Sprawozdanie I

Przeszukiwanie grafów i planowanie tras przejazdu

Sekcja “4 Wyszukiwanie najkrótszej ścieżki”:

1. Statystyki dla problemu bez ograniczeń

Ustawienia	**Długość ścieżki**	**Czas pracy [ms]**	**Liczba operacji**
Breadth-First-Search (domyślny)	15.07	1.8	1418
Dijkstra (domyślny)	15.07	4	1363
Depth-First-Search (domyślny)	15.07	91	1363
A (domyślny)*	15.07	90	1363
Depth-First-Search (dx)	16.24	15	207
A (dx)*	15.07	29	465
Depth-First-Search (Manhattan)	15.07	8.1	115
A (Manhattan)*	15.07	14	222
Depth-First-Search (Diagonal)	15.66	11	142
A (Diagonal)*	15.07	26	323
Depth-First-Search (Euclidean)	15.66	9.7	118
A (Euclidean)*	15.66	9.9	128

2. Jak wyglądają definicje heurystyki (w narzędziu on-line):
   a. Manhattan: dx+dy
   b. Diagonal: (dx + dy) - Math.min(dx, dy)
   c. Euclidean: dx**2+dy**2
3. Jaka heurystyka była w stanie doprowadzić algorytmy A* i Depth-First-Search do przejścia nad dłuższą krawędzią w **zadaniu 1**, podpunkt 3?
   dy**3

4. Statystyki dla problemu z **zadania 2**:

Ustawienia	**Długość ścieżki**	**Czas pracy**	**Liczba operacji**
A (Euclidean)*	38.97	41	646
A (Manhattan)*	36.49	77	1300
A (Diagonal)*	36.49	98	1707
A (dx)*	36.49	128	2252
A (5*dx)*	37.07	80	1415
A (dy)*	36.49	146	2507
Breadth-First-Search (bidirectional)	36.49	1.9	282
Dijkstra (bidirectional)	36.49	1	279
A (Euclidean, bidirectional)*	38.14	15	208
A (Manhattan, bidirectional)*	38.14	13	191

A (Diagonal, bidirectional)*	37.31	12	167
A (dx, bidirectional)*	36.49	10	148
A (5*dx, bidirectional)*	38.49	10	155
A (dy, bidirectional)*	38.14	21	324

5. Czy któraś heurystyka działała w lepiej w trybie jednokierunkowym?
   A* z wszystkimi heurystykami (oprócz Euklidesowej) znajdują krótsze scieżki, jednakże wyszukują je znacznie dłużej w trybie jednokierunkowym.

6. Zaproponuj najlepszą strategię przeszukiwania dla tego problemu i zapisz dla niej statystyki.
   (dx+dy)**2 {„len” : 38.14, „time” : 16, „operations” : 212}

Sekcja “5 Problem n-puzzli”

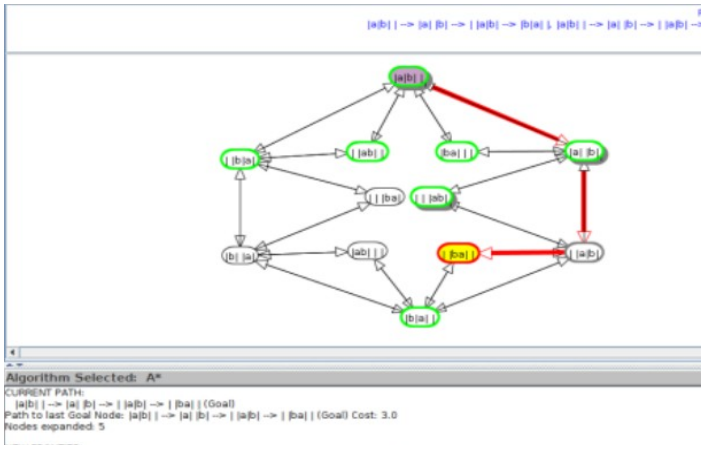
1. Graf reprezentujący puzzle 5x5 zajmie w pamięci 2 * 10**12 TB
2. Czy graf musi być w całości wygenerowany przed przystąpieniem do przeszukiwania? NIE
3. Statystyki dla różnych algorytmów i heurystyk dla domyślnego problemu:

Ustawienia	**Długość ścieżki**	**Open list**	**Closed list**
Breadth-First-Search	6	60	84
Iterative Deepening Search	6	2	17
Greedy (Euclidean)	6	4	7
Greedy (Manhattan)	7	4	7
Greedy (Tile out-of-place)	7	4	7
A (Euclidean)*	8	6	8
A (Manhattan)*	7	4	7
A (Tile out-of-place)*	7	4	7

4. Jak na proces rozwiązywania wpływa zamiana 1 i 2 w stanie początkowym? Jeżeli zmiana jest zauważalna, co jest jej przyczyną?
   Zamienilismy pozycję 1 i 2 ręcznie, ale w ramach zasad gry nie byłoby to możliwe, zatem można by rzec całkowicie zmieniliśmy przestrzeń stanów tej układanki co uniemożliwia osiągnięcie stanu docelowego, który zawierał się w przestrzeni przed modyfikacją.

Sekcja „6 Świat klocków”

1. Poniżej znajduje się screenshot z narysowanym grafem stanów dla świata klocków:



2. Wynik rozwiązania dla tego grafu:
    a. Ścieżka: |a|b| | --> |a| |b| --> | |a|b| --> | |ba| | (Goal)
    b. Długość (koszt ścieżki): 3
    c. Liczba rozwiniętych węzłów: 5

Sekcja „7 Zagadnienie automatycznego planowania”

1. Reguły dla brakujących akcji w solverze STRIPS:
    a. ACTION II:
    strips_rule(take_from_Y(X,Y),
    [handempty, on(X,Y), clear(X)],
    [holding(X), clear(Y)],
    [clear(X), on(X,Y), handempty]).
    b. ACTION III:
    strips_rule(stack_on_table(X),
    [holding(X)],
    [handempty, clear(X)],
    [holding(X)]).
    c. ACTION IV:
    strips_rule(take_from_table(X),
    [clear(X), handempty],
    [holding(X)],
    [clear(X), handempty]).
2. Plany znalezione przez solver dla problemów:
    a. problem1:

Init: [*ontable*(a), *ontable*(b), *ontable*(c), *clear*(a), *clear*(b), *clear*(c), hand empty]
Goal: [*on*(b, c), *on*(a, b)]
Plan: *take_from_table*(b)
stack(b, c)
take_from_table(a)
stack(a, b)

b. problem2:
c. Init: [*on*(a, c), *ontable*(b), *ontable*(c), *clear*(a), *clear*(b), handempty]
Goal: [*on*(b, c), *on*(a, b)]
Plan: *take_from_Y*(a, c)
stack_on_table(a)
take_from_table(b)
stack(b, c)
take_from_table(a)
stack(a, b)

d. problem3:
Init: [*on*(c, a), *ontable*(a), *ontable*(b), *clear*(b), *clear*(c), handempty]
Goal: [*on*(b, c), *on*(a, b)]
Plan: *take_from_Y*(c, a)
stack_on_table(c)
take_from_table(b)
stack(b, c)
take_from_table(a)
stack(a, b)

3. Dodatkowy problem:
   a. stan początkowy:
   [*on*(b, c), *on*(c, a), *ontable*(a), *clear*(b), handempty]
   b. stan końcowy:
   [*on*(a, b), *on*(b, c)]
   c. znaleziony plan:
   take_from_Y(b, c)
   stack_on_table(b)
   take_from_Y(c, a)
   stack_on_table(c)
   take_from_table(b)
   stack(b, c)
   take_from_table(a)
   stack(a, b)
4. Opisz poniżej, jaką wartość dostrzegasz w rozwiązywaniu problemów bez warunków `clear/1`
   `Kiedy w dodatkowym problemie zapomniałem dodać warunku clear(b) to problem został rozwiązany poprzez wyjęcie klocków z środka stosu. Może okazać się to przydatne przy kolejkowaniu i dzieleniu pomiędzy procesy zadań niezależnych od siebie.`