Offline 1-Minesweeper is NP-complete

James D. Fix Brandon McPhail

May 2004

Abstract

We use Minesweeper to illustrate NP-completeness proofs, arguments that establish the hardness of solving certain problems. In particular, we give a polynomial time reduction from boolean circuits to Minesweeper puzzles where each square in the puzzle has at most one mine surrounding it. Our construction encodes circuits as Minesweeper puzzles that look easy to solve, and suggests that solving Minesweeper puzzles off-line *is hard even when the density of mines is low.*

1 Introduction

We consider the difficulty of playing Minesweeper, a popular solitaire computer puzzle. In Minesweeper, the player is presented with a finite, rectangular grid of squares, where each square has a hidden binary state: it either has a mine or is empty. The player reveals the states of squares of his choosing, and with each revelation he is presented with a clue: a count of the number of mines within the square's immediate neighbors. The game ends either when the player correctly determines which squares have mines, in which case the player wins, or when the player attempts to reveal a square that has a mine, in which case he loses.

The reasoning involved in successfully inferring the location of mines, say, when the revealed clues allow the player to determine their location, is believed to be computationally intractable. Indeed, Kaye [5] defines a decision problem MINESWEEPER that characterizes solving Minesweeper puzzles and proves that it is NP-complete. The decision problem he devises is the following: given a partially-played grid with revealed mines and clues, determine whether there is a fully solved grid, that is, an assignment of mines to squares that is consistent with the revealed clues.

Kaye gives a polynomial-time reduction from the circuit satisfiability problem to the MINESWEEPER problem, thus showing that solving Minesweeper puzzles is at least as hard as any NP-complete problem in the formal sense. He shows how to construct portions of a Minesweeper puzzle that behave like logic gates, where certain squares of a puzzle generated by the reduction hold mines if and only if some component of the reduced circuit outputs a 1. His reduction is a good introduction to computational complexity for those unfamiliar with its techniques— boolean circuits are an illustrative model of computation, and the believed difficulty of finding a satisfying assignment for a circuit, in general, is suggested by the known difficulty of solving a popular game.

In the computer version of Minesweeper, the level of difficulty of the puzzles can be increased by increasing the density of the mines. In a similar fashion, one might parameterize puzzle instances of the Minesweeper decison problem, vary these parameters, and ask what parameters make the puzzle difficult to solve and what parameters make general solution tractable. Here, we consider a parameterization somewhat related to density by defining the k-MINESWEEPER problem: determine whether a Minesweeper puzzle has a consistent assignment of mines to squares when the revealed clues are at most k.

It is clear that 8-MINESWEEPER is NP-complete. In fact, in Kaye's circuit-to-puzzle constructions the resulting puzzle squares have at most six surrounding mines showing that 6-MINESWEEPER is NP-complete. Kaye's constructions were modified and tightened [6] to show that 3-Minesweeper is NP-complete. For a time, we were convinced that 1-MINESWEEPER had an efficient decision algorithm. Instead, we have proof of the following:

Proposition 1 1-*MINESWEEPER is NP-complete.*

The proof of this fact results from a reduction of the circuit satisfiability problem to 1-MINESWEEPER.

We sketch our construction of Minesweeper puzzles from boolean circuits in the remainder of this paper. We recommend that the reader be familiar with NP-completeness and NP-completeness reductions to fully understand our results (see [2], for example). However, it is possible to understand our construction, even its technical context, without this background armed with some familiarity with boolean logic. Consider these facts:

1. Circuits compute a boolean output from a set of boolean inputs and can be used to encode any arbitrary computable boolean function over a fixed number of inputs. Thus, they are a sufficiently "universal" model of computation.

2. Determining whether a boolean circuit is satisfiable, that is, can be made to output a 1, is considered to be hard, in general. Though it relates to an open question in theoretical computer science, it is strongly believed that there is no efficient algorithm for determining circuit satisfiability.

3. There is a collection of difficult problems (the NP-complete ones) also lacking efficient algorithms so far. It turns out that, by the universality of circuits, an efficient algorithm for circuit satifiability would give rise to efficient algorithms for these difficult problems.

4. Here, we give an efficient algorithm that converts any boolean circuit into an equivalent Minesweeper puzzle instance. The existence of this algorithm provides proof that Proposition 1 holds.

5. Finally, a corollary of Proposition 1 can be expressed informally: if an efficient algorithm for solving 1-Minesweeper puzzles were found, then the circuit satisfiability, and hence all the NP-complete problems, would have efficient solution, as well.

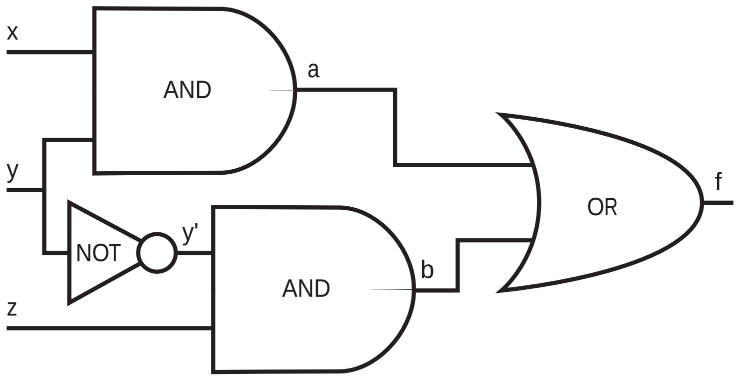


Figure 1: A boolean circuit that computes $f(x, y, z) = (x \wedge y) \vee (\neg y \wedge z)$.

For a better sense of the history you join with such puzzle play, see the classic compendium of NP-completeness problems [3].

We also suggest survey of Kaye's Minesweeper results [5, 4], as our construction simply builds from the ideas of that work. Our constructions are easily verifiable as a result of working out the details of, and trying to simplify, his. Kaye's constructed puzzles are additionally satisfying in that they would actually be difficult to solve in an on-line setting. In contrast, our 1-MINESEEPWR constructions would be easily solvable if they arose in the context of some interactive computer Minesweeper game.

2 Boolean Circuit Satisfiability

Briefly, a boolean circuit consists of a set of logic gates that compute simple boolean functions. Each gate has a set of inputs and, for the sake of this discussion, a single output. We can assign values to a gate's inputs, either 0 or 1, and that gate will output a 0 or 1 based on the boolean function it computes. For example, an AND gate takes two inputs, and outputs a 1 only when both inputs are assigned the value 1. Otherwise, for any other assigned inputs an AND gate outputs 0.

A collection of logic gates can be connected with wires to form a boolean circuit. For example, Figure 1 depicts a simple boolean circuit with four logic gates. Like gates, a circuit has a set of inputs— these are the wires that are only connnected to the inputs of gates. No wire can be connected to more than one output of a gate. It may be connected to more than one input to a gate, in which case it forms a branch. The circuits we consider have a distinct output wire— one that is connected only to the output of a gate. The circuit in Figure 1 takes three inputs and is composed of four logic gates, it has a branching input wire labelled y, and has an output wire labelled f.

If we assign values to the input wires, then a circuit computes an output value in a straighforward manner: when all the inputs to a gate have been assigned a value the gate outputs a value. Thus, in the example circuit, if x and z are assigned the value 1, and y a zero, then the output of the top AND gate, the value on wire a, is 0. Similarly, the value output by the lower AND gate on wire b is 1. As a result, the OR gate gets inputs of 1 and 0, and so the circuit outputs a 1. We say that the assignment $x := 1$, $y := 0$, and $z := 1$ *satisfies* the circuit.

The circuits we consider are *combinatorial*, that is, there is no wire path through the gates that, following outputs to inputs between gates, forms a cycle (there is no feedback in the circuit). We can evaluate the circuit with any assignment of input values to yield a particular output of 0 or 1, and thus the circuit computes a boolean function. The circuit in Figure 1 computes the function $f(x, y, z) = (x \wedge y) \vee (\neg y \wedge z)$ because it is satisfied only when x and y are assigned 1, or when y and z are assigned 0 and 1, respectively.

Using the logic gate components we describe, any boolean function over a fixed number of boolean inputs can be computed by a boolean circuit. Using only NOTs and ANDs is sufficient.

Determining, in general, whether a circuit is satisfiable or whether instead it outputs a 0 for any assigned input values is the *circuit satisfiability problem.* If we were to replace the OR in Figure 1 with an AND, then the circuit would no longer be satisfiable by any assigned set of input values. As of yet, no efficient algorithm is known for determining circuit satisfiability. One could certainly try all possible input assignments to see if any satisfy a circuit, but this could require an exponential number of attempts over the number of inputs.

The circuit satisfiability decision problem is NP-complete [1, 2], suggesting that no efficient algorithm for its general solution exists. If we found such an algorithm or proved that one did not exist, we'd have solved a major open question in mathematics and computer science.

3 Minesweeper Puzzle Satisfiability

In the Introduction, we gave some details of how one might formally define Minesweeper so as to make it possible to formally prove that its general solution is hard. As we did with circuits above, Kaye[5] defined a Minesweeper satisfiability decision problem. Formally,

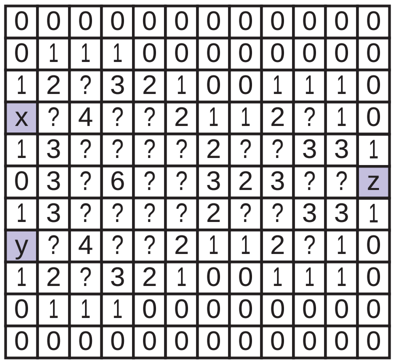
Definition 1 *an* $r \times c$ **Minesweeper instance** *consists of a table* M *with* r *rows and* c *columns where each table entry* $M[i, j]$, *for* $1 \leq i \leq r$ *and* $1 \leq i \leq c$, *is from the set* $\{\boxed{?}, \boxed{\ast}, \boxed{0}, \boxed{1}, \ldots, \boxed{8}\}$. *Let the* **neighborhood** *of [i,j] be*

$$\mathcal{N}_{[i,j]} = \{[i', j'] \mid |i - i'| \leq 1,\ |j - j'| \leq 1,\ 1 \leq i' \leq r \text{ and } 1 \leq j' \leq c\},$$

that is, the squares at most one away from square $[i, j]$. *An instance* M *is* **satisfiable** *whenever there exists a* **mine assignment** *table* A *with entries* $A[i, j] \in \{0, 1\}$ *in which the following hold*

1. $A[i, j] = 1$ *whenever* $M[i, j] = \boxed{\ast}$.
2. $A[i, j] = 0$ *and* $k = \sum_{[i',j'] \in \mathcal{N}_{[i,j]}} A[i', j']$ *whenever* $M[i, j] = \boxed{k}$.

Definition 2 **MINESWEEPER** *is the set of all Minesweeper instances* M *that are satisfiable.*



0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	0
1	2	?	3	2	1	0	0	1	1	1	0
x	?	4	?	?	2	1	1	2	?	1	0
1	3	?	?	?	?	2	?	?	3	3	1
0	3	?	6	?	?	3	2	3	?	?	z
1	3	?	?	?	?	2	?	?	3	3	1
y	?	4	?	?	2	1	1	2	?	1	0
1	2	?	3	2	1	0	0	1	1	1	0
0	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

Figure 2: A Minesweeper puzzle that behaves like an OR gate.

The problem is to determine whether there exists an assignment of mines to squares consistent with the revealed clues of that instance. In other words, devise an algorithm that efficiently determines whether or not $M \in$ MINESWEEPER for any MInesweeper instance M.

As we've noted, MINESWEEPER is NP-complete as it is possible to construct instances that behave like circuits. Figure 2, for example, gives a Minesweeper instance that behaves like an OR gate. Note that square z has a mine exactly when at least one of x and y have mines.

Definition 3 *For* $1 \leq k \leq 8$ *the language* k-**MINESWEEPER** *is the set of Minesweeper instances*

$$\{M \mid \textit{some } A \textit{ satisfies } M \textit{ and } M[i,j] \leq k \textit{ for all squares } [i,j] \textit{ of } M\},$$

that is, those instances where the revealed clues do not exceed k.

In other words, the problem is to determine whether a Minesweeper instance has an assignment where all revealed and potential clues are bounded by k.

A natural question arises: for which k is k-MINESWEEPER NP-complete?

4 1-MINESWEEPER Circuits

The key to our construction is a set of puzzle region tiles that correspond to wires, gates, inputs, and output. Figure 3 shows a 1-Minesweeper puzzle that results from our construction and encodes the same boolean function as Figure 1. It is designed so that the square labelled o can have a mine if and only if squares labelled y and z have mines or squares labelled y' and x have mines. Regions of the puzzle are nearly in one-to-one correspondence with the parts of the circuit. Those regions that correspond roughly to gates are outlined in the figure.

Figure 4 depicts a Minesweeper encoding of a circuit wire, one of the simplest but fundamental components of the construction. The left portion of width four by the bold lines shows a wire tile (a 1-wire) and the entire figure consists of ten of these tiles. For clarity in all of these diagrams, certain squares that designate an unrevealed square whose state corresponds to circuit components that carry a 0/1 value are labelled with variable names. In the actual construction, they would instead be presented as hidden squares $\boxed{?}$. In this case, an assignment to the leftmost square labelled a either has a mine or not. The key to its design is that the square labelled a' to its right, because of the in-between $\boxed{1}$ squares, can be assigned a mine if and only if the a-labelled square is not assigned a mine. The remaining a- and a'-labelled squares behave in a similar way, and so, for example, the rightmost square labelled a can be assigned a mine if and only if the leftmost a-labelled square is assigned a mine.

Wires must bend to connect gates in circuits, and our construction needs to bend 1-wires to route between our gate-like components. Figure 5 illustrates the simplest way to put a bend in a layout of 1-wire tiles, using a 1-bend tile. We have found other bends, for example, a 90° elbow, but the 1-bend is sufficient for the construction. Again, if the leftmost a-labelled square in the figure is assigned a mine, then a satisfying assignment must assign a mine to the rightmost a-labelled square, also.

Figure 6 depicts a 1-branch, allowing our construction to route an input or the output of a gate to multiple gate inputs. Note that its design is not that distinct from a 1-wire.

The key component for the construction, shown in Figure 7, is a NAND 1-gate. A NAND logic gate is equivalent to the composition of an AND gate whose output is fed into a NOT gate— it outputs 0 exactly when both its inputs are assigned the value 1. The key to our 1-gate's design is the center set of squares labelled x, y, and z. If a and b are both are mined (a' and b' are both unmined) then one of x and c' must have a mine, and one of y and c' must have a mine. Squares x and y cannot both be mined because of the center, so c' must be mined and c is then unmined. If instead at least one of a' and b' is mined, then c' must be unmined and c mined. This is exactly the logic of a NAND gate. We leave the reader to verify that there is an assignment to the unspecified squares for all four possible mine assignments to a and b.

We can easily build OR and AND 1-gates from the NAND 1-gate by the following identities:

$$\neg x \equiv \neg(x \wedge x), \tag{1}$$

$$x \wedge y \equiv \neg(\neg x \wedge \neg y). \tag{2}$$

Thus, a NOT 1-gate can be constructed from a NAND 1-gate and a 1-branch, and an OR 1-gate can be constructed from three NOTs and a NAND.

Finally, we need to cap the "loose" 1-wire ends in the 1-circuits. Figure 8 gives the input and output 1-caps needed to complete the puzzle construction. The cap on the right of the figure forces the output 1-wire to have a mine for any of its satisfying assignments, and the cap on the left allows any mine assignment for the input 1-wires. As a result, a mine assignment that satisfies the 1-circuit puzzles of our construction exists exactly when the original circuit has a satisfying truth assignment.

There are further details in the construction, namely proof that, when given any circuit, the Minesweeper puzzle can be output in polynomial time (polynomial in the size of that circuit) but we leave these details for the reader.

References

[1] Stephen Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[2] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, Cambridge, Massachusetts, 2nd edition, 2001.

[3] M.R. Garey and D.S. Johnson. *Computers and Intractability. A guide to the theory of NP-Completeness.* W.H. Freeman and Company, 1979.

[4] Richard Kaye. Richard kaye's minesweeper page. `http://web.mat.bham.ac.uk/R.W.Kaye/minesw/minesw.htm`.

[5] Richard Kaye. Minesweeper is NP-complete. *The Mathematical Intelligencer*, 22(2):9–15, 2000.

[6] Brandon McPhail. Some NP-complete aspects of combinatorial puzzles. Undergraduate thesis, Reed College, Mathematics Department, December 2003.

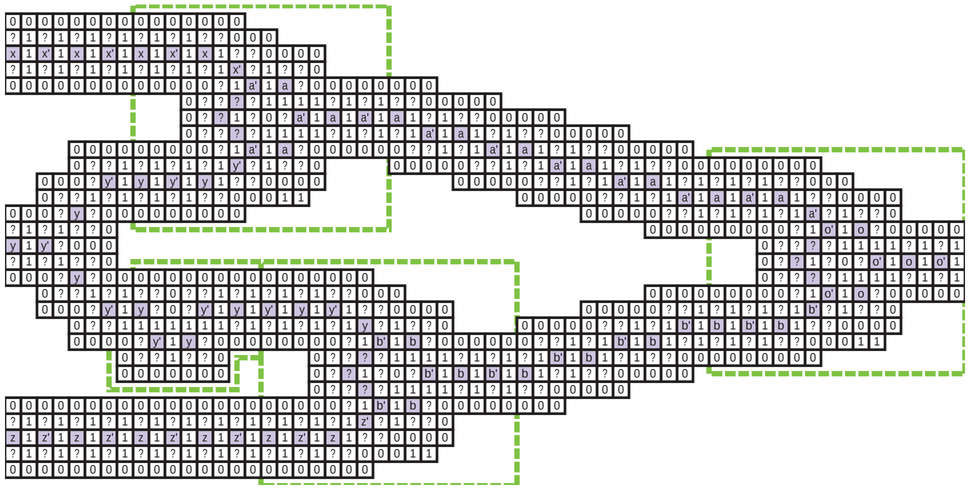


Figure 3: A Minesweeper puzzle that encodes the same boolean function.

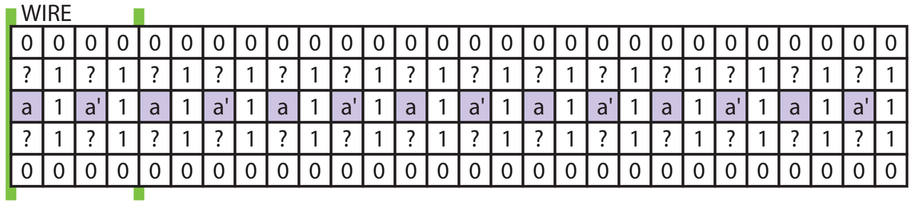


Figure 4: A set of Minesweeper 1-wire tiles that encode a circuit wire.

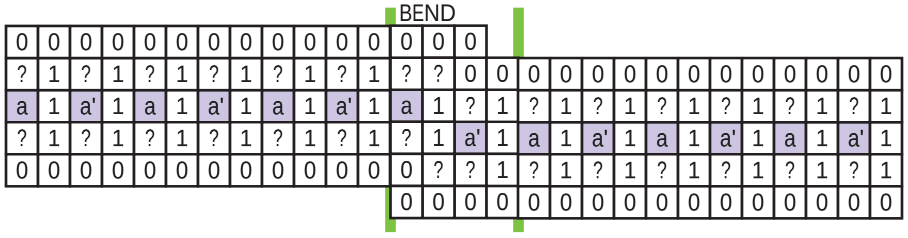


Figure 5: Bending a set of 1-wires with a 1-bend.

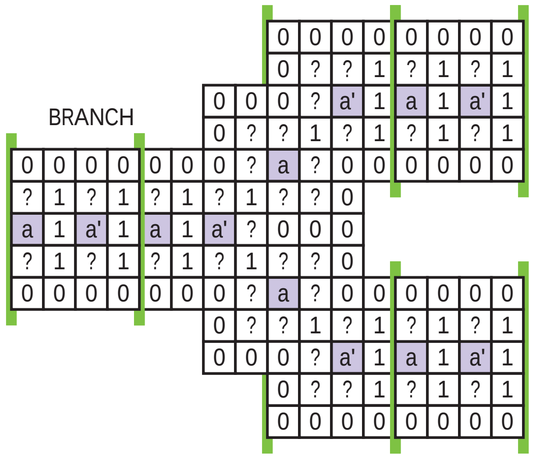


Figure 6: A 1-branch

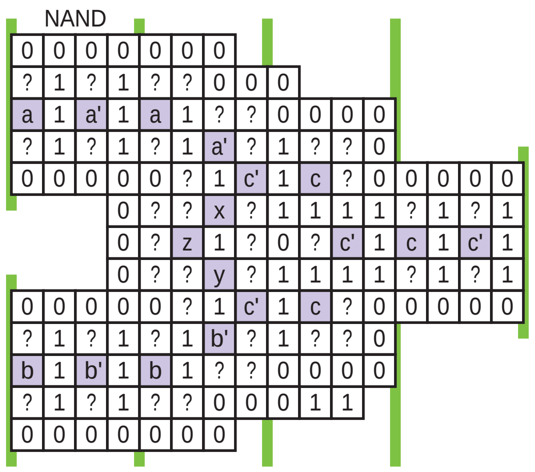


Figure 7: A NAND (NOT-AND) 1-gate.

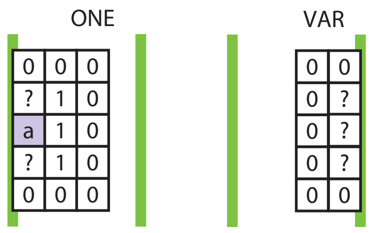


Figure 8: Wire caps.