Sztuczna Inteligencja i Inżynieria Wiedzy

Sprawozdanie numer 2

Autor: Patryk Konopka, 237980

Data: 14.04.2019

1. Cel ćwiczenia

Celem ćwiczenia było zapoznanie się z podstawowymi algorytmami stosowanymi do rozwiązywania problemów spełniania ograniczeń (CSP). W moim przypadku do realizacji wybrałem problem N-hetmanów.

2. Algorytm genetyczny

Mój algorytm składa się z następujących plików:

- Main.py – plik wykonawczy
- Forward.py – algorytm forward checking
  - checkForward(**board, N, row, col**) – metoda uzupełniająca dziedzinę w tablicy **board** o rozmiarach **N**. Metoda uzupełnia dziedzinę zaczynając od wiersza **row** i kolumny **col** (aby uniknąć sprawdzania dziedziny w miejscach już sprawdzonych)
  - forwardChecking(**board, row, N**) – algorytm forward checking
- Backtracking.py – algorytm backtracking
  - isSafe(**board, row, column, N**) – metoda sprawdzająca czy dopuszczone jest wstawienie hetmana w tablicy **board** o wymiarach **N**. Metoda sprawdza zaczynając od wiersza **row** i kolumny **col** (aby uniknąć sprawdzania w miejscach już sprawdzonych wcześniej)
  - solveBackTracking(**board, column, N**) – algorytm backtracking

3. Działanie algorytmu

Obydwa moje algorytmy zwracają pierwsze znalezione rozwiązanie. W poniżej tabeli chciałbym porównać nakład obliczeniowy (liczbę wywołań) oraz czas wykonywania backtrackingu. W tabeli umieszczę wyniki dla głównej metody (**solveBackTracking**), wewnętrznej metody pomocniczej **(isSafe)** oraz dla całego programu.

N	isSafe (wywołania)	isSafe (czas)	solveBackTracking (wywołania)	solveBackTracking (czas)	Razem (wywołania)	Razem (czas)
4	38	0	12	0	261	0,001
5	20	0,001	7	0,001	318	0,002
6	117	0	23	0	527	0,004
7	63	0	13	0	575	0,004
8	348	0,001	48	0,002	1023	0,005
9	99	0	16	0	886	0,004
10	1915	0,024	197	0,026	3043	0,030
11	264	0,003	30	0,003	1401	0,080
12	8634	0,059	726	0,064	10659	0,068
13	87	0,007	74	0,008	2452	0,014
14	25305	0,148	1815	0,158	28851	0,161
15	4425	0,03	303	0,032	6699	0,036
16	80952	0,555	5068	0,587	88247	0,591
17	13328	0,095	793	0,099	16620	0,103
18	539649	3,93	29990	4,127	572426	4,131

Jak możemy wyczytać z tabeli wartości te nie rosną w ustalony sposób. Wynika to zapewne z „przypadkowego” szybkiego znalezienia prawidłowego rozwiązania. Wyniki byłyby bardziej wiarygodne jeżeli byśmy wprowadzili losowego hetmana na początkowej tablicy lub zwracalibyśmy wynik dopiero po znalezieniu wszystkich możliwych kombinacji. Nie udało mi się czynnika, który powoduje znaczne wydłużenie działania algorytmu dla parzystych N.

W poniżej tabeli porównam nakład obliczeniowy (liczbę wywołań) oraz czas wykonywania dla forward checkingu. W tabeli umieszczę wyniki dla głównej metody (**forwardChecking**), wewnętrznej metody pomocniczej **(checkForward)** oraz dla całego programu.

N	checkForward (wywołania)	checkForward (czas)	forwardChecking (wywołania)	forwardChecking (czas)	Razem (wywołania)	Razem (czas)
4	5	0,001	5	0,003	305	0,003
5	4	0	5	0,003	361	0,003
6	28	0,001	22	0,004	970	0,004
7	8	0	8	0,004	638	0,004
8	84	0,002	62	0,007	2414	0,007
9	29	0,001	23	0,006	1349	0,006
10	90	0,002	71	0,007	2881	0,007
11	50	0,001	41	0,006	2157	0,006
12	226	0,005	180	0,011	6332	0,011
13	102	0,002	87	0,006	3771	0,006
14	1759	0,046	1428	0,074	41934	0,074
15	1211	0,034	1000	0,056	29784	0,056
16	9372	0,289	7855	0,432	219689	0,432
17	5095	0,153	4477	0,232	122915	0,232
18	38130	1,225	32800	1,826	897236	1,826

W tym przypadku wyniki są już bardziej uporządkowane, jednak ciągle daleko im do liniowości. Samych wykonań algorytmów jest znacznie mniej niż w przypadku backtrackingu, jednakże o wiele więcej wykonuje się funkcji zewnętrznych (m. in. Z bibliotek Pythona). Głównymi sprawcami tutaj są metody deepCopy oraz sum.

Wyniki byłyby bardziej wiarygodne jeżeli byśmy wprowadzili losowego hetmana na początkowej tablicy lub zwracalibyśmy wynik dopiero po znalezieniu wszystkich możliwych kombinacji. Nie udało mi się znaleźć czynnik, który powoduje znaczne wydłużenie działania algorytmu dla parzystych N.

4. Wnioski

Realizacja tego ćwiczenia znacznie poszerzyła moją wiedzę. Dowiedziałem się o tym jak można zdefiniować sudoku, czy o problemie N-hetmanów. Udało mi się również zaimplementować swój pierwszy program, który rozwiązuje problem CSP. Ten program nie jest idealny i ma dużo braków (brak wyszukiwania wszystkich rozwiązań, problemy optymalizacyjne itp.). Jednakże z braku czasu i natłoku zadań wymaganych na innych kursach dopracowanie tego programu aktualnie nie jest możliwe.

Moje algorytmy opierają się na rekurencji, bez której realizacja tych algorytmów była dla mnie bardzo trudna. Największym problem, było dla mnie kontrolowanie dziedziny w algorytmie forwardChecking, jednak udało mi się nad tym zapanować dzięki kopiowaniu szachownicy w odpowiednim momencie.

Podsumowując całe zadanie było ciekawym zagadnieniem. Znacząco pomogło mi w rozwoju moje wiedzy z zakresu programowania w Pythonie oraz problemów CSP.