Solving Puzzles with A*-GAC

Anders Sildnes, Andrej Leitner *students*

***Abstract*—This text answers assignment 3 and 4: Solving puzzles with our A*-GAC system solver.**

CSP solutions defines variables with domains. A valid solution is one where the domain of all variables has been reduced to the singleton domain. This occurs by making sure that a:) no constraints are violated and b) all variables are iterated over, and been locked to a single value in their domain.

Generality of A*-GAC Solver

The solvers for each puzzle is implemented using the same source code as in the previous project. *Essentially*, we use the exact same code for out CNET[1]. The changes we have made are:

1) Added methods "addCons(...)" and "addLambda(...)", which can take either a lambda, or a string that is parsable as a lambda, and translate that into a constraint. Example: addCons([1,2], "A <B") adds a constraint for vertex instance 1 and 2, and maps them onto A, B respectively (the first index maps to the first capital letter in the string).
2) Special case handling for when the domains consist of *sets* rather than single numbers, as used in the previous assignment.

Apart from item 1 and 2, this assignement is solved by subclassing the class "Problem" and implementing its methods[2] and also a parser for the input files.

[1]ConstraintNETwork, explained in project 1

[2]in "astar.py": triggerStart(), genNeighbour(), destructor(), updateStates()

Modeling Numberlink

Both problems could be fully expressed as SAT, so we found both problems to be NP-hard. We chose to use this representation for the flow puzzle. While the benchmark of Michael Spivey

[http://spivey.oriel.ox.ac.uk /corner/Programming_competition_results]

found that it can be faster to represent point in the graph by its incoming/outgoing direction of flow (NE, NW, NS, WE, WS ...), we chose SAT because of its simplicity and ease of implementing it in our model, and also with the knowledge that the input files were no bigger than 10×10, and that small differences in performance is not critical to our evaluation.

In our model, we represent the board as a 2D-grid of cells, each representing one vertex instance $v_{i,j}$ for poosition i, j in the cartesian plane. For each v, you can assign k colors, where k is given by the number of endpoints / 2. Each cell $v_{i,j}$ must be a part of the path [3] from an endpoint $e_{i,j}$ to another endpoint $e_{i+\alpha,j+\beta}$. Therefor our constraints are modelled as follows:

$$\forall i, j : leastTwoOf(v_{i+1,j}, v_{i+1,j}, v_{i+1,j}, v_{i+1,j},) == v_{i,j}$$

this can also be thought of as applying a 5-point stencil to the grid and validating each neighbour value

Modelling Nonograms

[3]I will assume the reader is has a mutual understanding of "path"

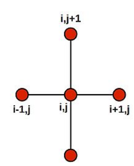

Fig. 1: Stencil operation