The Maze Generation Problem is NP-complete

Mario Alviano

Department of Mathematics, University of Calabria, 87030 Rende (CS), Italy
alviano@mat.unical.it

Abstract. The *Maze Generation* problem has been presented as a benchmark for the Second Answer Set Programming Competition. We prove that the problem is NP–complete and identify relevant classes of unsatisfiable instances.

The *Maze Generation* (MG) problem has been presented as a benchmark for the Second Answer Set Programming (ASP) Competition[1]. The problem has been placed in the NP category, which contains problems in NP for which an algorithm in P is unknown. Here, we prove that the problem is in fact complete for the complexity class NP. In addition, we identify classes of unsatisfiable instances that are efficiently recognizable. We start the discussion by defining a maze.

Definition 1 (Maze). *A maze is an $m \times n$ grid, in which each cell is empty or a wall and two distinct cells on the edges are indicated as entrance and exit, satisfying the following conditions: (1) Each cell in an edge of the grid is a wall, except entrance and exit that are empty; (2) There is no 2×2 square of empty cells or walls; (3) If two walls are on a diagonal of a 2×2 square, then not both of their common neighbors are empty; (4) No wall is completely surrounded by empty cells; (5) There is a path from the entrance to every empty cell.*

The MG problem is the decision problem concerning the possibility to build a maze by extending a partially fixed grid.

Definition 2 (Maze Generation problem). *An instance of the MG problem is a structure of the form $\langle G, I, O, W, E\rangle$, where G is a set of cells (pairs of integers) representing a grid, I and O are two distinct cells of G, W and E are subsets of G. The MG problem is then defined as follows: Given a structure $\langle G, I, O, W, E\rangle$, is there a maze of the same size of G such that I and O are entrance and exit, respectively, each cell in W is a wall and each cell in E is empty?*

An instance of MG is represented in Fig. 1. In the picture, filled cells belong to W, while empty cells are in E. Entrance and exit are marked with I and O, respectively, while undefined cells (cells not in $W \cup E$) are indicated by a question mark (?). This is a *yes*-instance, as proved by the maze in Fig. 2.

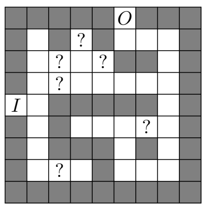

Fig. 1. An instance of Maze Generation.

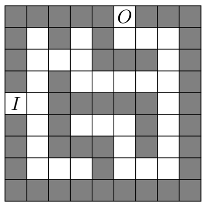

Fig. 2. A maze for the instance in Fig. 1.

[1] http://www.cs.kuleuven.be/~dtai/events/ASP-competition/

A non-deterministic algorithm for MG is "Guess a value in $\{\text{empty}, \text{wall}\}$ for each undefined cell and check that all the conditions of a maze are satisfied". The check can be performed in linear time, and so MG belongs to NP. In order to prove the NP–completeness of MG, we present a reduction from the Satisfiability problem (SAT). The proof is similar to the one presented for Minesweeper[2]: We show how to represent a logical circuit with blocks of cells of an MG instance, proving that any boolean formula F can be mapped to an instance $\langle G, I, O, W, E\rangle_F$ of MG such that $\langle G, I, O, W, E\rangle_F$ is a *yes*-instance if and only if F is satisfiable. The blocks required for representing a logical circuit by an instance of MG are shown in Fig. 3–9. In the pictures, filled cells belong to W, while empty cells belong to E. The other cells are undefined and marked with the literal they represent, where $\overline{x}$ is the complement of x. For a cell marked with a literal, we say that empty corresponds to false and wall to true. Each block is now briefly described. The block in Fig. 3 represents a wire, while the block in Fig. 4 is used to terminate a wire. Unused blocks are filled with the block in Fig. 5, while a signal is split with the block in Fig. 6. Blocks are connected by one of the connectors in Fig. 7. The non-empty connector is used for propagating literal values between blocks, and the empty connector in all other cases. The single cell between four connected blocks is filled with an empty cell. An OR gate is shown in Fig. 8. In this block, the value of the undefined cells is determined by the cells representing x, y and z in the left-middle part of the block. If empty is assigned to both x and y, then $\overline{z}$ must be wall in order to avoid an isolated wall. On the other hand, if empty is assigned to the cells marked with z, then both $\overline{x}$ and $\overline{y}$ must be wall to avoid an isolated wall. So $\overline{z} \equiv \overline{x} \wedge \overline{y}$, thus $z \equiv x \vee y$ holds. Note that an OR gate is an 11×11 block, using the same space as combining four 5×5 blocks. A NOT gate is shown in Fig. 9. In this block, the value of all the undefined cells is fixed by the cells in the vertical center of the block. Consider the right-middle part of the block. If empty is assigned to the cells marked with x, then y must be wall in order to avoid an isolated wall. The same holds for $\overline{x}$ and $\overline{y}$ on the left-middle part of the block: $\overline{x}$ empty implies $\overline{y}$ wall. Then $y \equiv \overline{x}$ holds. Since the signal is propagated from the first to the last row, a NOT gate can be combined with a splitter to obtain a bent wire. Note that a NOT gate is a 17×11 block, using the same space as combining six 5×5 blocks. In all the presented blocks, with the exception of the splitter, there is a path connecting each pair of empty cells, for any literal assignment. We can then avoid isolated empty cells by attaching wires/terminators

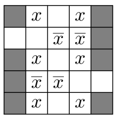
Fig. 3. Wire.

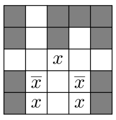
Fig. 4. Terminated wire.

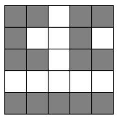
Fig. 5. Filler.

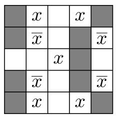
Fig. 6. Splitter.

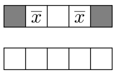
Fig. 7. Blocks connectors.

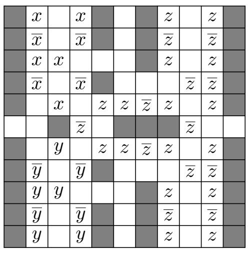
Fig. 8. OR gate.

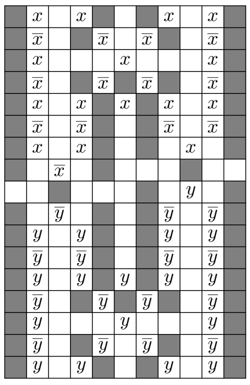
Fig. 9. NOT gate.

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

to all the edges with literals of any used splitter. Moreover, since a wire ends in a terminator (Fig. 4) or in a gate (Fig. 8 and 9), cells marked with the same literal must be either all empty or all wall.

Instances of MG are planar, while circuits may include crossed wires (which are non-planar), so we have to find a suitable planar representation of these. Assuming the availability of an XOR gate, crossed wires can be simulated by three splitters and three XORs, as shown in Figure 10[3]. The XOR gate itself can be simulated in the plane by OR and NOT gates, as shown in Fig. 11. Since any boolean function can be represented by a combination of OR and NOT gates, these building blocks are sufficient for building any logical gate or circuit. For example, the formula $\neg(\neg x \vee \neg y)$ is equivalent to an AND gate.

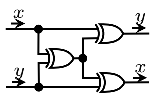

Fig. 10. Crossing two wires with three XOR gates.

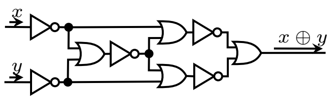

Fig. 11. Making an XOR gate with OR and NOT gates.

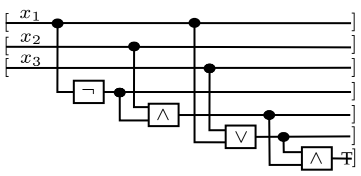

Fig. 12. A Maze Generation circuit for $((\neg x_1 \wedge x_2) \wedge (x_1 \vee x_3))$.

Given a boolean formula F, we map F into an instance of MG by building a logical circuit like the one in Fig. 12, which represents $((\neg x_1 \wedge x_2) \wedge (x_1 \vee x_3))$. In the picture, lines are wires, crossed lines are cross-overs, filled-in circles are splitters, boxes are gates and square brackets are terminators. The terminator marked with T fixes the signal to *true* (by fixing to wall the cell marked with x in the third row of the block in Fig. 4). The circuit can be built by a depth-first visit of the tree structure of the boolean formula. For each node, after all its children have been visited, a new gate is introduced and the wires of the children are split and connected to the gate. Each element of such a circuit can be built with an $N \times N$ block, for some fixed N not depending on the boolean formula, and then all the circuit has size $(Nn)^2$, where n is the number of symbols in the boolean formula. The logical circuit can be then built in polynomial time. We complete the construction by putting the circuit inside a border of empty cells (for connecting all blocks to the entrance), surrounded by a border of undefined cells. In one edge of the undefined cells, we arbitrarily choose two non-adjacent non-corner cells representing entrance and exit.

Maze Generation and Unsatisfiability. We now associate a bit string to each row of a maze, where 1 stays for a wall and 0 for an empty cell. If a is a bit string of length $k \geq 1$, a_i denotes the i-th bit of a, $1 \leq i \leq k$.

Definition 3 (0-block). *A 0-block of length $k \geq 3$ is a bit string a of length k, where $a_1 = a_k = 1$ and $a_2 = \cdots = a_{k-1} = 0$. If k is even, then the 0-block will be called an even 0-block, otherwise an odd 0-block. Given a bit string b, let the number of* even 0-blocks *in b be denoted by $\#_b$.*

An even 0-block is given in Fig. 13. Properties 2 and 3 of a maze deal with 2×2 squares of cells (see Fig. 14). We then introduce a formal definition of (2×2) square in terms of bit strings.

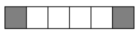
Fig. 13. An even 0-block.

[3] Leslie M. Goldschlager. The monotone and planar circuit value problems are log space complete for P. *ACM SIGACT News*, 9 (2): 25–29, 1977.

Definition 4 (Squares). *Let a and b be two bit strings of length $k \geq 2$. Then in the pair ab we identify $k-1$ squares: $[a_i, a_{i+1}, b_i, b_{i+1}]$, $1 \leq i \leq k-1$. We define a "bad square" as a square in which $a_i = b_{i+1}$ and $a_{i+1} = b_i$. A bad square is called a 0-square if $a_i = a_{i+1} = 0$, a 1-square if $a_i = a_{i+1} = 1$, a D-square if $a_i \neq a_{i+1}$.*

Given the definition above, condition 2 and 3 of Definition 1 are equivalent to "there is no bad square" on 2 consecutive rows. The notions of 0-block and squares are exploited to show our results. In particular, the next lemma states an important property regarding the number of even 0-blocks in two bit strings starting and ending in 1 and without bad squares.

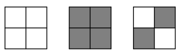
Fig. 14. A 0-square, a 1-square, and a D-squares.

Lemma 1. *Let a and b be two bit strings of length $k \geq 1$ such that ab has no bad square, and $a_1 = a_k = b_1 = b_k = 1$. Then k is odd if and only if $\#_a$ and $\#_b$ have the same parity.*

To simplify the presentation, we postpone the proof of the lemma above after the presentation of the classes of unsatisfiable instances. The first class is introduced by the next theorem.

Theorem 1. *Let $\langle G, I, O, W, E \rangle$ be an instance of MG such that I and O are adjacent and not in a corner. If the edge with I and O has odd length, then the instance is unsatisfiable.*

Proof. Let m be the number of rows in G and G_i the bit string associated to the i-th row of the grid, $1 \leq i \leq m$. Assume that I and O are in the first row, so that $\#_{G_1} = 1$. Since I and O are not located in a corner, the first and last bits of G_i are equal to 1, $1 \leq i \leq m$, and so we can apply Lemma 1 to each pair $G_i G_{i+1}$, $1 \leq i \leq m-1$, and conclude that each row of G has an odd number of even 0-block. This is impossible because the last row must be filled with only walls, that is, $G_m = 1 \cdots 1$. □

Before introducing the second class of unsatisfiable instances we define a relaxed problem that allows us to apply Lemma 1 in case entrance and exit are not adjacent. The problem asks for a quasi-maze in an $m \times n$ grid, where a quasi-maze is defined as follows.

Definition 5 (Quasi-maze). *A quasi-maze is an $m \times n$ grid in which all the edges are walls, each cell is empty or a wall, and there is no bad square in two consecutive rows.*

A quasi-maze can be obtained from a maze if entrance and exit are not adjacent.

Lemma 2. *From a maze in which entrance and exit are not adjacent it is always possible to construct a quasi-maze simply by putting a wall in the entrance and exit cells.*

Proof. Since entrance and exit are not adjacent, they cannot be on a corner of the maze, otherwise they are not reachable from the other empty cells. So entrance and exit have 3 adjacent cells. Two of these are on the edge, and so are walls. The remaining one must be empty, in order to avoid an unreachable cell. Let a be the edge with entrance/exit, and b the row or column adjacent to a. So, if a_i is the entrance/exit cell, then $a_{i-1} = a_{i+1} = 1$

and $b_i = 0$ holds. Thus, independently of the value of a_i, we never have a bad square in the two squares in which a_i is involved. Now note that the other squares come from a maze, and so they contain no bad square. □

The relaxed problem is unsatisfiable if both m and n are even.

Theorem 2. *There is no quasi-maze of size $m \times n$ if both m and n are even.*

Proof. Let m be the number of rows in G and G_i the bit string associated to the i-th row of the grid, $1 \leq i \leq m$. Since all the edge cells of a quasi-maze are filled with walls, the first and last bits of G_i are equal to 1, $1 \leq i \leq m$. So we can apply Lemma 1 to each pair G_iG_{i+1}, $1 \leq i \leq m-1$, and conclude that $\#_{G_{2k-1}}$ is even, while $\#_{G_{2k}}$ is odd, $1 \leq k \leq \frac{m}{2}$. This is impossible because the last row must be filled with only walls, that is, $G_m = 1 \cdots 1$. □

The second class of unsatisfiable instances is then identified.

Corollary 1. *Let $\langle G, I, O, W, E\rangle$ be an instance of MG such that G is an even $\times$ even grid. If I and O are not adjacent, then the instance is unsatisfiable.*

Proof. From Lemma 2 and Theorem 2.

Proof of Lemma 1. We will prove Lemma 1 using strong induction by splitting the strings at a 0-block of length at least 4. The next lemma assures that there is such a 0-block for strings of even length (greater than 2).

Lemma 3. *Let a and b be two bit strings of length $k \geq 4$ such that ab has no bad square, and $a_1 = a_k = b_1 = b_k = 1$. If k is even, then there is a 0-block of length at least 4 in a or b.*

Proof. We assume that a has no 0-block of length at least 4 and use induction on k to show that b has a 0-block of length at least 4. Suppose first that $k = 4$. In this case, at least one of a_2 and a_3 must be 1, in order to avoid a 0-block of length 4; we can safely assume that it is a_2 (a_3 is symmetric). Then b_2 is 0 to avoid a 1-square. Now note that b_3 must be 0 because if a_3 is 0 we have to avoid a D-square, while for $a_3 = 1$ a 1-square must be avoided. So b is a 0-block of length 4, and the claim is true. Suppose the claim is true for all the pairs of bit strings of length $k-2$ in order to show that it holds for a pair of bit strings of length k. If $b_2 = b_3 = 0$ the claim is trivially true. If $b_2 = 1$, then $a_2 = 0$ to avoid a 1-square, which implies $a_3 = 1$ to avoid a 0-block of length greater than 3, which in turn implies $b_3 = 1$ to avoid a D-square; on the other hand, if $b_2 = 0$ and $b_3 = 1$, then $a_3 = 1$ holds for otherwise we would have a D-square or a 0-block of length at least 4 in case $a_2 = 1$ or 0, respectively. So, in both cases, $a_3 = b_3 = 1$. We can then apply the induction hypothesis on $a_3 \cdots a_n$ and $b_3 \cdots b_n$ and conclude that the claim is true. □

Lemma 1 requires that both the bit strings start and end in 1. In order to apply the inductive hypothesis after the strings are split at a 0-block (of length at least 4), we can possibly add some bits to the string with the 0-block preserving the number of even 0-blocks.

Proposition 1. *Let a be a bit string of length $k \geq 4$ such that $a_1 = a_k = 1$. If $a_i \cdots a_j$ is a 0-block, $1 \leq i < j \leq k$, such that $j - i + 1 \geq 4$, then $\#_{a_1 \cdots a_i} = \#_{a_1 \cdots a_{i+1} 1}$, and $\#_{a_j \cdots a_k} = \#_{1 a_{j-1} \cdots a_k}$.*

Proof. Indeed, since $a_i = a_j = 1$, no even 0-block is added to $a_1 \cdots a_i$ and $a_j \cdots a_k$, regardless the value of a_{i+1} and a_{j-1}. □

To simplify the discussion we introduce a compact notation for the bit strings preceding and following a 0-block.

Definition 6 (Left and right sub-strings). *Let a and b be two bit strings of length $k \geq 4$ such that ab has no bad square, and $a_1 = a_k = b_1 = b_k = 1$. Let $a_i \cdots a_j$ be a 0-block, $1 \leq i < j \leq k$, such that $j - i + 1 \geq 4$. The left and right sub-strings of a and b w.r.t. $a_i \cdots a_j$, denoted $a^{\prec i}, b^{\prec i}$ and $a^{\succ j}, b^{\succ j}$, respectively, are defined as follows:*

$$\begin{cases} a^{\prec i} = a_1 \cdots a_i \\ b^{\prec i} = b_1 \cdots b_i \end{cases} \text{, if } b_i = 1 \quad \begin{cases} a^{\succ j} = a_j \cdots a_k \\ b^{\succ j} = b_j \cdots b_k \end{cases} \text{, if } b_j = 1$$
$$\begin{cases} a^{\prec i} = a_1 \cdots a_{i+1} 1 \\ b^{\prec i} = b_1 \cdots b_{i+1} 1 \end{cases} \text{, if } b_i = 0 \quad \begin{cases} a^{\succ j} = 1 a_{j-1} \cdots a_k \\ b^{\succ j} = 1 b_{j-1} \cdots b_k \end{cases} \text{, if } b_j = 0$$

After applying induction w.r.t. a 0-block of one of the strings, we can determine the number of even 0-blocks in the other string by summing the number of even 0-blocks in its left and right sub-strings.

Proposition 2. *Let a and b be two bit strings of length $k \geq 4$ such that ab has no bad square, and $a_1 = a_k = b_1 = b_k = 1$. Let $a_i \cdots a_j$ be a 0-block, $1 \leq i < j \leq k$, such that $j - i + 1 \geq 4$. Then $\#_b = \#_{b^{\prec i}} + \#_{b^{\succ j}}$.*

Proof. Since ab has no 0-squares and no D-squares, $b^{\prec i}$ and $b^{\succ j}$ are in fact sub-strings of b and the sub-string obtained from b by removing $b^{\prec i}$ and $b^{\succ j}$ has no consecutive 0's. □

We can now prove Lemma 1.

Proof (Lemma 1). We use strong induction on the length of the bit strings. Observe that for $k = 1$ and $k = 3$ the claim is trivially true because $\#_a = \#_b = 0$, while $k = 2$ cannot hold because ab has no 1-square. For $k = 4$, at least one of a_2, a_3, b_2, b_3 must be 1, in order to avoid a 0-square; without loss of generality we can assume that it is a_2. Then from Lemma 3 we have that b has at least one 0-block of length at least 4. This implies that b is a 0-block itself, and so the claim is true because b has even length.

Suppose the claim is true for all the pairs of bit strings of length k', with $k' < k$, in order to prove that it holds for a pair of bit strings of length k.

Consider first the case in which $\#_a = \#_b = 0$. Assume, by contradiction, that k is even. We can apply Lemma 3 and conclude that there is a 0-block of length at least 4 in a or b; we may assume that $a_i \cdots a_j$, $1 \leq i < j \leq k$, is a 0-block such that $j - i + 1$ is at least 5 and odd. Since k is even and $j - i + 1$ odd, exactly one of i and $k - j + 1$ is odd; we may assume that it is i by symmetry. We now apply the induction hypothesis to $a^{\prec i}$ and $b^{\prec i}$, and to $a^{\succ j}$ and $b^{\succ j}$. From Proposition 1 and Proposition 2 we obtain that $\#_{b^{\prec i}}$ is even, while $\#_{b^{\succ j}}$ is odd, and so $\#_b$ is odd, contradicting the assumption that $\#_b = 0$. We can conclude that k is odd.

From now on we may assume that at least one of $\#_a$ and $\#_b$ is different from 0. Again, we may assume that $\#_a$ is greater than 0. Let $a_i \cdots a_j$, $1 \le i < j \le k$, be the first even 0-block in a. Then, since $\#_{a^{\prec i}} = 0$, by applying the induction hypothesis to $a^{\prec i}$ and $b^{\prec i}$, we conclude that $\#_{b^{\prec i}}$ is even if and only if i is odd. In addition, $\#_{a^{\succ j}} = \#_a - 1$, and so, by applying the induction hypothesis to $a^{\succ j}$ and $b^{\succ j}$, we conclude that if $\#_{a^{\succ i}}$ is even, $\#_{b^{\succ j}}$ is even if and only if $k - j + 1$ is odd, while for $\#_{a^{\succ i}}$ odd, $\#_{b^{\succ j}}$ is even if and only if $k-j+1$ is even. Moreover, $k-j+1$ depends on k and i, and so $k - j + 1$ is even if and only if k and i have the same parity. Then there are only eight cases to be considered, regarding the parities of k, i and $\#_a$, as reported in Table 1.

Table 1. Parities.

k	i	$\#_a$	$k-j+1$	$\#_{a^{\succ j}}$	$\#_{b^{\prec i}}$	$\#_{b^{\succ j}}$	$\#_b$
even	even	even	even	odd	odd	even	odd
even	even	odd	even	even	odd	odd	even
even	odd	even	odd	odd	even	odd	odd
even	odd	odd	odd	even	even	even	even
odd	even	even	odd	odd	odd	odd	even
odd	even	odd	odd	even	odd	even	odd
odd	odd	even	even	odd	even	even	even
odd	odd	odd	even	even	even	odd	odd

The columns following the first double vertical line are obtained from the preceding columns by following the previous observations:

- $k - j + 1$ is even if and only if k and i have the same parity;
- $\#_{a^{\succ j}}$ is obtained by "switching" the parity of $\#_a$;
- $\#_{b^{\prec i}}$ is obtained by "switching" the parity of i;
- $\#_{b^{\succ j}}$ is odd if and only if $k - j + 1$ and $\#_{a^{\succ j}}$ have the same parity;
- $\#_b$ is obtained by "summing" the parities of $\#_{b^{\prec i}}$ and $\#_{b^{\succ j}}$.

The first four rows of Table 1 show that the claim is true for k even, that is $\#_a$ and $\#_b$ have different parities, while the last four rows prove the claim for k odd, that is $\#_a$ and $\#_b$ have the same parity. $\square$

Conclusion. We proved that the Maze Generation problem is NP–complete. We also identified two relevant classes of unsatisfiable instances. Recognizing instances belonging to these classes is easy, and so a solver can use the presented results in practice. Moreover, the proof of NP–completeness satisfies the reachability condition without using it, so proving the NP–completeness of MG also if condition (5) is absent. Finally, Theorem 1 holds regardless of conditions (4) and (5), while Lemma 2, and so Theorem 2 and Corollary 1, holds independently of condition (4).

Acknowledgments. Thanks to John van Bon and Wolfgang Faber for helping me in proving and presenting the results reported in this paper.