**AI Final Project  
Due Color Nonogram Game**

Adam Sarsour – [adam.sarsour@mail.huji.ac.il](mailto:adam.sarsour@mail.huji.ac.il)

[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)

**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**

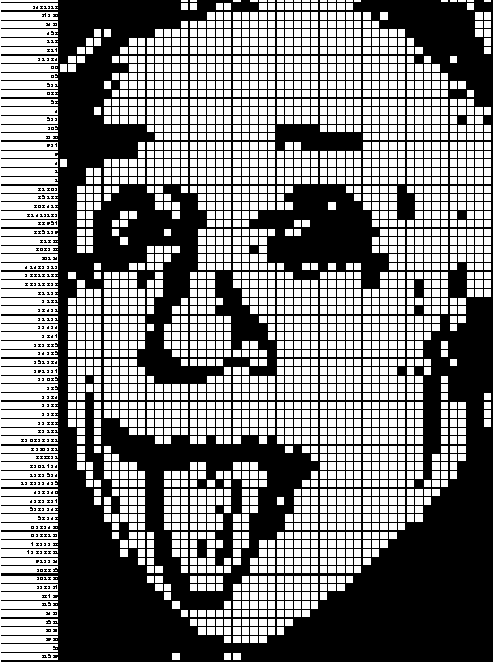
Nonogram game also Known as Hanji, Paint by Numbers, is a logical puzzle in which cells in a grid must be colored or left blank according to numbers at the side of the grid to reveal a hidden pixel art-like picture. These puzzles are often black and white but, in newer versions, it could be colored. 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.

[it is good here to add an example with empty board, then filled board (6x6) board or less]

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**

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. ***A\*: [I copied ameer’s A\* definition, Idk if we need to change it]***This search is an informed search algorithm. Starting from a specific starting node, it finds a path to the given goal node having the smallest cost. A\* selects the path that minimizes the following equation: 𝑓(𝑣) = 𝑔(𝑣) + ℎ(𝑣)   
   where 𝑔(𝑣) is the exact cost of the path from the start node to 𝑣, ℎ(𝑣) is a heuristic function which estimates the cost of the path from 𝑣 to the goal, and 𝑓(𝑣) is the estimated cost our path, from the starting node to the goal that contains 𝑣.
4. ***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)
5. **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.

There are different heuristics that we used in CSP that will help us solve the board faster than without these heuristics

1. ***Minimum Remaining Values:***

Choosing the variable with the fewest “legal” remaining values in its domain.

1. ***Degree Heuristics:***Choosing the variable that is involved in the largest number of constraints on other unassigned variables.
2. ***Least Constraining Value:***prefers the value that rules out the fewest choices for the neighboring variables in the constraint graph.
3. ***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.
4. ***Arc Consistency:***  
   Simplest form of propagation makes each arc consistent. is consistent iff for every value x of X there is some allowed y.

**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, A\* and LBS.

1. ***A\* Heuristics:***Choosing the variable that is involved in the largest number of constraints on other unassigned variables.
2. ***LBS Heuristics:***

**Results**

**Conclusion**