AUA
American University *of* Armenia

Zaven & Sonia Akian College of Science & Engineering

CS246: Artificial Intelligence

Japanese Crossword/Nonogram Puzzle

Group: *Anna Mikayelyan, Ani Harutyunyan, Anzhela Davityan, Viktoria Melkumyan*

Instructor: *Vahram Tadevosyan*

Fall, 2024

Abstract

Japanese crosswords are logic puzzles also known as nonograms. Our paper demonstrates the algorithms that can be used to solve Japanese crosswords. We concentrate on the black-white version of the puzzle. Through the paper we explore and analyze the following algorithms:

- Backtracking Search Algorithm
- Line Solver

We implemented each algorithm to evaluate its effectiveness and understand how correctly that particular approach solves the nonogram of different grid sizes and levels of difficulty. We tested each algorithm to check its accuracy, how quick it works, and how well it is able to tackle larger and more complex puzzles. Based on the results of our experiments, we suggest the most accurate approaches to solve the Japanese crossword puzzle and highlight each algorithm's strengths and drawbacks.

***Keywords*:** *AI, Puzzle, Algorithms, project, Japanese Crossword, Nonogram, Constraint Satisfaction, Backtracking, Local Search, Backtracking, Forward Checking Genetic Algorithm*

Content

List of Figures

Chapter 1

Introduction

Nonograms [1], also known as Hanjie, Paint by Numbers, or Griddlers, are one of the popular logic puzzles invented by a Japanese graphics editor named Non Ishida in 1988. In 1990, the U.K. newspaper "The Sunday Telegraph" started publishing nonogram puzzles on a weekly basis [7]**.** In the beginning, Nonogram puzzles were considered to be the simpler version of Sudoku and a feature in most magazines. At the beginning of the new millennium, they gained popularity throughout the whole world.

Its earlier versions introduced a grid which was made of squares, along with horizontally and vertically aligned spaces that required certain numbers to be placed on top of each other. Players could use these numbers to understand which squares of the grid they needed to fill in to reveal a shape or image that was made up by creators.

1.1 Problem Setting and Description

The puzzle provides us with **clues** *see* ***Fig. 1.1.a*** at the beginning of each row and column. The clues are presented as numbers in the puzzle representing the lengths of consecutive groups of filled cells for the particular row or particular column and each group is separated by at least one empty cell. For example, if there

is written "3 2" for a row as a clue it means there is a group of three filled cells, followed by at least one empty cell, and then two consecutive filled cells.

These puzzles can be set up on an m x n matrix and may contain as few as 5 or 10 numbers though they may contain even many more numbers and there is more thinking needed to find the right solution. Let's understand the solution process which follows a set of rules to ensure accuracy:

- **A cell can be either filled or left blank** depending on the tip given in each row as well as the tip given in each column.
- **Each row and column must be completed** in a way that matches its clues exactly.
- **Clues must be respected in both directions:** take care of rows and columns since the aim is to meet all constraints without violating any hint.

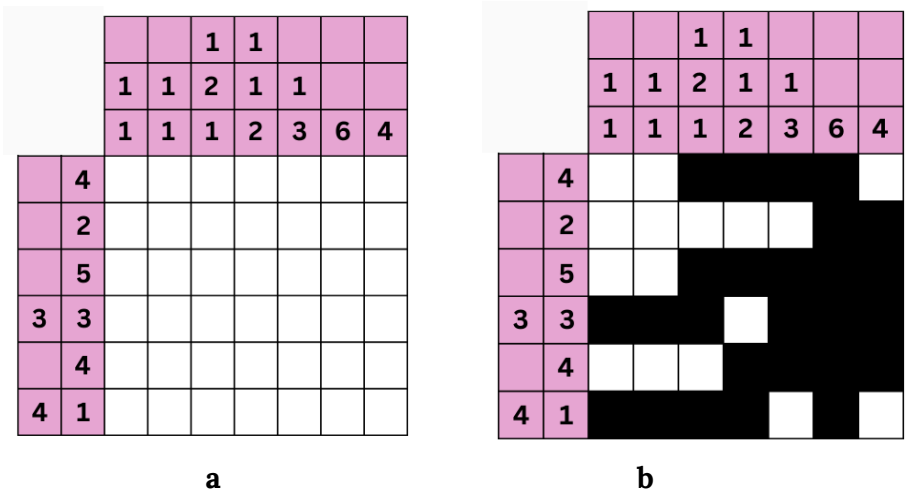


Figure 1.1: *Example of Nonogram puzzle: Initial & Final states*

So, for this example, we have a grid of 6 x 7 size with the corresponding clues for each row and column. The puzzle can be considered successfully completed, if the grid is filled with black squares without violating the given clues, *see* ***Fig. 1.1.b*** In

other words all the constraints should be met without any remaining conflict. The solution is expected to be valid, complete and verifiable.

1.2 Known Manual Solution Approaches

In this subsection several approaches for manually solving the Japanese Crosswords/Nonograms will be introduced and explained with their corresponding examples[9].

1.2.1 Evident Deduction

Fill an entire row or column that can be filled automatically when all the cells that need to be filled in that particular line have already been given in the clue. For example, *see* ***Fig 1.2*** when we have a clue containing the number “10” or “2, 7”, “5, 4” and targeting a row with 10 cells, the entire row should be filled. This gradually reduces the number of empty cells.

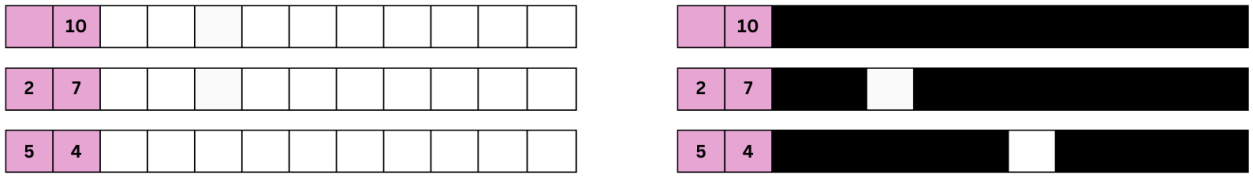


Figure 1.2: *Evident deduction examples*

1.2.2 Overlapping Method

By comparing all the possible placements of filled blocks, the solver tries to identify overlapping regions. For example, if we have a row with a clue of "7" in a row containing 10 cells, we can surely state that the central 4 cells will always be filled *see* ***Fig 1.3***.

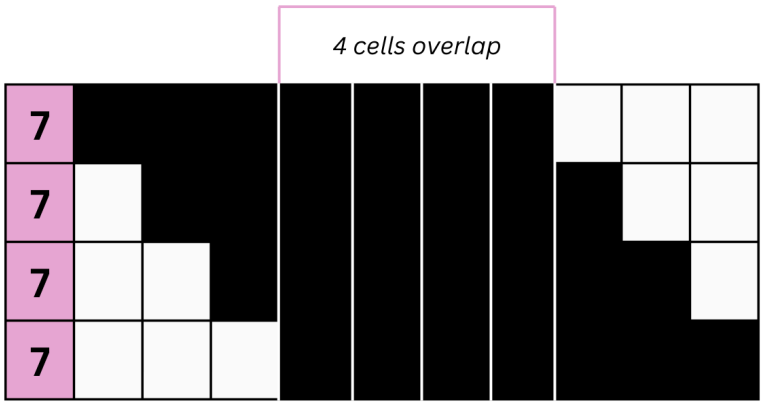


Figure 1.3: *Overlapping method, possible options example*

1.2.3 Gap Identification Method

Based on the given separation rules between blocks, solvers use mathematical reasoning to figure out some of the spaces that should not be filled so as to meet the requirements of the cell spacings. For better understanding *see* ***Fig 1.4*** and the explanation below.

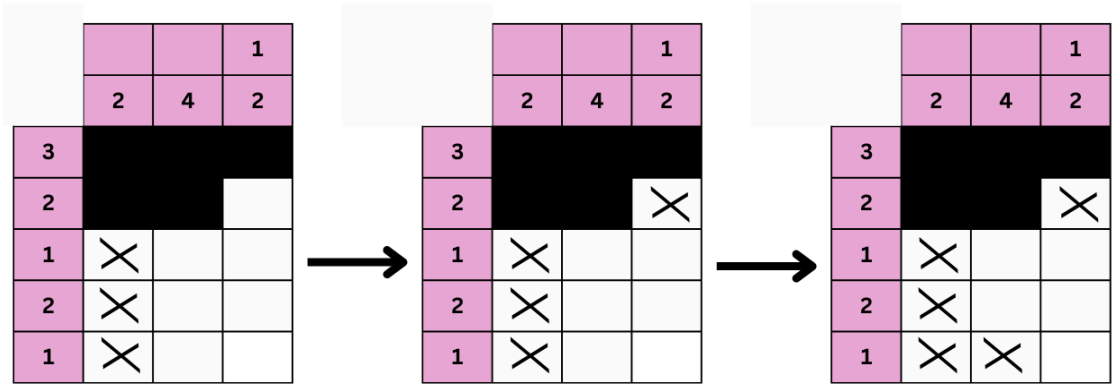


Figure 1.4: *Steps of Gap Identification method*

The first grid is the state of a grid, when some steps have been already performed by a solver. As we can see, the first 2 rows are filled and satisfy the row constraints, looking at the 1st column constraint of 2, the solver will exclude the last 3 cells of the first column, in other words he will identify the gaps of the 1st cell because it already satisfies to both (column and row) constraints. Similarly, other gaps can be identified during the solution steps.

Chapter 2

Literature Review

Based on research, there are several algorithmic approaches to solve black and white Nonogram puzzles including the methods, such as Depth-First Search (bruteforce) method [2], the Iterative one [3], a Genetic Algorithm [4], SAT approach [8]. In addition to these solving algorithms, Bosch's Integer Linear Programming (ILP) approach [6] offers an efficient formulation of the problem as a Constraint Satisfaction Problem (CSP).

2.1 Depth-First Search (brute force)

In this method, the state space encompasses all possible configurations of the puzzle, such as a grid where each cell can be black, white, or unknown. After filling out the cells based on the puzzle constraints, the transition model defines how to move between states. The algorithm explores every possible combination without imposing depth limits and backtracking when conflicts arise, because of which unnecessary states can be explored, making it inefficient for large problems. The goal state is reached when a configuration satisfies all constraints[2].

For example, for a black and white Nonogram Puzzle of size 6x7, for a line of size 7, there exist 6 possibilities for the blocks of size 3 and 1, *see* ***Fig 1.5.***

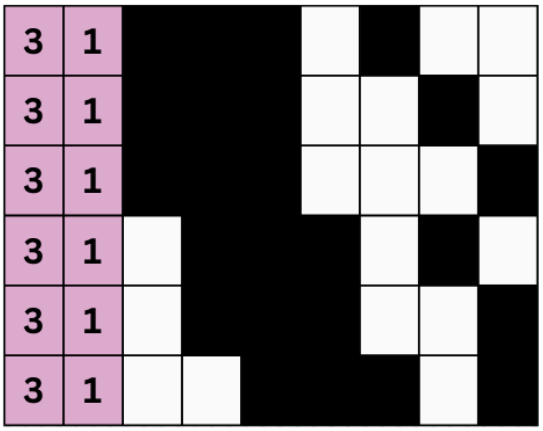


Figure 1.5: *Nonogram puzzle: Depth-First Search (bruteforce)*

2.2 Depth-First Search (Iterative)

An alternative way for solving black-and-white Nonogram problems is the Depth-First Search Iterative method. The state space and transition model are similar to those in the DFS brute force approach, but it is more systematic. It first explores shallow states before expanding deeper, so the algorithm focuses on finding shallower goal states sooner, ensuring memory efficiency. The process starts by solving a row, checking it against column hints, and if it is valid, then moving to the next row; if not, another solution is tried for the same row, with backtracking applied if no solution is found[3].

Below is a black-and-white Nonogram example of size 5x3. The initial state is shown in ***Fig. 1.6.a***. The process begins by filling cells from the leftmost cell, and when column constraints are violated ***(Fig. 1.6.c)***, filled cells are moved one step to the right ***(Fig. 1.6.d)***. This approach continues row by row until column hints are violated in the final row ***(Fig. 1.6.g)***, triggering backtracking. After further violations ***(Fig.1.6.h)***, the program repeatedly backtracks until it reaches the first row ***(Fig. 1.6.m)*** and resumes filling out the cells. Ultimately, the puzzle is solved ***(Fig. 1.6.p)***.

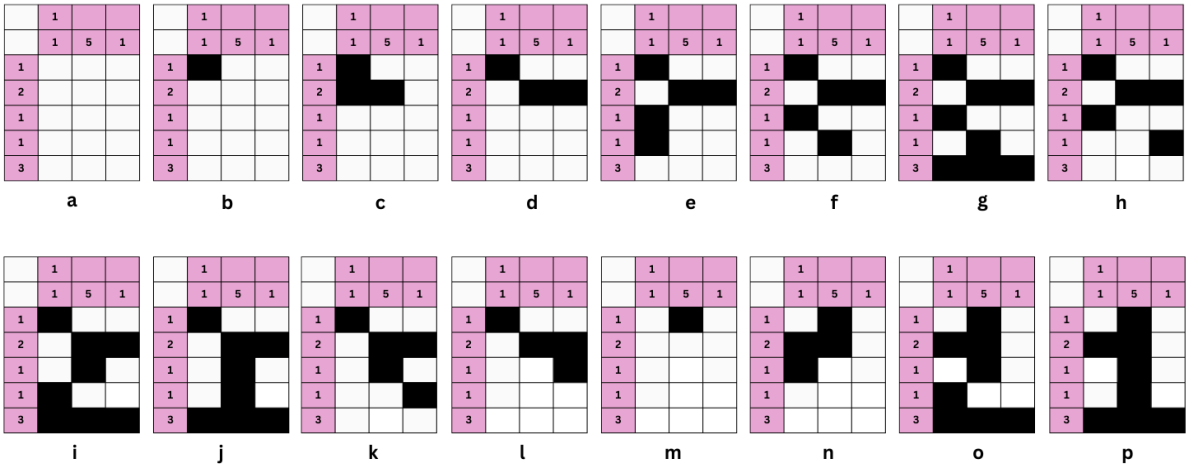


Figure 1.6: *Nonogram puzzle: Depth-First Search (Iterative)*

2.3 ILP by Bosch: *CSP Formulation*

This algorithm [6] concentrates on solving the nonogram not by filling in separate black/white values, but rather treating the group of a certain amount of black pixels as a whole and placing the resulting blocks in the grid. The problem can be divided into two subproblems: placing the row blocks and placing the column blocks. Terms blocks and clusters will be used interchangeably in this section.

Notation

Let

m = the number of rows,

n = the number of columns,

k_i^r = the number of clusters in row i,

k_j^c = the number of clusters in column j,

$s_{i,1}^r, s_{i,2}^r, \ldots, s_{i,k_i^r}^r$ = the cluster-size sequence for row i,

$s_{j,1}^c, s_{j,2}^c, \ldots, s_{j,k_j^c}^c$ = the cluster-size sequence for column j,

In addition, let

$e_{i,t}^r$ = the smallest value of j such that row i's t^{th} cluster can be placed in row i with its left-most pixel occupying pixel j,

$l_{i,t}^r$ = the largest value of j such that row i's t^{th} cluster can be placed in row i with its left-most pixel occupying pixel j

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

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

"e" stands for earliest and "l" stands for latest

Formulation of the set of **variables** includes three indicator variables:

1) Indicates whether the square in the grid is coloured black or white. For each $1 \le i \le m$ and $1 \le j \le n$, let

$$z_{i,j} = \begin{cases} 1 & \text{if row } i\text{'s } j^{\text{th}} \text{ pixel is painted black,} \\ 0 & \text{if row } i\text{'s } j^{\text{th}} \text{ pixel is painted white} \end{cases}$$

The next two variables concern the placement of blocks.

2) For each $1 \le i \le m$, $1 \le t \le k^r_i$, and $e^r_{i,t} \le j \le l^r_{i,t}$, let

$$y_{i,t,j} = \begin{cases} 1 & \text{if row } i\text{'s } t^{\text{th}} \text{ cluster is placed in row } i \text{ with its leftmost pixel occupying pixel } j, \\ 0 & \text{if not.} \end{cases}$$

3) For each $1 \le j \le n$, $1 \le t \le k^c_j$, and $e^c_{j,t} \le i \le l^c_{j,t}$, let

$$x_{j,t,i} = \begin{cases} 1 & \text{if column } j\text{'s } t^{\text{th}} \text{ cluster is placed in column } j \text{ with its topmost pixel occupying pixel } i, \\ 0 & \text{if not.} \end{cases}$$

After defining the variables, we need to impose some constraints on those variables, according to the conditions of our problem in order to convert this problem to CSP.

Constraints

We need to make sure that each block appears in that row exactly once, which means that it should have only one starting point. Since the variable is assigned value 1 only once in the row, the sum of indicator variables should be equal to 1.

$$\sum_{j=e^r_{i.t}}^{l^r_{i,t}} y_{i,t,j}=1.$$

Additionally, each next block in the same row should be placed to the right of the previous one. This means that i^{th} $(t + 1)^{th}$ block is placed to the right of i^{th} row's t^{th} block.

$$y_{i,t,j} \leq \sum_{j'=j+s^r_{i.t}+1}^{l^r_{i,t+1}} y_{i,t+1,j'}$$

$$\text{for each } e^r_{i,t} + 1 \leq j \leq l^r_{i,t}$$

Those constraints include only row blocks. Similar constraints are imposed on the columns of the grid. After imposing separate constraints on rows and columns of the grid, it is important to note that rows and columns, for which we solve the problem, are in the same grid and have common squares that need to be coloured with the same color. If the square in the position (i, j) is painted black, it means that either there is a row/column block starting at (i,j) or there is a row/column block, for which (i, j) is placed between its upper and lower bounds.

$$z_{i,j} \leq \sum_{t=1}^{k^r_i} \sum_{j'=\min\left\{e^r_{i,t}, j-s^r_{i,t}+1\right\}}^{\max\left\{l^r_{i,t}, j\right\}} y_{i,t,j'},$$

$$z_{i,j} \leq \sum_{t=1}^{k^c_j} \sum_{i'=\min\left\{e^c_{j,t}, i-s^c_{j,t}+1\right\}}^{\max\left\{l^c_{j,t}, i\right\}} x_{j,t,i'}$$

When placing the clusters, we also need to make sure that white squares are not covered by row/column blocks.

For each $1 \le i \le m$, each $1 \le j \le n$, each $1 \le t \le k^r_i$ and each j' that satisfies the inequalities $j - s^r_{i,t} + 1 \le j' \le j$ and $e^r_{i,t} \le j' \le l^r_{i,t}$ we impose $z_{i,j} \ge y_{i,t,j'}$.

And for each $1 \le i \le m$, $1 \le j \le n$, $1 \le t \le k^c_j$, and each that satisfies the inequalities $e^c_{j,t} \le i' \le l^c_{j,t}$ and $i - s^c_{j,t} + 1 \le i' \le i$, we impose $z_{i,j} \ge x_{j,t,i'}$.

2.4 Genetic Algorithm

The aim of this algorithm is to find the chromosome with the best fitness value by optimizing the fitness function. The algorithm works with the three main operators: selection, crossover, and mutation. These operators use the algorithm's datasets, which also are considered chromosomes. The initial step starts with having the populations of chromosomes containing the values 0 and 1, which corresponds to their fitness. In *Fig 1.7*, the value of the fitness function is indicated on the right side of the rows, which shows the count of ones in the chromosomes. The next step is the selection part, which is often done randomly, but with a bias for the chromosomes with higher fitness function. Afterwards, two chromosomes are selected for the upcoming steps. In the crossover process, genes up to the crossover point are exchanged between parents; in this case, values of the first two cells of the first chromosome are exchanged with the other chromosome's first two cells. Eventually, the mutation operator changes the value of a gene in one of the chromosomes shown on *Fig 1.7* below[4].

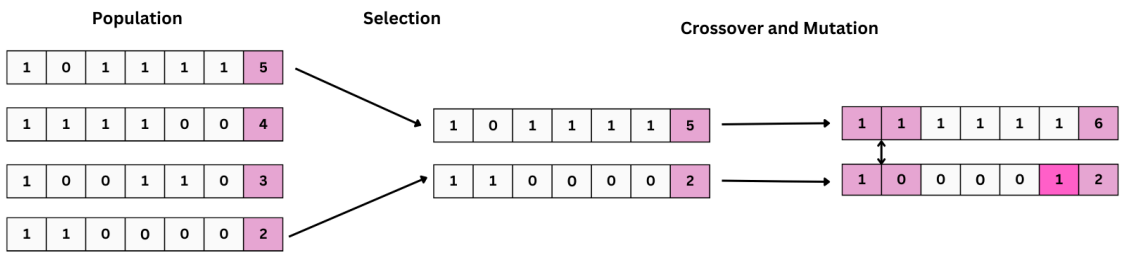
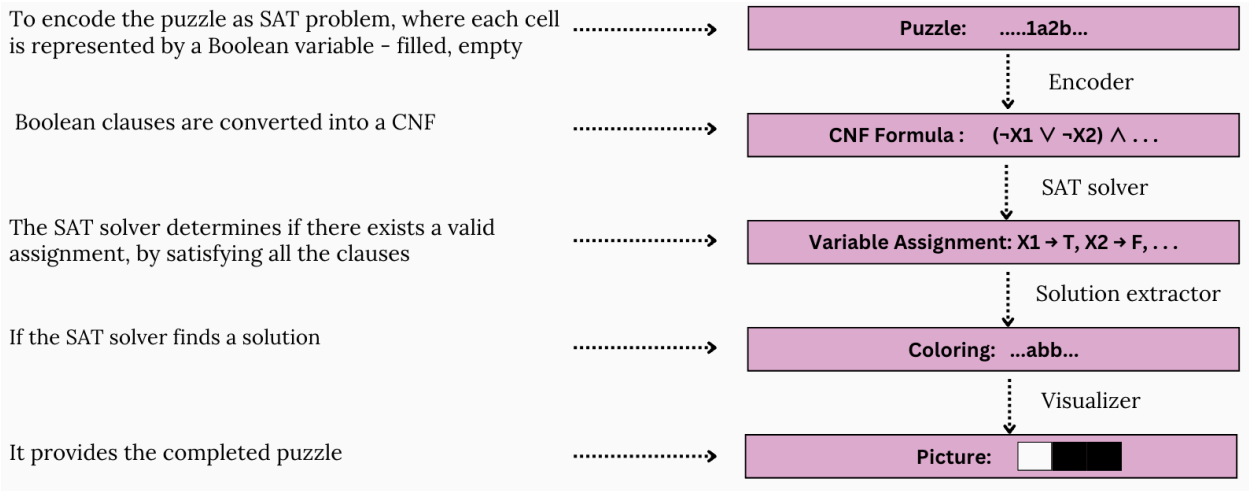


Figure 1.7: *Nonogram puzzle: Genetic Algorithm*

2.5 SAT approach

The SAT approach [8] is represented as a schema below.



This approach is considered a good method, especially for complex puzzles, as SAT solvers are highly optimized for constraint satisfaction problems.

Chapter 3

Method

3.1 Backtracking Search Algorithm

For this part we will try to solve our Japanese Crossword (Nonogram puzzle) with the CSP method; more specifically - *Backtracking Search Algorithm*. You can see the pseudo-code of the algorithm below.

```
function BACKTRACKING-SEARCH(csp)                    ▷ Returns a solution, or failure
  return BACKTRACK({}, csp)

function BACKTRACK(assignment, csp)                  ▷ Returns a solution, or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
    if value is consistent with assignment then
      add {var = value} to assignment
      inferences ← INFERENCE(csp, var, value)
      if inferences ≠ failure then
        add inferences to assignment and csp.
        result ← BACKTRACK(assignment, csp)
        if result ≠ failure then
          return result
        remove inferences from assignment and csp.
      remove {var = value} from assignment
  return failure
```

3.1.1 CSP formulation for a backtracking algorithm

- The ***variables*** should be the columns and rows: for example, for each m × n nonograms, there will be m + n variables as we have m rows and n columns.

- The ***domains*** of variables for our black and white puzzle would be a combination of white and black squares. Detailed explanation of the function for generating valid domains will be discussed later.

- The ***constraints*** for each line will be a list, meaning that if we have constraint of [2, 5] for the variable x_i , then we are supposed to have 2 black squares on that line followed by a gap with length ≥1 and then again 5 consecutive black squares. These are unary constraints however they will affect other lines as well. Hence, there is a binary constraint for the pairs of row lines and column lines, we have to make sure that there is a consistency between rows and columns. For instance,

  $r_1[1] = c_1[1], r_4[2] = c_2[4]$, for each $r_i[j] = r_j[i]$ where $1 \leq i \leq m$, $1 \leq j \leq n$.

Finally, we get the following problem setting:

X = $\{r_1, r_2, \dots, r_m, c_1, c_2, \dots, c_n\}$

D = $\{D_{r1}$ = {11.....1, 11...0, 11..01, 11..010...}, D_{rm} = {11.....1, 11...0, 11..01, 11..010...}, ... , D_{cn} = {11.....1, 11...0, 11..01, 11..010...}} where 1 is Black, 0 is White

C = {< r_1, [a, b, .. , c] >, ... < c_n, [d, e..., f] >, (unary constraints),

< (r_1, c_1), r_1[1] = c_1[1] >, ..., < (r_m, c_n), r_m[n] = c_n[m] > (binary constraint)}

Initially, for m rows the domain size would be 2^n (length = n) and for the n columns we would have a domain of size 2^m (length = m). However, in this way, if we had a line with length of 10, the domain size would be $2^{10} = 1024$. A better idea was to get the number of permutations of a fixed number of black and white cells. That is, if the constraint list is [2, 5], then the number of black cells is 7 and the number of white cells is 10-7 = 3. Using this way of generating domains would give us a domain size of 10 choose 3 , (10× 9×8)/3! = 120. But there was still room for improvement. Given the constraint [2, 5], we know that the first 2 black cells should be consecutive and the same applies for the last 5 cells. Thus, we could combine these 2 and (5+1) cells into 2 distinct blocks (because of the constraint that these two blocks should be connected with at least one white cell). As a result we would get 4 blocks, and decide the places of the remaining two white free cells. 4 choose 2, (4 × 3)/2=6. Thus, the domain size will be reduced from 1024 to 6.

Afterwards, we can choose the variables either randomly, or lexicographically, that is, c_1.... c_n, r_1, ..., r_n. Our algorithm will choose the variables with MRV (Minimum Remaining Value) heuristic. After selecting the variable, the

process will be repeated until we either get a complete consistent assignment or encounter inconsistency/empty domain for any variable.

For the given variable, when we assign a value such that it causes inconsistency in any two variables, then we will try another value for that variable, and if we see that none of them work, we will backtrack to the previous variable, changing the assigned value with another. So we can see that we are doing the same thing recursively and by that we are generating a tree. The worst case scenario would be finding the solution after generating the whole tree, whereas the best case scenario will be finding the solution from the very first try, that is, we never change the assigned value for any variable and we never backtrack. Time complexity depends on the size of the nonogram and the constraints, the tighter the constraints, the more pruning will happen and thus time complexity will be smaller. However in the best case scenario it would be just m+n and in the worst case scenario it would have to backtrack for each value of each variable and since we had m+n variables and for the time complexity b^{m+n} will be an upper bound, where be will be the domain size for the variable with largest domain.

3.2 Line Solver Algorithm

This method concentrates on finding partial solutions for different lines(rows or columns) one at a time. Each iteration of a line solver is guaranteed to find a consistent partial solution to the problem and eliminate some invalid cases for the remaining variables using arc-consistency. However, in some cases the line solver may get stuck, because of the existence of several solutions. At that point, we will use the backtracking search algorithm on the consistent partial solution line solver has found in order to move forward and achieve a complete solution at some point.

For line solver algorithms another CSP formulation should be introduced, which will allow to distinguish the colors of some cells, not necessarily requiring a complete solution for a line at a given time.

3.2.1 CSP formulation for a line solver

For this algorithm another CSP formulation should be introduced, which will allow to separately declare the colors of the grid cells, not necessarily requiring a complete solution for a line at a given time. For an n x m grid we need to have following sets of **variables**:

Variables

V = {$X_{1,1}, X_{1,2}$, ... , $X_{2,1}$, ... , $X_{m,n}$, R_1, ..., R_m, C_1, ..., C_n }

$X_{r,c}$ - a cell in the grid located at the intersection of row r (r < m) and column c (c < n)

R_r - row r (r < m)

C_c - column c (c < n)

Domains for these sets of variables are:

$X_{r,c}$ = {0,1} 0: cell is white, 1: cell is black (Domain size: 2)

R_r = {00..0, 00..01, ... , 11..11} (Domain size: 2^n)

C_c = {00..0, 00..01, ... , 11..11} (Domain size: 2^m**)**

For an m x n grid we have n*m cells ($X_{r,c}$), m rows (R_r) and n columns (C_c), with an overall number of variables being equal to n*m + n + m.

Constraints imposed on the variables are:

C = {< R_1, [a, b, .. , c] >, ... < C_n, [d, e..., f] >, (unary constraints),

$X_{r,c} = R_r[c]$, $X_{r,c} = C_c[r]$ }

MRV heuristic will be used for the line selection with a slight modification. The heuristic may repeatedly select the same line due to its small domain size, resulting in no further progress as all deductions from that line have already been made. This can lead to inefficiencies and prevent exploration of longer, more informative lines. To address this, a set of previously explored but unproductive lines is maintained and temporarily excluded from consideration. Upon achieving progress with any line, the set is cleared, as changes in the problem space may render previously uninformative lines valuable for further deductions.

3.2.2 Algorithm steps

The steps for solving a nonogram using line solver method are as follows:

1. Ensure node consistency for each line, that is, eliminate all the values in the domains that do not satisfy the variable's unary constraints. For this method the unary constraints of the problem include placing black blocks of a certain length listed in the clue list, so that there is at least one white cell separating them.

At this step the domain sizes of the variables decrease significantly making it computationally easier to process with the next steps.

2. Choose a column or a row with the defined heuristic.
3. For each cell variable $X_{r,c}$, which is in the constraint relation with the chosen line, that is, belongs to that line, we need to ensure arc consistency, e.g $X_{r,c}$ -> R_r or $X_{r,c}$ -> C_c.
4. If the domain of any cell variable is revised and thus turned into a singleton, the only remaining value in the domain is guaranteed to be a consistent solution for the variable. Consequently, it must be assigned to a variable.

5. For any assignment of the cell variable $X_{r,c}$ arc consistency R_r -> $X_{r,c}$ or C_c -> $X_{r,c}$ should be imposed, updating corresponding row or column variables' domains.
6. After several iterations, in the worst case when all the contained cell variables are assigned, the domains of line variables are expected to contain only one value. It is safe to assume that those values are the solution for that line, so it needs to be assigned to the corresponding row or column variable.
7. If all the cell variables are assigned, the algorithm should be stopped.

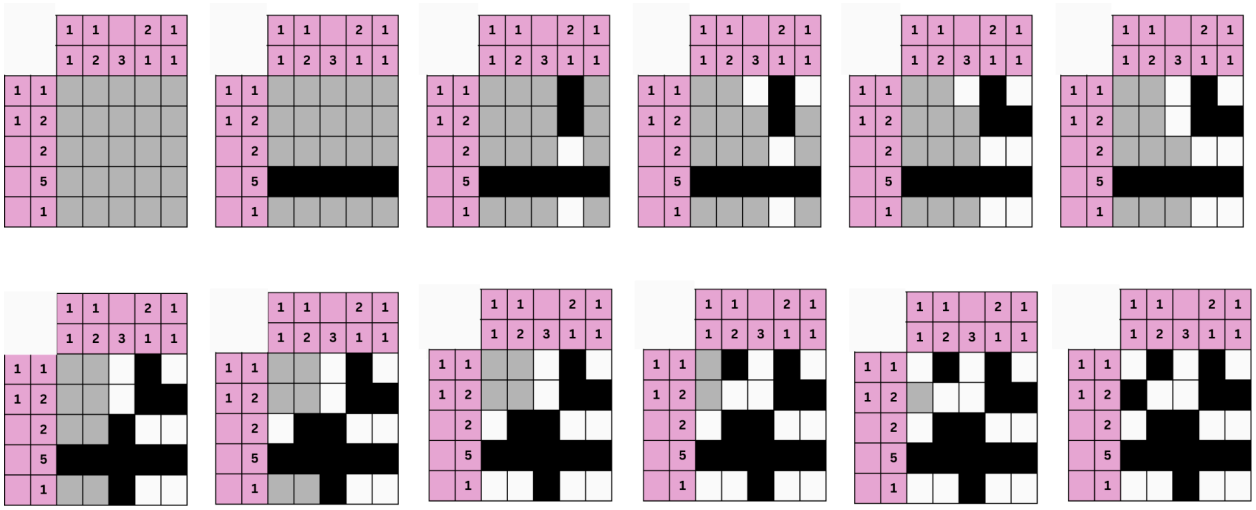


Figure 1.9: *Line Solver Steps / Example*

Fig. 1.9 illustrates how the line solver algorithm works, reaching the goal state in 11 iterations. During each iteration, it can be clearly seen what line has been chosen for exploration, reflecting the algorithm's structured approach.

Time complexity for the line solver algorithm tightly depends on the line constraints and how much they reduce the line domains. The best case will occur in case all line domains are singleton or become so during the exploration of other

lines. This means that the heuristic will always work with lines with a single value. The line heuristic prioritizes the rows over columns, thus it will always assign rows with singleton domains first, making the best-case time complexity O(m) for an mxn grid. Let's compute worst-case time complexity.

Generating all line sequences for each line and filtering:	$(m*2^m + n*2^n)^2$
Choosing a line (if the last column has the smallest domain):	$m+n$
Enforcing arc consistency for the cells of row/column:	$(\max\{m,n\})^2$
Enforcing arc consistency for the lines after assignment:	$(\max\{m,n\})^{\max\{m,n\}}$
Checking the stopping condition:	$m*n$

So, the upper bound for the algorithm's time complexity is

$(m*2^m + n*2^n)^2 + m+n + (\max\{m,n\})^2 + (\max\{m,n\})^{\max\{m,n\}} + m*n$

Line solver algorithm will always converge to the solution, provided that the constraints are defined correctly, that is, the problem has a unique solution. In any other case, the algorithm will not be able to move further with simply ensuring arc consistencies.

Chapter 4

Evaluation

4.1 Results and Conclusion

We have tested our implementations on 80 different problems having different difficulty levels. The summary results are shown below,see *Fig. 1.8*.

		Backtracking with MRV & FC		Line Solver	
level	quantity	mean	sd	mean	sd
easy peasy	19	0.071428571	0.032547514	0.002415207	0.00290165
easy	47	5.5246	16.63339415	0.3107192	0.6147679
medium	14	19.167	24.5651533	4.810475	4.03433092

dim	Backtracking with MRV & FC	Line Solver
10*22	couldn't solve	16 sec
25*21	couldn't solve	19 sec
28*27	couldn't solve	couldn't solve
20*17	couldn't solve	8.09 sec

Figure 1.8: *Algorithms Evaluations*

The level easy peasy represents the nonograms having size of $n \times m$ where $n < 10$, $m < 10$, easy - $11 < n < 20$, $11 < m < 20$, medium - $n > 20$, $m > 20$. In all of the cases, Line Solver performed significantly better on every nonogram. In case of easy peasy nonograms, it solved the problems on average 28 times faster. On easy nonograms, it was 18 times faster than Backtracking algorithm, and on medium problems it was still 4 times faster. Standard deviation of Backtracking with MRV & FC was very large compared to Line Solver's standard deviation, meaning that it had

dramatically different results and solving times were different. This is because of the domain sizes and the unary constraints. The stronger the constraint is, the smaller is the domain. However in many cases, when the size of the nonogram was slightly larger, the algorithm would try to generate all the valid domains for more than 30 minutes. These were the cases when the Backtracking algorithm never gave a final result. On the other hand, Line Solver handled these cases quite well. Though it took a long time to solve the problem, it indeed solved it.

4.2 Further Improvements

4.2.1 Line Solver Algorithm

1. As mentioned in *Sec. 3.2.2*, the algorithm is not able to handle problems with non-unique solutions. To overcome this, a backtracking algorithm can be implemented on a partial solution to help the line solver algorithm move further.
2. Another improvement can be suggested regarding the heuristic. MRV heuristic with the additions discussed works with the domain sizes only, not considering the line constraints themselves. By refining the heuristic, variables subject to tighter or more restrictive constraints can be prioritized, as they are more likely to significantly impact subsequent decisions.

4.2.2 Backtracking Search Algorithm

To use value ordering heuristic like Least Constraining Value (LCV) to choose variable values that start and/or end with the value 1.

Reference List

[1] Wu, I.-C., Sun, D.-J., Chen, L.-P., Chen, K.-Y., Kuo, C.-H., Kang, H.-H., & Lin, H.-H. (2013). An efficient approach to solving nonograms. *IEEE Transactions on Computational Intelligence and AI in Games*, 5(3), 251–264. https://doi.org/10.1109/TCIAIG.2013.2251884

[2] Zavistanavičius, R. (2012). *Nonogram solving algorithms analysis and implementation for augmented reality system* (Master's thesis). Kaunas University of Technology, Kaunas.

[3]Michael Ray(2018-2019.)*Depth first search algorithm for solving nonogram puzzles.* https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2018-2019/Makalah/Makalah-Stima-2019-001.pdf

[4]W. Wiggers(2004). A *comparison of a genetic algorithm and a depth first search algorithm applied to japanese nonograms*. Paper, Faculty of EECMS, University of Twente. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=7f10e79106bb147def51b03e5662be08490e15f8

[5]Daniel Berenda, Dolev Pomeranz , Ronen Rabani , Ben Raziel(2014) *Nonograms: Combinatorial questions and algorithms (p.3-4)* https://www.sciencedirect.com/science/article/pii/S0166218X14000080

[6] Hurkens, C. A. J., & Woeginger, G. J. (2001). Solution of a weighing problem. OPTIMA, 65, 16–17. Mathematical Programming Society.

[7] The Puzzle Museum, "Origins of cross reference grid & picture grid puzzles," 2012 [Online]. Available: puzzlemuseum.com/griddler/grid hist.htm

[8]Jan Frederik Schaefer(2024)."Assignment 3: Solve Nonograms AI-1 Systems Project (Winter Semester 2023/2024)".Friedrich-Alexander-Universität Erlangen-Nürnberg, Department Informatikhttps://kwarc.info/teaching/AISysProj/WS2324/assignment-1.3.pdf

[9] Chugunnyy, K.A. (KyberPrizrak). (n.d.). *Methods of solving Japanese crosswords*. Nonograms.org. Retrieved December 8, 2024, from https://www.nonograms.org/methods