Exploring the NP-Complete Space with Nonograms

Scott Ratchford

Abstract

A Nonogram puzzle (also known as Picross puzzle) is a type of puzzle that is NP hard to solve. For proof of this, see "Complexity and solvability of Nonogram puzzles" by R.A. Oosterman. Oosterman proved that the Nonogram problem is reduceable to SAT. He also proved that the Bounded Nondeterministic Constraint Logic (NCL) problem is reduceable to Nonogram, thus confirming Nonograms as NP-Complete. In this paper, I will explain how to reduce the Nonogram to SAT and how to reduce SAT to Nonogram. Familiarity with Nonogram puzzles and their operation is required to understand this paper, but a brief introduction is included.

Introduction

The code present in the solution is largely self-documenting. This code explains a majority of the mappings to and from Nonograms. The figures below also show the mappings and demonstrate multiple examples of the mappings.

Nonogram Definition

A classic Nonogram consists of sets of horizontal clues, sets of vertical clues, and a grid of size n by m. The code included implements the Nonogram with an additional feature, the key.

Line will be used to refer to either a row or column. Each line has a corresponding set of clues.

A Nonogram solution consists of a grid of size n by m, where n, m ≥ 1. Each cell in the solution has a value equal to 0 or 1. Each row and each column of the Nonogram has a corresponding set of clues. Each clue represents a number of consecutive cells in that line that are equal to 1 in the solution. No other cells in that line should be equal to 0.

A Nonogram is *solvable* if there are one or more combinations of cells such that all the sets of clues are satisfied. A Nonogram is *unsolvable* if there is no combination of cells such that all the sets of clues are satisfied. A solvable Nonogram may have either a single solution or multiple solutions.

The number of clues in a set is the cardinality of the set of clues. This is represented by |clues| in the following constraint.

$$0 \leq |\text{clues}| \leq floor(\frac{|row|}{2}) + (|row|\%2)$$

Thus, a set of clues may be empty. It may also contain any positive integer number of clues up to half the size of the row, if and only if the row has an even size. If and only if the size of the row is an odd number, the row may contain a number of clues equal to half the size of the row, rounded down, plus one.

The size of the largest possible clue, *y*, in a set, *S*, is subject to the following constraint.

$$1 \leq y \leq |line| - (2 * |clues|) + 2$$

Y is a positive integer value. Y cannot be less than 1, because if y were less than 1, there would be no clues in the set, making the set empty. Thus, the set would not contain y.

In a Nonogram with n columns and m rows, $|column| = m$ and $|row| = n$ for any row or column in the Nonogram.

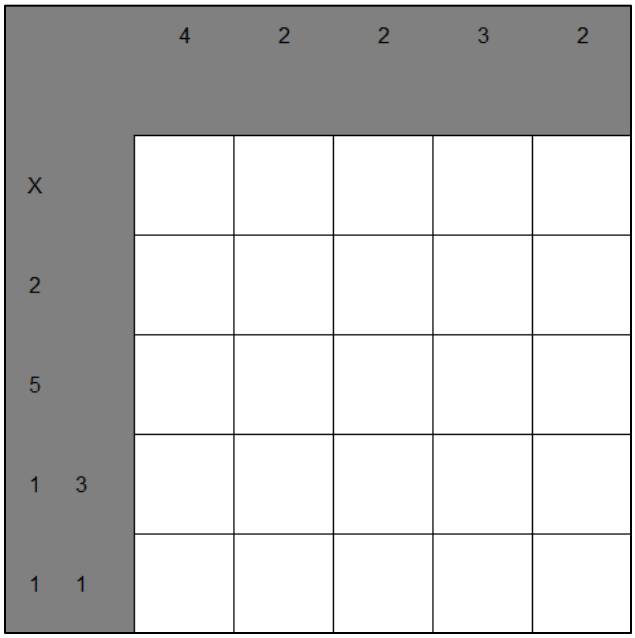


Figure 1 – example of an unsolved 5 by 5 Nonogram, with its clues next to the corresponding lines

In the implementation of Nonograms provided by the included code, a key is generated first and the clues are populated accordingly. This guarantees all generated Nonograms have at least one solution. For a given Nonogram, it is possible that there are multiple solutions, but in such cases, it can never be determined which solution is the key generated by the program without attempting solutions blindly until the key is found. For further consideration, see Footnote 1 in the appendix.

Method of Solving Nonograms

Solving by Brute Force

A Nonogram has 2^{n*m} possible combinations of cell values, making brute force attempts to solve the puzzle incredibly inefficient. Instead of checking every possible combination of values against the key, the included code attempts to solve the puzzle heuristically. If the heuristic attempt fails, a *partial brute force* method is attempted. In the partial brute force method, all cells that the heuristic method successfully confirmed to be either a 0 or 1 are left unchanged. The partial brute force method tries every combination of the remaining undetermined cells until it solves the puzzle. This could be anywhere from 4 cells to 2^{n*m} cells but is usually less than or equal to $½ * (4 + 2n * m)$ attempts.

Solving by Heuristic

The included code represents each cell with the following main states:

State	Integer Value	Description
UNKNOWN	-1	Unknown status
FILL	1	Cell should be 1
ELIM	0	Cell should be 0

A line is *trivial* to solve if its set of clues is empty, if $|clues| = 1$ and the only element of clues is equal to |line|, or if $|line| = sum(clues) - |clues| - 1$. If the set of clues is empty, the line must consist only of cells of value 0. If the set of clues contains only a single clue and that clue is equal to the size of the line, the line must consist only of cells of value 1. If the size of the line is equal to the sum of all its clues, minus the number of clues, minus 1, it can be guaranteed that there is only a single possible solution to the line. The clues, with a single cell of value of 0 between each, add to the size of the line.

At the start of the heuristic algorithm, the program searches for overlapping clues and edits the values of the cells accordingly. A line is said to have *overlapping clues* if every possible solution of the line has an overlapping cell from the same clue.

Figure 2a – the line and its clue set

Figure 2b – first of the three possibilities

Figure 2c – second of the three possibilities

Figure 2d – last of the three possibilities

Figure 2e – the overlapping cell

Next, it repeatedly runs *solve_line* on every row and every column (with the line's corresponding clues) until no changes are made after operating on each row and column.

The following section presents the steps of the *solve_line* function, which returns an integer array with a length equal to that of its first parameter.

1. *Solve_line* is called. Its parameters are a single line of the puzzle and its corresponding clues. It is assumed that the line and its clues are valid.
2. If the line is already completed, the line is returned and execution halts.
3. Otherwise, the function continues.
4. If the line is trivial to solve, it is filled in accordingly and returned.
5. Otherwise, the function continues.

6. If there are eliminated cells on either end of the line as it was passed into the function, *solve_line* is called recursively on a subset of the line with the same clues as before. The subset of the line is smaller than the line by the number of eliminated cells on the ends.
7. The line is returned, potentially with cells with updated values. No cells will ever be changed from 0 or 1 to -1. Cells may or may not be changed from -1 to 0 or 1.

There are additional steps that could be taken to improve *solve_line*, but only the above steps have been implemented. The steps above are sufficient to solve most Nonograms of size n by m, where $1 \leq n, m \leq 4$.

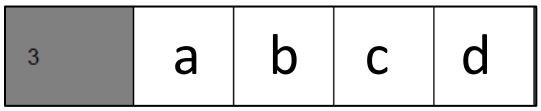
When the heuristic solver fails, it calls the partial brute force algorithm to complete the solution.

Mappings

Nonogram to SAT

Any Nonogram line's key can be expressed in SAT form with a number of variables equal to the size of the line. The SAT will have a number of clauses equal to the number of variables. Each clause will either be (variable) if the cell is equal to 1 or (NOT variable) if the cell is not equal to 0.

A Nonogram line's clue set can also be converted to SAT form, given the size of the line. An example is given in Figure 3.



$b \cap c \cap (a \cup d) \cap (\neg a \cup \neg d)$

Figure 3 – converting the clue set {3} to SAT form with a line of size 4

Because of the overlap principle discussed in the section "Solving By Heuristic", both cells b and c must have value 1 (True). Because of the clue 3, only three cells may have value 1 (True) and the remaining cells must have value 0 (False). With $|line| = 4$, this leaves one cell to have the value 0. Because two cells have already been designated with value 1, we know that only one cell that remains may have the value 1. Unfortunately, without other clues to guide us, we cannot determine the values of cells a or d. This leaves us with two cells of unknown value.

If the heuristic algorithm were to encounter the situation in Figure 4 and did not have the clues for the columns containing cells a and d, it would leave the two unknown cells as they are and finish executing. Because not all required cells were set to value 1, the Nonogram would not yet be solved. Thus, the partial brute force algorithm would be used on the remaining cells. As with any execution of the partial brute force algorithm, there would be 2^x possible solutions, where *x* is the number of unknown cells remaining.

Let us consider Figure 3 further. Suppose a Nonogram, *Q* has four columns and one row. The only row of Q has the corresponding clue set S_x, where $S_x = \{3\}$. The columns of a Nonogram have a length equal to the number of rows. Thus, the columns of Nonogram Q are each of length 1. The number of possible keys to a line of size *n* is 2^n. $n = 1$ and $2^1 = 2$. Each column of Q has the possible set of clues S_1.

$$S_1 = \{\emptyset, \{1\}\}$$

Each of the elements of S_1 has one possible key on a line of size 1. Because a line may only have one set of clues, we can use the set of clues for each column in Q to trivially determine the key to each line. Because we can determine the key to each column in Q, we can determine the value of each cell in Q. Thus, given a Nonogram with at least one dimension of exactly size 1, we can determine every cell of the Nonogram. Therefore, it must be solvable and have a unique solution.

Using the principle of overlapping clues, we know that there are two possible Nonograms Q, known as Q_1 and Q_2. These discrete Nonograms have the same set of clues on their sole rows. Their second and third columns have the same sets of clues. However, their first and fourth columns have swapped clue sets.

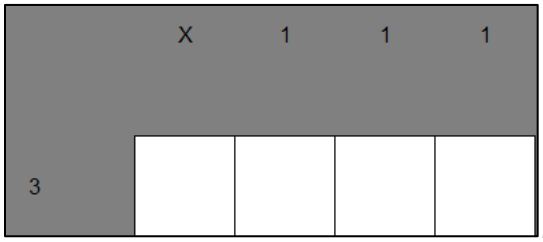


Figure 4a – Nonogram Q_1

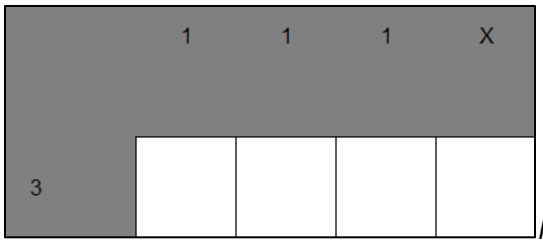


Figure 4b – Nonogram Q_2

For any Nonogram, if either exactly one row or exactly one column contains unknown cells, the Nonogram is solvable and has a unique solution. This is because this nearly-solved Nonogram can be reduced to a Nonogram with at least one dimension of size 1.

Consider the Nonogram *G* with two rows and four columns. G has rows R_1 and R_2. G also has columns C_1, C_2, C_3, and C_4. The corresponding sets of clues for R_1 and R_2 are {4} and {1, 1}, respectively. The corresponding sets of clues for C_1, C_2, C_3, and C_4 are {2}, {1}, {2}, and {1}, respectively.

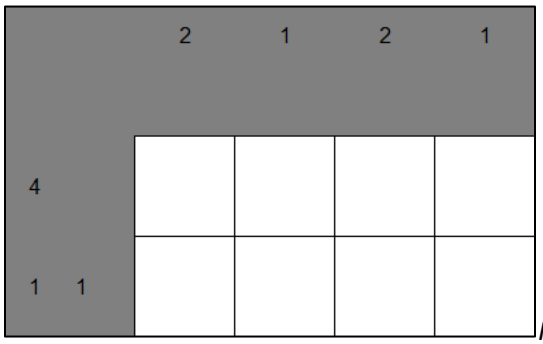


Figure 5a – Nonogram G before solving

In this instance, the Nonogram is trivially solvable, but the principles described in this situation are applicable to any Nonogram with either exactly one row or exactly one column that contains unknown cells.

Let us solve R_1, as it is trivial. The Nonogram can now be treated as an equivalent Nonogram, *G'*. We shall solve both to prove the point.

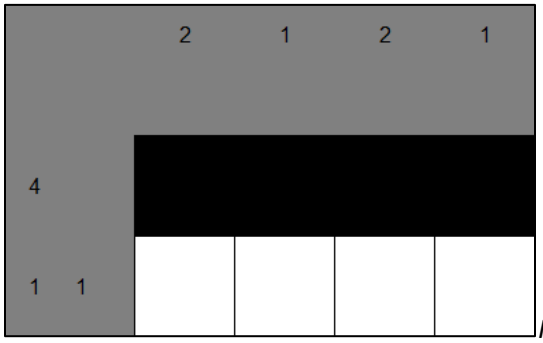


Figure 5b – Nonogram G after solving R_1

Because we have determined the correct values of every cell in R_1, we can now confirm more cells in each column of G. We can eliminate the remaining cells of C_2 and C_4.

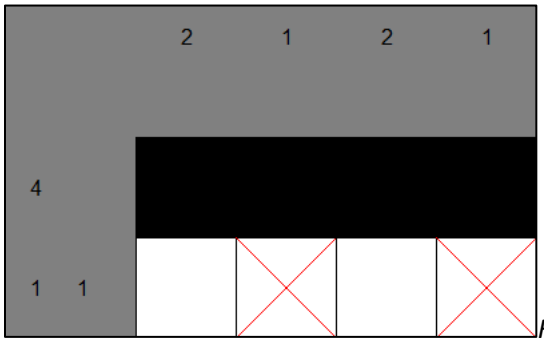


Figure 5c – Nonogram G after confirming the remaining cells of C_2 and C_4

There are now two remaining cells in R2, and the clue set of R2 sums to two. Thus, the clues now fit perfectly in the remaining cells. We can finish the Nonogram.

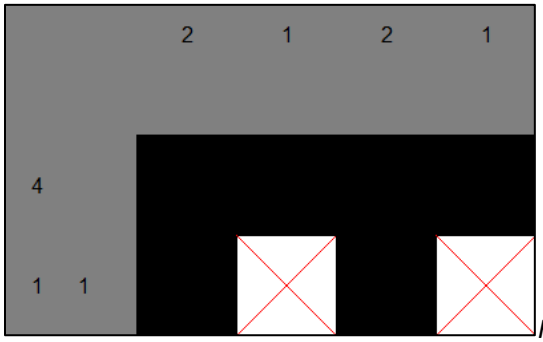


Figure 5d – Nonogram G after solving all cells

Let us consider Nonogram G', which is equivalent to Figure 5b. Nonogram G' has one row and four columns. Its clues differ from those of G, which is necessary to produce the same key for the single row of G' as the second row of G.

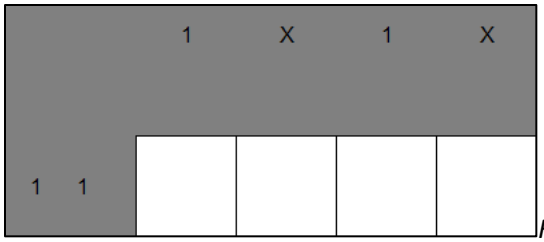


Figure 5e – Nonogram G'

The process of determining how to convert a Nonogram with a single unsolved line remaining to a Nonogram containing only that line is left as an exercise to the reader.

A set of clues that is not close to being trivial may be converted to multiple SAT problems instead of just one. A Nonogram can be converted to a k-SAT problem by creating a k-SAT problem out of every line's set of SAT problems, then creating a k-SAT problem out of those k-SAT problems. This is left as an exercise to the reader.

Thus, any Nonogram problem may be mapped to one or more SAT puzzles.

SAT to Nonogram

Every satisfiable SAT problem can be reduced to one or more sets of clues for a Nonogram line. A SAT with *x* variables can be converted to a line of length *x*.

If the SAT has a unique solution, it may be reduced to a unique Nonogram. If the SAT has a unique solution, it may be reduced to either a single Nonogram with nonunique solutions or multiple Nonograms with potentially unique solutions.

In some cases, it may be possible to reduce an unsatisfiable SAT to an unsolvable Nonogram. Consider the following SAT, known as *K*. We shall call the first, second, third, and fourth clauses of this SAT problem as A, B, C, and D, respectively. This are not to be confused with the individual cells of the Nonogram *Z*.

SAT K: $(\neg a \cup \neg b) \cap (\neg c \cup \neg d) \cap (a \cup c) \cap (\neg b \cup \neg d)$

This SAT can be reduced to Nonogram B, as depicted in Figure 6.

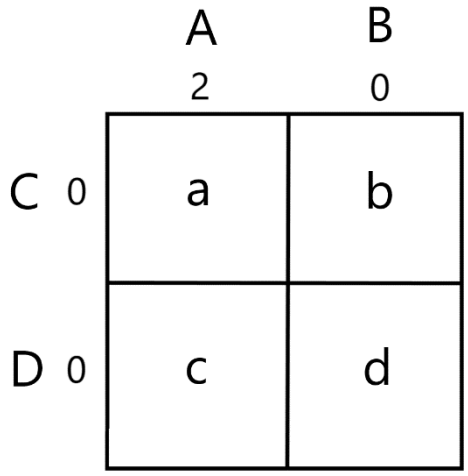


Figure 6 – Nonogram Z

Nonogram Z is unsolvable, which means that SAT K is unsatisfiable. Nonogram Z can be proven to be unsolvable quite simply by the Pigeonhole Principle. Because of the clue set of line A, cell a must equal 1 and cell c must equal 1. If this the case, cell a does not equal 0. By the clue set of line C, cells a and b must both equal 0. Because $a = 1 \neq 0$, Nonogram Z is unsolvable. Thus, SAT K is unsatisfiable.

If the number of variables in a SAT problem do not lend themselves to forming a Nonogram of the desired size, one may insert cells already set to value 0 or 1, but the clues must be adjusted accordingly. This is analogous to inserting additional variables into a SAT problem to map it to 3-SAT or k-SAT.

Thus, any SAT problem may be mapped to one or more Nonogram puzzles. See Footnote 2 for an explanation of how the included code converts SAT puzzles to Nonograms.

Appendix

Footnote 1

Nonogram puzzles “nonogram_2x2_6.dat” and “nonogram_2x2_9.dat” have identical sets of clues, but their keys differ. To demonstrate, follow the instructions in the README file to solve these files. If you have not downloaded the data files, they can be generated by “nonogram.py” by following the instructions in the README file for generating all puzzles of size 2x2 or by generating the 2x2 puzzles with keys 6 and 9.

Footnote 2

Figure 7 explains the process “sat_to_nonogram.py” performs to convert a SAT file into a Nonogram file. This process does not result in simplified SAT problems. It typically outputs several SAT problems per Nonogram.

For further information on the SAT to Nonogram or Nonogram to SAT conversion processes, refer to the included code.

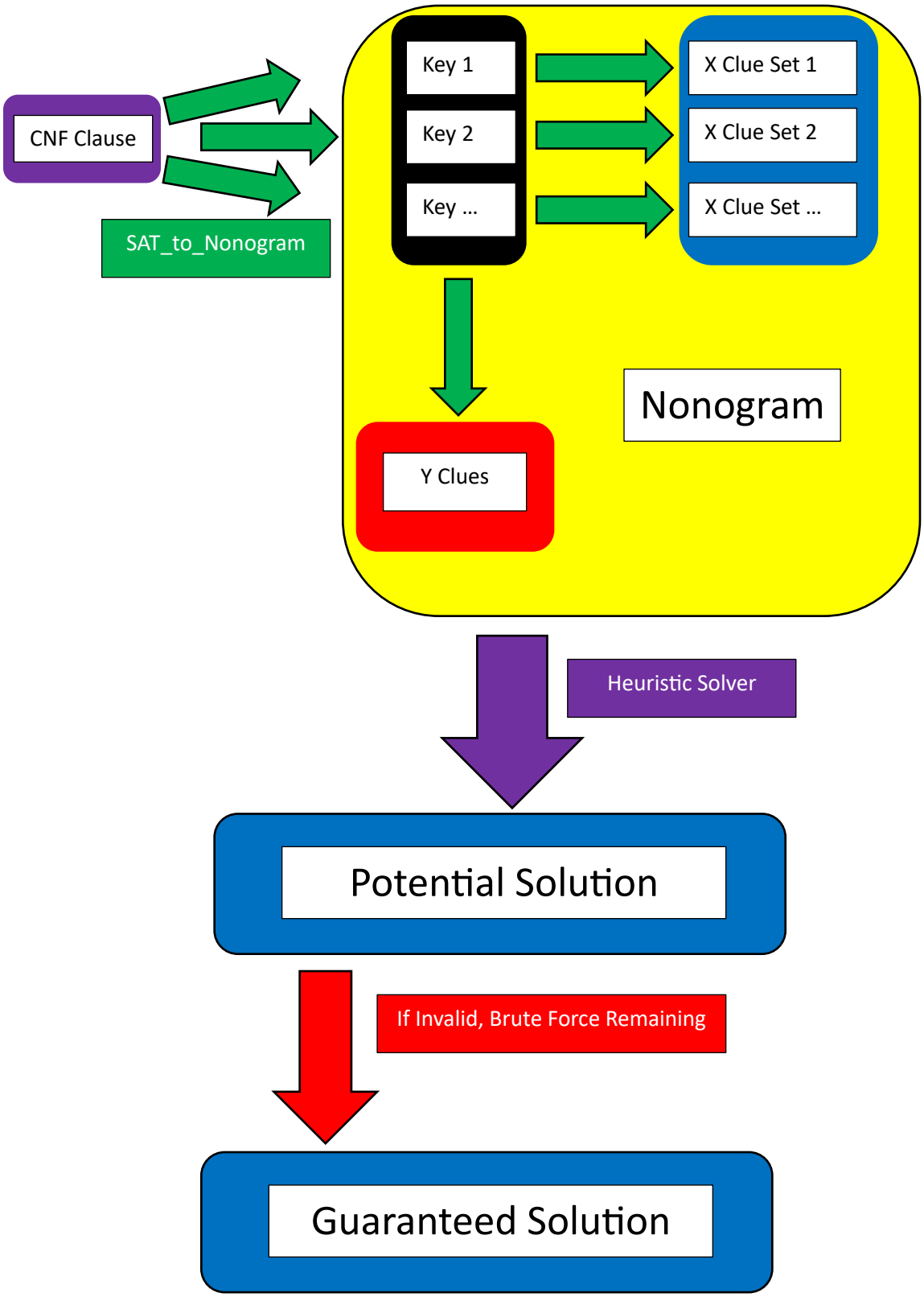


Figure 7 – the SAT to Nonogram conversion process