Vilde Schliekelmann Barth & Vilde Graven Stokke
IT3105 – Artificial Intelligence Programming

Module 2

Combining Best-1st Search and Constraint-Satisfaction to Solve Nonograms

Introduction

The purpose of Module 2 is to solve Nonogram puzzles using a combination of the A* algorithm and GAC. This module also uses the highly general A* , but by creating a subclass *Nonogram(Astar),* we've been able to derive problem specific heuristics and a check for a solution, *isSolution*.

Representations

Variables

We have defined each segment on the board as a variable. The variable is represented as a four-valued tuple, being *(row/col, index, segment number, length),* where:

- **row/col:** 0 if it is a row segment and 1 if it is a column segment
- **index**: represents the index of cell where the segment begins
- **segment number:** represents the order of the segment in the same row or column
- **length**: represent the length of the segment

Domains

The domain for each variable is the possible start indices for the given segment.

The total domains are represented as a dictionary with variables as keys and their respective domains as the associated dictionary value, that is:

$$Domains = \{variable : [domain\ values]\}$$

When a Nonogram-board is read from file and initialized, the domains are reduced to only include their allowed start values. This includes segment not being allowed to overlap or go outside the boundaries of the board.

Constraints

We have three types of constraints: *CellConstraints*, *RowColConstraints* and *BoardConstraint*. We have one superclass, *Constraint*, which all the constraints inherit from. All constraints are connected to the variables that are affected by them. This coupling happens in the initialization of the constraints.

All the constraints have a function *isValid* that returns True if the *domain-value* for the *variable* is valid in the current state, if not, it returns False.

CellConstraint

For each cell, there is a cell constraint which is connected to all the variables in the row and column that intersects the cell.

The *isValid*-function checks the following: it loops through all the variables with opposite orientation to the variable to be checked (that is coupled with the current cell). For each of these variables, it checks if any of them has a domain value that makes it possible for the current segment to intersect this cell. If any of opposite segments intersects the cell, the function returns True, else the value is invalid, and the function returns False.

RowColConstraint

For each row and column, there is a RowColConstraint. It is connected to all the segments in a row or column that have the same orientation.

The *isValid*-function checks if, with a given domain-value for the variable:

- the segment before (if there is any) has a value in its domain so that it leaves one cell blank in-between the two segments.
- the segment after (if there is any) has a value in its domain that is bigger than the value of the variable to be checked + its length + 1, and thus leaves one cell blank in-between.

If both are satisfied, the domain-value for the variable is valid and the function returns True.

BoardConstraint

We have created one constraint that concerns the board as a whole that is connected to all the variables. When the board constraint is initialized it calculates the legal sum of lengths of the segments in each row and column.

The *isValid*-function checks whether the current sum of lengths for a given column/row exceeds the legal sum for the given column/row. If it does, the current placement of segments is invalid.

Node

Each node represents a state in the search, and the variables assumed and the value assumed for each variable is unique for each node.

Each node has the following attributes:

- **parent**: node the current node was expanded from (which is the one with least cost - if the state is reached through various paths)
- **children**: all nodes created when the current node is expanded
- **h:** heuristic value for the node
- **g:** cost from root node to the current node
- **f:** sum of h and g
- **state:** represents if the current node has been expanded or not. True if not expanded and still in the *opened* queue, false if expanded and in the *closed* list
- **variable**: current variable to check
- **setVariables**: list of the variables with only one value in its domain

- **setValues**: domain-values chosen for the above setVariables

Heuristics

Our main heuristic is based on

$$f(node) = g(node) + h(node)$$

For the A*-GAC used to solve Nonograms, we have however two heuristic functions, the first being the heuristic used in A*. Our A* heuristic calculates the size of each domain, subtracts one and sums up all these values. This gives an admissible heuristic to estimate distance from the goal (being all domains having just one value), which gives us the number of domain-values we have yet to reduce. The heuristic can be written as:

$$h(node) = \sum(len(domain) - 1)$$

Our second heuristic selects a variable to make the next assumption for. This heuristic filters out variables with only one remaining domain-value, and finds the variable with the smallest domain to base the next assumption on. This heuristic is given by the formula:

$$next\_var \leftarrow \min_i len(var_i)$$

A*-GAC

Our general A* mentioned in Module 1 is also being used to solve the Nonogram puzzles along with a subclass for this specific problem called Nonogram(Astar). Here we define the problem specific heuristics and check for a solution. The heuristic is as mentioned above, and our goal test checks whether all domains have been reduced to only a singleton domain. If this is the case, the Nonogram is solved.

Our GAC is quite general and follows the provided pseudo-code. It's divided into three processes: initialization, domain-filtering loop and rerun. When reduction of the domains is no longer possible, the GAC will call upon A* to further assume domain-values and hence reduce the domains. The queue consists of pairs of variables and constraints for further reduction of domains.