Informatyka Stosowana, studia dzienne, inż I st. semestr IV

Sztuczna inteligencja i systemy ekspertowe 2022/2023

Prowadzący: dr inż. Krzysztof Lichy Poniedziałek, 15:45

Data oddania: _____ Ocena: _____

Grzegorz Bednarek 242359
Jędrzej Kostyk 242430

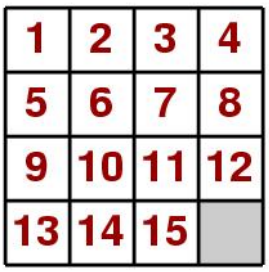
Zadanie 1: Piętnastka

1. Cel

Celem zadania było napisanie programu, który będzie rozwiązywał łamigłówkę "Piętnastka" przy użyciu różnych metod przeszukiwania stanu:
— Strategii "wszerz" - algorytm BFS
— Strategii "w głąb" - algorytm DFS
— Strategii "najpierw najlepszy" - algorytm A*
  — Metryką Hamminga
  — Metriką Manhattan

2. Wprowadzenie

Piętnastka to klasyczna łamigłówka, która składa się standardowo z 15 elementów na planszy o wymiarach 4x4.
Na każdym z tych elementów znajduje się liczba w przedziale od 1 do 15 włącznie. Jedno miejsce na planszy jest puste i umożliwia ono na zamianę miejsc z sąsiadującymi elementami na planszy.
Za stan rozwiązany przyjmuje się taki układ, gdzie numery są ustawione rosnąco od lewej do prawej oraz od góry do dołu:



Rysunek 1: *Ułożona układanka. która jest też poprawnym rozwiązaniem.*

Przesuwanie elementów na planszy umożliwia puste pole. Układanie tych elementów można potraktować jak przeszukiwanie drzewa składającego się z węzłów o wartościach "R", "D", "U" oraz "L", które odpowiadają za przesunięcia pustego pola w prawo, dół, górę i lewo.

3. Opis implementacji

Program, który rozwiązuję układankę "Piętnastkę" został przez nas napisany w języku Python.
Zawiera on następujące klasy:
— **Puzzle**
— **AStar**

3.1. Puzzle

Klasa ta zawiera większość implementacji programu. Przede wszystkim, zawiera ona konstruktor układanki, który bierze pod uwagę takie parametry jak wymiary i wartości układanki, strategię oraz jej parametry,
dodatkowe zmienne wykorzystywane w zapisie statystyk rozwiązania, oraz
Metody można podzielić na trzy zasadnicze funkcjonalności:
— Operacje na plikach
— Poruszanie się po planszy
— Algorytmy rozwiązania układanki
W tej klasie są cztery najważniejsze metody: `bfs`, `dfs`, `aStar` oraz `moveNode`. Metoda `bfs` implementuje algorytm BFS do rozwiązania układanki. Podobną implementację ma metoda `dfs`, która odpowiada za algorytm DFS.
`aStar` jest odpowiedzialna za utworzenie obiektu klasy AStar, który wywołuję metodę `solve` do rozwiązania układu. Metoda `moveNode` pozwala na poruszanie pustym polem po planszy, po czym zwraca plansze z wykonanym ruchem.

3.2. AStar

Klasa ta implementuje trzy najważniejsze metody: `manhattanDistance`, `hammingDistance` oraz `solve`. `manhattanDistance` oraz `hammingDistance` odpowiadają za implementację heurystyk Hamminga oraz Manhattan wymagane do algorytmu A*. Metoda `solve` implementuje algorytm A*, który korzysta właśnie z tych heurystyk.

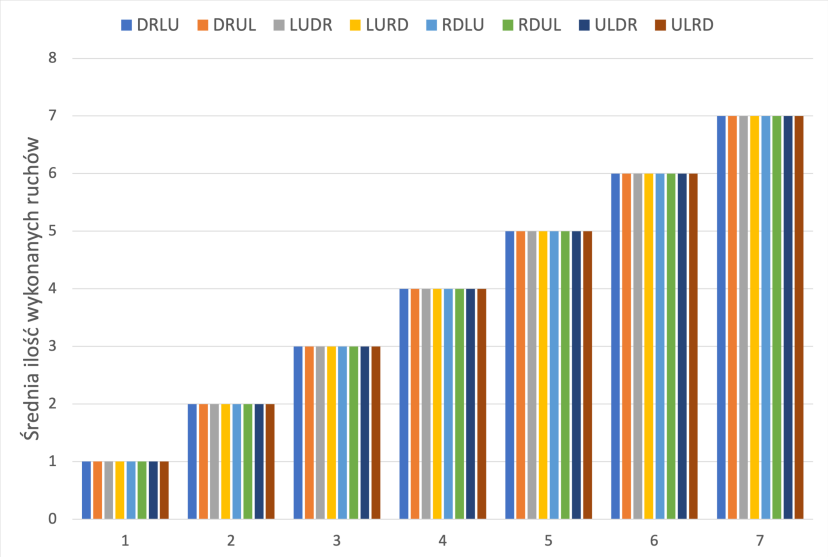
4. Materiały i metody

Badania zostały przeprowadzone przy użyciu załączonych skryptów `bash`, które były do pobrania na stronie przedmiotu. Wygenerowaliśmy wszystkie możliwe układanki dla ruchów od 1 do 7, czyli łącznie 413 układanek.
Następnie przy pomocny skryptu "Uruchamiacz przeszukiwań" uruchomiliśmy naszą aplikacji dla następujących parametrów:

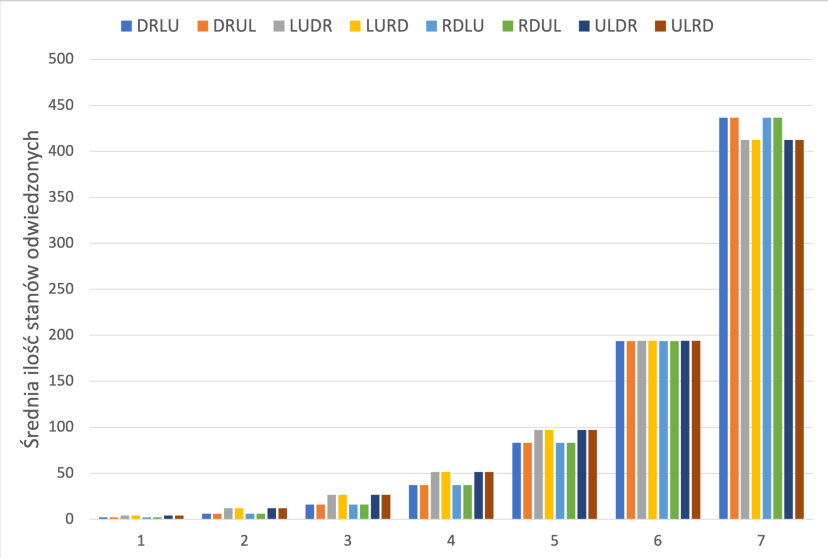
— **Algorytmy BFS i DFS**
  — *prawo-dół-góra-lewo;*
  — *prawo-dół-lewo-góra;*
  — *dół-prawo-góra-lewo;*
  — *dół-prawo-lewo-góra.*
  — *lewo-góra-dół-prawo.*
  — *lewo-góra-prawo-dół;*
  — *góra-lewo-dół-prawo;*
  — *góra-lewo-prawo-dół;*
— **Algorytm A***
  — Metryka Hamminga
  — Metryka Manhattan

5. Wyniki

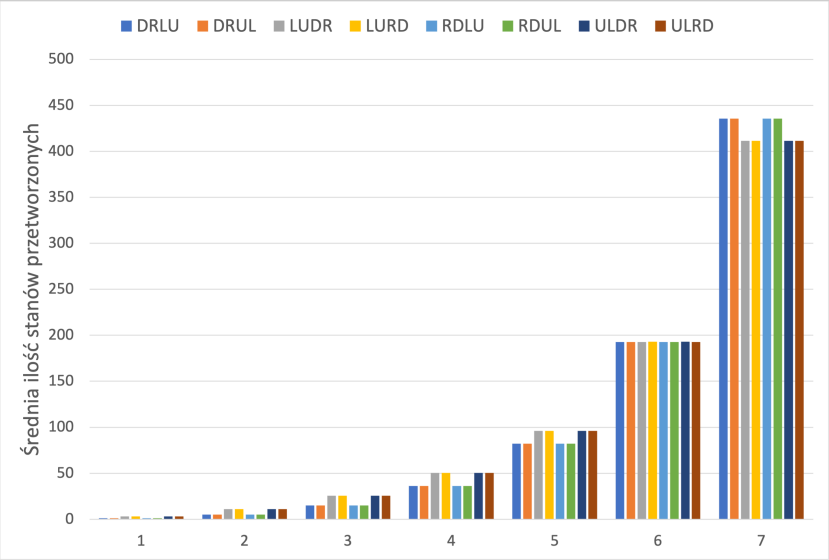
Wyniki naszych badań zostały przedstawione na wykresach poniżej. Podzieliliśmy je ze względu na wykorzystaną strategię przeszukiwania. Na każdym wykresie oś X przedstawia odległość początkowej układanki od rozwiązania.



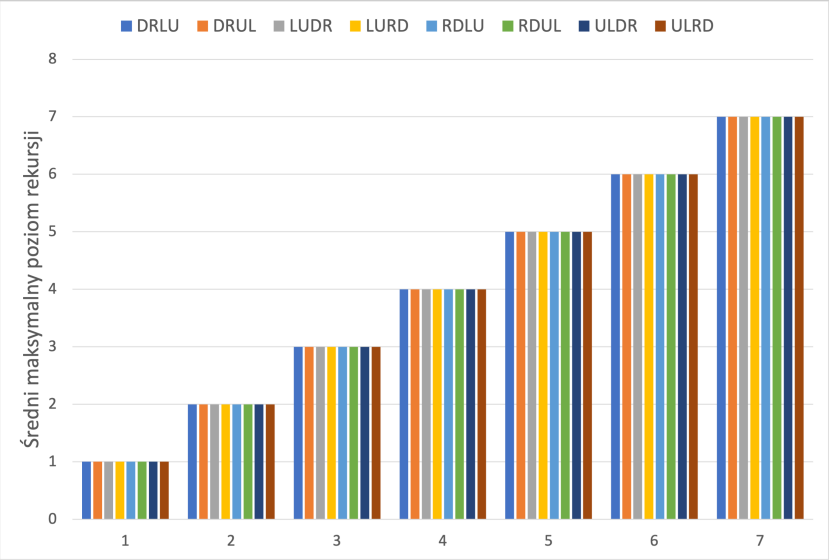
Wykres 1: *Średnia liczba wykonanych ruchów dla BFS.*



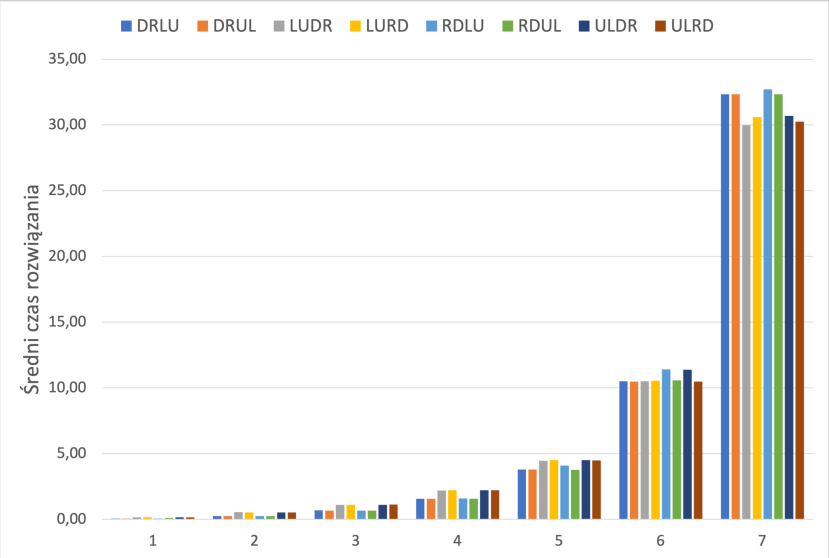
Wykres 2: *Średnia liczba odwiedzonych stanów dla BFS.*



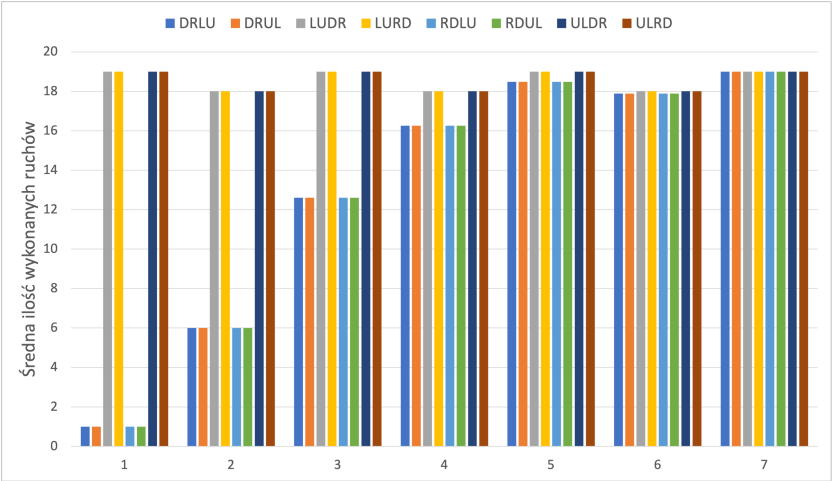
Wykres 3: *Średnia liczba przetworzonych stanów dla BFS.*



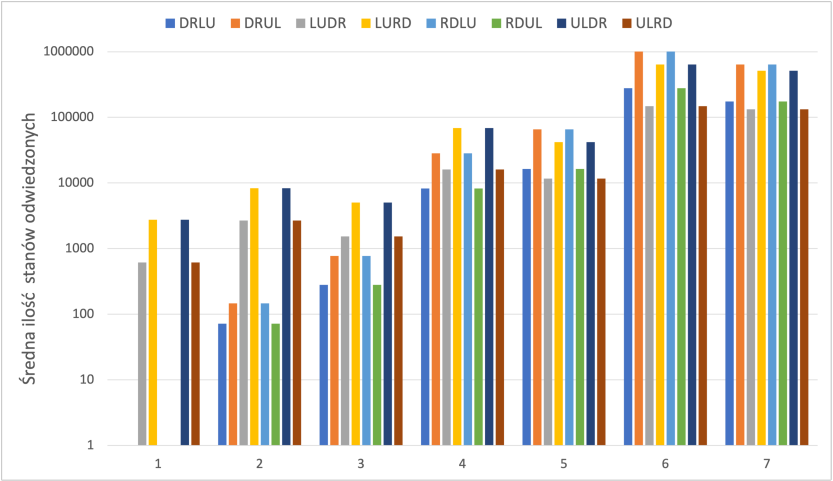
Wykres 4: *Średni maksymalny poziom rekursji dla BFS.*



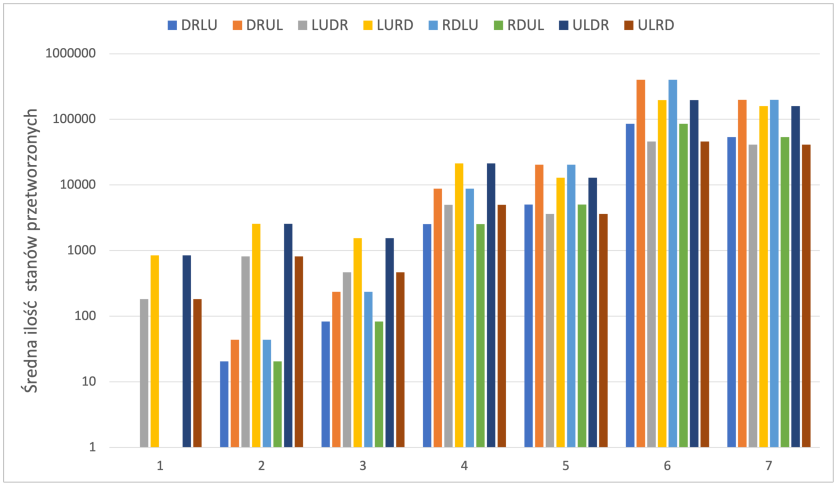
Wykres 5: *Średni czas rozwiązania dla BFS.*



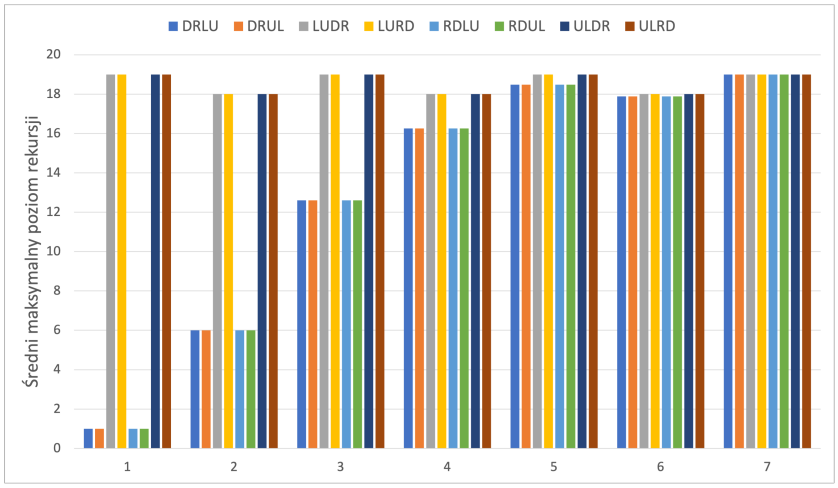
Wykres 6: *Średnia liczba wykonanych ruchów dla DFS.*



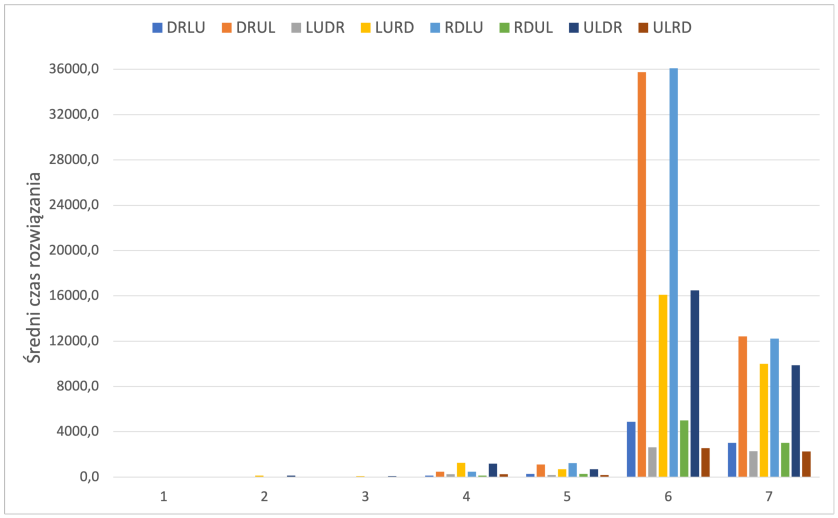
Wykres 7: *Średnia liczba odwiedzonych stanów dla DFS.*



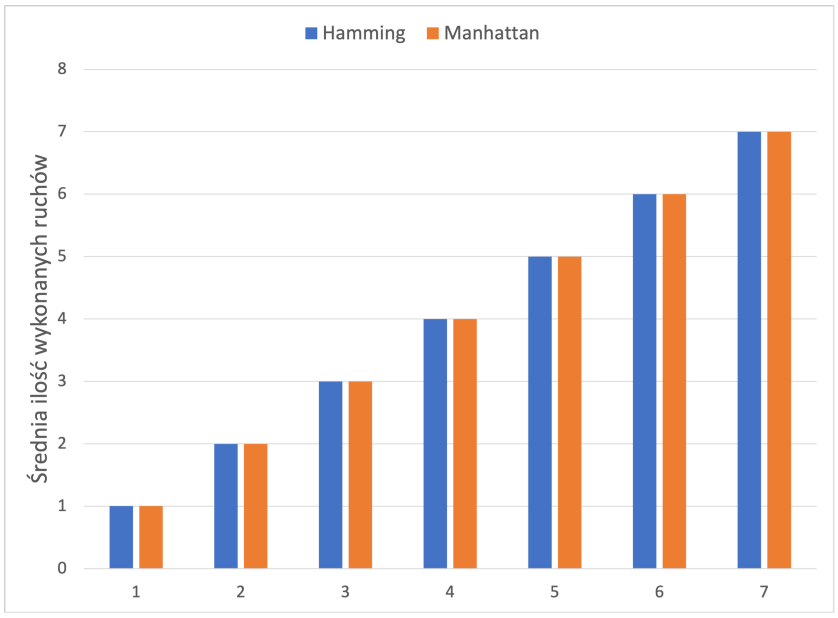
Wykres 8: *Średnia liczba przetworzonych stanów dla DFS.*



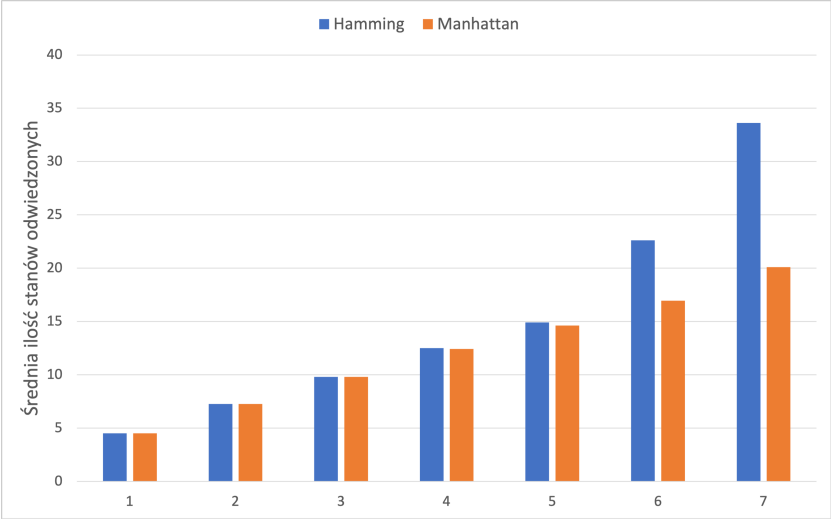
Wykres 9: *Średni maksymalny poziom rekursji dla DFS.*



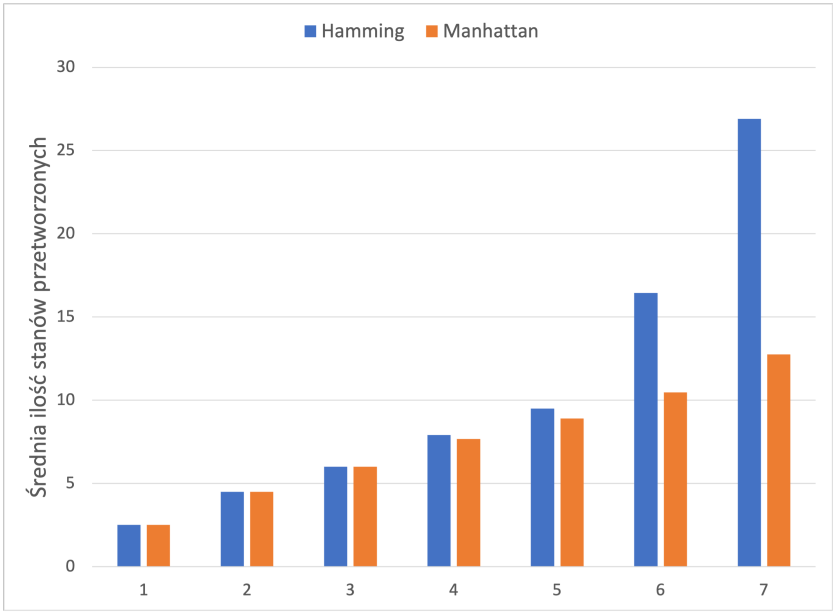
Wykres 10: *Średni czas rozwiązania dla DFS.*



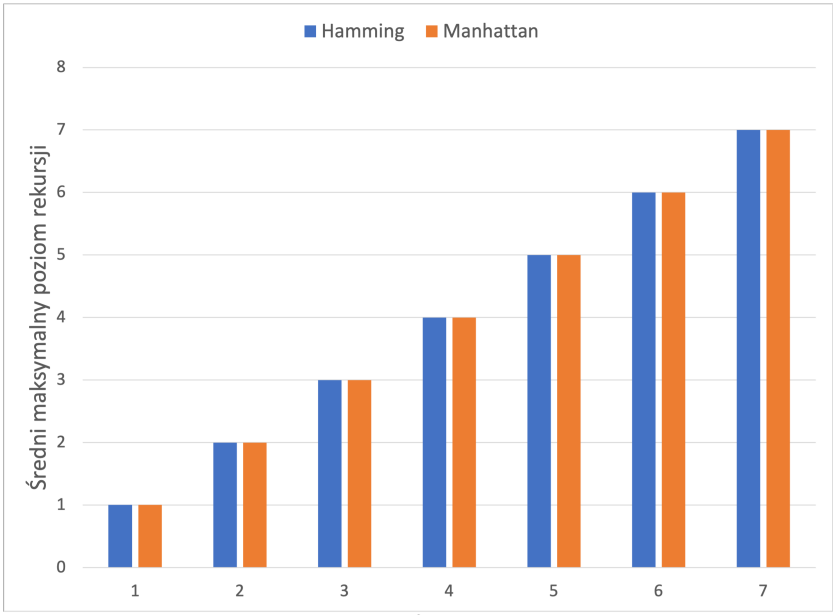
Wykres 11: *Średnia liczba wykonanych ruchów dla A*.*



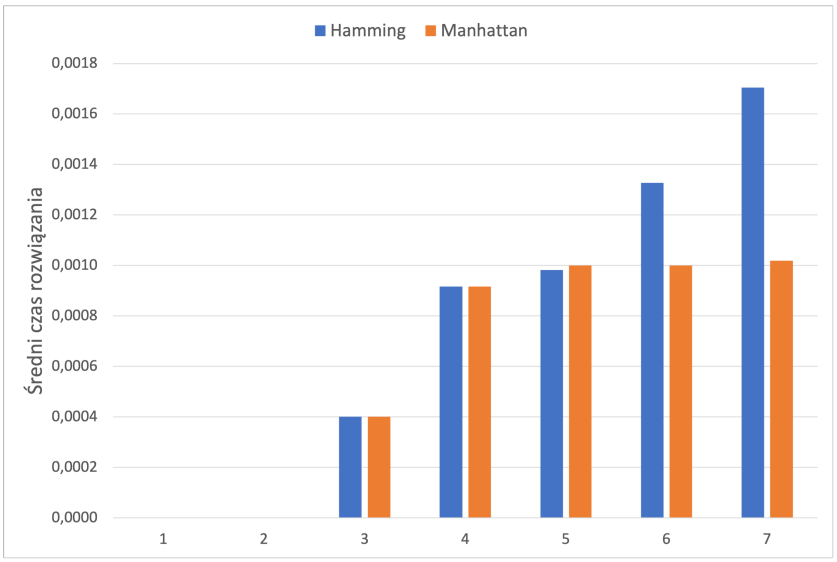
Wykres 12: *Średnia liczba odwiedzonych stanów dla A*.*



Wykres 13: *Średnia liczba przetworzonych stanów dla A*.*



Wykres 14: *Średni maksymalny poziom rekursji dla A*.*



Wykres 15: *Średni czas rozwiązania dla A*.*

6. Dyskusja

6.1. Strategia BFS

BFS szuka rozwiązania optymalnego, o minimalnym koszcie.
Wynika to z góry ustalonego porządku, który nie pozwoli przetwarzać stanów o koszcie wyższym przed przeanalizowaniem wszystkich stanów o koszcie niższym. W strategii BFS nigdy nie przekroczymy głębokości przy przeszukiwaniu rozwiązania. BFS stosunkowo szybko znajduje rozwiązanie, nie biorąc pod uwagi odległości równej 7, gdzie algorytm ten potrafi znaleźć rozwiązanie znacznie wolniej od algorytmu A*.

6.2. Strategia DFS

Porównując DFS do pozostałych algorytmów wypada on najgorzej.
Rozwiązania były optymalne dla odległości od 1 do 3, lecz dalsze trwały znacznie dłużej. Jeśli algorytm ten nie potrafi znaleźć rozwiązania przy pierwszych kilku iteracjach, to odbiega on znacząco od prawidłowego rozwiązania, praktycznie prawie dochodząc do maksymalnej dopuszczalnej głębokości rekursji, przetwarzając to kolejne stany w okolicach tej głębokości, aż do znalezienia rozwiązania.

6.3. Strategia A*

Algorytm A* jest najbardziej optymalny pod względem odwiedzonych i przetworzonych stanów. Wynika to z faktu, że w A* kolejność stanów do przetwarzania jest ustalana na podstawie heurystyki, a nie na z góry ustalonego porządku, tak jak ma to miejsce w strategiach BFS i DFS. Różnice w heurystykach nie różniły się od siebie znacząco w odległościach od 1 do 5, natomiast widać różnicę na odległościach 6 i 7, gdzie metoda Hamminga odbiega od wyników metody Manhattan przy odwiedzonych i przetworzonych stanach, jak i również w czasie rozwiązania.

7. Wnioski

— Wydajność strategii BFS znacząco spada wraz ze wzrostem odległości od układu wzorcowego.
— Algorytm BFS gwarantuje znalezienie najkrótszego rozwiązania.
— DFS jest strategią najmniej optymalną i już na początku potrafi odbiec bardzo daleko od rozwiązania, co powoduje przetwarzanie dużej ilości stanów.
— Algorytm DFS jest najwolniejszym ze wszystkich.
— W strategii A* różnice pomiędzy heurystykami zazwyczaj są nieznaczne, lecz mogą się od siebie różnić w zależności od odległości badanego układu.

Literatura

[1] https://www.overleaf.com/learn/latex/Tutorials
[2] https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/
[3] https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/
[4] https://www.geeksforgeeks.org/a-search-algorithm/
[5] https://en.wikipedia.org/wiki/Hamming_distance
[6] https://inventwithpython.com/recursion/chapter12.html
[7] https://ftims.edu.p.lodz.pl/mod/page/view.php?id=47860
[8] https://ftims.edu.p.lodz.pl/course/view.php?id=16#section-1