Semestrální práce BI-ZUM.

SUDOKU.

**Zadání semestrální práce.**

V této semestrální práci se zaměříme na SUDOKU jako na optimalizační problém, kde je klíčovým faktorem náhoda a štěstí. Namísto systematických algoritmů se budeme soustředit na metody optimalizaci. V mém případě to je metody genetických algoritmů, které pracují s relativně velkým počtem řešení. Cílem semestrální práci je vyzkoušet genetické algoritmy při řešení SUDOKU.

**Implementace.**

Pro implementaci problému jsem vybral jazyk Python s využítím vhodných knihoven. Úlohy jsou uloženy jednotlivě v souborech. Po čtení souborů používám implementované algoritmy k nalezení řešení.

**Analýza problému.**

Problém, který jsem vybral, spočívá v řešení hádanek Sudoku. Je dána nevyřešená mřížka Sudoku a cílem je vyplnit ji čísly. Místo systematických algoritmů prohledávání stavového prostoru, soustředím se na optimalizační prohledávání celého stavového prostoru pomocí evolučních algoritmů. Cílem je najít jednoduchá řešení SUDOKU v optimálním čase.

**Výběr metody. Genetické algoritmy.**

Při řešení tohoto problému jsem si vybral genetické algoritmy. Neúplnost takového způsobu je kompenzován jeho relativní jednoduchostí a dobou řešení. Toto prohledávání pokračuje až do získání správného řešení pomocí kombinace backtrackingu a genetických algoritmů.

Implementoval jsem následující funkce:

* Genetic algorithm. Tato funkce implementuje genetický algoritmus pro řešení hádanek Sudoku. Přijímá několik parametrů, jako je velikost populace, maximální počet generací a míra mutace. Inicializuje populaci mřížek Sudoku, vyhodnotí fitness každé mřížky, vybere nejlepší jedince pro křížení, vytvoří další generaci pomocí křížení a mutace a pokračuje v tomto procesu po zadaný počet generací nebo dokud není nalezeno řešení
* Crossover. Tato funkce provádí křížení mezi dvěma rodičovskými mřížkami a vytváří podřízenou mřížku. Vybere náhodný bod křížení a zkombinuje části obou rodičovských mřížek, aby vytvořila podřízenou mřížku.
* Mutate. Tato funkce zavádí do mřížky náhodné mutace. Vybere náhodnou buňku v mřížce a změní její hodnotu na náhodně vybrané číslo. To pomáhá vnést do populace rozmanitost a potenciálně zlepšit řešení.
* Evaluate grid. Tato metoda vyhodnocuje vhodnost mřížky sudoku. Vypočítá skóre na základě toho, kolik unikátních čísel se nachází v každém řádku, sloupci a podmřížce. Vyšší skóre znamená lepší řešení. Toto skóre se používá k určení vhodnosti jednotlivých mřížek v populaci.

**Popis programu.**

* + **Inicializace:**

**Třída SudokuSolver je inicializována s danou mřížkou Sudoku.**

* + **Vyhodnocení řešení:**

**Metody is\_valid a evaluate vyhodnocují vhodnost aktuální mřížky a kontrolují, zda nejsou porušena pravidla Sudoku.**

* + **Výběr správného řešení:**

**V metodě genetic\_algorithm jsou sousední řešení generována pomocí operací křížení a mutace.**

* + **Aktualizace řešení:**

**Po vygenerování řešení řešitel aktualizuje aktuální řešení na vybrané s lepší fitness.**

* + **Opakování:**

**Řešitel opakuje proces generování, vyhodnocování a aktualizace řešení, dokud nenajde optimální řešení nebo dokud nedosáhne maximálního počtu populací.**

**Výsledky.**

**Začal jsem se sudoku velikosti 4x4. Řešení byla nalezena a nalezení řešení trvalo méně než 0.0 sekund. Počet iterací je přibližně 10-12. Pak jsem zkusil sudoku 9x9. Všechna řešeni byla nalezena. Doba nalezení a počet iterací teď už zaleží na počtu čísel, které máme nalézt. Maximální doba je 0.04 sekund, což je, avšak méně 1 sekundy. Maximální počet iterací je 4208. Zvládnul jsem i sudoku 16x16, ale řešení bylo nalezeno za přibližně 5 minut, počet iterací při tom 14 milionů. Pro větší rozměry sudoku bude to ještě těžší nalézt řešení v optimálním čase.**

1. **9x9, 0.04 s, 4208;**
2. **9x9, 0.0 s, 200;**
3. **9x9, 0.0 s, 173;**
4. **16x16, 354.3 s, 14579934;**
5. **16x16, 0.01 s, 256;**
6. **2x2, 0.0 s, 12;**
7. **9x9, 0.01 s, 769;**
8. **9x9, 0.0 s, 6;**
9. **16x16, 19.62 s, 1272138;**
10. **4x4, 0.0 s, 15;**