

Gridsweeper: NP-Completeness of Variants to the Minesweeper Solvability Problem

Tommy Schneider

January 3, 2018

1 What is Minesweeper Solvability?

1.1 Traditional Minesweeper

Minesweeper Solvability is a decision problem based on the classic game of Minesweeper. The rules of Minesweeper can be found many places online [1]. We'll begin by constructing a $M \times N$ Minesweeper board.

Let G be the graph $G = (V, E)$ where

$$V = \{(i, j) \mid 0 \leq i < M \text{ and } 0 \leq j < N\}$$

and

$$E = \{((i, j), (i', j')) \mid (i, j) \in V; \quad (i', j') \in V; \quad (i, j) \neq (i', j'); \\ i' - i = -1, 0, \text{ or } 1; \quad j' - j = -1, 0, \text{ or } 1\}.$$

A *boardstate* of the game Minesweeper is defined as a function

$$\varphi : V \rightarrow \{0, 1, 2, 3, 4, 5, 6, 7, 8, *, \square\}$$

where $\varphi(v) = *$ represents that a vertex v has a mine on it, a $\varphi(v) = x$ represents that the vertex v has been revealed and that it has x mines on adjacent vertices, and $\varphi(v) = \square$ represents that that v has not yet been revealed (and thus may or may not have a mine on it). Let \mathcal{B} be the set of boardstates.

We'll call a boardstate φ a *full* boardstate if φ never maps to \square . A full boardstate is called *valid* if every vertex with a number x on it has exactly x adjacent vertices with mines on them.ⁱ A full boardstate is otherwise called *invalid*.

ⁱA subtlety here that we'll make more clear later is that we do not require there to be a certain number of mines to be present on a full boardstate for it to be valid (per the rules of traditional Minesweeper). This turns out not to matter for sufficiently large boards.

The Minesweeper Problem is as such: Given a boardstate φ , does there exist a replacement of blank spaces (\square) with mines ($*$) and numbers ($x \in \{0, \dots, 8\}$) that yields a valid full boardstate?

The NP-Completeness of this problem was shown by Richard Kaye in the year 2000 [2]. The technique Kaye uses is SAT reduction via the construction of ‘circuits’ on the Minesweeper board. Proving that the solvability of Minesweeper is NP-Complete shows that there is no simple solution to the game. This ensures the game has no simple solution and means it is more likely to be interesting for human players.

1.2 Our Variation: Minesweeper on a Grid

While Minesweeper is often described as being played on a grid, this is actually not true (at least based on our representation of the game). As can be in the previous section, the graph underlying the traditional version of Minesweeper is actually played on a grid *with the inclusion of diagonals*. This is because ‘clicks’ in Minesweeper potentially reveal information about the number of mines across corners as well across lines.

We can easily generalize the game of Minesweeper, along with the Minesweeper problem, to other classes of graphs. To do this, we must not only replace our class of graphs, but also the range of our boardstate functions to include the numbers 0 through the max degree of the graph.

The main set of graphs we’ll be considering in this paper is $M \times N$ grids. That is, graphs $G = (V, E)$ where

$$V = \{(i, j) \mid 0 \leq i < M \text{ and } 0 \leq j < N\}$$

and

$$E = \{((i, j), (i', j')) \mid (i, j) \in V; (i', j') \in V; (i', j') \in \{(i+1, j), (i-1, j), (i, j+1), (i, j-1)\}\}.$$

Since the maximum degree of any vertex in such a graph is 4, we have that boardstates are functions from V to $\{0, 1, 2, 3, 4, *, \square\}$.

For convenience, we’ll refer to this variant as Gridsweeper. See Figure 1 for examples of possible Gridsweeper boardstates, with the convention that the upper-left corner represents φ of $(0, 0)$ and the bottom-right corner represents φ of $(M-1, N-1)$.

2 Gridsweeper is NP-Complete

Clearly, Gridsweeper Solvability is in NP. Given a full boardstate, it is easy to determine whether the boardstate is valid—we simply verify each vertex’s validity in-

Full and Valid:

*	1	1	1
2	1	*	*
*	2	*	3
1	0	1	*

Full and Invalid:

*	2	1	1
3	1	*	*
*	2	*	1
2	0	1	*

Incomplete:

*	1	1	1
	1	*	
*	2		3
1	0		

Figure 1: Various possible boardstates.



dividually. In order to show that it is NP-Complete, we will construct mock circuits and reduce SAT to Gridsweeper Solvability, much like what Kaye does in his paper.

2.1 Wires, Bends, Splits, and Ends

Before we construct circuits, we must definite what our variables are. Let x be some variable we want to represent and x' to be its negation. Then if there is a mine at square marked x then every square marked x must have a mine and every square marked x' must not have a mine. Similarly, if a square marked x' has a mine then every square marked x' must have a mine and every square marked x must not have a mine. We can consider a square with x to have the value of ‘True’ if there is a mine on it and have the value of ‘False’ otherwise.

Now, we’ll construct ‘wires’, which are sections of a boardstate that look like so:

A Wire:

	1						
1	x	1					
	1	x'	1				
		1	x	1			
			1	x'	1		
				1	x	1	
					1	x'	1
						1	

Note that wires must output what is input into them (that’s what it means to be a wire). Thus, wires *must be odd-length segments* of the described pattern. The arrows indicated the ‘direction’ the wire might go—if there is a mine in the upper-left x then 4 squares to the right and 4 squares down there must be another mine at that x .

Here are implementations of some other useful circuit components:

A NOT Gate:

↘	1						
1	x	1					
	1	x'	1				
		1	x	2			
			2	x	1		
				1	x'	1	
					1	x	1
						1	↘

A Bent Wire:

			1	1	1		
		1	*	*	*	1	
		1	x	2	x	1	
	1	x'	1		1	x'	1
1	x	1				1	x
↗	1					1	↘

A Terminated Wire:

↘	1						
1	x	1					
	1	x'	1				
		1	x	1			
			1	x'	1	1	
				1	x	*	1
				1	*	2	
					1		

2.2 Splitting and Crossing Wires

The next bit of circuitry we will represent is the splitting of a wire. The key to designing such components is to lay out the x, x', x, x', \dots pattern as desired, then to place mines on any square that is adjacent to x and x' in different numbers. This hides the information about whether x or x' has a mine on it from a potential player. At this point, the numbers that must be placed on the remaining squares are completely determined.

A Wire-Splitter:

				↖	1				
				1	x	1			
↘	1			1	*	x'	1		
1	x	1	1	1	x	2	x	1	
	1	x'	*	x'	2	x'	*	x'	1
		1	x	2	x	1	1	1	↘
			1	x'	1				
			1	*	1				
			1						

Note that whether the ends of this circuit are x or x' is irrelevant as the output wires can always reversed via a NOT gate.

It is well-known that we can simulate the crossing of a wire with three XOR gates and wire-splitters. Moreover, XOR gates can be constructed by OR gates, NOT gates, and wire-splitters [2]. This brings us to the next section.

2.3 Creating the OR Gate

Constructing an OR gate is significantly more complex than constructing other gates. In fact, we must rely a bit more heavily details of SATisfability. Namely, a problem in SAT is stated as such: given a boolean formula ϕ with variables x_1, \dots, x_n , does there exist some truth assignment of x_1, \dots, x_n that satisfies ϕ ?

Thus, in constructing an OR gate on two inputs x and y , we are allowed to introduce a finite number of extra variables a_1, a_2, \dots, a_n so long as there exists some truth assignment of a_1, a_2, \dots, a_n if and only if $x \vee y$ is both true. Now, consider the board section (with the key section boxed):

An OR gate:

\swarrow	1			1			1	\searrow
1	x	1	2	*	2	1	y	1
	1	x'	*	s	*	y'	1	
		1	x	2	y	1		
			*	t'	*			
			2	1	1			
		1	*	t	1			
			1	1	t'	1		
					1	\swarrow		

Let's take note of a couple of facts. First of all, it's easy to see that exactly one of the set of squares labeled x or x' has mines on them (same for y and y' as well as t and t'). Say that the square x having a mine on it means that the value of the variable x is true, and the same for y , s , and t . Now, we must show two things:

- (1) If t is true then for all values of s we have that $x \vee y$ is true. $[t \Rightarrow \forall s(x \vee y)]$
- (2) If $x \vee y$ is true then there exists a value of s such that t is true. $[(x \vee y) \Rightarrow \exists s(t)]$

If t is true, then t' has no mine on it. Thus, at least one of x or y must have a mine. Hence, we have (1).

Next, consider the case where both x and y have mines on them. Then letting s have no mine means that t' must have no mine, and hence t must be true. Now consider the case where exactly one of x or y has a mine. Then letting s have a mine on it means that t' cannot have a mine on it. Hence, t is true. This gives us (2).

2.4 Some Bookkeeping

It is well established one can construct any circuit from wires, bends, splits, NOT gates, and OR gates. This allows us to reduce SAT to Gridsweeper solvability.

But before we finish, some bookkeeping needs to be done. Returning to our definition of validity (as defined in section 1.1), one might ask: Why don't we care about the total number of mines on our board? The reason we're able to sweep this question under the rug, so to speak, is that we can designate a section of our board to create pockets of spaces, each of which may or may not contain a mine, and which do not affect any other portion of the board. For example, consider a section of the board looks like this:

*	*	*	*	*	*	*	*	*
*	x_1	*	*	x_2	*	*	x_3	*
*	*	*	*	*	*	*	*	*

There is no way to get any information about whether the squares labeled x_1 , x_2 , and x_3 have mines on them or not. Thus, it doesn't really matter whether we're keeping track of and have a requirement for the total number of mines since we can always construct a section of the board which leaves room for (potential) excess mines.

3 What's Next?

Knowing that Gridsweeper Solvability is NP-Complete gives us evidence that Gridsweeper may be an interesting game to play. Moreover, the proof in this paper gives us some insight into *why* certain variants of Minesweeper may be difficult. Since a player can never get information about the squares surrounding a mine, precise placement of mines on would-be high-information-containing squares can make the game very difficult, yet still possible.

It would be interesting to see when Minesweeper Solvability is NP-Complete for other classes of graphs as well. For example, playing Minesweeper on the class of graphs where each node has degree at most 2 seems like an easier game. What about the class of graphs of degree at most 3—or Minesweeper played on a triangle-grid? These are potential avenues for future research.

References

- [1] Wikipedia: Minesweeper (video game).
- [2] Richard Kaye. Minesweeper is np-complete. *Springer-Verlag*, 22(2):9–15, 2000.