Sudoku(https://en.wikipedia.org/wiki/Sudoku) je jednoduchá algoritmicky rozhodnutelná úloha. Cílem semestrální práce bylo to ukázat s využitím dvou různých přístupů a porovnat jejích efektivitu.

1) Intuitivně lze políčka chápat jako proměnné, které nabývají hodnot 1 až 9. Pak řešení instance sudoku lze chápat jako takové ohodnocení proměnných, které splňuje pravidla hry, což véde na formulaci úlohy jako CSP(https://en.wikipedia.org/wiki/Constraint_satisfaction_problem). Formálně:

$X = \left\{ x_{i,j} \middle| i, j \in \{1, \dots, 9\} \right\}$ jsou proměnné, kde *x,y* označují pozice;

$D = \{ 1, \dots, 9 \}$ je doména;

$C = \{ allDifferent(x_{i,1}, x_{i,2}, \dots, x_{i,9}) \mid i = 1,2, \dots, 9;$

$allDifferent(x_{1,j}, x_{2,j}, \dots, x_{9,j}) \mid j = 1,2, \dots, 9;$

$allDifferent(x_{3*k+1,3*l+1}, x_{3*k+1,3*l+2}, x_{3*k+1,3*l+3},$

$x_{3*k+2,3*l+1}, x_{3*k+2,3*l+2}, x_{3*k+2,3*l+3},$

$x_{3*k+3,3*l+1}, x_{3*k+3,3*l+2}, x_{3*k+3,3*l+3}) \mid k, l = 0,1,2 \}\}$

jsou omezení.

Takový problém pak lze vyřešit například pomocí backtrackingu(https://en.wikipedia.org/wiki/Backtracking)

Implementace:

Třída Sudoku obsahuje metodu solve_as_csp(), která implementuje backtracking pomocí rekurze(DFS). *allDifferent* je naimplementována tak, že porovnává velikost vstupního pole s množinou vytvořenou z prvků tohoto pole.

2) Myšlenku z minulé metody rozšíříme tak, že proměnné jsou $\left\{ a_{x,y,z} \middle| x, y \in \{1, \dots, 9\}, z \in \{1, \dots, 9\} \right\}$, kde *x,y* označují pozice, *z* označuje číslici. Takové proměnné nabývají dvou hodnot : True, False. Například „ $a_{0,0,3}$:= True” znamená, že políčko v horním levém rohu ma hodnotu „3”. Evidentně takové proměnné tvoří výrokové proměnné. Nabízí se úlohu převést na výrokovou formuli a vyřešit jako SAT problem(https://en.wikipedia.org/wiki/Boolean_satisfiability_problem). Klauzule formule v KNT pak vypadají následovně:

- žádné políčko není prázdné:

$$\bigwedge_{x=1}^{9} \bigwedge_{y=1}^{9} \bigvee_{z=1}^{9} a_{xyz}$$

- všechny řádky obsahují navzájem různé číslice:

$$\bigwedge_{y=1}^{9} \bigwedge_{z=1}^{9} \bigwedge_{x=1}^{8} \bigwedge_{i=x+1}^{9} \neg a_{xyz} \vee \neg a_{iyz}$$

- všechny sloupce obsahují navzájem různé číslice:

$$\bigwedge_{x=1}^{9} \bigwedge_{z=1}^{9} \bigwedge_{y=1}^{8} \bigwedge_{i=y+1}^{9} \neg a_{xyz} \vee \neg a_{xiz}$$

- 3x3 bloky obsahují navzájem různé číslice:

$$\bigwedge_{z=1}^{9} \bigwedge_{i=0}^{2} \bigwedge_{j=0}^{2} \bigwedge_{x=1}^{3} \bigwedge_{y=1}^{3} \bigwedge_{k=y+1}^{3} \neg a_{(3i+x)(3j+y)z} \vee \neg a_{(3i+x)(3j+k)z}$$

$$\bigwedge_{z=1}^{9}\bigwedge_{i=0}^{2}\bigwedge_{j=0}^{2}\bigwedge_{x=1}^{3}\bigwedge_{y=1}^{3}\bigwedge_{k=x+1}^{3}\bigwedge_{l=1}^{3} \neg a_{(3i+x)(3j+y)z} \vee \neg a_{(3i+k)(3j+l)z}$$

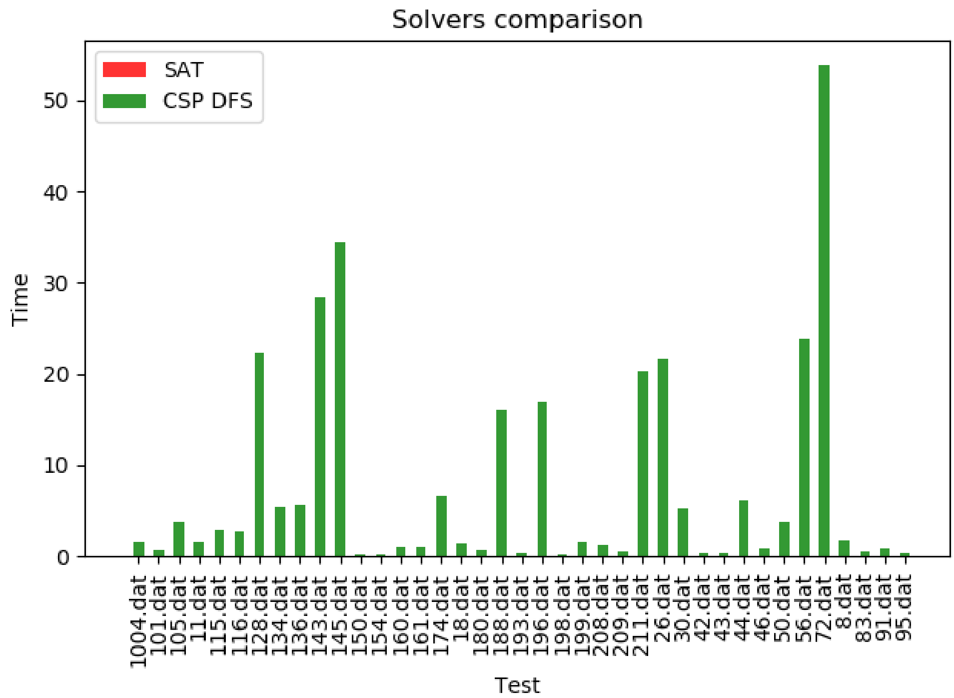
- počáteční podminky():

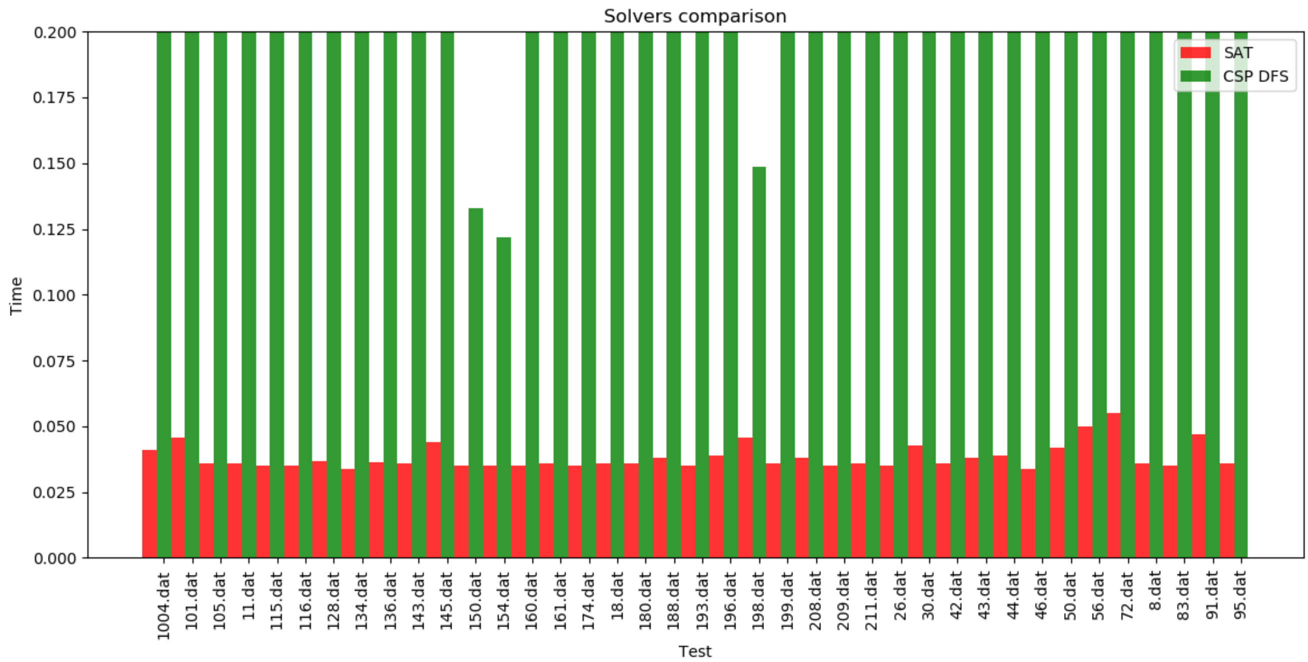
$$\bigwedge_{a_{xyz}\in P} a_{xyz} \quad , kde\ P = \left\{ a_{x,y,z} \middle| \ v\ počátečním\ stavu\ sudoku\ na\ pozici\ x, y\ je\ číslice\ z \right\}$$

Implementace:

Třída Sudoku obsahuje metodu solve_as_sat(), která implementuje redukci podle předpisu popsaného výše. Formuli pak řeší pomocí solveru Glucose3.

Porovnání





Výsledky jsou očekavané. Na jednoduchých příkladéch jsou obě metody skoro stejně efektivní. S rostem složitosi úloh roste i čas potřebný CSP solveru, protože prohledává více kandidatů a “bloudí“ stavovým prostorem pro nalezení řešení. SAT solveru je naopak “jedno”, jak složita úloha je, protože formule obsahuje v každém případě skoro ty samé klauzule(líší se pouze počáteční podminky). Díky tomu je druhý přístup obecně efektivnější.