DEPARTMENT OF MATHEMATICS
RADBOUD UNIVERSITY NIJMEGEN The Netherlands

NP-COMPLETENESS OF MASTER MIND AND MINESWEEPER

Michiel de Bondt

Report No. 0418 (November 2004)

DEPARTMENT OF MATHEMATICS
RADBOUD UNIVERSITY NIJMEGEN
Toernooiveld
6525 ED Nijmegen
The Netherlands

NP-completeness of Master Mind and Minesweeper

Michiel de Bondt
Radboud University
Nijmegen, The Netherlands
M.deBondt@math.ru.nl

The problem whether P equals NP is one of the most important open problems in mathematics and computer science at this time. That is why one million dollars is put on the problem.To determine whether P equals NP, it is sufficient to determine for any NP-complete problem whether it is in P or not. Roughly, this means that you have to show for any NP-complete problem that there is a fast algorithm to solve it or that no such algorithm exists. In this article, we show that Master Mind and hexagonal and triangular Minesweeper are NP-complete. R. Kaye already showed that normal minesweeper is NP-complete.

An example of an NP-complete problem is (1 in 3)-SAT. In (1 in 3)-SAT, the problem is to find solutions over $\{0,1\}$ of equations of the form

$$\begin{aligned} a_i + a_j + a_k &= 1 \\ (1-a_i) + a_j + a_k &= 1 \\ (1-a_i) + (1-a_j) + a_k &= 1 \\ (1-a_i) + (1-a_j) + (1-a_k) &= 1 \end{aligned}$$

with $i \neq j \neq k \neq i$. Now you might think: "Solving equations? I have learned that on secondary school. Let us get one million bucks!" The problem is not to find a parameterization of the solutions over, say, the real numbers, but to find solutions in (that parameterization with all variables contained in) $\{0,1\}$.

Another NP-complete problem is solving polynomial equations over $\mathbb{F}_2$. We now show that this problem is NP-complete, by reducing (1 in 3)-SAT to polynomial equations over $\mathbb{F}_2$. Reducing any problem of the pool of known NP-complete problems to the problem at hand is *the* way to show that the problem at hand is NP-complete. All problems that are currently known to be NP-complete have been proven to be so by this method, except one. In the beginning, the pool of known NP-complete problems was empty, and Cook and Levin showed from scratch that SAT was NP-complete.

But let us show now that solving polynomial equations over $\mathbb{F}_2$ is NP-complete. Write a_i' for the *inverse* $1 - a_i$ of a_i and let $a_i^{(0)} = a_i$ and $a_i^{(1)} = a_i'$ (just as in a calculus course). All equations of (1 in 3)-SAT are of the form

$$a_i^{(e_1)} + a_j^{(e_2)} + a_k^{(e_3)} = 1$$

and can be expressed by the polynomial equations

$$\begin{aligned} a_i^{(e_1)} + a_j^{(e_2)} + a_k^{(e_3)} &\equiv 1 \pmod 2 \\ a_i^{(e_1)} \cdot a_j^{(e_2)} &\equiv 1 \pmod 2 \end{aligned}$$

Furthermore, the equalities $a_i + a'_i = 1$ can be expressed by

$$a_i + a'_i \equiv 1 \pmod 2$$

So a fast algorithm for solving polynomial equations over $\mathbb{F}_2$ can be used to solve (1 in 3)-SAT fast. It follows that solving polynomial equations over $\mathbb{F}_2$ is NP-complete.

For your information, the problem of SAT is solving inequalities of the form

$$a_{i_1}^{(e_1)} + a_{i_2}^{(e_2)} + \cdots + a_{i_k}^{(e_k)} \geq 1$$

where k may differ for any such inequality. Now you may guess yourself how the intermediate problems (1 in)-SAT and 3-SAT are formulated. Both problems are NP-complete as well.

Now you might think: "Isn't solving linear equations over $\mathbb{F}_2$ NP-complete? For the above quadratic equations look so simple." The answer is no. If linear equations over $\mathbb{F}_2$ would be NP-complete, then you could apply what you have learned for the field $\mathbb{R}$ or $\mathbb{Q}$ on secondary school on an arbitrary field, which $\mathbb{F}_2$ is, and make one million bucks.

1 NP-completeness of Master Mind

I did not found a reference on Master Mind being NP-complete, but that does not mean that it is hard to show that Master Mind is NP-complete. Apparently, no one found it interesting enough to write it down this far.

We reduce from (1 in 3)-SAT. Assume we have an instance of (1 in 3)-SAT, say with n variables and m equations. We take the code to be broken $2n$ coordinates long, and index it with the n variables a_i and the n inverses a'_i. Next, we set up $n + m + 2$ turns. The first turn will be an all red-turn and the second turn will be an all-blue turn. These turns are needed to indicate that the code to be broken consists of n red and n blue mushrooms. The idea is that from each pair of coordinates $\{a_i, a'_i\}$ one is red and one is blue in the code to be broken, corresponding to the fact that one of these variables equals one and the other equals zero. We use exactly n black pegs and no white pegs for the first and second turn, to indicate that half of the mushrooms is red resp. blue. Talking about the pegs, I will assume that the black ones indicate "correct and on the right place" and the white ones indicate "correct but on the wrong place". Some people do it the opposite.

In the next n turns, we make turns corresponding to the equations $a_i + a'_i = 1$. In the $(2 + i)$th turn, we make the variables a_i and a'_i red and all other variables blue, for all i with $1 \leq i \leq n$. Each of these turns gets exactly n black pegs and 2 white pegs, to indicate that n blue mushrooms plus 2 red mushrooms are correct but not necessarily on the right place. To indicate that one of the red mushrooms, and hence one of the blue mushrooms also, is on the wrong place, indeed exactly 2 white pegs are needed. If we see a red mushroom on $a_i^{(e_1)}$ in the code to be broken as $a_i^{(e_1)} = 1$, then turn $2 + i$ indicates exactly that a'_i is the inverse of a_i, since exactly one of both red mushrooms is indicated to be on the correct place.

The remaining m turns will correspond to the equations of the instance of (1 in 3)-SAT. If $a_1^{(e_1)} + a_2^{(e_2)} + a_3^{(e_3)} = 1$ is such an equation, then there will be a turn where the mushrooms on the coordinates corresponding to $a_1^{(e_1)}$, $a_2^{(e_1)}$ and $a_3^{(e_1)}$ are red and all other mushrooms are blue. Now the pegs for that turn should indicate that exactly one red mushroom is on the right place, for exactly one of the variables $a_1^{(e_1)}$, $a_2^{(e_2)}$ and $a_3^{(e_3)}$ must be equal to 1. This is done with $n-1$ black pegs and 4 white pegs. Why? Since the turn has 3 reds and the code has n reds, the total number of pegs must be $2n - (n-3) = n+3$. Furthermore, there are 2 red mushrooms not on the right place, but then, there are also 2 blue mushrooms on the wrong place: there are exactly two coordinates that are not blue in the turn, but are blue in the code to be broken. So exactly 4 of the $n+3$ pegs must be white.

The condition that exactly one red mushroom on $a_1^{(e_1)}$, $a_2^{(e_2)}$ and $a_3^{(e_3)}$ is right corresponds to $a_1 + a_2 + a_3 = 1$, if we see a red mushroom on $a_i^{(e_i)}$ in the code as $a_i^{(e_i)} = 1$ and a blue mushroom on $a_i^{(e_i)}$ in the code as $a_i^{(e_i)} = 0$. So the only way to find a candidate for the code to be broken is to solve the instance of (1 in 3)-SAT.

An example with four variables a_1, a_2, a_3, a_4 and the three equations $a_1 + a_2 + a_3 = 1$, $a_1 + a_3' + a_4 = 1$ and $a_2' + a_3 + a_4' = 1$ is shown in figure 1.

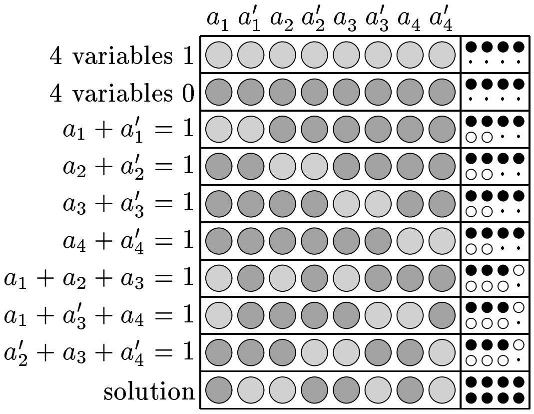

Figure 1: Master Mind is NP-complete

Consistent Master Mind

Now we showed that Master Mind is NP-complete, but you might object that you never play this way. After the first turn, it is known that the code to be broken consists of n red mushrooms exactly, and can never be the all-blue code. For that reason, we now show that consistent Master Mind is NP-complete. In consistent Master Mind, each turn must be a valid candidate for the code to be broken, given the previous turns and their pegs.

$$a_1 + a'_1 + b_1 + c_{11} + c_{12} + c_{13} + c_{14} + c_{15} = 1$$

$$a_2 + a'_2 + b_2 + c_{12} + c_{22} + c_{23} + c_{24} + c_{25} = 1$$

$$a_3 + a'_3 + b_3 + c_{13} + c_{23} + c_{33} + c_{34} + c_{35} = 1$$

$$b_1 + b_2 + b_3 + b_4 + b_5 + b_6 + b_7 + b_8 = 0$$

$$a_1 + a_2 + a_3 + c_{41} + c_{42} + c_{43} + c_{44} + c_{45} = 1$$

$$a_1 + a'_2 + a'_3 + c_{51} + c_{52} + c_{53} + c_{54} + c_{55} = 1$$

$$c_{83} + c_{74} + c_{65} + c_{51} + c_{42} + c_{33} + c_{24} + c_{15} = 0$$

$$c_{84} + c_{75} + c_{61} + c_{52} + c_{43} + c_{34} + c_{25} + c_{11} = 0$$

$$c_{85} + c_{71} + c_{62} + c_{53} + c_{44} + c_{35} + c_{21} + c_{12} = 0$$

$$c_{81} + c_{72} + c_{63} + c_{54} + c_{45} + c_{31} + c_{22} + c_{13} = 0$$

$$c_{82} + c_{73} + c_{64} + c_{55} + c_{41} + c_{32} + c_{23} + c_{14} = 0$$

solution candidate 1

solution candidate 2

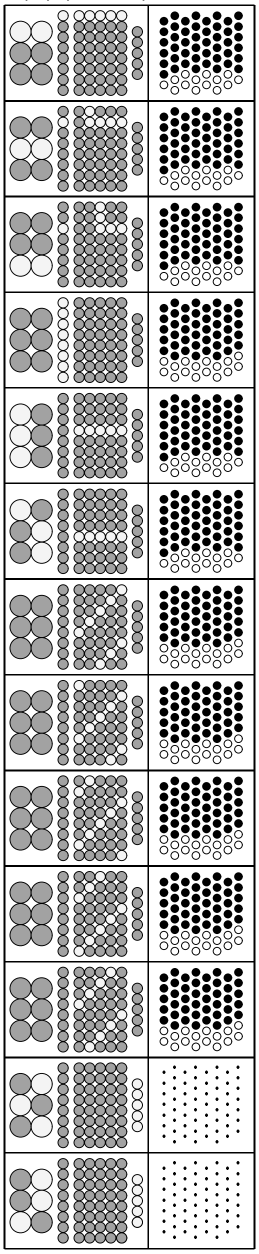

Figure 2: Consistent Master Mind is NP-complete

Showing that consistent Master Mind is NP-complete is somewhat more difficult. Again we reduce from (1 in 3)-SAT, but we assume that each pair of equations has at most one term in common. There are twice as many terms as variables, since each variable has an inverse. At the end of this section, we show that this assumption is justified.

Say we have an instance of (1 in 3)-SAT, such that two equations do not share more than one term, say with n variables and m equations. We use $(n+m+3)(n+m+2)+n$ coordinates in our Master Mind game, corresponding to variables which we divide in four groups:

- The n variables $a_1, a_2, \ldots, a_n$ of the instance of (1 in 3)-SAT and their inverses $a'_1, a'_2, \ldots, a'_n$. Half of these $2n$ coordinates are one, i.e. correspond to a red coordinate in the code to be broken,
- Variables b_i for all i with $1 \leq i \leq n+m+3$, all zero, i.e. corresponding to a blue coordinate in the code to be broken,
- Variables c_{ij} for all i, j with $1 \leq i \leq n+m+3$ and $1 \leq j \leq n+m$, all zero, i.e. corresponding to a blue coordinate in the code to be broken,
- Variables $d_1, d_2, \ldots, d_{m+3}$, all one, i.e. corresponding to a red coordinate in the code to be broken.

The first turn will correspond to the equation

$$a_1 + a'_1 + b_1 + c_{11} + c_{12} + \cdots + c_{1(n+m)} = 1$$

so the red mushrooms are on the variables in the sum on the left hand side exactly and all other coordinates get a blue mushroom. There are exactly $n+m+3$ red pegs in the code and the same number of red pegs in the first turn, so the first turn gets as many pegs as there are coordinates, i.e. $(n+m+3)(n+m+2)+n$ pegs. $2(n+m+2)$ pegs will be white, to indicate that exactly one red peg in on the correct place, corresponding to the fact that the sum of the red variables in the first turn equals one.

The second turn will correspond to

$$a_2 + a'_2 + b_2 + c_{12} + c_{22} + \cdots + c_{2(n+m)} = 1$$

The equation of the second turn has exactly one variable in common with the first turn (c_{12} to be precise). Furthermore, it has $n+m+3$ variables also. The same holds for the code to be broken: $n+m+3$ red variables of which exactly one can be found in the equation of the first turn. It follows that the second turn is consistent.

More generally, the ith turn will correspond to

$$a_i + a'_i + b_i + c_{1i} + \cdots + c_{ii} + \cdots + c_{i(n+m)} = 1$$

for all $i \leq n$. It is consistent since it shares exactly one red coordinate with each of the previous turns.

The $(n+1)$-th turn will correspond to

$$b_1 + b_2 + \cdots + b_{n+m+3} = 0$$

So the coordinates b_i get a red mushroom and all other coordinates get a blue mushroom. $2(n+m+3)$ white pegs will be used, to indicate that all red mushrooms are on the wrong place, corresponding to the fact that the sum of the red variables in this turn equals zero.

Next, m turns corresponding to the m equations of the instance of (1 in 3)-SAT follow. Say that $\alpha_{i1} + \alpha_{i2} + \alpha_{i3} = 1$ is the i-th equation, where each α_{ij} is one of the variables a_k or a'_k. The $(n+1+i)$th turn will correspond to an equation of the form

$$\alpha_{i1} + \alpha_{i2} + \alpha_{i3} + \gamma_{(n+i)1} + \cdots + \gamma_{(n+i)(n-1+i)} + c_{(n+i)(n+i)} + \cdots + c_{(n+i)(n+m)} = 1$$

where $\gamma_{(n+i)j} \in \{c_{(n+i)j}, c_{j(n+i)}\}$. For consistency, this turn must not have red mushrooms on the red coordinates of the $(n+1)$th turn and must share exactly one red coordinate with all other previous turns. The former is automatically satisfied, since no variables b_j occur in the above equation. About the latter, the partial sum $\alpha_{i1} + \alpha_{i2} + \alpha_{i3}$ has at most one variable in common with any equation of a previous turn, say the j-th turn, with $j \neq n+1$. The whole sum $\alpha_{i1} + \alpha_{i2} + \alpha_{i3} + \gamma_{(n+i)1} + \cdots + \gamma_{(n+i)(n-1+i)} + c_{(n+i)(n+i)} + \cdots + c_{(n+i)(n+m)}$ must have exactly one variable in common with the equation of the j-th turn, in order to get consistency with respect to the j-th turn.

Assume first that $j < n+1$. Then $c_{j(n+i)}$ is a variable of the j-th turn. If $\alpha_{i1} + \alpha_{i2} + \alpha_{i3}$ does not have a variable in common with the equation of the j-th turn, then $\gamma_{(n+i)j} = c_{j(n+i)}$. This way, both equations have variable $c_{j(n+i)}$ in common. If $\alpha_{i1} + \alpha_{i2} + \alpha_{i3}$ do already have a variable in common with the equation of the j-th turn, then $\gamma_{(n+i)j} = c_{(n+i)j}$. In both cases, the equations of the jth turn and the $(n+1+i)$th turn have exactly one variable in common. The case $j > n+1$ is similar. We adjust $\gamma_{(n+i)(j-1)}$ now instead of $\gamma_{(n+i)j}$.

Turn $n+1+m+k$ will correspond to

$$\sum_{i+j \equiv k \pmod{n+m+3}} c_{ij} = 0$$

and since these equations share exactly one variable with the equations of turn 1 to n and the equations of turn $n+2$ to $n+m+1$, and no variables with each other and the $(n+1)$th equations, these turns are consistent.

After all turns, it is known that all b_i and all c_{ij} are zero. Furthermore, the first n equations imply that from each pair $\{a_i, a'_i\}$, exactly one is equal to one. So besides the d_i, there are exactly n red coordinates in the code to be broken. Since there are exactly $n+m+3$ red coordinates in the code to be broken, the coordinates $d_1, d_2, \ldots, d_{m+3}$ must be red in the code to be broken, and candidates for the code to be broken correspond to solutions of the instance of (1 in 3)-SAT.

An example with three variables and the two equations $a_1 + a_2 + a_3 = 1$ and $a_1 + a'_2 + a'_3 = 1$ is shown in figure 2. Since this system of equations has two solutions, there are two candidates for the solution. Furthermore, not all mushrooms have the same size and the red mushrooms are yellow here.

Final issues

So consistent Master Mind is NP-complete. A somewhat stronger concept than NP-completeness is ASP-completeness, see [3]. Since(1 in 3)-SAT is ASP-complete [3, Th. 3.6] and the NP-reductions in the above proof are even ASP-reductions, consistent Master Mind is even ASP-complete, if I am not mistaken.

Oh no, no I am not mistaken, but I forgot to justify the assumption that two equations share at most one term, remember? Suppose that this is not the case, say $a_1 + a_2 + a_3 = 1$ and $a_1 + a_2 + a_k^{(e)} = 1$ are both equations of the system. There are two ways to deal with this situation.

- The theoretical approach:
  The information that $a_1 + a_2 + a_k^{(e)} = 1$ adds to $a_1 + a_2 + a_3 = 1$ is that $a_k^{(e)} = a_3$. This information can be coded in a different matter as well. Instead of $a_1 + a_2 + a_k^{(e)} = 1$, we use the equations

$$\begin{aligned} a_3 + b_1' + b_2 &= 1 \\ a_k^{(e)} + b_3' + b_4 &= 1 \\ b_1' + b_3 + b_5 &= 1 \\ b_2' + b_4 + b_5 &= 1 \\ b_2 + b_4' + b_5 &= 1 \\ b_2 + b_4 + b_5' &= 1 \end{aligned}$$

  Summing the last three equations of the above, we get $2b_2 + b_2' + 2b_4 + b_4' + 2b_5 + b_5' = 3$, i.e. $b_2 + 1 + b_4 + 1 + b_5 + 1 = 3$, so $b_2 = b_4 = b_5 = 0$. It follows that $b_1 = a_3$, $b_3 = b_1$ and $a_k(e) = b_3$, so $a_k(e) = a_3$. So the system of equations still has the same solutions after this procedure. Furthermore, there is at least one equation less that shares two variables with another equation. So by induction, it follows that we may assume that two equations do not share more than one variable.

- The practical approach:
  Since $a_k^{(e)} = a_3$ follows from $a_1 + a_2 + a_3 = 1$ and $a_1 + a_2 + a_k^{(e)} = 1$, we can replace $a_k^{(e)}$ by a_3 and $a_4^{(1-e)}$ by a_3' all over the system of equation. This way, the equation $a_1 + a_2 + a_k^{(e)} = 1$ becomes $a_1 + a_2 + a_3 = 1$, which is already in the system, so the equation $a_1 + a_2 + a_k^{(e)} = 1$ is effectively eliminated.

  But other things might happen. We might get equations of the form

$$a_3 + a_3 + a_5 = 1$$

  which is equivalent to $a_3 = 0$ and $a_5' = 0$, and

$$a_3 + a_3' + a_5 = 1$$

which is equivalent to $a'_5 = 0$. So we must have a cure for variables being equal to zero, say that we have an equation $a_l = 0$. Say that a_l occurs also in an equation

$$a_l + a_6 + a_7 = 1$$

and a'_l occurs in an equation

$$a'_l + a_8 + a_9 = 1$$

Remove all occurences of a_l in all equations, so $a_l + a_6 + a_7 = 1$ becomes $a_6 + a_7 = 1$, which is equivalent to $a_6 = a'_7$. We already know what to do with $a_6 = a'_7$, since it is the same type of equation as the equation $a_3 = a^e_k$ that we started with. Replace equations with a'_l, say $a'_l + a_8 + a_9 = 1$, by $a'_8 = 0$ and $a'_9 = 0$. We already know what to do with these equations as well.

So after some fast (i.e. polynomial-time) clean-ups, we have either solved the system of equations, or reduced it to a smaller system of equations such that each pair of equations do not share more than one variable.

2 NP-completeness of Minesweeper on several grids

Richard Kaye showed in [2] that Minesweeper is NP-complete. Did you read that article? I did, and I can tell you that it is one of the coolest articles I ever read. In fact, he builds a logical circuit on the minesweeper board, in order to compute a boolean expression. Next, he enforces the outcome of the boolean expression to be true, whence the problem becomes finding boolean values for the variables such that the logical expression evaluates to true.

We are going to reduce Minesweeper to solving polynomial equations over $\mathbb{F}_2$, for all three regular tesselations of the plane (triangular, normal and hexagonal Minesweeper). For this purpose, we build circuitry as well, this time circuitry to compute polynomials over $\mathbb{F}_2$. This is because my advisor is a specialist in polynomial mappings, this was the only way that he would agree me to write an article so far from his subject. No, just kidding. In figure 3, a circuit with two splitters, one crossover, two multipliers and two adders is given.

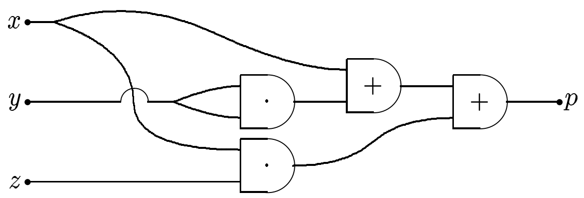

Figure 3: A circuit that computes the polynomial $p = x + y^2 + xz$

In order to make such circuits on a minesweeper board, we first need a way to code wires on it. This is done in figure 4.

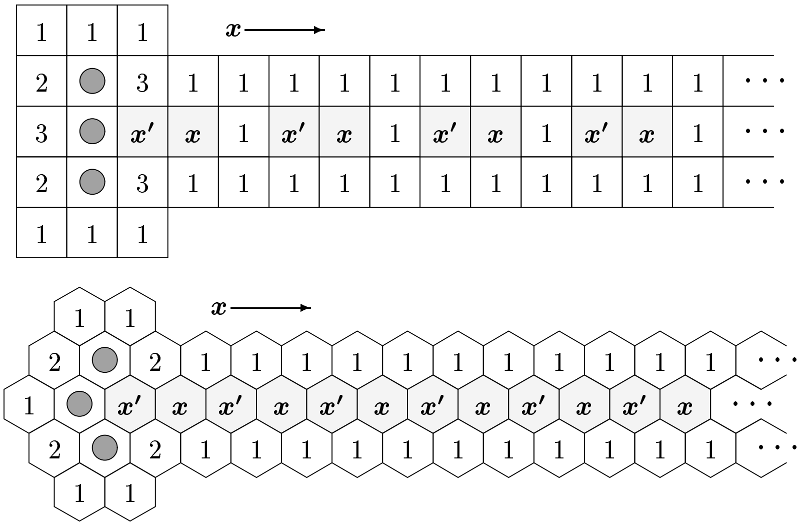

Figure 4: A wire that conducts x, running from the starting point of x

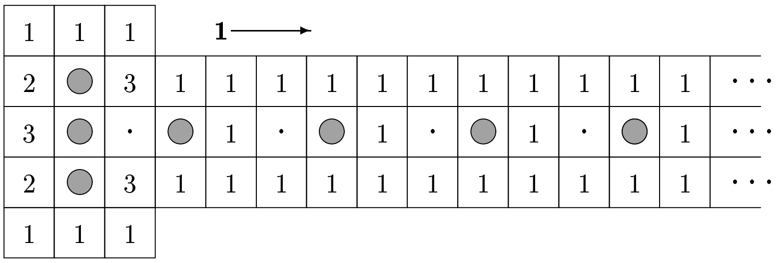

Figure 5: A wire that conducts one

In figure 4, we do not know what the compartments with x and x' contain, but we do know that either exactly all compartments with x contain a mine (figure 5) or exactly all compartments with x' do (figure 6). Which of both cases are possible follows from the global structure of the Minesweeper board. To make polynomial equations from the polynomial expressions, it suffices to force these expressions to have a given value in $\{0, 1\}$. This can be done by forcing a wire that conducts a certain expression to conduct a given value. Figure 7 showns how to force a wire to conduct one and zero respectively.

Small computational components

Next, we need to have curves to bend wires (figure 8) and splitters to duplicate wires (figure 9).

Something that is not shown in figure 3, but that we might need, are so-called phase-shifters (figure 10). The phase-shifter with square compartments is stolen from R. Kaye [2].

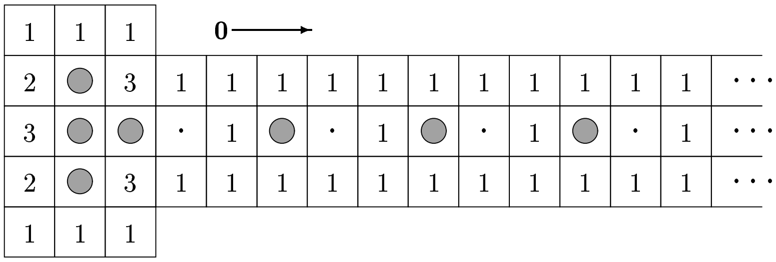

Figure 6: A wire that conducts zero

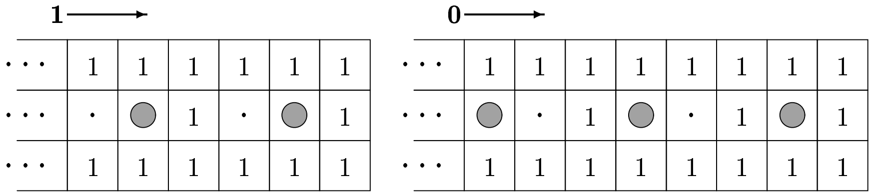

Figure 7: A wire that is forced to conduct a given value

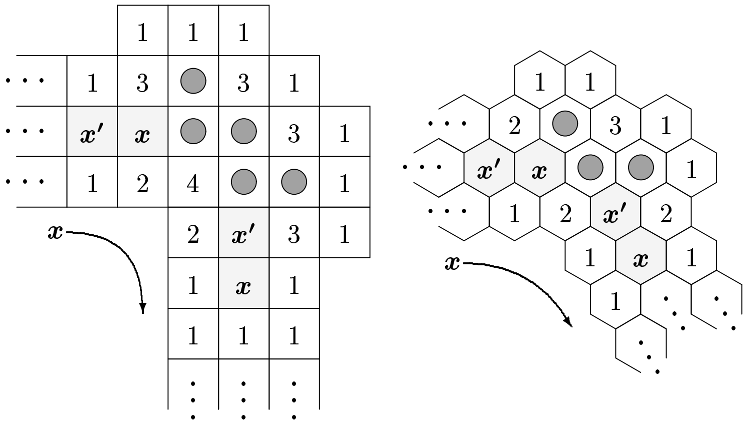

Figure 8: A curve of a wire with x

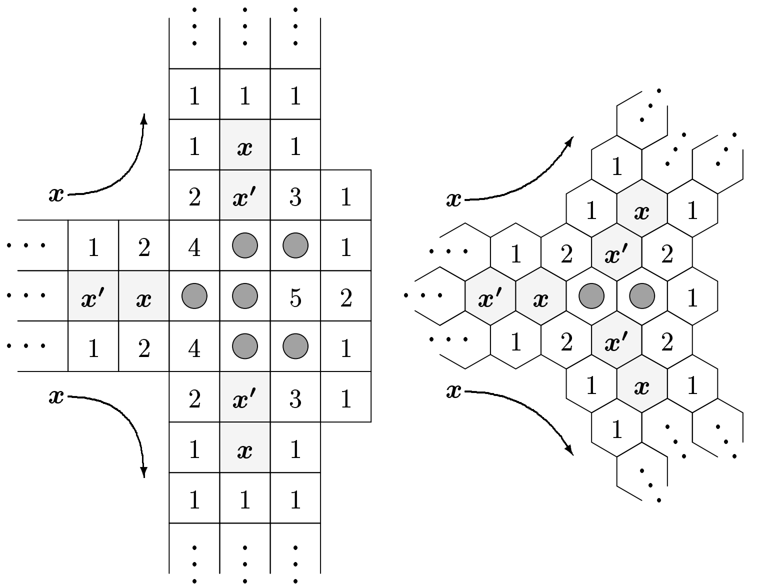

Figure 9: Merge two curves to get a splitter

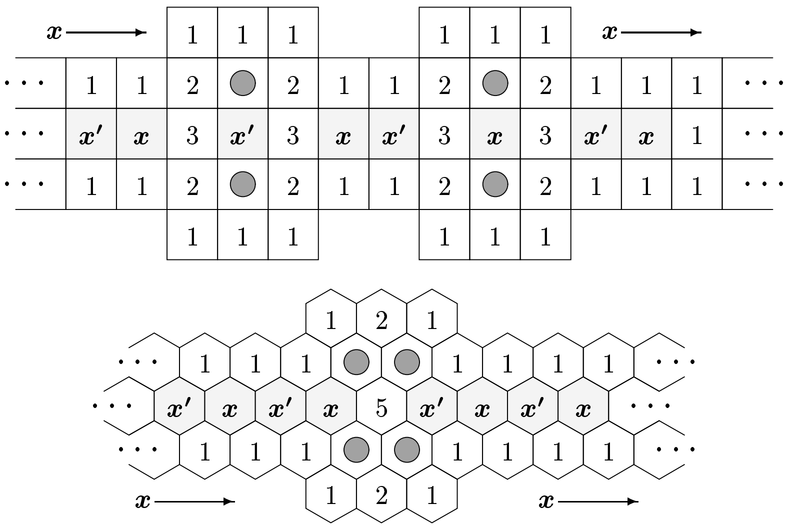

Figure 10: A phase-shifter

You might think that the phase-shifter with hexagonal compartments is a so-called invertor as well. In that case, I do not agree with you, and since this is my article, I am right by definition. A real invertor is shown in figure 11.

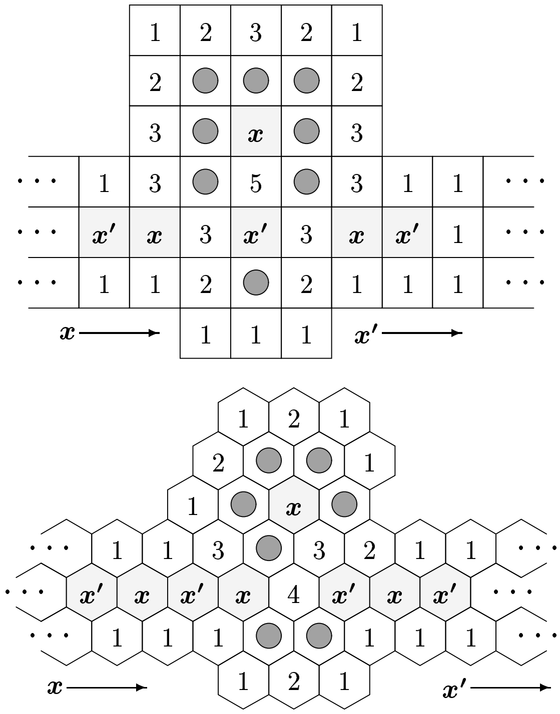

Figure 11: An invertor

In case you might not have noticed already, the hexagonal phase-shifter is not a good invertor because the mine-counter might given information about the circuit then, possibly screwing up all our efforts with it.

Larger computational components

A crossover for Minesweeper with square compartments is shown in figure 12. Look at the square in the center with 3. It is red because it is the heart of the crossover. You might wonder why there is no crossover given with hexagonal compartments. The answer is that I did not found one by direct construction. But a crossover can also be made from three splitters and the same number of adders, see [2] or figure 13.

But before we have such a crossover, we must first make a hexagonal adder. Figure 14 shows an adder-multiplier combi. If you only want to use the adder, you just cut off the wire of the multiplier output. A wire cut-off is the same as the start of a variable in figure 4.

Since the adder-multiplier combi is the most complicated component by far, it needs some explanation. Let us concentrate on the one for normal minesweeper first.

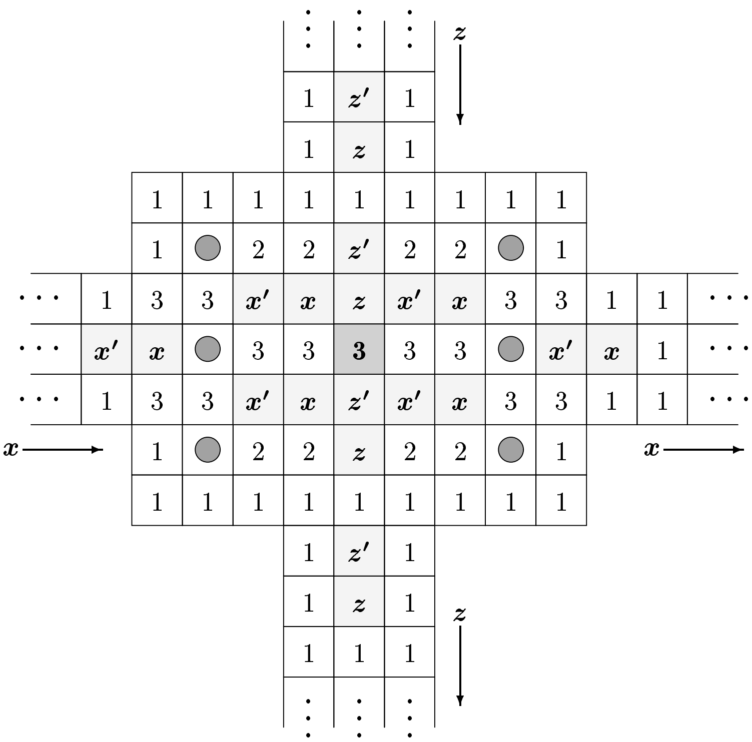

Figure 12: A crossover

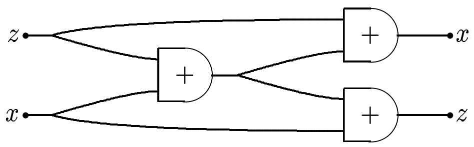

Figure 13: A crossover circuit

The heart on the left hand side with the number 5 enforces the equation $x+z+a'+v' = 2$ over $\mathbb{Z}$. Since also $a+v+a'+v' = 2$ over $\mathbb{Z}$, the sets $\{x, z\}$ and $\{a, v\}$ are enforced to be equal. So the heart on the left hand side in in fact the heart of a *shaker*: its outputs a and v are a nondeterministic permutation of x and z.

We show that $a = \min(x, z)$ and $v = \max(x, z)$. Suppose that this is not the case. Then $a = 1$ and $v = 0$. So $a' = v = 0$. It follows that the heart on the right hand side is surrounded by 3 mines at most. This contradicts the number 4 in it, so $a = \min(x, z)$ and $v = \max(x, z)$.

Since $x \cdot z = \min(x, z)$, the output a equals $x \cdot z$. Next, the heart on the right hand side enforces the equation $a' + v + c' = 2$ over $\mathbb{Z}$. Since $v = 0$ implies $a' = 1$ and $a' = 0$ implies $v = 1$, $1 \leq a' + v \leq 2$ over $\mathbb{Z}$. So c' can be chosen such that $a' + v + c' = 2$ over $\mathbb{Z}$.

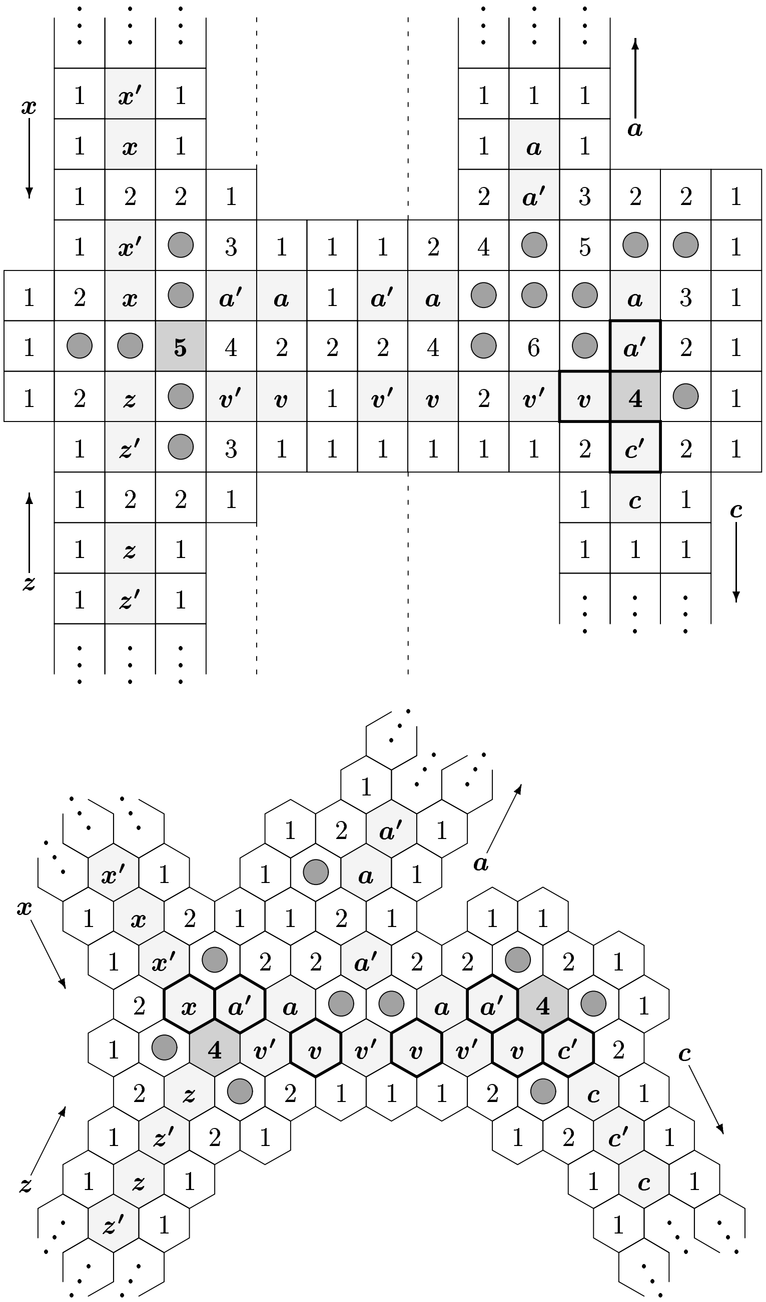

Figure 14: A component that outputs $a = x \cdot z$ and $c = x + z$

Modulo 2, the heart on the right hand side gives the following information:

$$0 \equiv a' + v + c' \equiv a + v + c \equiv x + z + c \pmod 2$$

It follows that $c = x + z$ is satisfied over $\mathbb{F}_2$.

Are we done now with the adder-multiplier combi for normal minesweeper. No, for some of the square compartments are bold. They are bold because revealing these compartments might give new information about the board at first glance. But that only seems so. If you e.g. reveal the bold compartment with v, then $v = 0$ and you already know before revealing it that $a' = c' = 1$.

The adder-multiplier combi for hexagonal minesweeper works essentially the same as that for normal minesweeper. Notice that the heart on the left hand side is not the heart of a buggy shaker this time, due to some bold compartments surrounding it. But the subcomponent is correct within its context.

Triangular Minesweeper

You probably already wondered where the figures with triangular compartments stayed. I have a disappointing announcement to make: there will not be very many of them. One reason is that I can not play triangular Minesweeper. Playing hexagonal Minesweeper with HexMines of William Hause was fun, but I got sick of shareware reminders when I tried to play triangular Minesweeper with Professional Minesweeper by Bojan Urosevic. Another reason is that this article is already quite long due to the large number of figures.

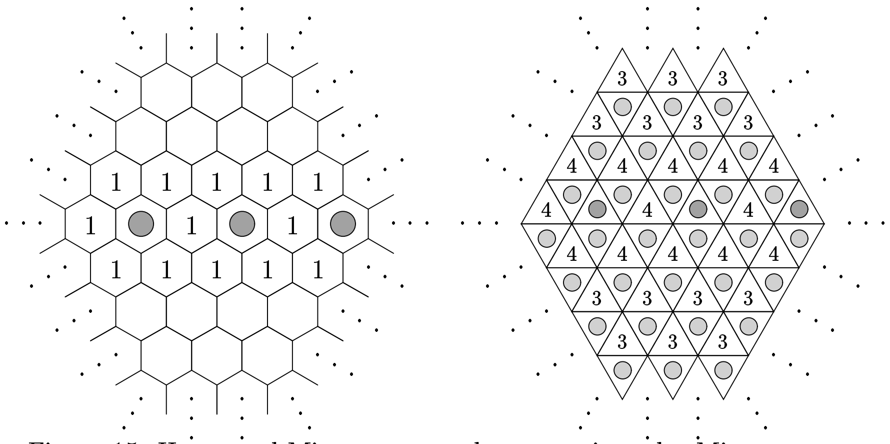

Figure 15: Hexagonal Minesweeper reduces to triangular Minesweeper

In figure 15, each hexagonal compartment on the left hand side is replaced by two triangular compartments: one with a red mine that points downwards and another that points upwards. The triangle that points upwards contains a blue mine if the corresponding hexagonal compartment on the left hand does. Otherwise, it contains the number of the corresponding hexagon plus 3. The "plus 3" counts (for) the red mines surrounding the triangular compartment. So apart for some border issues that are ignored here, we have a reduction from hexagonal minesweeper to triangular minesweeper here. It follows that triangular minesweeper is NP-complete.

2.1 A stronger result on normal Minesweeper

Since Richard Kaye already proved that Minesweeper is NP-complete, it would be nice to improve on that. But how? To observe that Minesweeper is ASP-complete? It is true that all three variants of Minesweeper discussed above are indeed shown to be ASP-complete, while Richard Kaye's proof is not an ASP-proof (if the inputs u and v of his AND-gate are both zero, then r and s can be interchanged).

But that is not what I want to discuss here. No, we are going to show that solving a minesweeper board of which only one square is uncovered initially is NP-complete. Of course, the uncovered square can only be surrounded by zero mines, since otherwise it would be impossible to uncover any other square, in which case you will not get any further.

Ok, say that there is one uncovered square with no mines surrounding it. Then you can uncover all surrounding squares, and for each such square that has no mines surrounding it either you can uncover the surrounding squares as well, etc. All programs for minesweeper do this automatically.

So we get an area of uncovered squares consisting of connected squares with no mines around them, from now on called a *whitespace component*. Furthermore, the border of the first whitespace component is uncovered as well, but the border squares do have mines around them.

In order to show that solving a minesweeper board of which only one square is uncovered initially is NP-complete, it suffices to be able to do the following by local reasoning:

- reason through wires to uncover all whitespace components,
- get to know all components except for the values of their variables.

Figure 16 shows how to get to know wires and to reason through them.

All other components of normal Minesweeper that are presented here can be figured out as well, provided all whitespace components are uncovered and the phases of all wires are known (see figure 16 as well), except the crossover. But as we know already, the crossover is not necessary (although a direct construction is still possible, see figure 17 if the editor has some space left). Furthermore, the adder-multiplier combi needs the part between the dashed lines very sorely now.

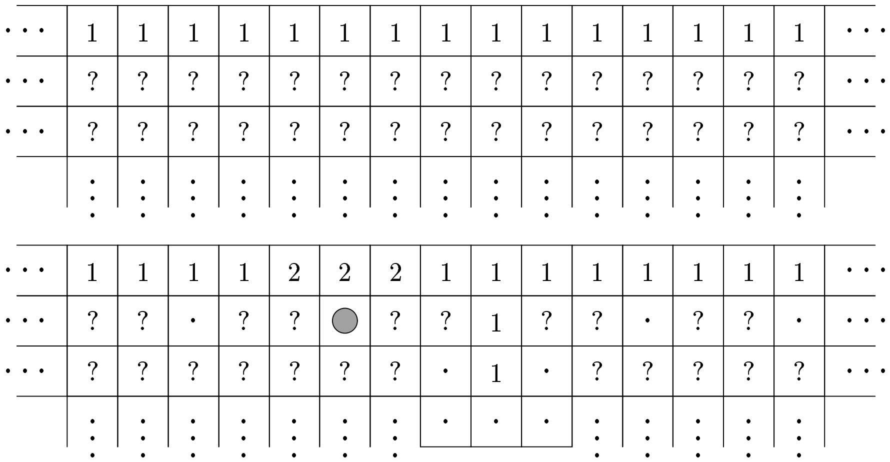

Figure 16: Mark the phase of a wire by an extra mine an you can reason through it

References

[1] D. Eppstein, Computational Complexity of Games and Puzzles, `http://www.ics.uci.edu/~eppstein/cgt/hard.html`

[2] R. Kaye, Minesweeper is NP-complete, *The Mathematical Intelligencer* **22**, nr. 2, 2000.

[3] T. Yato and T. Seta, Complexity and Completeness of Finding Another Solution and its Applications to Puzzles, *IEICE Trans. Fundamentals*, Vol. E86-A, No. 5, pp. 1052-1060, 2003.

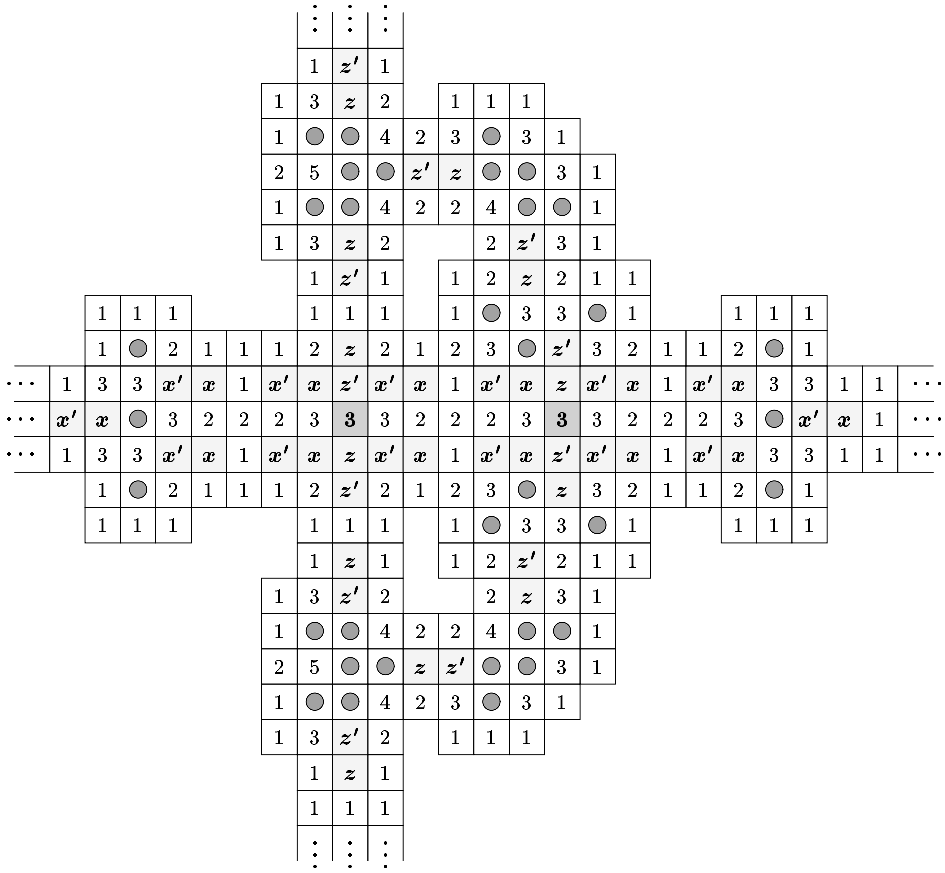

Figure 17: Another crossover