Assignment 1
ID2204 – Constraint Programming

Antonios Kouzoupis <890121-8837> – Lorenzo Corneo <890225-1290>
{antkou,corneo}@kth.se

April 11, 2015

1 Sudoku

For the Sudoku puzzle, initially we had to add a custom option handler. We did this to parameterize which puzzle from the examples would solve. The Script takes the command line options as argument and parses the puzzle to solve and the Integer Consistency Level. It also initializes a *IntVarArray* matrix with size 81 (9 * 9) and values from 1 to 9.

After that, it creates an integer array by flattening the sudoku example matrix. The Sudoku grid is represented by the *IntVarArray*, created before, and transformed into a Matrix object. The constraints we put are integer equality for every element in the sudoku example that is different from zero. Also, the numbers in each row, column and every 3x3 sub-matrix should be distinct.

For branching there was no strategy that outperformed an other one. We experimented with different parameters and we concluded that in most of the examples the strategy that performed better is INT_VAR_MIN_MAX for variable selection and INT_VAL_MED for value selection.

Similarly, there is no best option for the Consistency Level. The Domain propagation performed better than other levels in most of the given sudoku puzzles. But for instance, the Domain propagation for example 4 results in 4 node traversed and 1 failures whereas the Default level results in 23 nodes traversed and 10 failures. The Bounds propagation performs better than the default one and the Domain propagation by achieving 2 nodes traversed and 0 failures. On puzzle number 17 all Consistency levels traverse the same number of nodes and perform the same number of failures.

2 n-Queens

2.1 Constraints

The constraints that have to be applied to our model are essentially 3: row, column and diagonal constraints. In a matrix modelization of the chessboard for the n-queens problem, the management of the diagonal constraints resulted quite tricky. In fact, for each generic cell of the matrix we have to define a constraint.

Nevertheless, an optimization to reduce the number of constraints for the diagonal can be made if we propagate these constraints starting from the edges of the matrix (3 are enough). This because, if we defined a diagonal constraint for each cell we would obtain overlapping! For example, in a 4x4 matrix we define only main diagonal constraint for cells [0,0], [0,1], [0,2], [1,0], [2,0] while for the secondary diagonal constraints [0,1], [0,2], [0,3], [1,3], [2,3].

All the types of constraints are defined by the method `linear(*this, x, IRT_LQ, 1)` because each constraint has to be less equal than one (this guarantees only correct placements of queens).

2.2 Branching

We applied two strategies for the branching method: INT_VAL_MAX and IN_VAL_MIN. In the former case, the brancher finds firstly the solution that have as many as possible MAX_VAL (in our case 1) in the beginning of the matrix (first elements of the array). As a consequence we expect to find a 1 in the first position (if there is such a solution). In the latter case, we expect exactly the opposite so that we will find in the firsts positions the value zero.

2.3 Pros and cons

One of our favourite pros, but less relevant, is that the matrix is identical to the chessboard and for that it is immediate to understand. In the method presented during the lectures we had to put some more effort to figure out the representation of the solution.

In addition, on this model the definition of the constraints is less complicated because the matrix is the same model of the chessboard. Also this one is not a pros concerning performance or computation, but nice if you have to develop the solution.

Furthermore, in the given example the number of propagators is $3n^2$ while in our solution is $7n - 8$.

The only cons we have found is that the matrix uses more memory than the simple array. This result in a less efficient model from the point of view of the resources used.

3 Compile and Run

To compile the two programs type `make` in the main directory. To delete the object files type `make clean` and to delete both executable and object type `make dist-clean`.

In order to run the sudoku program, type `./sudoku [-puzzle NUM]` where NUM is the number of the sudoku puzzle in the given examples file. Default puzzle is the first one. Also it supports every option of the Options class.