ČVUT, FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ZÁKLADY UMĚLÉ INTELIGENCE
LETNÍ SEMESTR 2019/2020
REPORT K SEMESTRÁLNÍ PRÁCI

Řešení sudoku pomocí metody Optimalizace mravenčí kolonií

Kristýna Klesnilová

29. května 2020

OBSAH

1 Zadání

V této semestrální práci jsem se zabývala řešením hlavolamu sudoku velikosti 9x9 pomocí metody Optimalizace mravenčí kolonií. Inspirovala jsem se článkem https://www.groundai.com/project/solving-sudoku-with-ant-colony-optimisation/1.

2 Popis algoritmu

Fungování algoritmu popisuje následující pseudokód:

```
read in puzzle;
for all cells not failed and with fixed value:
    propagate constraints;
end for
for all cells with more possible values:
    try deducing cell value;
end for
if some cell failed:
    return false;
if all cells fixed:
    return as solution;
initialize global pheromone matrix;
while puzzle is not solved do
    give each ant a local copy of puzzle;
    assign each ant to a different randomly selected cell;
    for number of cells do
        for each ant do
            if current cell has more possible values:
                choose value from possible values based on pheromone level;
                set cell value;
                propagate constraints;
                for all cells with more possible values:
                    try deducing cell values;
                end for
                update local pheromone;
            end if
            move to next cell;
        end for
    end for
    find best ant;
    if best ant has all cell values fixed:
        return as solution
    do global pheromone update;
    do best value evaporation;
end while
```

2.1 Předzpracování sudoku

2.1.1 Constraint propagation

Po načtení konkrétní instance sudoku začnu tím, že si sudoku předzpracuji pomocí metody Constraint propagation. Pro každé vyplněné políčko projdu políčka ve stejném řádku, sloupci a čtverci a vymažu ze seznamu jejich možných hodnot hodnotu vyplňěného políčka. Vznikne-li mi přitom nové vyplněné políčko, v procesu rekurzivně pokračuji.

2.1.2 Dedukce hodnot

Následně opět projdu všechna políčka a v případě, že má políčko více možných hodnot na doplnění, kouknu se, jestli náhodou jedna z těchto hodnot není zakázaná ve všech ostatních políčkách ve stejném řádku, sloupci nebo čtverci. Kdyby náhodou byla, vydedukuji si tak, že tato hodnota se musí nacházet právě na tomto políčku a nastavím ji tam.

2.2 Optimalizace mravenčí kolonií

2.2.1 Matice feromonů

Když v téhle chvíli ještě nemám řešení, pokračuji s hledáním řešením pomocí Optimalizace mravenčí kolonií. Inicializuji si matici feromonů M_{ph}, která pro každé políčko obsahuje hodnotu feromonů pro všechny možné hodnoty, které se na políčko dají doplnit (1-9). Počáteční hodnotu všech feromonů ph_0 nastavím na hodnotu:

$$ph_0 = \frac{1}{cell\_cnt} \tag{2.1}$$

kde *cell_cnt* je počet políček sudoku.

2.2.2 Pohyb mravenců

Následně nastává cyklus, který opakuji, dokud nenajdu řešení. Každému z *NUM_ANTS* mravenců přiřadím hlubokou kopii počáteční plochy sudoku po aplikaci předzpracování pomocí Constraint propagation a dedukce hodnot. Každému mravenci také přiřadím referenci na matici M_{ph} a náhodně mu vyberu políčko p, na kterém bude začínat. Každý mravenec se koukne jestli je políčko na kterém aktuálně stojí validní a jestli ještě nemá určenou fixní hodnotu. V takovém případě mu přiřadí novou hodnotu v podle aktuální hodnoty feromonů v matici feromonů M_{ph} na této pozici. Konstanta *GREEDINESS* kontroluje pravděpodobnost, s jakou bude hodnota v vybrána pomocí ruletové selekce. Na políčko poté aplikuje Constraint propagation a na všechny políčka v matici znovu aplikuje dedukci hodnot.

2.2.3 Lokální aktualizace matice feromonů

Poté provede lokální aktualizaci matice feromonů M_{ph}, podle vzorce:

$$M_{ph}[p][v] = (1 - LOCAL\_PHER\_UPDATE) \cdot M_{ph}[p][v] + LOCAL\_PHER\_UPDATE \cdot ph_0 \tag{2.2}$$

kde parametr *LOCAL_PHER_UPDATE* je malé číslo v rozmezí <0,1>. Díky tomu sníží pravděpodobnost, že mravenec co přijde po něm, si do svého řešení vybere tu samou hodnotu a zajistí tak větší variabilitu vygenerovaných řešení jednotlivými mravenci. Až toto vykonají všichni mravenci, všichni se opět pohnou o další krok, dokud takto neprojdou celou matici.

2.2.4 Nejlepší mravenec

Po tom, co všichni mravenci projdou celou matici, najdu nejlepšího mravence jako takového, kterému se podařilo do sudoku doplnit nejvíce validních hodnot *best_ant_fixed_cnt*. Pokud se mu podařilo validně zafixovat všechna políčka sudoku, vrátím řešení tohoto mravence jako řešení celkové. V opačném případě spočítám kolik feromonu přidat do matice feromonů na základě řešení tohoto mravence *pher_to_add* jako:

$$pher\_to\_add = \frac{cell\_cnt}{cell\_cnt - best\_ant\_fixed\_cnt} \tag{2.3}$$

Jestliže číslo *pher_to_add* vyjde lepší než aktuálně přidávaná přidávaná hodnota do matice feromonů *best_pher_to_add*, tak číslo *best_pher_to_add* aktualizuji na tuto hodnotu a řešení tohoto mravence uložím jako nejlepší, v opačném případě si ponechám řešení nejlepšího mravence z minulých kol a *best_pher_to_add* neaktualizuji.

2.2.5 Globální aktualizace matice feromonů

Poté provedu globální aktualizaci matice feromonů. Všechna políčka v matici feromonů M_{ph}, která jsou v aktuálním nejlepším řešení zafixovaná, aktualizuji podle vzorce:

$$M_{ph}[p][v] = (1 - GLOBAL\_PHER\_UPDATE) \cdot M_{ph}[p][v] + GLOBAL\_PHER\_UPDATE \cdot best\_pher\_to\_add \tag{2.4}$$

kde p jsou pozice zafixovaných políček, v jejich hodnoty a parametr *GLOBAL_PHER_UPDATE* velké číslo z rozmezí <0,1>. Tím zvýším pravděpodobnost, že v dalších kolech budou mravenci na těchto pozicích opět vybírat hodnotu v.

2.2.6 Aktualizace best_pher_to_add

Na konci cyklu ještě akutalizuji hodnotu *best_pher_to_add* podle vzorce:

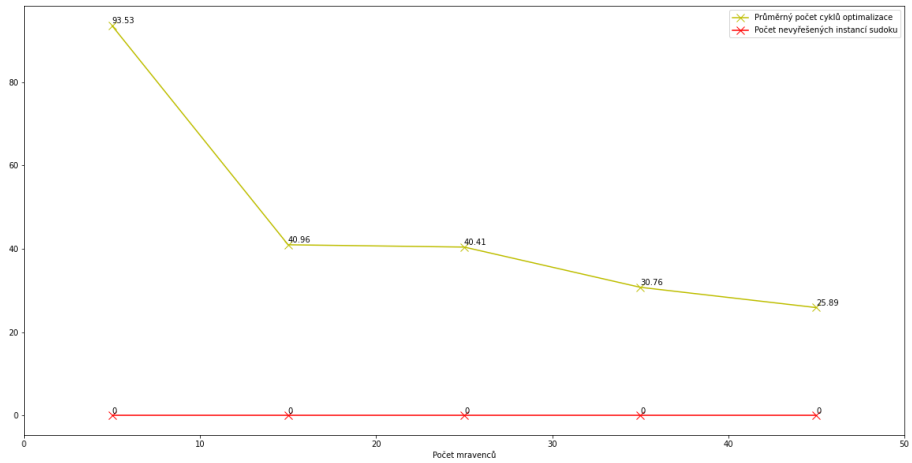
$$best\_pher\_to\_add = best\_pher\_to\_add \cdot (1 - BEST\_PHER\_EVAPORATION) \tag{2.5}$$

kde parametr *BEST_PHER_EVAPORATION* je nějaké malé číslo. Tím se snažím zabránit uváznutí v lokálním optimu.

3 Provedené experimenty a jejich výsledky

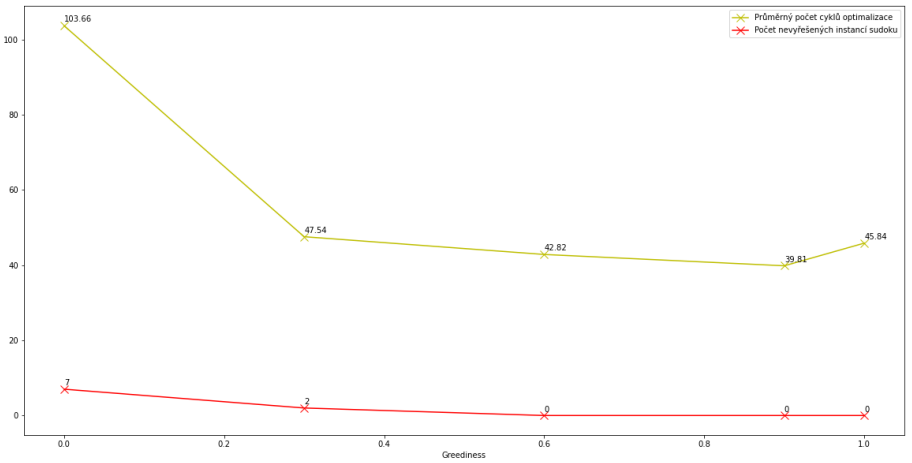
Na řešení instancí sudoku nacházející se ve složce *easy_to_solve* programu často stačí pouze Constraint propagation a dedukce hodnot. Na vyřešení náročnějších z těchto hlavolamů mu pak stačí provést jednotky cyklů Optimalizace mravenčí kolonií. Hlavolamy v této složce jsou běžné sudoku hlavolamy určené pro řešení lidmi. Instance sudoku ve složce *hard_to_solve* už se tak jednoduše řešit nedají, spíše než pro řešení lidmi jsou určené pro řešení počítačem.

Optimální hodnotu parametru *NUM_ANTS* jsem experimentálně určila na 15. Pro instanci sudoku *hard_to_solve/golden_nugget.txt* jsem postupně pro počty mravenců 5, 15, 25, 35 a 45 100krát spustila program a do grafu 3.1 zakreslila pro každý počet mravenců počet nevyřešených instancí sudoku a průměrný počet cyklů optimalizace, které program musel vykonat, než našel řešení. S vyšším počtem mravenců než 15 dostávám správné řešení v průměru po jen o



Obrázek 3.1: Optimální počet mravenců

něco málo nižším počtu cyklů průběhu optimalizace, ale za cenu vyšší výpočetní náročnosti. Při snížení počtu mravenců na 5 už program potřebuje na vrácení správného výsledku v průměru významně více cyklů optimalizace.



Obrázek 3.2: Optimální hodnota parametru *GREEDINESS*

Následně jsem zkoumala optimální hodnotu parametru *GREEDINESS*. Opět jsem pro instanci sudoku *hard_to_solve/golden_nugget.txt* 100krát spustila program pro hodnoty konstanty *GREEDINESS* 0, 0.3, 0.6, 0.9 a 1 a do grafu 3.2 opět zanesla počet nevyřešených instancí sudoku a průměrný počet cyklů optimalizace. Z grafu je vidět, že vysoká hodnota parametru *GREEDINESS*, tedy časté používání ruletové selekce, pomáhá správnosti a rychlosti najití výsledku. S nízkou hodnotou parametru *GREEDINESS* se program občas zasekne v lokálním optimu a výsledek nenajde. Parametr *GREEDINESS* jsem tedy nastavila na hodnotu 0.9.

Zbylé konstanty *LOCAL_PHER_UPDATE*, *GLOBAL_PHER_UPDATE* a *BEST_PHER_EVAPORATION*

jsem nastavila na následující hodnoty:

$$\begin{aligned} LOCAL\_PHER\_UPDATE &= 0.1 \\ GLOBAL\_PHER\_UPDATE &= 0.9 \\ BEST\_PHER\_EVAPORATION &= 0.005 \end{aligned} \tag{3.1}$$

4 Instrukce ke spuštění

Práce je napsaná v jazyce Python. Nachází-li se člověk ve složce *sudoku*, spustí ji pomocí terminálového příkazu:

```
python3 src/main.py <cesta k souboru obsahující instanci sudoku> <gui/console>
```

Soubory s instancemi sudoku se nacházejí ve složce *data*, ve složce *src* se pak nacházejí veškeré zdrojové kódy. Pro nejlepší vizualizaci funkcionality je dobré program spouštět s instancemi sudoku nacházejícími se ve složce *data/hard_to_solve*, jejichž řešení není pro program triviální a program tak neběží tak krátce. Při spuštění s argumentem *console* program vypisuje do terminálu, s argumentem *gui* se řešení vizualizuje pomocí Python knihovny PyGame. Na začátku program zobrazí úvodní instanci sudoku po načtení souboru, změny po provedení Constraint propagation a dedukce a poté již vizualizuje samotný průběh Optimalizace mravenčí kolonií. Vždy je vidět aktuální nejlepší řešení po provedení jednoho cyklu optimalizace. Při spuštění v PyGame se také v pravém horním rohu každého políčka sudoku zobrazuje to číslo na doplnění, které má pro danou pozici aktuálně největší hodnotu feromonů.