Informatyka, studia niestacjonarne semestr V

Sztuczna inteligencja i Systemy Ekspertowe **2018/2019**

Prowadzący: dr Nowak niedziela,

Data oddania: _____ Ocena: _____

Marcin Pajkowski 211968
Rafał Warda 214067

Przeszukiwanie przestrzeni stanów

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. Układanka posiada wymiar MxN (najczęściej są to układy o równych wymiarach - 4x4 lub 3x3). Jedno pole jest zawsze wolne. Gracz wykonuje ruchy przemieszczając klocki z wykorzystaniem wolnego pola. Elementy układanki mogą reprezentować liczby - wówczas zadaniem gracza jest ułożenie elementów w odpowiednim szeregu.

2.2. Złożoność problemu i rozwiązywalność

Ilość stanów każdej układanki można wyznaczyć za pomocą wzoru:

$$iloscStanw = iloscPl!$$

Dla układanki 4x4 będzie to liczba

$$16! = 2.0922789888 * 10^{13}$$

Jednakże 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 pochodzące poprzez przesuwanie klocków w obrębie wolnego pola, począwszy od układu docelowego, są układami parzystymi.

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

Do przeszukiwania grafu przestrzeni stanów w celu znalezienia rozwiązania układanki zostaną zaprezentowane następujące podejścia:

— 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 (inaczej metryka taxicab lub metryka miejska).

Algorytm **BFS** przeszukuje graf **wszerz** - w pierwszej kolejności odwiedzany jest stan początkowy, następnie sąsiedzi stanu początkowego, dalej sąsiedzi sąsiadów rozwiniętych w poprzednich iteracjach - do momentu znalezienia stanu wzorcowego.

Inny algorytm z tej grupy - **DFS** przeszukuje graf **w głąb** - w pierwszej kolejności odwiedzany jest stan początkowy, następnie sąsiedzi stanu początkowego (zgodnie z podanym wcześniej porządkiem przeszukiwania). Następnie odwiedzany jest stan-dziecko będący blabla (TODO)

Zasadniczą różnicą między tymi dwoma podejściami jest to, że klasyczne algorytmy poszukują rozwiązanie zgodnie z określonym porządkiem i nie próbują aproksymować zasadności następnego ruchu. W praktyce nie musi to oznaczać, że algorytmy klasyczne są nieoptymalne - przykładowo algorytm BFS znajduje zawsze rozwiazanie optymalne.

3. Implementacja

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++.

4. Materiały i metody

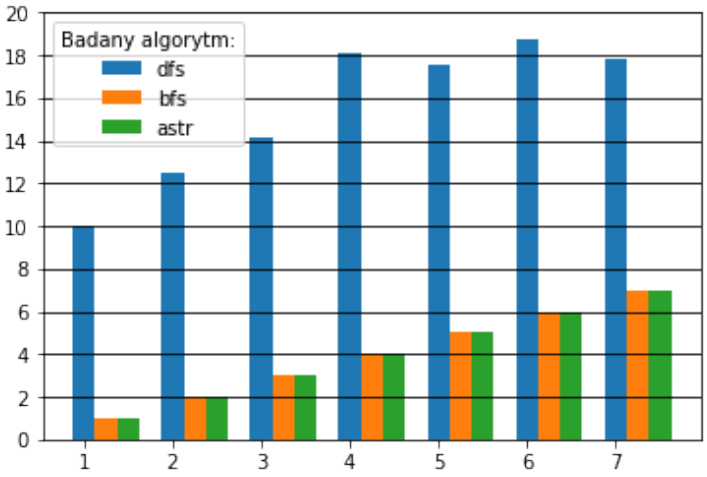
Do zrealizowania zadania zostały użyte następujące programy i skrypty wspomagające dostarczone razem z kartą przedmiotu:

— Generator układanek,
— Walidator układanek,
— Wizualizator układanek,
— Skrypt uruchamiające program przeszukujący w trybie wsadowym (powłoka bash),
— Skrypt ekstraktujący dane wygenerowane podczas przeszukiwania (powłoka bash).

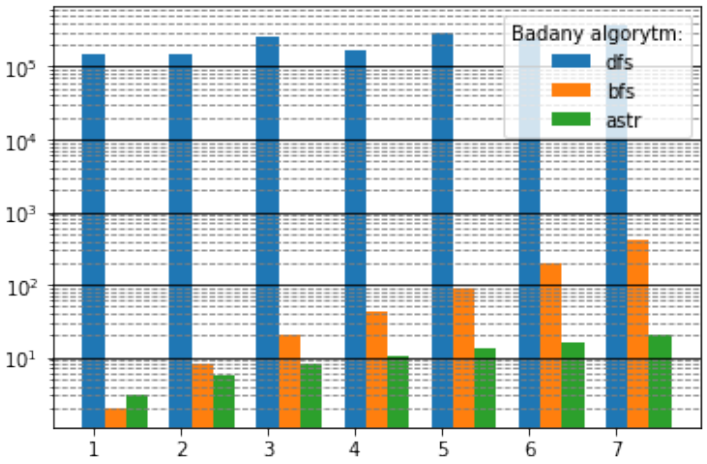
Do stworzenia pliku binarnego solvera z kodu źródłowego C++ użyto kompilatora Clang z pakietu LLVM. Prezentacja graficzna wyników została wstępnie przetworzona za pomocą oprogramowania Jupyter Notebook (kernel Python 3).

5. Wyniki

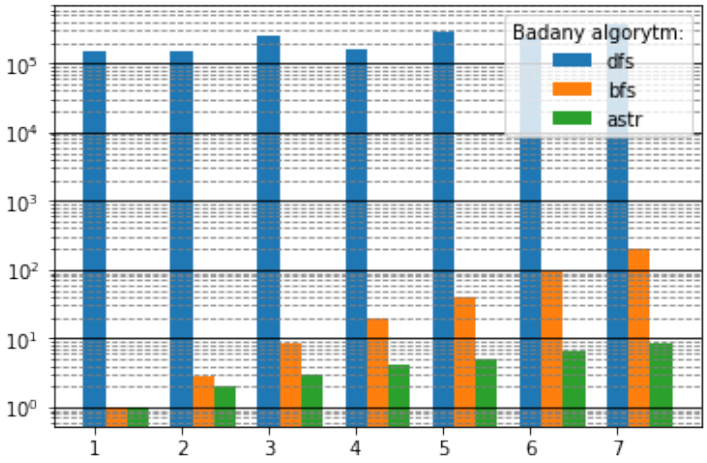
Uśredniona długość rozwiązania



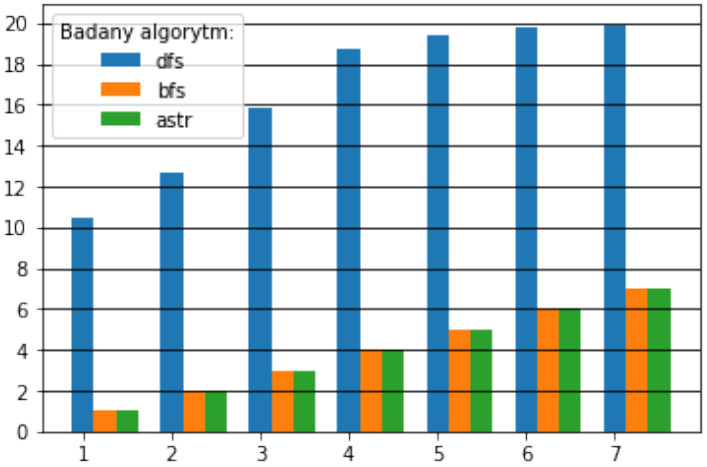
Uśredniona liczba odwiedzonych stanów



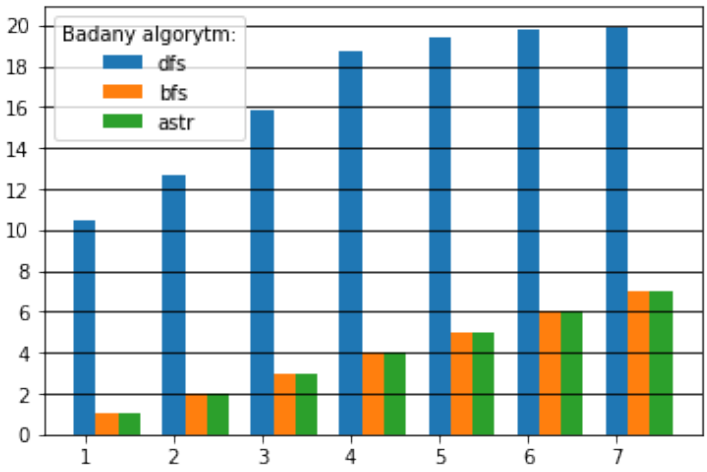
Uśredniona liczba przetworzonych stanów



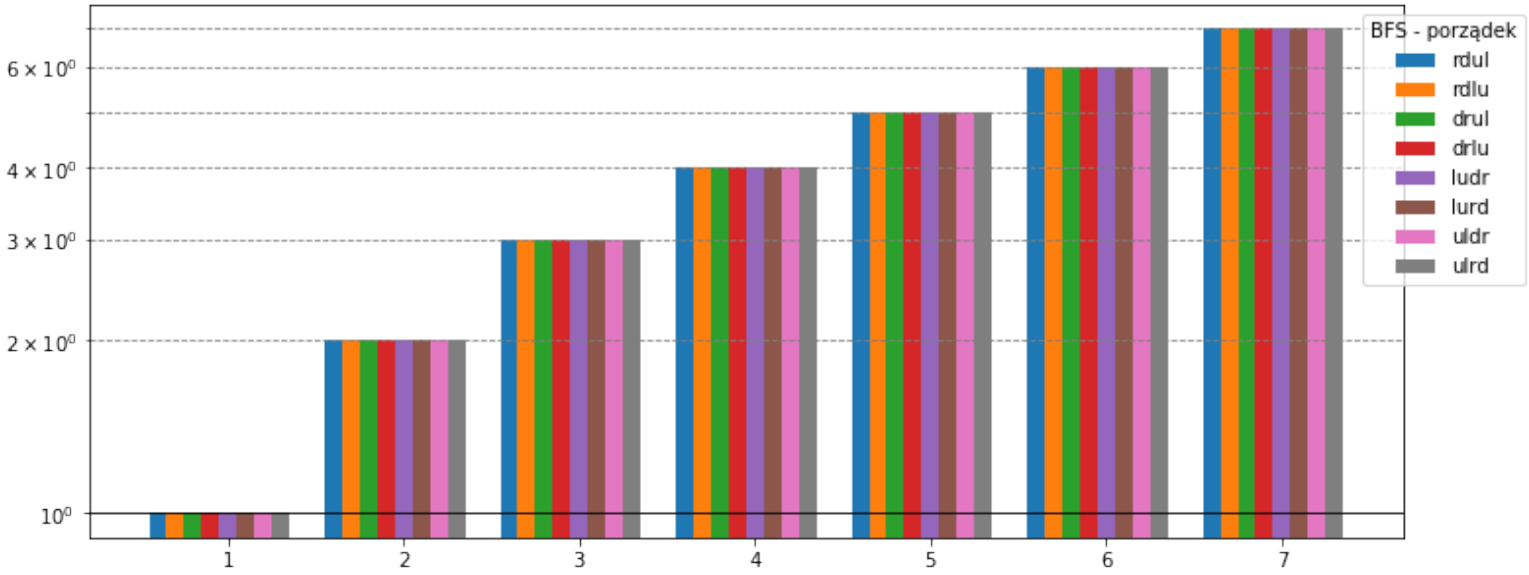
Uśredniony maksymalny stopień rekursji



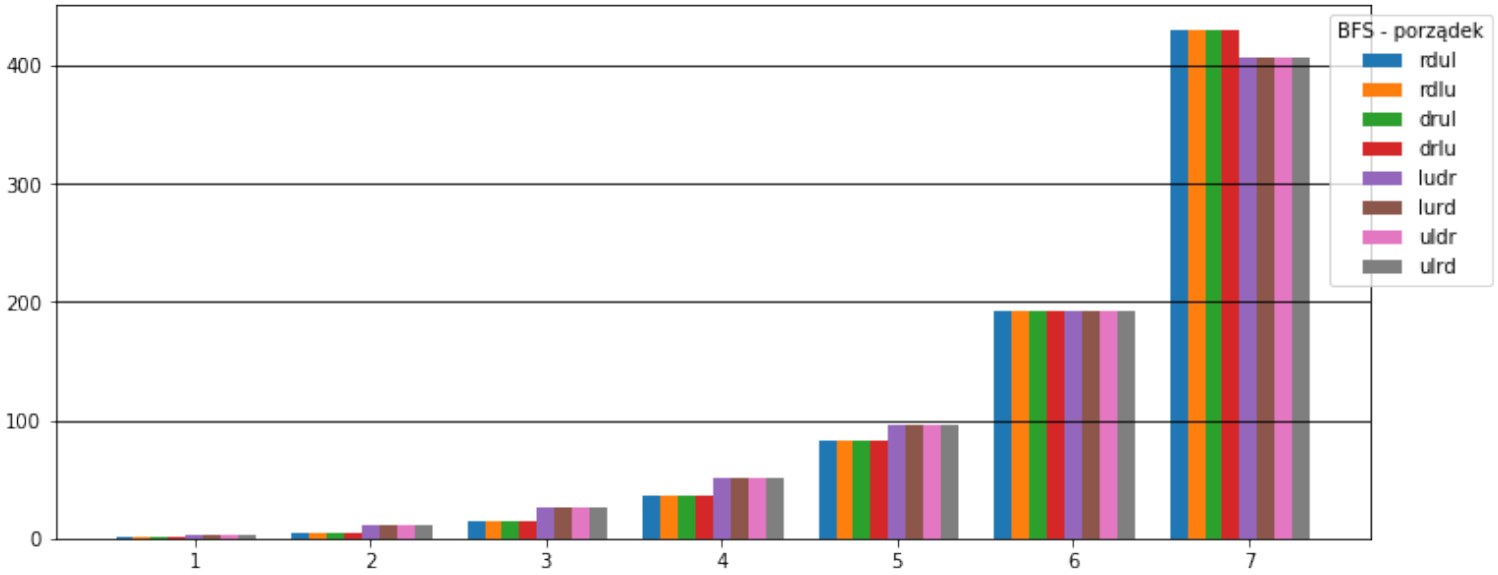
Uśredniony maksymalny stopień rekursji



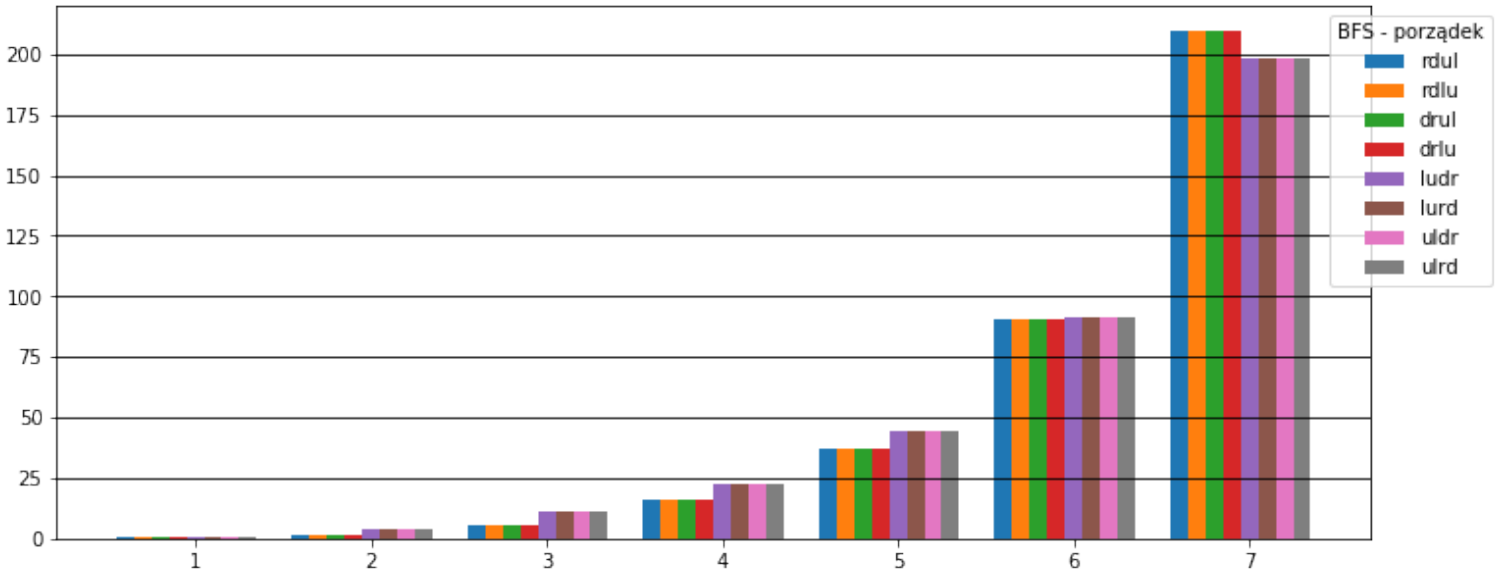
Uśredniona długość rozwiązania

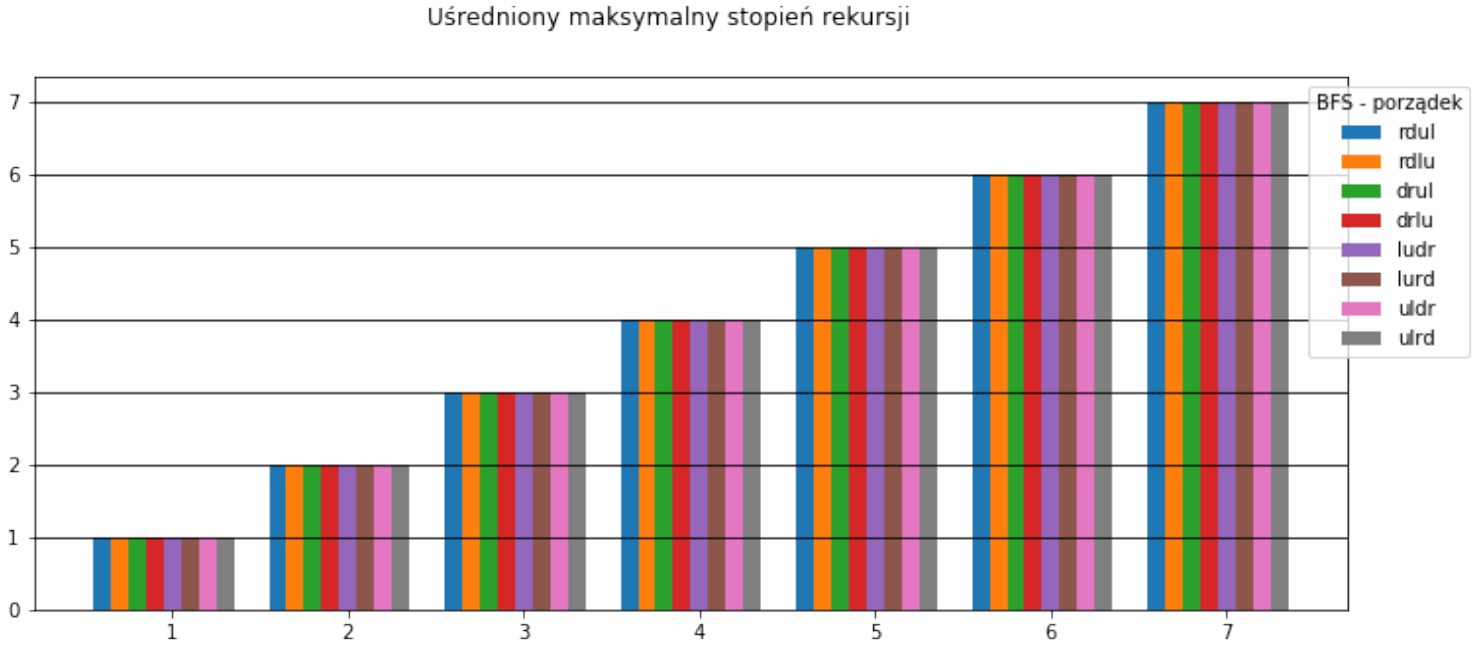


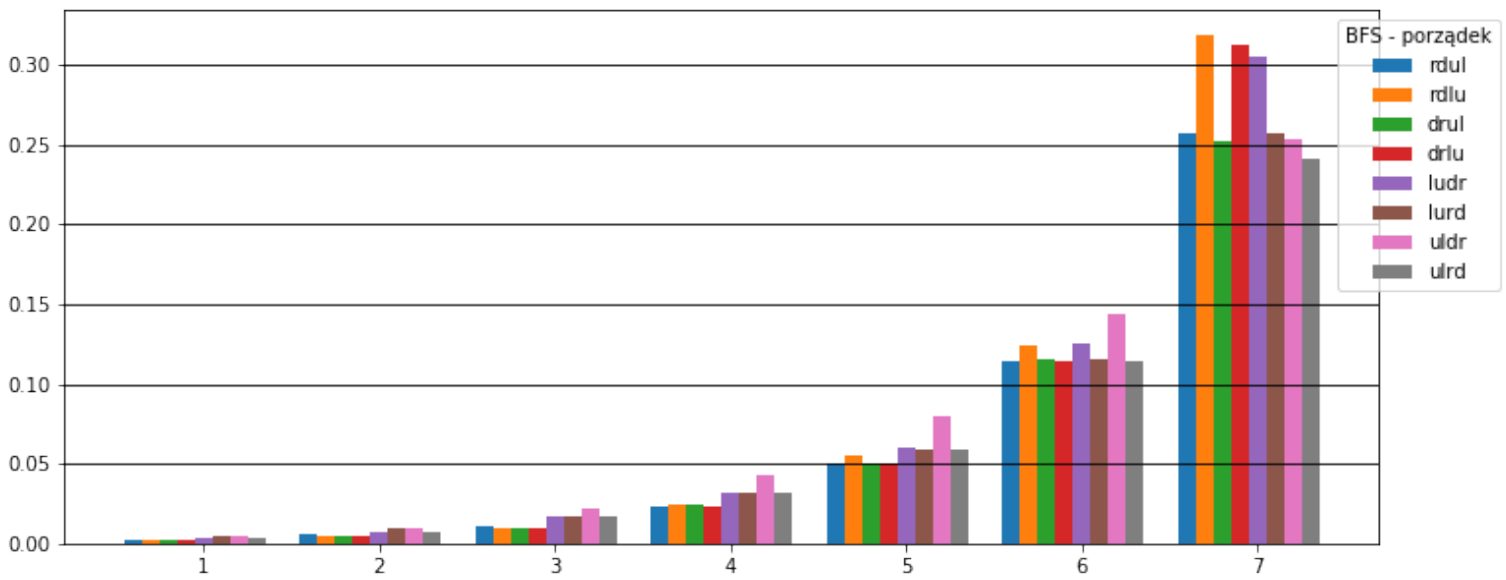
Uśredniona liczba odwiedzonych stanów



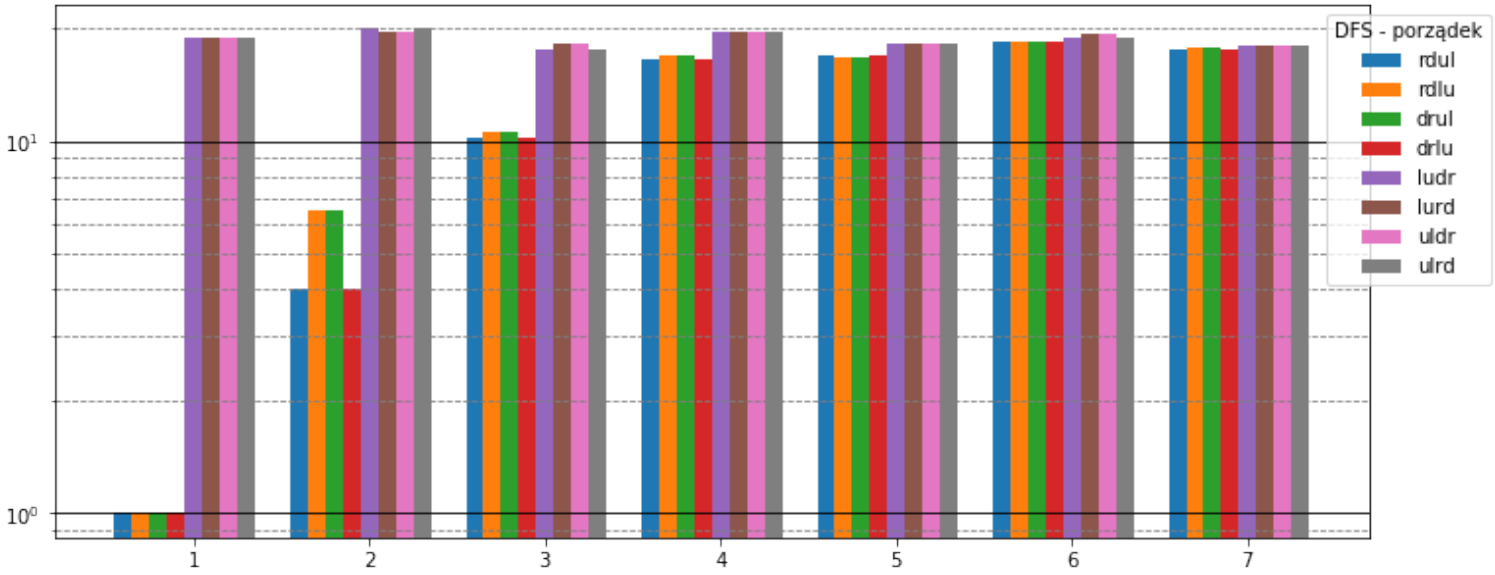
Uśredniona liczba przetworzonych stanów



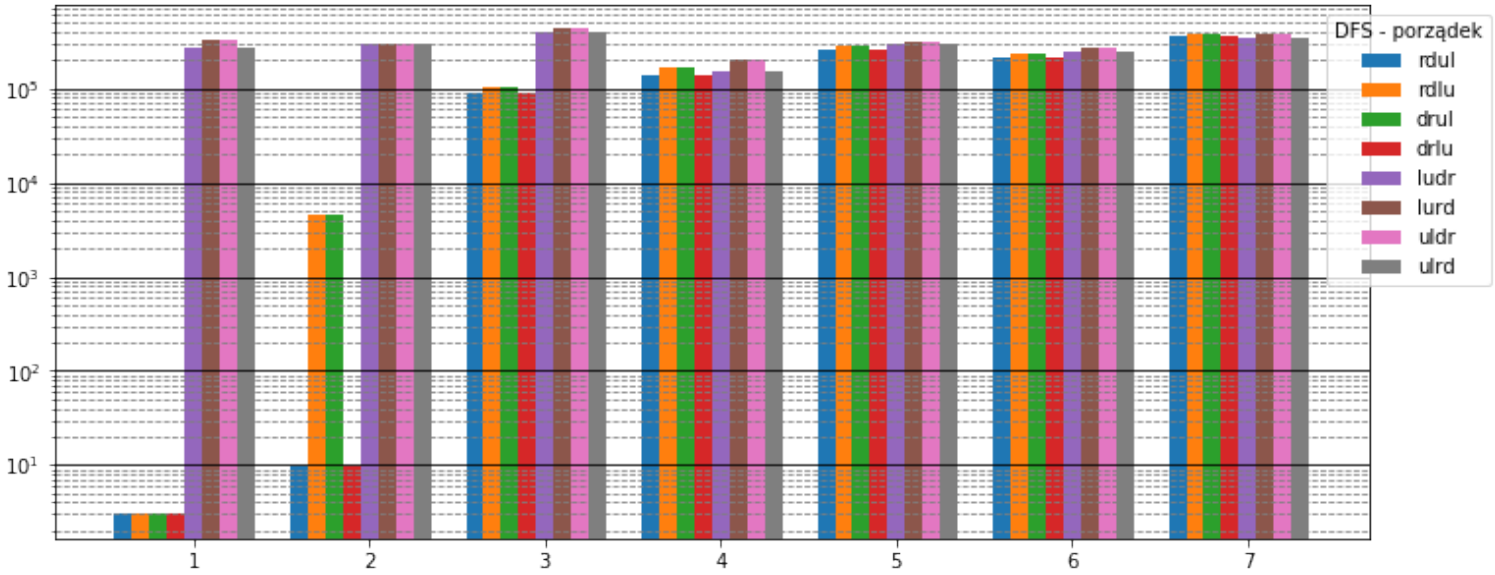

Uśredniony maksymalny stopień rekursji
BFS - porządek
rdul
rdlu
drul
drlu
ludr
lurd
uldr
ulrd
0
1
2
3
4
5
6
7
1
2
3
4
5
6
7


Uśredniony czas [ms]
BFS - porządek
rdul
rdlu
drul
drlu
ludr
lurd
uldr
ulrd
0.00
0.05
0.10
0.15
0.20
0.25
0.30
1
2
3
4
5
6
7

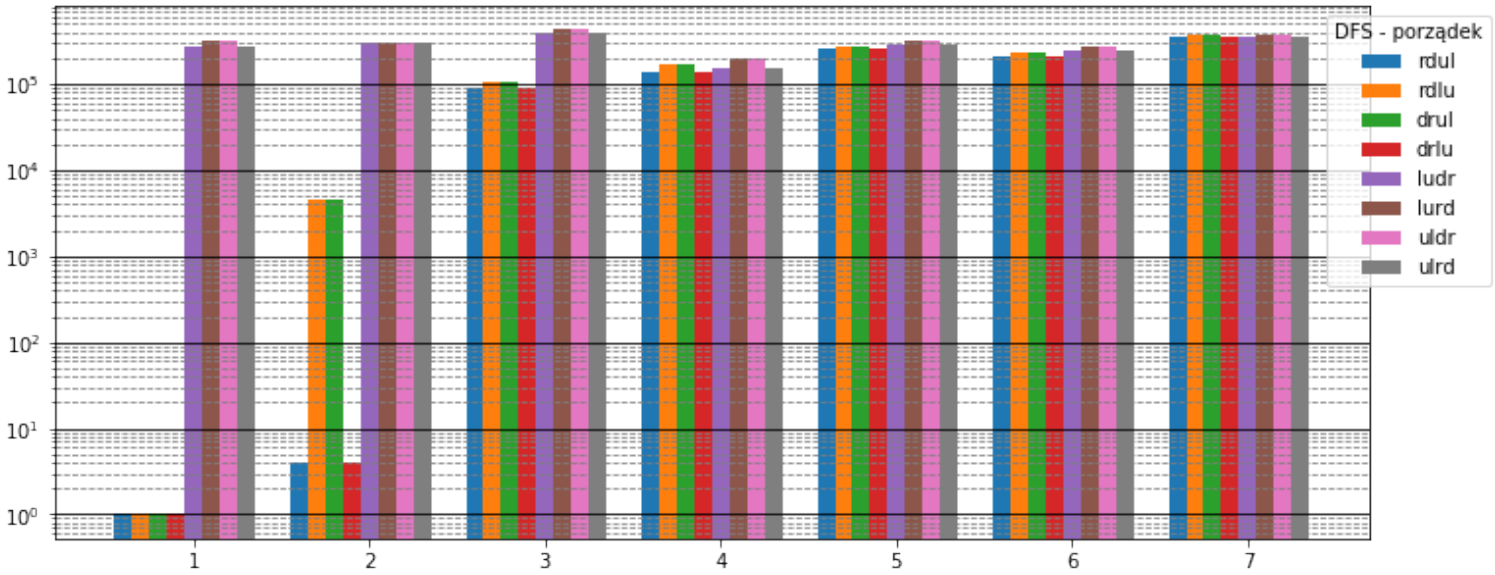
Uśredniona długość rozwiązania

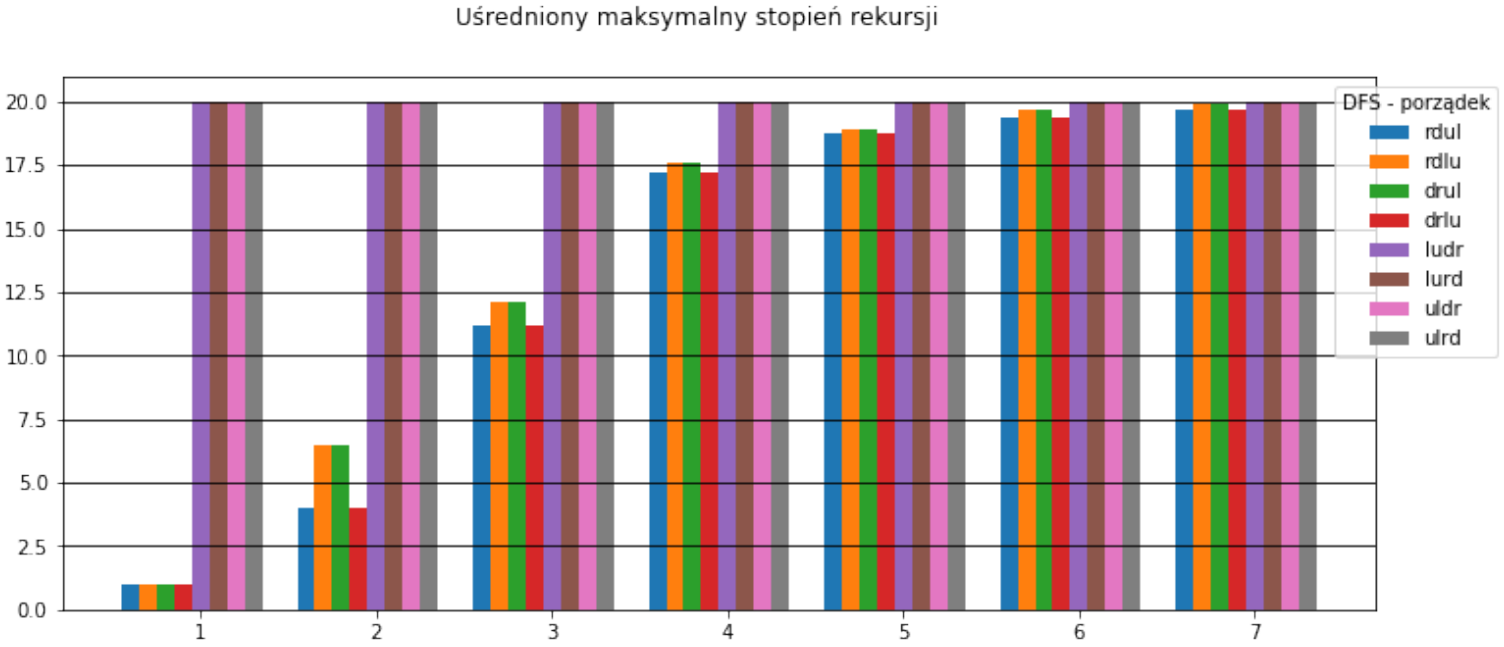


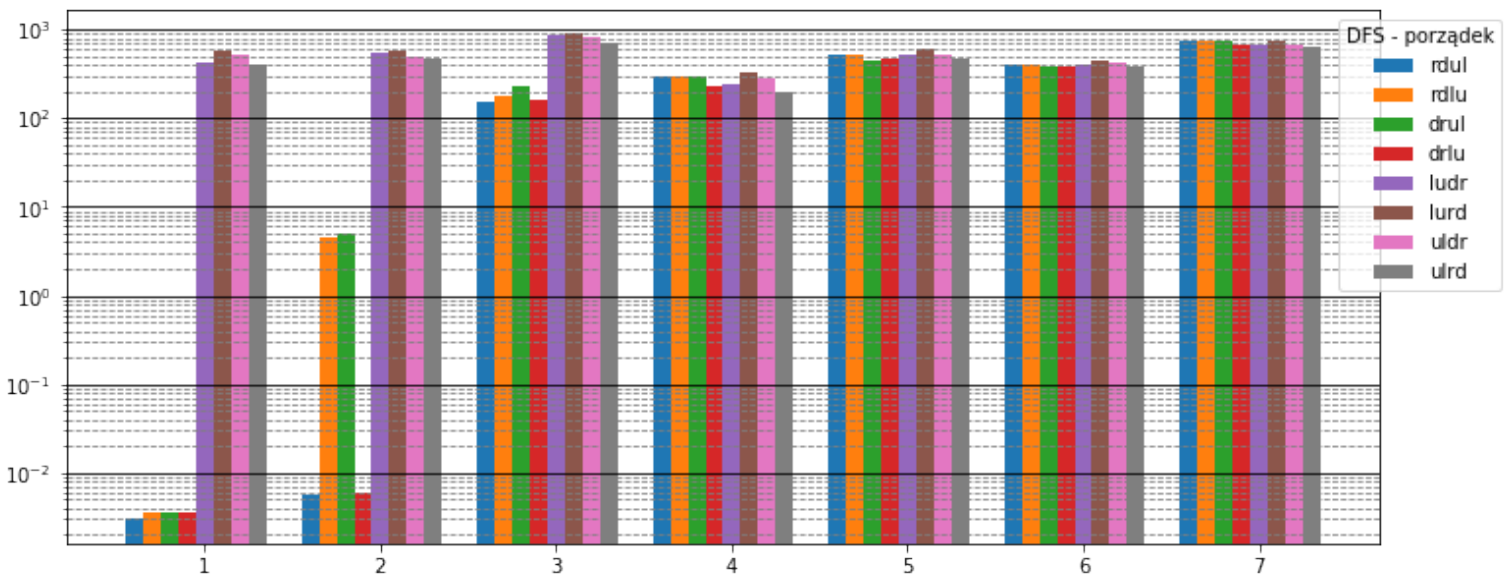
Uśredniona liczba odwiedzonych stanów



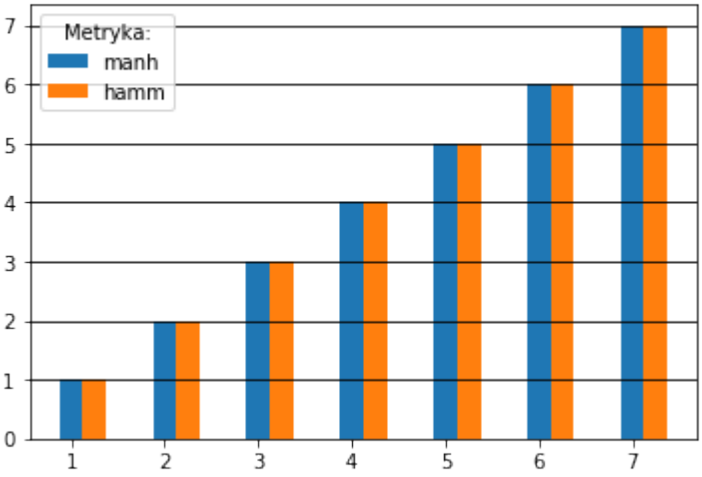
Uśredniona liczba przetworzonych stanów



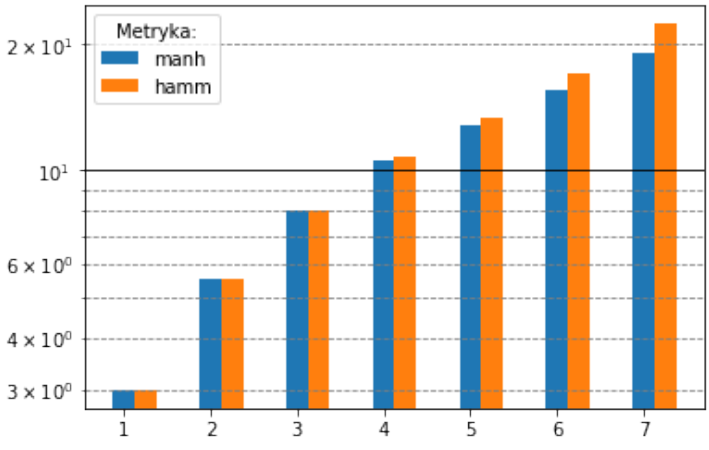

Uśredniony maksymalny stopień rekursji
DFS - porządek
rdul
rdlu
drul
drlu
ludr
lurd
uldr
ulrd
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
1
2
3
4
5
6
7


Uśredniony czas [ms]
DFS - porządek
rdul
rdlu
drul
drlu
ludr
lurd
uldr
ulrd
10^{3}
10^{2}
10^{1}
10^{0}
10^{-1}
10^{-2}
1
2
3
4
5
6
7

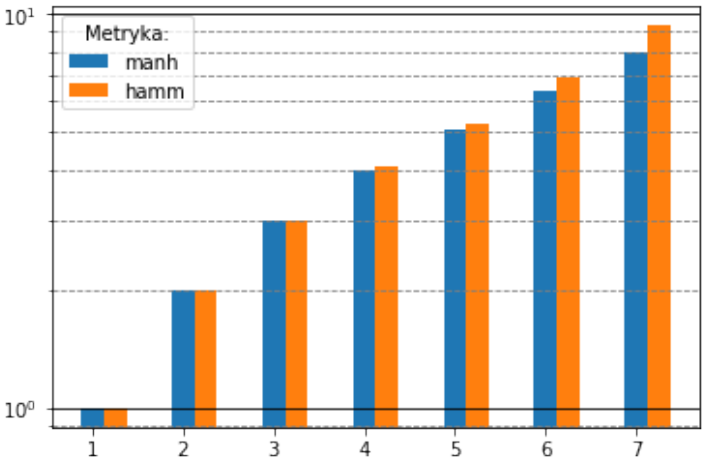
Uśredniona długość rozwiązania



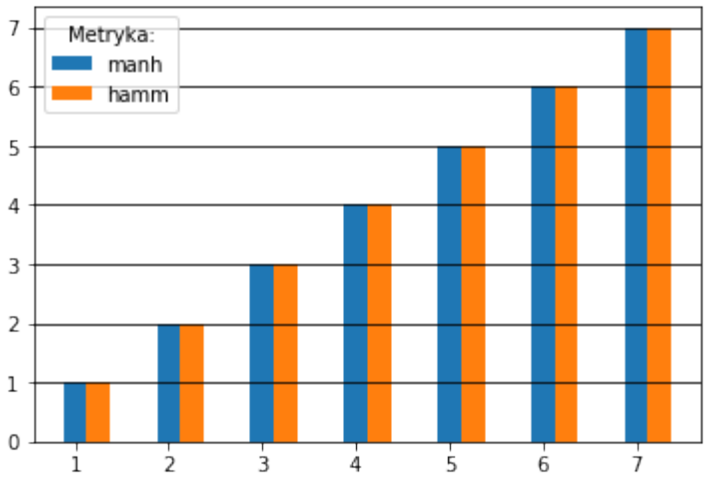
Uśredniona liczba odwiedzonych stanów



Uśredniona liczba przetworzonych stanów



Uśredniony maksymalny stopień rekursji



Uśredniony czas [ms]