Informatyka, studia dzienne, inż. I st. semestr VI

**Sztuczna inteligencja i systemy ekspertowe 2016/2017**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prowadzący: dr inż. Krzysztof Lichy | | | piątek, 10:15 | |
|  |  |  |  |  |
| Data oddania: |  |  | Ocena: |  |

|  |  |
| --- | --- |
| Alicja Gałkiewicz | 195589 |
| Maciej Ślusarz | 189789 |

Zadanie 2: Sieć neuronowa

**1. Cel**

Celem zadania było napisanie programu implementującego sieć neuronową typu wielowarstwowy perceptron, nauczaną metodą wstecznej propagacji błędu. Sieć neuronowa miała nauczyć się obliczania pierwiastka drugiego stopnia liczby – czyli sieć ma za zadanie przeprowadzać aproksymację funkcji. Następnie, zbadano, jak wpływają parametry sieci i parametry uczenia sieci na wynik jej działania.

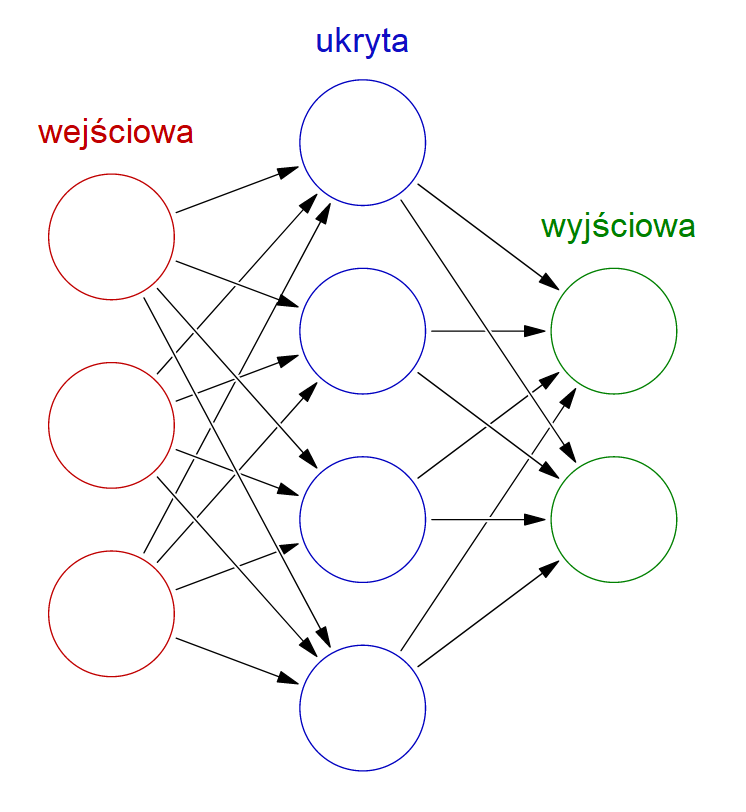
**2. Wprowadzenie**

Sieci neuronowe są cyfrową symulacją sposobu działania realnych biologicznych układów nerwowych poprzez symulację komórek nerwowych oraz ich połączeń. Stosując propagację wsteczną błędu przez sieć, możliwe jest dostosowanie współczynników (wag - inicjowanych jako losowe) połączeń między neuronami. W konsekwencji zwiększa się lub zmniejsza emfaza pewnych cech analizowanego sygnału wejściowego. Efektem tego procesu na danych których oczekiwaną wartość znamy, jest nauczenie sieci rozpoznawania cech i wzorców w danych wejściowych, co umożliwia późniejsze stosowanie jej do przetwarzania nieznanych wcześniej danych.

  
*rys. 1. Model pojedynczego neuronu.*

Zachowanie, elastyczność i czas uczenia się sieci jest w znacznym stopniu zależna od liczby neuronów wchodzących w jej skład. Neurony są pogrupowane w warstwy i podczas propagacji sygnału przez sieć, kluczowym jest zachowanie narzuconej przez warstwy kolejności rozpatrywania ich. Warstwy sieci neuronowej mogą być podzielone na trzy kategorie:

* wejściowa - składa się z określonej, stałej dla danej sieci liczby neuronów, każdy z nich przyjmujący cały wektor danych wejściowych
* ukryte - dowolna określona dla danej sieci liczba warstw z których pierwsza pobiera dane z warstwy wejściowej a pozostałe z poprzedniej warstwy ukrytej. Z założenia użytkownik nie ma bezpośredniego dostępu ani do sygnału wejściowego ani wyjściowego żadnego neuronu należącego do dowolnej warstwy ukrytej.
* wyjściowa - neurony wyjściowe pobierają dane z ostatniej warstwy ukrytej i są ostatnim czynnikiem przetwarzającym w sieci neuronowej. Dane wyjściowe neuronów tej warstwy są bezpośrednio dostępne dla użytkownika.

  
*rys. 2. Model sieci neuronowej typu wielowarstwowy perceptron.*

Zaprogramowana przez nas sieć jest przykładem tzw. sieci gęstej. Każdy neuron danej warstwy był połączony z każdym neuronem kolejnej. Dodatkowo, do przeprowadzenia części doświadczeń konieczne było utworzenie neuronu stronniczego (ang. biased neuron). Taki neuron niezależnie od danych jakie dostanie jako sygnał wejściowy, zwraca jedną, z góry określoną wartość wyjściową, najczęściej po prostu 1. Obecność takiego neuronu ma na celu zwiększenie elastyczności sieci poprzez poprawienie jej reakcji na skrajne, znacznie odbiegające od danych treningowych lub po prostu bliskie zeru dane wejściowe.

(1)

Kolejnymi czynnikami wpływającymi na proces uczenia się sieci są prędkość oraz pęd (ang. momentum) uczenia. Podczas procesu propagacji wstecznej błąd *δ* (1) obarczony poprzednim rezultatem z każdego neuronu wyjściowego jest przekazywany wraz z wagami połączeń do ostatniej warstwy ukrytej a następnie wraz z ich błędami głębiej w sieć. W tym procesie wagi każdego z neuronów są poprawiane (2), a współczynnik wpływu błędu na modyfikację wagi to prędkość uczenia *η*. Dodatkowo korzystnym jest wykorzystanie wiedzy o modyfikacjach wagi w poprzednich krokach. Zastosowanie poprzedniej zmiany wagi może w znacznym stopniu przyspieszyć uczenie sieci. Współczynnik wpływu poprzedniej modyfikacji wagi na aktualną to pęd *α* (ang. momentum).

(2)

**3. Opis implementacji**

Program został napisany obiektowo w języku Java, z wykorzystaniem biblioteki JFreeChart do generowania wykresów.

Jedną z najistotniejszych klas z punktu widzenia kodu jest klasa Neuron, będąca modelem pojedynczego neuronu. Klasa Neuron przechowuje dane wag wejść, wektor wejściowy, sygnał wyjściowy neuronu, błąd, a także funkcję aktywacji każdego neuronu. Klasa ta ma zaimplementowane metody obliczające odpowiednie wartości, np. sumowanie sygnałów wejściowych, obliczanie sygnału wyjściowego, czy modyfikację wag w ramach uczenia.

Po klasie Neuron dziedziczą klasy HiddenNeuron, OutputNeuron i BiasedNeuron, które różnią się chociażby sposobem implementacji uczenia neuronów.

Dodatkowymi klasami są klasy odpowiedzialne za modelowanie funkcji aktywacji: liniowej (dla neuronu z warstwy wyjściowej), lub sigmoidalnej (dla neuronów z warstw ukrytych). Klasy te mają również metody zwracające wartość pochodnej funkcji – implementacja pochodnej jest w postaci wzoru wpisanego ręcznie w kodzie.

Klasa NeuralNetwork odpowiada za przechowywanie   
neuronów-obiektów w odpowiednich strukturach danych a także za sam algorytm propagacji sygnału przez sieć, oraz wstecznej propagacji błędu. Ma metody przekazujące dane pomiędzy neuronami, obliczające błąd, zwracające wyjście sieci, itp.

**4. Materiały i metody**

W funkcji main naszego programu, na początku generujemy zbiór danych, który ma przypominać ciągłą funkcję . Jest to zbiór punktów , … , takich, że dla każdego : , oraz dla każdego : . Ten zbiór punktów reprezentuje naszą funkcję **aproksymowaną**, a także posłuży jako **zbiór argumentów testowych**, by wygenerować podobną – „prawie-ciągłą” funkcję **aproksymującą**.

Następnie, wybraliśmy liczbę punktów treningowych = 50. Zostało wylosowanych 50 liczb z przedziału (1; 100), oraz każdej z tych liczb została przyporządkowana wartość funkcji . W ten sposób otrzymaliśmy listę punktów **treningowych**.

W kolejnym kroku została stworzona sieć neuronowa o 4 warstwach ukrytych oraz o 10 neuronach w każdej z ukrytych warstw.

Następnie sieć została nauczona na podstawie danych treningowych, z następującymi parametrami uczenia sieci:

prędkość uczenia 0.0005,

pęd uczenia 0.6,

liczba epok uczenia 3000

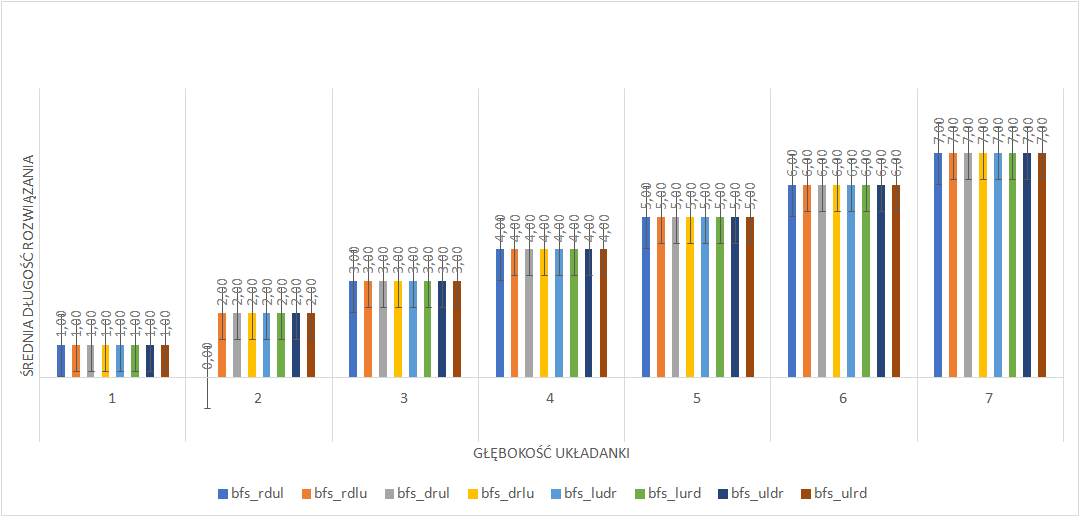
Po zakończeniu uczenia, na wejście sieci została wprowadzona każda z liczb *x* ze **zbioru argumentów testowych**, i zostały utworzone pary liczb takich, że jest odpowiedzią sieci na wejście . W ten sposób powstała lista punktów odpowiadająca **funkcji aproksymującej**.

Eksperyment powtórzono dla różnych parametrów sieci (liczba neuronów, liczba warstw), a także dla różnych parametrów uczenia sieci (prędkość uczenia, pęd uczenia, liczba epok). Wyniki działania sieci zostały przedstawione na wykresach poniżej.

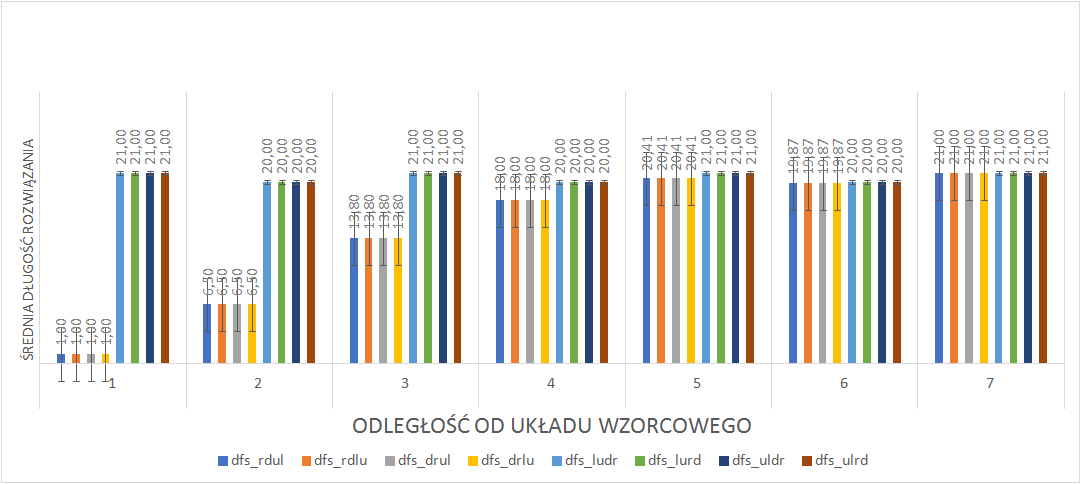
**5. Wyniki**

**5.1. Średnia długość rozwiązania**

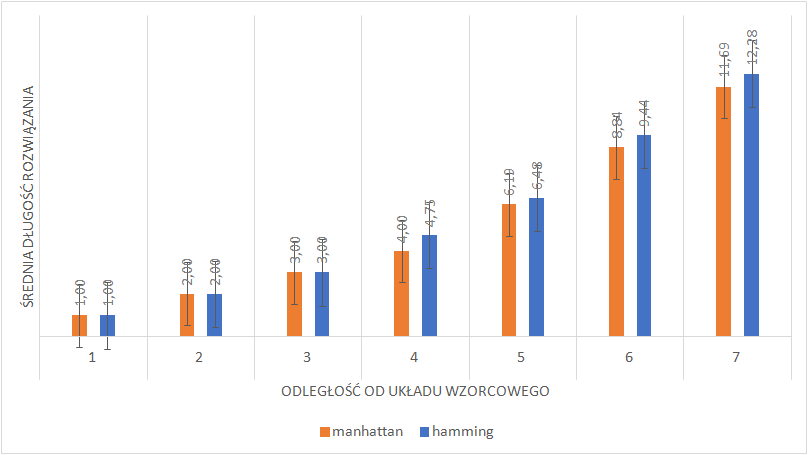
Rysunek 2. Średnia długość rozwiązania



Rysunek 3. BFS - średnia długość rozwiązania dla poszczególnych porządków przeszukiwania sąsiedztwa

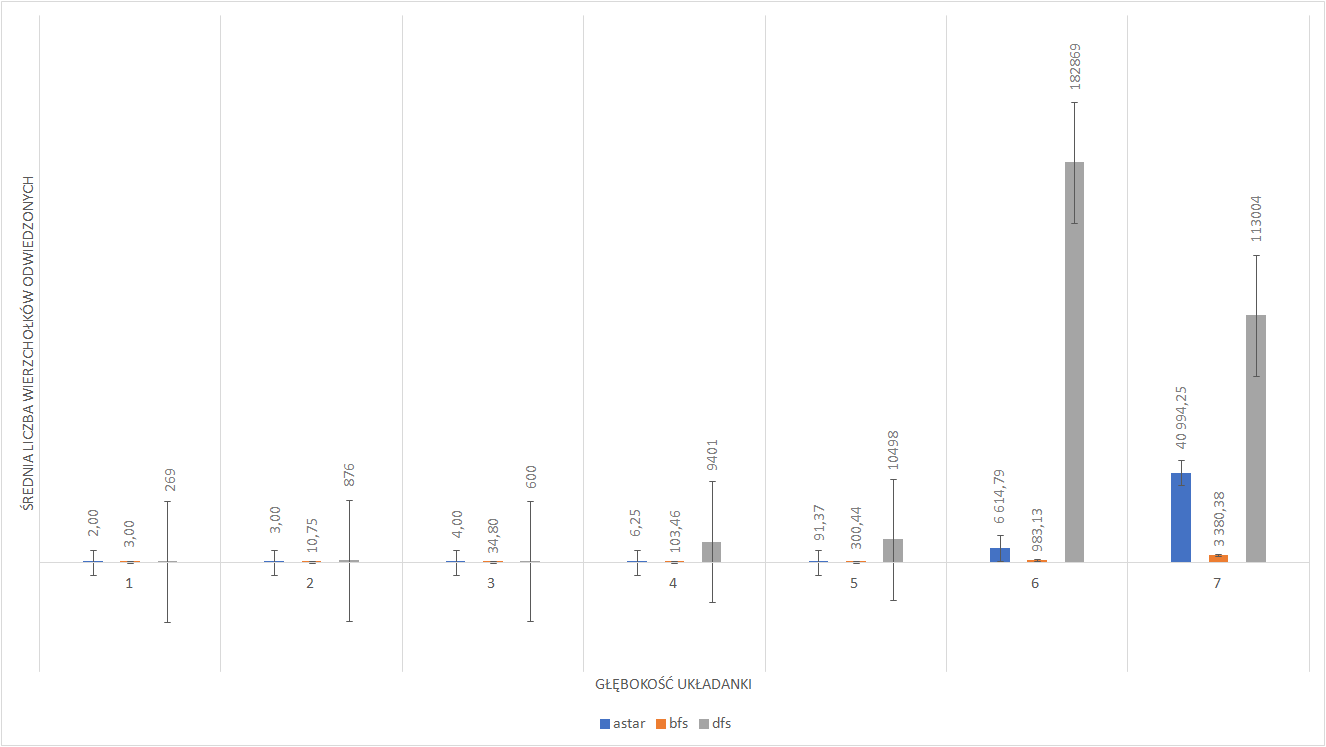


Rysunek 4. DFS - średnia długość rozwiązania dla poszczególnych porządków przeszukiwania sąsiedztwa

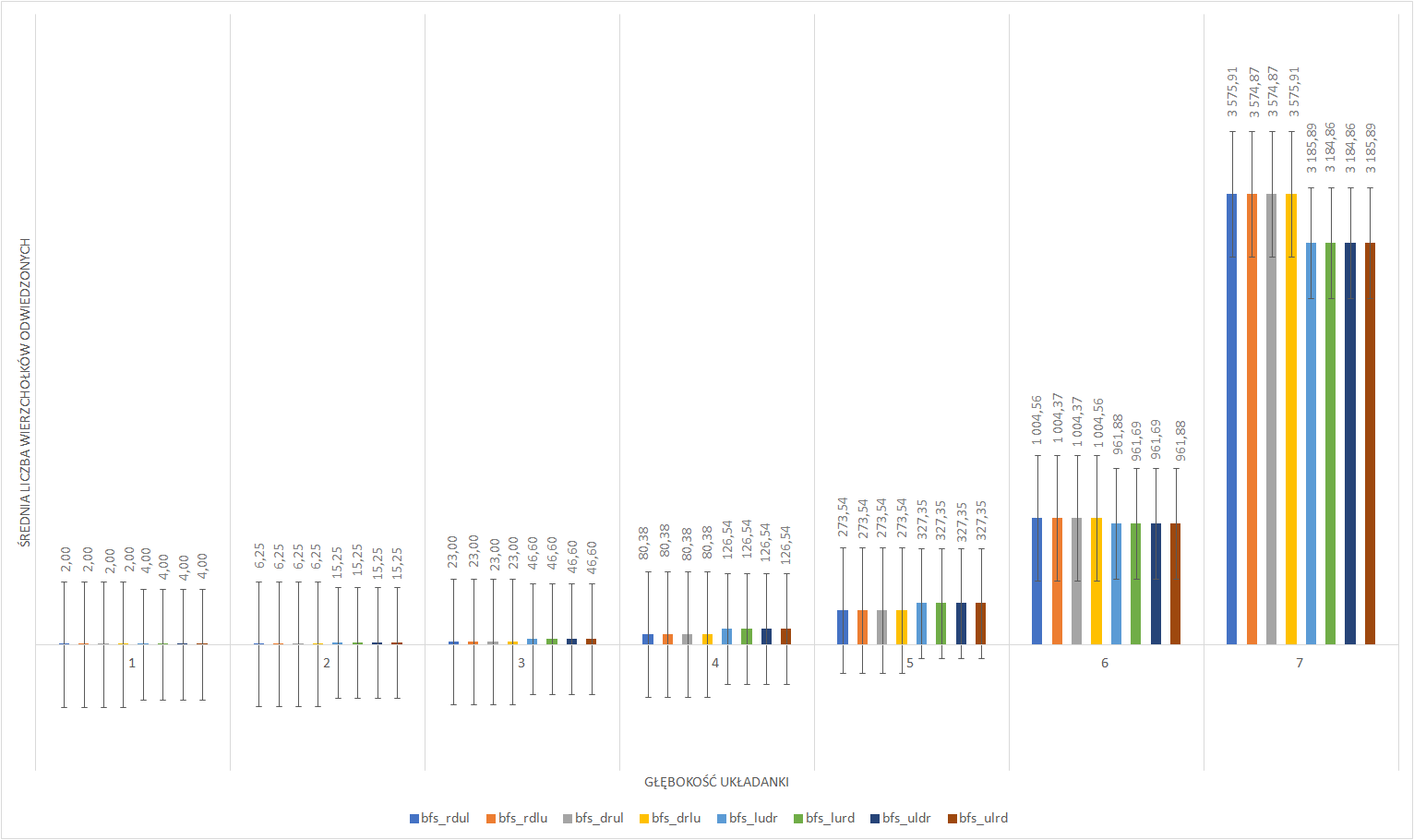


Rysunek 5. A\* - średnia długość rozwiązania dla różnych heurystyk

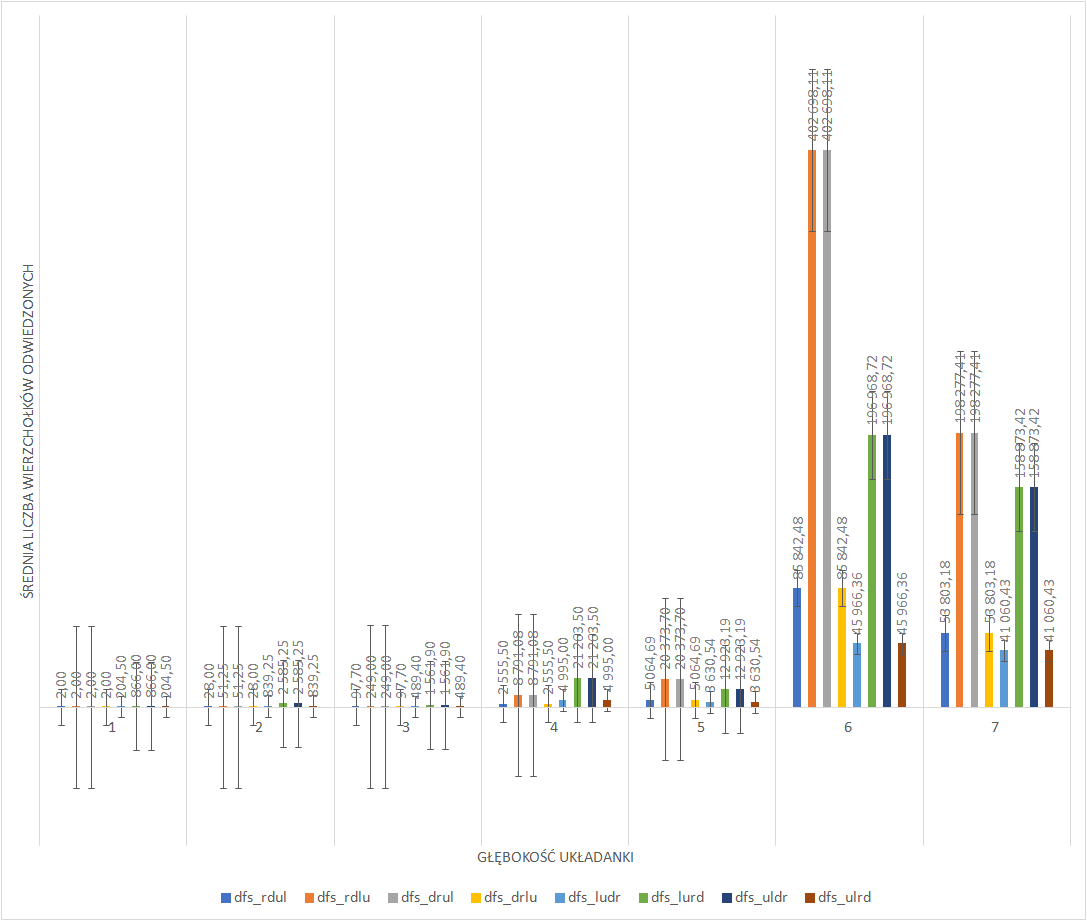
**5.2. Średnia liczba wierzchołków odwiedzonych**



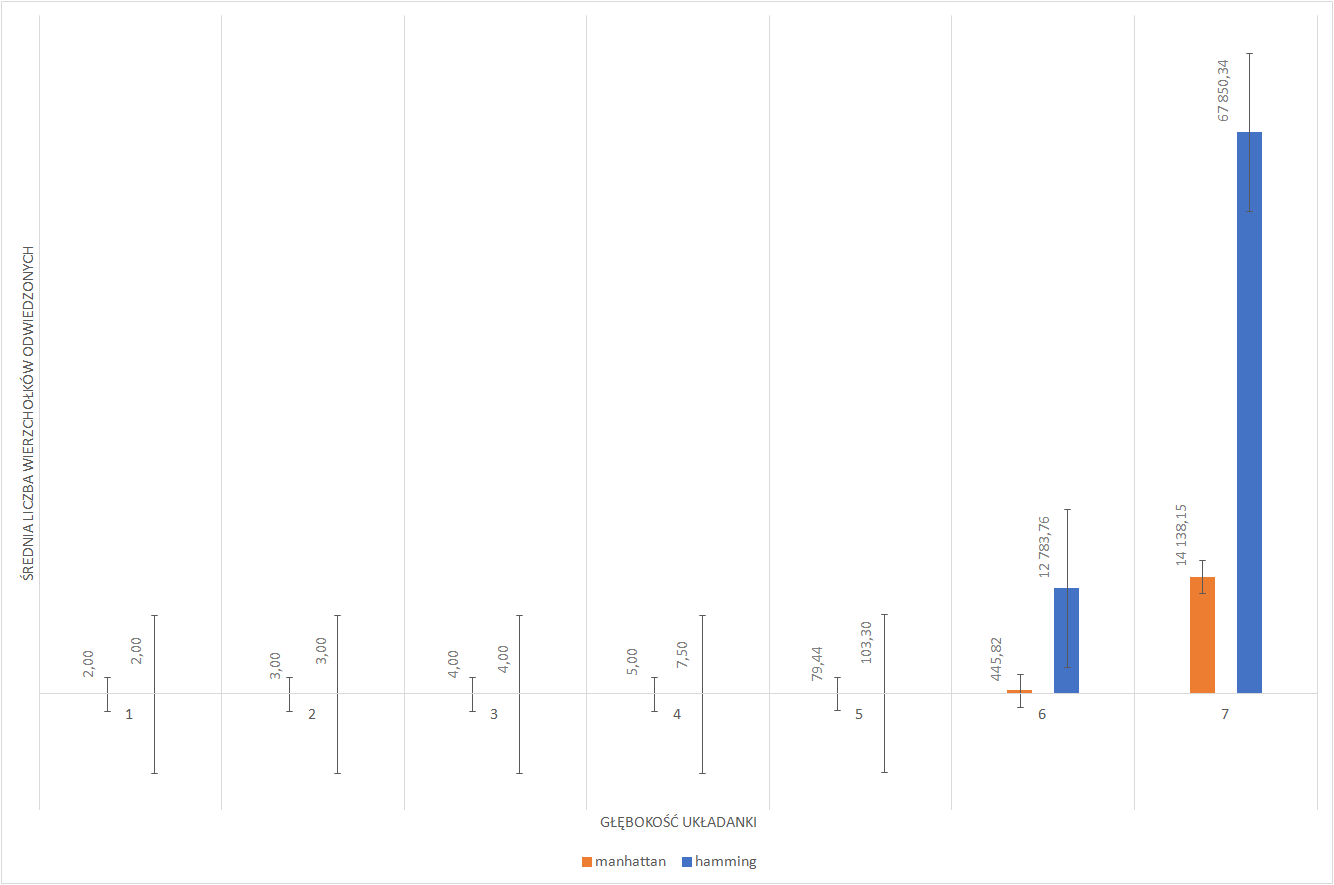
Rysunek 6. Średnia liczba wierzchołków odwiedzonych



Rysunek 7. BFS – średnia liczba wierzchołków odwiedzonych dla poszczególnych porządków przeszukiwania sąsiedztwa

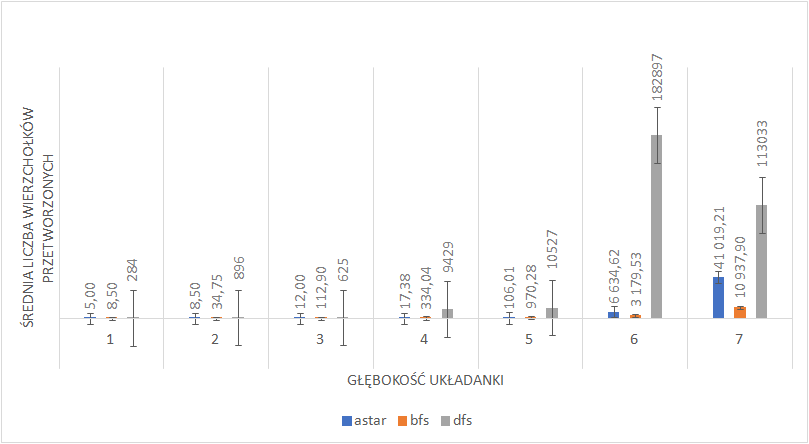


Rysunek 8. DFS – średnia liczba wierzchołków odwiedzonych dla poszczególnych porządków przeszukiwania sąsiedztwa

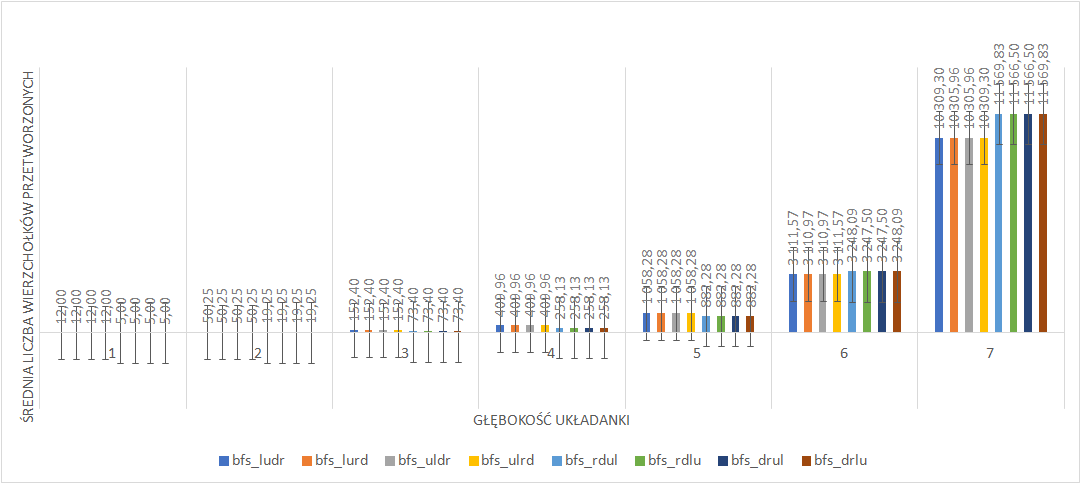


Rysunek 9. A\* - średnia liczba wierzchołków odwiedzonych dla poszczególnych heurystyk

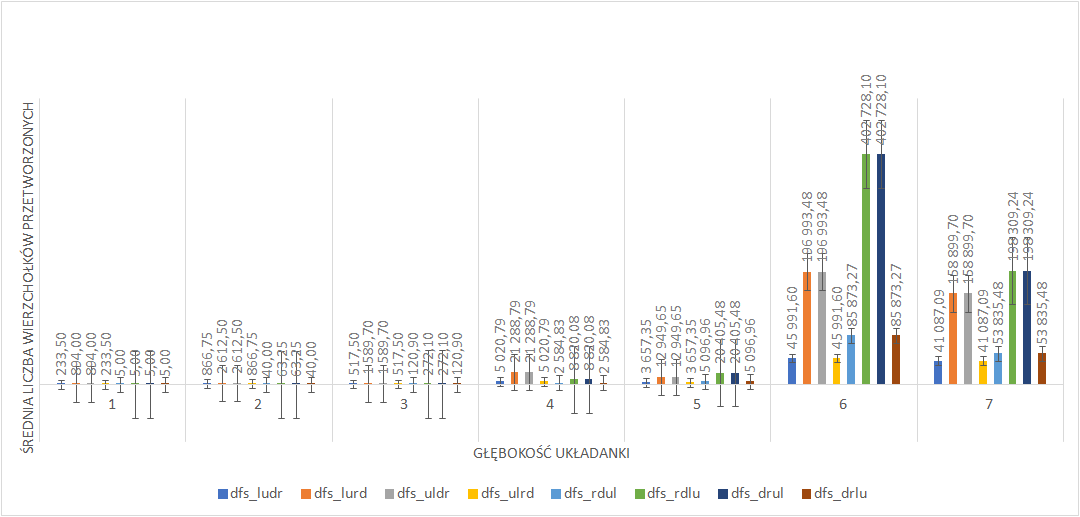
**5.3. Średnia liczba wierzchołków przetworzonych**



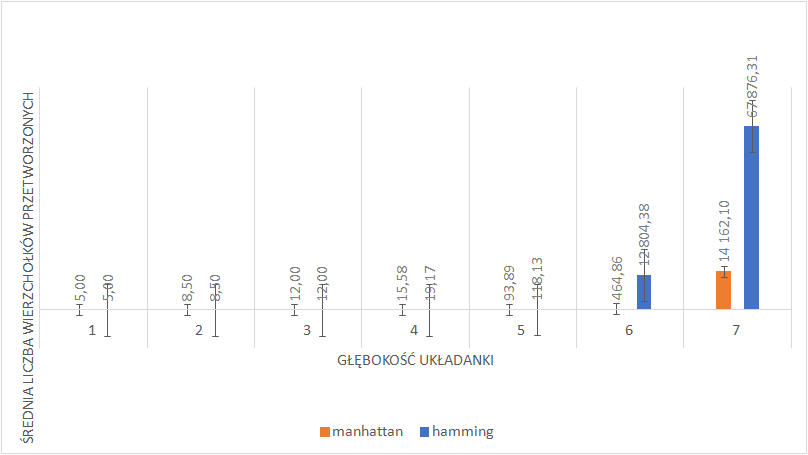
Rysunek 10. Średnia liczba wierzchołków przetworzonych



Rysunek 11. BFS – średnia liczba wierzchołków przetworzonych dla poszczególnych porządków przeszukiwania sąsiedztwa

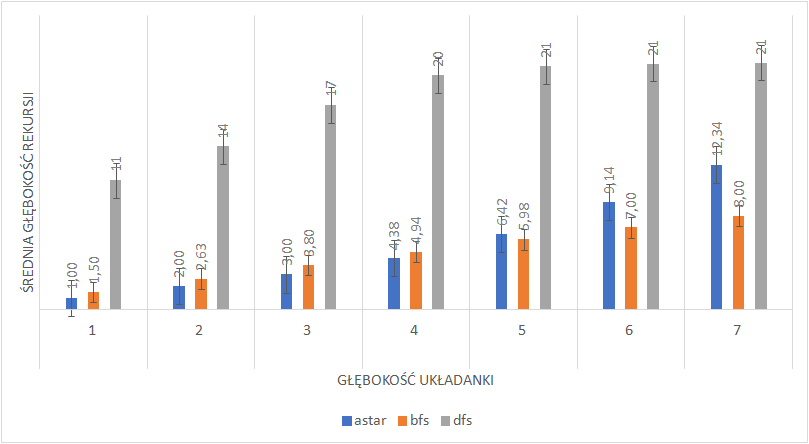


Rysunek 12. DFS – średnia liczba wierzchołków przetworzonych dla poszczególnych porządków przeszukiwania sąsiedztwa

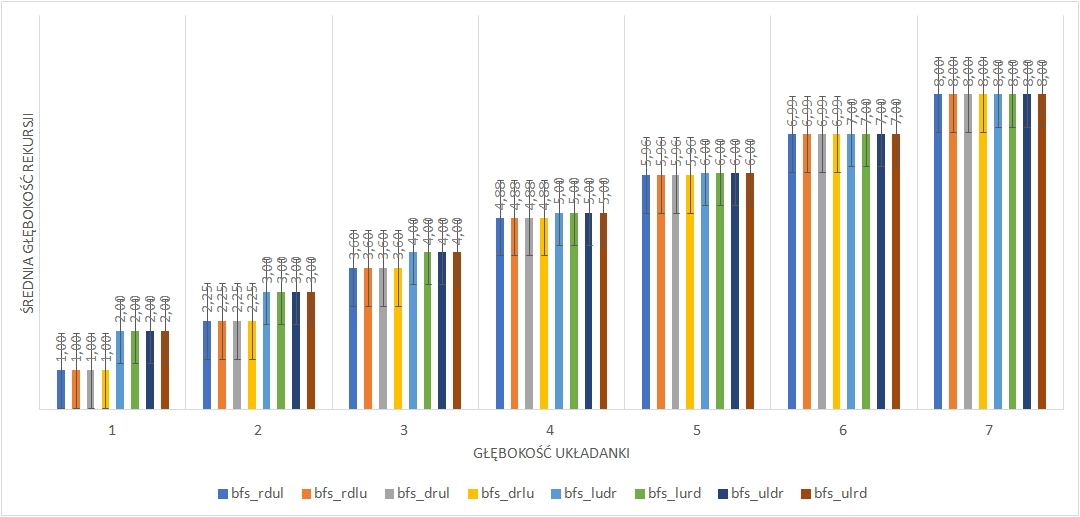


Rysunek 13. A\* -średnia liczba wierzchołków przetworzonych dla poszczególnych heurystyk

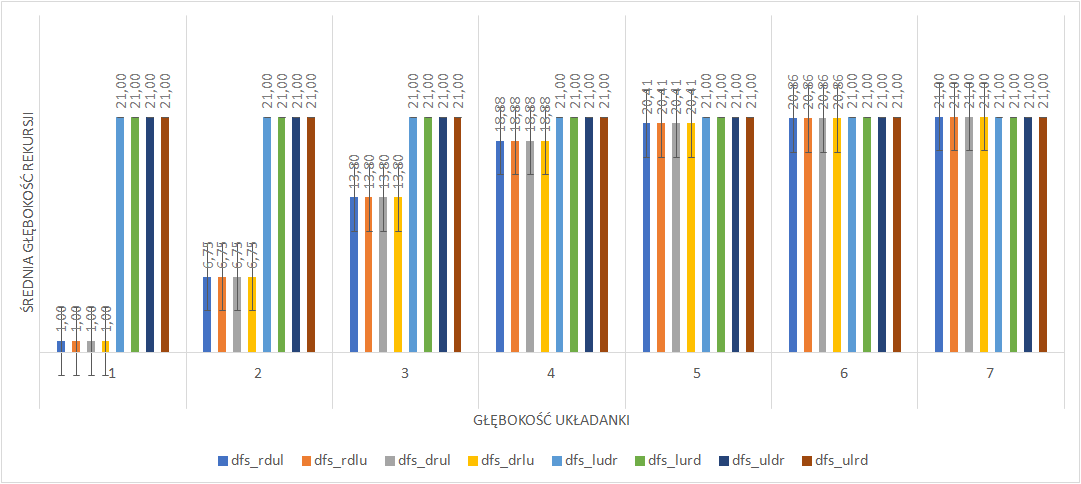
**5.4. Średnia maksymalna głębokość rekursji**



Rysunek 14. Średnia maksymalna głębokość rekursji



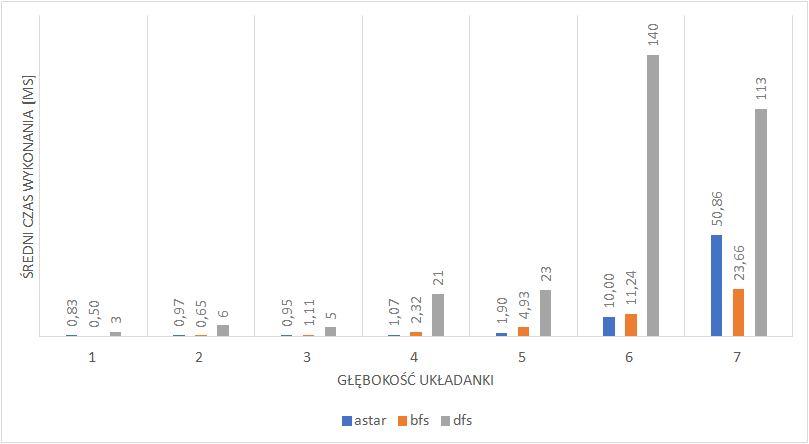
Rysunek 15. BFS – średnia maksymalna głębokość rekursji dla poszczególnych porządków przeszukiwania sąsiedztwa



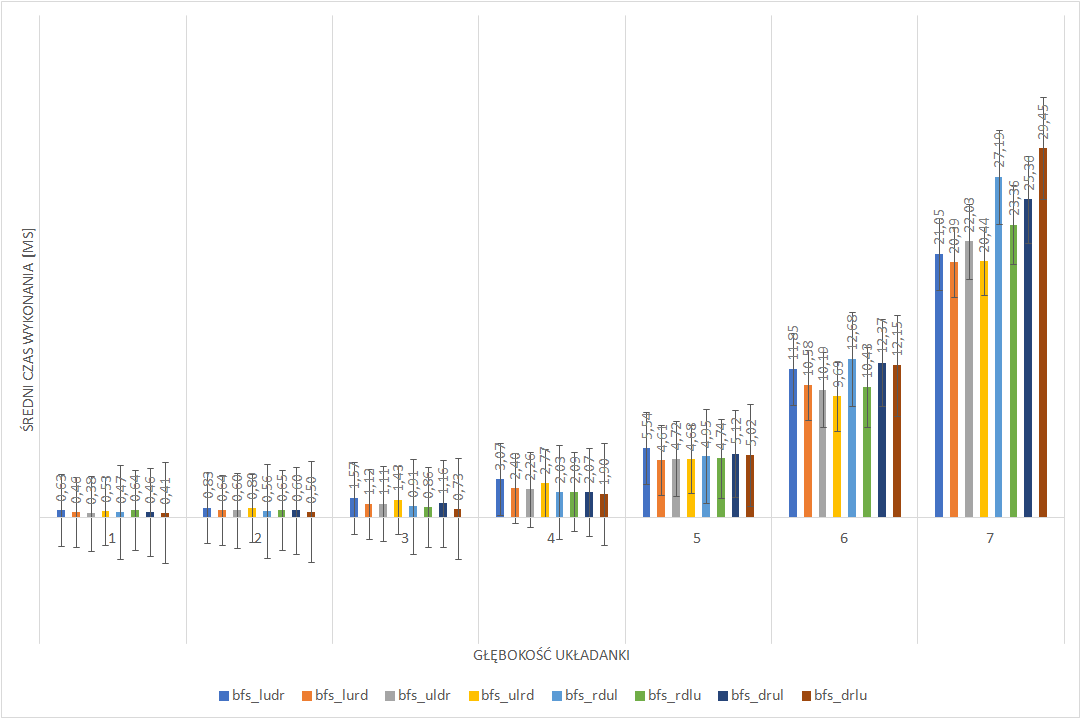
Rysunek 16. DFS – średnia maksymalna głębokość rekursji dla poszczególnych porządków przeszukiwania sąsiedztwa

Rysunek 17. A\* -średnia maksymalna głębokość rekursji dla poszczególnych heurystyk

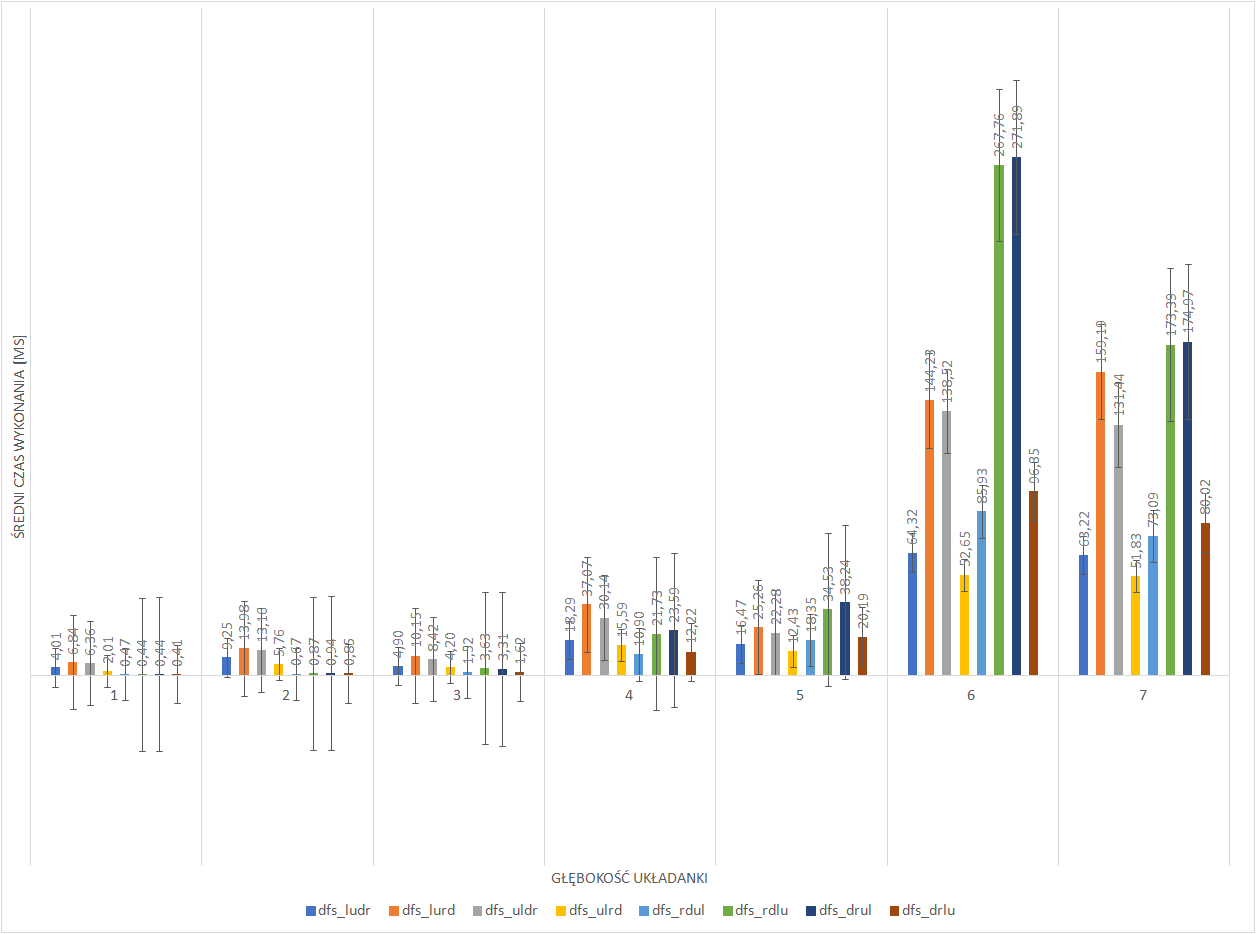
**5.5. Średni czas trwania procesu obliczeniowego**



Rysunek 18. Średni czas trwania procesu obliczeniowego



Rysunek 19. BFS – średni czas trwania procesu obliczeniowego dla poszczególnych porządków przeszukiwania sąsiedztwa



Rysunek 20. DFS – średni czas trwania procesu obliczeniowego dla  poszczególnych porządków przeszukiwania sąsiedztwa

Rysunek 21. A\* - średni czas trwania procesu obliczeniowego dla poszczególnych heurystyk

**6. Dyskusja**

**6.1. Średnia długość rozwiązania**

Porównując całościowo strategie przeszukiwania, najlepiej z zadaniem rozwiązania układanki poradził sobie BFS. Średnia długość rozwiązania odpowiadała odległości układanki od układu wzorcowego. Z zadaniem rozwiązania układanki dobrze poradziła sobie również strategia A\*. Przy odległościach 1 – 4 długość rozwiązania odpowiadała tym odległościom. Przy większych odległościach układanki dało się zaobserwować zwiększanie się rozbieżności pomiędzy odległością układanki a długością rozwiązania.

Porównując poszczególne porządki przeszukiwania sąsiedztwa, BFS – długość rozwiązania wynosiła tyle co odległość układanki od układu wzorcowego niezależnie od porządku. DFS – przy odległościach 1 -3 widać zdecydowaną przewagę porządków rd../dr.. w stosunku do porządków ul../lu.. . Im większa odległość, tym mniejsza różnica pomiędzy tymi porządkami.

A\* - przy odległościach 1 -3 obie heurystyki sprawdzały się tak samo. Przy większych odległościach w układance, Hamming okazał się być gorszy od Manhattana.

**6.2. Średnia liczba wierzchołków odwiedzonych**