Intelligent Systems - BDD

Hansen, Daniel Rosenberg
daro@itu.dk

Nguyen, Dennis Thin Tan
dttn@itu.dk

Olesen, Thor Valentin
tvao@itu.dk

April 3, 2018

1 Goal of project

In this project, we are to use *binary decision diagrams* **(BDD)** to solve the n-queens problem. To solve this problem, a BDD library has been provided which allow one to create and manipulate BDDs that represents the n-queen problem. The product of this project is a piece of interactive software written in *Java* that enables a user to place n-queens on an *n*n* chessboard by highlighting which cells are placeable such that all n queens can be placed. Any other cells are marked illegal.

2 Using BDD

Before BDDs can be manipulated and used with the library, one must initialize a BDD factory. The BDD factory acts as a factory that produces the variables and constants as well manipulating BDDs. The library provides a method called *JFactory.init(nodes,cache)* that we can use to initialize the BDD factory. As recommended in the project description, the implemented BDD Factory is initialized with *2.000.000* nodes and a cache size of *200.000*.

Once the factory is defined, the number of variables to satisfy the n-queens problem must be set. In this case, the number of variables is *n*n*. That is, each cell represents a variable that can either be true or false, where the boolean state represents whether a queen can be placed or not.

Finally, the initial BDD can be produced which is just a BDD that is **True** since no rules have been applied yet. The reason for why the initial BDD has to be true is because when the rules are added to the BDD one needs to make a conjunction of the rule and the initial BDD. If it was initialized with false, then the conjunction will always be false regardless of any rules and therefore not satisfy the requirements of this project.

3 Our implementation

When the logic is initiated the BDD library is initialized using *initializeBDD();* and the rules for the board are created using *initializeRules();*

The rules have been divided into three different kinds of rules. Rules for horizontal (*createHorizontalRules*), vertical (*createVertialRules*) and diagonal cells (*createDiagonalRules*) in the board, tying the different cells together according to their placement thus creating the rules for the game. Similarities between the different methods are that the binary operations of the neighbors are connected with disjunctions where the different rows, columns etc are connected by conjunctions in the BDD creating a CNF.

TraverseDiagonal(int x, int y)
For the diagonal calculations, a helper function has been implemented to make the function more readable.

The *insertQueen* functionality is maintained by a function called *restrictBoard* taking a x and a y value and restricting the board from the placement. Thereafter the resulting board is calculated using the *isCellValid* to check what cells are now valid after the restriction.

Three helper functions for maintaining the different functions created for this project are:
atCell(int x, int y): returning the variables ID from its coordinates.
getVar(int cell): returning the BDD for the var of ID cell
negateVar(int cell): returning the BDD for the negated var of ID cell

4 Challenges

Some difficulties were encountered throughout the design of the implementation. The fact that the whole BDD should be a conjunction of disjunctions were initially hard to grasp initially the functions were tested using only conjunctions. This gave some incorrect results when computing the new board because all cells were suddenly false because one value was set to true. Initially, it was also thought that the BDD should include cases for when queens were not placed, this was later found to be wrong. Lastly, the best way to calculate diagonals were also difficult to resolve. An implementation using $O(N^4)$ were initially created but later discarded because a better solution was found.

5 Conclusion

In this project, a piece of software was implemented that allows users to solve the n-queens problems. The software uses BDDs to check if a given cell is valid to use such that the problem can be solved. The learning curve of using the BDD library was initially difficult but once figured out it was easy to use. The main challenges where rather on how to build the BDD such that all rules were satisfied. This was especially with the diagonal rule. Regardless, the project satisfies the project goals.