Solving sudoku as an Integer Programming problem

- A standard way to solve sudoku is by applying recursion, an algorithm where the solution depends on solutions to smaller instances of the reference problem. One checks all the combinations (for every cell it considers all integers between 1 and 9) and stops when finding a solution which satisfies the constraints.

- Alternatively we can apply Integer Programming optimization. Because in fact there is no function we want to maximize, the sudoku problem can be called a *satisfiability* or *feasibility* problem.

- The requirements are to include in each row, column and 3×3 square (out of the nine adjacent ones) all the integers from 1 to 9. Known cells can be treated as additional constraints.

The 1st equation below corresponds to the constraint on columns, the 2nd one refers to the constraint on rows and the 3rd one to the constraint on the 3×3 squares. x_{ijk} assumes the value of 1, if element (i, j) of the sudoku matrix contains k, and 0 otherwise.

$$\sum_{i=1}^{9} x_{ijk} = 1 \qquad \text{for} \quad j, k = 1 \text{ to } 9$$

$$\sum_{j=1}^{9} x_{ijk} = 1 \qquad \text{for} \quad i, k = 1 \text{ to } 9$$

$$\sum_{j=3p-2}^{3p} \sum_{i=3q-2}^{3q} x_{ijk} = 1 \qquad \text{for} \quad k = 1 \text{ to } 9 \text{ and } p, q = 1 \text{ to } 3$$

$$\sum_{k=1}^{n} x_{ijk} = 1 \qquad \text{for} \quad i, j = 1 \text{ to } 9$$

$$x_{ijk} = 1 \qquad \forall\, (i, j, k) \in G \mathrel{\widehat{=}} \text{all the known cells}$$

- The 4th equation assures that every position in the sudoku matrix is filled. If we were interested in finding any solution for an empty grid these four equations would suffice.

- However, if we are interested in finding a solution to a grid with some numbers already on it, we additionally have to consider the 5th constraint, where the corresponding x_{ijk} are assumed to equal one.

- The five constraints can be translated to AMPL in the following way:

```
set N := 1..9;
set DATA within {N cross N cross N};
var x {(i,j,k) in {N cross N cross N}} binary;

minimize nothing:   x[1,1,1];

subject to Columns {j in N, k in N}:
     sum {i in N} x[i,j,k] = 1;

subject to Rows {i in N, k in N}:
     sum {j in N} x[i,j,k] = 1;

subject to Squares {k in N, p in 1..3, q in 1..3}:
     sum {j in (3*p-2)..(3*p)} sum {i in (3*q-2)..(3*q)} x[
       i,j,k] = 1;

subject to all_filled {i in N, j in N}:
     sum {k in N} x[i,j,k] = 1;

subject to known {(i,j,k) in DATA}:
     x[i,j,k] = 1;
```

There are both sudokus with one solution and with multiple ones. Consider the following example, for which we know there are exactly five possible ways to solve. Using the above AMPL model code, we are going to find a possible solution.

8			6			9		5
				2		3	1	
		7	3	1	8		6	
2	4						7	3
		2	7	9		1		
5				8			3	6
		3						

- For AMPL to read the known cells we can define the following data code. Set DATA consists of 25 triples representing the known cells.
- For a triple the third number should stay in the row represented with the first number and the column represented with the second one.
- Please note that we are not restricted to the size of the set DATA.

```
###########     DATA CODE    ###########################

set DATA:=
    (1, 1, 8) (1, 4, 6) (1, 7, 9) (1, 9, 5) (3, 5, 2)
    (3, 7, 3) (3, 8, 1) (4, 3, 7) (4, 4, 3) (4, 5, 1)
    (4, 6, 8) (4, 8, 6) (5, 1, 2) (5, 2, 4) (5, 8, 7)
    (5, 9, 3) (7, 3, 2) (7, 4, 7) (7, 5, 9) (7, 7, 1)
    (8, 1, 5) (8, 5, 8) (8, 8, 3) (8, 9, 6) (9, 3, 3);
```

- The model itself generates $9 \times 9 \times 9 = 729$ variables and $4 \times 81 + 25 = 349$ constraints, which are above the limits of 300 in the student's version of AMPL.

- Because of the known 25 numbers on the sudoku grid `presolve` returns a problem with significantly less variables and constraints, so that the size constraints of our sudoku are not violated.

- Otherwise, use the Network–Enabled–Optimization–Software!

- Now your task is to write an AMPL `.run` file, where you specify the solver to be used, read the data and model codes, which have been just discussed, and return the filled sudoku grid.

- What happens if the `binary` constraint is replaced with `integer`?

- What if we consider the problem as a pure LP problem?

As already said, to the given sudoku there are five possible solutions. The AMPL's choice is based on the objective function, the used solver, as well as on the order of the constraints and possible starting points, which can be additionally specified.

8	3	4	6	7	1	9	2	5
1	2	5	8	3	9	6	4	7
7	9	6	5	2	4	3	1	8
9	5	7	3	1	8	4	6	2
2	4	1	9	5	6	8	7	3
3	6	8	2	4	7	5	9	1
6	8	2	7	9	3	1	5	4
5	1	9	4	8	2	7	3	6
4	7	3	1	6	5	2	8	9