




[INF432] Project - Light Up

Da Costa Barros - Pellieux - Segransan

1st May 2020

1 Rules

A grid of any size (in general a square but it is not mandatory) is composed of :

 : Free Case  N /  : Walls (With or without number N going from 0 to 4)

1. Dispose light to enlighten all the free cases of the grid. A light enlighten all his column and line until the light touch a wall or the border of the grid
2. A light cannot enlighten an other light
3. It is impossible to put a light on a wall.
4. A wall can have a number from 0 to 4. This number is indicating how many light must be placed on an adjacent* box. A wall without numbering is not constraint by this kind of rules.
Ex: In there is a number N written on the wall there should be N light directly touching his side.
If there is no number, there should be any number of light directly touching his side.

*Adjacent : *one side shared so it is not adjacent if there is only a corner shared*

2 Modelisation

We will present how we think to model the game and the Conjunctive Normal Form (CNF) associated to each rules.

First, we have one variable by case denoted by an integer X computed as following $(x + y * width) + 1$ (x and y starting from 0). Hence $X \in [1, width * height]$ and it will be easier to write in DIMACS format as propositional variable must be greater or equal to 1 in this format . If X is true then : “There is a light in the case of coordinate (x, y)”. We have one variable by case to know if we could put a light or not in each case. We call each horizontal(resp. vertical) interval with free cases without wall, starting from a wall / the border of the grid to a wall / the border of the grid, a sub line(resp. row).

1. Rule n°1:

1	2	3	4	5	6
7	8	9	10	11	12

A grid example with their respective propositional logic variable number

Take the first line. To represent the fact that the grid must be fill by light there should be at least one on any sub row or sub line :

$$\sum_{i \in \text{Sline} \cup \text{Srow}} i$$

2. Rule n°2:

To represent the fact that a light could not enlighten. There should be no more than one light on any sub row/line :

$$\begin{aligned} &(\neg 1 \vee \neg 2) \wedge (\neg 1 \vee \neg 3) \wedge (\neg 1 \vee \neg 4) \wedge (\neg 1 \vee \neg 5) \wedge (\neg 1 \vee \neg 6) \wedge (\neg 2 \vee \neg 3) \wedge (\neg 2 \vee \neg 4) \wedge (\neg 2 \vee \neg 5) \\ &\wedge (\neg 2 \vee \neg 6) \wedge (\neg 3 \vee \neg 4) \wedge (\neg 3 \vee \neg 5) \wedge (\neg 3 \vee \neg 6) \wedge (\neg 4 \vee \neg 5) \wedge (\neg 4 \vee \neg 6) \wedge (\neg 5 \vee \neg 6) \end{aligned}$$

Modelling this example we distinguish two parts, the red one that assure that there will be one light on the sub line or the sub row and the blue one that prevent to have two light on a sub line. To have the general formula (in Boolean algebra) we have to exclude two by two each pair and to have big clause with all the elements of $\text{subline} \cup \text{subrow}$.

$$\prod_{(i,j) \in \text{Sline} \cup \text{Srow}} (\bar{i} + \bar{j}) * \sum_{i \in \text{Sline} \cup \text{Srow}} i$$

3. Rule n°3 :

The most obvious rule to model is the rule n°3 . If there is a wall empty or not on the case X we had the clause $\neg X$ to our final CNF. (or \bar{X}).

Now, it last to model the walls with number and using some exclusions formulas.

4. Rule n°4 :

As explain before, we will simply call the top, left, right and bottom respectively 2,4,6 and 8. In a more complex grid top, left, right and bottom will respectively be calculated as following $N - \text{width}$, $N - 1$, $N + 1$ and $N + \text{width}$. We consider this sub case as only neighbourhood box matter and it will simplify the formula.

1	2	3
4	N	6
7	8	9

In this context we have 5 cases (but it's like we have only 3) and their formulas.

- (a) For $N = 0$: $\neg 2 \wedge \neg 4 \wedge \neg 6 \wedge \neg 8$ which is a CNF with each clause containing one variable.
- (b) Symmetrically, for $N = 4$: $2 \wedge 4 \wedge 6 \wedge 8$
- (c) For $N = 1$,

$$\begin{aligned}
& (1 \vee 2 \vee 3 \vee 4) \wedge (\neg 1 \vee \neg 2 \vee \neg 3 \vee \neg 4) \wedge \\
& (1 \vee \neg 2 \vee \neg 3 \vee \neg 4) \wedge (\neg 1 \vee 2 \vee \neg 3 \vee \neg 4) \wedge (\neg 1 \vee \neg 2 \vee 3 \vee \neg 4) \wedge (\neg 1 \vee \neg 2 \vee \neg 3 \vee 4) \wedge \\
& (\neg 1 \vee \neg 2 \vee 3 \vee 4) \wedge (\neg 1 \vee 2 \vee \neg 3 \vee 4) \wedge (\neg 1 \vee 2 \vee 3 \vee \neg 4) \wedge (1 \vee \neg 2 \vee \neg 3 \vee 4) \wedge (1 \vee \neg 2 \vee 3 \vee \neg 4) \wedge \\
& (1 \vee 2 \vee \neg 3 \vee \neg 4)
\end{aligned}$$

On the one hand the black assure at least one is true. On the other hand, the blue parts assure that there is not 4 light adjacent to the case. Green is killing the model with three light and red the one with 2 light.

(d) For N=3, only the green part is inverted and this give :

$$\begin{aligned}
& (1 \vee 2 \vee 3 \vee 4) \wedge (\neg 1 \vee \neg 2 \vee \neg 3 \vee \neg 4) \wedge \\
& (\neg 1 \vee 2 \vee 3 \vee 4) \wedge (1 \vee \neg 2 \vee 3 \vee 4) \wedge (1 \vee 2 \vee \neg 3 \vee 4) \wedge (1 \vee 2 \vee 3 \vee \neg 4) \wedge \\
& (\neg 1 \vee \neg 2 \vee 3 \vee 4) \wedge (\neg 1 \vee 2 \vee \neg 3 \vee 4) \wedge (\neg 1 \vee 2 \vee 3 \vee \neg 4) \wedge (1 \vee \neg 2 \vee \neg 3 \vee 4) \wedge (1 \vee \neg 2 \vee 3 \vee \neg 4) \wedge \\
& (1 \vee 2 \vee \neg 3 \vee \neg 4)
\end{aligned}$$

(e) Finally, for N = 2, it re-use the same patterns:

$$\begin{aligned}
& (1 \vee 2 \vee 3 \vee 4) \wedge (\neg 1 \vee \neg 2 \vee \neg 3 \vee \neg 4) \wedge \\
& (\neg 1 \vee 2 \vee 3 \vee 4) \wedge (1 \vee \neg 2 \vee 3 \vee 4) \wedge (1 \vee 2 \vee \neg 3 \vee 4) \wedge (1 \vee 2 \vee 3 \vee \neg 4) \wedge \\
& (1 \vee \neg 2 \vee \neg 3 \vee \neg 4) \wedge (\neg 1 \vee 2 \vee \neg 3 \vee \neg 4) \wedge (\neg 1 \vee \neg 2 \vee 3 \vee \neg 4) \wedge (\neg 1 \vee \neg 2 \vee \neg 3 \vee 4) \wedge
\end{aligned}$$

In the case where there is less than 4 adjacent boxes, we consider the boxes outside the grid as a wall (by Rule 3 the variable associated to is false). As explain before, we do it during the construction of the CNF by the algorithm. We don't want to compute value for case outside of the grid which implies to consider a grid of $(width + 2) * (height + 2)$ and consider the border boxes as wall. It's more complex for each grid given to the algorithm but it still the solution for less algorithm computations and more sat solver one. The problem is why consider a more complicated grid instead of getting rid of this problem during the building of the CNF. For example, if $(x, y+1) \in R(i)$ and $y+1 > width$, we consider the same clause without the $R(i)$ as if $R(i)$ is false (a wall), we can just forget it as $X \vee \perp \equiv X$. We just don't add a variable if it's outside the grid.

If there is a numbered wall with not enough adjacent boxes (too much outside the grid). The program will stop and generate a basic unsolvable CNF $(\neg i \wedge i)$ as it is unsolvable. Again to avoid considering the grid surrounded by wall as it increase quickly the numbers of variables (as it depend on the height and width of the grid).

3 Modeling the problem using first order logic

Small remember the index of box at coordinate (x, y) is calculated as follow : $i = (x + y * width) + 1$. Relation $SubC(i, j)$ is true if and only if i and j is in the same sub column. Similarly, $SubR(i, j)$ is true if and only if i and j is in the same sub row. Another relation needed is $W(i)$ which is true if and only if case of index i is a Wall (with or without number). The constraints of the problems are modelled as follows :

- Dispose light to enlighten all the free cases of the grid : $\forall i, \exists j, SubC(i, j) \Rightarrow j$
 $\forall i, \exists j, SubR(i, j) \Rightarrow j$
- A light cannot enlighten an other light : $\forall i, \forall j, (SubC(i, j) \wedge (i \neq j) \Rightarrow \neg i \wedge \neg j)$
 $\forall i, \forall j, (SubR(i, j) \wedge (i \neq j) \Rightarrow \neg i \wedge \neg j)$
- It is impossible to put a light on a wall : $\forall i, (W(i) \Rightarrow \neg i)$
- A wall can have a number from 0 to 4. This number is indicating how many light must be placed on an adjacent box. A wall without numbering is not constraint by this kind of rules :
 We can introduce a new function which map a wall to the number written on it, we call it N. Moreover, functions $A(i)$, $B(i)$, $L(i)$, $R(i)$ correspond respectively to the index of the above box, to the index of the bellow box, to the index of the left box and to the index of the right box.

- $\forall i(W(i) \wedge N(i) = 0 \Rightarrow \neg A(i) \wedge \neg B(i) \wedge \neg L(i) \wedge \neg R(i))$
- $\forall i(W(i) \wedge N(i) = 1 \Rightarrow (A(i) \vee B(i) \vee L(i) \vee R(i)) \wedge (\neg A(i) \vee \neg B(i) \vee \neg L(i) \vee \neg R(i)) \wedge$
 $(\neg A(i) \vee B(i) \vee L(i) \vee R(i)) \wedge (A(i) \vee \neg B(i) \vee L(i) \vee R(i)) \wedge$
 $(A(i) \vee B(i) \vee \neg L(i) \vee R(i)) \wedge (A(i) \vee B(i) \vee L(i) \vee \neg R(i)) \wedge$
 $(\neg A(i) \vee \neg B(i) \vee L(i) \vee R(i)) \wedge (\neg A(i) \vee B(i) \vee \neg L(i) \vee R(i)) \wedge (\neg A(i) \vee B(i) \vee L(i) \vee \neg R(i)) \wedge$
 $(A(i) \vee \neg B(i) \vee \neg L(i) \vee R(i)) \wedge (A(i) \vee \neg B(i) \vee L(i) \vee \neg R(i)) \wedge (A(i) \vee B(i) \vee \neg L(i) \vee \neg R(i)))$
- $\forall i(W(i) \wedge N(i) = 2 \Rightarrow (\neg A(i) \wedge \neg B(i) \wedge \neg L(i) \wedge \neg R(i)) \wedge (A(i) \wedge B(i) \wedge L(i) \wedge R(i))) \wedge$
 $(\neg A(i) \vee B(i) \vee L(i) \vee R(i)) \wedge (A(i) \vee \neg B(i) \vee L(i) \vee R(i)) \wedge$
 $(A(i) \vee B(i) \vee \neg L(i) \vee R(i)) \wedge (A(i) \vee B(i) \vee L(i) \vee \neg R(i)) \wedge$
 $(A(i) \vee \neg B(i) \vee \neg L(i) \vee \neg R(i)) \wedge (\neg A(i) \vee B(i) \vee \neg L(i) \vee \neg R(i)) \wedge$
 $(\neg A(i) \vee \neg B(i) \vee L(i) \vee \neg R(i)) \wedge (\neg A(i) \vee \neg B(i) \vee \neg L(i) \vee R(i)))$
- $\forall i(W(i) \wedge N(i) = 3 \Rightarrow (A(i) \vee B(i) \vee L(i) \vee R(i)) \wedge (\neg A(i) \vee \neg B(i) \vee \neg L(i) \vee \neg R(i)) \wedge$
 $(A(i) \vee \neg B(i) \vee \neg L(i) \vee \neg R(i)) \wedge (\neg A(i) \vee B(i) \vee \neg L(i) \vee \neg R(i)) \wedge$
 $(\neg A(i) \vee \neg B(i) \vee L(i) \vee \neg R(i)) \wedge (\neg A(i) \vee \neg B(i) \vee \neg L(i) \vee R(i)) \wedge$
 $(\neg A(i) \vee \neg B(i) \vee L(i) \vee R(i)) \wedge (\neg A(i) \vee B(i) \vee \neg L(i) \vee R(i)) \wedge (\neg A(i) \vee B(i) \vee L(i) \vee \neg R(i)) \wedge$
 $(A(i) \vee \neg B(i) \vee \neg L(i) \vee R(i)) \wedge (A(i) \vee \neg B(i) \vee L(i) \vee \neg R(i)) \wedge (A(i) \vee B(i) \vee \neg L(i) \vee \neg R(i)))$
- $\forall i(W(i) \wedge N(i) = 4 \Rightarrow A(i) \wedge B(i) \wedge L(i) \wedge R(i))$

4 Resolution by a SAT-solver

We solve using minisat and reading the result file. The format of the result is the following. The first line is "SAT" or "UNSAT". If it is "UNSAT", there is no solution to the problem and nothing is following. If it is "SAT", the following line give the solution. There is a assignation for each variable i separated by space. If it is "i" it means it's true in the model found, if it's "-i" it's means the variable is false in the model found. Reading the output file, for each box, we put a lamp on it box if his index i is true in the model and we display the corrected grid.

Example of output for solvable 3 by 3 grid

SAT

-1 -2 3 4 -5 -6 -7 8 -9 0

Example of output for unsolvable

UNSAT

5 Optional part : SAT solver

We use the same algorithm describe in the project description (WalkSat : see 6). Here a table summing the best heuristics and constant choice :

$P \setminus N$	0	0.1	0.2	0.4	0.6	0.8	1
100							
1000	775,2	658,2	767,1	696,1	677,5		
10000	4924,4	4.967,3	5.346,6	5.636,1	6.153,2	6.681,1	6258,0
100000	11558,6	10.927,4	12.943,6	19.538,3	30.760,3	44.369,4	58942,0

Mean number of flip on 1000 executions to find a model depending on constants P and N

$P \setminus N$	0	0.1	0.2	0.4	0.6	0.8	1
100	0	0	0	0	0	0	0
1000	29	41	29	10	2	0	0
10000	584	565	496	321	148	62	9
100000	1000	1000	1000	1000	980	839	360

Number of model found on 1000 executions depending on constants P and N

To sum up, the tables show that the number of solutions found increase with P (which is the maximum number of flip), it is instinctive that more the algorithm try more he have a chance to find a solution. About N , we can see that a small amount of random can be greater than 0 for $P=100.000$ but we need to keep in mind that it depend on the sample. In general, when N increase the part of random in choosing the variable to flip increase and the number of model found decrease and the mean number of flip to find a model increase. For $N \leq 1000$, the data about P is not significant because we never find more than 30 model.

Mean number of flip and percentage of success to find a model depending on heuristics ($P=0,1$, $N=100.000$, 1.000 executions)

Heuristics	JW	MOMS	Score	Least Modified
Flip needed	10.927,4 (100%)		25.610,6 (97.5%)	1.071,8 (100%)

Table 1: 3-SAT problem

Heuristics	JW	MOMS	Score	Least Modified
Flip needed	479,0 (100%)	27.064,5 (77,1%)	5.886,3 (100%)	107,8 (100%)

Table 2: SAT problem

We test the program with the same problem on the different heuristics but it's 3-Sat reduced before the test of the first table. The 3-SAT reduction increase the number of variable and the number of clause but make the comparison. We see that it's harder for the algorithm to solve the 3-SAT reduced problem and we observe it on the table as for each heuristics choice the average of number of flip needed is higher on the 3-SAT problem. The interest in MOMS heuristics is reduced as there is no "small clause" they are all of size 3 and can explain his bad performance. We just observe that the best heuristics is the "Least modified" one when a 3SAT problem is provided. It should come from the fact that it flip the first literal encountered at first flip on a clause because nothing has been modified. The interesting variable are always the second one and it come it really fast after the first flip. Actually, this heuristics work nice with our 3-Sat reduction algorithm. Then more generally heuristics from the best to the worst are JW, MOMS and Score.

For the output, we copy the same format as the Minisat output to just replace it and keep the same executable to display the correction.

6 Random generation

: Empty wall 0,1,2,3,4 : Numbered walls
 _ : Free case . : Case that should be empty (Cleared after generation)

We have a generator of grid coded in C that is working nearly good the two problem is that if we asked a not that high density of a wall it will give an unsolvable grid. In fact, the proportion of solvable grid depend on the density asked. The real problem is that we don't check that each box can be enlighten and we only focus on numbered wall which is the source of a lot of unsolvable grid if not handled. We can have this kind of pattern for example making the grid unsolvable :

```
#-
_1
#-
```

As we need, to light the 3 case putting a lamp on them but we can only put one lamp on the 3 free boxes. Our generator allow that as we can put 1 lamp next to the 1 wall but don't verify that we could light all the grid.

Finally, the grid is not completely solved at the end of the generator only the part with wall it is and just have to remove the light from the grid to make it playable. Here is some grid at different size generated:

Density : 0.3 Size : 7 (solvable)

```
  - - - - 1 -
  - 0 _ 0 - -
  - - 0 - - -
  - - - - 3
  - - - - -
  - - - - # -
  - - - - -
```

Density : 0.3 Size : 7 (solvable)

```
  - - - - 3 -
  - - - # - -
  - - - - -
  - - - 0 - -
  - - 1 - # -
  1 - - - # -
  - - - # - -
```

Density : 0.4 Size : 7 (unsolvable)

```
  2 - - - - -
  - - 0 _ 0 -
  - - 1 - - 2
  0 - - - # -
  - # - 1 - # -
  - - 0 _ 2 - #
  1 - - - - # #
```

Density : 0.1 Size : 20 (Solvable generated after several attempt)

```
- - - - # # 1 1 - - - # - - - -
- - - - - - - 0 - - 0 - - - -
- - - - - - - - - - - - - - -
- - - - 0 - - - - - - - 0 - -
- - - - # 1 - - - - - - - - -
- - - - 0 - - - - 0 - - - - -
- - - - 1 - - - - - - - - -
- 0 - - - - - # - - - - - -
- - - - - - - - - - - - - -
- - - - - - - - - - - 0 - -
- - - # - - - - - - - - - -
- - - # - - - - 0 - 0 - 0 - -
- - - # # - - - - - - - - -
- - - 0 - - - - - - - - - -
- - - - - - - 0 - - - - -
0 - - - - 0 - - - - - 0 - - #
- - - - - - - - - - - 0 - #
- - - 0 - - - - - - - - - -
- - - # - - - - - - - - - -
- - - - - - - 2 - - - - -
```

Annexes

sec:Annexes

WalkSat algorithm :

```
1: Draw an assignment v at random according to a uniform distribution
2: i = 0
3: WHILE (v is not a model) and i < N DO
4:   Pick a clause C amongst clauses C' such that v(C') = false
5:   Draw a real number q in [0,1] according to a uniform distribution
6:   IF q < P THEN
7:     Pick a variable x in C according to a uniform distribution
8:   ELSE
9:     Pick a variable x in C deterministically
10:  END IF
11: Flip the value of v(x) in the assignement v
12: i++
13: END WHILE
14: IF v is a model THEN
15:  RETURN v
16: ELSE
17:  RETURN "undecided"
18: ENDIF
```