Foot Note 791 by Patrick Prosser
Limited Circulation[1] August 19, 2002

Propagation in the 4 Queens

Introduction

Have you ever run a demo of the 4 queens? Have you noticed that a queen is placed in the first column of the first row, then an attempt is made on the second row only for this to fail? Why is that? Why didn't search fail on the very first decison? I have noticed this on two different demos, one in the choco benchmark suite, the other in the Oz library. In answering this I hope to send out a cautionary message to those of us that actually write constraints, rather than just use those that are in the box.

The binary not equals to constraint

When we write a constraint in choco (and I imagine this will be similar in other langauges) we redefine some of the following methods: awakeOnInf, awakeOnSup, awakeOnInst, awakeOnRem. These methods handle propagation across a single constraint when the lower bound of a variable in the constraint increases, the upper bound decreases, a variable is instantiated, or a value is removed. The language is itself defined in terms of these methods.

In the binary *not equals to* constraint NotEqualxyc, i.e. $x \neq y + c$ where x and y are constrained integer variables and c is a constant, the awakeOnRem method is void, performing no activity. That is, if either x or y lose a value we cannot easily infer anything because the relation is not functional nor increasing nor decreasing.

Frequently the n queens constraint is defined between pairs of variables using the not equals to constraint. A constraint propagation event will only occur when a row loses an extreme position (left most or right most) or a queen is placed onto a row. In the majority of cases search will perform no better than forward checking.

The binary n queens constraint

When defining our own n queens constraint, we can define the awakeOnRem method in terms of the revise function. That is, when variable $v1$ in the constraint loses a value we can search for support for each value in the domain of $v2$. Any value in $v2$ without support is deleted and this in turn kicks off another call to awakeOnRem. Obviously the revise function is costly, $O(m^2)$ in its simplest form.

With such an encoding, in the 4 queens we find a solution with one node and one backtrack. That is, with only one decision we discover that there is no solution with a queen in the first colun of the first row. For the above encoding, using the not equals constraint, we find the first solution with 2 nodes and 2 backtracks.

Why is this interesting?

Clearly, if we are studying encodings and representations we have to know what's going on inside the propagation engine. We need to know just what happens when a value is removed, and if this action is the same for any value. And, when we write our own constraint we need to think about all these possible event. This note is interesting as a cautionary tale.

[1] I have decided to take my work underground, to stop it falling into the wrong hands.