The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

Minesweeper with Limited Moves

Serge Gaspers
UNSW Sydney and
Data61, CSIRO
Australia

Stefan Rümmele
UNSW Sydney and
University of Sydney
Australia

Abdallah Saffidine
UNSW Sydney, Australia

Kevin Tran
UNSW Sydney, Australia

Abstract

We consider the problem of playing Minesweeper with a limited number of moves: Given a partially revealed board, a number of available clicks k, and a target probability p, can we win with probability p. We win if we do not click on a mine, and, after our sequence of at most k clicks (which reveal information about the neighboring squares) can correctly identify the placement of all mines. We make the assumption, that, at all times, all placements of mines consistent with the currently revealed squares are equiprobable.

Our main results are that the problem is PSPACE-complete, and it remains PSPACE-complete when p is a constant, in particular when $p = 1$. When $k = 0$ (i.e., we are not allowed to click anywhere), the problem is PP-complete in general, but co-NP-complete when p is a constant, and in particular when $p = 1$.

Introduction

Minesweeper is a puzzle game where the player is initially presented with a blank grid of squares. Some squares contain a mine, while others do not. The player can click on a square to reveal information about the grid. If the clicked square is a mine, the player instantly loses. Otherwise, a number from 0 to 8 is revealed, indicating the number of immediately adjacent squares that contain a mine. The goal of the game is to determine the location of all the mines without being blown up.

Traditionally, the game ends once the player has revealed every square that is not a mine. However we consider a slight variant where the player instead can, at any time, state for each square whether there is a mine there. The player wins if they are right, otherwise they lose.

A natural question is: given a partially revealed board, does the player have a strategy that wins with probability p? This question has been studied by de Bondt (2012). We will consider the variant which combines this question of winning probability with a restriction on the number of times the player can click. We assume, at any time, that each placement of mines that is consistent with the partially revealed board is equiprobable.

We consider three questions about winning probability (de Bondt (2012) considers the latter two).

Table 1: Complexity for Minesweeper problems when varying the number of clicks and probability of winning. The results labeled * were obtained by de Bondt (2012).

Clicks	Probability		
	1	constant	input
0	co-NP-c	co-NP-c	PP-c
unbounded	$\in \mathbf{\Delta_P^2}$	PP-h*	PSPACE-c*
input	PSPACE-c	PSPACE-c	PSPACE-c

- Can the player win for certain (with probability 1)?
- Can the player win with probability C, for some constant $C \in (0, 1)$?
- Can the player win with probability p, where p is part of the input?

We consider four restrictions on the number of clicks the player can make:

- The player may click 0 times.
- The player may click as many times as they want (N times, where N is the number of squares).
- The player may click k times, where k is part of the input.

This leads to 9 combinations of problems. We mainly focus on the 6 variants where the number of clicks is 0 or part of the input, and our results are summarized in Table 1.

Related work

Kaye (2000) established that the Minesweeper Consistency problem is NP-complete. This is the problem to decide whether, for a partially revealed board, there is a placement of mines consistent with the board. This result is among the most well-known NP-completeness results, and helped popularize the famous P vs NP problem (Stewart 2000). Fix and McPhail (2004) showed that Minesweeper Consistency remains NP-complete when each cell is surrounded by at most 1 mine. Pedersen (2004) proved that checking whether a partially revealed board has a unique solution is DP-complete, as is checking whether a move is safe. In that paper, a move is a click or a placement of a flag indicating that the cell contains a mine and the input is not guaranteed

to be a consistent Minesweeper board. He also showed that computing the number of consistent placements of mines is #P-complete. Heinrich (2006a) extended Minesweeper to graphs. The complexity of the problem for restrictions of these graphs has been studied in several papers (Golan 2011; Heinrich 2006a; 2006b). de Bondt (2012) showed that the Minesweeper Consistency problem is also NP-complete for hexagonal and triangular grids, as well as for square boards where only a single cell is initially uncovered. Further, de Bondt (2012) analysed the complexity of actually playing Minesweeper, assuming, at every step that all remaining consistent configurations of mines are equiprobable. For winning with constant probability, he established that Minesweeper is PP-hard, and for winning with a probability given in the input, the problem is PSPACE-complete. Here, an unbounded number of clicks is allowed.

Computational techniques to compute the probability that a cell has a mine or to compute the next best move include constraint programming (Bayer, Snyder, and Choueiry 2006; Becerra 2015; Collet 2005; Studholme 2000), Bayesian networks (Vomlelova and Vomlel 2009), and reinforcement learning (Nakov and Wei 2003). Kadlac (2003) gave an algorithm enumerating all consistent placements of mines on an $n \times n$ board in time $O^*(2^{n^2})$. She also showed that the Minesweeper Consistency problem is polynomial-time solvable for $1 \times n$ grids, and Hu and Lin (2007) extended this result to $2 \times n$ grids.

Preliminaries

Propositional Logic. We consider propositional logic over some fixed universe U of propositional atoms and standard connectives $\vee$, $\wedge$, and $\neg$. A literal is an atom x or a negated atom $\neg x$. A clause is a disjunction of literals. A formula in conjunctive normal form (CNF) is conjunction of clauses. An interpretation or assignment is a set $I \subseteq U$ containing all variables set to true. If an interpretation I satisfies a formula ϕ, we call I a model of ϕ. The formula ϕ is called satisfiable if there exists a model of ϕ. The NP-complete problem SAT takes as input a formula ϕ in CNF and asks whether ϕ is satisfiable. A quantified boolean formula (QBF) is an expression of the form $\exists x_1 \forall x_2 \exists x_3 \dots Q_n x_n \phi$, where Q_n is $\exists$ if n is odd and $\forall$ otherwise and ϕ is a propositional formula over atoms $\{x_1, \dots, x_n\}$. The PSPACE-complete QBF problem takes as input a QBF formula ψ and asks whether ψ is true.

Minesweeper. A Minesweeper board consists of an $m \times n$ grid of squares. The neighbourhood of a square p contains all squares of Chebyshev distance 1, that is all squares that are at most 1 row and at most 1 column away from p. If p is not located at the boarder of the grid, its neighbourhood has cardinality 8. Each square either contains a mine or it does not. Additionally, each square can be in one of two states, either revealed or hidden (blank). A revealed square containing a mine displays this information. A revealed square that does not contain a mine shows a number from 0 to 8. This number corresponds to how many of its neighbours contain a mine. When the player clicks on a hidden square, it becomes revealed. If it shows a mine, the player loses. We say a board is partially revealed, if it contains revealed and hidden squares. If a partially revealed board is given as problem input, we assume that none of the revealed squares shows a mine. A solution candidate to a Minesweeper board is the location of all mines, that is mapping from hidden squares to true or false. The player wins by identifying the location of all mines, that is giving a solution candidate that matches the real location of mines. The (x, y)-MS decision problem gets as input a partially revealed board and the question is if the player can identify the location of all mines with probability at least y, using no more than x clicks. The probability factor comes into play when guessing a solution candidate as well as clicking on hidden squares. For example, $(N, 1)$-MS asks the player to win for certain, making as many clicks as they would like. $(\text{input}, 1)$-MS (a slight abuse of notation) asks the player to win for certain, given some number of clicks part of the input.

$(0, C)$-MS

$(0, C)$-MS
Input: A partially played Minesweeper grid.
Output: Is it possible to determine the location of all the mines with probability C, with no clicks?

Since the player can make no clicks, the best they can do is to report an arbitrary assignment of mines to the hidden cells that is consistent with the partially observed grid. This gives the player a $1/s$ chance of winning, where s is the number of different consistent assignments of mines to the grid. So, the problem is equivalent to answering: are there at most $s = \lfloor 1/C \rfloor$ different consistent assignments? This problem is in co-NP, since a NO-instance can be verified by giving $s + 1$ consistent assignments. To show co-NP-hardness, consider the following related problem:

FEWER-THAN-k-SSAT
Input: A **satisfiable** formula ϕ.
Output: Does ϕ have fewer than k satisfying assignments?

Lemma 1. FEWER-THAN-k-SSAT *is co-*NP*-hard, for any k with $k \geq 2$.*

We note that FEWER-THAN-1-SSAT is trivial since the input formula is satisfiable by assumption.

Theorem 2. $(0, C)$-MS *and* $(0, 1)$-MS *are co-*NP*-complete.*

Proof. We have argued co-NP-membership above. To show hardness, we reduce FEWER-THAN-k-SSAT to $(0, C)$-MS. Kaye (2000) shows how to embed a SAT formula onto a Minesweeper grid as a Boolean circuit, such that each valid placement of mines corresponds uniquely to a satisfying assignment. We can use the same construction here to embed an instance of FEWER-THAN-k-SSAT onto the grid (we require that the formula have at least one satisfying assignment to make a valid Minesweeper game, which FEWER-THAN-k-SSAT promises). $(0, \frac{1}{k-1})$-MS is a YES-instance if and only if there are fewer than k satisfying assignments.

This shows that $(0, C)$-MS is co-NP-hard, and thus co-NP-complete for any C of the form $1/k$, $k \in \mathbb{Z}^+$. In particular, this also holds for $k = 1$. Hence, $(0, 1)$-MS is co-NP-complete as well.

With no clicks, the probability of winning is always of the form $1/k$, $k \in \mathbb{Z}^+$, since given no clicks, the game amounts to guessing one of the possible assignments of mines that satisfy the revealed numbers. So it is not necessary to consider other values of C. □

$(0, \text{input})$-MS

$(0, \text{input})$-MS
Input: A partially played Minesweeper grid and probability C.
Output: Is it possible to determine the location of all the mines with probability C, with no clicks?

To show that this problem is PP-complete, consider the following problem:

THRESHOLD-SSAT
Input: A satisfiable formula ϕ and an integer k.
Output: Does ϕ have at least k satisfying assignments?

Lemma 3. THRESHOLD-SSAT *is* PP*-complete.*

Theorem 4. $(0, \text{input})$-MS *is* PP*-complete.*

Proof. An instance of THRESHOLD-SSAT can be embedded onto a Minesweeper grid as a Boolean circuit using the technique of Kaye (2000). $(0, \text{input})$-MS is a NO instance with this grid and winning probability $C = \frac{1}{k-1}$ if and only if the formula has k satisfying assignments. Since PP is closed under complement, this shows that $(0, \text{input})$-MS is PP-hard.

To show that it is in PP (and so PP-complete), we reduce the other way. Let each non-revealed square in the grid be a variable, where an assignment of true indicates that the square contains a mine (false indicates it does not). The formula is satisfied only if the placement of mines represented by the variables fits all the revealed numbers. W.l.o.g. let a revealed square contain the number i and let $v_1, \ldots, v_m$ be the variables representing the neighbours of this revealed square. We need a subformula that expresses that exactly i of the variables $v_1, \ldots, v_m$ are set to true. Such cardinality constraints can be encoded in polynomial size using for example the methods of Sinz (2005). The number s of possible satisfying assignments of the resulting formula is equal to the number of mine placements that satisfy the partial grid. Furthermore, the probability of winning without clicks is $\frac{1}{s}$. Hence, let $k \in \mathbb{Z}^+$ be the smallest integer such that $C \leq \frac{1}{k}$. We can win with probability $\geq C$ if and only if the formula has fewer than $k + 1$ satisfying assignments, that is the formula together with $k + 1$ is a NO instance of THRESHOLD-SSAT. □

$(\text{input}, 1)$-MS

In this section, we prove that playing Minesweeper with winning probability 1 and a limited number of clicks given as input is intractable. More precisely, we show that the following decision problem is PSPACE-complete.

$(\text{input}, 1)$-MS
Input: A partially played Minesweeper grid and an integer k.
Output: Is it possible to determine the location of all mines with probability 1 in k clicks?

We defer membership in PSPACE to Theorem 8 where we show PSPACE-membership of the more general problem $(\text{input}, \text{input})$-MS. For hardness, we first consider a variant played on digraphs and show that it is PSPACE-hard via a reduction from QBF. Then we give a reduction from the digraphs variant to the Minesweeper grid of $(\text{input}, 1)$-MS.

PSPACE-hardness of $(\text{input}, 1)$-MS on a digraph

To show PSPACE-hardness, we first consider a variant of Minesweeper generalised to digraphs, and then reduce from this. In this generalization, the board consists of vertices representing squares that can again contain mines and might be hidden or revealed. The neighbourhood of a vertex is defined via directed edges. When clicking on a hidden vertex that does not contain a mine, the vertex displays a number corresponding to the number of outgoing edges that point to vertices that contain a mine.

MINESWEEPER ON A DIRECTED GRAPH (MSDG)
Input: A partially revealed directed graph Minesweeper representation and an integer k.
Output: Is it possible to determine the location of all mines with probability 1 in k clicks?

To show that this problem is PSPACE-hard we will reduce from QBF.

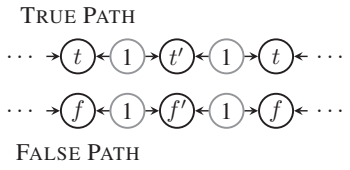
Dual Rail Logic We now construct an alternative variant of wires and logic gate gadgets, which we use to represent the QBF formula. Firstly the wire consists of two paths of vertices connected as in Fig. 1a. It is clear that in each path, either every second unknown vertex is a mine, or every other one is. If one knows whether or not a particular vertex in one of the paths contains a mine, then one can infer the state of all the vertices in that path. Denote one of the paths in the wire as the *True* path and denote the other the *False* path.

- If the player knows about the state of the vertices in the *True* path, then the wire transmits true.
- Likewise if the player knows about the state of the vertices in the *False* path, then the wire transmits false.
- If the player knows the state of neither path, then the wire does not transmit any value.
- We will ensure in our construction that no winning strategy can know the state of both paths at the same time.

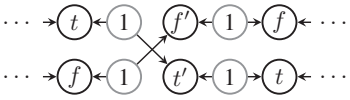
Constructing a NOT gate and a wire splitter is fairly straightforward (Fig. 1b and 1c).

Lastly, we need to construct an AND gate (combining it with the NOT gate) before we can represent every formula. To do this, we need another gadget.

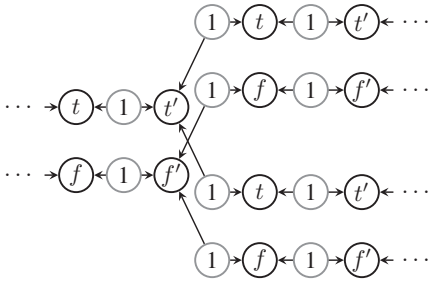
Force-click gadget We would like to restrict where the player can click to make the player follow our construction as planned. So we create a gadget that forces the player to click at least once in a subset S of vertices.



(a) A dual rail wire gadget. In the *True* path all the t are mines, or all the t' are mines. Likewise for the *False* path with respect to f and f'.



(b) Dual Rail NOT gate.



(c) Dual Rail Wire Splitter.

Figure 1: Dual Rail logic.

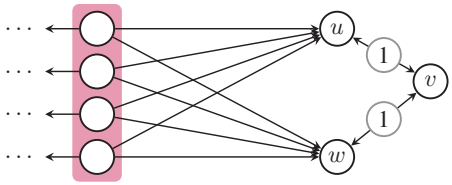


Figure 2: In order to win with certainty, the player must click one of the highlighted vertices (representing the set S).

The construction in Fig. 2 goes as follows: create three vertices u, v, and w. The pair u, v as well as the pair v, w is pointed at by a vertex with value 1. So either u and w both contain a mine or v does. For each vertex in S, create an edge from it that points to u and one that points to w.

The player cannot click u, v, or w directly, since either of them could contain a mine. So any strategy that wins all the time must click one of the vertices in S.

S may have edges to other vertices, so we must be careful to make sure that this construction does not interfere with the information the player would normally gain from clicking vertices in S. We will ensure that in our constructions, every vertex that is unknown has outdegree at most 1, with the exception of vertices belonging to a force-click set, which have outdegree 3. Since the status of u and w coincides, information revealed by vertices in S allows us to pinpoint the locations of the mines at u, v, and w. If the vertex from S reveals a number that is at most 1, then v contains a mine, otherwise u and w contains mines. We will set k to be exactly the number of force-click sets to make sure that the player must guess exactly one from each set.

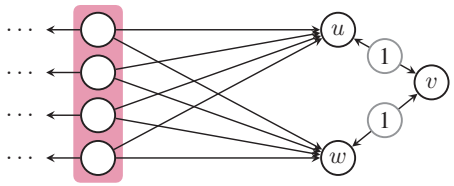


Figure 3: XOR gate if we take "having a mine" to indicate true: x is a mine only when either of t_1 and t_2 is a mine but not both.

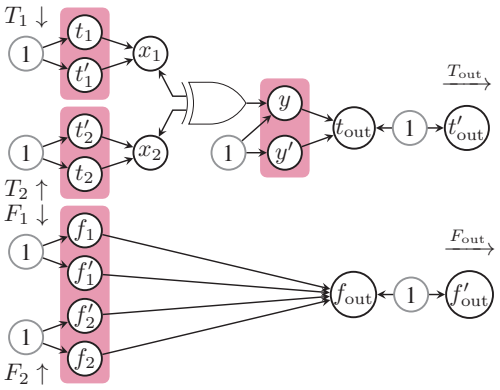


Figure 4: AND gate gadget. The XOR node is the one described in Fig. 3, with x_1 and x_2 being the inputs and y being the output.

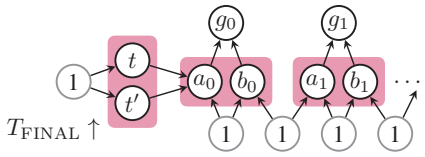
Dual Rail AND gate We do this by first constructing a XOR gadget ⊻ as in Fig. 3. Now, we can construct an AND gate. Let the true path for the first (second) input to the AND gate be named T_1 (T_2) and the false path F_1 (F_2). Denote the output wire's true (false) path to be T_{out} (F_{out}). Figure 4 shows how to construct an AND gate. The player is forced to click exactly one vertex in each purple box by the force-click gadget.

The state of T_{out} can only (safely) be determined if the player knows the state of T_1 and T_2. The XOR gadget from Fig. 3 ensures that the state of T_{out} can only be determined if the player knows both the state of T_1 and T_2.

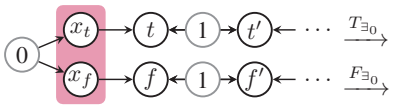
The state of F_{out} can be determined if either F_1 or F_2 are known. If F_1 (F_2) is known, the player can infer which of f_1 (f_2) and f_1' (f_2') does not contain a mine and can click it to determine the state of F_{out}.

It is not possible to figure out the state of both T_{out} and F_{out} without guessing.

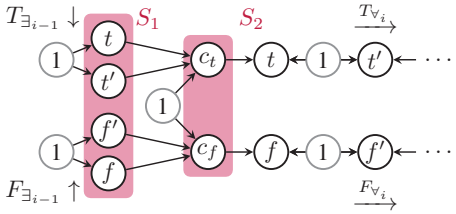
Existential & Universal Qualifiers Now we need to construct gadgets for universal and existential choices. We can assume without loss of generality that the quantifiers are alternating and start with an existential one. Fig. 5b shows how



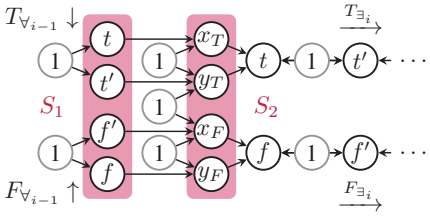
(a) Winning gadget.



(b) First existential quantifier. Clicking on x_t (x_f) corresponds to existentially setting the first variable to true (false).



(c) Revealing universal choices.



(d) Existentials after the first one.

Figure 5: Construction for the winning gadget and for the quantifier gadgets.

to construct the first existential quantifier. Since the player can click only one of x_t and x_f, they can reveal the state of only one of the paths $T_{\exists_0}$ and $F_{\exists_0}$. Hence, this choice corresponds to setting the truth value of the first variable.

Once the player has chosen the value of existential variable $\exists_{i-1}$, the construction in Fig. 5c can be used to reveal the universal variable $\forall_i$. The player clicks the vertex in the set S_1 which they know to be safe (which is only the case after selecting $\exists_{i-1}$). This reveals which vertex is safe in S_2. If it is c_t (c_f), the player can reveal the state of path $T_{\forall_i}$ ($F_{\forall_i}$).

The gadget for existential choices ($\exists_i$) after the first one ($i > 0$), is shown in Fig. 5d. In this gadget, either both the vertices labelled x_T and x_F contain mines (and y_T, y_F do not), or the vertices labelled y_T and y_F contain mines (and x_T, x_F do not). The player can click the vertex of S_1 that they know to be safe (after revealing the previous universal), which reveals the placement of mines in S_2. If the player wishes to set $\exists_i$ to true (false), then they should click x_T or y_T (x_F or y_F), whichever is safe.

Each quantified variable feeds into the next so that the player must select them in order. Each variable also splits off into a circuit that represents the formula, made of the constructions previously described. If the circuit is satisfied, then the player can infer the state of the true path in the final output wire, T_{FINAL}. We would like this to lead into a winning gadget.

Winning Gadget For every vertex g_i that is initially unknown in our construction so far (excluding the vertices in the force-click gadget) create a pair of vertices a_i and b_i, both of which point to it. The a_i and b_i are connected to each other as shown in Fig. 5a. This construction ensures that either all of the a_i are mines (and the b_i are not) or all the b_i are mines (and the a_i are not). Knowing the state of path T_{FINAL} will allow the player to determine which is the case, allowing them to use this gadget (clicking each of the a_i or b_i) to find out the state of all the remaining unknown vertices. If the player does not know the state of T_{FINAL} (i.e., they have not satisfied the circuit), then they cannot determine which vertices have mines. The player cannot win without access to this gadget since they must at some point determine the state of the a_i and b_i vertices.

Lemma 5. MSDG *is* PSPACE*-hard.*

Proof. The construction above reduces an instance of QBF in polynomial time to an instance of MSDG. The player has a 100% winning strategy if and only if the QBF can be satisfied, showing PSPACE-hardness. □

Converting back to the grid

Kaye (2000) showed how to encode an instance of SAT into a Minesweeper grid. For our conversion, we will use Kaye's wire and wire splitter gadget. The core part of a wire for variable x is a sequence of unknown cells labelled alternately x and x'. These have the property that either all x or all x' cells contain a mine. Each unknown vertex y in the original graph will be represented as a wire on the grid. If the wire transmits *true* (i.e., when each y cell contains a mine), then the corresponding vertex has a mine in it. If the wire transmits *false* (each y' contains a mine), then the corresponding vertex does not have a mine. Each vertex in our gadgets for MSDG that contain a number have outdegree at most 3. So the construction in Fig. 6 can be used to enforce that the right number of adjacent vertices contain mines.

Lastly, the gadget in Fig. 7 can be used to mimic clicking in the original graph. The bold x is the only cells that will give information when clicked[1], so this gadget can only be used if x does not contain a mine, i.e., when the vertex represented by the wire x does not contain a mine. The depicted gadget represents the case of a click in our force-click gadget in Fig. 2. The wires u and w wires correspond to the two neighbours of x that either both have a mine or none of them (the translation of this constraint is not depicted in Fig. 7). The y wire corresponds to the other neighbour of x (outside of the force-click gadget). Hence, all three different numbers

[1]We could click any other individual cell to tell if there is a mine in them or not, but if there is, we automatically lose. So it is just as good to assume that it is not a mine and not click. de Bondt (2012) presents this reasoning.

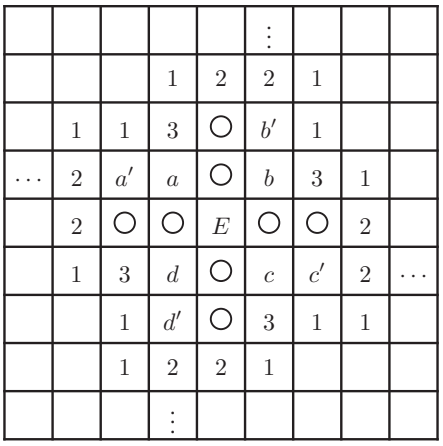


Figure 6: Circles represent revealed mines. Exactly $E-4$ of the vertices represented by wires a, b, c and d can contain mines. If there are fewer than 4 wires, the gadget can be slightly altered by revealing the areas of unused wires.

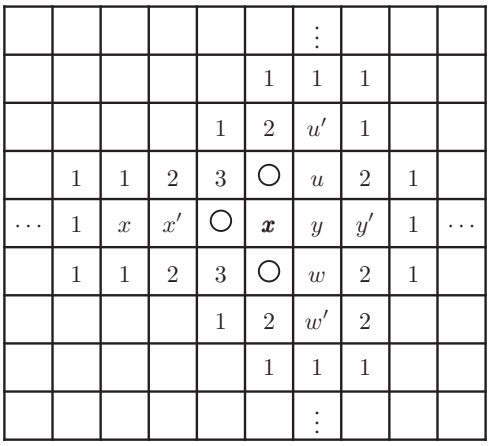


Figure 7: Clicking any of the non-bold squares in this construction is either unsafe or reveals no information.

that the bold x can reveal allow us to identify the position of the mines among squares labeld u, y, and w.

Every hidden vertex in our construction has either outdegree 3 and is part of a force-click gadget, or it has outdegree 1. In the latter case we can simplify the gadget in Fig. 7 by ommiting the u and w wires and adjusting the revealed numbers accordingly.

Theorem 6. $(\text{input}, 1)$*-MS is* PSPACE*-hard.*

Proof. The construction above is a polynomial time reduction from MSDG to $(\text{input}, 1)$-MS. Hence, $(\text{input}, 1)$-MS is PSPACE-hard. □

(input, C)-MS and $(\text{input}, \text{input})$-MS

We will show that determining if there is a strategy that wins $C = p/q$ of the time, for constant p/q is also PSPACE-complete.

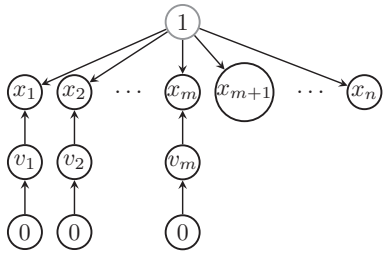


Figure 8: Chance gadget, that can be won with probability $\frac{m+1}{n}$, given m clicks.

(input, C)-MS
Input: A partially revealed Minesweeper grid and an integer k.
Output: Is it possible to determine the location of all mines with probability $C = p/q$ in k clicks?

Theorem 7. (input, C)*-MS and* $(\text{input}, \text{input})$*-MS are* PSPACE*-hard.*

Proof. We will show that (input, C)-MS is PSPACE-hard. Since (input, C)-MS is a special case of $(\text{input}, \text{input})$-MS, it will follow that $(\text{input}, \text{input})$-MS is also PSPACE-hard. We will make use of the instance shown in Fig. 8 that can be won with probability $\frac{m+1}{n}$ given m clicks. Only the vertices v_i are safe and give additional information when clicked, so it is optimal to spend all m clicks on these vertices. If the single mine that is hidden among $x_1, \ldots, x_n$ is revealed to be among the first m of these vertices the player can correctly solve this gadget with probability 1. Otherwise, she needs to guess the position among the $n-m$ remaining vertices. The probability of losing, over all possible distributions of mines is $\left(1-\frac{m}{n}\right) \cdot \frac{n-m-1}{n-m} = \frac{n-m-1}{n}$. Hence, the probability of wining is $\frac{m+1}{n}$.

So, by setting $m+1=p$ and $n=q$, we have an instance that can be won p/q of the time. We will use the same technique as for $(\text{input}, 1)$-MS to transform our gadget to the Minesweeper grid. The only problem is, that the gadget for converting numbered vertices (Fig. 6) is limited to outdegree of up to 4. But our chance gadget contains the vertex labeled 1 with outdegree n. We can use the construction in Fig. 9 to iteratively split a numbered 1 vertex with high outdegree in multiple ones having outdegree 4. We introduce new hidden vertices $y_1, \ldots, y_{\lceil \frac{n}{3} \rceil}$ and $y'_1, \ldots, y'_{\lceil \frac{n}{3} \rceil}$ and make sure that the primed versions correspond to the negation of the non-primed ones. We can use these new vertices to split the outdegree as shown in Fig. 9.

Now we can transform a QBF instance into an instance of (input, C)-MS. We create an instance that is an isolated combination of our chance gadget from Fig. 8 and our QBF gadget used for showing Theorem 6. The player must win with probability p/q, making up to $m+k$ clicks, where k is the number of force-click gadgets in the QBF gadget.

If the player uses exactly m clicks in the chance gadget and k clicks in the QBF gadget, then the player can win with probability p/q if and only if the QBF can be satisfied (since knowing how to satisfy the QBF allows the QBF gadget to

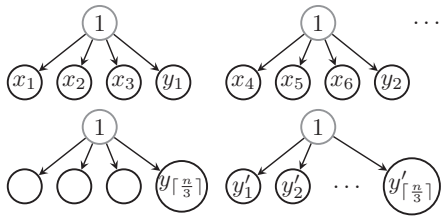


Figure 9: Iteratively split a numbered 1 vertex with high outdegree into multiple ones having outdegree 4.

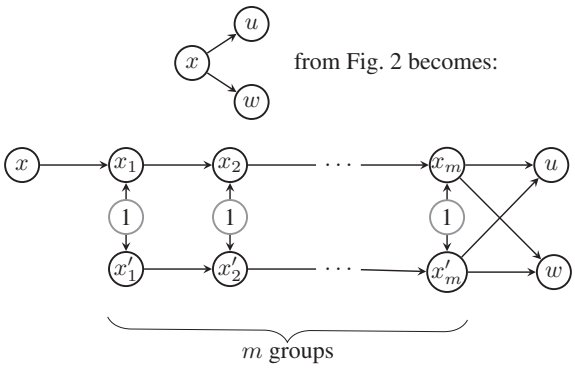


Figure 10: Clicking v reveals which of v_1 and v'_1 is safe to click, which in turn reveals which of v_2 and v'_2 is safe and so on, eventually revealing u.

be won with probability 1, for a total probability of p/q with the chance gadget included).

However, the player is not obliged to use m clicks in the chance gadget and k in the QBF gadget. It cannot help to use more than m clicks in the chance gadget (since there are only m useful places to click), but it might happen that the player can solve the chance gadget with a single click, for example if clicking on v_1 reveals that x_1 contains the mine. To prevent that the saved $m-1$ clicks can be used to influence the outcome, we modify the QBF gadget by modifying every vertex x in a force-click set (Fig. 2) as shown in Fig. 10. This delay gadget now requires the player to click $m+1$ times to actually reveal the information one would get from clicking x before the modification. The player is accordingly given $k(m+1)$ clicks in the QBF gadget. Now the player cannot save enough clicks from the chance gadget to actually help in the QBF gadget.

So this shows that the only way to win with probability C is to satisfy the QBF. □

Theorem 8. $(\text{input}, \text{input})$*-MS is in* PSPACE.

Discussion and Future Work

We have considered the complexity of Minesweeper when all consistent placements of mines are, at any time, equiprobable, we are given an upper bound k on the number of clicks we can make, and we have a lower bound p

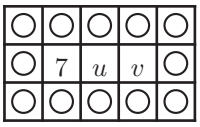


Figure 11: Chance gadget, if the player is to win for certain, they must spend one click on u (which is certainly safe) to tell if v contains a mine.

on the winning probability. de Bondt (2012) already considered the case when the number of clicks is unbounded and p is a constant or part of the input. This leaves the question when $k=\infty$ and $p=1$. To solve such an instance, one can keep making safe clicks until there is no more safe click. If, after that, the only consistent assignment of mines to cells is the one where all hidden cells have a mine, we have a yes-instance, otherwise we have a no instance. The problem of checking whether a click is safe is in co-NP, since a no-certificate is a consistent assignment of mines that places a mine where we'd like to click. Checking whether the only consistent assignment of mines to cells is the one where all hidden cells have a mine is in co-NP as well since a no-certificate is a different consistent placement of mines. Therefore, the $(\text{unbounded}, 1)$-MS problem is in $\mathsf{P}^{\mathsf{NP}} = \Delta_{\mathsf{P}}^2$.

For a constant number of clicks and $p=1$, the problem turns out to be co-NP-complete.

Lemma 9. $(K, 1)$*-MS is co-*NP*-complete.*

Proof sketch. For hardness, we reduce from the co-NP-complete $(0,1)$-MS problem. We simply add K isolated copies of Fig. 11. The player must spend all K of their clicks (one for each of the K copies) here, leaving 0 clicks to solve the original $(0,1)$-MS instance.

For memberships, consider the game tree for Minesweeper. It has depth at most K since the player can make at most K clicks. The branching factor is $\mathcal{O}(N)$, since there are at most N places the player can click. For each click, there are 10 possible results: either the player hits a mine and loses or a number from 0 to 8 is revealed. Since K is a constant, the game tree is polynomial in size, $\mathcal{O}(N^K)$.

For each leaf, we can certify to a polynomial verifier that it has non-zero probability of losing by giving two consistent assignments of mines, and since there are a polynomial number of leaves, all the certificates for the losing leaves can be encoded in a polynomial length string.

A verifier that runs in polynomial time can also run minimax on the game tree since it is polynomial in size. Thus there is a polynomial verifier that can verify a no instance with a polynomial length certificate. □

For a constant number of clicks, we leave the complexity of (K, C)-MS and (K, input)-MS open.

Acknowledgments

Serge Gaspers is the recipient of an Australian Research Council (ARC) Future Fellowship (FT140100048). Abdal-

lah Saffidine is the recipient of an ARC DECRA Fellowship (DE150101351). This work received support under the ARC's Discovery Projects funding scheme (DP150101134).

References

Bayer, K. M.; Snyder, J.; and Choueiry, B. Y. 2006. An interactive constraint-based approach to minesweeper. In *Proceedings of The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference (AAAI 2006)*, 1933–1934. AAAI Press.

Becerra, D. J. 2015. Algorithmic approaches to playing minesweeper. Bachelor's thesis, Harvard.

de Bondt, M. 2012. The computational complexity of minesweeper. Technical Report 1204.4659, arXiv CoRR.

Collet, R. 2005. Playing the minesweeper with constraints. In *Proceedings of the Second International Conference on Multiparadigm Programming in Mozart/Oz (MOZ 2004)*, 251–262. Springer.

Fix, J. D., and McPhail, B. 2004. Offline 1-Minesweeper is NP-complete. Unpublished Manuscript, Available at: www.minesweeper.info/articles/Offline1MinesweeperIsNPComplete.pdf.

Golan, S. 2011. Minesweeper on graphs. *Applied Mathematics and Computation* 217(14):6616–6623.

Heinrich, M. 2006a. Komplexität und Varianten von Minesweeper. Diplomarbeit, TU Wien.

Heinrich, M. 2006b. More properties for NP-complete Minesweeper graphs. Unpublished Manuscript, Available at: www.minesweeper.info/articles/MorePropertiesForNPCompleteMinesweeperGraphs.pdf.

Hu, S.-C., and Lin, S.-S. 2007. $2 \times n$ minesweeper consistency problem is in p. In *Proceedings of the 2007 National Computer Symposium (NCS 2007)*.

Kadlac, M. 2003. Explorations of the minesweeper consistency problem. Oregon State University, Research Experiences for Undergraduates.

Kaye, R. 2000. Minesweeper is np-complete. *The Mathematical Intelligencer* 22(2):9–15.

Nakov, P., and Wei, Z. 2003. Minesweeper, #Minesweeper. Unpublished Manuscript, Available at: http://www.minesweeper.info/articles/Minesweeper(Nakov,Wei).pdf.

Pedersen, K. 2004. The complexity of Minesweeper and strategies for game playing. Technical report, University of Warwick.

Sinz, C. 2005. Towards an optimal CNF encoding of boolean cardinality constraints. In van Beek, P., ed., *Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005, Sitges, Spain, October 1-5, 2005, Proceedings*, volume 3709 of *Lecture Notes in Computer Science*, 827–831. Springer.

Stewart, I. 2000. Million-dollar minesweeper. *Scientific American* 283(4):94–95.

Studholme, C. 2000. Minesweeper as a constraint satisfaction problem. University of Toronto project report.

Vomlelova, M., and Vomlel, J. 2009. Applying bayesian networks in the game of minesweeper. In *Proceedings of the 12th Czech-Japan Seminars on Data Analysis and Decision Making under Uncertainty*, 153–162.