Proving NP-Completeness of the Battleship Decision Problem via Karp Reduction

Introduction to Computational Complexity
Winter Semester 2024-2025

Giovanni Filomeno (K12315325)
Farhad Arezo (K12325125)
Jasmin Kronsteiner (K12204076)

December 20, 2024

Abstract

This project demonstrates the NP-completeness of the Battleship Decision Problem by providing a thorough description of the problem, justifying its selection, arguing its membership in NP, and executing a Karp reduction from the known NP-complete problem (3,3,(4))-SAT. The reduction is meticulously constructed to ensure that the Battleship Decision Problem is at least as hard as (3,3,(4))-SAT, thereby establishing its NP-completeness.

Contents

1 Explanation of the Problem

The “Battleship Decision Problem” can be framed as follows: Given a partially completed Battleship board along with certain constraints (such as how many ships of each type should appear, and the ship placement rules), can we determine whether it is possible to legally arrange the remaining ships so that all conditions are satisfied? In other words, starting from predefined grid layout, some cells possibly revealed, some ships or partial ships place, and a known set of rules and ship counts, we ask whether there exists a complete, valid ship placement that matches all given criteria

2 Key Considerations and Constraints

2.1 Rules of Battleship

Typically, ships are placed on a grid such that they do not overlap and often do not even touch each other diagonally. Different puzzle variants may have slightly different rules (e.g., strict separation requirements, fixed ship sizes, or known distributions of ships).

2.2 Input Specification

The input can include different parameters:

- The size of the board (e.g., 10x10)
- Constraints on how many ships of each type (e.g., 1x1 submarines, 2x1 destroyers, etc.) must be placed.
- Information about which cells are already revealed as water (empty) or as part of a ship.
- Additional rules (for example, whether ships can touch diagonally, or whether certain rows and columns must contain a fixed number of ship cells).
-

2.3 Decision Problem

The question is binary: yes or no. Given the partial configuration and the constraints, can we complete the puzzle so that all ships are correctly placed?

3 Why this Problem is Interesting

We chose the Battleship Decision Problem for two main reasons:

3.1 Personal Involvement in the Topic

Giovanni was born in Taranto, a city in southern Italy known for having a military port of the Italian Navy and its arsenal. Games related to sea battles are part of popular folklore.

3.2 Engaging and Intuitive Nature

Battleship is a well-known puzzle and game, widely recognized and enjoyed by many. Transforming it into a formal decision problem helps connect a familiar, intuitive scenario with more abstract concepts from computation complexity theory. The problem is easy to understand yet potentially challenging to solve, making it an appealing testbed for complexity analysis and Karp reductions from known NP-complete problems.

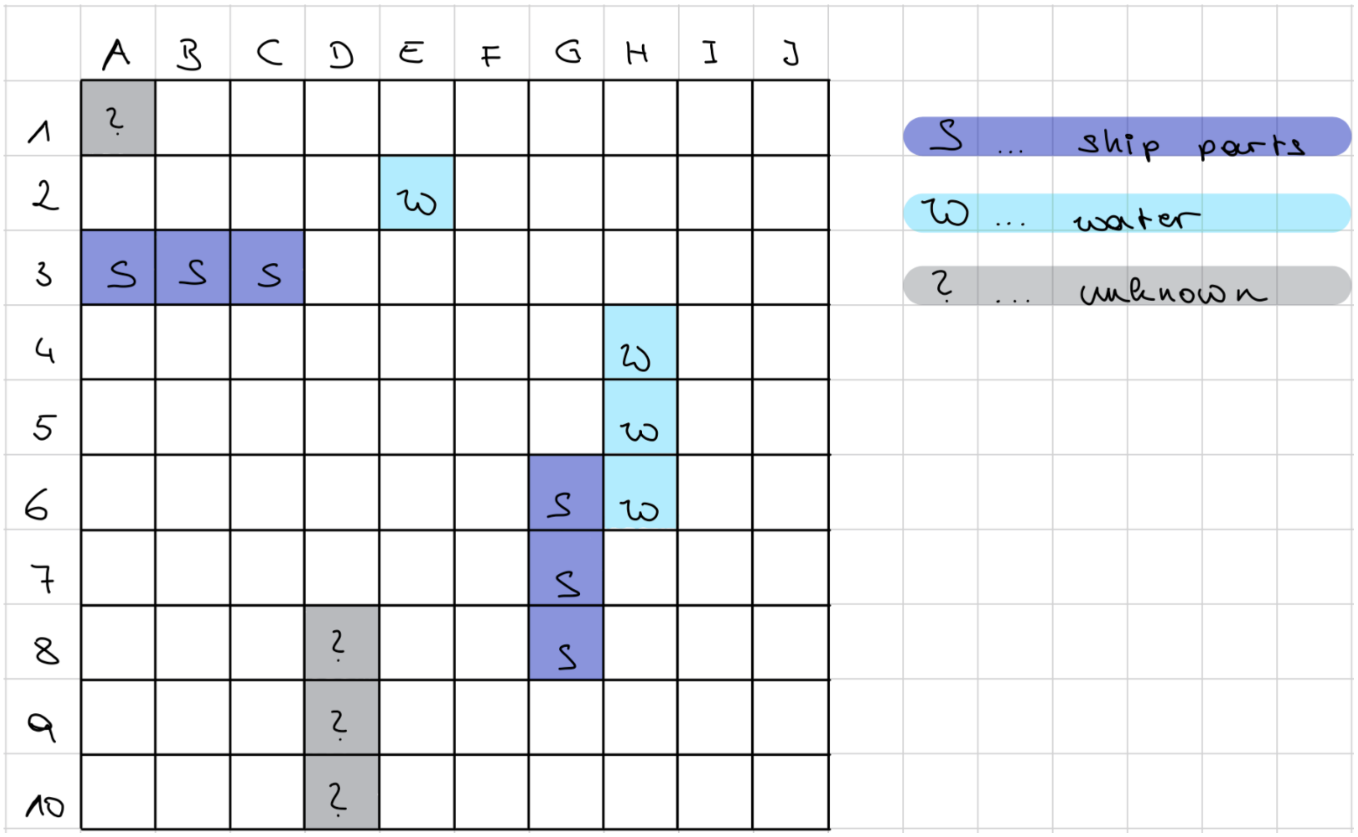
4 Potential Complexity Challenges

While initial attempts might have aimed to integrate these complexity considerations directly into our implemented game, we quickly realized that determining whether an arbitrary Battleship puzzle has a valid solution is non-trivial. It is not simply a matter of trial-and-error, as naive approaches can explode combinatorially. This complexity makes it an interesting candidate for exploring NP-completeness proofs, showing reductions from other well-known NP-complete problems (such as 3-SAT) to Battleship, and thus demonstrating that solving arbitrary Battleship configurations efficiently is likely intractable.

5 Definition of the Required Rules

To define a valid Battleship puzzle instance, we establish the following rules:

- **R1:** All pre-defined ships must be placed on the board exactly as specified
- **R2:** All ships must fit entirely within the boundaries of the grid. No ship segment may extend beyond the edges of the board.
- **R3:** No two ships may touch each other, not even diagonally. In other words, there must be at least one cell of water (or an empty cell) separating ships in every direction.
- **R4:** The number of ship segments in each row and each column is known. This means that for every row and column, we have a constraint on how many cells in that line must be occupied by ship segments.



6 Formal Definition of a Battleship Puzzle Instance

Consider an instance of the Battleship puzzle defined on an m×nm×n grid. We represent such an instance as a tuple

where:

$$(I, R, C, F)$$

- **I – The Initial Configuration:**

1. I – **The Initial Configuration:** The function

$$I : \{1, \ldots, m\} \times \{1, \ldots, n\} \rightarrow \{ship, water, ?\}$$

specifies the initial state of each cell. For each cell (x,y)(x,y):

- $I(x, y) = ship$: This cell is already known to contain a ship segment.
- $I(x, y) = water$: This cell is known to be empty (no ship segment).
- $I(x, y) = ?$: The status of this cell is not determined yet; it may either be ship or water, depending on the solution.

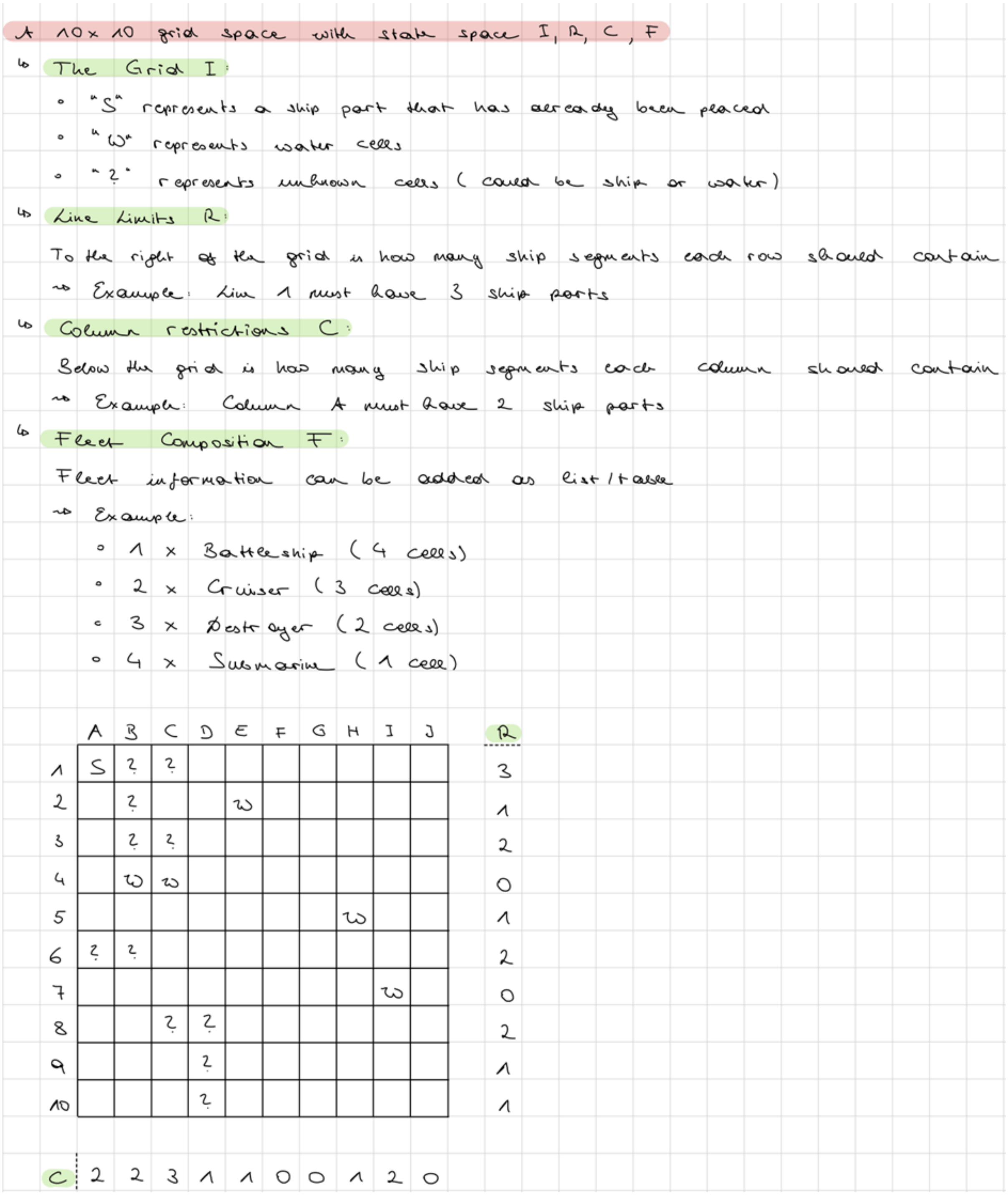
The board consists of $m \times n$ cells in total. The initial configuration II thus provides partial information that the solution must respect.

- **R – Row Constraints:** The function RR gives the number of ship segments required in each row. Formally, for each row $\boldsymbol{i} \in \{\mathbf{1}, \ldots, \boldsymbol{m}\}, \boldsymbol{R}(\boldsymbol{i})$ is the number of ship segments that row ii must contain in a valid solution. In a complexity-oriented formalization, one might represent these constraints in a binary format. The given expression $|R| = n \times log_2(m)$ can be interpreted as an estimate of the space needed to encode all row constraints. In simpler terms, R can be viewed as an array or list of mm integers, each integer specifying the required number of ship segments in that particular row.
- **C – Column Constraints:** Similarly, C is a function that provides the number of ship segments required in each column. For each column j∈{1,...,n},C(j) gives the required number of ship segments in that column. Analogous to R, the expression |C|=m×log_2 (n) suggests a measure of the information content or complexity needed to represent all column constraints. Conceptually, C can be seen as an array of n integers, each specifying the required count of ship segments for its corresponding column.
- **F – Fleet Composition:** The set F describes the fleet that must be placed on the board. It includes the types of ships (e.g., destroyers of length 2, cruisers of length 3, battleships of length 4, etc.) along with how many of each type must be placed. The notation |F|=n_type×log_2 (Max-Anzahl) indicates that to fully describe the fleet, we might need a certain amount of information or encoding length. In practical terms, F could be viewed as a list or a multiset of ships, where each ship type is accompanied by a quantity.

  For example:

$$F = \{(2, count = 3), (3, count = 2), \ldots\}$$

which means three ships of length 2 (first part), two ships of length 3 (second part).



A 10 x 10 grid space with state space I, R, C, F

↳ The Grid I:

- "S" represents a ship part that has already been placed
- "W" represents water cells
- "?" represents unknown cells (could be ship or water)

↳ Line Limits R:

To the right of the grid is how many ship segments each row should contain

→ Example: Line 1 must have 3 ship parts

↳ Column restrictions C:

Below the grid is how many ship segments each column should contain

→ Example: Column A must have 2 ship parts

↳ Fleet Composition F:

Fleet information can be added as list / table

→ Example:

- 1 x Battleship (4 cells)
- 2 x Cruiser (3 cells)
- 3 x Destroyer (2 cells)
- 4 x Submarine (1 cell)

	A	B	C	D	E	F	G	H	I	J	R
1	S	?	?								3
2		?			W						1
3		?	?								2
4		W	W								0
5								W			1
6	?	?									2
7									W		0
8			?	?							2
9				?							1
10				?							1
C	2	2	3	1	1	0	0	1	2	0	

7 Interpretation and Goal of the Puzzle

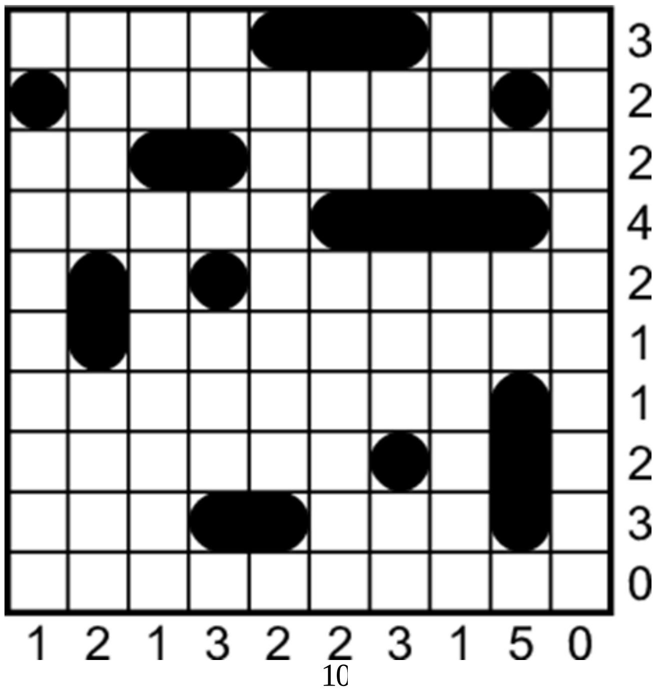
Given the tuple (I, R, C, F), a valid Battleship solution must assign either “ship” or “water” to every unknown cell in I, such that:

- All rules (R1–R4) are satisfied.
- Each row and column meets its ship count constraints given by R and C.
- The prescribed fleet F is fully placed on the board.
- No invalid configurations arise, such as ships touching each other diagonally.

The decision problem is to determine whether such a valid arrangement exists. This problem is not just a simple puzzle, but has been studied in a computational complexity context. It can be shown to be NP-complete, implying that, in general, there is no known efficient algorithm that can solve all instances quickly. By formalizing the puzzle as (I,R,C,F), we can more easily apply complexity-theoretic concepts and prove that standard techniques for solving NP-complete problems (like Karp reductions from 3-SAT) also apply to the Battleship Decision Problem.

8 Example of a Puzzle Instance

Consider a particular Battleship puzzle: you are given a grid, partial placements of ships and water cells, and constraints on how many ship segments each row and column must contain. The question is whether it is possible to complete the configuration so that all rules are satisfied and the correct fleet is placed.



Initial State: Battleship Board representing a (3, 3, (4)) - SAT Problem

	A	B	C	D	E	F	G	H	I	J	R
1	?	?	?	S							4
2			w								0
3	?			S		?					3
4		S					w				1
5	?					?		S			3
6		w					S				1
7				?				?			2
8		?	?	S				w			3
9									?		1
10										?	1
C	3	3	2	4	0	2	1	2	1	1	

Explanation:

- I (Initial Configuration):
  - "S" = Ship segment
  - "W" = Water
  - "?" = unknown
- R (Row constraints):
  Number on the right of each row indicate the required ship segments in that row
- C (Column Constraints):
  Number at the bottom of each column indicate the required ship segments in that column
- F (Fleet Composition):
  - 1 x Battleship (4 cells)
  - 2 x Cruiser (3 cells)
  - 3 x Destroyer (2 cells)
  - 4 x Submarine (1 cell)

Solution State: Completed Board with a Satisfying SAT-assignment

	A	B	C	D	E	F	G	H	I	J	R
1	S	S	S	S							4
2			w								0
3	S			S		S					3
4		S					w				1
5	S					S		S			3
6		w					S				1
7				S				S			2
8		S	S	S				w			3
9									S		1
10										S	1
C	3	3	2	4	0	2	1	2	1	1	

→ All row and column constraints (R, C) are satisfied

→ The fleet F is fully placed according to the initial configuration

9 Argument for Battleship Being in NP

To show that deciding the solvability of a Battleship puzzle is in NP, consider the following:

9.1 Efficient Verification

Once a candidate solution (a fully completed grid configuration) is provided, it is straightforward and efficient to verify its correctness. The solution itself can serve as a certificate.

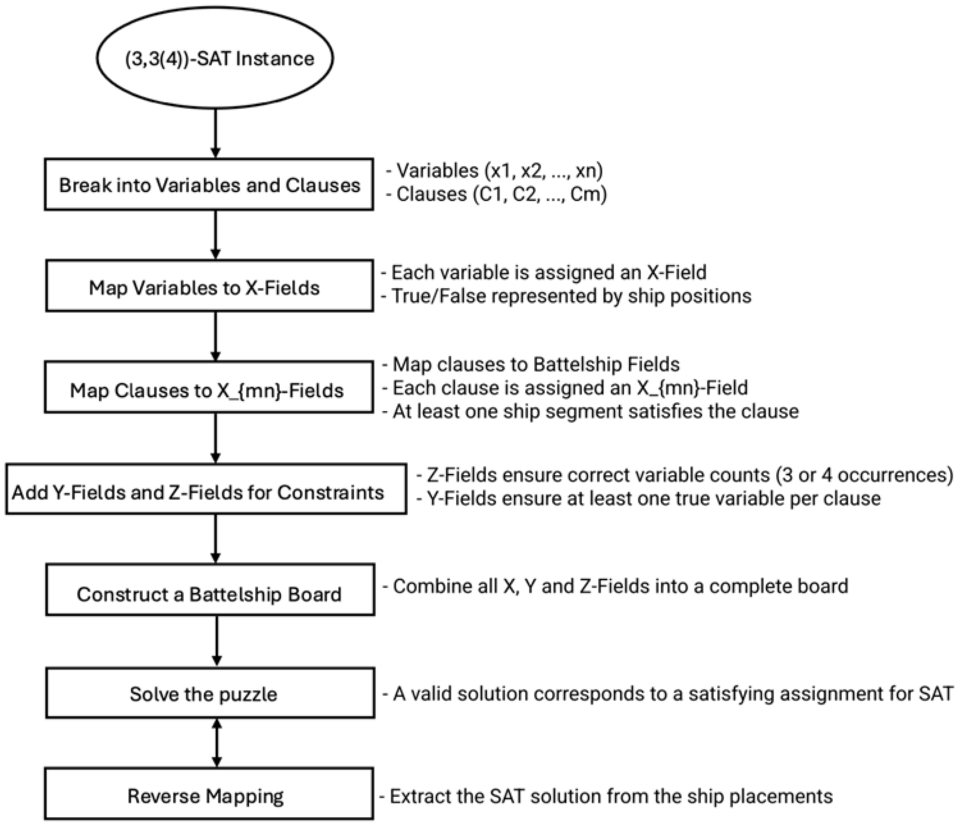
9.2 Verification Steps

- Check that all placed ships correspond exactly to the required fleet composition F.
- Verify that no ships touch each other, not even diagonally (Rule R3).
- Ensure that the counts of ship segments in each row and column match the given constraints R and C (Rule R4).
- Confirm that the solution respects all pre-placed ship and water cells specified by I.

Since each of these checks can be performed in polynomial time relative to the size of the grid and the input constraints, the verification process is efficient. This implies that the decision problem belongs to NP: if a solution exists, you can quickly verify it given the correct "certificate" (the fully filled-in puzzle).

10 Karp Reduction from (3,3,(4))-SAT to Battleship

To prove NP-completeness, it is not enough to show that Battleship is in NP. We also need to show that any known NP-complete problem can be reduced (in polynomial time) to it. This establishes that Battleship is at least as hard as that NP-complete problem.


(3,3(4))-SAT Instance
Break into Variables and Clauses
- Variables (x1, x2, ..., xn)
- Clauses (C1, C2, ..., Cm)
Map Variables to X-Fields
- Each variable is assigned an X-Field
- True/False represented by ship positions
Map Clauses to X_{mn}-Fields
- Map clauses to Battelship Fields
- Each clause is assigned an X_{mn}-Field
- At least one ship segment satisfies the clause
Add Y-Fields and Z-Fields for Constraints
- Z-Fields ensure correct variable counts (3 or 4 occurrences)
- Y-Fields ensure at least one true variable per clause
Construct a Battelship Board
- Combine all X, Y and Z-Fields into a complete board
Solve the puzzle
- A valid solution corresponds to a satisfying assignment for SAT
Reverse Mapping
- Extract the SAT solution from the ship placements

10.1 Known NP-Complete Problem: (3,3,(4))-SAT

We start with a variant of the SAT problem, called (3,3,(4))-SAT, which is known to be NP-complete. In (3,3,(4))-SAT:

- We are given a Boolean formula $\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$.
- Each clause C_i has exactly 3 variables (or their negations).
- Each variable x_j appears in either exactly 3 or exactly 4 clauses in the entire formula.

The goal is to determine whether there is an assignment of truth values to the variables $x_1, x_2, \ldots, x_n$ that satisfies all clauses $C_1, \ldots, C_m$.

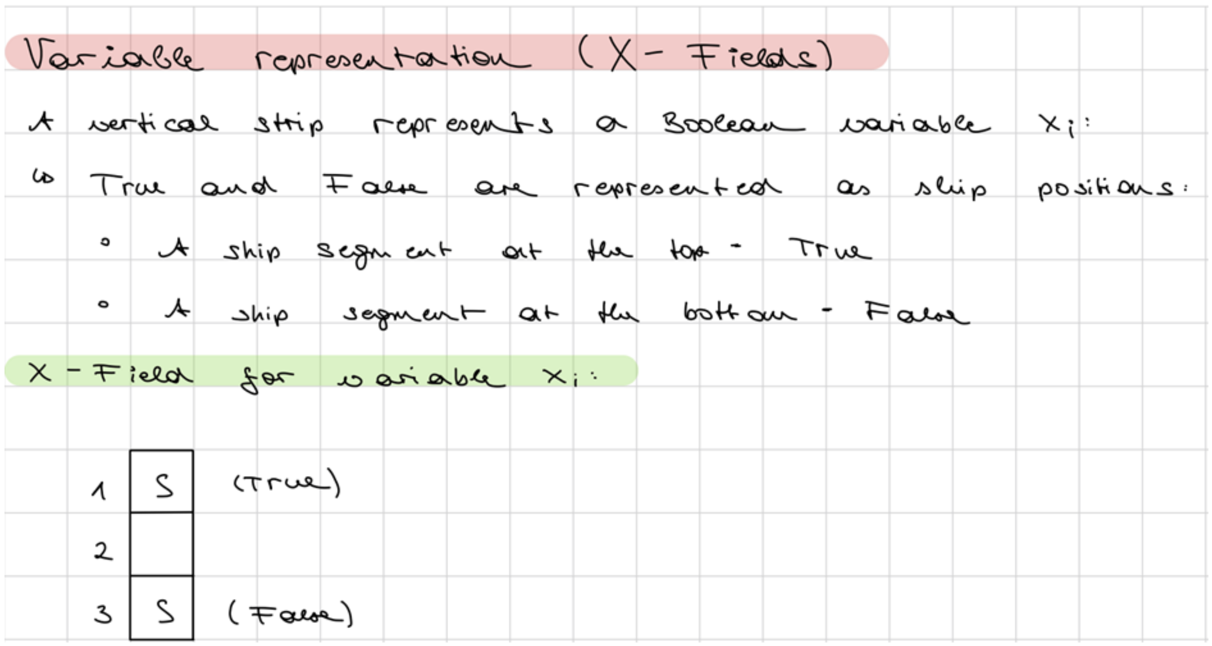
10.2 The Core Idea of the Reduction

We construct a Battleship instance that encodes the Boolean formula. The construction ensures that placing ships in certain positions corresponds directly to assigning truth values to variables, and the constraints enforce that all clauses must be satisfied. If a valid fleet configuration exists that satisfies all the Battleship constraints, it will map back to a satisfying truth assignment of the formula, and vice versa.

10.3 The Basic Layout of the Encoding

The puzzle board is arranged into distinct regions, each corresponding to variables and clauses. A conceptual diagram (as shown in the provided figures) partitions the board into blocks representing:

- Variable Blocks (X-Fields): Each variable X_n is represented by a specific configuration of cells (white and gray). Placing a ship segment in a certain position in these blocks will represent the variable being assigned "true" or "false."



Variable representation (X - Fields)

A vertical strip represents a Boolean variable x_i:

↳ True and False are represented as ship positions:

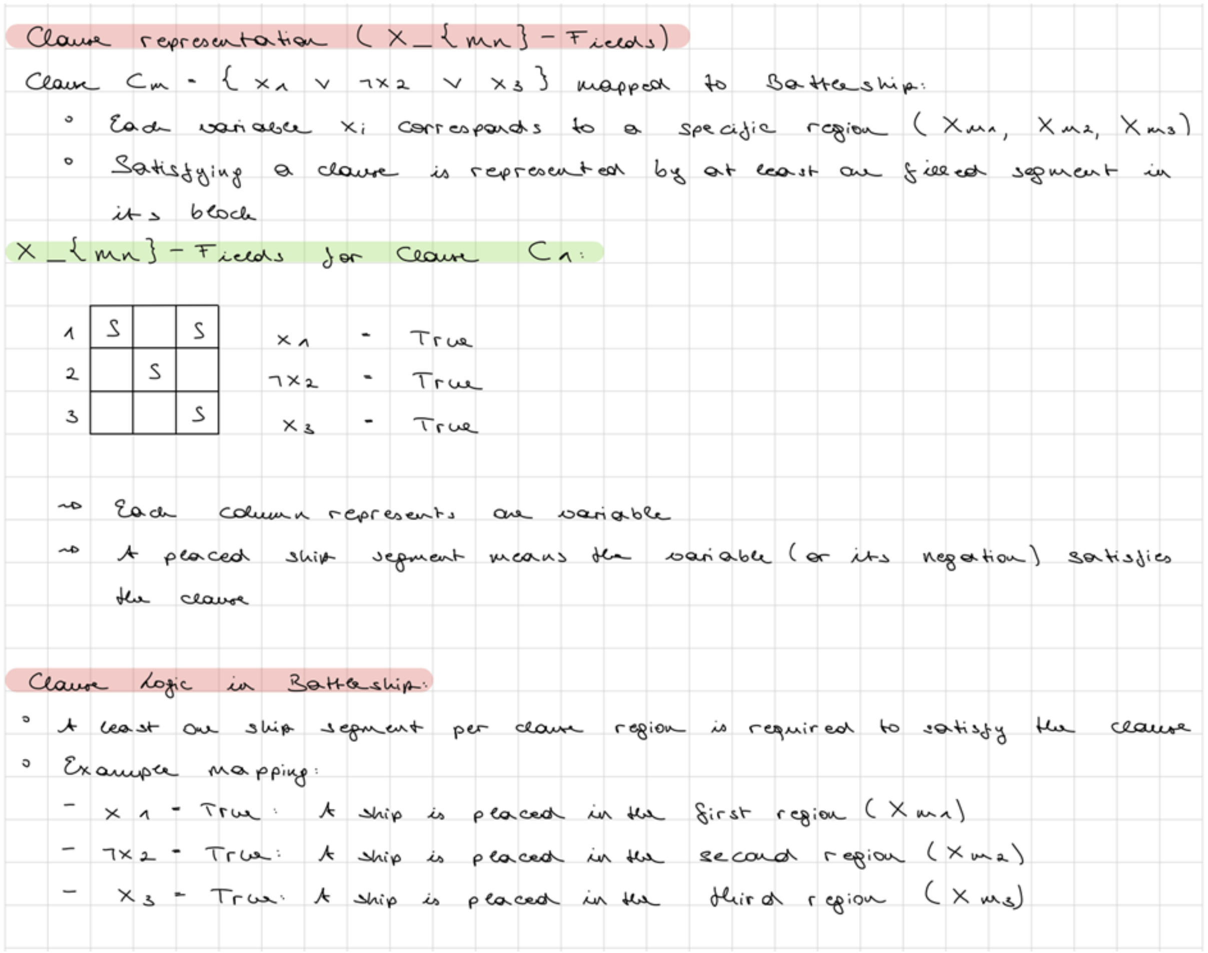
- ◦ A ship segment at the top - True
- ◦ A ship segment at the bottom - False

X - Field for variable x_i:

1	S	(True)
2		
3	S	(False)

- **Clause Blocks (X_{mn}-Fields):** Each occurrence of a variable x_n in clause C_m is represented by a small pattern of cells. Depending on whether x_n appears positively, negatively, or not at all, the pattern of water and ship cells that must be placed will differ.

- **If x_n is positive in clause C_m:** The puzzle's structure forces a ship to appear in a specific position if x_n is chosen to be true, and no ship if x_n is false.
- **If xnxn is negative:** The opposite arrangement of ship placement is enforced.
- **If x_n does not appear in that clause:** The block arrangement is such that no additional constraint is imposed for that particular variable/clause pair.



Clause representation (X_{mn} - Fields)

Clause $C_m = \{ x_1 \lor \neg x_2 \lor x_3 \}$ mapped to Battleship:

- Each variable x_i corresponds to a specific region (X_{m1}, X_{m2}, X_{m3})
- Satisfying a clause is represented by at least one filled segment in its block

X_{mn} - Fields for Clause C_1:

1	S		S
2		S	
3			S

x_1 = True

$\neg x_2$ = True

x_3 = True

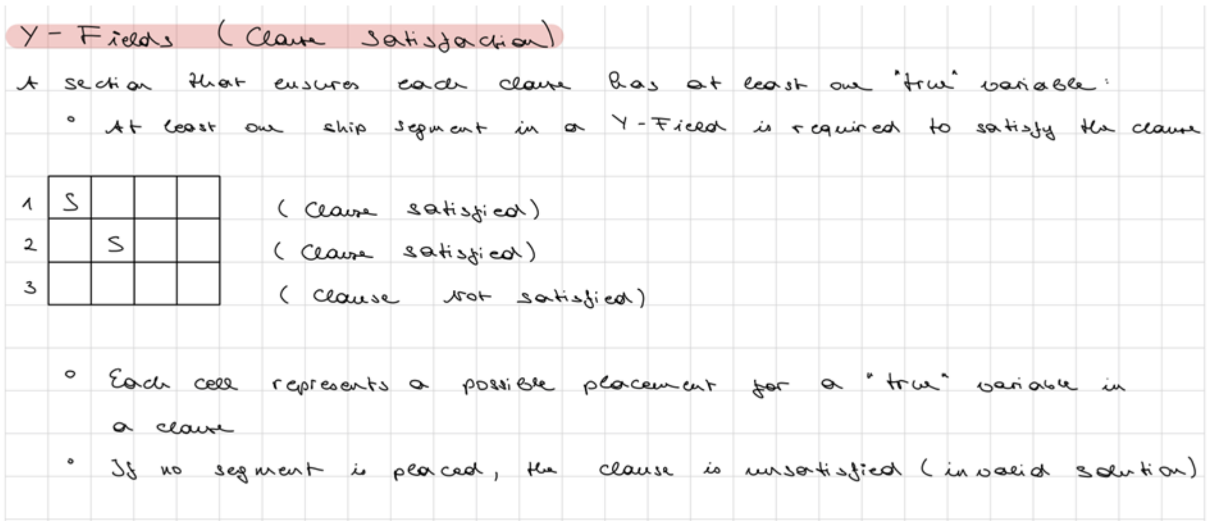
⇝ Each column represents one variable

⇝ A placed ship segment means the variable (or its negation) satisfies the clause

Clause Logic in Battleship:

- At least one ship segment per clause region is required to satisfy the clause
- Example mapping:
  - x_1 = True: A ship is placed in the first region (X_{m1})
  - $\neg x_2$ = True: A ship is placed in the second region (X_{m2})
  - x_3 = True: A ship is placed in the third region (X_{m3})

- **Y-Fields:** These are auxiliary constructs to enforce certain global conditions, such as ensuring that for each clause at least one variable evaluates to true. By cleverly arranging ships and water cells, the Y-fields help ensure that if no variable in a clause can produce a “true” signal (i.e., no corresponding ship placement), the puzzle becomes unsolvable.



Y - Fields (Clause satisfaction)

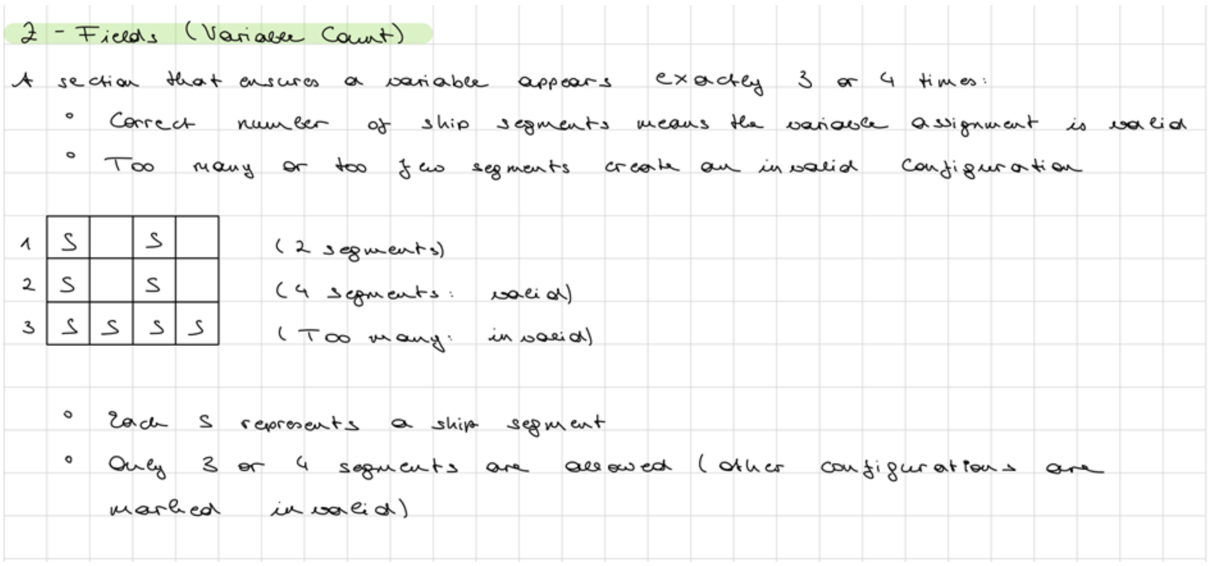
A section that ensures each clause has at least one "true" variable:

- At least one ship segment in a Y-Field is required to satisfy the clause

1	S				(Clause satisfied)
2		S			(Clause satisfied)
3					(Clause not satisfied)

- Each cell represents a possible placement for a "true" variable in a clause
- If no segment is placed, the clause is unsatisfied (invalid solution)

- **Z-Fields:** Since each variable appears either 3 or 4 times, the Z-fields are used to control and check the exact number of occurrences. They encode the condition that a variable must appear a certain number of times and link the corresponding ship placements together so that a consistent truth assignment is required to achieve a valid Battleship solution.



Z - Fields (Variable Count)

A section that ensures a variable appears exactly 3 or 4 times:

- Correct number of ship segments means the variable assignment is valid
- Too many or too few segments create an invalid configuration

1	S		S		(2 segments)
2	S		S		(4 segments: valid)
3	S	S	S	S	(Too many: invalid)

- Each S represents a ship segment
- Only 3 or 4 segments are allowed (other configurations are marked invalid)

11 Reading the Configuration (Example Cases)

11.1

- **Variable Representation (X-Fields):**
  Consider a vertical strip representing a single variable x_n. This strip has predefined water cells (gray) and free cells (white). Depending on how you place a ship segment in these free cells, the variable is effectively set to true or false. The puzzle's solution will require a placement consistent across all occurrences of x_n in different clauses.
- **Clause Representation (Involving X_{mn}-Fields):**
  Each clause C_m is associated with a pattern of sub-blocks $X_{m1}, X_{m2}, \ldots, X_{mn}$ that correspond to each variable. If the clause must be satisfied, then at least one of these sub-blocks must allow a ship placement consistent with a "true" assignment for a variable that appears in the clause. If no such placement is possible, the clause remains unsatisfied, and thus no solution to the Battleship puzzle is possible.
- **Y-Fields and Z-Fields:**
  The Y-fields may, for example, implement a "check" so that for each clause there is at least one segment placed to represent a true variable. If no variable can meet this requirement, the configuration of Y-fields cannot be completed according to the Battleship rules.
  The Z-fields handle the constraint that each variable appears 3 or 4 times. Depending on whether a variable must appear 3 or 4 times, these fields impose counting constraints that can only be met if the chosen truth assignments (and thus ship placements) across the board are consistent.

12 Conclusion of the Reduction

By constructing the Battleship board, the initial placements (both ship and water), and the constraints on rows, columns, and the fleet composition in a way that mirrors a given (3,3,(4))-SAT formula, we form a Karp reduction. This means:

- If the (3,3,(4))-SAT formula is satisfiable, then there is a way to place the ships according to our encoding so that all Battleship constraints are met.
- If the formula is not satisfiable, no valid Battleship solution exists that meets all constraints.

Because (3,3,(4))-SAT is known to be NP-complete, this reduction proves that the Battleship decision problem is also NP-complete.