Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

Dissertação de Mestrado

Mestrado em Engenharia Informática

Solving Colored Nonograms

Luís Pedro Canas Ferreira Mingote
(aluno nº 29634)

2º Semestre de 2008/09
29 de Julho de 2009

Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

Dissertação de Mestrado

Solving Colored Nonograms

Luís Pedro Canas Ferreira Mingote
(aluno nº 29634)

Orientador: Prof. Doutor Francisco de Azevedo

Trabalho apresentado no âmbito do Mestrado em Engenharia Informática, como requisito parcial para obtenção do grau de Mestre em Engenharia Informática.

2º Semestre de 2008/09
29 de Julho de 2009

Acknowledgements

In advance, I would like to thank my wife Marta and my daughters Margarida and Matile for all their support and patience during the time I spent with this work.

I also would like to thank Prof. Francisco de Azevedo for his time and orientation. I cannot thank him enough for all the patience in reading, understanding and proposing improvements to my writings.

I also would like to thank Prof. Paula Amaral, from the Mathematics Department, for making available CPLEX for testing.

I would also like to thank all my family for their support.

Resumo

Nesta dissertação aprofundamos o estudo da resolução de nonogramas coloridos utilizando Programação Linear Inteira (PLI). As formas conhecidas de resolução deste tipo de problemas são a força-bruta, o método iterativo e PLI.

A nossa aproximação generaliza a utilizada por Robert A. Bosch desenvolvida para, apenas, nonogramas a preto e branco, tornando assim disponível uma solução nova e universal para a resolução de nonogramas utilizando PLI.

Sendo as implementações do método iterativo as que apresentam melhores resultados ao nível do desempenho, desenvolvemos também um método híbrido que combina esta aproximação e PLI.

Estes puzzles têm, muitas vezes, várias soluções. A forma de as encontrar pelo modo iterativo é uma pesquisa em árvore com retrocesso. De forma a encontrar as restantes soluções na nossa aproximação aplicamos um algoritmo que utiliza um corte binário para excluir soluções já conhecidas.

Para efeito de testes comparativos entre as diversas aproximações ao problema, desenvolvemos um gerador de nonogramas que permite definir a resolução do puzzle, o seu número de cores e a densidade (número de células pintadas vs. resolução).

Finalmente comparamos o desempenho da nossa aproximação para resolver nonogramas coloridos com o da aproximação interativa.

Palavras-chave: Nonograma, pintar-por-números, PLI, Programação Linear Inteira

Abstract

In this thesis we deepen the study of colored nonogram solving using Integer Linear Programming (ILP). The known methods for solving this kind of problems are the depth-first search (brute-force) one, the iterative one and the ILP one.

Our approach generalizes the one used by Robert A. Bosch which was developed for black and white nonograms only, thus providing a new universal solution for solving nonograms using ILP.

Since the iterative implementations are the ones that present better performance results, we also developed a hybrid method that combines this approach and the ILP one.

This puzzles often have more than one solution. The way to find them using the iterative method e to make a tree search with backtracking. In order to find the remaining solutions using our approach, is to apply an algorithm that uses a binary cut to exclude already known solutions.

In order to perform comparative tests between approaches, we developed a nonogram generator that allows us to define the resolution of the puzzle, its number of colors and its density (number of painted cell vs. resolution).

Finally we compare the performance of our approach in solving colored nonograms against the iterative one.

Keywords: Nonogram, paint-by-numbers, ILP, Integer Linear Programming

Contents

1 Introduction **1**
1.1 Context 1
1.2 Problem Description 1
1.2.1 Initial problems 1
1.2.2 Nonograms 1
1.2.2.1 Black and White Nonograms 2
1.2.2.2 Colored Nonograms 2
1.3 Scope of work and main contributions 3
1.4 Document structure 4

2 Nonograms **5**
2.1 Black and white Nonograms 5
2.1.1 Simple boxes 6
2.1.2 Punctuating 7
2.1.3 Simple spaces 8
2.1.4 Mercury 9
2.1.5 Forcing 9
2.1.6 Glue 10
2.1.7 Joining and splitting 10
2.2 Colored Nonograms 11
2.2.1 Simple boxes 12
2.2.2 Punctuating 14
2.2.3 Simple spaces 14
2.2.4 Mercury 15
2.2.5 Forcing 16
2.2.6 Glue 16
2.2.7 Joining and splitting 17
2.3 Approaches to solving Nonograms 18
2.3.1 Depth-first search (brute-force) 19
2.3.2 Iterative approach 19
2.3.3 Integer Linear Programming approach 21

3 An ILP model for solving Colored Nonograms **23**
3.1 Model Description 23
3.1.1 Notation 23
3.1.2 Variables 24
3.1.3 Block constraints 25
3.1.4 Double Coverage Constraints 26

3.1.5 Objective Function 28
3.1.6 Pre-conditions 28
3.2 Instantiation to Black and White Nonograms 29
3.3 Finding Multiple Solutions 29
3.4 Hybrid model 30

4 Results 33
4.1 Pure ILP approach 33
4.2 Nonogram Generator 34
4.3 Hybrid ILP approach 36

5 Conclusions and Future Work 39

A Full Results 41

B Nonogram File Formats 53
B.1 Bosch based file format 53
B.2 Olšák file format 58
B.3 Hett based file format 59

List of Figures

1.1 Black and white nonogram example (unsolved: left, solved: right) 2
1.2 Colored Nonogram Example - "Fall" from [2] 3

2.1 Black and white nonogram example 6
2.2 Example for the method Simple boxes in black and white nonograms 7
2.3 Example for the method Punctuating 8
2.4 Example 1 for the method Simple spaces applied to a black and white nonogram 8
2.5 Example 2 for the method Simple spaces applied to a black and white nonogram 8
2.6 Example for the method Mercury 9
2.7 Example for the method Forcing 10
2.8 Example for the method Glue 10
2.9 Example for the method Joining and splitting applied to a black and white nonogram 11
2.10 Solving a black and white Nonogram Example 11
2.11 Solving a black and white Nonogram Example — after last (horizontal) iteration 12
2.12 Example 1 for the method Simple boxes 13
2.13 Example for the method Punctuating 14
2.14 Example 1 for the method Simple spaces 14
2.15 Example 2 for the method Simple spaces 15
2.16 Example for the method Mercury 15
2.17 Example for the method Forcing 16
2.18 Example for the method Glue 16
2.19 Example for the method Joining and splitting 17
2.20 Solving a Colored Nonogram Example — "Fall" from [2] 17
2.21 Solving a Colored Nonogram Example — "Fall" from [2] 18
2.22 Solving a Colored Nonogram Example — "Fall" from [2] — After final (horizontal) iteration 19
2.23 Depth-first search — all possibilities for a line 20

4.1 Generated nonogram 35

List of Tables

2.1	Experimental Results (in seconds)	22
4.1	Experimental Results (in seconds)	33
4.2	Results of adding equation (4.1) to ILP (in seconds)	34
4.3	Number of solved puzzles by method and dimension	37
4.4	Number of solved puzzles by method and density	37
4.5	Average time to solve a puzzle by dimension	38
4.6	Average time to solve a puzzle by density	38
A.1	Full Results (in seconds)	42

1. Introduction

1.1 Context

The work hereby presented was developed during the Master Program in Computer Science Engineering (Bologna second cycle), under the original theme "Solving Problems from CSPLib".

1.2 Problem Description

1.2.1 Initial problems

Initially, the purpose of this work was to solve problems from CSPLib (www.csplib.org), a known library of problems for modeling and solving. Given the lack of knowledge about the majority of the existing problems and our interest in exploring and solving new problems, thus broadening our knowledge base, we decided to analyze the following five:

- prob012: Nonograms
- prob020: Darts tournament
- prob022: Bus driver scheduling
- prob032: Maximum density still life
- prob037: Peg solitaire

Although some work was done on problem "prob032 - Maximum density still life", specifically the implementation of the Bucket Elimination algorithm by [11], we decided to deepen the study about problem "prob012 - Nonograms" since it appeared to us that there were approaches that had not been explorer, specially to what concerns colored nonograms.

1.2.2 Nonograms

Nonograms are a popular kind of puzzle whose name varies from country to country, including Paint by Numbers and Griddlers. The goal is to fill cells of a grid in a way that contiguous blocks of the same color satisfy the clues, or restrictions, of each line or column.

According to Wikipedia [21], these kind of puzzles were created in 1987 by Non Ishida, a Japanese graphics editor, and Tetsuya Nishio, a professional Japanese puzzler, at the same time and with no relation whatsoever. Soon after, nonograms started appearing in Japanese puzzle magazines and later as electronic games. Today, magazines with nonogram puzzles are published in several countries and are available as electronic games in a variety of platforms.

Ueda e Nagao prove in [19] that this problem is NP-Complete.

1.2.2.1 Black and White Nonograms

In black and white nonograms the clues indicate the sequence of contiguous blocks of cells to be filled (e.g. the clue 3,1,2 indicates that there is a block of 3 contiguous cells, followed by a sequence of one or more empty cells, then a block of one cell filled, followed by another sequence of one or more empty cells, finally followed by a sequence of two filled cells in that row or column). Figure 1.1 shows an example of a black and white nonogram (unsolved, to the left, solved, to the right).

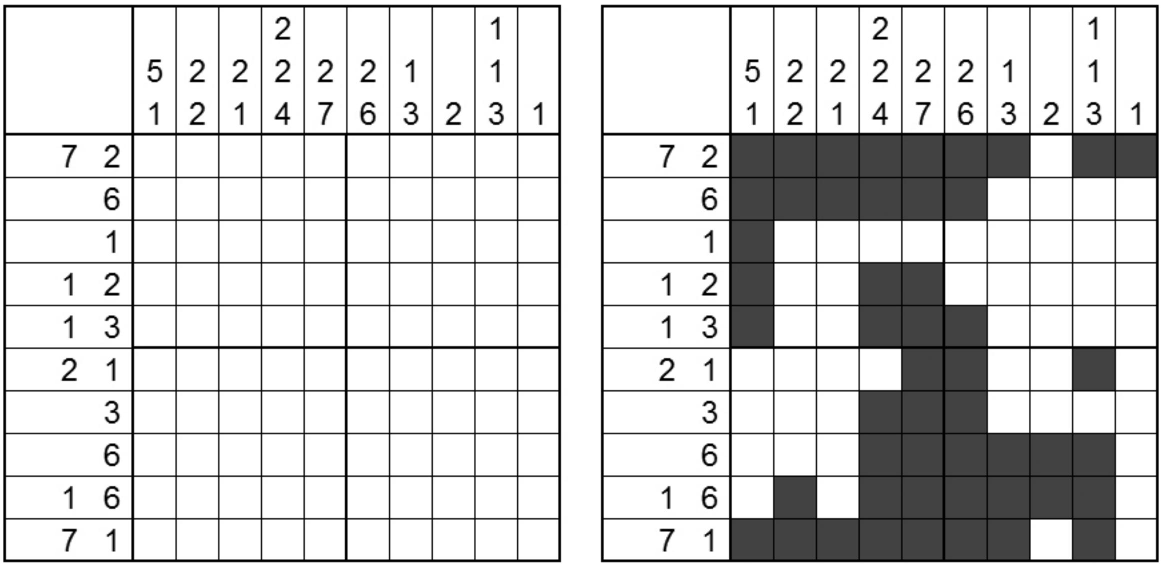


Figure 1.1 Black and white nonogram example (unsolved: left, solved: right)

Known approaches to solving black and white nonograms are the depth-first search (brute-force) one, the iterative one, the ILP one by Bosch [7] and a genetic algorithm by Wouter Wiggers [20].

1.2.2.2 Colored Nonograms

In colored nonograms the clues are composed of pairs that indicate the size and color of each sequence of blocks to be filled. For example, the clue <(3,Red), (1,Green), (2,Blue)> indicates that there is a block of 3 contiguous cells of red, followed by a block of one green cell separated or not by a sequence of empty cells, followed by a sequence of two blue cells separated or not from the green block by a sequence of empty cells, in that row or column. The general rule for separating blocks is that if a block is of the same color of the previous one in the respective sequence then they must be separated by at least an empty cell. Otherwise (i.e., the two blocks have different colors), they may have no cells in between, i.e., they may be adjoining blocks. Note that in the particular case of black and white nonograms this means that blocks in a sequence must always be separated by at least one empty cell. Figure 1.2 represents an example of a colored nonogram with 10 lines by 8 columns with 3 colors.

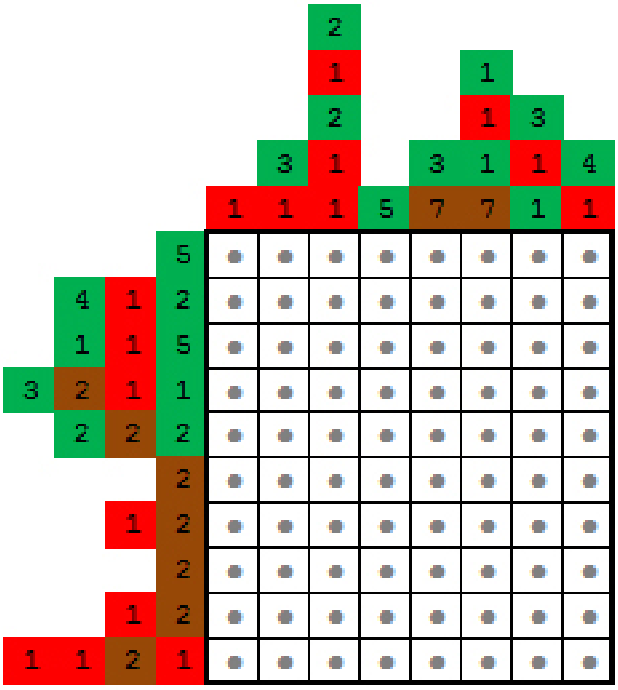


Figure 1.2 Colored Nonogram Example - "Fall" from [2]

1.3 Scope of work and main contributions

Within the scope of this work, colored nonograms were studied in order to develop an Integer Linear Programming model for solving them. This model is based on the one provided by Bosch in [7] for black and white nonograms and generalizes it so it can solve both black and white and colored puzzles. Bosch's file format was also adapted to colored nonograms and in our implementation also supports lines with no clues.

Since the iterative approach is the one that presents the best results, according to Jan Wolter in [23] and according to the tests we performed (shown ahead), we decided to build an hybrid model that integrates both approaches.

In order to compare results of both our models and the iterative one, we built a nonogram generator that can generate puzzles given a resolution (width × height), a number of colors and a density (global or by color).

This allowed us to broaden the sample set used in comparing the different approaches to solving nonograms thus providing a deeper comparison between them with tests over several instances of different dimensions and difficulty.

Our ILP approach was also enhanced in order to obtain more solutions to the same problem, if applicable. This enhancement was accomplished by simplifying a known algorithm that finds multiple solutions in order to make it more efficient to this specific problem.

In summary, the main contributions of this work are:

- An ILP model for solving colored nonograms;
- A nonogram instance generator;
- An hybrid implementation between the iterative approach and our ILP approach for colored nonogram solving.
- A more systematic study of the different nonogram solving approaches
- An adaptation of an algorithm that obtains multiple solutions to an ILP problem, with a simplification that makes it more efficient to specific problems

1.4 Document structure

This document is organized in the following way: In Chapter 2 nonograms (both black and white and colored) are described in full and the best known approaches are detailed.

In Chapter 3 the ILP model we developed to solve colored nonograms is described, including a demonstration that this model corresponds to the one by Bosch in [7] for black and white nonograms. It is also shown how to apply a simple technique in order to find additional solutions in case the first solution obtained for a puzzle is not unique. A description of the hybrid approach between the iterative approach and the hereby presented ILP model is also described.

In Chapter 4 results from the presented solutions are compared to its iterative counterpart. A description of the nonogram generator is also presented.

In Chapter 5 the results of the previous chapter are analyzed and the conclusions of this work are presented. We also suggest some future work based on the one presented here.

In appendix A a table with the result of all tests is show.

in appendix B an example of each file format used is shown.

2 . Nonograms

In the previous chapter a brief description of nonograms was presented. In this one a more detailed explanation about nonograms is shown.

Nonograms are a popular kind of puzzle whose name varies from country to country, including Paint by Numbers and Griddlers. The goal is to fill cells of a grid in a way that contiguous blocks of the same color satisfy the clues, or restrictions, of each line or column.

According to Wikipedia [21], these kind of puzzles were created in 1987 by Non Ishida, a Japanese graphics editor, and Tetsuya Nishio, a professional Japanese puzzler, at the same time and with no relation whatsoever. Soon after, nonograms started appearing in Japanese puzzle magazines and later as electronic games. Today, magazines with nonogram puzzles are published in several countries and are available as electronic games in a variety of platforms.

The most common nonograms are black and white, but they exist also in colors. In fact, black and white nonograms are a specialization of colored nonograms, i.e., are two colored nonograms.

Also there is a different kind of nonogram — called *triddlers* — in which cells are triangles. In this kind of puzzles we have three sets of clues instead of only two. These puzzles can also exist in multiple colors.

Ueda e Nagao prove in [19] that the nonogram problem is NP-Complete.

2.1 Black and white Nonograms

In black and white nonograms the clues indicate the sequence of contiguous blocks of cells to be filled (e.g. the clue 3,1,2 indicates that there is a block of 3 contiguous cells, followed by a sequence of one or more empty cells, then a block of one cell filled, followed by another sequence of one or more empty cells, finally followed by a sequence of two filled cells in that row or column). Figure 1.1 shows an example of a black and white nonogram (unsolved, to the left, solved, to the right).

According to Wikipedia [21], in order to solve this kind of puzzle it is necessary to determine which cells will be filled (black) and which will be empty (white). Determining which cells will be empty is as important as determining which will be filled because the former will help delimiting the solutions for the blocks of each line or column[1].

Simpler puzzles, like the one shown in figure 2.10, can usually be solved by applying the following methods to each line at a time.

[1]For the sake of simplicity, from this point forward, only lines will be mentioned, since the reasoning is the same for columns.

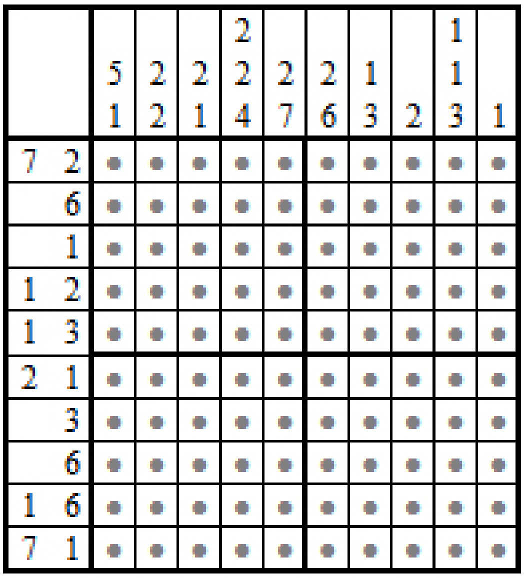


Figure 2.1 Black and white nonogram example

2.1.1 Simple boxes

At the beginning of the solution, when there are no filled cells, for each block $b_i \in \{b_1, ..., b_B\}$ in each row, the space available $S(b_i)$ for it is determined, assuming that the remaining blocks are moved closer to the extremities of the grid as possible (previous blocks to the left and subsequent block to the right). b_i represents a set of filled cells in sequence (vector). The value for $S(b_i)$ can be calculated using equation 2.5, where L represents the size of the line, B represents the number of blocks on the line and $T(b_i)$ represents the size of b_i.

$$S(b_i) = L - B + 1 - \sum_{k \neq i}^{B} T(b_k) \tag{2.1}$$

It is also possible to know for each block what is the potential first cell that it can occupy through equation 2.7, where $b_i[1]$ is block's b_i first cell position in the grid.

$$b_i[1] = \begin{cases} 1 & ;\text{ i = 1} \\ b_{i-1}[1] + T(b_{i-1}) + 1) & ;\text{ i > 1} \end{cases} \tag{2.2}$$

Within this set of cells it is possible to determine which subset is actually filled by analyzing the extremities of the solution, i.e., sliding the block as far to the left as possible and then as far to the right as possible and checking which cells are common to both solutions. In this way, equation 2.3 gives the size of this sub-block, where $T(s_i)$ is the size of the sub-block s_i that can be determined for block b_i.

$$T(s_i) = 2T(b_i) - S(b_i) \tag{2.3}$$

In the same way, it is possible to obtain the first cell (consequently the remaining) of this sub-block through equation 2.4, where $s_i[1]$ is the position of the first cell of sub-block s_i.

$$s_i[1] = b_i[1] + S(b_i) - T(b_i) \quad ; \quad T(s_i) > 0 \tag{2.4}$$

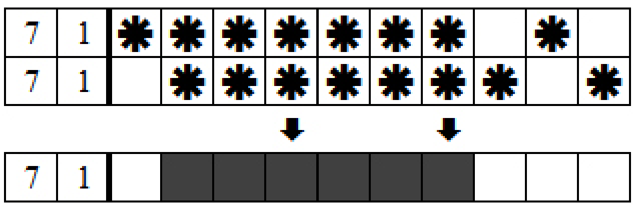


Figure 2.2 Example for the method Simple boxes in black and white nonograms

As an example, for the 10th line of the puzzle shown in figure 2.1, $L = 10$, $B = 2$, $T(b_1) = 7$ and $T(b_2) = 1$. Therefore the space available for the first block is $S(b_1) = 10 - 2 + 1 - 1 = 8$ and $S(b_2) = 10 - 2 + 1 - 7 = 2$. The leftmost indexes each can occupy are $b_1[1] = 1$ and $b_2[1] = 1 + 7 + 1 = 9$.

As for the sub-blocks of cells that can be filled at this point, $T(s_1) = 2 \times 7 - 8 = 6$ and $T(s_2) = 2 \times 1 - 2 = 0$, i.e., it is not possible to fill, for now, any cell in respect to the second block, but it is possible to fill six cells with respect to the first one. It is yet to determine the starting cell of the first and second sub-blocks: $s_1[1] = 1 + 8 - 7 = 2$, i.e., it is possible to fill, at this point, cells 2 through 7 of that line.

Figure 2.2, from line 10 of the puzzle shown in figure 2.1, exemplifies this method for a size 10 line with with two blocks of sizes 7 and 1.

2.1.2 Punctuating

In order to solve the puzzle it is also very important to enclose with empty cells the extremities of each completed block, immediately, as described in the method *Simple spaces*. Precise punctuating usually leads to more *Forcing* and can be vital to finishing the puzzle.

Figure 2.3 exemplifies this method for line 9 of puzzle shown in figure 2.1.

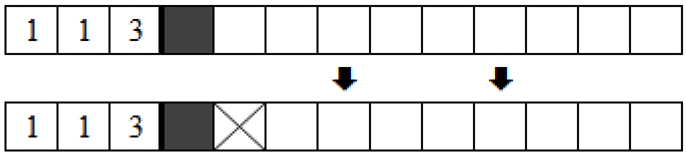


Figure 2.3 Example for the method Punctuating

2.1.3 Simple spaces

The purpose of this method is to find cells that can not be filled by any block due to the constraints imposed by filled cells. For example, a block that is already complete may have at least an empty cell to its left and at least another one to its right, unless it is adjacent to the beginning or the end of the line.

Figure 2.4 from column 8 of the puzzle shown in figure 2.1 shows an example of this method.

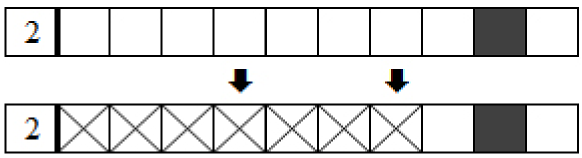


Figure 2.4 Example 1 for the method Simple spaces applied to a black and white nonogram

In figure 2.5, based on one from Wikipedia, a more illustrative example of this method is shown.

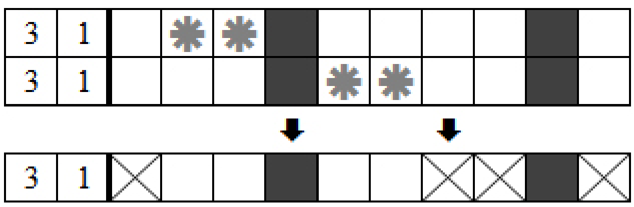


Figure 2.5 Example 2 for the method Simple spaces applied to a black and white nonogram

First, clue 1 is complete which means that there will be an empty cell to its left and another to its right (*Punctuating*). Then, from clue 3 it is possible to conclude that its block can only expand between the second and the sixth cell because it has to include the fourth cell. This means that cells 1 and 7 will be empty.

2.1.4 Mercury

Mercury is a special case of *Simple spaces*. The name comes from the way mercury pulls back from the sides of a container.

If there is a filled cell on a line that is at the same distance from the border as the size of the first block, then the first cell has to be empty. This is true because the first block would not fit to the left of the filled cell. It will have to spread through that cell leaving the first cell behind. Besides, when the cell is in reality a set with cells more to the right, there will be more spaces at the beginning of the line, determined by applying this method several times.

In figure 2.6, from Wikipedia, an example of this method is shown.

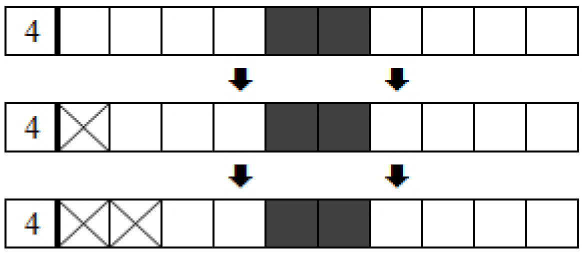


Figure 2.6 Example for the method Mercury

2.1.5 Forcing

In this method the importance of empty cells is demonstrated. En empty cell in the middle of an incomplete line can force a block to complete itself to one of the sides of the empty cell.

In figure 2.7, from line 8 of the puzzle shown in figure 2.1, an example of this method is shown.

The first block (3) will have to be to the left of the first cell already marked as empty. The empty one between the two cells already marked as empty cannot belong to any block from that line which means it has to be empty. Finally, the second block will have to occupy a subset of the last three cells of the line. Applying method *Simple boxes* to both blocks turns out to fill cells 2, 3 and 9.

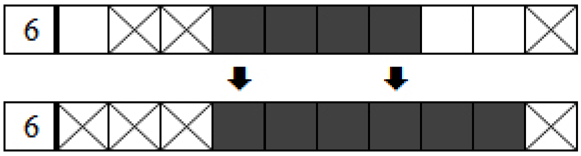


Figure 2.7 Example for the method Forcing

2.1.6 Glue

In this method a full cell at the beginning (or the end) of the possible space for a block forces the completion of that block to the empty side. In the same way, an empty cell in the middle of the possible space for a block can condition the placement of that block's cells.

In figure 2.8, from column 1 of the puzzle shown in figure 2.1, an example of this method is shown.

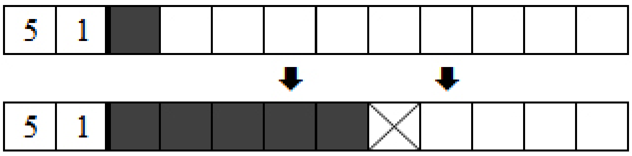


Figure 2.8 Example for the method Glue

In this case, the filled cell in position 1 indicates that the size 5 block has to fill cells 1 through 5. Since the block becomes complete, we mark cell 6 of that column as empty through method punctuating.

2.1.7 Joining and splitting

Filled cells nearby one another can be united or separated according with the number and size of that line's blocks. In this case the whole line has to be analyzed together with the information available for every block.

In figure 2.9, from Wikipedia, an example of this method is shown.

The clue of 5 will join the first two blocks into one large block because a space would produce a block of only 4 cells. Cell 7 will have to be empty, otherwise a 3 size block would form which is not indicated for that line. In this case, the size 2 blocks will also complete, however that is a result of applying the Glue method described earlier.

Using these methods we can easily solve these more simple puzzles. Figures 2.10(a) through

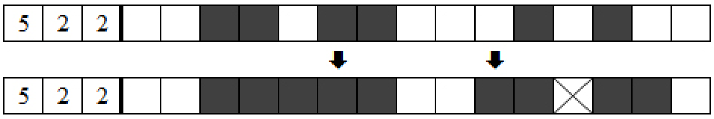
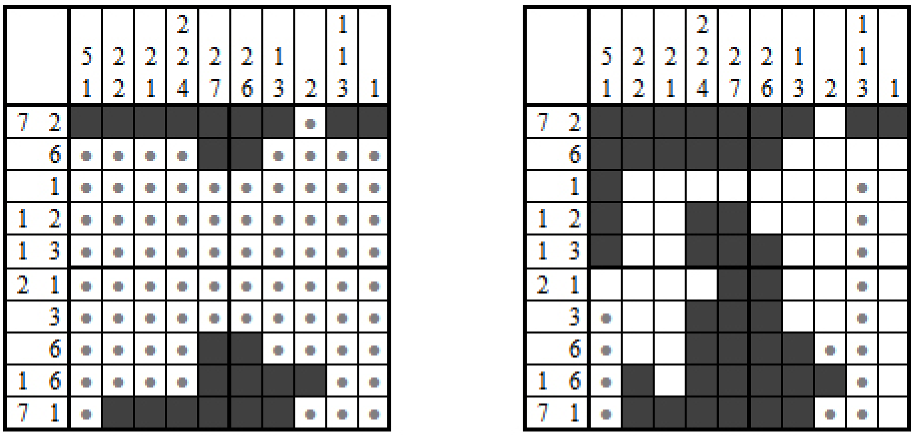


Figure 2.9 Example for the method Joining and splitting applied to a black and white nonogram

2.11 show the two horizontal iterations and the vertical one made in order to solve the puzzle shown in figure 2.1.



(a) After first horizontal iteration

(b) After first vertical iteration

Figure 2.10 Solving a black and white Nonogram Example

2.2 Colored Nonograms

In colored nonograms the clues are composed of pairs that indicate the size and color of each sequence of blocks to be filled. For example, the clue <(3,Red), (1,Green), (2,Blue)> indicates that there is a block of 3 contiguous cells of red, followed by a block of one green cell separated or not by a sequence of empty cells, followed by a sequence of two blue cells separated or not from the green block by a sequence of empty cells, in that row or column. The general rule for separating blocks is that if a block is of the same color of the previous one in the respective sequence then they must be separated by at least an empty cell. Otherwise (i.e., the two blocks have different colors), they may have no cells in between, i.e., they may be adjoining blocks. Note that in the particular case of black and white nonograms this means that blocks in a sequence must always be separated by at least one empty cell. Figure 1.2 represents an

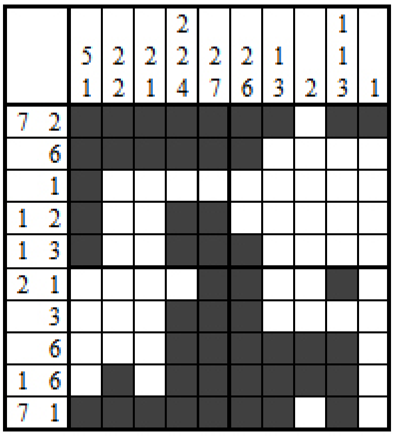


Figure 2.11 Solving a black and white Nonogram Example — after last (horizontal) iteration

example of a colored nonogram with 10 lines by 8 columns with 3 colors.

In the same way as black and white nonograms, in order to solve this kind of puzzle it is necessary to determine which cells will be filled (colored) and which will be empty (white). Determining which cells will be empty is as important as determining which will be filled because the former will help delimiting the solutions for the blocks of each line or column.

As referred earlier, black and white nonograms are a special case of colored nonograms (are two colored nonograms). In that way, the same methods, with some nuances, can be applied to colored nonograms, each line at a time, in order to solve them.

These methods are explained again, but now applied to colored nonograms.

2.2.1 Simple boxes

At the beginning of the solution, when there are no filled cells, for each block $b_i \in \{b_1, ..., b_B\}$ in each row, the space available $S(b_i)$ for it is determined, assuming that the remaining blocks are moved closer to the extremities of the grid as possible (previous blocks to the left and subsequent block to the right). b_i represents a set of filled cells in sequence (vector). The value for $S(b_i)$ can be calculated using equation 2.5, where L represents the size of the line, P represents the number of pairs of contiguous blocks of the same color on the line and $T(b_i)$ represents the size of b_i.

$$S(b_i) = L - P - \sum_{k \neq i}^{B} T(b_k) \tag{2.5}$$

For black and white nonograms equation 2.5 becomes equation 2.1 where B represents the number of block on the line.

It is also possible to know for each block what is the potential first cell that it can occupy through equation 2.7, where $b_i[1]$ is block's b_i first cell position in the grid and f is a function that returns 1 if the blocks are of the same color and 0 otherwise (see equation 2.6 where C_{b_i} is the color of block i).

$$f(b_i, b_j) = \begin{cases} 0 & ; C_{b_i} \neq C_{b_j} \\ 1 & ; C_{b_i} = C_{b_j} \end{cases} \tag{2.6}$$

$$b_i[1] = \begin{cases} 1 & ; \text{i} = 1 \\ b_{i-1}[1] + T(b_{i-1}) + f(b_i, b_{i-1}) & ; \text{i} > 1 \end{cases} \tag{2.7}$$

For black and white nonograms f always returns 1 and equation 2.7 becomes equation 2.2.

Within this set of cells it is possible to determine which subset is actually filled by analyzing the extremities of the solution, i.e., sliding the block as far to the left as possible and then as far to the right as possible and checking which cells are common to both solutions. In this way, equation 2.8 gives the size of this sub-block, where $T(s_i)$ is the size of the sub-block s_i that can be determined for block b_i.

$$T(s_i) = 2T(b_i) - S(b_i) \tag{2.8}$$

In the same way, it is possible to obtain the first cell (consequently the remaining) of this sub-block through equation 2.9, where $s_i[1]$ is the position of the first cell of sub-block s_i.

$$s_i[1] = b_i[1] + S(b_i) - T(b_i) \quad ; \quad T(s_i) > 0 \tag{2.9}$$

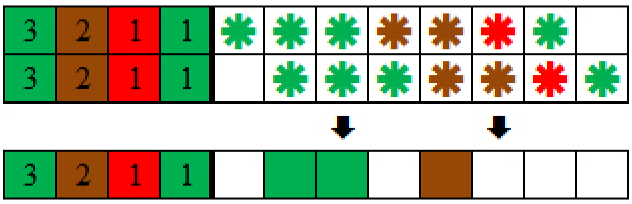


Figure 2.12 Example 1 for the method Simple boxes

As an example, for the fourth line of the puzzle shown in figure 1.2, $L = 8$, $P = 0$, $B = 4$, $T(b_1) = 3$, $T(b_2) = 2$, $T(b_3) = 1$ and $T(b_4) = 1$. Therefore the space available for the first block is $S(b_1) = 8-0-4 = 4$, $S(b_2) = 8-0-5 = 3$, $S(b_3) = 8-0-6 = 2$ and $S(b_4) = 8-0-6 = 2$. The leftmost indexes each can occupy are $b_1[1] = 1$, $b_2[1] = 1+3+0 = 4$, $b_3[1] = 4+2+0 = 6$ and $b_4[1] = 6+1+0 = 7$.

As for the sub-blocks of cells that can be filled at this point, $T(s_1) = 2\times3-4 = 2$, $T(s_2) = 2\times2-3 = 1$, $T(s_3) = 2\times1-2 = 0$ and $T(s_4) = 2\times1-2 = 0$, i.e., it is not possible to fill, for now, any cell in respect to the third and fourth blocks, but it is possible to fill two cells with respect to the first one and one cell with respect to the second one. It is yet to determine the starting cell of the first and second sub-blocks: $s_1[1] = 1+4-3 = 2$ and $s_2[1] = 4+3-2 = 5$., i.e., it is possible to fill, at this point, cells 2, 3 and 5 of that line.

2.2.2 Punctuating

In order to solve the puzzle it is also very important to enclose with empty cells the extremities of each completed block that is the same color as the adjacent one, immediately, as described in the method *Simple spaces*. Precise punctuating usually leads to more *Forcing* and can be vital to finishing the puzzle.

Figure 2.13 exemplifies this method for line line 6 of puzzle shown in figure 1.2.

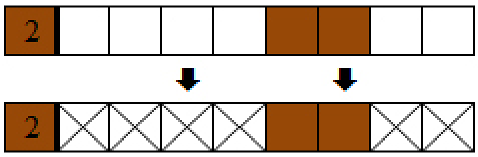


Figure 2.13 Example for the method Punctuating

2.2.3 Simple spaces

The purpose of this method is to find cells that can not be filled by any block due to the constraints imposed by filled cells. For example, a block that is already complete may have at least an empty cell to its left and at least another one to its right, unless it is adjacent to the beginning or the end of the line.

Figure 2.14 from line 2 of the puzzle shown in figure 1.2 shows an example of this method.

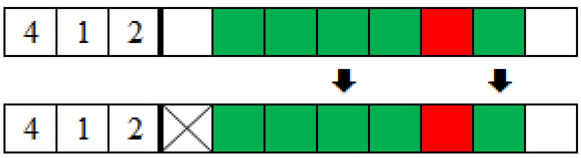


Figure 2.14 Example 1 for the method Simple spaces

In figure 2.15, based on one from Wikipedia, a more illustrative example of this method is shown.

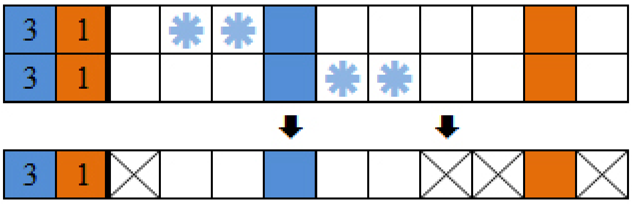


Figure 2.15 Example 2 for the method Simple spaces

First, clue 1 is complete which means that there will be an empty cell to its left and another to its right (*Punctuating*). Then, from clue 3 it is possible to conclude that its block can only expand between the second and the sixth cell because it has to include the fourth cell. This means that cells 1 and 7 will be empty.

2.2.4 Mercury

Mercury is a special case of *Simple spaces*. The name comes from the way mercury pulls back from the sides of a container.

If there is a filled cell on a line that is at the same distance from the border as the size of the first block, then the first cell has to be empty. This is true because the first block would not fit to the left of the filled cell. It will have to spread through that cell leaving the first cell behind. Besides, when the cell is in reality a set with cells more to the right, there will be more spaces at the beginning of the line, determined by applying this method several times.

In figure 2.16, from line 1 of the puzzle shown in figure 1.2, an example of this method is shown.

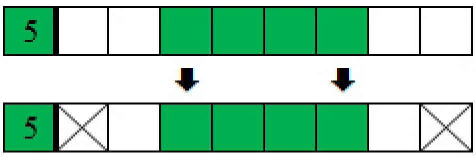


Figure 2.16 Example for the method Mercury

2.2.5 Forcing

In this method the importance of empty cells is demonstrated. En empty cell in the middle of an incomplete line can force a block to complete itself to one of the sides of the empty cell.

In figure 2.17, base on the one from Wikipedia, an example of this method is shown.

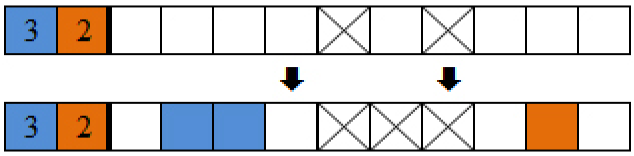


Figure 2.17 Example for the method Forcing

The first block (3) will have to be to the left of the first cell already marked as empty. The empty one between the two cells already marked as empty cannot belong to any block from that line which means it has to be empty. Finally, the second block will have to occupy a subset of the last three cells of the line. Applying method *Simple boxes* to both blocks turns out to fill cells 2, 3 and 9.

2.2.6 Glue

In this method a full cell at the beginning (or the end) of the possible space for a block forces the completion of that block to the empty side. In the same way, an empty cell in the middle of the possible space for a block can condition the placement of that block's cells.

In figure 2.18, from column 5 of the puzzle shown in figure 1.2, an example of this method is shown.

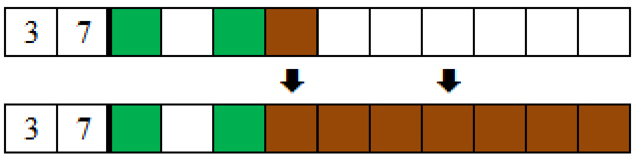


Figure 2.18 Example for the method Glue

In this case, filled brown cell in position 4 preceded by filled green cell in position 3 indicates that the size 7 brown block has to fill cells 5 through 10.

2.2.7 Joining and splitting

Filled cells nearby one another can be united or separated according with the number and size of that line's blocks. In this case the whole line has to be analyzed together with the information available for every block.

In figure 2.19, from column 2 of the puzzle shown in figure 1.2, an example of this method is shown.

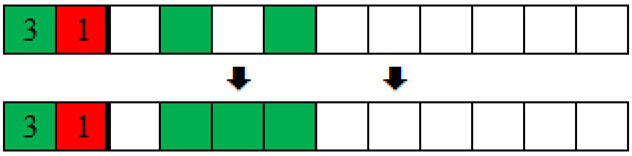
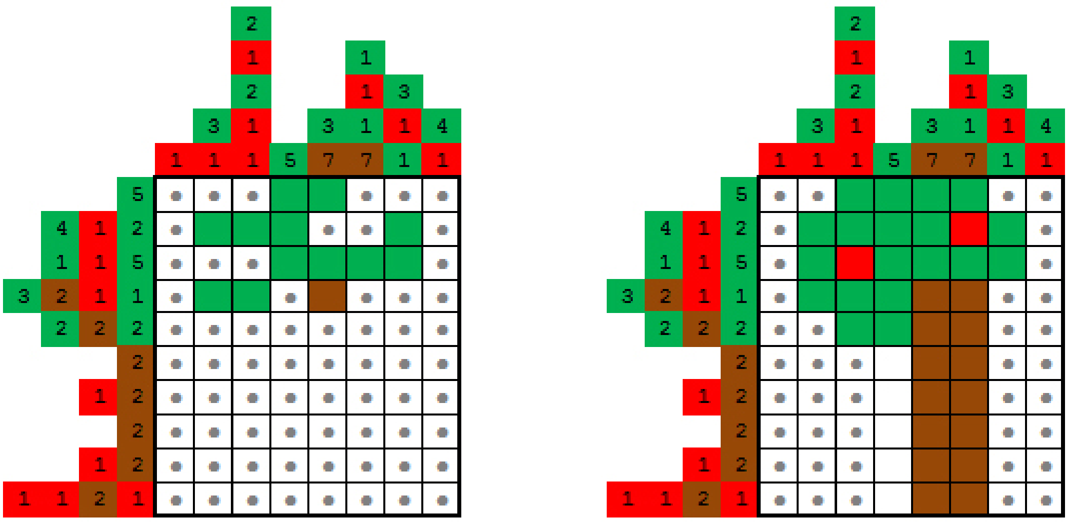


Figure 2.19 Example for the method Joining and splitting

The clue to the size 3 green block will make that the two green cells unite because a space in cell 2 would divide the first block in two.

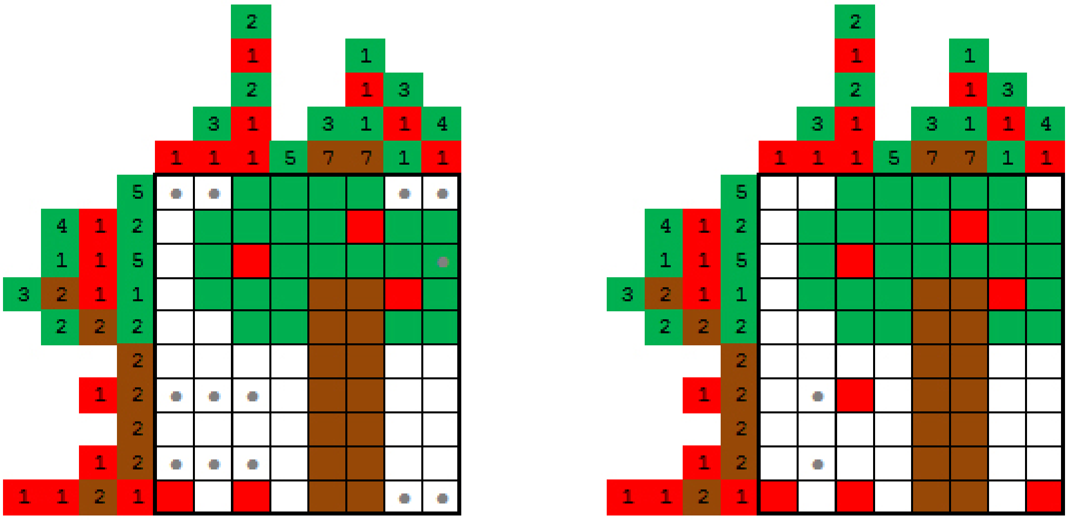
Using these methods one can easily solve these more simple puzzles. Figures 2.20(a) through 2.22 show the three horizontal iterations and the two vertical ones made in order to solve the puzzle shown in figure 1.2.



(a) After first horizontal iteration

(b) After first vertical iteration

Figure 2.20 Solving a Colored Nonogram Example — "Fall" from [2]



(a) After second horizontal iteration

(b) After second vertical iteration

Figure 2.21 Solving a Colored Nonogram Example — "Fall" from [2]

2.3 Approaches to solving Nonograms

In the previous section we showed how simpler puzzles can be solved by looking at each line at a time and applying one or more methods to color cells or mark them as spaces. For more complex puzzles we can reach a state where we can not fill more unknown cells by applying those methods. At that point we have to try and guess a value (color or space) for a cell and then reapply the aforementioned methods to try to reach a solution or a contradiction. Eventually we will reach another state where another guess must be made to continue to try to solve the puzzle, and so on. If a contradiction is reached, then the value we chose for a determined cell is wrong. In black and white puzzles this means that the cell will have the opposite value (empty if the chosen value was filled, filled otherwise), but in colored nonograms another color can be chosen for that cell. These more complex puzzles are usually difficult to solve by a human.

This is where computer based approaches can be useful.

Known approaches for solving nonograms are the depth-first search (brute-force), the iterative approach and the ILP approach. A comparison between a genetic algorithm and the depth-first search algorithm, by Wouter Wiggers [20], was also found. As mentioned in the article, the genetic algorithm not always reaches a solution, however it reaches a near solution very quickly.

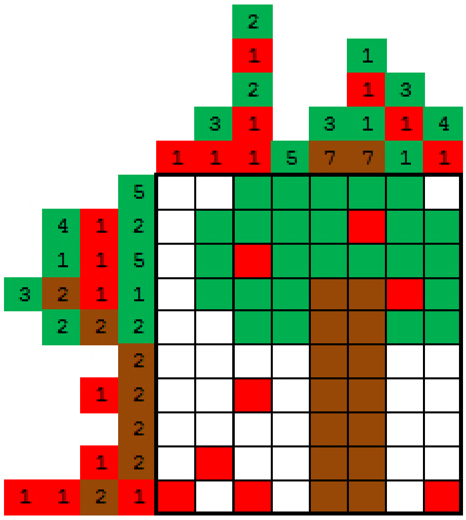


Figure 2.22 Solving a Colored Nonogram Example — "Fall" from [2] — After final (horizontal) iteration

2.3.1 Depth-first search (brute-force)

This approach tries all possible combinations for the set of blocks of each line. For example, for a size 10 line, belonging to a black and white nonogram, with two blocks of sizes 5 and 1, we would have 10 possibilities only for that line, as shown in figure 2.23.

An optimization of this algorithm is to begin with the lines that have fewer possibilities. However, if we want to find all solutions then all possibilities must be explored.

The following are implementations of this approach:

- ECLIPSE program by Joachim Schimpf [14]
- P-99: Ninety-Nine Prolog Problems [10]
- Colin Barker's Home Page - LPA Win-Prolog Goodies [6]

These implementations only work for black and white nonograms.

2.3.2 Iterative approach

The iterative technique consists in determining, for every line, cyclically, which cells can be considered filled and which cells can be considered empty, in accordance to the information available at the moment, until a solution is reached or no more cells can be determined.

To find this information an algorithm is applied to each line at a time. This algorithm is called a *line-solver*. A line-solver is an algorithm that given a single line (row or column), and the state of that line so far, tries to figure out what additional cells can be marked.

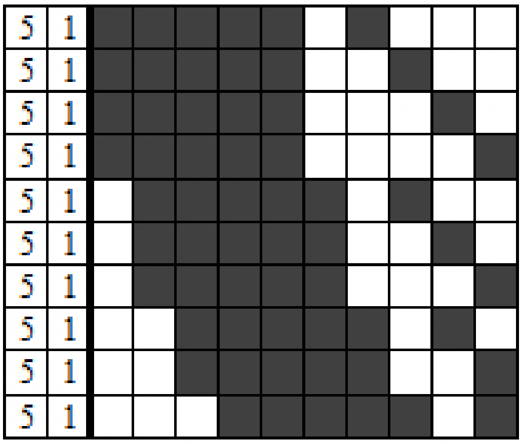


Figure 2.23 Depth-first search — all possibilities for a line

When the successive application of the *line-solver* stops contributing to the puzzle's resolution, the search for contradictions can help.

This method includes:

1. Forcing an unknown cell to be empty or full;

2. Reapply the methods mentioned in order to find a solution;

3. If a contradiction is found then the value chosen for that cell was not the correct on and another cell must be tried, or another value must be tried for that cell (chronologic backtracking).

The problem to this method is the choice of a cell to try a contradiction, i.e., having an heuristic to find the best cells to try a value. Besides, while trying a cell for a contradiction another situation may arise in that another try to find a contradiction must take place, and so forth.

Usually, the best cells to initiate a contradiction try are the following:

- Cells that have many filled neighbors;

- Cells near the border or nearby sets of empty cells;

- Cells that are between lines that consist of more empty cells.

Steven Simpson, in his site [16], describes his algorithm for the resolution of nonograms. As mentioned above, the algorithm tries to solve, or partially solve, a line for each iteration. The order in which lines are tried to be solved is defined by the value of equation 2.10, Where

B is the number of blocks of that line, L is the size of the line and $T(b_1)$ to $T(b_B)$ are the sizes of each block. When non-negative, the result is the number of filled cells that can be determined from an empty line. A negative value indicates a shortfall of pre-determined cells. Note that when $B = 1$ and $T(b_i) = L$ then $I = L$ and this is the maximum value.

$$I = (B+1)\sum_{i=1}^{B} T(b_i) + B(B-L-1) \tag{2.10}$$

Exceptionally, if $B = 0$ (empty line) then $I = L$.

After a line is chosen a *line-solver*, or a sequence of line-solvers, are applied to it in order to fill as many cells as possible. The line-solvers are applied to the line in a predefined rank order, i.e, higher ranked line-solvers are only applied after lower ranked ones don't reveal more cells. There are four well-known line-solvers: *fast*, *complete*, *olsak* [13] and *fcomp*. The first gets most of the available information available; the second gets everything logically deductible, but is very inefficient; the third is a variation of the first one, but is a little more exhaustive and gets all the information; the fourth is a revised version of the second one, but is significantly more efficient.

In [23], Jan Wolter compares several nonogram solvers in which the best three (Wolter's *pbnsolve* [22], Simpson's *nonogram* [15] and Olšák's *grid* [13]) use one or more of these line-solvers. Simpson' is the only that does not solve colored nonograms.

2.3.3 Integer Linear Programming approach

Robert A. Bosch [7] presented in 2001 a solution based on Integer Linear Programming as well as the code that converts the definition of a puzzle in a program that can be used with CPLEX [3] to solve the puzzle.

The mentioned program only works for puzzles that have clues for all the lines.

Since this approach only solves black and white nonograms we proposed to develop an ILP model that solves colored nonograms.

The performance results of our approach compared to an adaptation for colored nonograms of the depth-first search provided by Hett [10], an adapted version of the optimized depth-first search approach also by Hett and Olšák's *grid* are shown in table 2.1.

The times were measured on a 2.4 GHz Intel© Centrino© vPro™with 2 GB of RAM running Microsoft© Windows©. The Prolog program was run in ECLiPSe [1] and the generated ILP problems were run on SCIP [4]. Results are shown in table 4.1, where NPC stands for "Number of Painted Cells".

Given the good performance of the iterative approaches we also proposed to develop a hybrid model between this approach and the ILP one.

The ILP approach presented here starts from scratch with an empty grid and, in general, could not improve the Iterative method for the available tests, although already presented similar results using a non commercial tool.

Table 2.1 Experimental Results (in seconds)

Puzzle	R×C×Col	NPC	Brute-force (Prolog)	Brute-force opt (Prolog)	Iterative	ILP
Fall	10x8x3	47	1,050.70	0.03	0.07	0.03
Fish	16x16x2	164	(too long)	0.08	0.07	0.21
AtoZ	16x16x2	50	(too long)	0.92	0.10	23.04
Time	35x30x5	520	(too long)	(out of memory)	0.21	3.51

Our initial idea when developing the ILP model, in addition to the new theoretical results, was to use it together with the Iterative method, which we knew was efficient to quickly fill many cells of the grid using simple inferences on the rows and columns clues. That is precisely what we proposed to develop, by applying the ILP model only after the Iterative technique already filled many cells, thus reducing a lot the model complexity by converting many variables to constants.

Both Simpson and Wolter have references to other nonogram solvers in [17] and [23], respectively.

3. An ILP model for solving Colored Nonograms

In the previous chapter we verified that no ILP model for solving colored nonograms exists. In this chapter we describe the model we developed for this purpose.

3.1 Model Description

As in [7], our approach is to think of a colored nonogram as a problem comprised of two interlocking tiling problems: one involving the placement of the row blocks, and the other involving the placement of the column blocks. If a cell is painted (it can be assumed that unpainted cells are painted white) then it must be covered by both a row block and a column block; if it is painted white (not painted) then it must be left uncovered by the row blocks and the column blocks.

3.1.1 Notation

The notation used here is similar to the one used by Bosch in [7], as follows.

m = the number of rows,

n = the number of columns,

o = the number of colors excluding white (We use a sequence of natural numbers to identify colors, starting at 1 $(1, 2, \dots, o)$.),

b^r_i = the number of blocks in row i, $1 \leq i \leq m$,

b^c_j = the number of blocks in column j, $1 \leq j \leq n$,

$s^r_{i,1}, s^r_{i,2}, \dots, s^r_{i,b^r_i}$ = the block-size sequence for row i,

$s^c_{j,1}, s^c_{j,2}, \dots, s^c_{j,b^c_j}$ = the block-size sequence for column j,

$c^r_{i,1}, c^r_{i,2}, \dots, c^r_{i,b^r_i}$ = the block-color sequence for row i,

$c^c_{j,1}, c^c_{j,2}, \dots, c^c_{j,b^c_j}$ = the block-color sequence for column j.

In addition, let

$e^r_{i,t}$ = the smallest value of j such that row i's t^{th} block can be placed in row i with its leftmost pixel occupying cell j,

$l^r_{i,t}$ = the largest value of j such that row i's t^{th} block can be placed in row i with its leftmost pixel occupying cell j,

$e^c_{j,t}$ = the smallest value of i such that column j's t^{th} block can be placed in column j with its topmost pixel occupying cell i,

$l^c_{i,t}$ = the largest value of i such that column j's t^{th} block can be placed in column j with its topmost pixel occupying cell i.

These are constants valid for the empty puzzle. (The letters "e" and "l" stand for "earliest" and "latest"). In our example puzzle, the second row's first block must be placed so that its leftmost pixel occupies cell 1 or 2, the second block must be placed so that its leftmost pixel occupies cell 5 or 6, and the third block must be placed so that its leftmost pixel occupies cell 6 or 7. In other words

$$e^r_{2,1} = 1, l^r_{2,1} = 2, e^r_{2,2} = 5, l^r_{2,2} = 6, e^r_{2,3} = 6 \quad \text{and} \quad l^r_{2,3} = 7.$$

These values are obtained by iteratively placing the blocks in their leftmost or topmost possible cells and then placing them in their rightmost or bottommost possible cells. In our example, the first block's first cell is 1 and, since the first block's size is 4 and the color of both blocks is different, the second block's first possible cell is 5. Then, since the color of the third block is also different from the second one and the size of the second block is 1, the third block's first possible cell is 6. Now, the third block is pushed to its rightmost cell (7) and one finds out that the second block's last possible cell is 6 and the first block's last possible cell is 2.

Note that the rules for determining these values are the same for colored or black and white nonograms. Of course, in black and white puzzles all the blocks are of the same color, which means they have to be separated by at least one empty cell.

3.1.2 Variables

As in the approach by Bosch in [7], in our approach there are three sets of variables. One set specifies the color of each cell:

$$\forall_{1 \le i \le m, 1 \le j \le n} \quad z_{i,j} = \begin{cases} 1 \le c \le o & \text{; if row } i\text{'s } j^{th} \text{ cell is painted with color } c \\ 0 & \text{; if row } i\text{'s } j^{th} \text{ cell is not painted} \end{cases} \tag{3.1}$$

The other two sets of variables are concerned with placements of the row and column blocks.

$$\forall_{1 \le i \le m, 1 \le t \le b^r_i, e^r_{i,t} \le j \le l^r_{i,t}} \quad y_{i,t,j} = \begin{cases} 1 & \text{; if row } i\text{'s } t^{th} \text{ block is placed in row } i \text{ with its leftmost pixel occupying cell } j \\ 0 & \text{; if not} \end{cases} \tag{3.2}$$

$$\forall_{1\le j\le n, 1\le t\le b^c_j, e^c_{j,t}\le i\le l^c_{j,t}} \quad x_{j,t,i} = \begin{cases} 1 & \text{; if column } j\text{'s } t^{th} \text{ block is placed in column } j \text{ with its topmost pixel occupying cell } i \\ 0 & \text{; if not} \end{cases} \tag{3.3}$$

3.1.3 Block constraints

To ensure that row i's t^{th} block appears in row i exactly once, the following imposes

$$\forall_{1\le i\le m, 1\le t\le b^r_i} \quad \sum_{j=e^r_{i,t}}^{l^r_{i,t}} y_{i,t,j} = 1 \tag{3.4}$$

For line 2 of our example we have

$$y_{2,1,1} + y_{2,1,2} = 1,$$

$$y_{2,2,5} + y_{2,2,6} = 1,$$

$$y_{2,3,6} + y_{2,3,7} = 1.$$

For the next two constraints the auxiliary function (3.5) is defined. This function, which was already defined as equation 2.6 in chapter 1, returns the value 1 if the two arguments are the same, and 0 otherwise, which will be useful to compare colors of two contiguous blocks.

$$eq(c_1, c_2) = \begin{cases} 1 & \text{; if } c_1 = c_2 \\ 0 & \text{; otherwise} \end{cases} \tag{3.5}$$

To ensure that row i's $(t+1)^{th}$ block is placed to the right of its t^{th} block, the following imposes

$$\forall_{e^r_{i,t}+1\le j\le l^r_{i,t}} \quad y_{i,t,j} \le \sum_{j'=j+s^r_{i,t}+eq(c^r_{i,t}, c^r_{i,t+1})}^{l^r_{i,t+1}} y_{i,t+1,j'} \tag{3.6}$$

In line 2 of our example we have

$$y_{2,1,2} \le y_{2,2,6},$$

$$y_{2,2,6} \le y_{2,3,7}.$$

To ensure that column j's t^{th} block appears in column j exactly once, the following imposes

$$\forall_{1\le j\le n, 1\le t\le b^c_j} \quad \sum_{i=e^c_{j,t}}^{l^c_{j,t}} x_{j,t,i} = 1 \tag{3.7}$$

To ensure that column j's $(t+1)^{th}$ block is placed under its t^{th} block, the following imposes

$$\forall_{e^c_{j,t}+1\le i\le l^c_{j,t}} \quad x_{j,t,i} \le \sum_{i'=i+s^c_{j,t}+eq(c^c_{j,t},c^c_{j,t+1})}^{l^c_{j,t+1}} x_{j,t+1,i'} \tag{3.8}$$

3.1.4 Double Coverage Constraints

To guarantee that each painted cell is covered by both a row block and a column block, the following pair of inequalities imposes:

$$\forall_{1\le i\le m, 1\le j\le n} \quad z_{i,j} \le \sum_{t=1}^{b^r_i} \sum_{j'=max\{e^r_{i,t}, j-s^r_{i,t}+1\}}^{min\{l^r_{i,t}, j\}} y_{i,t,j'} \times c^r_{i,t} \tag{3.9}$$

$$\forall_{1\le i\le m, 1\le j\le n} \quad z_{i,j} \le \sum_{t=1}^{b^c_j} \sum_{i'=max\{e^c_{j,t}, i-s^c_{j,t}+1\}}^{min\{l^c_{j,t}, i\}} x_{j,t,i'} \times c^c_{j,t} \tag{3.10}$$

Without these restrictions the model would allow having cells painted by row blocks, but not painted by any column block, or vice versa. The first inequality (3.9) states that if row i's j^{th} cell is painted, then at least one of row i's blocks must be placed in such a way that it covers row i's j^{th} cell. (The upper and lower limits of the second summation make sure that j' satisfies the two pairs of inequalities $e^r_{i,t} \le j' \le l^r_{i,t}$ and $j - s^r_{i,t} + 1 \le j' \le j$. The first pair holds if, and only if, row i's t^{th} cell is covered when row i's t^{th} block is placed in row i with its leftmost pixel occupying cell j'. The second pair holds if and only if row i's j^{th} pixel is covered when row i's t^{th} block is placed in row i with its leftmost pixel occupying pixel j'). The other inequality (3.10) makes sure that if row i's j^{th} cell is painted, then at least one of column j's blocks covers it. For line 2 of our example we have for cell $z_{2,4}$ that

$$z_{2,5} \le y_{2,1,2} \times c^r_{2,1} + y_{2,2,5} \times c^r_{2,2},$$

$$z_{2,5} \le x_{5,1,1} \times c^c_{5,1} + x_{5,1,2} \times c^c_{5,1}$$

If $z_{2,5}$ is painted, the right hand terms of these inequalities will yield exactly its color value in a solved puzzle. Otherwise (empty cell), the terms hold value 0. Ideally, the model should express this disjunction directly, allowing only those 2 values. However, in order to allow ILP solving, it is kept as a linear inequality. Nevertheless, below it is proven that this is sufficient for a correct and complete model, in the presence of the other constraints.

Finally, constraints that prevent unpainted cells from being covered by the row blocks or column blocks are included — inequalities (3.11) and (3.12).

$$\forall_{1\leq i\leq m,\ 1\leq j\leq n,\ 1\leq t\leq b^r_i,\ j-s^r_{i,t}+1\leq j'\leq j,\ e^r_{i,t}\leq j'\leq l^r_{i,t}} \quad z_{i,j} \geq y_{i,t,j'} \times c^r_{i,t} \tag{3.11}$$

$$\forall_{1\leq i\leq m,\ 1\leq j\leq n,\ 1\leq t\leq b^c_j,\ e^c_{j,t}\leq i'\leq l^c_{j,t},\ i-s^c_{j,t}+1\leq i'\leq i} \quad z_{i,j} \geq x_{j,t,i'} \times c^c_{j,t} \tag{3.12}$$

In line 2 of our example we have

$$z_{2,5} \geq y_{2,1,2} \times c^r_{2,1}, \quad z_{2,5} \geq y_{2,2,5} \times c^r_{2,2},$$

$$z_{2,5} \geq x_{5,1,1} \times c^c_{5,1}, \quad z_{2,5} \geq x_{5,1,2} \times c^c_{5,1}.$$

One might think that it is necessary to ensure that each painted cell must be covered by one row block and one column block of the same color. However, the remaining constraints ensure that there is only the need to guarantee that a painted cell must be covered by one row block and one column block. In order to prove it, let us explore all the possibilities regarding the coverage of some cell z:

1. No block covers cell z;
2. Only one block covers cell z and it is of the same color;
3. Only one block covers cell z and its color is smaller than the color of z;
4. Only one block covers cell z and its color is greater than the color of z;
5. More than one block covers cell z;

Of these five possibilities, only the first two are possible in real puzzles. The last three are the ones that our model has to avoid.

In sake of simplicity, but with no loss of generality, only inequality (3.9), for lines, of the double coverage constraints will be used in our case analysis for these five possibilities:

Possibility 1: The only way to satisfy this possibility is with an empty cell z, with value 0, which, by inequality (3.9), will guarantee that no block covers it (forcing the respective $y_{i,t,j'}$ variables to be 0), i.e.

$$\sum_{t=1}^{b^r_i} \sum_{j'=min\{e^r_{i,t}, j-s^r_{i,t}+1\}}^{max\{l^r_{i,t}, j\}} y_{i,t,j'} \times c^r_{i,t} = 0.$$

Possibility 2: This possibility fully satisfies inequality (3.9), corresponding to the equality of both terms.

Possibility 3: If a single block of smaller color than the color of cell z covers it then inequality (3.9) is not satisfied, thus disallowing such possibility, as desired.

Possibility 4: In the case that there may be one block that covers cell z, and which color is greater than the color of z, then inequality (3.9) would be satisfied. However, this would violate inequality (3.11) thus turning the solution invalid.

Possibility 5: If more than one block covers cell z, inequality (3.9) could only be satisfied if the sum of the colors of the covering blocks is less than or equal to the color of cell z. But this would violate equation (3.4) thus turning the solution invalid.

3.1.5 Objective Function

Since this is a satisfaction problem there is no need for an objective function, but since ILP solvers need one, the following is included (note that this function is a constant and we already know its value):

$$minimize/maximize \sum_{i=1}^{m} \sum_{j=1}^{n} z_{i,j} \tag{3.13}$$

3.1.6 Pre-conditions

We also include in our approach one pre-condition in order to verify whether the puzzle is trivially impossible to solve, before even trying to search for a solution (another improvement with respect to [7]). This is a necessary, but not sufficient condition that will save the time of trying to solve a puzzle that is impossible, and that also helps determining whether there is any error in the definition of the puzzle. This condition, shown by equation (3.14), checks whether the sum of the sizes of all blocks of each color is the same for both the rows and columns clues.

$$\forall_{c \in \{1..o\}} \quad \sum_{i=1}^{m} \sum_{t=1}^{b_i^r} f(s_{i,t}^r, c_{i,t}^r, c) = \sum_{j=1}^{n} \sum_{t=1}^{b_j^c} f(s_{j,t}^c, c_{j,t}^c, c) \tag{3.14}$$

where $f(s, c_1, c_2) = s$ if $c_1 = c_2$, and 0 otherwise.

3.2 Instantiation to Black and White Nonograms

If o is set to 1 ($o = 1$), thus allowing only black and white in a puzzle, our model becomes the one provided by Bosch in [7], i.e., equation (3.1) becomes

$$z_{i,j} = \begin{cases} 1 & \text{; if row } i\text{'s } j^{th} \text{ cell is painted} \\ 0 & \text{; if row } i\text{'s } j^{th} \text{ cell is not painted} \end{cases} \tag{3.15}$$

Equations (3.2) and (3.3) are kept from the approach provided by Bosch. Equation (3.4) is equal to the one in the approach by Bosch, but inequality (3.6) was extended so block $t+1$ can follow block t immediately, due to possible contiguous blocks of different colors. For black and white puzzles it corresponds exactly to the formulation in [7] since all blocks have the same color which leads the *eq* function to always yield value 1. Inequalities (3.7) and (3.8) are similar, but regard columns. Finally, since the only possible color takes value 1, the double coverage constraints set by inequalities (3.9) and (3.10) become

$$\forall_{1 \le i \le m, 1 \le j \le n} \quad z_{i,j} \le \sum_{t=1}^{b_i^r} \sum_{j'=max\{e_{i,t}^r, j-s_{i,t}^r+1\}}^{min\{l_{i,t}^r, j\}} y_{i,t,j'}, \tag{3.16}$$

$$\forall_{1 \le i \le m, 1 \le j \le n} \quad z_{i,j} \le \sum_{t=1}^{b_j^c} \sum_{i'=max\{e_{j,t}^c, i-s_{j,t}^c+1\}}^{min\{l_{j,t}^c, i\}} x_{j,t,i'}, \tag{3.17}$$

$$\forall_{1 \le i \le m, 1 \le j \le n, 1 \le t \le b_i^r, j-s_{i,t}^r+1 \le j' \le j, e_{i,t}^r \le j' \le l_{i,t}^r} \quad z_{i,j} \ge y_{i,t,j'} \tag{3.18}$$

and

$$\forall_{1 \le i \le m, 1 \le j \le n, 1 \le t \le b_j^c, e_{j,t}^c \le i' \le l_{j,t}^c, i-s_{j,t}^c+1 \le i' \le i} \quad z_{i,j} \ge x_{j,t,i'} \tag{3.19}$$

as in [7] (where the *min* and *max* functions are incorrectly swapped in the summation limits).

3.3 Finding Multiple Solutions

The described ILP model allows finding a single solution to a puzzle, which actually is the best one, although in this case all solutions are alike since the optimizing function is a constant.

Nonograms are satisfaction problems, which in ILP must be modeled as optimization problems. Since it is possible that the obtained solution is not unique, we also try to find additional solutions to a puzzle. For that, the algorithm developed by Jung-Fa Tsai et al. described in [18] was first considered. This algorithm uses an integer cut to exclude the previously found solution, extending the ILP model to a Mixed ILP model (MILP), which is the general approach to finding additional solutions in ILP. But, in fact, a much simpler approach was used by applying a binary cut similar to the one proposed by Balas and Jeroslow in [5].

Since our binary variables (either $y_{i,t,j}$ or $x_{j,t,i}$) are enough to provide the solution (they completely determine the filled puzzle, since clues are constant), a binary cut is enough.

The cut that needed to be applied to exclude an existing solution is shown in (3.20) using the y set of variables (the x set of variables could also be used).

$$\sum_{(i,t,j)\in A} y_{i,t,j} - \sum_{(i,t,j)\in B} y_{i,t,j} \leq |A| - 1, \qquad \begin{array}{l} A = \{(i,t,j) \mid y_{i,t,j} = 1\}, \\ B = \{(i,t,j) \mid y_{i,t,j} = 0\} \end{array} \tag{3.20}$$

Basically, after finding a solution to the problem, the constraints in inequality (3.20) are added to the problem and another try is made to find another solution.

3.4 Hybrid model

The hybrid model we propose here basically consists in substituting the search part of the iterative approach by our ILP model.

At first, the puzzle is logically solved, i.e., one or more *line-solvers* are applied to every line of the puzzle, repeatedly, until there is no more information that can be inferred. Then, if the puzzle is not completely solved, the ILP model is instantiated.

Our implementation generates a CPLEX LP file (.lp) that represents the current state of the puzzle according to the model presented in section 3.1. This approach is more flexible than generating the ILP model specifically for a solver like SCIP [4] or CPLEX [3] because it allows the comparison of results between different ILP solvers.

SCIP is currently one of the fastest non-commercial mixed integer programming (MIP) solver. ILOG CPLEX©is a commercial mathematical programming optimizer that, among other things, solves mixed integer programs. Although SCIP is advertised as the best performing non-commercial MIP solver, CPLEX — the best MIP solver — is five times faster.

The process of instantiating our ILP model, and subsequently generating the LP file, envolves the following steps:

- Compute earliest and latest constants
- Write objective function to file
- Write block constraints to file
- Write double coverage constraints to file
- Write bounds to file: here is where the partial solution found by the iterative approach is inserted in the ILP model
- Write all the variables to file

Since among the best performing implementations of the iterative approach only *pbnsolve* and Olšák's (*grid*) can solve colored nonograms, we decided to adapt *pbnsolve* into our hybrid approach. The reason we did not choose *grid* was that the program code comments are in Czech. On the other hand, *pbnsolve's* code comments are very complete and understandable. Steven Simpson's *nonogram* [16] can not solve colored nonograms.

Also, for testing purposes, we did not implement Balas and Jeroslow's binary cut in this approach.

4. Results

In the previous chapter our ILP model for solving colored nonograms was described. We also described an hybrid approach to solving colored nonograms between the iterative and the ILP ones.

Here, we present the results of the performance tests we ran in order to compare the different approaches to solving colored nonograms.

First we present the results between our pure ILP approach and the iterative and the depth-first search ones. Then we show the results obtained by comparing our hybrid approach and the iterative one.

4.1 Pure ILP approach

In order to test the performance of the model described in Chapter 3 (without the use of Balas and Jeroslow's algorithm) it was tested against three algorithms: one adaptation (the original program solves only black and white nonograms) of an implementation in Prolog of a brute force search by Werner Hett [10], an optimized variant of this implementation (by altering the ordering of the line tasks) and an implementation in C of the iterative approach by Mirek Olšák and Petr Olšák available in [13].

Four puzzles were used for the purpose of these tests: the "Fall" puzzle from Griddlers.net [2] (10x8x3, i.e. a 10 by 8 grid with 3 colors) used as an example in this dissertation (figure 1.2), the "Fish" and the "AtoZ" puzzles (16x16x2) from Ali Corbin's web page [8], and the "Time" adapted from the copyrighted Sunday Telegraph & Aenigma Design and colored by Brian Grainger (35x30x5) [9].

The times were measured on a 2.4 GHz Intel© Centrino© vPro™with 2 GB of RAM running Microsoft© Windows©. The Prolog program was run in ECLiPSe [1] and the generated ILP problems were run on SCIP [4]. Results are shown in table 4.1, where NPC stands for "Number of Painted Cells".

As shown in table 4.1 the first puzzle was solved almost instantly by both the iterative implementation and the ILP approach. The brute-force implementation took about 17 minutes to return the results. With some optimization applied to the brute-force approach, namely by

Table 4.1 Experimental Results (in seconds)

Puzzle	R×C×Col	NPC	Brute-force (Prolog)	Brute-force opt (Prolog)	Iterative	ILP
Fall	10x8x3	47	1,050.70	0.03	0.07	0.03
Fish	16x16x2	164	(too long)	0.08	0.07	0.21
AtoZ	16x16x2	50	(too long)	0.92	0.10	23.04
Time	35x30x5	520	(too long)	(out of memory)	0.21	3.51

Table 4.2 Results of adding equation (4.1) to ILP (in seconds)

Puzzle	ILP	ILP w/ AC
Fall	0.03	0.03
Fish	0.21	0.11
AtoZ	23.04	33.58
Time	3.51	2.45

re-sorting the line tasks, the puzzle is also solved almost instantly. The "Fish" puzzle is a little harder to solve. The brute-force approach was not able to solve it in a timely fashion although all other approaches solved it pretty quickly. The other 16x16 puzzle — "AtoZ" — is even harder to solve. This was the hardest puzzle to solve by the ILP approach. The fourth (and biggest) puzzle could not be solved by the brute-force algorithms. The iterative approach found all 14 solutions to the puzzle in less than half a second and the ILP approach took about 3.5 seconds to find the fist one.

In order to try to improve the results of the ILP approach we added equation (4.1) to the set of constraints, where the right-hand term is a constant.

$$\sum_{i=1}^{m}\sum_{j=1}^{n} z_{i,j} = \sum_{i=1}^{m}\sum_{t=1}^{b_i^r} s_{i,t}^r \times c_{i,t}^r \tag{4.1}$$

We believed that by adding this constraint, the solver would reach a solution sooner since the objective value for the problem was already defined (is a constant).

The results are shown in table 4.2, where AC means "Additional Constraint".

Only the performance on the hardest puzzle was not improved which turns out to be inconclusive as to the advantages of adding this extra constraint.

4.2 Nonogram Generator

In order to test the different approaches in a proper manner a set of a substantial number of puzzles with varying dimensions, number of colors and densities should be used. To accomplish this we decided to create a nonogram puzzle generator.

By developing this generator we also solved the problem of converting the puzzles to the different file formats supported by each approach.

This generator allows the generation of puzzles based on number of rows, number of columns, number of colors, global density (amount of painted cells vs. available cells — *number of rows* × *number of columns*) and density by color. The puzzles are generated by painting randomly chosen cells with specific or randomly chosen colors and then obtaining the clues from the grid. The generated puzzle can then be saved as a Bosch based file format (adapted for colored nonograms), an Olšák file format or a Hett [10] list based Prolog format (also adapted for colored nonograms).

Examples of the file formats created by our generator for puzzle shown in figure 4.1 are shown in appendix B.

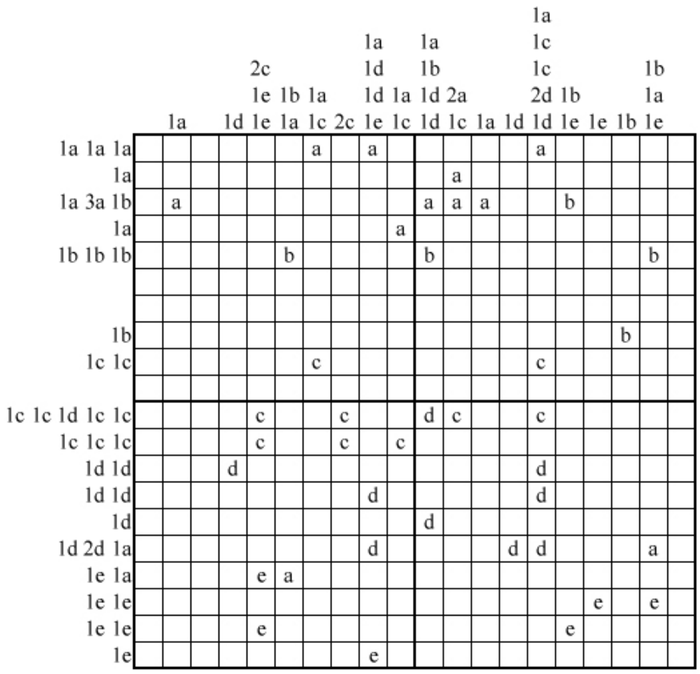


Figure 4.1 Generated nonogram

Bosch's based file format begins with a puzzle definition section where the dimension of the puzzle and the its number of colors are defined (excluding the background color). we also added a title field in order to identify the puzzle more easily:

```
title: TEST_20x20x5_101
number_of_rows: 20
number_of_columns: 20
number_of_colors: 5
```

After the puzzle definition section follows the clues for the rows. Here, the number of clusters is defined for each row and after, the blocks' sizes and colors are defined:

```
row_1:
number_of_clusters: 3
size(s): 1 1 1
color(s): 1 1 1
```

The last section of the file defines the clues for the columns and its format is similar to the previous one.

In Olšák's file format all text before "#" or ":" in the first column is ignored. If the puzzle has colored blocks then we need to write "#D" or "#d" in the first column.

This line denotes the start of the color declaration. The color declaration ends by a ":" in the first column and the block declarations follow at the next line immediately.

Lines of color declarations have the following format:

<spaces><inchar><colon><outchar><spaces><word_XPM><spaces><comment>

The < spaces> denotes zero or more spaces or tabs. The exception: <word_XPM> has to be terminated by one or more spaces and/or tabs.

<inchar> is a character used to identify a color in the block declaration section, after the numbers that represent their sizes. digits, spaces, commas or tab can not be used for <inchar> declaration. The "0" and "1" are exceptions, see bellow.

<outchar> is a character which will be used to represent that color in the terminal printing of the solution.

<word_XPM> is the word (without spaces) used in XPM format for color declaration. we can use the natural word for the color (e.g. blue) or a six hexadecimal digits preceded by a "#" that represents a RGB color (e.g. "#0000FF"). In order to use a natural word for colors they have to be defined in the rgb.txt of the X window system where program runs.

If <inchar> is "0", then this line declares the color for the background of the image. If this declaration is omitted white will be used as the background color.

If <inchar> is "1", then this line declares the "default" color of blocks. This color is used if no <inchar> follows the block declaration. If this line is omitted then the color must be specified for each block declaration.

Each block declaration section (one for row and one for columns) begins after a line with a colon. For every line of the puzzle a sequence of size and color pairs (without spaces separating the size and the color) separated by spaces or tabs.

Hett's based file format is defined as a predicate with three arguments: a title and two lists. Each list defines the list of blocks for rows and columns and is composed of a list of blocks that can be empty. Each block is another list with two elements: a size and a color.

4.3 Hybrid ILP approach

For the purpose of this work 270 problems were generated divided in three large subsets of $20\times20\times5$ (number of rows by number of columns with number of colors), $40\times60\times5$ and $100\times100\times5$. Each of these subsets contains 90 problems divided by density (10 of each density — 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80% and 90%).

This hybrid ILP solution was tested against *pbnsolve* — the implementation in C of the iterative approach by Jan Wolter available in [22]. Due to the poor results shown by the Prolog implementations by Werner Hett [10] they were removed from this test.

We imposed a time limit of 15 minutes for solving each of the puzzles in both approaches.

The results were not the ones we expected. The iterative approach is still the fastest to solve colored nonograms and was the one that solved more nonograms within the 15 minutes timeframe we imposed. Also, in terms of memory consumption, the iterative approach is better. Although the save and load times of the .lp files generated from our sample set of nonograms were not taken into account, some were over 100 MB in size. This means that if these times were added the results would be worse. Of course, if these times were taken into account we would be penalizing the ILP model with hard disk access (much slower than memory access). A solution to this problem would be to completely integrate the hybrid approach, i.e., without generating any files.

In table 4.3 the number of puzzles solved by each approach and by dimension is shown. Note that if the puzzle is logically solvable it does not count to either the ILP or the iterative approaches.

Table 4.3 Number of solved puzzles by method and dimension

	100x100	40x60	20x20	Total
Logically solvable	0	4	22	26
Iterative with search	50	61	68	179
ILP	42	5	68	115
Total	92	70	158	320

In table 4.4 the number of puzzles solved by each approach and by puzzle density is shown. Again, if the puzzle is logically solvable it does not count to any other approach.

Table 4.4 Number of solved puzzles by method and density

	10%	20%	30%	40%	50%	60%	70%	80%	90%	Total
Logically solvable						1	5	7	13	26
Iterative with search	18	12	10	15	30	29	25	23	17	179
ILP	10	10	10	10	17	19	15	14	10	115
Total	28	22	20	25	47	49	45	44	40	320

Table 4.5 presents the average time each approach took to solve the different puzzle by size. The values include the logical solving.

Table 4.6 presents the average time each approach took to solve the puzzles by density. This values include logical solving.

It seems clear that lower density puzzles are harder to solve, specially if their size is large. In fact, when there are few cells to fill in the grid it becomes harder to logically solve the puzzle. This means that the puzzle is solved largely by search with backtracking, in case of the iterative approach, or by applying our ILP model. In either case the process is computationally heavy.

Table 4.5 Average time to solve a puzzle by dimension

	Average of ILP Total Time	Average of Iterative Total Time
100x100	39,031728	0,866221
20x20	2,600312	0,004404
40x60	6,886713	0,527231
Total	13,725823	0,380377

Table 4.6 Average time to solve a puzzle by density

	Average of ILP Time	Average of Iterative Time
10%	0,323448	0,768844
20%	14,997813	0,790693
30%	6,991948	0,011389
40%	0,454907	0,683674
50%	61,614073	1,389269
60%	12,522192	0,056861
70%	9,507734	0,016201
80%	6,576823	0,009129
90%	3,543865	0,004569
Total	13,725823	0,380377

Full results of these tests can be found in table A.1 in appendix A. The times are presented in seconds and were obtained on a 1.8 GHz Intel© Pentium© M with 1 GB of RAM. The generated ILP problems were run on SCIP [4].

We could also test our hybrid approach with CPLEX on a 3.0 GHz Intel© Core™Duo machine with 2 GB of RAM and although we could not analyze them in detail, the results were significantly better than results we obtained with SCIP. Some puzzles that could not be solved by SCIP within the 15 minutes window were solved by CPLEX and some puzzles were solved seven times (and more) faster than with SCIP.

5 . Conclusions and Future Work

In this dissertation we presented a new ILP approach to model the Colored Nonograms problem, which generalizes a known approach which was limited to black and white Nonograms. We demonstrated its correctness and, additionally, we also showed how to efficiently find possible additional solutions by a simple adaptation of a known technique using a binary cut, by taking advantage of the specificities of this problem. This work developed during this Master led to the publication of the article [12].

We also enhanced the aforementioned model by merging it with an iterative approach thus providing an hybrid approach to colored nonograms.

In order to provide a significant sample set of puzzles, for test and comparisons, we also developed a nonogram generator. This generator allows us to create puzzles given their width, height, color count and density (either global or by color) and then to save them in three formats: Bosch's variant for colored nonograms, Olšák's format and a Hett [10] list based Prolog format variant for colored nonograms.

The hybrid model results were not the ones we expected. The iterative approach is still the fastest to solve colored nonograms and was the one that solved more nonograms within the 15 minutes timeframe we imposed. Also, in terms of memory consumption, the iterative approach is better. Some of the .lp files generated from our largest sample nonograms were over 100 MB in size which is a consequence of the great amount of variables and constraints that consume a lot of memory.

This also means that maybe there is room for improvement. First by fully integrating the model in one tool and then by trying to fine tune the model. One example of this can be to change the objective function and include the objective function value as a constraint and use CPLEX to verify the results. Specially for the more complex puzzles that were not solved by either approach, within the 15 minutes window we defined.

Another way the model can be improved is by trying to implement a backtracking mechanism with ILP, i.e, instead of trying to find a final solution with the ILP model, we try to find a partial solution and then reapply the iterative approach the partial solution.

The model can also be improved in order to solve other problems, like triddlers.

The nonogram puzzle generator developed can also be improved by allowing to take into account the number of blocks of a puzzle. With the current approach, the puzzles generated have often a large number of small size blocks.

A . Full Results

Table A.1: Full Results (in seconds)

Title	W	H	C	Dens.	H Blks	V Blks	Cells logically solved	Cells	ILP state	Logic time	ILP Total Time	Iterative Total Time
TEST_100x100x5_101	100	100	5	10%	909	914	1137	10000	ILP unsolved	0,071626	(timeout)	(timeout)
TEST_100x100x5_102	100	100	5	10%	908	920	1016	10000	ILP unsolved	0,070137	(timeout)	(timeout)
TEST_100x100x5_103	100	100	5	10%	904	920	1209	10000	ILP unsolved	0,063196	(timeout)	(timeout)
TEST_100x100x5_104	100	100	5	10%	915	904	1330	10000	ILP unsolved	0,035417	(timeout)	(timeout)
TEST_100x100x5_105	100	100	5	10%	892	899	1512	10000	ILP unsolved	0,064483	(timeout)	(timeout)
TEST_100x100x5_106	100	100	5	10%	929	906	1190	10000	ILP unsolved	0,071708	(timeout)	(timeout)
TEST_100x100x5_107	100	100	5	10%	913	899	1527	10000	ILP unsolved	0,034598	(timeout)	(timeout)
TEST_100x100x5_108	100	100	5	10%	918	899	1227	10000	ILP unsolved	0,065956	(timeout)	(timeout)
TEST_100x100x5_109	100	100	5	10%	899	918	1041	10000	ILP unsolved	0,070309	(timeout)	(timeout)
TEST_100x100x5_110	100	100	5	10%	921	905	1075	10000	ILP unsolved	0,067073	(timeout)	(timeout)
TEST_100x100x5_201	100	100	5	20%	1670	1674	257	10000	ILP unsolved	0,084544	(timeout)	(timeout)
TEST_100x100x5_202	100	100	5	20%	1648	1685	155	10000	ILP unsolved	0,082706	(timeout)	(timeout)
TEST_100x100x5_203	100	100	5	20%	1670	1691	209	10000	ILP unsolved	0,081917	(timeout)	(timeout)
TEST_100x100x5_204	100	100	5	20%	1674	1642	172	10000	ILP unsolved	0,083765	(timeout)	(timeout)
TEST_100x100x5_205	100	100	5	20%	1666	1680	221	10000	ILP unsolved	0,078765	(timeout)	(timeout)
TEST_100x100x5_206	100	100	5	20%	1704	1686	145	10000	ILP unsolved	0,091690	(timeout)	(timeout)
TEST_100x100x5_207	100	100	5	20%	1673	1694	126	10000	ILP unsolved	0,084892	(timeout)	(timeout)
TEST_100x100x5_208	100	100	5	20%	1678	1707	226	10000	ILP unsolved	0,081119	(timeout)	(timeout)
TEST_100x100x5_209	100	100	5	20%	1684	1681	279	10000	ILP unsolved	0,079904	(timeout)	(timeout)
TEST_100x100x5_210	100	100	5	20%	1649	1681	219	10000	ILP unsolved	0,085691	(timeout)	(timeout)
TEST_100x100x5_301	100	100	5	30%	2278	2359	191	10000	ILP unsolved	0,085262	(timeout)	(timeout)
TEST_100x100x5_302	100	100	5	30%	2318	2350	200	10000	ILP unsolved	0,118743	(timeout)	(timeout)
TEST_100x100x5_303	100	100	5	30%	2306	2333	194	10000	ILP unsolved	0,084814	(timeout)	(timeout)
TEST_100x100x5_304	100	100	5	30%	2344	2329	209	10000	ILP unsolved	0,086428	(timeout)	(timeout)
TEST_100x100x5_305	100	100	5	30%	2298	2319	197	10000	ILP unsolved	0,088181	(timeout)	(timeout)

TEST_100x100x5_306	100	100	5	30%	2291	2324	181	10000	ILP	unsolved	0,089121	(timeout)	(timeout)
TEST_100x100x5_307	100	100	5	30%	2268	2288	208	10000	ILP	unsolved	0,086357	(timeout)	(timeout)
TEST_100x100x5_308	100	100	5	30%	2279	2372	249	10000	ILP	unsolved	0,089197	(timeout)	(timeout)
TEST_100x100x5_309	100	100	5	30%	2317	2342	144	10000	ILP	unsolved	0,125440	(timeout)	(timeout)
TEST_100x100x5_310	100	100	5	30%	2292	2341	165	10000	ILP	unsolved	0,087015	(timeout)	(timeout)
TEST_100x100x5_401	100	100	5	40%	2828	2876	532	10000	ILP	unsolved	0,083406	(timeout)	(timeout)
TEST_100x100x5_402	100	100	5	40%	2841	2945	687	10000	ILP	unsolved	0,130944	(timeout)	(timeout)
TEST_100x100x5_403	100	100	5	40%	2854	2876	508	10000	ILP	unsolved	0,089428	(timeout)	(timeout)
TEST_100x100x5_404	100	100	5	40%	2882	2890	593	10000	ILP	unsolved	0,087703	(timeout)	(timeout)
TEST_100x100x5_405	100	100	5	40%	2861	2932	760	10000	ILP	unsolved	0,090675	(timeout)	(timeout)
TEST_100x100x5_406	100	100	5	40%	2878	2907	751	10000	ILP	unsolved	0,122355	(timeout)	(timeout)
TEST_100x100x5_407	100	100	5	40%	2894	2900	723	10000	ILP	unsolved	0,122503	(timeout)	(timeout)
TEST_100x100x5_408	100	100	5	40%	2849	2900	620	10000	ILP	unsolved	0,157251	(timeout)	(timeout)
TEST_100x100x5_409	100	100	5	40%	2856	2943	441	10000	ILP	unsolved	0,086984	(timeout)	(timeout)
TEST_100x100x5_410	100	100	5	40%	2886	2917	612	10000	ILP	unsolved	0,127249	(timeout)	(timeout)
TEST_100x100x5_501	100	100	5	50%	3433	3435	7543	10000	ILP	unsolved	0,092937	(timeout)	6,170296
TEST_100x100x5_502	100	100	5	50%	3356	3405	7796	10000	ILP	unsolved	0,077249	(timeout)	4,136845
TEST_100x100x5_503	100	100	5	50%	3402	3475	8165	10000	ILP	unsolved	0,097762	(timeout)	2,387747
TEST_100x100x5_504	100	100	5	50%	3399	3497	8393	10000	ILP	solved	0,097296	466,887296	2,412709
TEST_100x100x5_505	100	100	5	50%	3403	3486	8032	10000	ILP	unsolved	0,083820	(timeout)	3,678880
TEST_100x100x5_506	100	100	5	50%	3340	3450	7219	10000	ILP	unsolved	0,093507	(timeout)	6,750002
TEST_100x100x5_507	100	100	5	50%	3373	3405	8286	10000	ILP	solved	0,082643	515,762643	2,677742
TEST_100x100x5_508	100	100	5	50%	3397	3445	7666	10000	ILP	unsolved	0,104757	(timeout)	2,671261
TEST_100x100x5_509	100	100	5	50%	3468	3491	7910	10000	ILP	unsolved	0,086351	(timeout)	3,011243
TEST_100x100x5_510	100	100	5	50%	3383	3458	7410	10000	ILP	unsolved	0,075568	(timeout)	7,042581
TEST_100x100x5_601	100	100	5	60%	3925	3957	9747	10000	ILP	solved	0,028383	25,268383	0,143986
TEST_100x100x5_602	100	100	5	60%	3972	3989	9784	10000	ILP	solved	0,029014	24,969014	0,126308
TEST_100x100x5_603	100	100	5	60%	3907	3974	9780	10000	ILP	solved	0,031004	24,401004	0,135391
TEST_100x100x5_604	100	100	5	60%	3889	3927	9723	10000	ILP	solved	0,029205	25,119205	0,196939
TEST_100x100x5_605	100	100	5	60%	3918	4006	9881	10000	ILP	solved	0,029003	23,699003	0,075729
TEST_100x100x5_606	100	100	5	60%	3846	3930	9723	10000	ILP	solved	0,028700	24,698700	0,183959

TEST_100x100x5_607	100	100	5	60%	3929	4040	9723	10000	ILP solved	0,029289	25,849289	0,203829
TEST_100x100x5_608	100	100	5	60%	3853	3939	9725	10000	ILP solved	0,027012	25,377012	0,208551
TEST_100x100x5_609	100	100	5	60%	3919	3968	9721	10000	ILP solved	0,028461	24,858461	0,187681
TEST_100x100x5_610	100	100	5	60%	3906	3950	9798	10000	ILP solved	0,027075	24,247075	0,131486
TEST_100x100x5_701	100	100	5	70%	4387	4472	9903	10000	ILP solved	0,018426	18,738426	0,043642
TEST_100x100x5_702	100	100	5	70%	4271	4426	9937	10000	ILP solved	0,017756	18,437756	0,030240
TEST_100x100x5_703	100	100	5	70%	4340	4469	9894	10000	ILP solved	0,019906	18,509906	0,052573
TEST_100x100x5_704	100	100	5	70%	4398	4440	9943	10000	ILP solved	0,018453	18,768453	0,028317
TEST_100x100x5_705	100	100	5	70%	4333	4452	9919	10000	ILP solved	0,019676	19,009676	0,033532
TEST_100x100x5_706	100	100	5	70%	4404	4452	9892	10000	ILP solved	0,019487	19,009487	0,054427
TEST_100x100x5_707	100	100	5	70%	4344	4426	9933	10000	ILP solved	0,018895	19,198895	0,033170
TEST_100x100x5_708	100	100	5	70%	4358	4478	9890	10000	ILP solved	0,019200	19,349200	0,058385
TEST_100x100x5_709	100	100	5	70%	4400	4510	9928	10000	ILP solved	0,018345	18,898345	0,035825
TEST_100x100x5_710	100	100	5	70%	4412	4494	9896	10000	ILP solved	0,019212	19,279212	0,059264
TEST_100x100x5_801	100	100	5	80%	4915	5082	9948	10000	ILP solved	0,014690	13,764690	0,024836
TEST_100x100x5_802	100	100	5	80%	4869	5018	9922	10000	ILP solved	0,015201	13,985201	0,036287
TEST_100x100x5_803	100	100	5	80%	4965	5077	9972	10000	ILP solved	0,014675	13,884675	0,017973
TEST_100x100x5_804	100	100	5	80%	4843	4950	9960	10000	ILP solved	0,014769	13,484769	0,020390
TEST_100x100x5_805	100	100	5	80%	4957	5026	9954	10000	ILP solved	0,014461	13,594461	0,021018
TEST_100x100x5_806	100	100	5	80%	4925	5011	9956	10000	ILP solved	0,014343	13,964343	0,021563
TEST_100x100x5_807	100	100	5	80%	4846	4932	9968	10000	ILP solved	0,014140	13,544140	0,018527
TEST_100x100x5_808	100	100	5	80%	4890	5048	9936	10000	ILP solved	0,014968	14,064968	0,027588
TEST_100x100x5_809	100	100	5	80%	4945	5030	9976	10000	ILP solved	0,014346	13,454346	0,016940
TEST_100x100x5_810	100	100	5	80%	4890	4941	9932	10000	ILP solved	0,014684	13,754684	0,030590
TEST_100x100x5_901	100	100	5	90%	5406	5583	9996	10000	ILP solved	0,009972	8,019972	0,010225
TEST_100x100x5_902	100	100	5	90%	5432	5640	9984	10000	ILP solved	0,009949	8,179949	0,011321
TEST_100x100x5_903	100	100	5	90%	5475	5559	9980	10000	ILP solved	0,010443	8,370443	0,012427
TEST_100x100x5_904	100	100	5	90%	5362	5496	9988	10000	ILP solved	0,009765	8,179765	0,010747
TEST_100x100x5_905	100	100	5	90%	5474	5598	9976	10000	ILP solved	0,010170	8,180170	0,012770
TEST_100x100x5_906	100	100	5	90%	5424	5475	9988	10000	ILP solved	0,009826	7,899826	0,010810
TEST_100x100x5_907	100	100	5	90%	5450	5581	9984	10000	ILP solved	0,009727	8,269727	0,011162

TEST_100x100x5_908	100	100	5	90%	5419	5548	9996	10000	ILP solved	0,009577	8,139577	0,009752
TEST_100x100x5_909	100	100	5	90%	5429	5579	9980	10000	ILP solved	0,010378	8,030378	0,012155
TEST_100x100x5_910	100	100	5	90%	5446	5548	9984	10000	ILP solved	0,010038	8,230038	0,011422
TEST_20x20x5_101	20	20	5	10%	37	36	326	400	ILP solved	0,000414	0,230414	0,002913
TEST_20x20x5_102	20	20	5	10%	36	36	316	400	ILP solved	0,000513	0,250513	0,010351
TEST_20x20x5_103	20	20	5	10%	37	35	329	400	ILP solved	0,000404	0,200404	0,003400
TEST_20x20x5_104	20	20	5	10%	36	37	319	400	ILP solved	0,000385	0,320385	0,003437
TEST_20x20x5_105	20	20	5	10%	34	35	333	400	ILP solved	0,000387	0,240387	0,003885
TEST_20x20x5_106	20	20	5	10%	39	39	288	400	ILP solved	0,000520	0,430520	0,007968
TEST_20x20x5_107	20	20	5	10%	33	39	311	400	ILP solved	0,000409	0,260409	0,006111
TEST_20x20x5_108	20	20	5	10%	39	39	288	400	ILP solved	0,000570	0,480570	0,007043
TEST_20x20x5_109	20	20	5	10%	36	38	290	400	ILP solved	0,000447	0,350447	0,006824
TEST_20x20x5_110	20	20	5	10%	37	39	296	400	ILP solved	0,000431	0,470431	0,006595
TEST_20x20x5_201	20	20	5	20%	70	69	214	400	ILP solved	0,001145	15,121145	0,009163
TEST_20x20x5_202	20	20	5	20%	65	71	271	400	ILP solved	0,000755	1,130755	0,006837
TEST_20x20x5_203	20	20	5	20%	67	70	253	400	ILP solved	0,000609	1,370609	0,005472
TEST_20x20x5_204	20	20	5	20%	68	67	218	400	ILP solved	0,000852	7,410852	0,009516
TEST_20x20x5_205	20	20	5	20%	71	72	180	400	ILP solved	0,000881	37,100881	0,020792
TEST_20x20x5_206	20	20	5	20%	61	68	185	400	ILP solved	0,000882	1,940882	0,036938
TEST_20x20x5_207	20	20	5	20%	67	75	145	400	ILP solved	0,000659	50,740659	0,016752
TEST_20x20x5_208	20	20	5	20%	67	74	168	400	ILP solved	0,000892	13,930892	0,014408
TEST_20x20x5_209	20	20	5	20%	74	70	198	400	ILP solved	0,000840	19,730840	0,015476
TEST_20x20x5_210	20	20	5	20%	70	71	225	400	ILP solved	0,000619	1,500619	0,015034
TEST_20x20x5_301	20	20	5	30%	94	101	258	400	ILP solved	0,001057	0,831057	0,009405
TEST_20x20x5_302	20	20	5	30%	92	100	205	400	ILP solved	0,001001	2,451001	0,012291
TEST_20x20x5_303	20	20	5	30%	99	98	216	400	ILP solved	0,001086	1,501086	0,009935
TEST_20x20x5_304	20	20	5	30%	97	96	273	400	ILP solved	0,000938	0,700938	0,009325
TEST_20x20x5_305	20	20	5	30%	98	99	202	400	ILP solved	0,001065	6,351065	0,012081
TEST_20x20x5_306	20	20	5	30%	93	102	207	400	ILP solved	0,000732	2,380732	0,008592
TEST_20x20x5_307	20	20	5	30%	103	97	182	400	ILP solved	0,000775	30,420775	0,017811
TEST_20x20x5_308	20	20	5	30%	101	102	236	400	ILP solved	0,001004	1,361004	0,011676

TEST_20x20x5_309	20	20	5	30%	90	92	244	400	ILP solved	0,001057	1,051057	0,013764
TEST_20x20x5_310	20	20	5	30%	93	94	212	400	ILP solved	0,000763	22,870763	0,009014
TEST_20x20x5_401	20	20	5	40%	116	121	322	400	ILP solved	0,000869	0,390869	0,002639
TEST_20x20x5_402	20	20	5	40%	120	129	284	400	ILP solved	0,000930	0,470930	0,004633
TEST_20x20x5_403	20	20	5	40%	113	122	317	400	ILP solved	0,000855	0,420855	0,003291
TEST_20x20x5_404	20	20	5	40%	116	129	341	400	ILP solved	0,001019	0,331019	0,002445
TEST_20x20x5_405	20	20	5	40%	122	120	324	400	ILP solved	0,000855	0,430855	0,003408
TEST_20x20x5_406	20	20	5	40%	115	128	279	400	ILP solved	0,001039	0,901039	0,006697
TEST_20x20x5_407	20	20	5	40%	116	129	318	400	ILP solved	0,000776	0,380776	0,003141
TEST_20x20x5_408	20	20	5	40%	118	125	344	400	ILP solved	0,000819	0,310819	0,002001
TEST_20x20x5_409	20	20	5	40%	122	128	324	400	ILP solved	0,000900	0,350900	0,003001
TEST_20x20x5_410	20	20	5	40%	116	124	295	400	ILP solved	0,001006	0,561006	0,005598
TEST_20x20x5_501	20	20	5	50%	145	153	328	400	ILP solved	0,000841	0,390841	0,002214
TEST_20x20x5_502	20	20	5	50%	136	139	375	400	ILP solved	0,000703	0,250703	0,001143
TEST_20x20x5_503	20	20	5	50%	138	155	392	400	ILP solved	0,000706	0,240706	0,000791
TEST_20x20x5_504	20	20	5	50%	138	159	388	400	ILP solved	0,000749	0,240749	0,000926
TEST_20x20x5_505	20	20	5	50%	134	155	388	400	ILP solved	0,000647	0,220647	0,000732
TEST_20x20x5_506	20	20	5	50%	129	146	329	400	ILP solved	0,000874	0,330874	0,002320
TEST_20x20x5_507	20	20	5	50%	136	153	384	400	ILP solved	0,000734	0,230734	0,000995
TEST_20x20x5_508	20	20	5	50%	147	158	371	400	ILP solved	0,000758	0,260758	0,001309
TEST_20x20x5_509	20	20	5	50%	136	155	334	400	ILP solved	0,000731	0,370731	0,003172
TEST_20x20x5_510	20	20	5	50%	146	146	368	400	ILP solved	0,000728	0,280728	0,001533
TEST_20x20x5_601	20	20	5	60%	165	171	392	400	ILP solved	0,000661	0,200661	0,000750
TEST_20x20x5_602	20	20	5	60%	167	167	388	400	ILP solved	0,000612	0,230612	0,000778
TEST_20x20x5_603	20	20	5	60%	163	169	396	400	ILP solved	0,000657	0,210657	0,000684
TEST_20x20x5_604	20	20	5	60%	150	168	388	400	ILP solved	0,000737	0,210737	0,000841
TEST_20x20x5_605	20	20	5	60%	181	178	378	400	ILP solved	0,000691	0,240691	0,001306
TEST_20x20x5_606	20	20	5	60%	151	165	384	400	ILP solved	0,000644	0,200644	0,000911
TEST_20x20x5_607	20	20	5	60%	160	183	396	400	ILP solved	0,000639	0,210639	0,000666
TEST_20x20x5_608	20	20	5	60%	162	187	400	400	logically	0,000645	0,000645	0,000651
TEST_20x20x5_609	20	20	5	60%	166	171	392	400	ILP solved	0,000713	0,220713	0,000813

TEST_20x20x5_610	20	20	5	60%	149	173	389	400	ILP solved	0,000687	0,230687	0,000826
TEST_20x20x5_701	20	20	5	70%	189	207	396	400	ILP solved	0,000551	0,190551	0,000581
TEST_20x20x5_702	20	20	5	70%	182	199	396	400	ILP solved	0,000536	0,180536	0,000564
TEST_20x20x5_703	20	20	5	70%	190	205	400	400	logically	0,000494	0,000494	0,000499
TEST_20x20x5_704	20	20	5	70%	180	194	400	400	logically	0,000504	0,000504	0,000508
TEST_20x20x5_705	20	20	5	70%	184	194	392	400	ILP solved	0,000566	0,190566	0,000664
TEST_20x20x5_706	20	20	5	70%	187	198	389	400	ILP solved	0,000543	0,200543	0,000698
TEST_20x20x5_707	20	20	5	70%	182	206	400	400	logically	0,000564	0,000564	0,000504
TEST_20x20x5_708	20	20	5	70%	176	204	392	400	ILP solved	0,000584	0,190584	0,000689
TEST_20x20x5_709	20	20	5	70%	186	188	400	400	logically	0,000496	0,000496	0,000498
TEST_20x20x5_710	20	20	5	70%	183	202	400	400	logically	0,000493	0,000493	0,000493
TEST_20x20x5_801	20	20	5	80%	187	205	400	400	logically	0,000408	0,000408	0,000413
TEST_20x20x5_802	20	20	5	80%	194	229	396	400	ILP solved	0,000491	0,160491	0,000483
TEST_20x20x5_803	20	20	5	80%	210	221	400	400	logically	0,000412	0,000412	0,000413
TEST_20x20x5_804	20	20	5	80%	198	227	400	400	logically	0,000432	0,000432	0,000444
TEST_20x20x5_805	20	20	5	80%	199	220	400	400	logically	0,000403	0,000403	0,000403
TEST_20x20x5_806	20	20	5	80%	200	225	400	400	logically	0,000417	0,000417	0,000423
TEST_20x20x5_807	20	20	5	80%	201	212	396	400	ILP solved	0,000428	0,150428	0,000456
TEST_20x20x5_808	20	20	5	80%	213	227	396	400	ILP solved	0,000477	0,160477	0,000512
TEST_20x20x5_809	20	20	5	80%	191	216	400	400	logically	0,000432	0,000432	0,000437
TEST_20x20x5_810	20	20	5	80%	215	229	396	400	ILP solved	0,000424	0,140424	0,000470
TEST_20x20x5_901	20	20	5	90%	236	237	400	400	logically	0,000283	0,000283	0,000287
TEST_20x20x5_902	20	20	5	90%	229	239	400	400	logically	0,000298	0,000298	0,000304
TEST_20x20x5_903	20	20	5	90%	214	234	400	400	logically	0,000310	0,000310	0,000317
TEST_20x20x5_904	20	20	5	90%	215	236	400	400	logically	0,000294	0,000294	0,000311
TEST_20x20x5_905	20	20	5	90%	231	247	400	400	logically	0,000347	0,000347	0,000319
TEST_20x20x5_906	20	20	5	90%	231	258	400	400	logically	0,000307	0,000307	0,000311
TEST_20x20x5_907	20	20	5	90%	212	238	400	400	logically	0,000315	0,000315	0,000318
TEST_20x20x5_908	20	20	5	90%	207	241	400	400	logically	0,000340	0,000340	0,000346
TEST_20x20x5_909	20	20	5	90%	231	243	400	400	logically	0,000341	0,000341	0,000347
TEST_20x20x5_910	20	20	5	90%	204	231	400	400	logically	0,000304	0,000304	0,000306

TEST_40x60x5_101	40	60	5	10%	217	224	1049	2400	ILP unsolved	0,004056	(timeout)	7,161453
TEST_40x60x5_102	40	60	5	10%	221	225	1022	2400	ILP unsolved	0,006858	(timeout)	(timeout)
TEST_40x60x5_103	40	60	5	10%	218	218	1068	2400	ILP unsolved	0,004164	(timeout)	0,950963
TEST_40x60x5_104	40	60	5	10%	221	215	1048	2400	ILP unsolved	0,004068	(timeout)	(timeout)
TEST_40x60x5_105	40	60	5	10%	219	225	1090	2400	ILP unsolved	0,007291	(timeout)	0,878390
TEST_40x60x5_106	40	60	5	10%	215	216	989	2400	ILP unsolved	0,007316	(timeout)	0,960155
TEST_40x60x5_107	40	60	5	10%	228	225	1017	2400	ILP unsolved	0,004359	(timeout)	1,030183
TEST_40x60x5_108	40	60	5	10%	211	219	988	2400	ILP unsolved	0,004447	(timeout)	0,851491
TEST_40x60x5_109	40	60	5	10%	214	212	1055	2400	ILP unsolved	0,006911	(timeout)	0,858506
TEST_40x60x5_110	40	60	5	10%	219	218	993	2400	ILP unsolved	0,007247	(timeout)	1,089528
TEST_40x60x5_201	40	60	5	20%	398	402	486	2400	ILP unsolved	0,010791	(timeout)	(timeout)
TEST_40x60x5_202	40	60	5	20%	390	419	466	2400	ILP unsolved	0,009551	(timeout)	(timeout)
TEST_40x60x5_203	40	60	5	20%	387	412	450	2400	ILP unsolved	0,009522	(timeout)	(timeout)
TEST_40x60x5_204	40	60	5	20%	403	410	441	2400	ILP unsolved	0,015227	(timeout)	(timeout)
TEST_40x60x5_205	40	60	5	20%	398	426	389	2400	ILP unsolved	0,010486	(timeout)	(timeout)
TEST_40x60x5_206	40	60	5	20%	390	420	425	2400	ILP unsolved	0,011065	(timeout)	(timeout)
TEST_40x60x5_207	40	60	5	20%	390	411	354	2400	ILP unsolved	0,011416	(timeout)	3,118772
TEST_40x60x5_208	40	60	5	20%	393	405	453	2400	ILP unsolved	0,009704	(timeout)	6,219153
TEST_40x60x5_209	40	60	5	20%	401	406	500	2400	ILP unsolved	0,009744	(timeout)	(timeout)
TEST_40x60x5_210	40	60	5	20%	404	406	462	2400	ILP unsolved	0,010216	(timeout)	(timeout)
TEST_40x60x5_301	40	60	5	30%	547	571	414	2400	ILP unsolved	0,015649	(timeout)	(timeout)
TEST_40x60x5_302	40	60	5	30%	544	583	408	2400	ILP unsolved	0,016456	(timeout)	(timeout)
TEST_40x60x5_303	40	60	5	30%	544	573	421	2400	ILP unsolved	0,011400	(timeout)	(timeout)
TEST_40x60x5_304	40	60	5	30%	569	588	293	2400	ILP unsolved	0,012222	(timeout)	(timeout)
TEST_40x60x5_305	40	60	5	30%	562	579	270	2400	ILP unsolved	0,012200	(timeout)	(timeout)
TEST_40x60x5_306	40	60	5	30%	577	578	389	2400	ILP unsolved	0,013161	(timeout)	(timeout)
TEST_40x60x5_307	40	60	5	30%	557	565	359	2400	ILP unsolved	0,011697	(timeout)	(timeout)
TEST_40x60x5_308	40	60	5	30%	583	589	261	2400	ILP unsolved	0,017496	(timeout)	(timeout)
TEST_40x60x5_309	40	60	5	30%	546	559	372	2400	ILP unsolved	0,011416	(timeout)	(timeout)
TEST_40x60x5_310	40	60	5	30%	552	589	344	2400	ILP unsolved	0,017076	(timeout)	(timeout)
TEST_40x60x5_401	40	60	5	40%	682	695	1088	2400	ILP unsolved	0,014632	(timeout)	2,132511

TEST_40x60x5_402	40	60	5	40%	711	720	688	2400	ILP unsolved	0,015723	(timeout)	(timeout)
TEST_40x60x5_403	40	60	5	40%	670	703	926	2400	ILP unsolved	0,019880	(timeout)	(timeout)
TEST_40x60x5_404	40	60	5	40%	692	713	839	2400	ILP unsolved	0,020291	(timeout)	2,202518
TEST_40x60x5_405	40	60	5	40%	698	730	715	2400	ILP unsolved	0,016832	(timeout)	(timeout)
TEST_40x60x5_406	40	60	5	40%	710	724	848	2400	ILP unsolved	0,016570	(timeout)	(timeout)
TEST_40x60x5_407	40	60	5	40%	706	722	848	2400	ILP unsolved	0,015446	(timeout)	1,877640
TEST_40x60x5_408	40	60	5	40%	684	719	707	2400	ILP unsolved	0,016324	(timeout)	1,915703
TEST_40x60x5_409	40	60	5	40%	681	719	719	2400	ILP unsolved	0,020104	(timeout)	(timeout)
TEST_40x60x5_410	40	60	5	40%	681	695	1147	2400	ILP unsolved	0,018040	(timeout)	2,089891
TEST_40x60x5_501	40	60	5	50%	798	876	2192	2400	ILP solved	0,007171	4,687171	0,035270
TEST_40x60x5_502	40	60	5	50%	808	863	2089	2400	ILP solved	0,009775	38,719775	0,080904
TEST_40x60x5_503	40	60	5	50%	823	845	2068	2400	ILP solved	0,008539	6,218539	0,086764
TEST_40x60x5_504	40	60	5	50%	822	853	1992	2400	ILP solved	0,008390	7,738390	0,121264
TEST_40x60x5_505	40	60	5	50%	805	863	2176	2400	ILP solved	0,007948	4,607948	0,054787
TEST_40x60x5_506	40	60	5	50%	813	855	1902	2400	ILP unsolved	0,010539	(timeout)	0,152320
TEST_40x60x5_507	40	60	5	50%	827	844	2102	2400	ILP unsolved	0,010465	(timeout)	0,045805
TEST_40x60x5_508	40	60	5	50%	803	848	2133	2400	ILP unsolved	0,008828	(timeout)	0,069862
TEST_40x60x5_509	40	60	5	50%	828	864	2134	2400	ILP unsolved	0,007736	(timeout)	0,046670
TEST_40x60x5_510	40	60	5	50%	818	853	2227	2400	ILP unsolved	0,007792	(timeout)	0,029974
TEST_40x60x5_601	40	60	5	60%	929	961	2346	2400	ILP unsolved	0,004634	(timeout)	0,008215
TEST_40x60x5_602	40	60	5	60%	915	998	2315	2400	ILP unsolved	0,005015	(timeout)	0,013051
TEST_40x60x5_603	40	60	5	60%	911	982	2341	2400	ILP unsolved	0,004687	(timeout)	0,009482
TEST_40x60x5_604	40	60	5	60%	907	951	2312	2400	ILP unsolved	0,005095	(timeout)	0,012224
TEST_40x60x5_605	40	60	5	60%	957	996	2343	2400	ILP unsolved	0,004957	(timeout)	0,008120
TEST_40x60x5_606	40	60	5	60%	933	978	2353	2400	ILP unsolved	0,004897	(timeout)	0,007288
TEST_40x60x5_607	40	60	5	60%	959	963	2364	2400	ILP unsolved	0,004868	(timeout)	0,006551
TEST_40x60x5_608	40	60	5	60%	934	965	2330	2400	ILP unsolved	0,004742	(timeout)	0,010551
TEST_40x60x5_609	40	60	5	60%	936	985	2319	2400	ILP unsolved	0,004923	(timeout)	0,009544
TEST_40x60x5_610	40	60	5	60%	963	1011	2278	2400	ILP unsolved	0,004887	(timeout)	0,018731
TEST_40x60x5_701	40	60	5	70%	1056	1111	2365	2400	ILP unsolved	0,003758	(timeout)	0,004991
TEST_40x60x5_702	40	60	5	70%	1082	1138	2372	2400	ILP unsolved	0,003831	(timeout)	0,005366

TEST_40x60x5_703	40	60	5	70%	1017	1108	2369	2400	ILP unsolved	0,003817	(timeout)	0,005437
TEST_40x60x5_704	40	60	5	70%	1052	1088	2359	2400	ILP unsolved	0,004348	(timeout)	0,006447
TEST_40x60x5_705	40	60	5	70%	1062	1109	2372	2400	ILP unsolved	0,003827	(timeout)	0,004833
TEST_40x60x5_706	40	60	5	70%	1013	1092	2368	2400	ILP unsolved	0,003998	(timeout)	0,005422
TEST_40x60x5_707	40	60	5	70%	1048	1089	2384	2400	ILP unsolved	0,003638	(timeout)	0,004216
TEST_40x60x5_708	40	60	5	70%	1050	1091	2382	2400	ILP unsolved	0,003619	(timeout)	0,004144
TEST_40x60x5_709	40	60	5	70%	1069	1117	2368	2400	ILP unsolved	0,003793	(timeout)	0,004852
TEST_40x60x5_710	40	60	5	70%	1034	1102	2372	2400	ILP unsolved	0,003678	(timeout)	0,005246
TEST_40x60x5_801	40	60	5	80%	1171	1229	2392	2400	ILP unsolved	0,002943	(timeout)	0,003104
TEST_40x60x5_802	40	60	5	80%	1207	1264	2381	2400	ILP unsolved	0,003154	(timeout)	0,003855
TEST_40x60x5_803	40	60	5	80%	1211	1265	2384	2400	ILP unsolved	0,003259	(timeout)	0,003591
TEST_40x60x5_804	40	60	5	80%	1152	1228	2380	2400	ILP unsolved	0,002903	(timeout)	0,003822
TEST_40x60x5_805	40	60	5	80%	1199	1247	2400	2400	logically	0,002677	0,002677	0,002650
TEST_40x60x5_806	40	60	5	80%	1234	1312	2388	2400	ILP unsolved	0,003028	(timeout)	0,003408
TEST_40x60x5_807	40	60	5	80%	1221	1240	2392	2400	ILP unsolved	0,003086	(timeout)	0,003277
TEST_40x60x5_808	40	60	5	80%	1195	1225	2396	2400	ILP unsolved	0,002960	(timeout)	0,003011
TEST_40x60x5_809	40	60	5	80%	1192	1259	2396	2400	ILP unsolved	0,003330	(timeout)	0,003027
TEST_40x60x5_810	40	60	5	80%	1230	1243	2380	2400	ILP unsolved	0,003022	(timeout)	0,003947
TEST_40x60x5_901	40	60	5	90%	1332	1388	2400	2400	logically	0,002072	0,002072	0,002078
TEST_40x60x5_902	40	60	5	90%	1337	1388	2396	2400	ILP unsolved	0,001988	(timeout)	0,002040
TEST_40x60x5_903	40	60	5	90%	1334	1382	2396	2400	ILP unsolved	0,002103	(timeout)	0,002191
TEST_40x60x5_904	40	60	5	90%	1283	1366	2396	2400	ILP unsolved	0,002013	(timeout)	0,002061
TEST_40x60x5_905	40	60	5	90%	1326	1364	2396	2400	ILP unsolved	0,002260	(timeout)	0,002317
TEST_40x60x5_906	40	60	5	90%	1291	1344	2396	2400	ILP unsolved	0,002205	(timeout)	0,002254
TEST_40x60x5_907	40	60	5	90%	1302	1357	2400	2400	logically	0,001986	0,001986	0,001994
TEST_40x60x5_908	40	60	5	90%	1320	1368	2400	2400	logically	0,001861	0,001861	0,001865
TEST_40x60x5_909	40	60	5	90%	1344	1389	2396	2400	ILP unsolved	0,002485	(timeout)	0,002290
TEST_40x60x5_910	40	60	5	90%	1318	1383	2392	2400	ILP unsolved	0,001822	(timeout)	0,002014

Legend:

Title: Title of the nonogram
W: With of the nonogram
H: Height of the nonogram
C: Number of colors of the nonogram
Dens.: Global density of the nonogram
H Blks: Number of horizontal blocks of the nonogram
V Blks: Number of vertical blocks of the nonogram
Cells logically solved: Number of cells determined after ly solving (fully or partially) the nonogram
Cells: Number of cells of the nonogram (typically W×H)
ILP state: Logically solved, ILP solved or ILP unsolved
Logic time: Time spent logically solving the nonogram
ILP Total Time: Total time spent solving the nonogram using the hybrid, including the time spent logically solving it
Iterative Total Time: Total time spent solving the nonogram using the iterative approach, including the time spent logically solving it

B . Nonogram File Formats

B.1 Bosch based file format

```
title: TEST_20x20x5_101
number_of_rows: 20
number_of_columns: 20
number_of_colors: 5

row_1:
number_of_clusters: 3
size(s): 1 1 1
color(s): 1 1 1

row_2:
number_of_clusters: 1
size(s): 1
color(s): 1

row_3:
number_of_clusters: 3
size(s): 1 3 1
color(s): 1 1 2

row_4:
number_of_clusters: 1
size(s): 1
color(s): 1

row_5:
number_of_clusters: 3
size(s): 1 1 1
color(s): 2 2 2

row_6:
number_of_clusters: 0
size(s):
color(s):

row_7:
```

```
number_of_clusters: 0
size(s):
color(s):

row_8:
number_of_clusters: 1
size(s): 1
color(s): 2

row_9:
number_of_clusters: 2
size(s): 1 1
color(s): 3 3

row_10:
number_of_clusters: 0
size(s):
color(s):

row_11:
number_of_clusters: 5
size(s): 1 1 1 1 1
color(s): 3 3 4 3 3

row_12:
number_of_clusters: 3
size(s): 1 1 1
color(s): 3 3 3

row_13:
number_of_clusters: 2
size(s): 1 1
color(s): 4 4

row_14:
number_of_clusters: 2
size(s): 1 1
color(s): 4 4

row_15:
number_of_clusters: 1
```

```
size(s): 1
color(s): 4

row_16:
number_of_clusters: 3
size(s): 1 2 1
color(s): 4 4 1

row_17:
number_of_clusters: 2
size(s): 1 1
color(s): 5 1

row_18:
number_of_clusters: 2
size(s): 1 1
color(s): 5 5

row_19:
number_of_clusters: 2
size(s): 1 1
color(s): 5 5

row_20:
number_of_clusters: 1
size(s): 1
color(s): 5

column_1:
number_of_clusters: 0
size(s):
color(s):

column_2:
number_of_clusters: 1
size(s): 1
color(s): 1

column_3:
number_of_clusters: 0
size(s):
```

```
color(s):

column_4:
number_of_clusters: 1
size(s): 1
color(s): 4

column_5:
number_of_clusters: 3
size(s): 2 1 1
color(s): 3 5 5

column_6:
number_of_clusters: 2
size(s): 1 1
color(s): 2 1

column_7:
number_of_clusters: 2
size(s): 1 1
color(s): 1 3

column_8:
number_of_clusters: 1
size(s): 2
color(s): 3

column_9:
number_of_clusters: 4
size(s): 1 1 1 1
color(s): 1 4 4 5

column_10:
number_of_clusters: 2
size(s): 1 1
color(s): 1 3

column_11:
number_of_clusters: 4
size(s): 1 1 1 1
color(s): 1 2 4 4
```

```
column_12:
number_of_clusters: 2
size(s): 2 1
color(s): 1 3

column_13:
number_of_clusters: 1
size(s): 1
color(s): 1

column_14:
number_of_clusters: 1
size(s): 1
color(s): 4

column_15:
number_of_clusters: 5
size(s): 1 1 1 2 1
color(s): 1 3 3 4 4

column_16:
number_of_clusters: 2
size(s): 1 1
color(s): 2 5

column_17:
number_of_clusters: 1
size(s): 1
color(s): 5

column_18:
number_of_clusters: 1
size(s): 1
color(s): 2

column_19:
number_of_clusters: 3
size(s): 1 1 1
color(s): 2 1 5
```

```
column_20:
number_of_clusters: 0
size(s):
color(s):
```

B.2 Olšák file format

```
Title: TEST_20x20x5_101
#d
   a:a   1
   b:b   2
   c:c   3
   d:d   4
   e:e   5
: rows
1a 1a 1a
1a
1a 3a 1b
1a
1b 1b 1b


1b
1c 1c

1c 1c 1d 1c 1c
1c 1c 1c
1d 1d
1d 1d
1d
1d 2d 1a
1e 1a
1e 1e
1e 1e
1e
: columns

1a

1d
```

```
2c 1e 1e
1b 1a
1a 1c
2c
1a 1d 1d 1e
1a 1c
1a 1b 1d 1d
2a 1c
1a
1d
1a 1c 1c 2d 1d
1b 1e
1e
1b
1b 1a 1e

: end
```

B.3 Hett based file format

```
specimen_nonogram('TEST_20x20x5_101',
[[[1,1],[1,1],[1,1]]
[[1,1]]
[[1,1],[3,1],[1,2]]
[[1,1]]
[[1,2],[1,2],[1,2]]
[]
[]
[[1,2]]
[[1,3],[1,3]]
[]
[[1,3],[1,3],[1,4],[1,3],[1,3]]
[[1,3],[1,3],[1,3]]
[[1,4],[1,4]]
[[1,4],[1,4]]
[[1,4]]
[[1,4],[2,4],[1,1]]
[[1,5],[1,1]]
[[1,5],[1,5]]
[[1,5],[1,5]]
```

```
[[1,5]]
],
[[]
[[1,1]]
[]
[[1,4]]
[[2,3],[1,5],[1,5]]
[[1,2],[1,1]]
[[1,1],[1,3]]
[[2,3]]
[[1,1],[1,4],[1,4],[1,5]]
[[1,1],[1,3]]
[[1,1],[1,2],[1,4],[1,4]]
[[2,1],[1,3]]
[[1,1]]
[[1,4]]
[[1,1],[1,3],[1,3],[2,4],[1,4]]
[[1,2],[1,5]]
[[1,5]]
[[1,2]]
[[1,2],[1,1],[1,5]]
[]
]
).
--------------------------------
```

Bibliography

[1] The eclipse constraint programming system. http://www.eclipse-clp.org/.

[2] Griddlers net. http://www.griddlers.net/.

[3] Ilog cplex. http://www.ilog.com/products/cplex/.

[4] Scip: Solving constraint integer programs. http://scip.zib.de/.

[5] Egon Balas and Robert Jeroslow. Canonical cuts on the unit hypercube. *SIAM Journal on Applied Mathematics*, 23(1):61–69, 1972.

[6] Colin Barker. LPA Win-Prolog Goodies. http://pagesperso-orange.fr/colin.barker/lpa/lpa.htm.

[7] Robert A. Bosch. Painting by numbers. *Optima*, (65):16–17, May 2001. Also available here http://www.oberlin.edu/math/faculty/bosch/pbn.ps.

[8] Ali Corbin. Ali corbin's home page. http://www.blindchicken.com/~ali/.

[9] Brian Grainger. Pencil puzzles and sudoku. http://www.icpug.org.uk/national/features/050424fe.htm.

[10] Werner Hett. Hett nonogram solver in prolog. https://prof.ti.bfh.ch/hew1/informatik3/prolog/p-99/.

[11] Javier Larrosa and Enric Morancho. Solving 'still life' with soft constraints and bucket elimination. In Francesca Rossi, editor, *Principles and Practice of Constraint Programming-CP 2003: 9th International Conference, CP 2003, Kinsale, Ireland, September 29-October 3, 2003 : Proceedings*, pages 466–479, Kinsale, Ireland, 2003. Springer.

[12] Luís Mingote and Francisco Azevedo. Colored nonograms: an integer linear programming approach. In Lopes *et al.*, editor, *Progress in Artificial Intelligence, 14th Portuguese Conference on Artificial Intelligence, EPIA 2009*, Aveiro, Portugal, 2009. Springer.

[13] Mirek Olšák and Petr Olšák. Griddlers solver, nonogram solver. http://www.olsak.net/grid.html#English.

[14] Joachim Schimpf. ECLiPSe Code Samples. http://eclipse.crosscoreop.com/examples/.

[15] Steven Simpson. Nonogram programs. http://www.comp.lancs.ac.uk/~ss/software/nonowimp/.

[16] Steven Simpson. Nonogram solver. http://www.comp.lancs.ac.uk/~ss/nonogram/.

[17] Steven Simpson. Nonogram solver - solvers on the web. http://www.comp.lancs.ac.uk/~ss/nonogram/list-solvers.

[18] Jung-Fa Tsai, Ming-Hua Lin, and Yi-Chung Hu. Finding multiple solutions to general integer linear programs. *European Journal of Operational Research*, 184(2):802–809, 2008.

[19] Nobuhisa Ueda and Tadaaki Nagao. Np-completeness results for nonogram via parsimonious reductions. Technical Report TR96-0008, Tokyo Institute of Technology (Titech), Department of Computer Science, http://www.cs.titech.ac.jp/~tr/reports/1996/TR96-0008.ps.gz, May 1996.

[20] Wouter Wiggers. A comparison of a genetic algorithm and a depth first search algorithm applied to japanese nonograms. Paper, Faculty of EECMS, University of Twente, 2004.

[21] Wikipedia. Nonogram. http://en.wikipedia.org/wiki/Nonogram.

[22] Jan Wolter. The 'pbnsolve' paint-by-number puzzle solver. http://webpbn.com/pbnsolve.html.

[23] Jan Wolter. Survey of paint-by-number puzzle solvers. http://webpbn.com/survey/.