=1}^{n\cdot m}\bigwedge_{1\le k_1<k_2<k_3\le 8}\left(\overline{F_i6} \vee n_{k_1} \vee n_{k2} \vee n_{k3}\right)$ — $n \cdot m \cdot 56$ clauses, 4 literals each

at most 6: $j = 7$ — $\bigwedge_{i=1}^{n\cdot m}\bigwedge_{1\le k_1<\ldots<k_j\le 8}\left(\overline{F_i6} \vee \overline{n_{k1}} \vee \ldots \vee \overline{n_{kj}}\right)$ — $n \cdot m \cdot 8$ clauses, 8 literals each

Rule: If a Field has value 7, exactly 7 neighbours are bombs

at least 7: $j = 2$ — $\bigwedge_{i=1}^{n\cdot m}\bigwedge_{1\le k_1<k_2\le 8}\left(\overline{F_i7} \vee n_{k_1} \vee n_{k2}\right)$ — $n \cdot m \cdot 28$ clauses, 7 literals each

at most 7: $\bigwedge_{i=1}^{n\cdot m}\left(\overline{F_i7} \vee \overline{n_1} \vee \overline{n_2} \vee \ldots \vee \overline{n_8}\right)$ — $n \cdot m$ clauses, 9 literals each

Rule: If a field has value 8, all neighbours are bombs

$$\bigwedge_{i=1}^{n\cdot m}\bigwedge_{k=1}^{8}\left(\overline{F_i8} \vee n_K\right)$$

$n \cdot m \cdot 8$ clauses
2 literals each

for $1 \le x \le 7$

at least x bombs: $j = 8-x+1$ — $\bigwedge_{i=1}^{n\cdot m}\bigwedge_{1\le k_1<\ldots<k_j\le 8}\left(\overline{F_iX} \vee n_{k_1} \vee \ldots \vee n_{kj}\right)$ — $n \cdot m \cdot \binom{8}{j}$ clauses, $j+1$ literals each

at most x bombs: $j = x+1$ — $\bigwedge_{i=1}^{n\cdot m}\bigwedge_{1\le k_1<\ldots<k_j\le 8}\left(\overline{F_i6} \vee \overline{n_{k1}} \vee \ldots \vee \overline{n_{kj}}\right)$ — $n \cdot m \cdot \binom{8}{j}$ clauses, $j+1$ literals each

---

Count of all clauses

$$n\cdot m + n\cdot m\cdot 36 + \binom{n\cdot m}{n\cdot m-b+1} + \binom{n\cdot m}{b+1} + 2\cdot n\cdot m\cdot\binom{8}{j} + n\cdot m\cdot 8 + n\cdot m\cdot 8$$

($\binom{n\cdot m}{n\cdot m-b+1} = \binom{n\cdot m}{b+1}$; $x = 1 \le j \le 7$; $x = 0$; $x = 8$)

$$= n\cdot m\cdot\left(1 + 36 + 2\cdot\binom{n\cdot m}{b+1} + 8 + 8\right)$$

$$= n\cdot m\cdot\left(55 + 2\cdot\frac{(n\cdot m)!}{(b+1)!\cdot((n\cdot m)! - (b+1)!)}\right)$$ SAT clauses

$O(N)$ → at most polynomial ✓

certificate is in P

References

[1] *Minesweeper (video game)*. Dec. 23, 2025. URL: https://en.wikipedia.org/wiki/Minesweeper_(video_game) (visited on 12/23/2025) (cit. on pp. 1, 2).

[2] GamePro. *Minesweeper - Screenshots zum legendären Klassiker*. 2025. URL: https://www.gamepro.de/galerien/minesweeper,136622.html (visited on 12/23/2025) (cit. on p. 2).

[3] MSc ETH Univ.-Prof. Dr. Richard Kueng. *Introduction to Computational Complexity*. Fall 2025. 2025 (cit. on p. 3).

[4] Jonathan Hocheneder, Felix Wilhelmer, Ellena Pfleger, Yannick Ratzenböck, Simon Oppel, Michael Fletzberger. *The Minecraft Water Problem is NP complete*. Dec. 23, 2024. URL: https://github.com/JonathanDotExe/minecraft-water-problem/tree/main (visited on 12/23/2025) (cit. on p. 18).