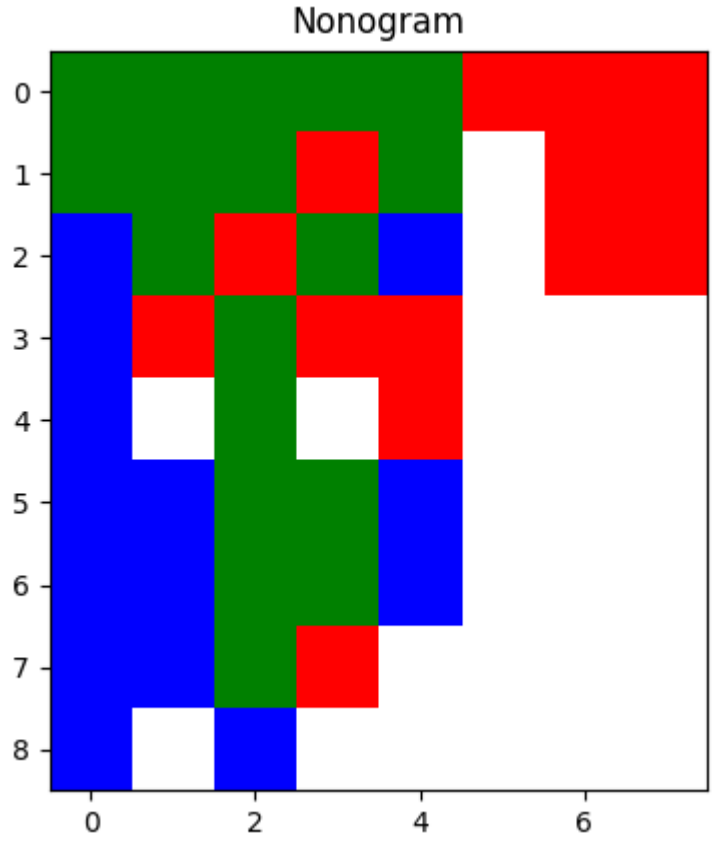
NTNU – Trondheim
Norwegian University of
Science and Technology

Artificial Intelligence programming IT3105

Report from Homework Module #2: “Combining Best-First Search and Constraint-Satisfaction to Solve Nonograms.”

By Jakob Svennevik Notland
Fall 2017



This is a report for the assignment in IT3105 where one shall combine best first search and constraint satisfaction to solve nonograms. Unfortunately I did not accomplish this task. Never the less, I will still create this report, and explain the representation and heuristic I managed to implement. I will also give a brief explanation of how I tried to solve the problem.

To run the program it must be executed with command line arguments. The syntax is as follows:
$ python3 gac.py <algorithm> <board> <display_solution> <display_agenda> <display_time>
Example:
$ python3 gac.py AStar monograms/mono-cat.txt no yes 1
Where the <algorithm> can be “Astar”, “DFS” or “BFS”. The <board> is a text file containing the board. <display_solution> and <display_agenda> are optional parameters which takes a boolean value. And has their default values as false. The last parameter <display_time> is a float where one can specify how many seconds to display each frame in display mode. Also the program will always print console output with the current node expanding, total nodes expanded and path length for the solution.

1. Representation

First I want to take a look at the picture on the front page. This is the print of the initial state for the “cat” nonogram. In my approach I first tried to create a whole solution, and then tweak it. The blue squares in the board represents segments that are only set in the row variables, the red ones are only set in the column variables and the green ones are values that are present in both a row and a column. I created this representation to give a rough representation of how far the current state is from the solution.

To represent the variables and domains in my “solution”, the program creates two matrices, one for the row variables, and one for the col variables. Each element in each row of the matrices represents a variable. The program always has the initial matrices containing all the variables with all the values in their domains. When a whole solution is created, the program picks one value for each variable. As seen on the front page. The initial whole solution just assigns the first value in each domain to its variable.

The constraints in my program are all stored in matrices, one for the rows and one for the columns. First they are created as strings, which are transformed into lambda functions.

2. Heuristic

For any state the heuristic will be the sum of all constraints violated.

3. My approach

When presented with this task of making a system to solve CSPs, I got very inspired by the approach where one guesses for a solution initially. And then tries to solve all conflicting variables. After a whole solution is made initially, I wanted my program to find the most conflicting variable. And then create all the children possible by changing this variable. A Star runs on top of this and the idea is that with the heuristics, the search program will find a solution. Which couldn’t be more wrong. I tried tweaking my code in many different ways to make this work somehow, but with no luck. I thought that I maybe were stuck in some local minimum, so I also tried to add some randomness. In the end I was just desperate for a solution, and did not care about the computational cost of the solution. Therefore the implementation I am delivering expands every variable that somehow has a conflict. Theoretically I would believe this can lead to a solution. But I was not able

to implement it correctly. The most frustrating thing about this project was that I always felt that I was getting closer to a solution. Unfortunately, when it was to late, I realized that I was just digging my own grave by trying to solve the problem with this approach. The day before the deadline I tried to implement a more traditional solution for this problem with use of domain reduction. However, I had to give up because my representation was not ideal for this approach, and my code had become to complicated at this point. So I could not figure out a way to solve it in so little time.