# Protokol k semestrální práci z BI-ZUM

### FIT ČVUT, LS 2022/23

**Jméno studenta**: Jakub Mikeš

**Username**: mikesj17

Název semestrální práce: **Sudokář - klikař**

---

## 1. Zadání semestrální práce.

Mým zadáním je pokusit se pomocí algoritmu hill climbing vyřešit sudoku. Následně vymyslet různé způsoby, jak měnit kandidáta a porovnat je.

## 2. Stručný rozbor, analýza zadání.

Jako téma semestrální práce jsem si vybral jedno z ukázkových témat a to Sudoku klikař, protože kombinuje logické uvažování, programování a aplikace matematických pravidel a algoritmů.

Cílem řešiče je pokusit se o nalezení řešení zadané sudoku. Hill climbing algoritmus pracuje tak, že neustále hledá lepší a lepší řešení daného problému. Algoritmus začíná s náhodně vygenerovaným řešením a pak postupně prochází sousední řešení, tedy řešení, která se liší od aktuálního řešení pouze jednou změnou (například změnou jednoho čísla v Sudoku řádce).

Existují však určité problémy s použitím hill climbing algoritmu pro řešení Sudoku. Jedním z problémů je nedostatek diverzity řešení - hill climbing algoritmus může vést k tomu, že se řešení neustále opakuje, což může vést k nedostatku různorodosti řešení. Toto ve své implementaci řeším ukládáním si již vyzkoušených řešení. Dalším problémem je uváznutí v lokálním extrému. Tento problém řeším občasnými restarty aktuálního řešení.

## 3. Popis aplikace metody na daný problém.

Nejprve je potřeba mřížku vyplnit čísly tak, aby jejich četnost odpovídala velikosti sudoku a na každé řádce se nacházela právě jednou. Následně pomocí Hill Climbing algoritmu hledám případné řešení dané sudoku.

Po analýze různých způsobů generování sousedního řešení, jsem se rozhodl generování řešit pomocí náhodné permutace pohyblivých prvků jedné řádky a občasné prohození dvou náhodných prvků na náhodně vybrané řádce. Nejprve jsem zkoušel prohazování dvou prvků v jedné řádce či více prohazování během jedné iterace, nicméně po důkladném testování jsem došel k závěru, že permutace jedné řádky a prohození 2 náhodných prvků je jak snadné na implementaci, tak se často dokáže přiblížit ideálnímu řešení. Inspiraci jsem čerpal z textu o $\beta$-hillclimbing, kde vygenerují souseda a následně vyberou náhodně 2 prvky a k jednomu přičtou 1

a druhému odečtou 1. Tato změna hodnot proběhne s určitou pravděpodobností. V mé implementaci jsem to implementoval s pravděpodobností 1/2.

Fitness hodnotu každého řešení počítám pomocí součtu jednotlivých sloupců, řad a čtverců oproti požadované sumě.

$$Min\ Z = \sum_{i=1}^{n} |\sum_{j=1}^{n} x_{ij} - \sum_{k=1}^{n} k| + \sum_{j=1}^{n} |\sum_{i=1}^{n} x_{ij} - \sum_{k=1}^{n} k| + \sum_{i=1}^{n} |\sum_{(l,m)\in B_i}^{n} x_{lm} - \sum_{k=1}^{n} k|$$

$x_{ij}$ zde reprezentuje hodnotu nacházející se v $i$-té řádce na $j$-té pozici. Poslední část reprezentuje kontrolu součtu v jednotlivých blocích. Nalezení potenciálního řešení lze poznat tím, že $Z = 0$. Toto řešení se dále musí zkontrolovat zda každý řádek, sloupec a čtverec obsahuje číslo právě jednou. Pokud jsou splněny všechny podmínky, lze řešení považovat za správné.

# 4. Implementace.

Má implementace dokáže zpracovat sudoku o velikosti $n$, kde $n \geq 4$ a je libovolná mocnina. Jinými slovy, program umí zpracovat sudoku libovolné čtvercové velikosti větší rovno 4.

**Popis jednotlivých částí programu:**

- třída `CHillClimber` a její metody
  - třída řeší celý průběh Hill climbing algoritmu včetně generování dalších řešení i jejich vyhodnocování
  - `hillClimb(CGrid& input)`
    - hlavní funkce algoritmu
    - počet iterací je omezen pomocí konstanty `MAX_ITERS`
  - `printFoundSol()`
    - vypíše nalezené řešení a počet potřebných iterací
  - `fillMatrix()`
    - vyplnění matici tak, aby v každé řádce bylo každé číslo právě jednou
  - `countFitness(CGrid& grid)`
    - spočítání fitness hodnoty aktuálního řešení
    - vrací vypočtenou hodnotu
  - `isLegit(CGrid& grid)`
    - kontrola nalezeného řešení
    - vrací true, pokud je řešení správné
  - `isMovable(int i, int j)`

        - kontrola, zda $x_{ij}$ nebylo zadáno
        - vrací true, pokud je pohyblivý
    - `swapTwoNodes( CGrid& grid )`
        - prohodí 2 náhodně vybrané prvky v jedné řadě
    - `permuteRow( CGrid& grid )`
        - náhodně permutuje všechny pohyblivé prvky v jedné řadě
- třída `CGrid` a její metody
    - třída reprezentující mřížku sudoku
    - `load( std::string& fileName )`
        - načte matici ze souboru
    - `operator << (std::ostream& os, const CGrid& grid)`
        - vykreslí sudoku
    - `checkSqrt(int size)`
        - kontrola, zda zadaná velikost sudoku je libovolná mocnina větší nebo rovno 4
        - vrací true, pokud velikost splňuje podmínku
- třída `CSolver` a její metody
    - třída reprezentující aplikaci
    - `solve()`
        - wrapper metoda pro aplikaci
        - zajišťuje načtení sudoku, řešení a vypsání následného řešení
    - `chooseFile()`
        - výběr ze souborů, které se nacházejí v `/data`
    - `isFile(std::string& fp)`
        - kontrola, zda uživatel vybral soubor, nikoli adresář
        - vrací true, pokud uživatel vybral soubor
- třída `CKeyHandler`
    - třída reprezentující ovladač zpracovávající klávesové signály
    - `kbhit()`
        - metoda, která detekuje stisknutí klávesy
    - `getch()`
        - metoda, která zjistí, která klávesa byla stisknuta

## 5. Reference

Al-Betar, Mohammed & Awadallah, Mohammed & Bolaji, Asaju & Alijla, Basem. (2017). β-Hill Climbing Algorithm for Sudoku Game. 84-88. 10.1109/PICICT.2017.11.