Informatyka, studia dzienne, inż I st. semestr IV

Sztuczna inteligencja i systemy ekspertowe 2021/2022
Prowadzący:

Data oddania: _____ Ocena: _____

Paweł Wieczorek

Badanie algorytmów: BFS, DFS, A*

1. Cel

Przedmiotem tego zadania jest gra-układanka zwana: "piętnastką". Należało wykonać trzy rodzaje algorytmów, które znajdowały sekwencję ruchów pozwalającą na doprowadzenie układanki ze stanu niepoukładanego do stanu poukładanego. W części ekperymentalnej należało zbadać jak opracowane algorytmy radzą sobie z uporządkowaniem 420 układów układanki w odległości 1-7 od układu wzorcowego. Odległoość np. o warości 4 oznacza wykonanie 4 ruchów na układance uporządkowanej dzięki czemu otrzymuje się układankę nieuporządkowaną.

2. Wprowadzenie

Układankę "piętnastka" można rozwiązać wykorzystując algorytmny przeszukujące graf stanów układanki: Breadth-first search (BFS), Depth-first search (BFS) i A* (ASTR). Korzeniem takiego grafu jest wybrana układanka wejściowa. Dziećmi każdego węzła w grafie (także korzenia) są układanki, które można otrzymać dokonując przemieszczenia wolnego pola układanki-rodzica. W zależności gdzie położone jest wolne pole w układance, mogą powstać tylko i wyłącznie: 2, 3 lub 4 węzły-dzieci. Dwa węzły-dzieci powstają gdy wolne pole znajdue się w jednym z narożników układanki. Trzy węzły-dzieci powstają gdy wolne pole znajdue się w na jednej z czterech krawędzi układanki. Cztery węzły-dzieci powstają gdy wolne pole znajdue się na innej pozycji niż wcześniej wspomniane.
Algorytmy BFS i DFS pracują według podanego przez programistę porządku przeszukiwania grafu stanów. W praktyce oznacza to, że programista podaje

w jakiej kolejności algorytmy BFS i DFS mają "odwiedzać" węzły-dzieci. Przykładowo przy wybraniu porządku poszukiwań prawo-dół-góra-lewo, algorytmy rozpoczną przetwarzanie węzłów-dzieci od węzłą, który reprezentuje układankę, która powstaje gdy układanka-rodzic przemieści położenie wolnego pola w prawą stronę.

3. Opis implementacji

W rozwiązaniu wykorzystano język Java. W celu rozwiązania problemu utworzono interfejs o nazwie Solver, który jest implementowany przez trzy klasy reprezentujące konkretne algorytmy: AstrSolver (algorytm A*), DfsSolver (algorytm DFS), BfsSolver (algorytm BFS). Dodatkowo stworzono klasę ze statycznymi funkcjami pomocniczymi o nazwie Helper. Możliwość utrwalania statystyk każdego z algorytmów została zaimplementowana poprzez klasę Tracker, która zbiera i zapisuje informacje w czasie rzeczywistym. Fukcjonalność zapisu raportów i podsumowań z klasy Tracker zaimplementowano w klasie OutputWriter, która zapisuje wspomniane dane w plikach txt. Utworzono klasę ExperimentTracker, której zadaniem było śledzenie pojedynczego eksperymentu (opisane w cz.4), obliczenie statystyk i zapisanie w pliku tekstowym: report. Utworzono klasę Executor, której zadaniem jest wywołanie fukcji main z klasy Program dla różnych wariantów eksperymentu.

4. Materiały i metody

Przebadano wszystkie układy początkowe układanki w odległościach 1-7 od układu wzorcowego (w sumie 420 układów).Wykorzystano wszystkie trzy algorytmy ze wcześniejszego podpunktu.

W przypadku algorytmów BFS i DFS wykorzystano 8 następujących porządków przeszukiwania sąsiedztwa: prawo-dół-góra-lewo, prawo-dół-lewo-góra, dół-prawo-góra-lewo dół-prawo-lewo-góra, lewo-góra-dół-prawo,
lewo-góra-prawo-dół, góra-lewo-dół-prawo, góra-lewo-prawo-dół. W programie wprowadzono skrócone oznaczenia algorytmów i porządków rzeszukiwań: "bfs" dla algorytmu przeszukującego wszerz, "dfs" dla algorytmu przeszukującego wgłąb "astr" dla algorytmu A*, natomiast porządek przeszukiwania jest skrótowo zapisywany w postaci czterech liter gdzie L oznacza: "lewo", R ozancza "prawo", U oznacza "góra", D oznacza "dół". Przykładowo "lurd" oznacza kolejność: lewo-góra-prawo-dół (wielkość liter nie jest istotna).
W przypadu algorytmu A* wykorzystano dwie heurystyki: heurystykę Manhattan oraz heurystykę Hamminga. Obie heurystyki pozwalają ocenić podobieństwo dwóch układanek. W naszym zadaniu wykorzsytujemy obie heurystyki do określenia podobieństwa danej układanki do układanki docelowej (uporządkowanej). Heurystyka Hamminga pozwala na przypisanie liczby całkowitej do układu, która określa jak bardzo dany układ jest podobny do docelowego. Wartość tej liczby całkowitej otrzymujemy zliczając w ilu odpowiadajcych sobie miejscach dwie układanki mają różne wartości. Wynika z tego, że w naszym przypadku minimalną wartością miary Hamminga jest

0 (dla identycznych układów), a największą 16 (dla dwóch zupełnie różnych układów). Heurystyka Manhattan dla każdego z elementów układanki zlicza ile ruchów należy wykonać aby przemieścić dany element na miejsce, który zajmuje w układance docelowej.
W części eksperymentalnej przetestowano 18 wariantów algorytmu, każdy wariant dla 420 układów nieuporządkowanych.

Eksperyment 1: Algorytm A*, heurystyka manhattan
Eksperyment 2: Algorytm A*, heurystyka Hamminga

Eksperyment 3: Algorytm BFS, prawo-dół-góra-lewo
Eksperyment 4: Algorytm BFS, prawo-dół-lewo-góra
Eksperyment 5: Algorytm BFS, dół-prawo-góra-lewo
Eksperyment 6: Algorytm BFS, dół-prawo-lewo-góra
Eksperyment 7: Algorytm BFS, lewo-góra-dół-prawo
Eksperyment 8: Algorytm BFS, lewo-góra-prawo-dół
Eksperyment 9: Algorytm BFS, góra-lewo-dół-prawo
Eksperyment 10: Algorytm BFS, góra-lewo-prawo-dół

Eksperyment 11: Algorytm DFS, prawo-dół-góra-lewo
Eksperyment 12: Algorytm DFS, prawo-dół-lewo-góra
Eksperyment 13: Algorytm DFS, dół-prawo-góra-lewo
Eksperyment 14: Algorytm DFS, dół-prawo-lewo-góra
Eksperyment 15: Algorytm DFS, lewo-góra-dół-prawo
Eksperyment 16: Algorytm DFS, lewo-góra-prawo-dół
Eksperyment 17: Algorytm DFS, góra-lewo-dół-prawo
Eksperyment 18: Algorytm DFS, góra-lewo-prawo-dół

W tabeli wyników, dla każdego eksperymentu, zaprezentowano:
a) średni czas rozwiązywania układanki
b) średnią długość znalezionego rozwiązania (liczba ruchów),
c) iczba unikalnych układów powstałych w czasie pracy algorytu
d) liczba układów procesowanych (układy, które były porównywane z układem docelowym)

Dla algorytmu DFS wyznaczono także średnią maksymalną osiągniętą wartość głębokości rekurencji (maksymalna ilość zagnieżdżonych wywołań funkcji rekurencyjnej).

5. Wyniki

Ponizsza tabela prezentuje wyniki wykonania wszystkich wariantów eksperymentu z punktu poprzedniego.

nr.	algorytm	polityka	czas [ms]	długość rozwiązania	unikatowe układy	układy procesowane	głębokość rekurencji
1	A*	hamm	0.463	6.13	44	20	-
2	A*	manh	0.173	6.13	27	12	-
3	BFS	rdul	5.026	6.13	3673	1162	-
4	BFS	rdlu	4.49	6.13	3672	1162	-
5	BFS	drul	4.44	6.13	3672	1162	-
6	BFS	drlu	4.442	6.13	3673	1162	-
7	BFS	ludr	4.755	6.13	3930	1245	-
8	BFS	lurd	4.652	6.13	3929	1244	-
9	BFS	uldr	4.758	6.13	3929	1244	-
10	BFS	ulrd	4.670	6.13	3930	1245	-
11	DFS	rdul	289.177	12.8	107371	107371	20
12	DFS	rdlu	280.642	13.35	102708	102708	20
13	DFS	drul	277.576	13.35	102708	102708	20
14	DFS	drlu	291.041	12.80	107371	107371	20
15	DFS	ludr	224.662	12.65	83133	83133	18
16	DFS	lurd	245.928	12.40	91423	91423	19
17	DFS	uldr	245.181	12.40	91423	91423	19
18	DFS	ulrd	223.437	12.65	83133	83133	18

Tabela 1. Wyniki Eksperymentów 1 - 18.

1) Algorytm A* okazał się zdecydowanie najszybszy
2) Algorytm BFS był kilkukrotnie wolniejszy od A*
3) Algorytm DFS okazał się o kilka rzędów wielkości wolniejszy od pozostałych.
4) Algorytmy BFS i A* znajdowały układy-rozwiązania o identycznych długościach.
5) Algorytmy DFS znajdował relatywnie długie rozwiązania
6) Algorytm A* generował bardzo niewiele unikatowych układów
7) Algorytm A* procesował bardzo niewiele układów ogółem.
8) Algorytm DFS generował tyle samo unikatowych układów co procesował

6. Dyskusja

W tej sekcji przeanalizowane zostaną zależności zaobserwowane w sekcji poprzedniej.
1) Algorytm A* okazał się najszybszy, ponieważ wykorzystywał heurystykę dzięki, której priorytetowo analizował układy bardziej zbliżone do rozwiązania.
2) Algorytm BFS był dużo wolniejszy od A*, ponieważ nie korzystał z heurystyk i ustalał priorytet kolejnych węzłów na podstawie polityki.
3) Algorytm DFS okazał się o kilka rzędów wielkości wolniejszy od pozostałych, ponieważ kolejne rekurencyjne wywołania z tą samą polityką powodowały bardzo szybkie oddalanie się od korzenia grafu co powodowało, że nawet dla trywialnych układanek, DFS znajdował bardzo zawiłe rozwiązania.

4) Algorytmy BFS i A* znajdowały układy-rozwiązania o identycznych długościach, ponieważ A* jest tak zaprojektowany aby zawsze znajdować najbardziej optymalną ścieżkę, natomiast BFS metodą "brute force", sparawdzał każdy węzeł na danym poziomie, więc prędzej czy później znajdował układankę docelową jednocześnie znajdując najbardziej optymalną ścieżkę.
5) Algorytmy DFS znajdował relatywnie długie rozwiązania, ze względu na mechanizm umówiony w podpunkcie 3)
6) Algorytm A* generował bardzo niewiele unikatowych układów, ponieważ korzystał z heurystyk, które sprawiają, że algorytm analizuje jedynie najbardziej obiecujące układy/ścieżki.
7) Algorytm A* procesował bardzo niewiele układów ogółem, ze względu na mechanizm opisany w punkcie wcześniejszym.
8) Algorytm DFS generował tyle samo unikatowych układów co procesował, ponieważ DFS zapamiętywał odwiedzone już układy, dzięki czemu nie procesował ich więcej niż raz.

7. Wnioski

— Algoytmy A* z odpowienią heurystyką jest zdecydowanie lepszy w rozwiązywnaniu probelemu "piętnastki" od algorytmów BFS i DFS.
— Algorytm BFS znajdował optymalne rozwiąznia.
— Algorytm DFS okazał się skrajnie niewydajny w tym problemie.
— Algorytm DFS znajdował bardzo skomplikowane rozwiąznia nawet dla trywialnych układanek.

Wiarygodność wyników, z pewnością możnaby poprawić poprzez przeprowadzenie eksperymentów dla jeszcze bardziej skomplikowanych układów niż te wykorzystane w zadaniu. Ze względu na to, że jako miarę użyteczności algorytmu wykorzystujemy średni czas rozwiązywania, należałoby wykonać złożoną analizę innych czynników, które mogą mieć wpływ na czas działania algorytmu.

Literatura

[1] T. Oetiker, H. Partl, I. Hyna, E. Schlegl. *Nie za krótkie wprowadzenie do systemu LaTeX2e*, 2007, dostępny online.
[2] Wikipedia. *Depth-first search*, dostępny online.
[3] Wikipedia. *Breadth-first search*, dostępny online.
[4] Wikipedia. *A* search algorithm*, dostępny online.