**AI Final Project  
Due Color Nonogram Game**

[Ibraheem Asad](https://moodle2.cs.huji.ac.il/nu21/user/view.php?id=460465&course=67842) - [Ibraheem.Asad@mail.huji.ac.il](mailto:Ibraheem.Asad@mail.huji.ac.il)

[Mohamed Abu Shakra](https://moodle2.cs.huji.ac.il/nu21/user/view.php?id=605778&course=67842) - Mohamed.AbuShakra@mail.huji.ac.il

[Mohammad Ghanayem](mailto:Mohammad%20Ghanayem) - [Mohammad.Ghanayem@mail.huji.ac.il](mailto:Mohammad.Ghanayem@mail.huji.ac.il)  
Adam Sarsour – [adam.sarsour@mail.huji.ac.il](mailto:adam.sarsour@mail.huji.ac.il)

**Abstract**

The report's main objective is to compare search algorithms that we learned and discussed in AI on the Due-color Nonogram Game. After all the programming and testing with different methods, we found out that: [here we fill what we have found in the results]

1. **Introduction**

Nonogram game also Known as Hanji, Paint by Numbers, is a logical puzzle in which cells in a grid must be colored or left blank according to numbers at the side of the grid to reveal a hidden pixel art-like picture. These puzzles are often black and white but, in newer versions, it could be colored. where, the number clues are also colored to indicate the color of the squares. Two differently colored numbers may or may not have a space in between them.

Your aim in the game is to color the whole grid in black and white (or other colors, in the colored version of the game) squares. Beside each row of the grid are listed the lengths of the runs of black and colored squares on that row. Above each column are listed the lengths of the runs of black and colored squares in that column.

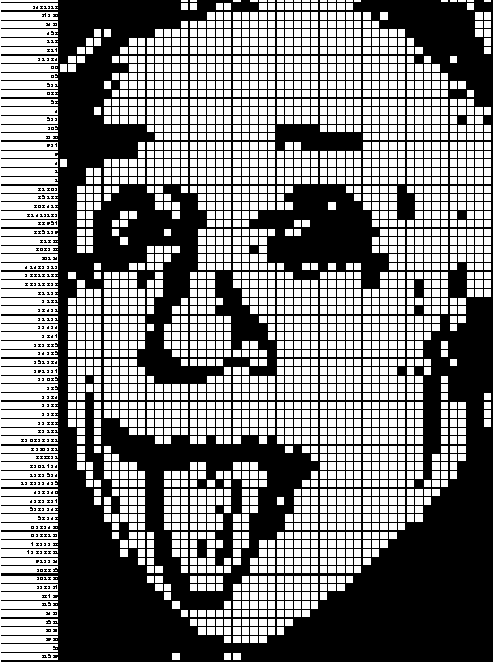
These numbers tell you the runs of black/colored squares in that row/column. So, if you see: '3 1 4', that tells you that there will be a run of exactly 3 red squares, followed by one or more white square, followed by a single red square, which may be followed by a white square or not, followed by 4 black squares. There may be more white squares before/after this sequence.

A picture containing text, crossword puzzle

Description automatically generatedTable

Description automatically generated

With all the information above, and after some searching, we could see that Nonogram considered as NP-Complete problem, which means that this problem cannot be completed in polynomial time.

Our research is about finding the best algorithm to solve a Due-color - (Red, Black) - Nonogram puzzle in **an optimal time**. For that purpose, we implemented searching algorithms such as: Backtracking (Brute Force), Depth-first search and Breadth-first search (BFS) and trying to implement heuristic algorithms such as: A\* as shortest path algorithm and Local-Beam search.   
this puzzle game is a great implementation as **Constraint satisfaction problems**, so we will also check five types of heuristics for the CSP: Minimum Remaining Values (MRV), Degree Heuristics, Least Constraining Value (LCV), Forward chaining (Checking), Arc Consistency.

1. **The goal of the game**

We need to fill all the cells in a given board, by following the constraint numbers that appears at the top of each column and at the left side of each row, with the least possible time following these simple rules:

1. we must fill the constraint numbers in the grid with same order appeared in the constraint.
2. between two constraint numbers with the same color must have at least one empty cell.
3. between two constraint numbers with different colors may or may not have one empty cell.
4. **Search Algorithms**

For the search algorithms we defined different problems to solve the game, we found the best way to solve the game is by checking all the cells one-by-one, so we used this method in the search algorithms. The other problems we checked: checking line by line each move (taking all combination for each line), another one by completing one constraint each move.

we will show you little information about search algorithms that we used:

1. ***Backtracking (Brute Force):***

This search algorithm solves a problem through exhaustion, it goes through all possible choices until a solution is found.

1. ***DFS:***This search algorithm starts at the root node and explores as far as possible along each branch before tracking, it works with the help of a stack, which is needed to keep track of the child nodes that were encountered but not yet explored. It works based on the principle of the last-in-first-out manner. The time and space complexity of the Depth-First Search algorithm is where b is the maximum branching factor, and m is the maximum depth of the state space.
2. ***BFS*:**This search algorithm explores all nodes at the present depth prior to moving on to the nodes at the next depth level. It works with the help of a queue, which is needed to keep track of the child nodes that were encountered but not yet explored. It works based on the principle of the first-in-first-out manner. The time and space complexity of the Breadth-First Search algorithm is where b is the maximum branching factor, d is the depth of the shallowest solution.
3. ***LBS*:**This search algorithm is a heuristic search algorithm that starts with k randomly states, at each level of the tree, it generates all successors of the states at the current level, sorting them in increasing order of heuristic cost, selecting the best k states in this level of successors and repeat until if any one of the successors is the goal. (Beam search is identical to BFS but with optimal memory, and may not find a solution even if BFS found one, it is related to the heuristic function)
4. **Constraint Satisfaction Problems**

the other thing we used was forming the game as a constraint satisfaction problem (CSP).  
the CSP is defined as a tripe , where:

* – is a set of variables.
* – is a set of variables’ respective domains of values
* – is a set of constraints between the variables.

A state of the problem is defined by an assignment of values to some or all the variables, an assignment that does not violate any constraints is called a consistent assignment, and a complete assignment is one in which every variable is mentioned, and a solution to a CSP is a complete assignment that satisfies all the constraints.

**Implementation of CSP**:  
in our game, we implemented the CSP like this: for the variables, we choose each row/column to be a variable. the domain for each variable, is the allowed values for that row/column following the rules of the game, and lastly the constraints, we choose the relation between row and every column to be the constraint.

1. **Heuristics:**

We used heuristics in two of the search algorithms, LBS and CSP.

1. ***LBS Heuristics:***in LBS algorithm, we chose our state to be the board, which we do it by choosing randomly k boards, that are also filled randomly (coloring them), and our heuristic would be to choose the board with the most filled cells, which we would be rated with lower cost, this would increase the chance to find a solution faster to our problem.
2. **CSP Heuristics:**
3. ***Minimum Remaining Values:***Choosing the variable with the fewest “legal” remaining values in its domain.
4. ***Degree Heuristics:***Choosing the variable that is involved in the largest number of constraints on other unassigned variables.
5. ***Least Constraining Value:***prefers the value that rules out the fewest choices for the neighboring variables in the constraint graph.
6. ***Forward Checking:***  
   detects the inconsistency earlier than simple backtracking and thus it allows branches of the search tree that will lead to failure to be pruned earlier than with simple backtracking.
7. ***Arc Consistency:***  
   Simplest form of propagation makes each arc consistent. is consistent iff for every value x of X there is some allowed y.
8. **Results**

We ran each one of the search methods and the CSPs on different boards, we tried to choose variety of boards to test and see what each method time at each setting, and to make our results more reliable, we used same board to each of the methods.  
I would like to mention that we differentiate between tests with different number of columns and rows, because in our methods, we relied more on of the sides more.  
the different settings we choose were:

1. ***Not Solvable Board:***in this test, just for fun, we decided to see what the fastest method is would tell us that there is no solution to a given board, and these are the results we got:

1. ***7x7 Board, Easy difficulty:***  
   in our game, we made two difficulties, the easy difficulty is by making the constraints of the rows or of the columns (the number on left or top of the board) big, which means we will fill cells with the same color continuously.  
   we believe Backtracking brute force method will do the best job here, since it will discover the mistakes on the go, and the results we got:
2. ***Small, medium, large, and EXTREME Boards, Hard difficulty:***the hard difficulty is by having random colors all over the board, which will make it more challenging to each method to solve it.  
   the small size we choose would be 5x5, medium size is 7x7, large board is 10x10, and the EXTREME board is 15x15.  
   we think that there won’t be that much of a difference between different methods at small and medium methods, and for large and EXTREME methods, CSP with arc consistency will do a magnificent work!, and the results:
3. ***Many rows and less columns, many columns and less rows, Hard difficulty:***as we mentioned above, that our approach to solve the board is favoring a side of the constraints (rows or columns), so we wanted to test boards where we have way more rows than columns and vice versa, which means we have 10x3 board, and 3x10 board. We suspect the one with small columns number will be faster using CSPs, because it will eliminate the columns and will choose with the remaining possibilities.  
   with no further a due, here are the results:
4. **Conclusion**

with these astonishing results we got, we can say about each algorithm that:

1. ***Backtracking (Brute Force):***we believed before that this is the lightest and easiest way to solve this board, since it makes sure each move is legit or not and continue. And by the results we say that it happened as we expected, or even better than other methods we thought will do well.
2. ***DFS:***it did a good job as expected, trying a cell then keep moving to check if it will work.
3. ***BFS:***the BFS was the worst case ever, since it checks if the goal states exist in the children first then move to next successors, and in this type of game, the goal state will be in one of the leaves.
4. ***Local Beam Search:***Everything is related to **k** (number of states we are checking every time)a higher chance of success when we have a larger k but takes more time.  
   Starting k-random states has lower success rate than stating with k-**clever** states (for example generating stats by making correct moves).
5. ***CSP:***

We can notice that the time to solve the board increased 0.01 seconds on average between easy and hard mode.and another thing to notice is that the solo heuristics (MRV, AC etc..) have the worst timing! (It increased X10 times), other than that, it's random who takes the top 5! but mostly a mix of 3 heuristics.

1. **References**

[1] Nonogram – Wikipedia

<https://en.wikipedia.org/wiki/Nonogram>

[2] Nonogram As NP-Complete Problem:  
  van Rijn, Jan N. (2012), [*Playing Games: The complexity of Klondike, Mahjong, Nonograms and Animal Chess*](http://www.liacs.nl/assets/2012-01JanvanRijn.pdf) *(PDF)*, master's thesis, Leiden Institute of Advanced Computer Science, Leiden University*, retrieved 2012-06-29.*

[3] How to play Nonogram:

<https://puzzlygame.com/pages/how_to_play_nonograms/>

[4] Constraints Satisfaction Problems:  
 Introduction to Artificial Intelligence - Unit 3   
 Constraint Satisfaction Problems Adversarial Search, Games Course 67842

[5] Constraints Satisfaction Problems in python:  
 <https://freecontent.manning.com/constraint-satisfaction-problems-in-python/>

[6] DFS, BFS and LBS:  
 Introduction to Artificial Intelligence - Unit 2  
 Informed Search Algorithms Course 67842