IT3105: AI Programming - Module 2
Nonograms

Lasse Henriksen
Ingunn Vallestad

September 2017

1 Introduction

The task was to create a Constraint Satisfaction Problem (CSP) solver, and combine this with the A* algorithm in order to solve nonogram puzzles. The complete algorithm is called A*-GAC.

2 Representation of nonograms as CSPs

For the representation of the problems as CSPs, the aggregate representation described in the assignment text was used.

- The **variables** of the CSP are the rows and columns of the nonogram.
- Their **domains** are all possible linear patterns that fill the row/column and satisfy the segment specifications.
- The **constraints** are based on the fact that each row and column pairing share a cell in the nonogram. The row and column patterns must agree on this value in order to solve the puzzle, therefore the constraint is that for each row and column, the following must hold: $row[columnIndex] = column[rowIndex]$.

The domains of the variables are filtered against each other in order to remove contradictory domain values. If all the domain values of a row have the same cell value at a certain index, the column that intersects the row at this index is required to share this cell value. Hence, all domain values of the column that does not meet this requirement can be removed from the domain.

3 Heuristics

3.1 A*'s traditional heuristic function

For our nonograms specific A* implementation, a "move" is considered removing one value from one variable's domain. When making an assumption for a value for a

variable, one value is chosen and the others are discarded. Hence, when generating successors and choosing one value for one variable the amount of values that was discarded from the variables domain is considered the amount of moves that was made by A*.

The heuristic evaluates each state of the CSP by looking at how many more values must be discarded from the variables before each variable only has one value left. This is the estimated cost/moves to the solution. With this definition the heuristic is in fact the true heuristic/cost to the solution because we know exactly how many more values that remain in each variables domain.

3.2 The heuristic in the choice of a variable on which to base the next assumption

There are two important alternatives for this heuristic presented in [1]: the minimum-remaining-values (MRV) heuristic and the degree heuristic.

For this problem, the degree heuristic - which selects the variable involved in the largest number of constraints - is pointless. In the chosen representation, all rows are involved in an equal amount of constraints (i.e. the number of columns), and equivalently, all columns are involved in a number of constraints equal to the number of rows.

The MRV heuristic chooses the variable with the fewest legal values, i.e. the smallest domain. This is an intuitive approach, since the chosen variable has few possible values, and therefore an assumption is more likely to be correct than if making assumptions on a large domain.

Something similar to the MRV heuristic is chosen for this problem. Instead of choosing the variable with the smallest domain, the heuristic chooses the variable with the smallest domain size larger than one. This is because when a variable's domain size is reduced to one, there is no need to make further assumptions - the only possible value for this variable is identified.

4 Brief overview of subclasses and methods needed to handle nonograms

NonogramCspNode

Represents the nonogram to be solved by the A*-GAC algorithm. The name contains the word *node* because each node in the A* search tree will be an instance of this class.

The class contains the representation of the nonogram (variables, domains and constraints) and functions needed to create the representation, and the variables and functions needed to specialize the A* algorithm.

ConstraintInstance

Contains the position of the cell involved in the constraint. The position is given as the row number and column number.

An important function is *ConstraintInstance.satisfied()* which compares the corresponding cell values of the row and column in the constraint. Returns *True* if they are equal, meaning that the constraint is satisfied.

VariableInstance

As mentioned earlier, a variable is a row/column in the nonogram. A *VariableInstance* is a specific row or column. It contains:

- Type - either row or column
- Number - the row or column position in the nonogram
- Domain - list of all possible configurations of segments for this variable

5 Important design decisions

As mentioned we used the aggregated representation described in the assignment text. Finding the full domain for each variable turned out to be the same as finding all restricted, weak compositions of a number [2] with the number equalling the number n of places any minimum length row/col compound could be placed, and number of parts k the amount of spaces to fill in before any segment plus one. Thus we knew that each domain had the size $\binom{n+k-1}{k-1}$.

This allowed us to solve all the nonograms that were given, except one, without ever needing to resort to searching with A*. By using an additional storage for the values generated, a hash-table (a python set), to keep track of which values that had been generated, the generation of values was very quick, even with variables that had tens of thousands of different possible values.

This however had one flaw, namely that (as warned) some variables had enormous domains. On testing the algorithm on a very difficult nonogram with dimensions 55x55, we found that certain variables had domains with over 100 million values, and storing those variables in RAM memory would take least double figure gigabytes. So while our algorithm easily solving small to medium sized nonograms, large and difficult nonograms faced a huge computational- and memory bottleneck.

References

[1] S. Russel and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Prentice Hall, 2010.

[2] Wikipedia, "Composition (combinatorics)," https://en.wikipedia.org/wiki/Composition_(combinatorics).