University of Toronto
Faculty of Arts and Science
CSC384f - Introduction to Artificial Intelligence
Final Project Report: Picross (Nonogram) Solver

Submission Date	December 8, 2015
Team Members	Flavio Matheus Muniz Ribeiro da Silva - CDF id: c5mubizr Rodrigo Longhi Guimarães - CDF id: c5longhi
Project Type	Constraint Satisfaction Problem (CSP)

1. The problem

Picrosses are one of the more challenging logic puzzles. Also known as nonograms, these puzzles are a grid of cells and, when solved by filling some cells with a color, will reveal an image. A solution to one of these puzzles is given on Figure 1.

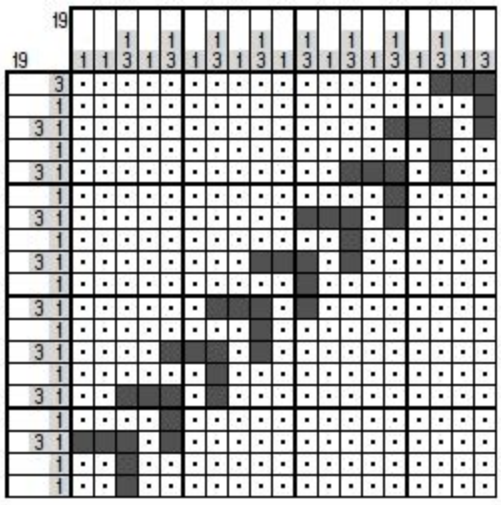


Figure 1 - Example of a Solved Picross[1]

The game is a rectangular grid, and in each row and column it possess a list of numbers. Both the order and the value of these numbers are useful to solve the puzzle. The rules are as follow:

A. Each square is either painted or blank. In Figure 1, the painted squares are represented by grey squares and the blank squares are represented by small dots;
B. The numbers represent the number of consecutive painted squares in the row/column they are in. Therefore, for a row with a single number, there is only a sequence of consecutive painted squares equal to this number. For a row with multiple numbers, there is a sequence of each number of painted squares, and all of these sequences are separated one from another by at least one empty space. The sequences are exactly in the order they appear in the number list.

It is noteworthy that some problems have multiple solutions and that some puzzles are very hard to solve based on logic only: large or sparse boards often force that you guess the state of a square.

This problem has already been solved in a multitude of ways, in various languages and with various approaches. Despite this, the problem is exponentially

harder depending on the board configuration, and new algorithms are always trying to find a way to increase performance and solve harder and harder puzzles.

2. Approach

Picrosses are all about figuring out the final image, the final state of the grid. Therefore, the order in what each of the cells is discovered to be painted or blank is not important, and this is a good indicator that search algorithms should not be used to solve this problem. Although the search algorithms could lead to a solution, they would need a lot of computational resources, more than CSPs seem to need for this type of problem. Moreover, Picross is a game. It has a set of rules that the player must obey in order to find the picture, and this is all what CSPs are about: finding assignments to a set of variables that satisfy a set of constraints. Lastly, since we are not dealing with any probabilities of a square being painted or blank whatsoever, Bayesian Nets don't seem like the perfect fit either. Clearly, based on the constraints for each row and column we could have calculated the probability of each square to be blank or painted and try to develop a solver using Bayesian nets.

After evaluating the three possibilities, we opted for using a CSP, since it seems to be a better fit for this problem, making it easier to model than the other options and still getting the solution.

To construct a CSP, you basically need to specify the variables and the constraints for the problem. As for the variables, it is not difficult to perceive that using each cell of the grid as a variable is not a bad approach: the other approach probably would be trying to use the rows and columns as variables, but that would generate gigantic domains and would need some extra constraints to ensure that the cells on the lines as columns that intersect each other should be equal.

The constraints, in turn, are already set as the numbers for each row and column, we just had to make the program interpret them as the rules of the game itself. To do that the constraints stored all possible solutions that a column or a row can have with the provided numbers.

After that, we just needed methods that used these constraints to find the unique solution for each column and row.

3. Performance and Results

Initially, some tests for correctness were performed, where we run the algorithm for some boards and compared the results with the solutions we had for these puzzles. After this, we decided how we would validate and test our algorithm: 1) First, we decided to see the impact of using general arc consistency (GAC) in relation to using

forward checking (FC) only, for this modelling of the problem; 2) Secondly, we planned on testing our algorithm for various boards, each time trying to increase the difficulty, and seeing how many of them it could solve in a reasonable amount of time; 3) Lastly, our idea was to compare our results with some of the state-of-the-art solvers for picrosses.

3.1 Forward Checking vs General Arc Consistency

Starting the first part of the tests we saw that, since our model of representing the boards had constraints over relatively large scopes (number of cells in a row or number of cell is a column), forward checking was not able to eliminate many values from the variables domains and this resulted in a very slow solver in comparison to the solver with general arc consistency, as you can see on Table 1 (DISCLAIMER: Most of the puzzles were exported from[4] and, as noted on the website, it is not a problem to use them if they are not posted to another website).

Puzzle	**FC Solution Time (s)**	**GAC Solution Time (s)**
5x5 X pattern	0.05336	0.00546
5x7 turtle	2.85422	0.01195
7x6 strange face	18.46682	0.00952
8x7 strange face	∞	0.01717
5x10 dancer	∞	0.04381

Table 1 - Solution time for different puzzles using forward checking and general arc consistency. Puzzles that took more than 10 minutes to solve were considered unsolvable (time = ∞).

Although we found some very small boards in which the forward checking was faster than the general arc consistency when the correctness tests were performed, it is pretty clear that the algorithm using GAC is many orders of magnitude faster than the FC when the board increases in difficulty.

3.2 Limits of our algorithm

Although we got great results for small boards, we saw that our algorithm was

not able to solve all boards in a reasonable time, and we would like to find out to which point it could be used. Therefore, we exported a set of puzzles from the Web Paint By Number website and started working on them. Our results are presented on Table 2.

Puzzle	**Solution Time (s)**
20x20 Cat	3.2
14x25 Skid	4.7
20x20 Smoke	11.1
23x20 All Animals go to Heaven	12.6
27x23 Bucks	153.75
25x25 Last Forever	∞
19x19 Domino	∞

Table 2 - Solution Time for different puzzles using our algorithm. Puzzles that took more than 20 minutes to solve were considered unsolvable (time = ∞).

Although it is possible to see a trend that makes the puzzle harder the larger the board is, it is also very clear that this is not the only factor. Analysing the numbers in the puzzle and the final solution for some of them, it is possible to see that sparse puzzles are harder to solve.

To understand this result, we can think of an example 10x10 board. This board would have 100 variables (one representing each cell) and 20 constraints (representing each row and column). Since the domain is true or false for each variable, we have a total of 2^{100} possible assignments to the board, making it completely impossible for an algorithm to check all the assignments. To surpass this, GAC will do a lot of elimination on the variable domains, which is equivalent to most of the techniques that human players use when they are playing. The problem is most of these techniques rely on painting squares that will be marked for sure, and on a sparse board there are very few of these.

3.3 State-of-the-art picross solving

While researching about nonograms some articles exploring efficient ways of

resolving them were found. In his blog post[2], Hakank explores algorithms that use constraints to solve them and how long they take to solve. He tested Google CP Solver, Geocode, MiniZinc LazyFD and others. In Table 3 you can see some of the results he achieved, these being in relation to the nonograms that we used to test our approach.

	Algorithms			
Puzzle	Google CP Solver	MiniZinc LazyFD	Geocode	Ours
20x20 Cat	0.12s	0.95s	0.1s	3.2s
14x25 Skid	0.03s	0.9s	0.01s	4.7s
20x20 Smoke	0.05s	1.18s	N/A	11.1s
19x19 Domino	3.9s	3.11s	2.59s	∞
27x23 Bucks	0.05s	1.6s	0.2s	153.75s
25x25 Last Forever	1.6s	1.99s	2.3s	∞

Table 3 - Time to solve nonograms for each one of the algorithms tested by Hakank and our algorithm (more tests on his blog post[2])

As you can see our algorithm had a poor performance compared to the other algorithms used to solve these puzzles. Getting to know a little more about these algorithms, mainly about Google Cp Solver[3], one of the best algorithms available for picross solving nowadays, we see that their approach is quite different: they use different and very efficient search algorithms and, mainly, they don't have to keep in memory all the assignments that satisfy a constraint. From that we can draw some conclusions: firstly, there is a better way to represent the nonograms than the one we used. Second, there are other ways of using CSPs and they can be more efficient than what we are using. Finally, we can say that our choice in using CSP to solve our problem was a good one, since other people seem to have used an approach very similar to it instead of using pure search or Bayes Nets to solve nonograms.

There is a lot more information about this subject available, and you can find a lot of resources about nonogram solvers in the Web Paint By Number tool for exporting puzzles[4].

4. Going Further - Multi-layered multi-colored puzzles

After testing our algorithm in terms of performance, we came to the conclusion that the amount of work needed to get close to the performance of the state-of-the-art algorithms is much greater than the subject would need as a final project. Therefore, instead of trying to get its performance better, we developed a creative way of changing the puzzle, to make it harder to solve and thus, creating the possibility to expand our algorithm to solve different problems. We found out that some multi-colored picrosses exist[5], and they are an expansion of the original game. We also found out that the majority of picross solvers are not able to solve multi-colored puzzles[4]. Inspired by this idea, we developed some rules to create a multi-layered multi-colored board that could represent richer images. An example of our game can be seen of Figure 2. The rules are as follow.

A. Each square is either painted or blank. When a square is painted, it can have exactly one color. The possible colors are found by looking the color at each number.
B. The numbers represent the number of consecutive painted squares in the row/column they are in. Therefore, for a row with a single number, there is only a sequence of consecutive painted squares equal to this number. For a row with multiple numbers, there is a sequence of each number of painted squares, and all of these sequences are separated one from another by at least one empty space or a space with a different color. The sequences are exactly in the order they appear in the number list. Each number has a color, and the consecutives squares it represents have all the same color as it.

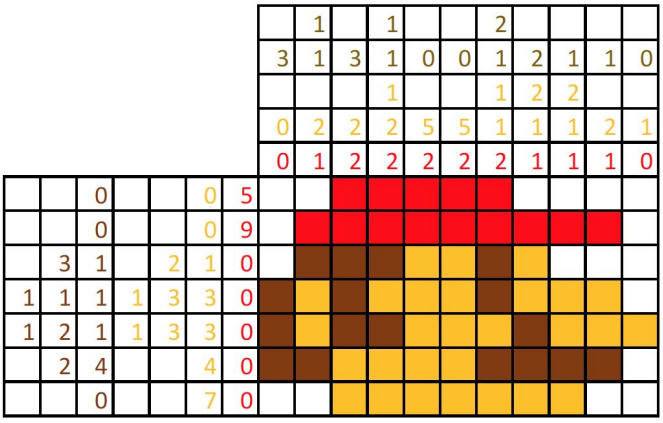


Figure 2 - Multi-Layered Multi-Colored Picross Boards respecting rules in section 4.

In this version of the game, the puzzle is basically multiple boards of monochromatic picrosses that have to be merged after each one is successfully solved. Using this idea, we constructed a function that splits the input board into multiple

monochromatic boards, solves each of them using the monochromatic model that we first proposed, and then merges all the result boards. To do this, this function supposes that none of the boards have multiple solutions that may lead to color overlapping, which is a reasonable assumptions for a big set of puzzles and is something that we did not worried about when we developed the rules for this new version of the game. Finally, this function calls another function that we have developed, that creates a PNM image file, containing the solution. The solution for the puzzle formulated in Figure 2 can be seen on Figure 3.

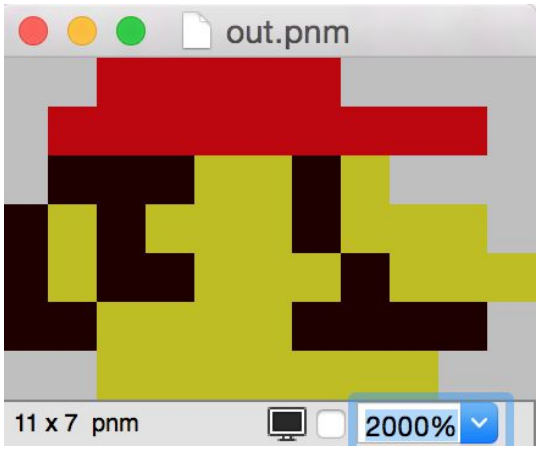


Figure 3 - Image generated as output when the board in Figure 2 is used as input. 20 times zoom was applied.

To represent the input board, the solver function expects a list with n different boards, where the boards are represented exactly the same as for the previous picross solver function and n is the number of colors in the puzzle. It also expects a list of lists, in which each of the inner lists is a set of 3 integers ranging from 0 to 255 and representing together a 24-bit RGB standard.

5. Final Thoughts

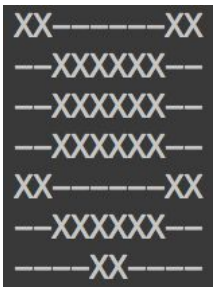
Our research, alongside with our results, proved that CSP are a good solution for this model, but there are other algorithms with better performance than purely using GAC and Forward Checking. Also, some types of search are very efficient in resolving the puzzles, such as Google CP using default search.

We also find that nonograms have been extensively researched by the community, and yet people still are trying to improve their findings even further. New types of nonograms have been created to increase the challenge and push further the boundaries of the game. 3D picrosses and colored nonograms are good examples of these new versions of the game that challenge the performance of solvers.

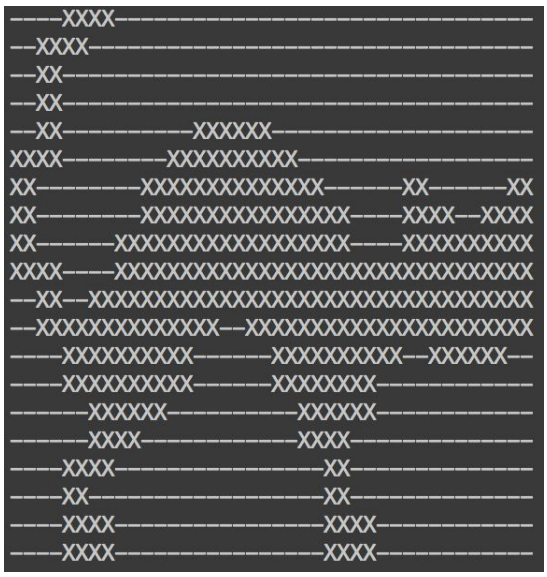
Appendix A

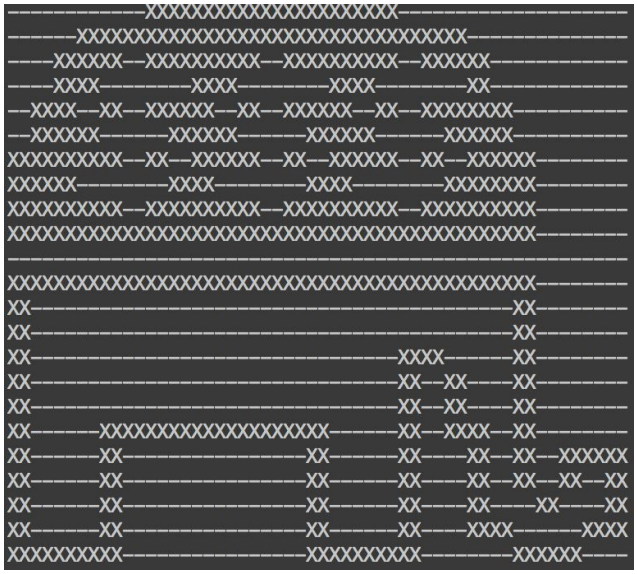
Solutions

These are the boards representing each one of the tests performed with our algorithm.

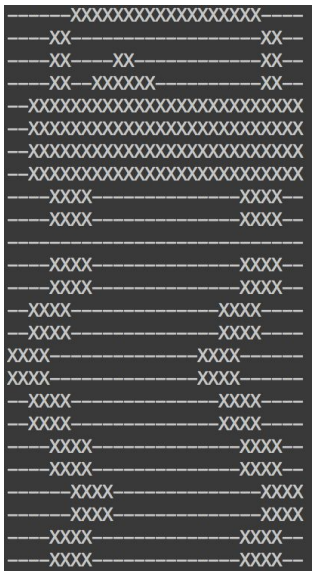


Figures 1, 2 and 3 - Strange Face 7x6, X Pattern 5x5 and Turtle 5x7





Figures 3 and 4 - Cat 20x20 and Bucks 27x23



Figures 6 and 7 - Skid 14x20 and Dancer 5x10

References

[1] Web Paint by Number. '9dom-c'. [Online]. Available at: http://webpbn.com/survey/9dom-c.png. [Accessed: 02- Dec- 2015].

[2] Hankank, 'Google CP Solver: A much faster Nonogram solver using DefaultSearch', 2010. [Online]. Available at: http://www.hakank.org/constraint_programming_blog/2010/11/google_cp_solver_a_much_faster_nonogram_solver_using_defaultsearch.html. [Accessed: 02- Dec- 2015].

[3] Google, 'Google Optimization Tools'. [Online]. Available at: https://developers.google.com/optimization/. [Accessed: 04- Dec- 2015].

[4] Web Paint by Number, 'Web Paint by Number Puzzle Export'. [Online]. Available at: http://webpbn.com/export.cgi. [Accessed: 05- Dec- 2015].

[5] Nonograms.org, 'Colour Japanese Crosswords'. [Online]. Available at: http://www.nonograms.org/nonograms2. [Accessed: 05- Dec- 2015].