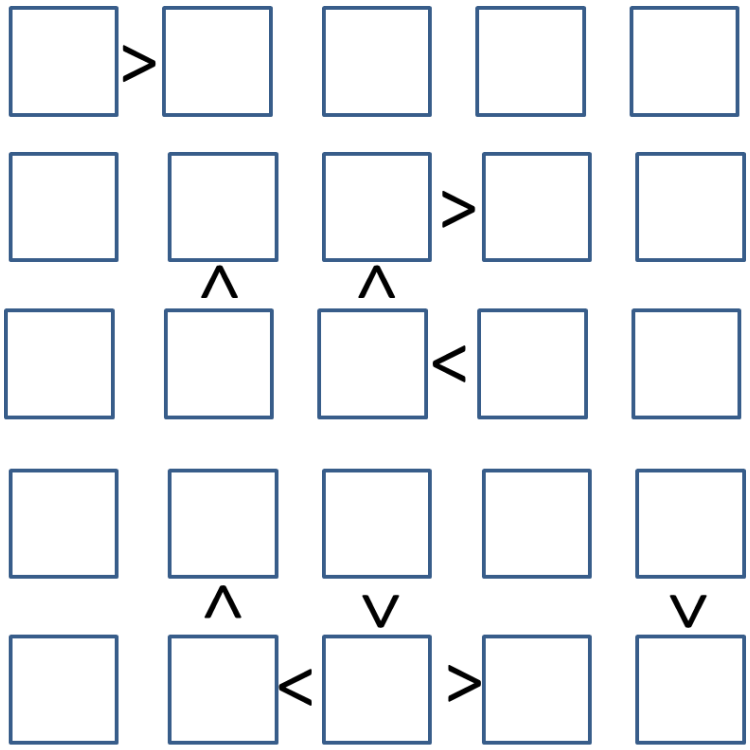
Futoshiki Puzzle

I saw the Futoshiki puzzle for the first time last month, reading the Gulf News while traveling in Abu Dhabi on business. The goal is to fill in the squares with the digits 1 thru 5, with each digit appearing only once in each row and column, while respecting the inequalities between certain cells. Some of them may have a digit or two filled in to start, but I particularly liked this one that I found online at http://www.futoshiki.org



Now, this puzzle takes just a few minutes to solve by hand, but then I got to wondering if I could do this via integer programming. I was chatting with Eli Olinick at SMU a couple of years ago and he mentioned that he liked to give the Sudoku puzzle as a question on the comprehensive exam for PhD students in Operations Research. The trick to solving Futoshiki, Sudoku and similar puzzles is handling the "all different" constraint. This type of constraint is really ugly in integer programming, but is handled well by constraint programming systems such as Ilog Solver.

The general setup for these puzzles is to use binary variables, where x[row,col,val] is equal to 1 if the cell row,col is equal to val.

$$\sum_{val} x_{row,col,val} = 1, \quad \forall row, col$$
$$\sum_{row} x_{row,col,val} = 1, \quad \forall col, val$$
$$\sum_{col} x_{row,col,val} = 1, \quad \forall row, val$$

Next, we need constraints to respect the inequalities given in the puzzle. Since we know that only one value can be switched on for a cell, the sum is equal to the value of the cell in the original puzzle.

$$\sum_{val1} x_{row1,col1,val1} * val1 \geq 1 + \sum_{val2} x_{row1,col1,val2} * val2$$

If the puzzle already has some of the squares filled in, then these would simply be equality constraints for a couple of the variables. In the puzzle above, there are none of these constraints.

The solver I chose to use is Gurobi, and programming it via the Python interface. It's a one-page problem, about 60 lines of code to setup the data, solve the puzzle and print the solution. Now, the interesting part is that the presolve routines completely solve the puzzle, so the optimizer performs no simplex iterations at all. Pretty neat.

```
Optimize a model with 85 rows, 125 columns and 475 nonzeros
Presolve removed 85 rows and 125 columns
Presolve time: 0.00s

Explored 0 nodes (0 simplex iterations) in 0.00 seconds
Thread count was 1 (of 4 available processors)

Optimal solution found (tolerance 1.00e-04)
Best objective 0.000000000000e+00, best bound 0.000000000000e+00, gap 0.0%

4 2 1 5 3

3 4 2 1 5

1 5 3 4 2

2 1 5 3 4

5 3 4 2 1
```