

Sudoku

Constraints

The sudoku model uses $9 + 9 + 9$ distinct constraints:

$\forall r \in \text{rows} \quad \text{distinct}(r)$
 $\forall c \in \text{columns} \quad \text{distinct}(c)$
 $\forall s \in \text{3x3-squares} \quad \text{distinct}(s)$

Branching and Propagation

The solvability of this sudoku model depends a lot on the branching strategy and propagator strengths. For instance if we choose to go with a lower propagation constraint, which implies more search, the branching strategy is very essential. E.g when using `IPL_VAL` as propagation level for all constraints, if we choose first-fail branching heuristic the model is solved rather quickly ($< 1s$, 214 nodes, 103 failures for sudoku instance 0), however if we choose the last-fail branching heuristic the model is not solved even solved after $> 2min$ search. A very interesting note here is that if we choose a stronger propagation level, like `IPL_DOM` the branch strategy is of less importance since some sudoku instances can even be solved without any search at all by using strong propagation, e.g no matter the branching strategy sudoku instance 0 can be solved in the same time ($< 1s$, 1 node, 0 failures, 172 propagations) by using propagation level `IPL_DOM`.

Illustration of the difference in amount of search necessary:



Figure 1: Search tree for propagation level `IPL_DOM`

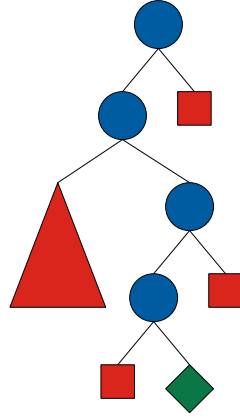


Figure 3: Search tree for propagation level IPL_BND and first-fail branch heuristic

To conclude, for this sudoku model stronger propagation is preferred, with IPL_DOM it is possible to achieve backtrack-free search and a solution in polynomial time.

n-Queens With 0/1 Variables

Modelling the problem as a $n \times n$ matrix using 0/1 variables means having one constraint per row, column, diagonal.

Formulation as a CSP

n-Queens, $n = 4$

$CSP = \langle \mathcal{V}, \mathcal{U}, \mathcal{C} \rangle$

Variables $\mathcal{V} = \{x_{1,1}, \dots, x_{n,n}\}$ ($x_{col,row}$)

Universe $\mathcal{U} = \{0, 1\}$

Constraints $\mathcal{C} = \{c_1, c_2, c_3, c_4, \dots, c_{18}\}$

Constraints for columns: $\{c_1, \dots, c_4\}$

Constraints for rows: $\{c_5, \dots, c_9\}$

Constraints for diagonals: $\{c_{10}, \dots, c_{18}\}$

Example constraint c_1 for column 1:

$\text{var}(c_1) = \langle x_{1,1}, x_{1,2}, x_{1,3}, x_{1,4} \rangle$

$\text{sol}(c_1) = \{ \langle 1, 0, 0, 0 \rangle, \langle 0, 1, 0, 0 \rangle, \langle 0, 0, 1, 0 \rangle, \langle 0, 0, 0, 1 \rangle \}$

Example constraint c_5 for row 1:

$\text{var}(c_5) = \langle x_{1,1}, x_{1,2}, x_{1,3}, x_{1,4} \rangle$

$\text{sol}(c_5) = \{ \langle 1, 0, 0, 0 \rangle, \langle 0, 1, 0, 0 \rangle, \langle 0, 0, 1, 0 \rangle, \langle 0, 0, 0, 1 \rangle \}$

Example constraint c_{10} for diagonal $x_{1,4} - x_{4,1}$:

$\text{var}(c_{10}) = \langle x_{1,4}, x_{2,3}, x_{3,2}, x_{4,1} \rangle$

$\text{sol}(c_{10}) = \{ \langle 1, 0, 0, 0 \rangle, \langle 0, 1, 0, 0 \rangle, \langle 0, 0, 1, 0 \rangle, \langle 0, 0, 0, 1 \rangle, \langle 0, 0, 0, 0 \rangle \}$

Model

Variables

Matrix<IntVarArray> of size $n \times n$

Constraints

The constraints can be implemented in Gecode by asserting that the sum of the rows and columns should be 1 (no two queens can share row/column) and that for each diagonal the sum can at most be 1 (there can only be at most one queen in each diagonal).

$$\forall c \in \text{Matrix.columns} \quad c.sum() \equiv 1$$

$$\forall r \in \text{Matrix.rows} \quad r.sum() \equiv 1$$

$$\forall d \in \text{Matrix.diagonals} \quad d.sum() \leq 1$$

Branching

What can you do about branching? Not that much as it seems (compared to other constraint problems). Since the variable domains are from the start very small $\{1, 2\}$, the effect of switching between for example INT_VAR_SIZE_MIN and INT_VAR_SIZE_MAX is not as evident compared to the queen-model with variable domains of larger size $\{1, \dots, 9\}$.

After doing some experiments mainly with INT_VAR_NONE, INT_VAR_SIZE_MIN, INT_VAR_RND, INT_VAL_SIZE_MAX, INT_VAL_SIZE_MIN. We have found that INT_VAR_SIZE_MIN (first-fail heuristic) was the most successful, choosing the value (0 or 1) heuristic was of minor importance regarding performance, but of course affected what type of solutions was found first.

The first-fail-heuristic outperformed INT_VAR_RND by a factor of ~ 5 and in fact INT_VAR_NONE and INT_VAR_SIZE_MIN gave similar results, fail-first slightly better only.

$n = 5$, first-fail vs random heuristic search trees:



Cons:

- The matrix model is less memory-efficient.
- A lot more variables.
- The example enforces one constraint just by modelling in a smart way: one queen per column constraint is enforced by the one-dimensional array. This is not utilised in the matrix-model.

How to run

Both Gecode models are subclasses of **Script** and you can run the from commandline with all options provided by default Gecode-script. For the sudoku model you can also specify which of the example-problems you want to solve by adding the flag: **-sudoku <num>**.

Examples:

Find all solutions to the 5-queens problem, not using GIST:

```
./queens -mode solution -solutions 0 10
```

```
.  
.
```

Queens Board Solution:

```
. . . . . Q  
. . . . . Q . .  
. . . . Q . . . .  
. . Q . . . . .  
Q . . . . . . .  
. . . . . Q . . .  
. Q . . . . . .  
. . . . . . Q .  
. . . . . Q . . .  
. . . Q . . . . .  
-----
```

Initial

```
propagators: 58  
branchers:   1
```

Summary

```
runtime:      0.088 (88.935 ms)  
solutions:    724  
propagations: 335188  
nodes:        11809  
failures:     5181  
restarts:     0  
no-goods:     0  
peak depth:   13
```

Solve sudoku example number 0 without gist and propagation-level: Domain-consistency:

```
./sudoku -sudoku 0 -mode solution -ipl dom
```

Sudoku

```
-----  
|3|7|8| |2|6|5| |9|1|4|  
|5|9|6| |8|1|4| |7|3|2|  
|1|4|2| |7|3|9| |5|6|8|  
  
|2|1|7| |3|8|6| |4|5|9|  
|8|5|4| |9|7|1| |6|2|3|  
|6|3|9| |5|4|2| |8|7|1|  
  
|7|8|5| |4|2|3| |1|9|6|  
|4|6|3| |1|9|7| |2|8|5|  
|9|2|1| |6|5|8| |3|4|7|  
-----
```

Initial

```
    propagators: 27  
    branchers:   1
```

Summary

```
    runtime:      0.000 (0.752 ms)  
    solutions:    1  
    propagations: 172  
    nodes:        1  
    failures:     0  
    restarts:     0  
    no-goods:     0  
    peak depth:   0
```