Informatyka, studia niestacjonarne semestr V

Sztuczna inteligencja i systemy ekspertowe **2018/2019**

Prowadzący: dr inż. Przemysław Nowak niedziela, 17:15

Data oddania: _____ Ocena: _____

Marcin Pajkowski 211968
Rafał Warda 214067

Przeszukiwanie przestrzeni stanów - gra "Piętnastka"

1. Cel

Implementacja grafowych algorytmów przeszukiwania przestrzeni stanów; porównanie metod ślepych i heurystycznych na przykładzie układanki "piętnastka".

2. Wprowadzenie

2.1. O grze "piętnastka"

"Piętnastka" jest grą z serii łamigłówek logicznych. W klasycznym wydaniu jest zbudowana jako ramka, wewnątrz której znajdują się klocki. Jedno pole tej konstrukcji zawsze jest wolne, umożliwia to graczowi przesuwanie klocków. Celem rozgrywki jest ułożenie klocków w określonej kolejności - może to być ciąg liczb naturalnych lub obrazek.

2.2. Złożoność problemu i rozwiązywalność

Nie wszystkie układy są rozwiązywalne. Możemy je podzielić na parzyste i nieparzyste - co oznacza, że do uzyskania stanu wzorcowego należy wykonać parzystą lub nieparzystą liczbę ruchów. Wszystkie układy powstające wskutek legalnego przesuwania klocków - wykonując pierwszy ruch począwszy od układu docelowego - są układami parzystymi. Ilość stanów każdej układanki można wyznaczyć za pomocą wzoru: $iloscStanow = \frac{iloscPol!}{2}$.

2.3. Grafowa reprezentacja przestrzeni stanów

Grafem nazywamy obiekt matematyczny składający się z niepustego zbioru wierzchołków W i zbioru krawędzi K łączącego niektóre z tych wierzchołków [1]. Graf może świetnie posłużyć do zapisu przebiegu gry "piętnastka". Za wierzchołki grafu można przyjąć kolejne stany planszy układanki, a krawędzie można zdefiniować jako kierunek przesunięcia wolnego pola (tj. zamiany wolnego pola z elementem znajdującym się względem niego nad nim, pod nim, z jego lewej lub prawej strony).

2.4. Metody przeszukiwania przestrzeni stanów

2.4.1. Podział algorytmów

- Metody ślepe (klasyczne):
  - breadth-first search (BFS)
  - depth-first search (DFS)
- Metody oparte o heurystyki:
  - algorytm A*

Algorytmy klasyczne jako dodatkowy parametr przyjmują *porządek przeszukiwania*, zaś algorytm A* do przyspieszenia procesu przeszukiwania wykorzystuje *metryki* - zostaną zaprezentowane metryka Hamminga oraz Manhattan (innymi nazwami tej metryki spotykane w literaturze metryka taksówkowa lub metryka miejska).

2.4.2. Breadth-First Search

Algorytm BFS przeszukuje graf *wszerz* - na początku odwiedzany jest stan początkowy, następnie jego sąsiedzi a w dalszej kolejności sąsiedzi sąsiadów rozwiniętych w poprzednich iteracjach - do momentu znalezienia stanu wzorcowego.

2.4.3. Depth-First Search

Inaczej działa algorytm DFS: w literaturze można spotkać się ze stwierdzeniem, że przeszukuje on graf *w głąb*. Można łatwo to uzasadnić - DFS po rozwinięciu stanu wyjściowego nie próbuje rozwijać stanów na tej samej wysokości jak ma to miejsce W BFS. Zamiast tego odwiedza stany-dzieci i rozwija je do momentu napotkanej przeszkody. Taką przeszkodą może być koniec ścieżki w głąb (w wypadku układanki - brak możliwości przesunięcia klocka w danym kierunku) lub arbitralnie ustalony limit osiąganej głębokości.

2.4.4. A*

Algorytm A* przyjmuje inną strategię działania niż metody omówione wyżej - nie opiera się na ściśle określonym porządku. Suplementem tego algorytmu wspomagającym ustalenie dalszej kolejności przeszukiwania jest *heurystyka*. Zostaną zaprezentowane dwie heurystyki - metryka Hamminga i Manhattan. Stany oczekujące na przetworzenie odkładane są do listy, która jest uporządkowana rosnąco według funkcji $f(x) = g(x) + h(x)$, gdzie $g(x)$ to długość ścieżki od stanu wyjściowego do stanu x, a $h(x)$ oznacza aproksymowana odległość do stanu docelowego według heurystyki h.

3. Implementacja

3.1. Stos technologiczny

Program został napisany w technologii C++ 17 z wykorzystaniem biblioteki Google Test wspierającej testy jednostkowe. Stan układanki przedstawiony jest jako klasa State. Klasa ta zawiera jednowymiarową tablicę o wielkości NxM - układanka została zrzutowana na jeden wymiar celem zredukowania zjawiska *cache miss*. Informacje o rzeczywystych wymiarach zapisane są w atrybutach klasy. Do zrealizowania poszczególnych metod przeszukiwania przestrzeni stanów posłużono się algorytmami i strukturami danych dostępnymi w bibliotece STL języka C++.

3.2. Usprawnienie DFS

W przypadku "czystego" algorytmu DFS niemożliwe było znalezienie rozwiązań układanki z arbitralnie ustawionym maksymalnym stopniu rekursji na 20. W tym celu zastosowano pewną modyfikację algorytmu - jeżeli stan rozwijany lub odwiedzany znajduje się bliżej stanu wyjściowego to stan ten będzie brany pod uwagę.

4. Materiały i metody

Do zrealizowania zadania zostały użyte następujące programy i skrypty wspomagające dostarczone razem z kartą przedmiotu:

1. Generator układanek,
2. Walidator układanek,
3. Wizualizator układanek,
4. Skrypt uruchamiające program przeszukujący w trybie wsadowym (powłoka bash),
5. Skrypt ekstraktujący dane wygenerowane podczas przeszukiwania (powłoka bash).

Do stworzenia pliku binarnego programu generującego rozwiązania z kodu źródłowego C++ użyto kompilatora Clang z pakietu LLVM. Wykresy zostały przygotowane w narzędziu Jupyter Notebook (kernel Python 3). Użyto bibliotek pandas, numpy i matplotlib.

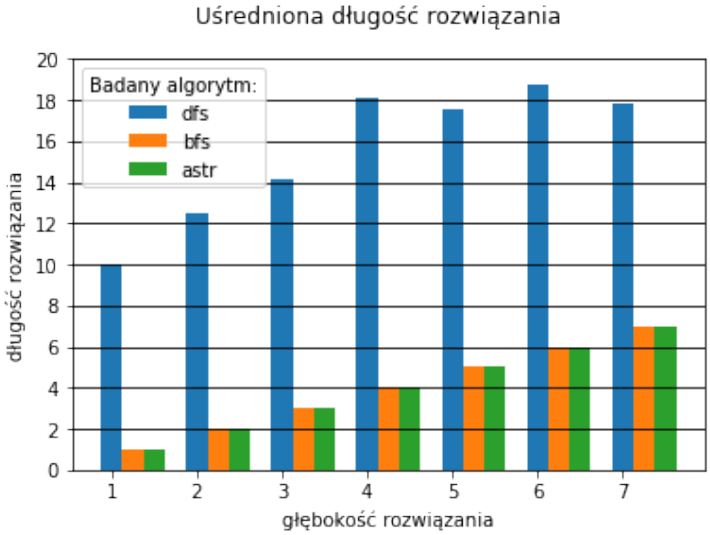
Algorytmy przeszukiwania przestrzeni stanów zostały porównane na podstawie parametrów takich jak:

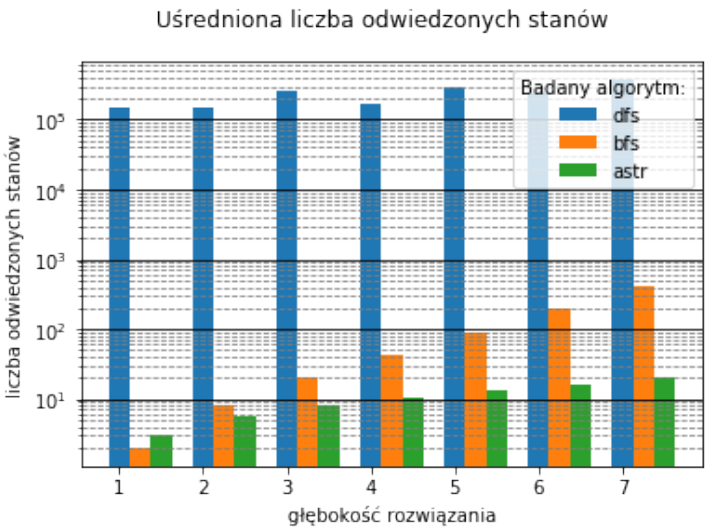
— długość ścieżki,
— ilość odwiedzonych stanów,
— ilość przetworzonych stanów,
— maksymalny stopień rekursji,
— czas przetwarzania algorytmu.

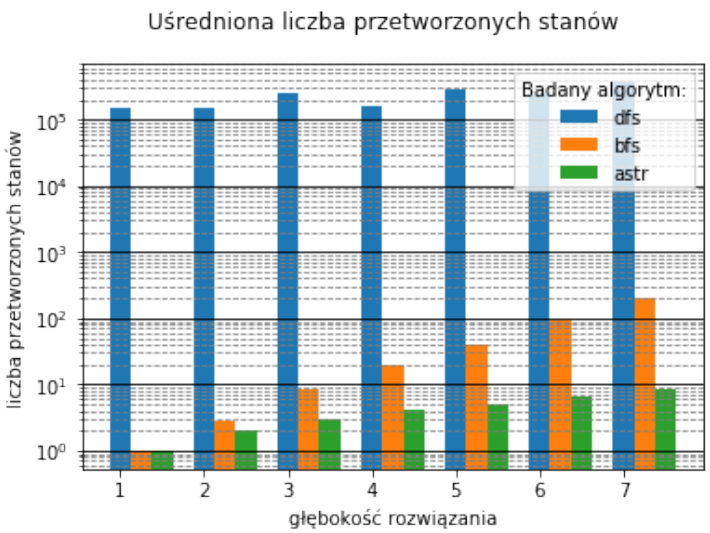
5. Wyniki

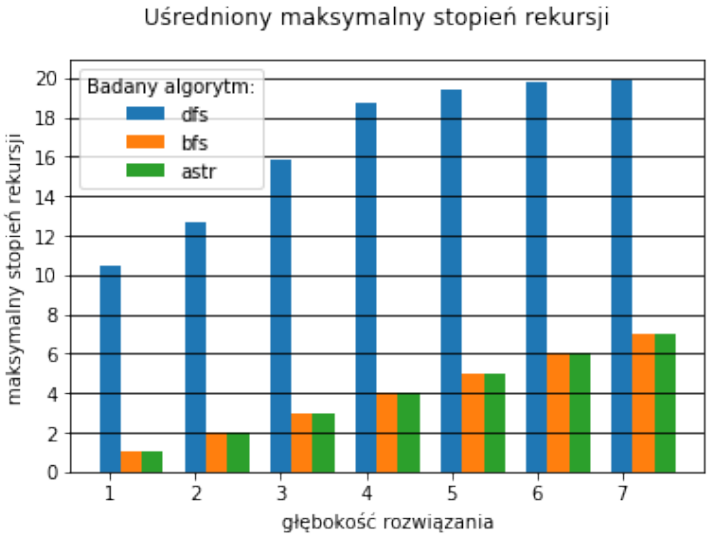
Wyniki zostaną zaprezentowane w następującej kolejności:

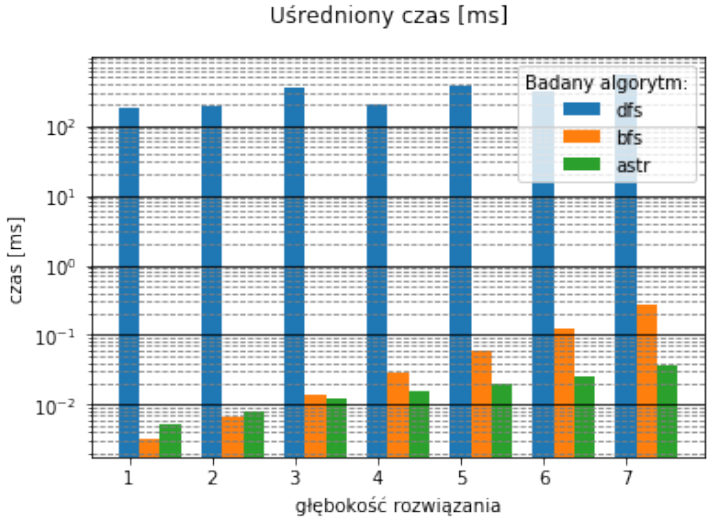
1. Porównanie wszystkich algorytmów
2. Wpływ wybranego porządku przeszukiwania dla algorytmów ślepych
   — BFS
   — DFS
3. Wpływ wybranej metryki dla algorytmu A*

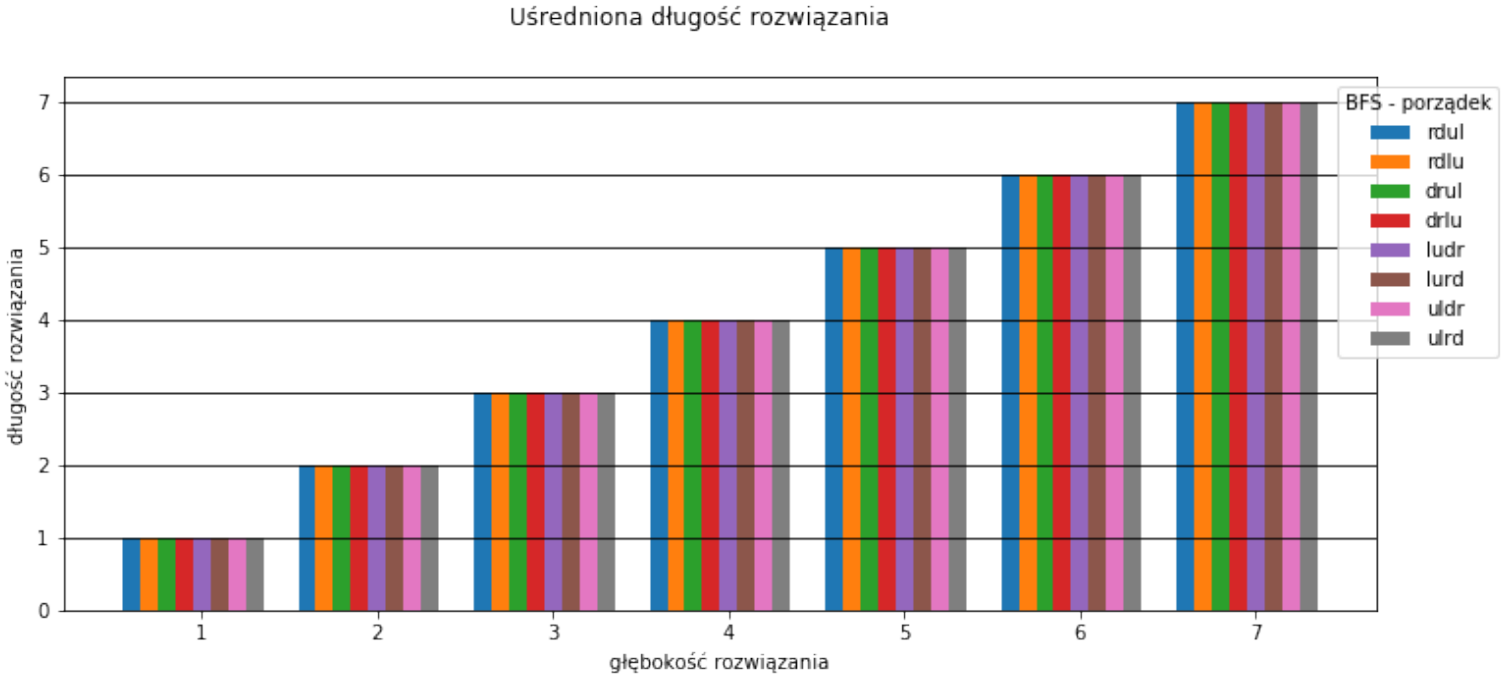

Uśredniona długość rozwiązania
Badany algorytm:
dfs
bfs
astr
długość rozwiązania
głębokość rozwiązania

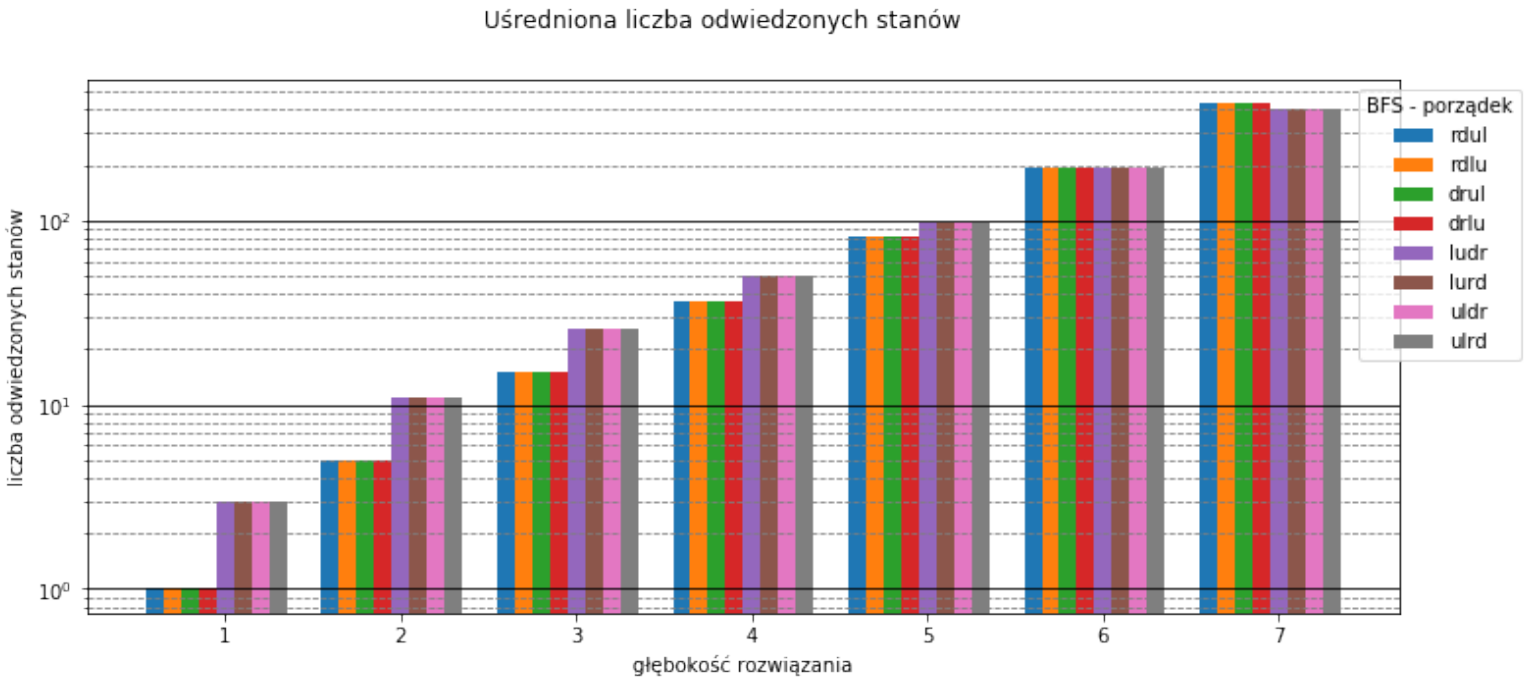

Uśredniona liczba odwiedzonych stanów
Badany algorytm:
dfs
bfs
astr
liczba odwiedzonych stanów
głębokość rozwiązania

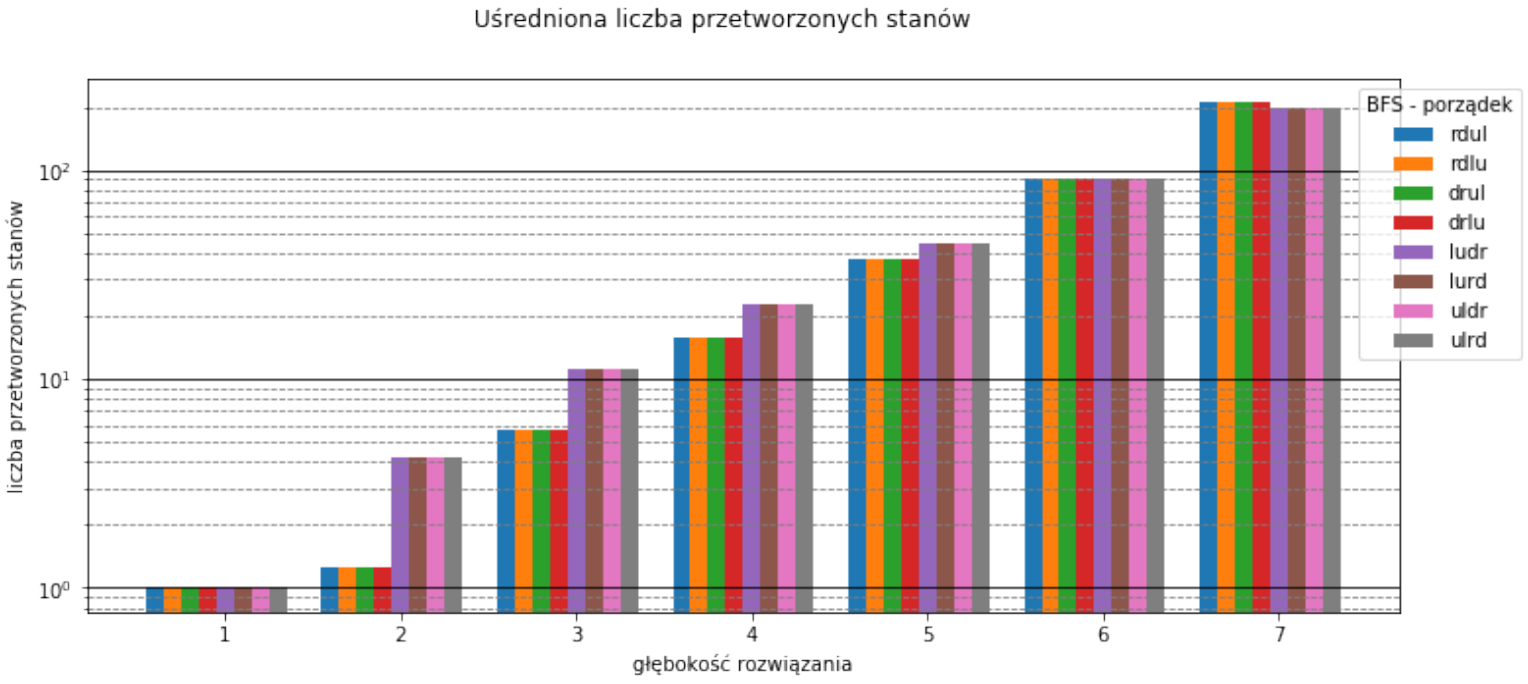

Uśredniona liczba przetworzonych stanów
Badany algorytm:
dfs
bfs
astr
liczba przetworzonych stanów
głębokość rozwiązania

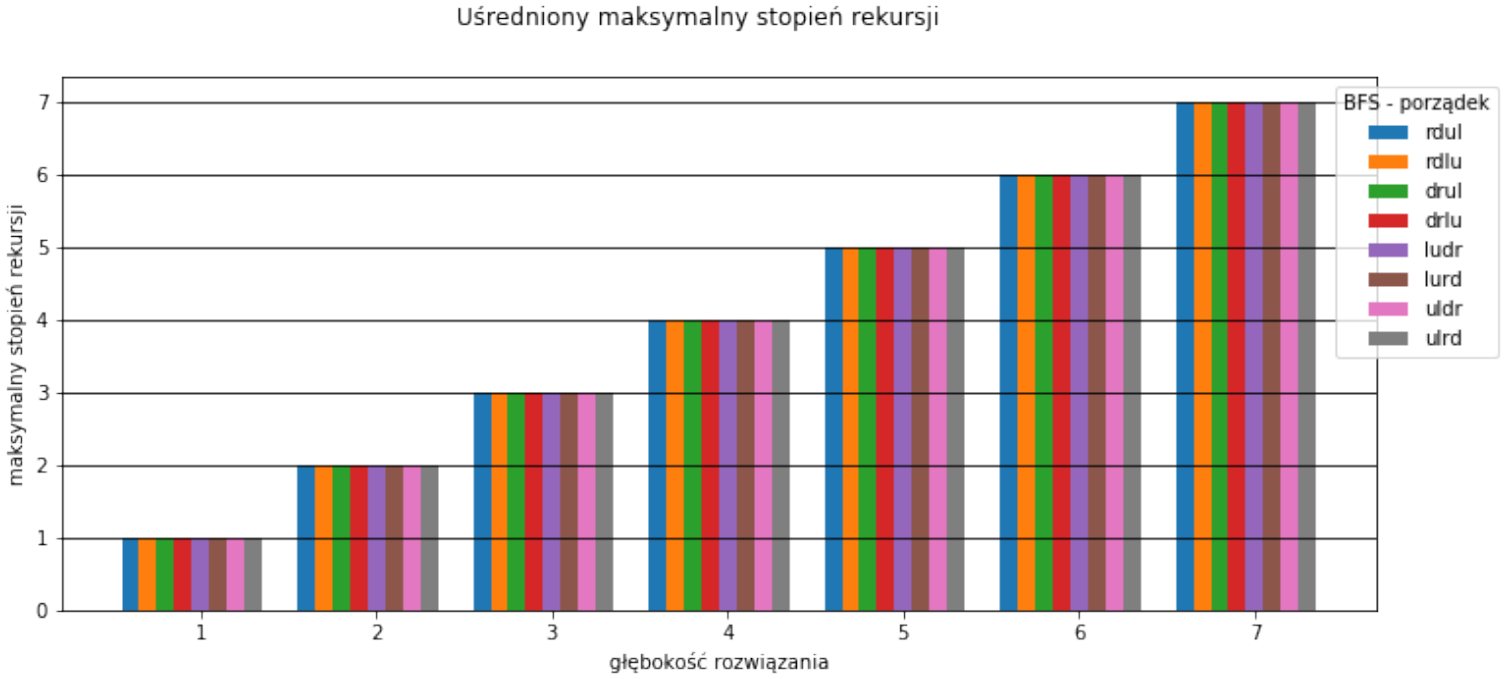

Uśredniony maksymalny stopień rekursji
Badany algorytm:
dfs
bfs
astr
maksymalny stopień rekursji
głębokość rozwiązania

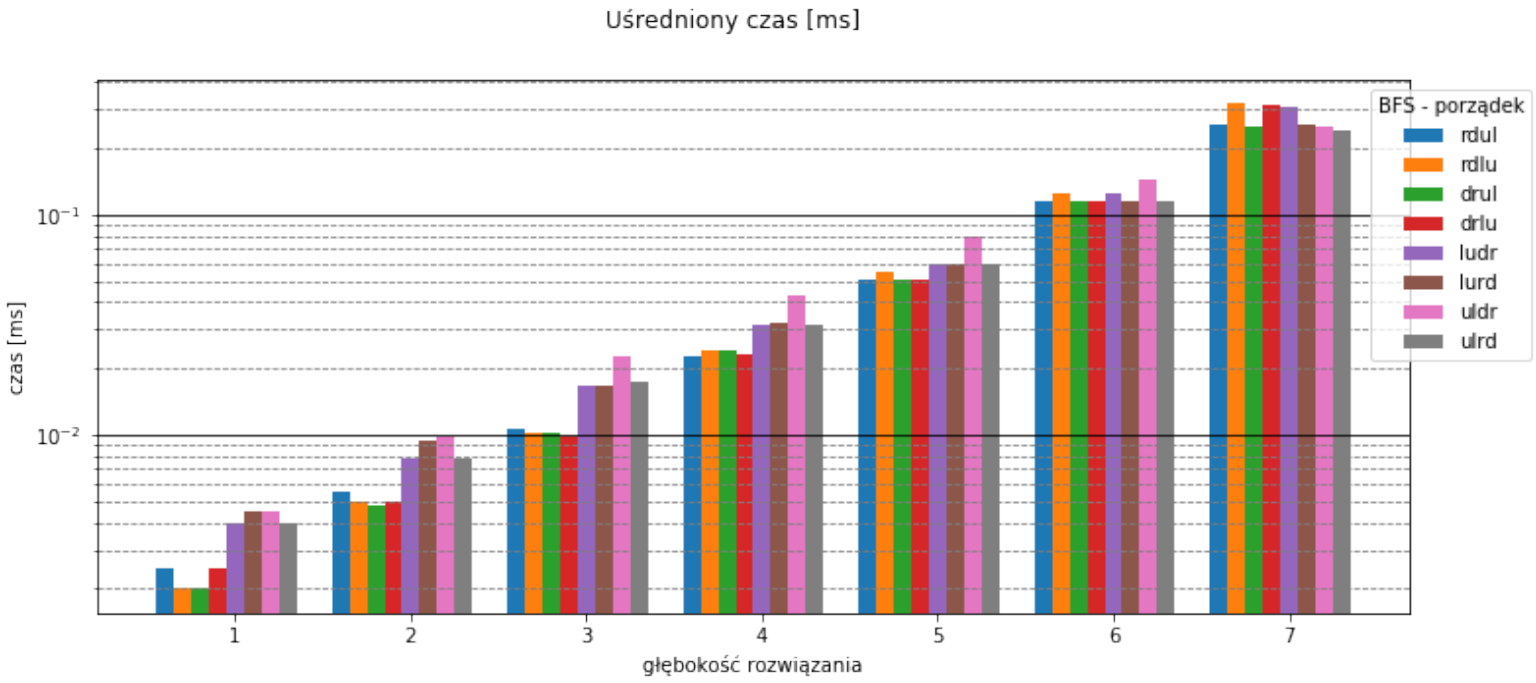

Uśredniony czas [ms]
Badany algorytm:
dfs
bfs
astr
czas [ms]
głębokość rozwiązania

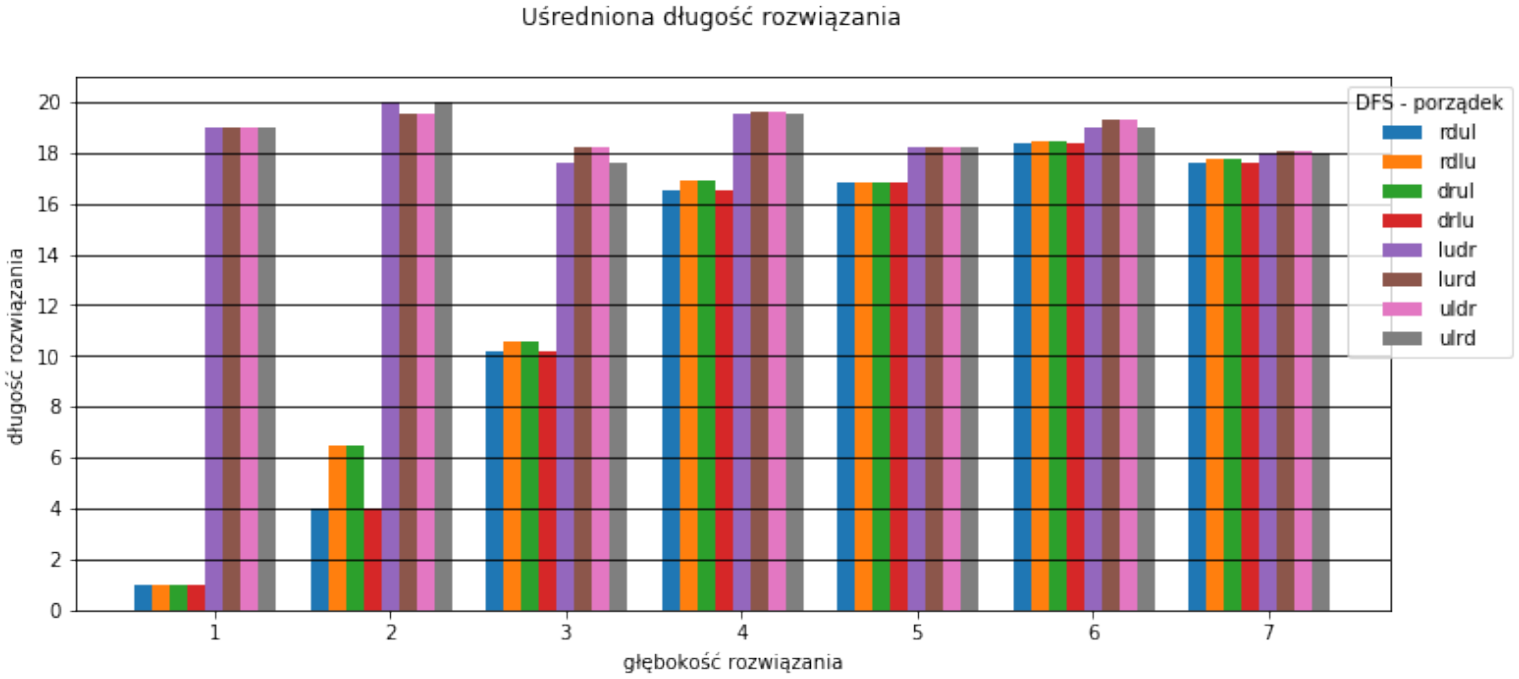

Uśredniona długość rozwiązania
długość rozwiązania
głębokość rozwiązania
BFS - porządek
rdul
rdlu
drul
drlu
ludr
lurd
uldr
ulrd

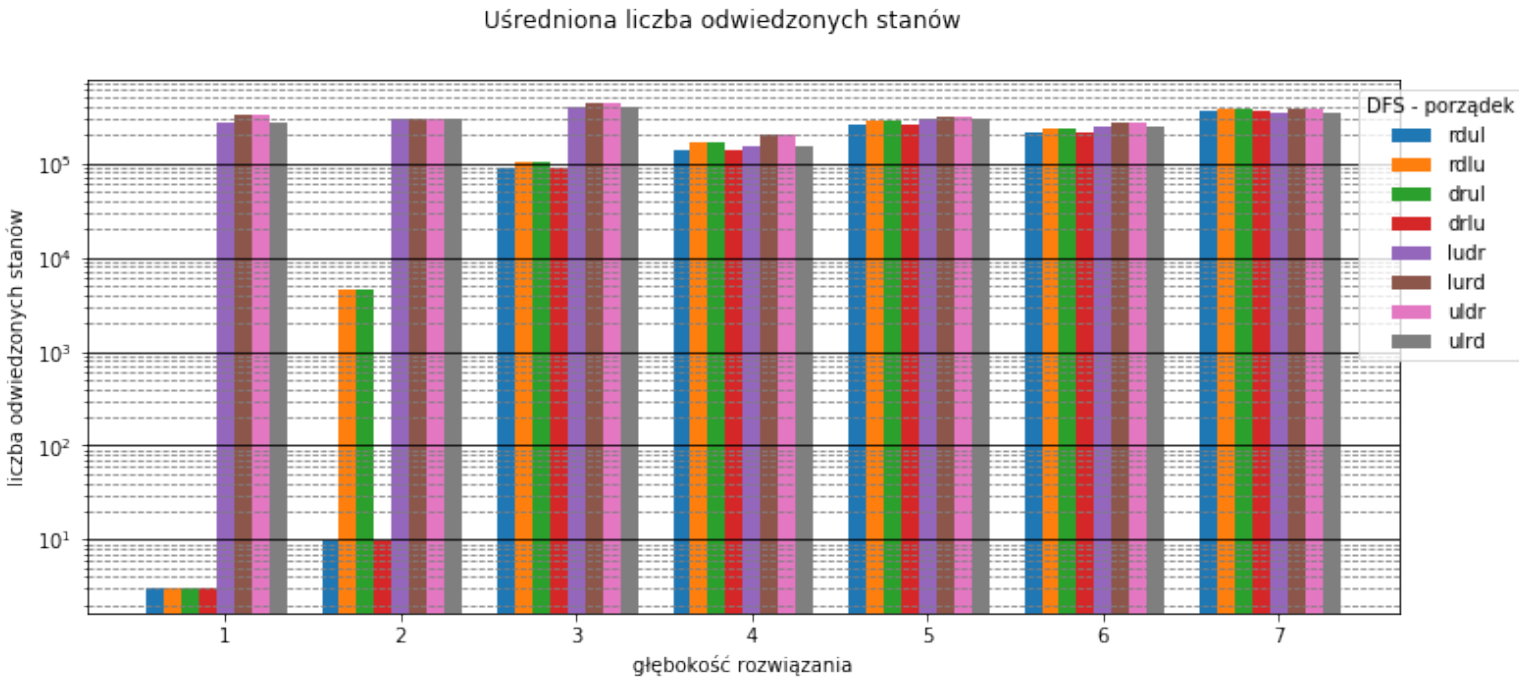

Uśredniona liczba odwiedzonych stanów
liczba odwiedzonych stanów
głębokość rozwiązania
BFS - porządek
rdul
rdlu
drul
drlu
ludr
lurd
uldr
ulrd

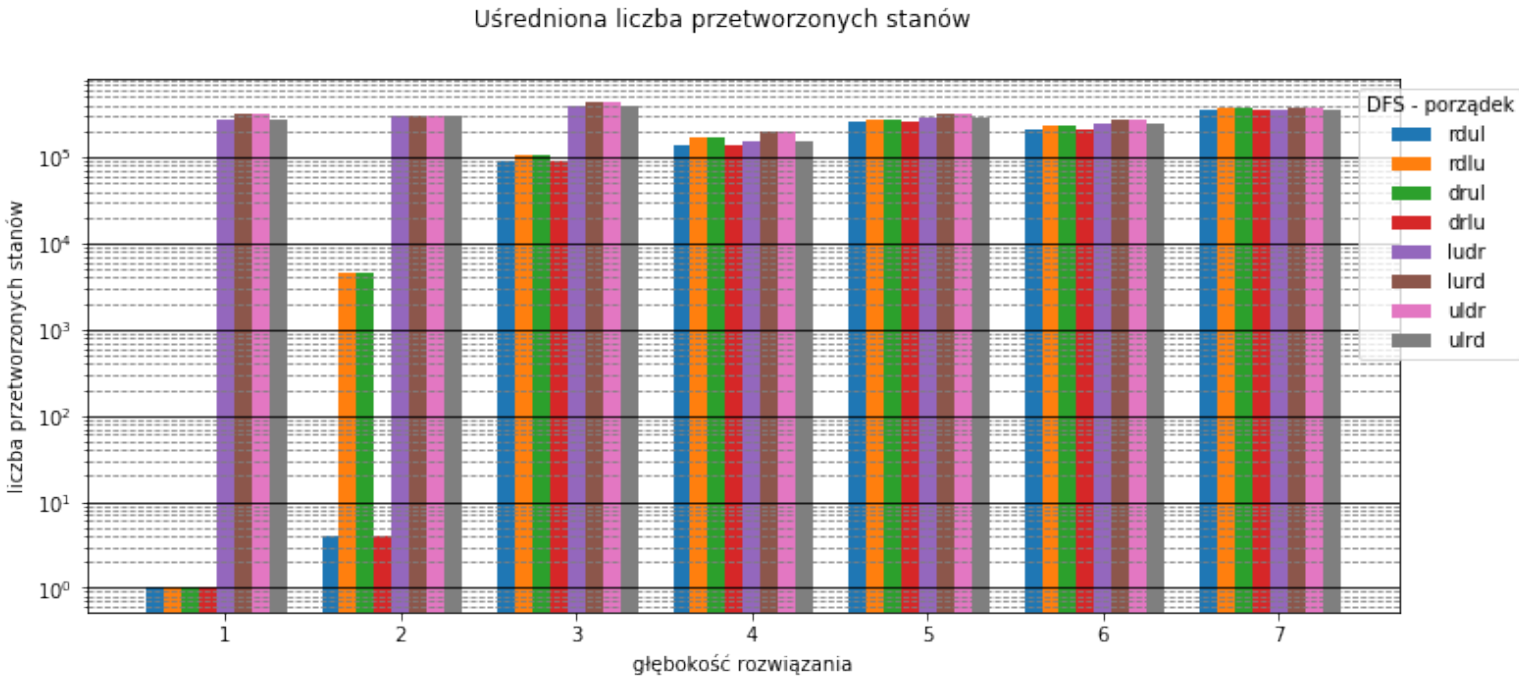

Uśredniona liczba przetworzonych stanów
liczba przetworzonych stanów
głębokość rozwiązania
BFS - porządek
rdul
rdlu
drul
drlu
ludr
lurd
uldr
ulrd

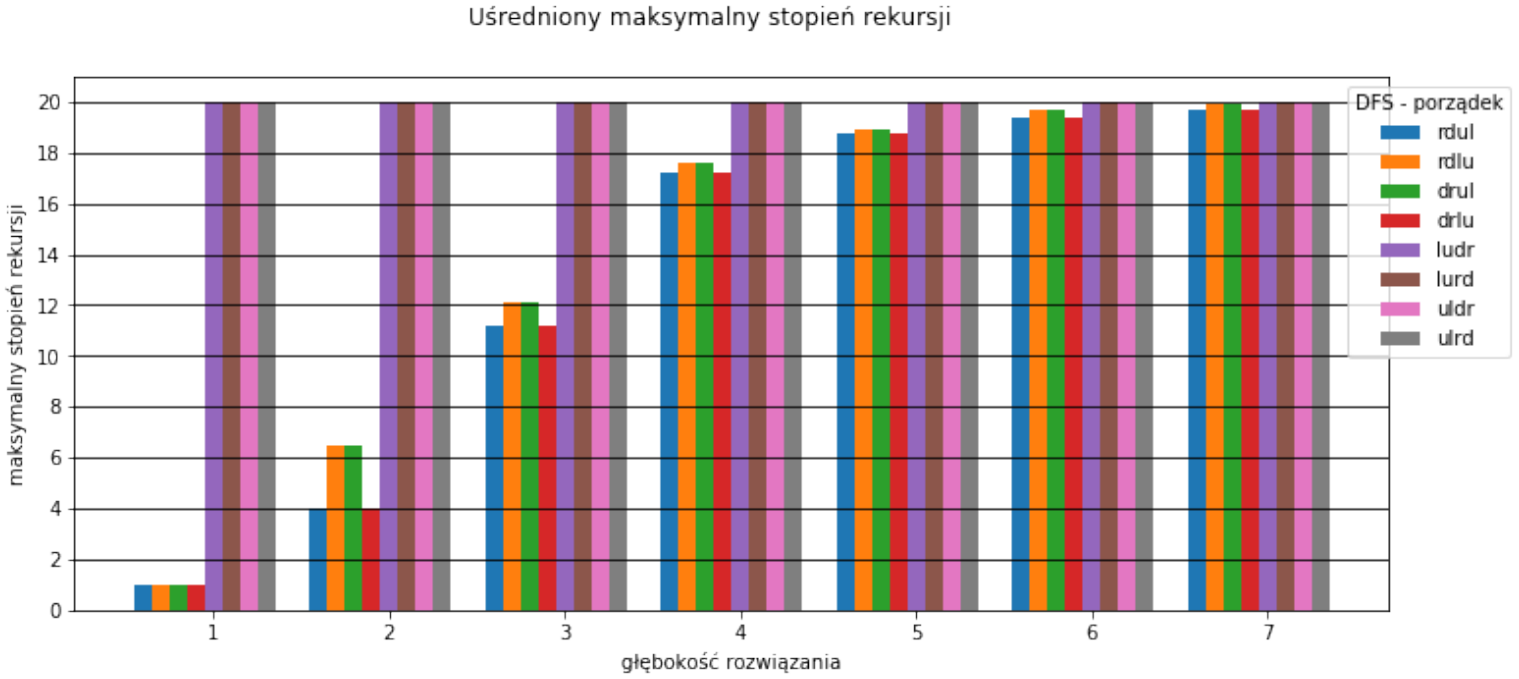

Uśredniony maksymalny stopień rekursji
maksymalny stopień rekursji
głębokość rozwiązania
BFS - porządek
rdul
rdlu
drul
drlu
ludr
lurd
uldr
ulrd


Uśredniony czas [ms]
czas [ms]
głębokość rozwiązania
BFS - porządek
rdul
rdlu
drul
drlu
ludr
lurd
uldr
ulrd

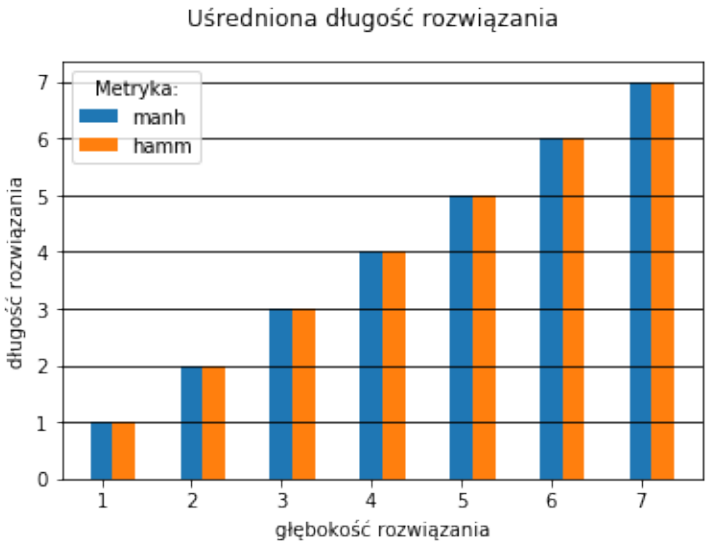

Uśredniona długość rozwiązania
długość rozwiązania
głębokość rozwiązania
DFS - porządek
rdul
rdlu
drul
drlu
ludr
lurd
uldr
ulrd

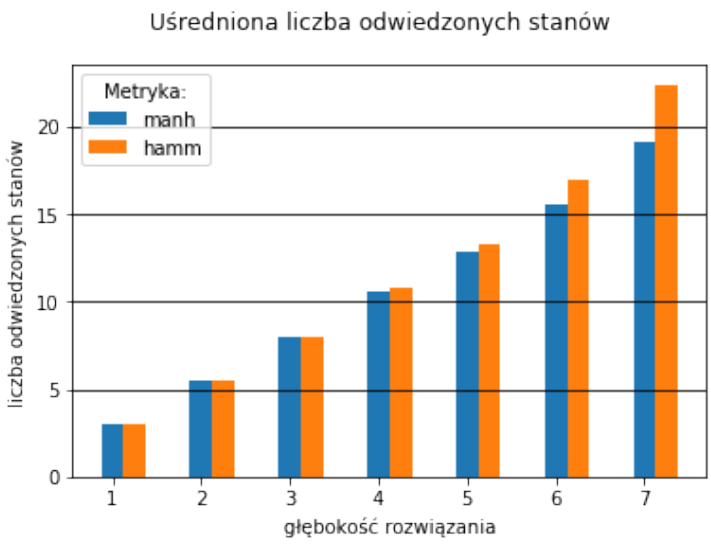

Uśredniona liczba odwiedzonych stanów
liczba odwiedzonych stanów
głębokość rozwiązania
DFS - porządek
rdul
rdlu
drul
drlu
ludr
lurd
uldr
ulrd

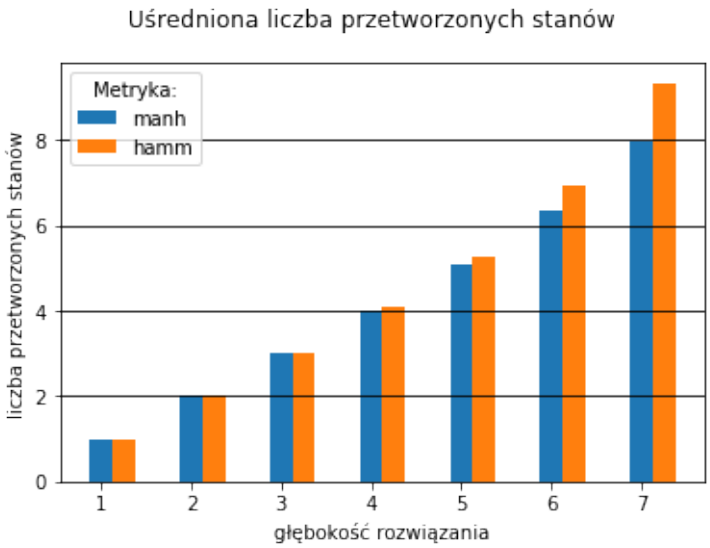

Uśredniona liczba przetworzonych stanów
liczba przetworzonych stanów
głębokość rozwiązania
DFS - porządek
rdul
rdlu
drul
drlu
ludr
lurd
uldr
ulrd

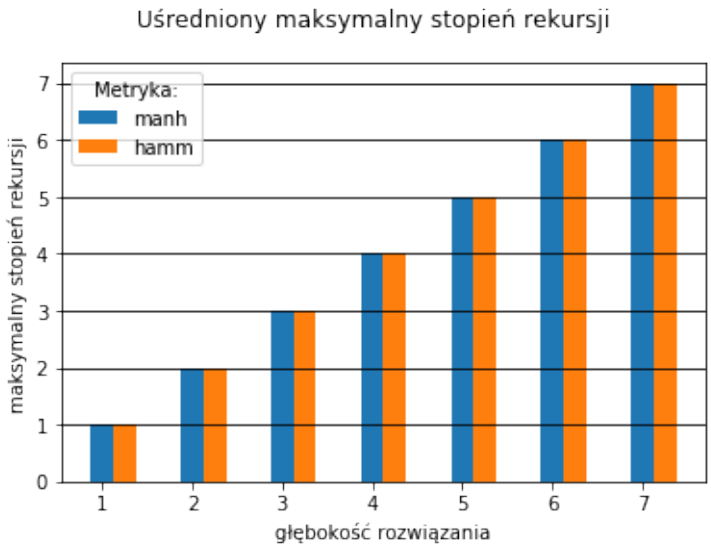

Uśredniony maksymalny stopień rekursji
maksymalny stopień rekursji
głębokość rozwiązania
DFS - porządek
rdul
rdlu
drul
drlu
ludr
lurd
uldr
ulrd

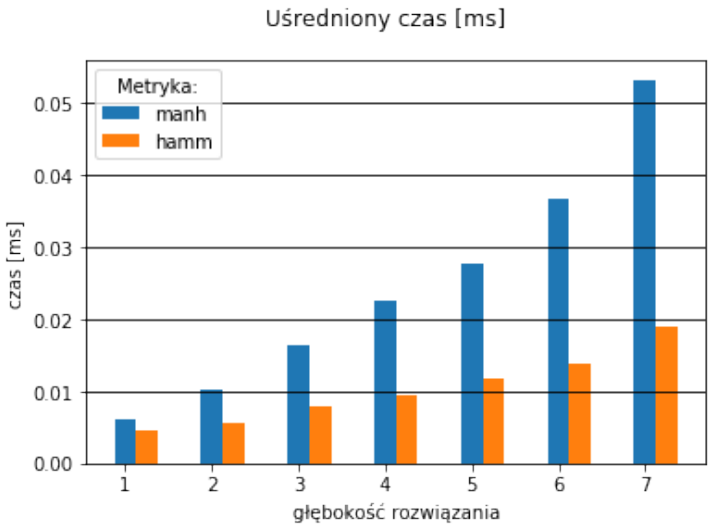

Uśredniony czas [ms]
czas [ms]
głębokość rozwiązania
DFS - porządek
rdul
rdlu
drul
drlu
ludr
lurd
uldr
ulrd


Uśredniona długość rozwiązania
Metryka:
manh
hamm
długość rozwiązania
głębokość rozwiązania


Uśredniona liczba odwiedzonych stanów
Metryka:
manh
hamm
liczba odwiedzonych stanów
głębokość rozwiązania


Uśredniona liczba przetworzonych stanów
Metryka:
manh
hamm
liczba przetworzonych stanów
głębokość rozwiązania


Uśredniony maksymalny stopień rekursji
Metryka:
manh
hamm
maksymalny stopień rekursji
głębokość rozwiązania


Uśredniony czas [ms]
Metryka:
manh
hamm
czas [ms]
głębokość rozwiązania

6. Dyskusja

6.1. Optymalność badanych algorytmów

Spośród testowanych algorytmów BFS i A* znajdowały zawsze rozwiązanie optymalne. W wypadku DFS w ogólności nie jest to prawdą. BFS sprawdza najpierw wszystkie stany znajdujące się w takiej samej odległości od stanu wyjściowego, dopiero potem schodzi "głębiej". Dzięki takiej taktyce odtworzenie ścieżki zawsze da optymalny wynik. Dla porównania, droga tworzona przez DFS jest zupełnie inna - algorytm dopóki nie napotka przeszkody (w wypadku układanki - np. brak możliwego ruchu w danym kierunku) będzie wchodził wgłąb grafu, przez co rozwiązania - nawet te znajdujące się naprawdę blisko - mogą zostać przeoczone.

Algorytm A* jest zawsze optymalny przy założeniu, że użyliśmy prawidłowej heurystyki, tzn takiej która nie przeszacowuje faktycznego kosztu.

6.2. Ilość przetwarzanych danych

DFS po raz kolejny niechlubnie wyróżnia się pośród badanych algorytmów - przetwarza zdecydowanie najwięcej stanów. Najlepiej w tym zestawieniu wypada algorytm A*, w szczególności z metryką Manhattan - można to zauważyć zwłaszcza dla głębokości większej niż 5.

6.3. Szybkość działania

Dla głębokości rzędów 1 i 2 BFS wypada nieznacznie lepiej, jednakże tendencja ta zaczyna się odwracać już w momencie, gdy szukamy rozwiązań układanek o głębokościach większych lub równych 3. Co ciekawe, W zestawieniu średnich czasów dla metryk Manhattan zostaje daleko w tyle za metryką Hamminga i ten trend wydaje się być rosnący. Te dwa spostrzeżenia - BFS szybszy dla głębokości mniejszej niż 3 oraz A* z metryką Hamminga szybsza niż A* z metryką Manhattan - mają wspólne wyjaśnienie. Przy małej głębokości układanki duże znaczenie ma koszt umieszczenia stanu na liście frontier. W implementacji stworzonej na potrzeby zadania dla BFS koszt ten jest równy kosztowi wstawienia elementu do kolekcji. W wypadku algorytmu A* koszt wstawienia jest dodatkowo powiększony o obliczenia związane z zastosowaną metryką, przy czym obliczenie odległości Hamminga jest znacznie tańsze niż obliczenie odległości taksówkowej. Aby dać możliwość głosu obrony metryki Manhattan w znajdowaniu najkrótszej drogi w grafie przeprowadzono dodatkowe badanie na układance o głębokości 20. W tej próbie odnotowano następujące czasy:

— Manhattan: 59.688 ms.
— Hamming: 581.177 ms.

Jest to niestety wyłącznie jedna próbka, zatem nie można na jej podstawie udowodnić tezy, że metryka Manhattan dla dużych głębokości pozwala znaleźć rozwiązanie szybciej niż metryka Hamminga, jednakże na pewno zachęca do przeprowadzenia dalszych badań na układankach o głębokościach większych niż 7.

Na temat szybkości algorytmu DFS można powiedzieć niestety tyle, że w ogólności odstaje ona od czasów wykonania algorytmów BFS i A*. Dobre rezultaty można zaobserwować jedynie w wypadkach korzystnych szeregów przeszukiwania.

7. Wnioski

— BFS i A* to algorytmy zupełne i optymalne,
— Nie należy stosować DFS do przeszukiwania grafu przestrzeni stanów,
— Zastosowanie przeszukiwania heurystycznego znacznie redukuje liczbę odwiedzonych i przetworzonych stanów.

Literatura

[1] *http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html*
[2] *http://wms.mat.agh.edu.pl/~zankomar/wyklady/Wyklad8.htm*
[3] *https://pandas.pydata.org/pandas-docs/stable/comparison_with_sql.html*