

IAIP Mandatory assignment 3 – N-Queens

Project by: Allan Thræn (thraen@itu.dk), Thomas Gravgaard (fehaar@itu.dk) and Peter Thygesen (thygesen@itu.dk), Spring 2007.

Introduction

This project describes our solution to solving the n-Queens problem using BDD's to guide the user towards a solution.

Overview of the code

We have chosen to implement the project using C#. This caused a few concerns at start since the provided BDD library is written in java and it seemed too large and complicated to port to J# or C#. Luckily we managed to find a java byte code to .NET IL translator^[1] which enabled us to convert the java BDD library to a .Net BDD Library.

The entire code can be downloaded from <http://www.itu.dk/people/thygesen/NQueens.zip>

The provided .Net solution consists of three parts. NQueenslib project contains the BDD logic. The NQueensStub and NQueensweb project are two different user interfaces. The first is a console version and the other a web version.

Unit test are provided for some of the BDD logic and can be found as a sub folder in the NQueenslib project.

Rule building

In order to validate possible queen positions we need to build a BDD with binary rules.

It's common knowledge that in chess, a queen can move horizontally, vertically and diagonally. These rules should be broken down to one-to-one rules for all the fields on a chess board.

We will consider each field on the board a boolean variable, either being true or false, true if a queen is placed there, otherwise false.

To construct the BDD we will iterate through all the fields, and then add the rules that are relevant for them. This means that we should add the rules that disallow other fields on the same horizontal, vertical or diagonal lines to be turned on at the same time as the current field. Whenever a new rule is created it is AND'ed to the current BDD.

We also need to make rules that ensure that all the queens are placed. A simple way to do this is to make a rule for each horizontal line that ensures at least one queen in each line (the other rules will make sure that it's only one queen).

In order to demonstrate the method, here is a simple example.

Consider the 5*5 board:

```
  1 2 3 4 5
1 A B C D E
2 F G H I J
3 K L M N O
4 P Q R S T
5 U V X Y Z
```

We will now consider the rules we need to generate for the first field, A.

Horizontal:

$(!A \mid !B) \& (!A \mid !C) \& (!A \mid !D) \& (!A \mid !E)$

Vertical:

$(!A \mid !F) \& (!A \mid !K) \& (!A \mid !P) \& (!A \mid !U)$

Diagonal:

$(!A \mid !G) \& (!A \mid !M) \& (!A \mid !S) \& (!A \mid !Z)$

One queen per column:

Eventually we also need to add the rules that will ensure that there's one queen in each column:

$(A \mid F \mid K \mid P \mid U) \& (\dots) \& (E \mid J \mid O \mid T \mid Z)$

Variable Ordering

The ordering of the variables in a BDD is essential for the performance of the BDD. However there doesn't seem to be any obvious way to arrange the fields significantly better for this specific problem, so we decided to stay with the straightforward approach, ordering the variables from one end.

Assigning Queens and Projecting other configurations

When one or more queens has been placed by the user, they are added to the BDD, simply by setting their fields to true and AND'ing that with the BDD.

Then we iterate through the unassigned fields, trying to assign each of them and checking if the new BDD is satisfiable.

The User Interface

As briefly mentioned earlier, our solution contains two different user interfaces.

Console version

We did not use any of the provided GUI classes, obviously since we wrote the assignment in C#. However we have provided a small console application that serves the purpose of displaying and adding queens to the board. The application is quite self-explanatory. You set the size of the board and apply queens by entering coordinates. The program also provides a neat little feature for displaying the remaining solutions on the screen.

Web version

In order to try out the N-Queens solver with a more friendly User Interface, we constructed a simple web-interface.

First, we used the console application to generate a series of files with the basic BDD's for each of the N-Queens problems ($3 < n < 12$) so we wouldn't have to generate them on the fly whenever the web application is being called.

The web application then simply loads the selected BDD and add's the queens that the user currently have placed, investigates which other fields are invalid and marks them. It then displays the board using databinding.

It's also included in our solution as a web application project, but it can also be seen online at <http://www.mizar.dk/nqueens>.

References

[1] IKVM.NET – Java virtual machine in .Net and java translator <http://www.ikvm.net>