Practical 1: Modelling

Student ID: 120017875
Module Code: CS4402
Module Title: Constraint Programming
Lecturers: Ian Miguel, Peter Nightingale
March 5, 2016

Overview

This practical is centred around constraint modelling for M-queens puzzle. Firstly, two models corresponding to two different viewpoints of the problem are designed, and the design decisions are discussed in the first section of the report. These models are then compared to each other, and experiments are carried out with changing optimisation levels and trying out various heuristics to see how models perform under multiple different conditions. The results of these experiments are described in the second section. Finally, symmetry breaking constraints and constraints stating the lower bound of the number of queens necessary, and their according improvement on the model performance are discussed in section three.

Models of M-Queens puzzle

M-Queens puzzle is a variation of Queens Problems. Problems of this type ask you to find a way of placing a number of queens on a chess board so that certain conditions are fulfilled. In the case of M-Queens, there are two conditions - independence and dominance. Independence requires the queens to be placed in such a way that no two queens attack each other, while dominance requires all the cells of the chess board to either be attacked by some queen(s) or occupied by a queen.

I will now discuss two different viewpoints of M-Queens – two different ways of representing a chess board with queens on it, and the constraints placed upon them.

0/1 model (part 1)

This is a very intuitive way of representing a chess-board. It is modelled as a two-dimensional matrix, indexed from 1 up to M (where M is a parameter specifying the board size, and thus giving an instance of the M-Queens problem class). Each cell of the matrix is assigned either a value 0 or 1, where cells with 1 correspond to squares with queens on them, and cells with 0 correspond to empty squares.

The two constraints can then be checked accordingly:

- **Independence:** require every column, row and diagonal to have no more than one queen placed on it. This can be obtained by requiring the sum of elements of each column, row and diagonal to be less than or equal to 1. For rows and columns it is very straight forward – use the matrix slicing operation to loop through row and column slices, requesting the sum in each to be less than or equal to one. With diagonals things are slightly more complicated, as they are more difficult to express. For diagonals going "upwards" ("/"), I used the fact that for each element of the diagonal its row and column indexes sum up to the same number. To obtain diagonals going "downwards" ("\"), I created a formula that picks out the necessary elements. The idea remains the same – loop through diagonals, requiring the sum of each to be no more than 1.

- **Dominance**: each square is uniquely identified by a row and column number. Therefore I use a double loop, where outer loop goes through rows and inner through columns, to loo at all the squares and check for each individually whether it is under attack. For square to be under attack there must be a queen on either its row, column or one of the diagonals. Therefore for each square I require the sum of the elements of these to be bigger or equal to one (it is allowed for a square to be under attack by multiple queens or contain a queen, in which case the sum will equal 4).

My own model (part 2)

It is sensible to include both SR and Minion times in your report, since SR does perform useful transformations to improve your model, but at the cost of some time spent. Not only that, but the time spent in SR will vary according to the input model/instance, so this is definitely something to measure and discuss in your report.

As discussed in lectures, as well as time, you might think about reporting search nodes in your results. This will give you some indication of the size of the search Minion is doing relative to the modelling choices you have made. It can also be instructive if you have made what you thought was an improvement to your model but the solving process then takes longer. It might be the case that search is being saved, but the cost of propagating the extra constraints (for example) that you have added is taking too long - this can show up as a reduced node count but increased time taken.

Bibliography

[1] Author, *Book Title*, publisher, year