Informatyka, studia dzienne, inż I st. semestr V

Sztuczna inteligencja i systemy ekspertowe 2018/2019

Prowadzący: Dr inż. Krzystof Lichy Piątek, 9:30

Data oddania: _____ Ocena: _____

Krzysztof Barden 210139

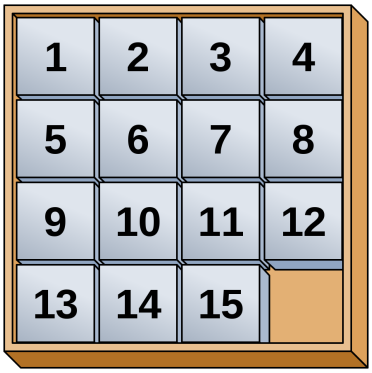
Zadanie 1: Piętnastka

1. Cel

Celem zadania było napisanie programu, który będzie rozwiązywał logiczną układankę zwaną piętnastką oraz zbadanie różnych metod przeszukiwania stanu.

2. Wprowadzenie

Piętnastka (ang. Fifteen Puzzle) to logiczna układanka składająca się z piętnastu kwadratowych klocków o jednakowych rozmiarach, numerowanych przeważnie od 1 do 15, ułożonych na planszy w kształcie kwadratu 4 na 4. Jedno miejsce na planszy pozostaje puste i umożliwia przesuwanie sąsiednich klocków względem siebie. Celem gry jest uporządkowanie klocków w kolejności rosnącej, odzwierciedlających poprawnie rozwiązany układ (rysunek 1). Istnieją inne warianty tej układanki, w której klocków i pól może być mniej lub więcej w innych kształtach planszy (warunkiem jest możliwoć rozwiązania układanki).



Rysunek 1. Poprawny układ piętnastki

Poszukiwanie rozwiązania łamigłówki można porównać do znajdowania ścieżki w grafie, gdzie stan układanki jest węzłem grafu, a pojedynczy ruch krawędzią. W celu odnalezienia ścieżki można zastosować różne strategie przeszukiwania przestrzeni stanów, między innymi:
— BFS (breadth-first search)
— DFS (depth-first search)
— A* (A-star) z heurystykąi Manhattan, Hamminga lub innych

2.1. BFS

Algorytm przeszukiwania "wszerz" rozpoczyna przechodzenie grafu od zadanego wierzchołka (w tym przypadku układu startowego układanki), odwiedzając wszystkie osiągalne z niego wierzchołki na tym samym poziomie rekursji. Następnie odwiedza wszystkie osiągalne wierzchołki z wierzchołków pochodnych od poprzedniego (już na kolejnym poziomie rekursji). Z samych założeń BFS zawsze znajduje najkrótszą ścieżkę.

2.2. DFS

Algorytm przeszukiwania "w głąb" rozpoczyna przechodzenie grafu od zadanego wierzchołka (w tym przypadku układu startowego układanki), odwiedzając pierwszy z pochodnych wierzchołków, powtarzając to dla każdego pochodnego wierzchołka. Jeżeli algorytm nie będzie mógł wchodzić dalej, cofa się o jeden poziom rekursji i bada krawędź kolejnego nieodwiedzonego jeszcze wierzchołka.

2.3. A*

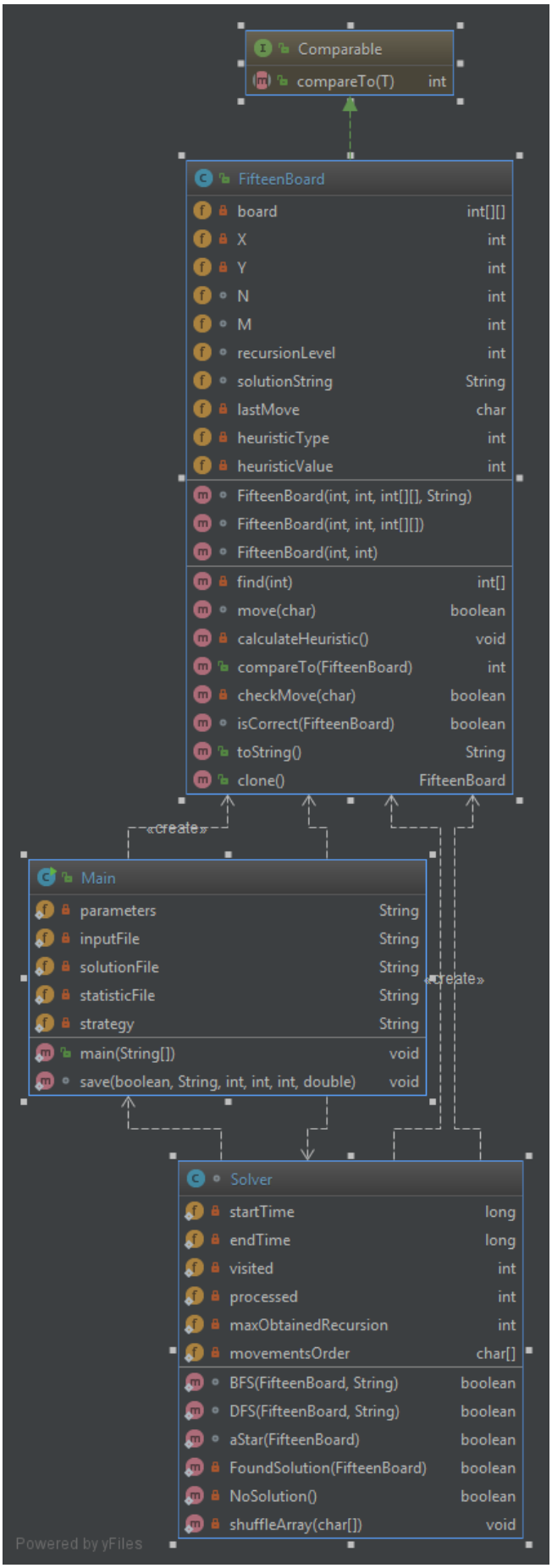
Algorytm heurystyczny jest to algorytmem zupełnym i optymalnym - znajduje najkrótsza ścieżkę, jeśli tylko taka istnieje. W przypadku piętnastki algorytm A* tworzy ścieżkę wybierając wierzchołek tak, aby minimalizować wartość heurestyki. Metody obliczania tej wartości to: metoda Hammminga, gdzie obliczamy ile klocków znajduje się na niewłaściwych pozycjach, metoda Manhattan, gdzie liczymy jakie odległości dzielą klocki od ich docelowych miejsc (kolumny + wiersze) i hiper-heurestyka łącząca powyższe heurestyki (suma wartosci wyliczonej z heurestyki Hamminga i heurestyki Manhattana).

3. Opis implementacji

Program został zaimplementowany w języku Java oraz zostały użyte dostępne na platformie WIKAMP programy i skrypty pomagające w generowaniu i przetwarzaniu danych. Na wejściu progarm odczytuje następujące dane:

1. Strategia:
   - BFS (w implementacji została użyta kolejka typu FIFO (first in, first out))
   - DFS (w implementacji została użyta kolejka typu LIFO (last in, first out), maksymalny poziom rekursji przy którym następuje nawrot został ustawiony na 20 poziom aby uniknąć zbyt długiego przeszukiwania jednej ścieżki, która i tak miałaby małą szansę na znalezienie rozwiązania)
   - ASTR (w implementacji tego algorytmu została użyta kolejka priorytetów ustawiana rosnąco zgodnie z wartością heurestyki)
2. Parametry wybranej strategii:
   - DFS lub BFS: permutacja liter: L, R, U, D wyznaczająca ciąg ruchów poszczególnych przesunięć
   - ASTR: heurestyka Hamminga (hamm) lub Manhattan (manh) lub własna/hiper-heurestyka (own); w celu optymalizacji w przypadku "remisów" kolejnosć ruchu jest ustalana losowo (Las Vegas Algorithm)
3. Nazwa pliku ze stanem początkowym układanki:
   - Pierwsza linia: wymiary łamigłówki
   - Druga linia: stan łamigłówki
4. Nazwa pliku wyjściowego z rozwiązaniem układanki:
   - Pierwsza linia: długość rozwiązania
   - Druga linia: sekwencja ruchów

5. Nazwa pliku wyjściowego ze statystykami:
   — Długość znalezionego rozwiązania
   — Liczbę stanów odwiedzonych
   — Liczbę stanów przetworzonych
   — Maksymalną osiągniętą głębokość rekursji
   — Czas trwania procesu obliczeniowego w milisekundach

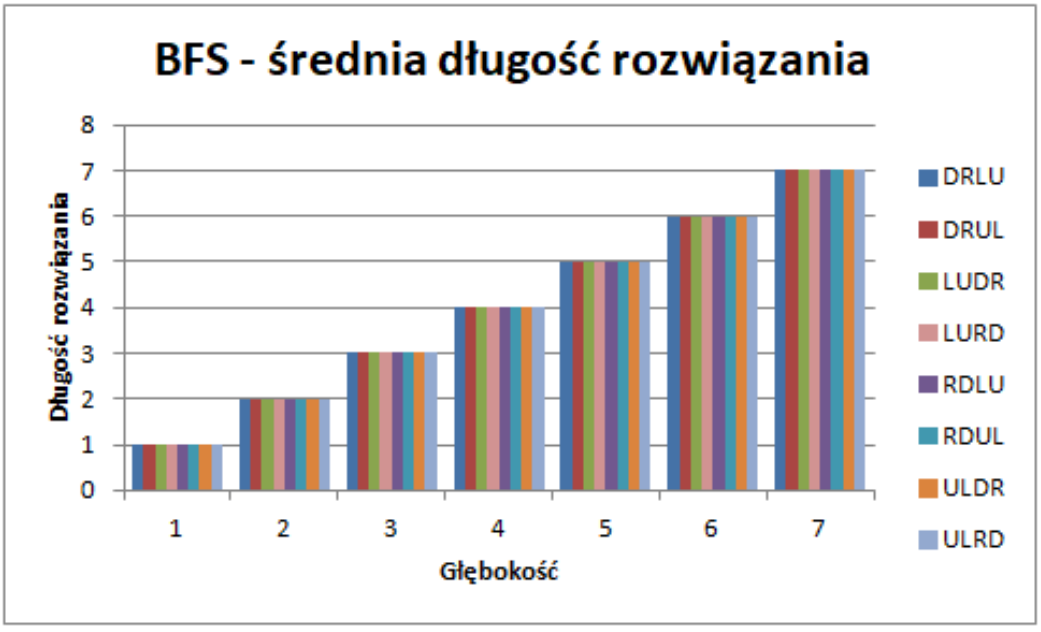


Rysunek 2. Diagram klas.

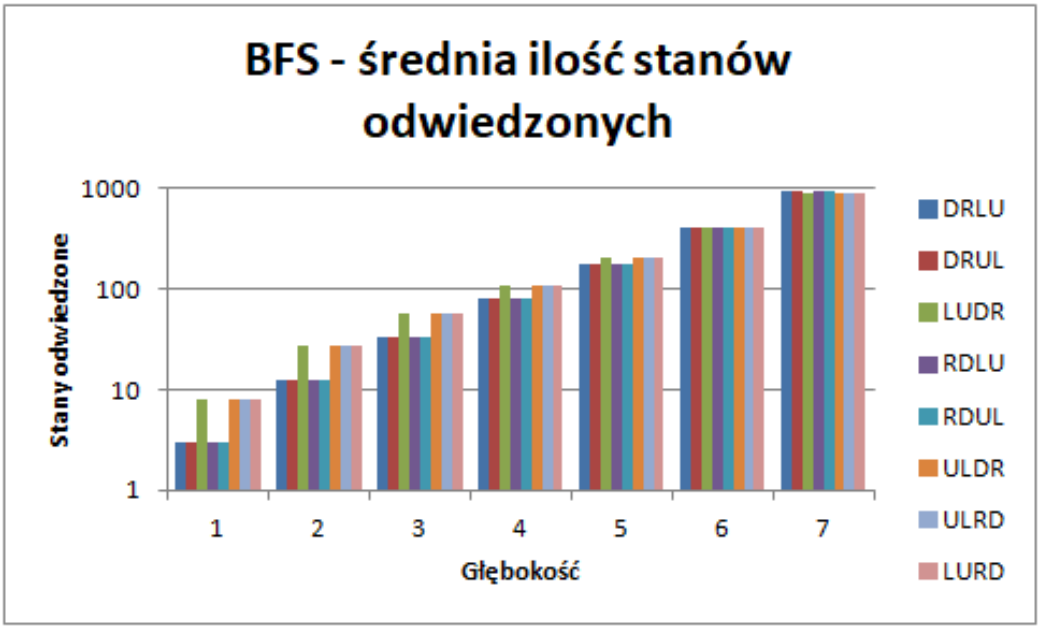
4. Materiały i metody

Za pomocą narzędzi dostępnych na platformie WIKAMP wygenerowaliśmy początkowe układy łamigłówki w odległościach 1-7. Przetestowaliśmy wyżej wymienione algorytmy w wymaganych wariantach opisanych w treści zadania korzystając z programów dostępnych na platformie WIKAMP (sprawdzanie poprawności rozwiązania). Dla przeszukiwania BFS oraz DFS skorzystaliśmy z następujących kombinacji dodatkowego parametru porządku przeszukiwania: RDLU, RDUL, DRUL, DRLU, LUDR, LURD, ULRD, ULDR. Dla przeszukiwania A*, każdy przypadek został przeprowadzony na obu heurystykach (Hamminga oraz Manhattan). Otrzymane wyniki przenieśliśmy do arkusza kalkulacyjnego, gdzie sporządziliśmy wymagane wykresy.

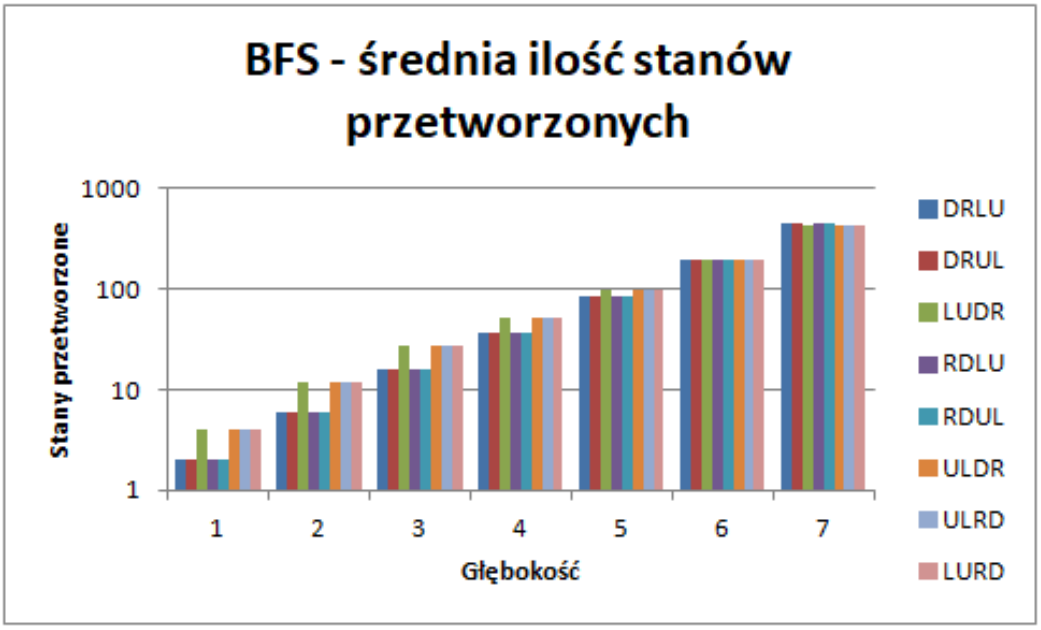
5. Wyniki



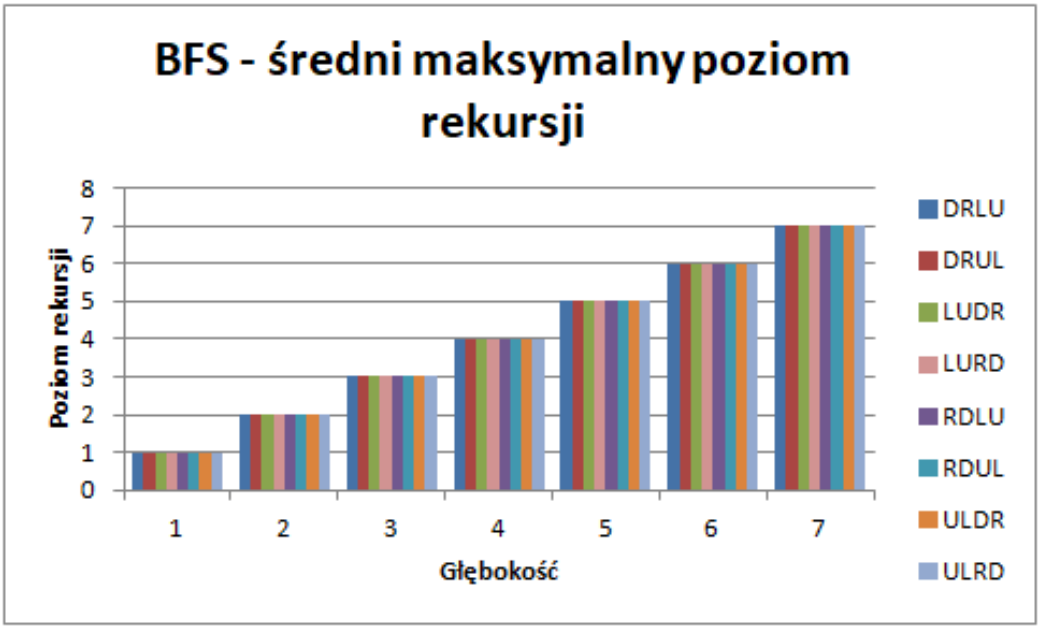
Rysunek 3. Średnia długość rozwiązania algorytmu BFS.



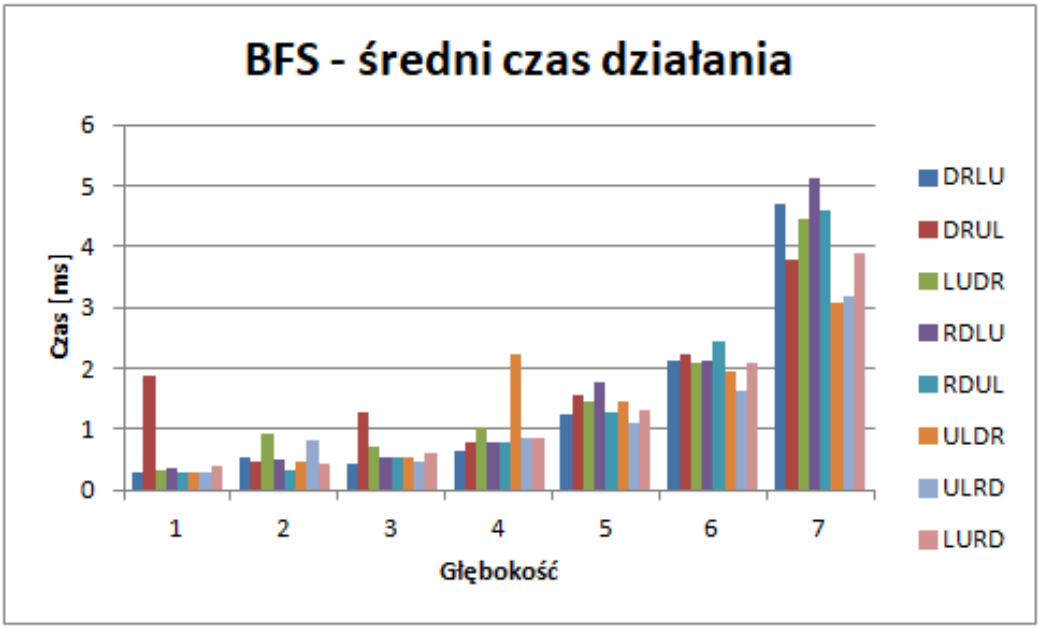
Rysunek 4. Średnia ilość stanów odwiedzonych algorytmu BFS.



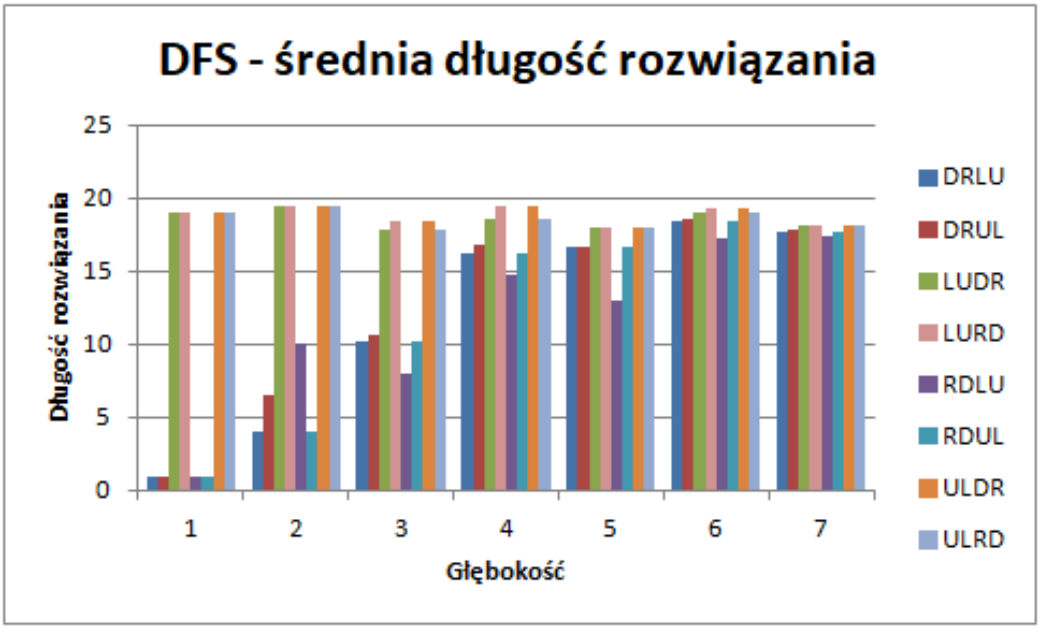
Rysunek 5. Średnia ilość stanów przetworzonych algorytmu BFS.



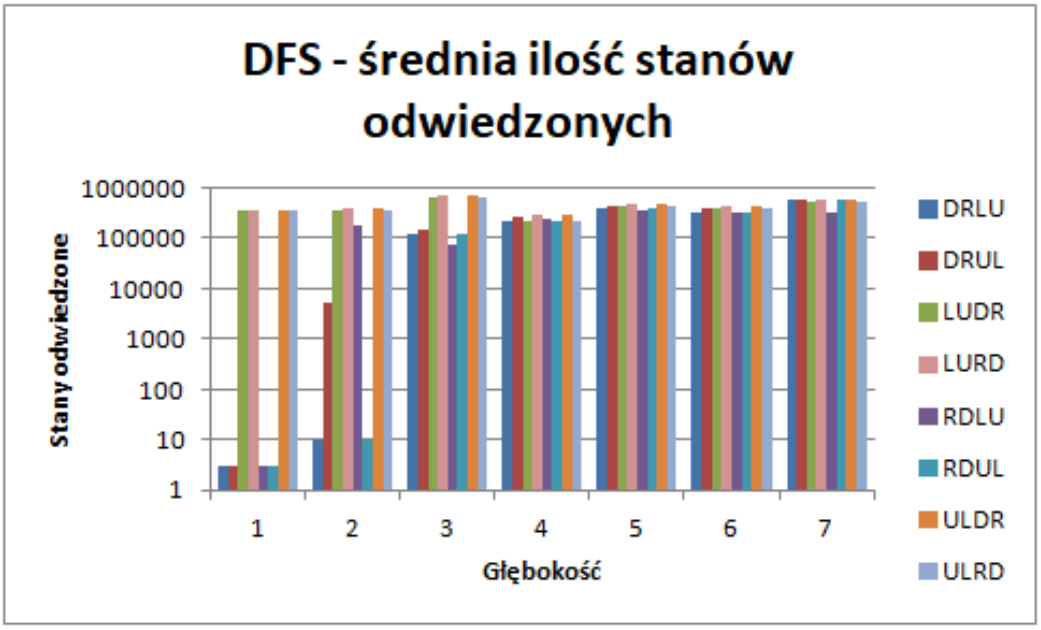
Rysunek 6. Średnia maksymalna rekursja algorytmu BFS.



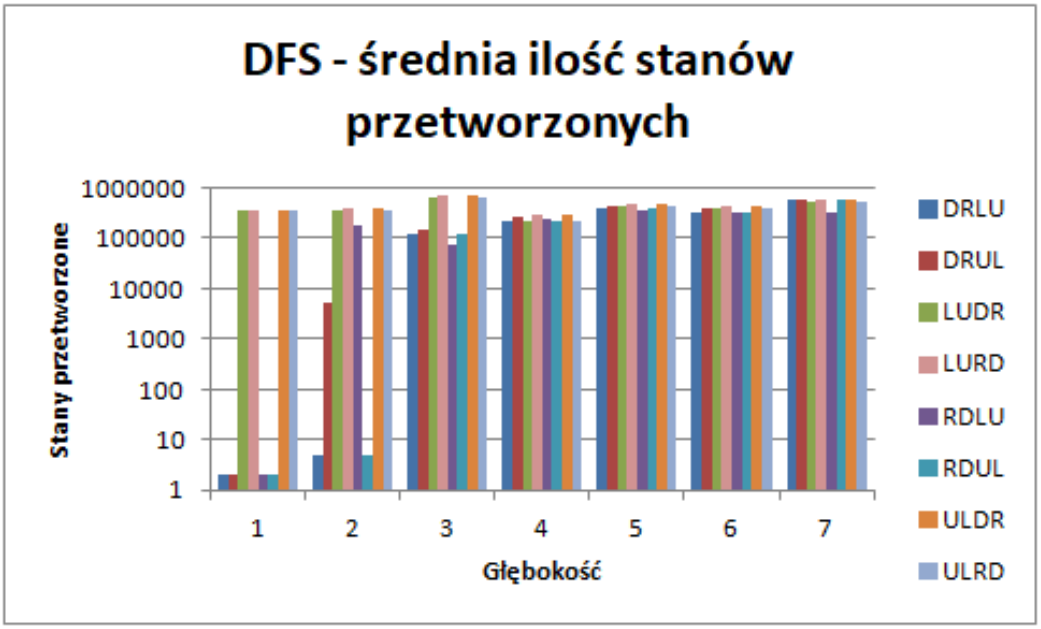
Rysunek 7. Średni czas działania algorytmu BFS.



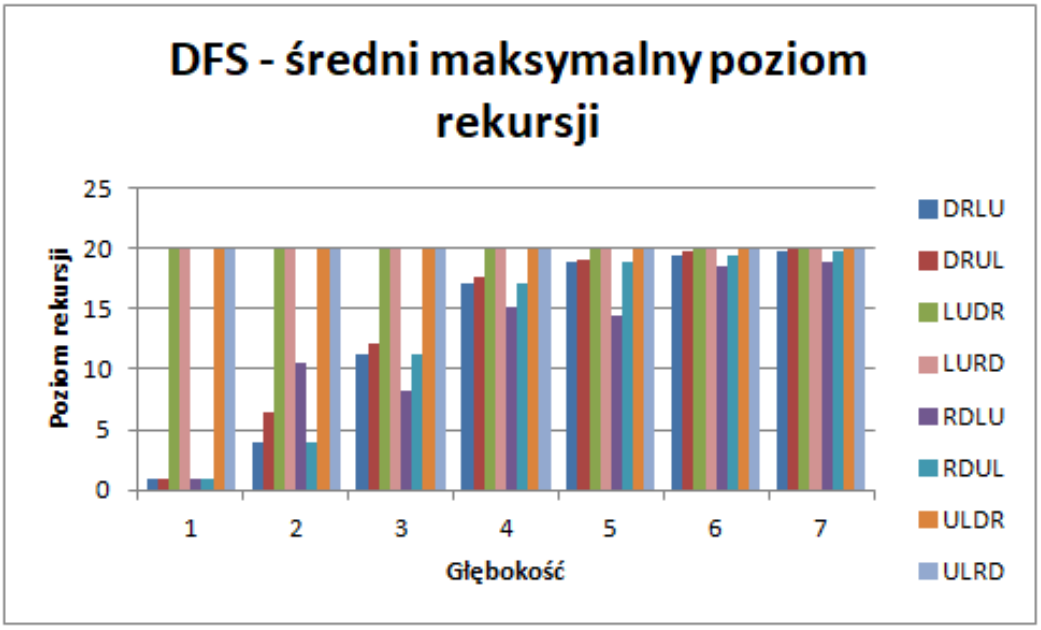
Rysunek 8. Średnia długość rozwiązania algorytmu DFS.



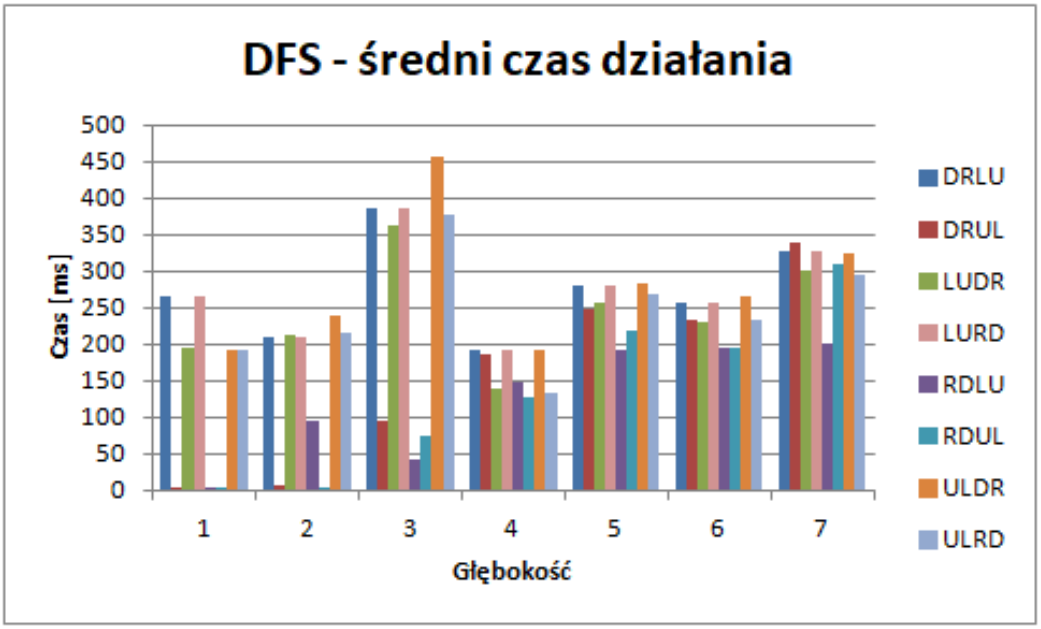
Rysunek 9. Średnia ilość stanów odwiedzonych algorytmu DFS.



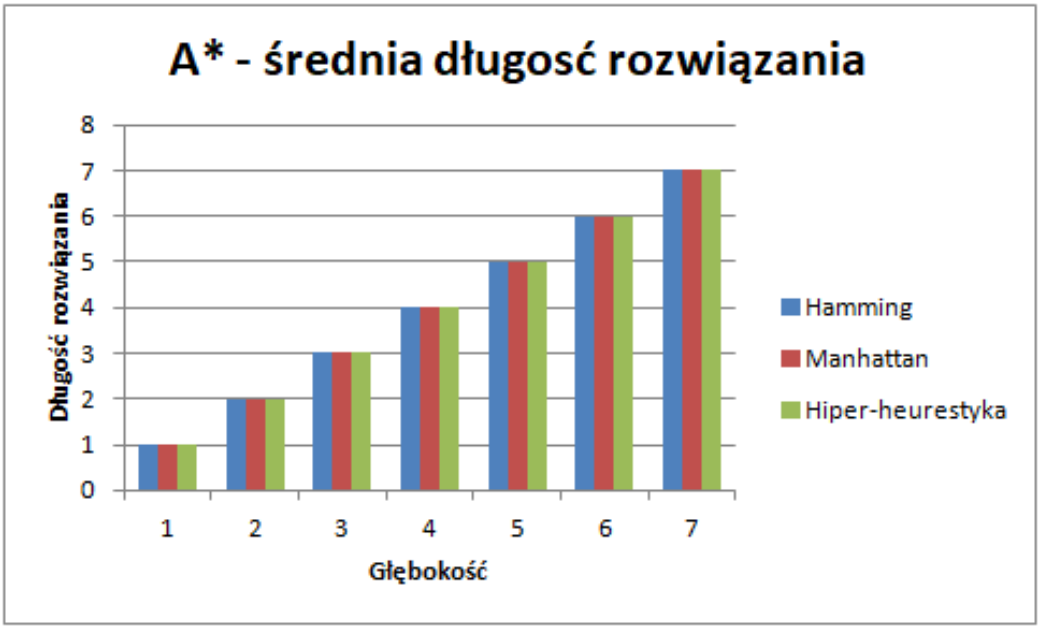
Rysunek 10. Średnia ilość stanów przetworzonych algorytmu DFS.



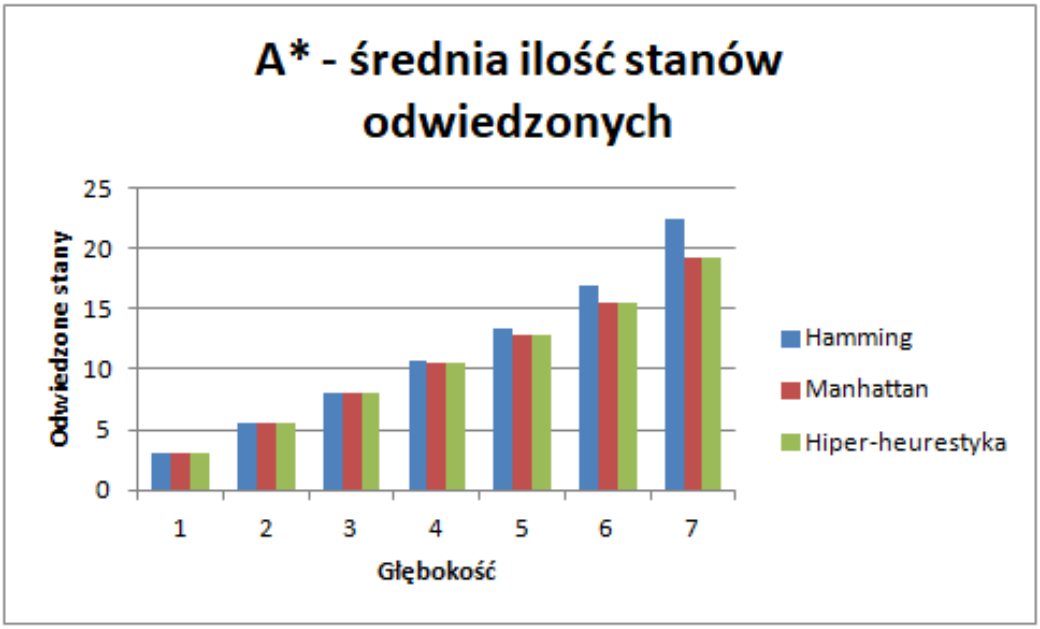
Rysunek 11. Średnia maksymalna rekursja algorytmu DFS.



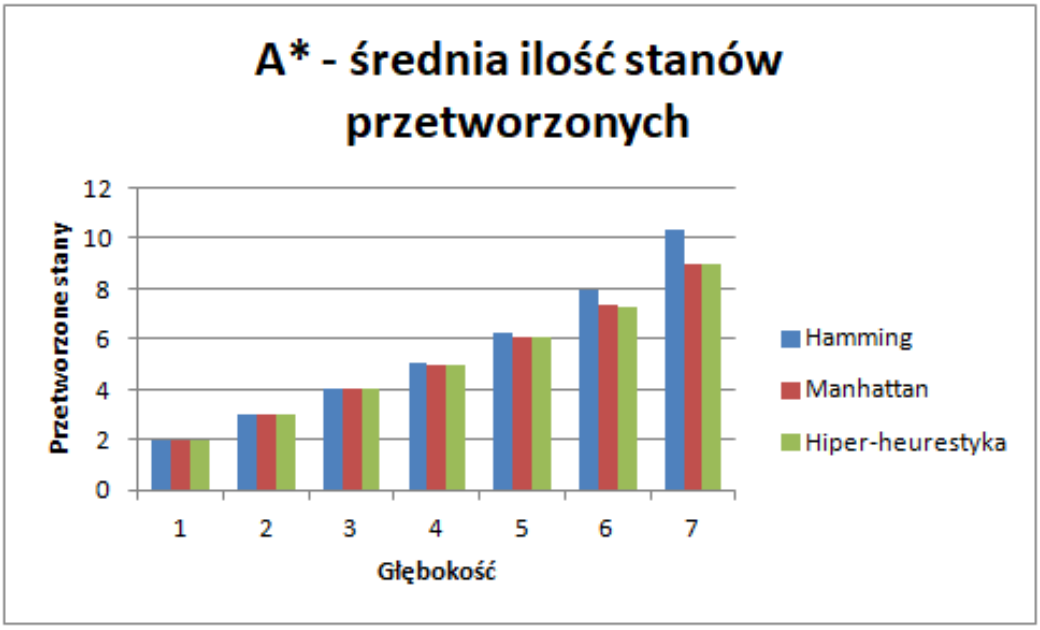
Rysunek 12. Średni czas działania algorytmu DFS.



Rysunek 13. Średnia długość rozwiązania algorytmu A*.

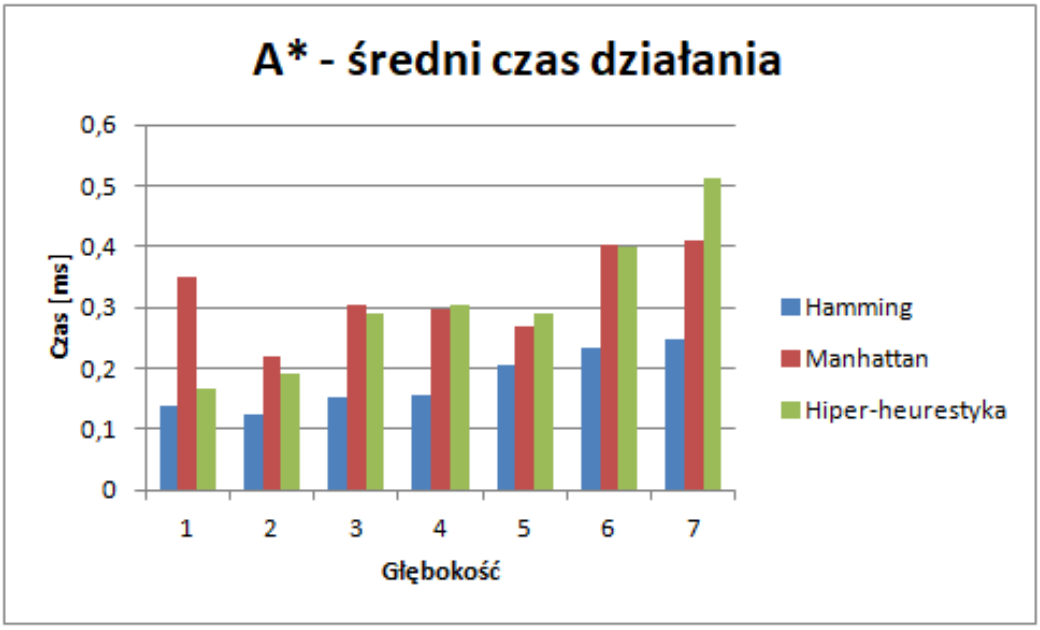


Rysunek 14. Średnia ilość odwiedzonych stanów algorytmu A*.



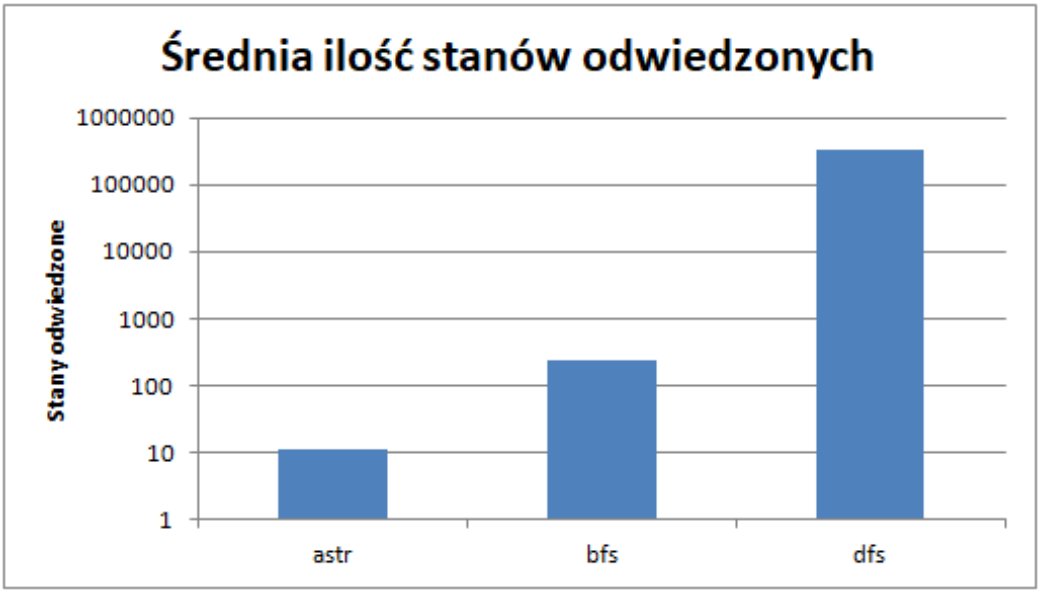
Rysunek 15. Średnia ilość przetworzonych stanów algorytmu A*.

Rysunek 16. Średnia maksymalna rekursja algorytmu A*.

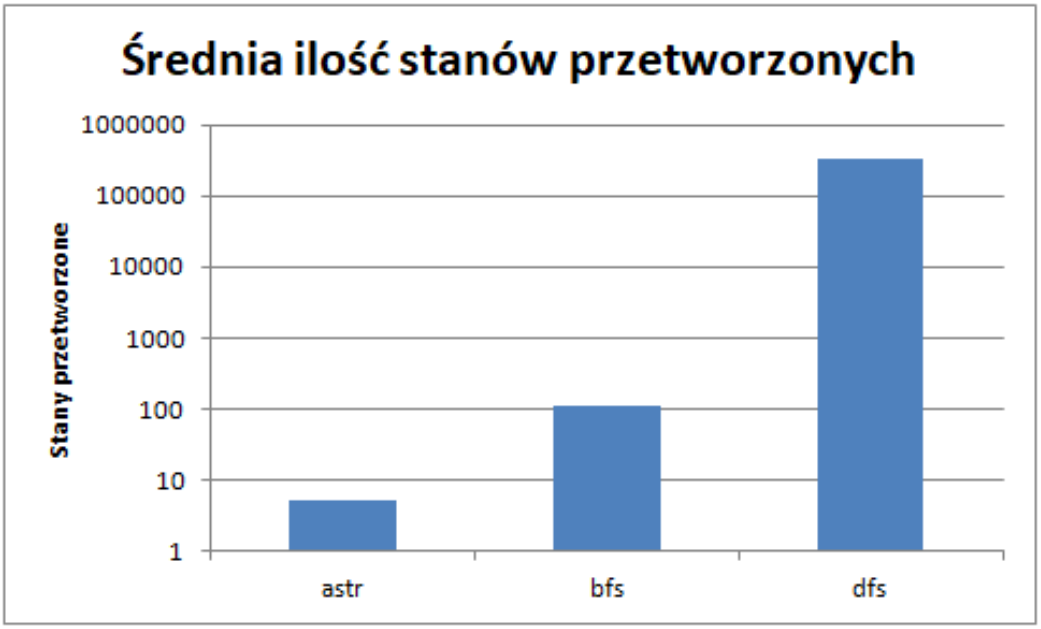


Rysunek 17. Średni czas działania A*.

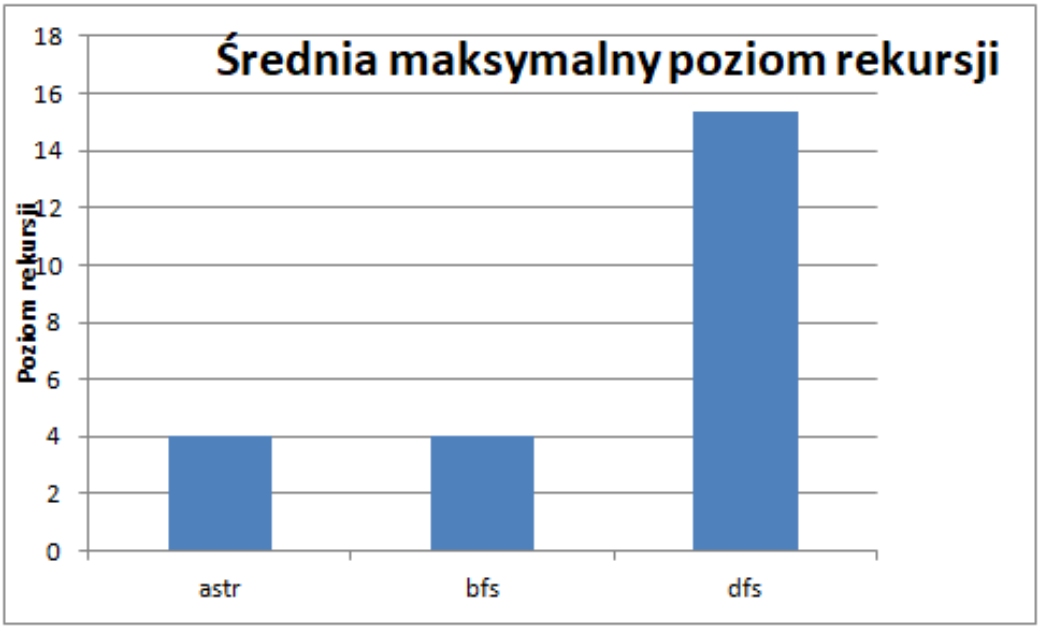
Rysunek 18. Średnia długość rozwiązania wszystkich algorytmów.



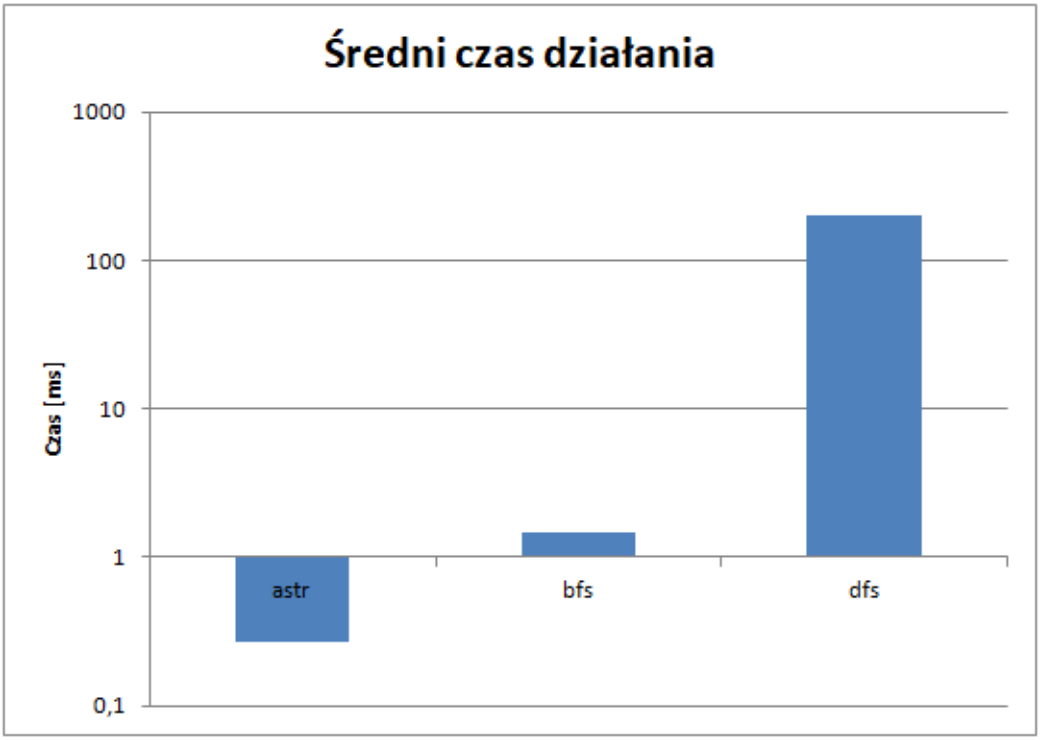
Rysunek 19. Średnia ilość odwiedzonych stanów wszystkich algorytmów.



Rysunek 20. Średnia ilość przetworzonych stanów wszystkich algorytmów.



Rysunek 21. Średnia maksymalna rekursja wszystkich algorytmów.



Rysunek 22. Średniczas działania wszystkich algorytmów.

6. Dyskusja

6.1. Średnia długość rozwiązania

Algorytmy BFS oraz A* zwracają rozwiązania o takiej samej długości co głębokość układanki - z samych założeń znajdują one najkrótsze rozwiązania.

Algorytm DFS rzadko znajduje najkrótsze rozwiązania. Jest to uzależnione od kolejności rozpatrywania ruchów. Niekorzystny układ ruchów może spowodować długie rozwiązania nawet przy układach o małej głębokosci (aż do 20 ruchów). Wraz ze wzrostem głębokości (a więc też ilości istniejących krawędzi w grafie) prawodobieństwo na znalezienie najkrótszego rozwiązania drastycznie maleją.

6.2. Średnia ilość odwiedzonych stanów

Ilość odwiedzonych stanów dla algorytmu A* jest najmniejsza ze wszystkich użytych algorytmów. Heurestyka Mahattana i hiper-heurestyka mają podobne wyniki i są lepsze niż heurestyka Hamminga. Wartości nie przekraczają 46.

Dla algorytmu BFS, wraz ze wzrostem głębokości wartości szybko rosną osiągając prawie 1000 odwiedzonych stanów.

Dla algorytmu DFS głębokość nie ma największego znaczenia. Najbardziej wpływa kolejność wykonywanych ruchów. Dla małych głebokości jest to najlepiej widoczne.

6.3. Średnia ilość przetworzonych stanów

Algorytm A* średnio odwiedza mniej niż 10 stanów w celu znalezienia rozwiązania (dla głębokości 7 herurestyka Hamminga potrzebuje średnio ok 10 przetworzonych stanów, zaś Manhattan i hiper-heurestyka niecałe 9).

Ilość przetworzonych stanów dla algorytmu BFS wynosi ok. połowe stanów odwiedzonych jednak nadal jest to więcej niż w przypadku A*. DFS przetwarza on ponad 99% odwiedzonych przez siebie rozwiązań, pokazując nieprzydatność do rozwiązywania tej układanki.

6.4. Średni maksymalny poziom rekursji

Zgodnie z definicją BFS i A* nie maksymalny poziom rekursji nie przekracza głębokości rozwiązania.

W przypadku algorytmu DFS zadany maksymalny poziom rekursji jest prawie zawsze osiągany co pokazuje słuszność użycia tego limitu.

6.5. Średni czas działania

Na czas działania algorytmu wpływają różne czynniki takie jak programy pracujące w tle lub sposób przydzielania przez system zasobów dla programu co wpływa na zakłócenia w pomiarach. Ilość odwiedzonych i przetworzonych stanów odzwierciedla się w czasie działania co powoduje że algorytm A* jest najszybszy. Jednak większy wpływ w przypadku tego algorytmu mają sposoby obliczania heurestyki. Sprawdzenie czy dany klocek w układance jest na swoim miejscu (Hamming) jest o wiele mniej kosztowne od liczenia odległości dla każdego klocka (Manhattan). Hiper-heurestyka jest zestawieniem obu tych heurestyk więc w teorii powinna zajmować najwięcej czasu na obliczenie.

Czas rozwiązywania za pomocą algorytmu BFS potrafi być zróżnicowany ze względu na możliwość przyjęcia różnych strategi przeszukiwania stanów. Korzystne kombinacje kroków dla większych głębokości są widoczne w lepszych czasach.

Średni czas działania algorytmu DFS jest ok 200 razy większy od czasu BFS, co pokazuje dużą słabość tego algorytmu przy rozwiązywaniu tej układanki.

6.6. Heurestyki

W wynikach widoczne jest podobieństwo hiper-heurestyki do heurestyki Manhattana. Spowodowane jest to tym, że sumy odległości są o wiele większe od ilości klocków nie na swoim miejscu. Widać też małe różnice w skuteczności heurestyk pomiędzy małymi a większymi głębokościami układanki.

7. Wnioski

— Algorytm A* najlepiej nadaje się do rozwiązywania "piętnastki" bez względu na wybraną heurestykę.

— Heurystyka Manhattana nadaję się do bardziej skomplikowanych ułożeń, a Hamminga do prostszych.
— Odpowiednie proporcje przy sumowaniu heurestyk w hiper-heurestyce mogłoby dać większe efekty.
— Algorytm BFS nie jest najbardziej optymalnym algorytmem do rozwiązywania tej układanki ale jego wyniki są zadowalające.
— Algorytm DFS nie jest odpowiedni do rozwiązywania "piętnastki" ze względu na zbyt długi czas działania, przypadkowość w znajdywaniu krótszych rozwiązań i zbyt duże pochłanianie zasobów programu.

Literatura

[1] http://www.math.edu.pl/pietnastka
[2] https://pl.wikipedia.org/wiki/Piętnastka_(układanka)
[3] http://jamie-wong.com/2011/10/16/fifteen-puzzle-algorithm/
[4] https://en.wikipedia.org/wiki/Hyper-heuristic
[5] https://pl.wikipedia.org/wiki/Algorytm_A*
[6] https://pl.wikipedia.org/wiki/Odległość_Hamminga
[7] https://en.wikipedia.org/wiki/Las_Vegas_algorithm