Struktury danych i złożoność obliczeniowa

Zadanie projektowe nr 2

Badanie efektywności algorytmów grafowych w zależności od rozmiaru instancji oraz sposobu reprezentacji grafu w pamięci komputera.

Maja Bojarska

nr indeksu: xxxxxx

8 czerwca 2019

Spis treści

Wstęp

1.1 Opis

W ramach projektu należało zaimplementować określone algorytmy grafowe oraz dokonać pomiarów czasu potrzebnego do ich wykonania w zależności od ilości wierzchołków i gęstości grafu.

1.1.1 Zrealizowane algorytmy

- Minimalne drzewo rozpinające
  - Algorytm Prima
  - Algorytm Kruskala
- Problem najkrótszej ścieżki
  - Algorytm Dijkstry
  - Algorytm Bellmana Forda

Powyższe algorytmy zostały zbadane dla każdej możliwej kombinacji liczb wierzchołków: 10, 25, 50, 75, 100 i gęstości grafu: 25%, 50%, 75%, 99%.

1.2 Założenia

1.2.1 Język programowania

Program został napisany obietkowo w języku C++ (standard C++11). Do implementacji algorytmów oraz struktur danych nie zostały wykorzystane narzędzia, takie jak STL, Boost lub inne podobne.

1.2.2 Narzędzia pomiarowe

Do pomiaru czasu rozpoczęcia i zakończenia operacji została wykorzystana klasa *std::chrono::high_resolution_clock*, która umożliwia pomiar czasu z dokładnością do jednej nanosekundy.

1.2.3 Metoda pomiarowa

W celu uzyskania wiarygodnych wyników pomiaru czasu, dla każdego zestawu parametrów wykonane po 100 prób. Na podstawie tak otrzymanego zbioru wyników z pojedynczych prób, policzone zostały średnie arytmetyczne, które stanowią ostateczny rezultat pomiarów przedstawiony w formie tabelarycznej oraz wykresu.

1.2.4 Dane wejściowe

Implementacja programu zakłada, że dane wejściowe zostaną wygenerowane losowo lub wczytane z pliku tekstowego (format jak specyfikacji wymagań zadania projektowego). Zakłada się również, że graf wejściowy jest spójny. W przeciwnym wypadku, program wyświetli komunikat dotyczący niespójności grafu, a uruchomiony algorytm grafowy zostanie przerwany.

1.2.5 Inne wykorzystane narzędzia

Wielokrotnie zmierzone czasy wykonania każdej próby dla danej operacji zostały zbiorowo zapisane w plikach w formacie .csv, który umożliwił łatwe wczytanie ich przez dowolny program obsługujący arkusze kalkulacyjne. Uśrednienie wyników zostało wykonane za pomocą pakietu Microsoft Excel 2017. Dokument został złożony w systemie LaTeX.

Minimalne drzewo rozpinające

2.1 Algorytm Prima

2.1.1 Teoretyczna złożoność obliczeniowa

$$O(ElogV)$$

Do sortowania rozpatrywanych krawedzi malejąco wzgledem wagi został wykorzystany kopiec typu minimum. Wykorzystanie tej struktury daje lepszą (niższą) pesymistyczną złożoność obliczeniową, niż kolejka priorytetowa, która skutkowałaby złożonością $O(ElogV)$.

2.1.2 Wyniki pomiarów

Wartości podane są w mikrosekundach.

Implementacja macierzowa

	Liczba wierzchołków				
Gęstość	10	25	50	75	100
25	83.502	459.849	1800.759	4013.412	7559.005
50	112.899	785.957	3295.278	8380.163	17141.571
75	162.179	1141.605	4926.099	12714.848	25403.364
99	214.216	1548.844	6818.221	17502.845	34148.497

Implementacja listowa

	Liczba wierzchołków				
Gęstość	10	25	50	75	100
25	83.502	459.849	1800.759	4013.412	7559.005
50	112.899	785.957	3295.278	8380.163	17141.571
75	162.179	1141.605	4926.099	12714.848	25403.364
99	214.216	1548.844	6818.221	17502.845	34148.497

2.2 Algorytm Kruskala

2.2.1 Teoretyczna złożoność obliczeniowa

$$O(ElogE)$$

Do sortowania krawędzi względem wag wykorzystany został kopiec typu minimum.

Algorytm Kruskala ma gorszy czas wykonania niż algorytm Prima, dla jednego wątku. Jednak ze względu na wykorzystanie struktury zbiorów rozłącznych, można go skutecznie zrównoleglić na maszynie dysponującej wieloma procesorami, co znacząco skróci całkowity czas wywołania algorytmu.

2.2.2 Wyniki pomiarów

Wartości podane są w mikrosekundach.

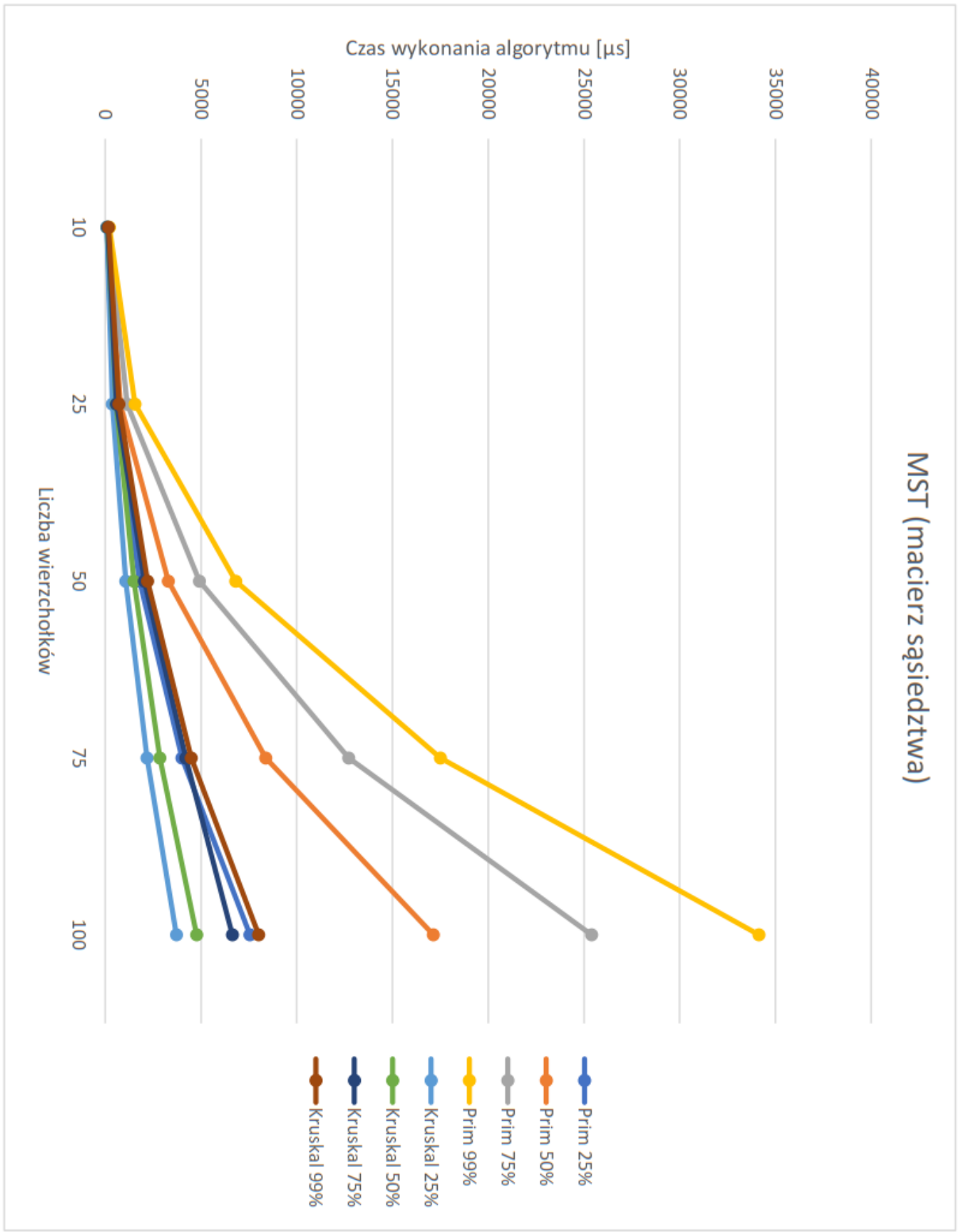
Implementacja macierzowa

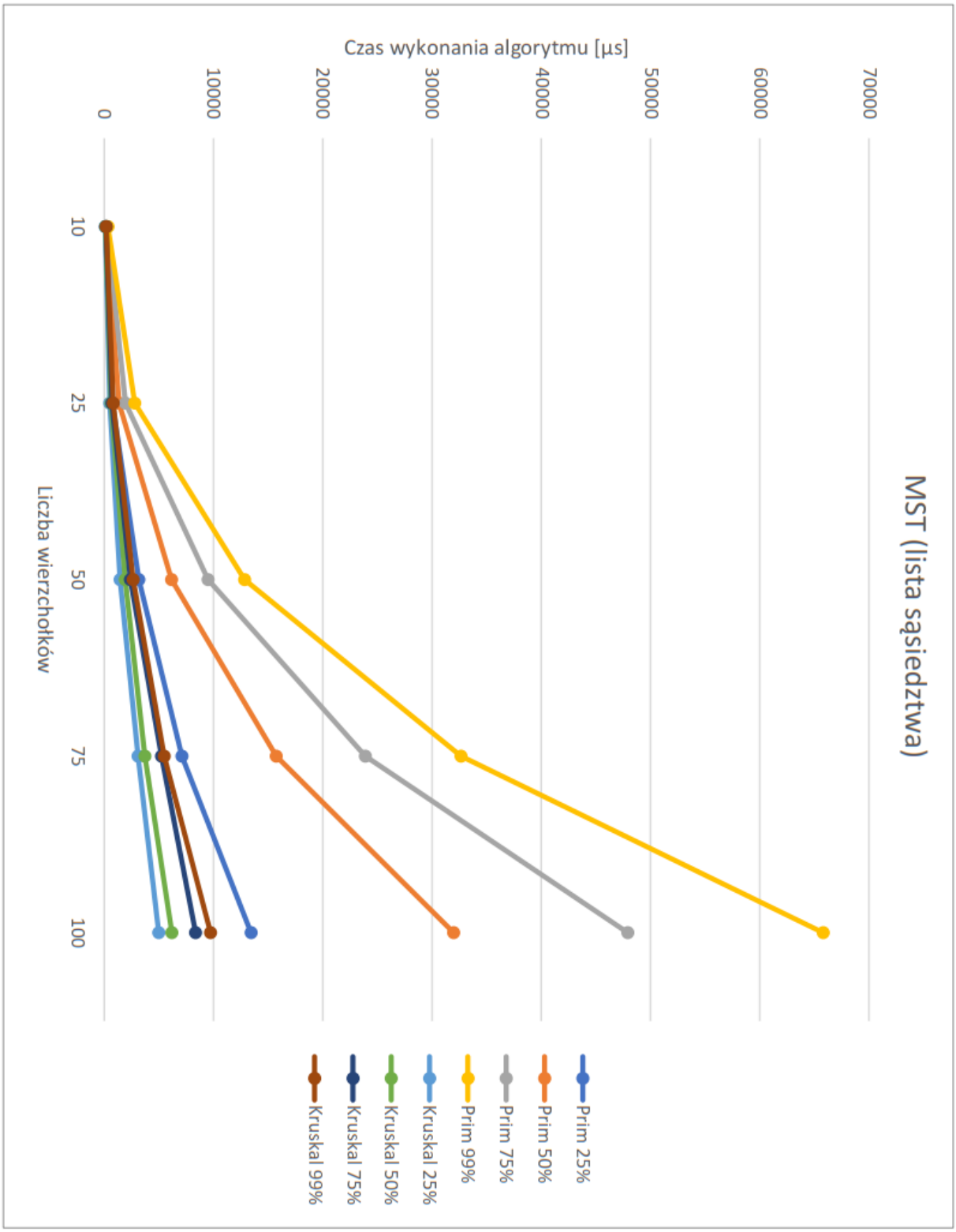
	Liczba wierzchołków				
Gęstość	10	25	50	75	100
25	82.901	368.578	1077.135	2182.742	3715.008
50	119.098	566.803	1500.982	2849.435	4770.979
75	136.385	592.724	2046.211	4213.245	6637.562
99	173.433	719.953	2209.304	4494.381	8015.769

Implementacja listowa

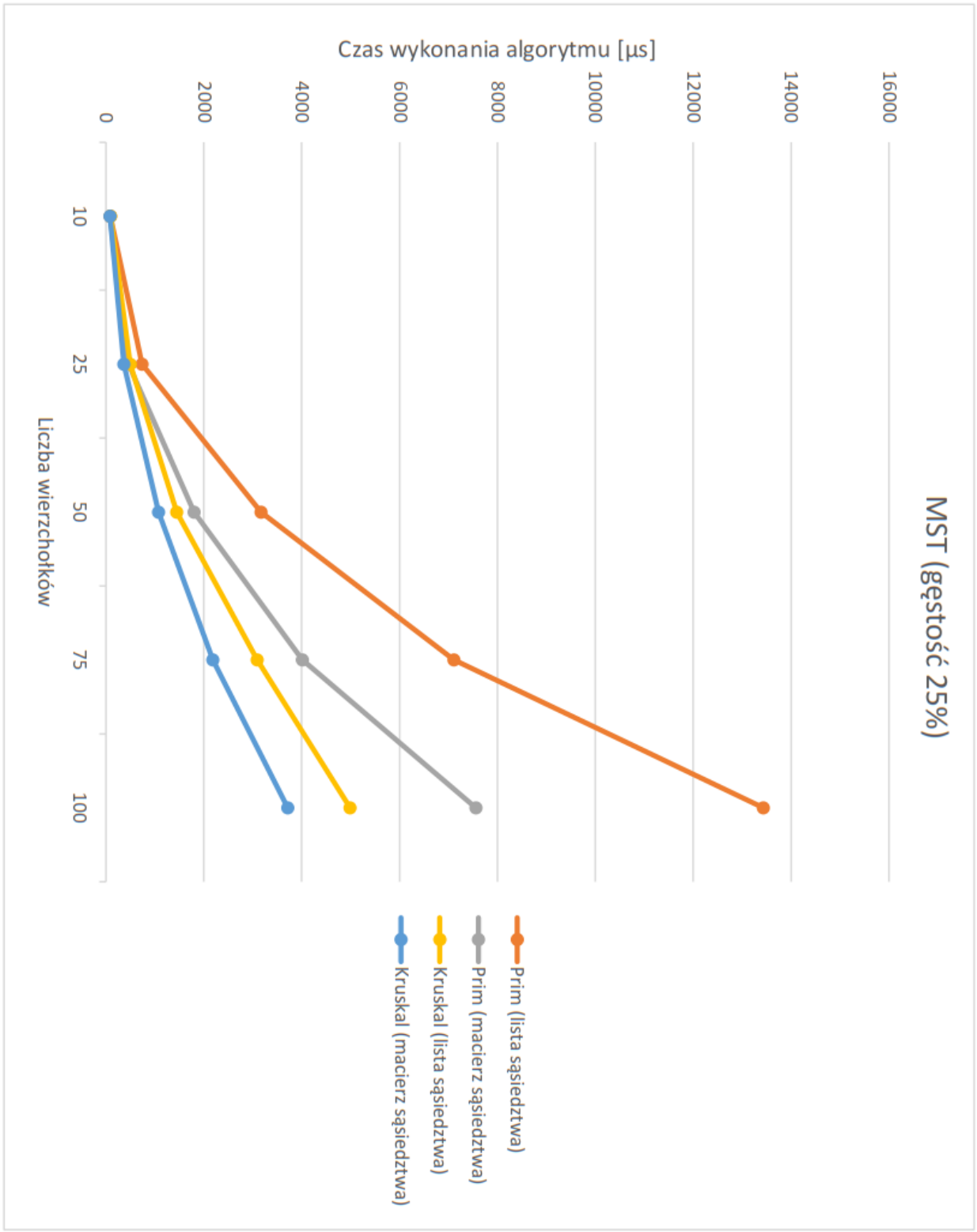
	Liczba wierzchołków				
Gęstość	10	25	50	75	100
25	96.497	496.252	1449.179	3093.235	4989.591
50	122.183	624.336	1923.969	3728.903	6193.025
75	165.308	701.609	2398.343	5250.746	8366.273
99	178.914	829.444	2631.650	5486.263	9731.364

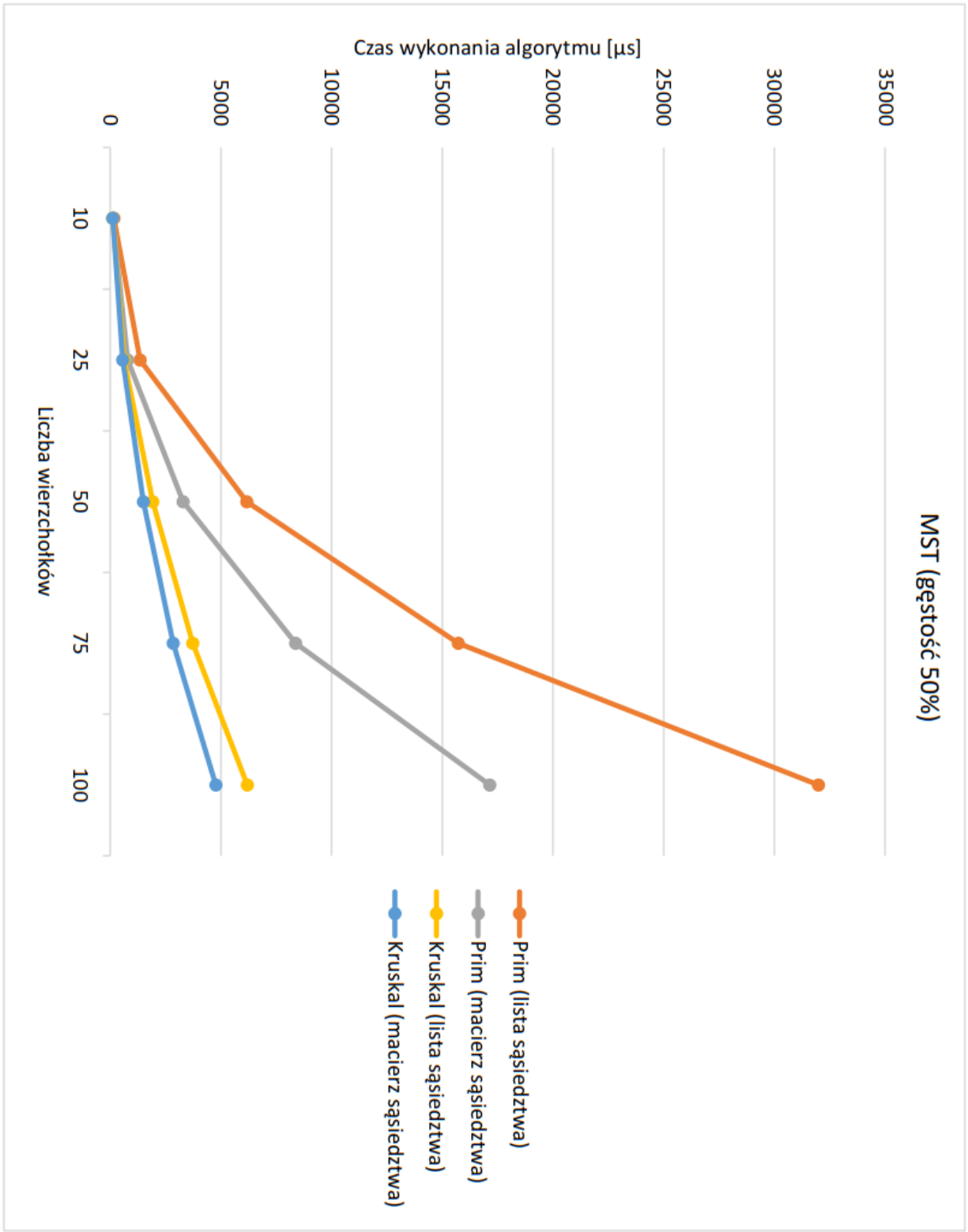
2.3 Wykresy typu I

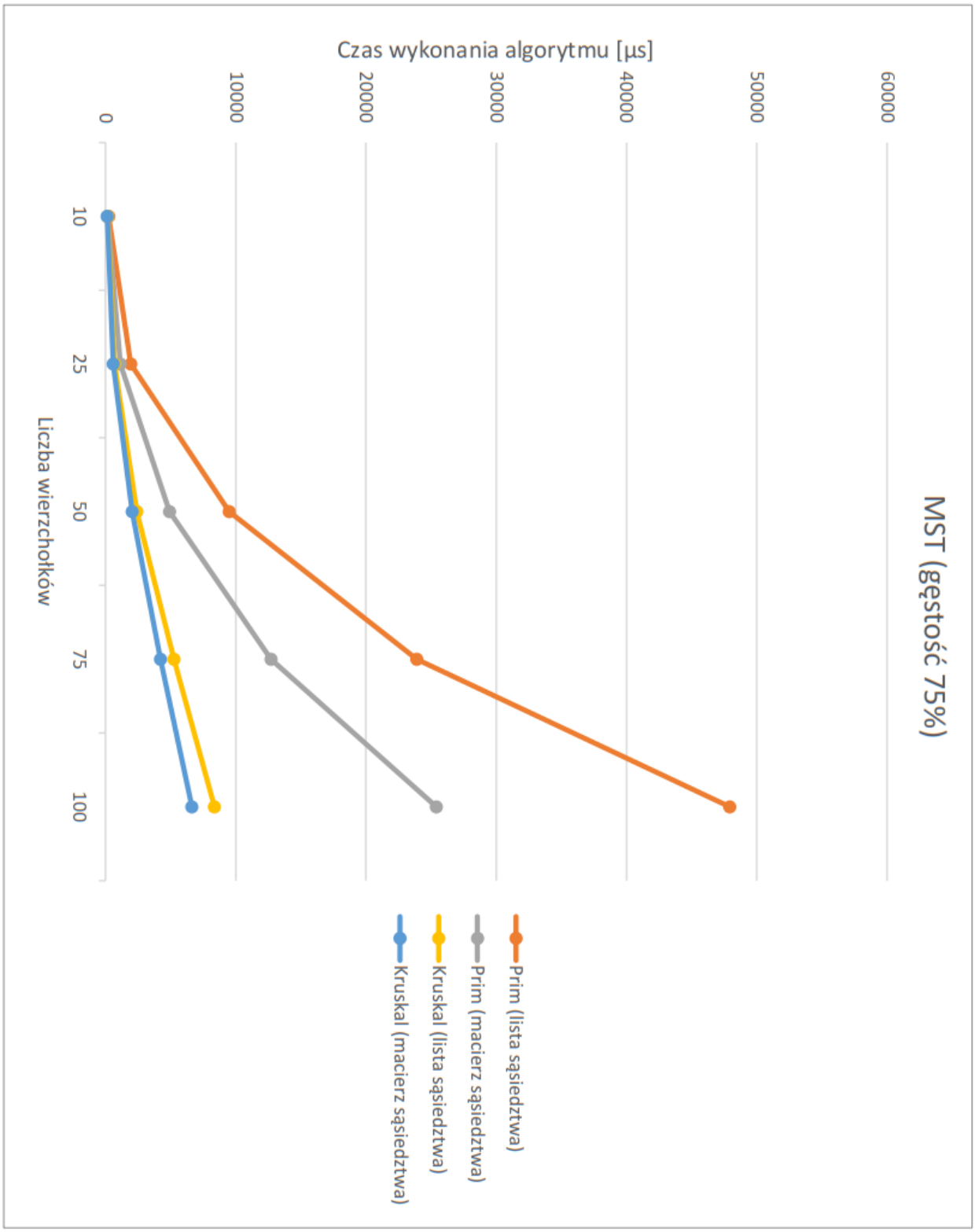


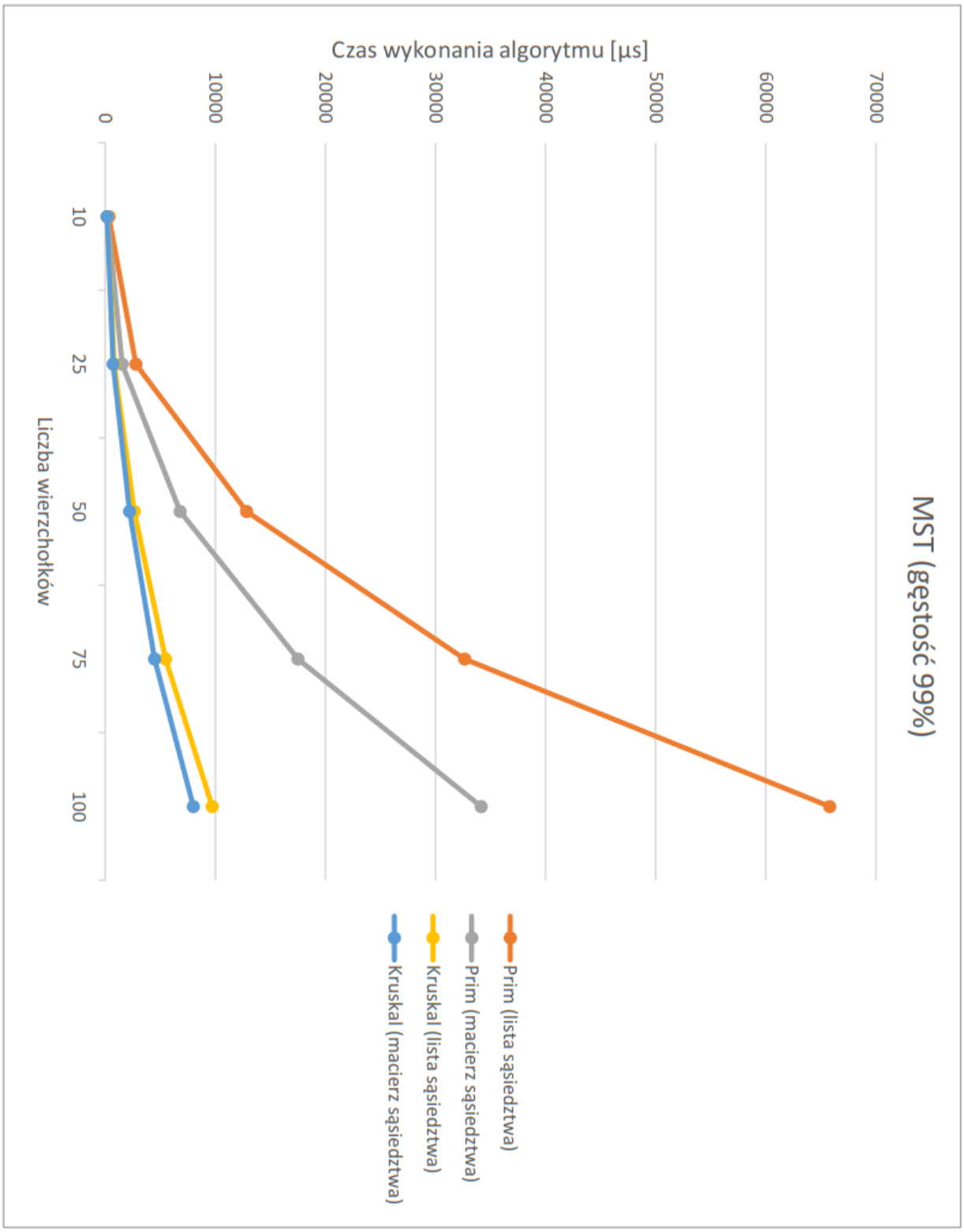

MST (lista sąsiedztwa)
Czas wykonania algorytmu [µs]
0
10000
20000
30000
40000
50000
60000
70000
Liczba wierzchołków
10
25
50
75
100
Prim 25%
Prim 50%
Prim 75%
Prim 99%
Kruskal 25%
Kruskal 50%
Kruskal 75%
Kruskal 99%

2.4 Wykresy typu II




MST (gęstość 50%)
Czas wykonania algorytmu [µs]
0
5000
10000
15000
20000
25000
30000
35000
Liczba wierzchołków
10
25
50
75
100
Prim (lista sąsiedztwa)
Prim (macierz sąsiedztwa)
Kruskal (lista sąsiedztwa)
Kruskal (macierz sąsiedztwa)


MST (gęstość 75%)
Czas wykonania algorytmu [µs]
0
10000
20000
30000
40000
50000
60000
Liczba wierzchołków
10
25
50
75
100
Prim (lista sąsiedztwa)
Prim (macierz sąsiedztwa)
Kruskal (lista sąsiedztwa)
Kruskal (macierz sąsiedztwa)


MST (gęstość 99%)
Czas wykonania algorytmu [µs]
0
10000
20000
30000
40000
50000
60000
70000
Liczba wierzchołków
10
25
50
75
100
Prim (lista sąsiedztwa)
Prim (macierz sąsiedztwa)
Kruskal (lista sąsiedztwa)
Kruskal (macierz sąsiedztwa)

Problem najkrótszej ścieżki

3.1 Algorytm Dijkstry

3.1.1 Teoretyczna złożoność obliczeniowa

$$O(E + VlogV)$$

Algorytm został zaimplementowany w sposób wydajny dzięki zastosowaniu kopca binarnego typu minimum, do ekstrakcji kolejnych nieodwiedzonych wierzchołków, o najmniejszym koszcie dotarcia. Wykorzystanie do tego celu struktury, takiej jak kolejka priorytetowa, też jest możliwe, ale skutkowałoby złożonością obliczeniową $O(E + V^2)$. Takie podejście nadal daje rozwiązanie w czasie wielomianowym, ale gorszym, niż wykorzystanie struktury drzewiastej (kopiec binarny, BST, kopiec Fibonacciego).

3.1.2 Wyniki pomiarów

Wartości podane są w mikrosekundach.

Implementacja macierzowa

	Liczba wierzchołków				
Gęstość	10	25	50	75	100
25	49.256	154.220	442.107	874.388	1573.010
50	46.884	180.987	597.432	1260.776	2468.146
75	54.545	225.960	799.412	1664.349	3148.328
99	58.178	259.879	936.046	2322.594	4585.338

Implementacja listowa

	Liczba wierzchołków				
Gęstość	10	25	50	75	100
25	36.525	142.345	405.416	810.632	1536.803
50	39.541	174.959	536.253	1259.534	2549.639
75	47.549	220.631	836.861	1725.832	3453.629
99	50.996	254.183	970.122	2573.976	5061.502

3.2 Algorytm Bellmana-Forda

3.2.1 Teoretyczna złożoność obliczeniowa

$$O(E * V)$$

Powyższa złożoność wynika z faktu, że w najgorszym wypadku algorytm musi "odwiedzić" wszystkie krawędzie w grafie V razy (gdzie V jest liczbą wierzchołków).

W przeciwieństwie do algorytmu Dijkstry, algorytm Bellmana-Forda jest skuteczny w wyszukiwaniu ścieżek w grafach zawierających cykle ujemne, gdyż kończy się w momencie zakończenia propagacji cykli ujemnych. Przykładowo, algorytm Dijkstry nie jest w stanie rozwiązać problemu najkrótszej ściezki dla grafu z ujemnymi cyklami (mógłby się nigdy nie zakończyć).

3.2.2 Wyniki pomiarów

Wartości podane są w mikrosekundach.

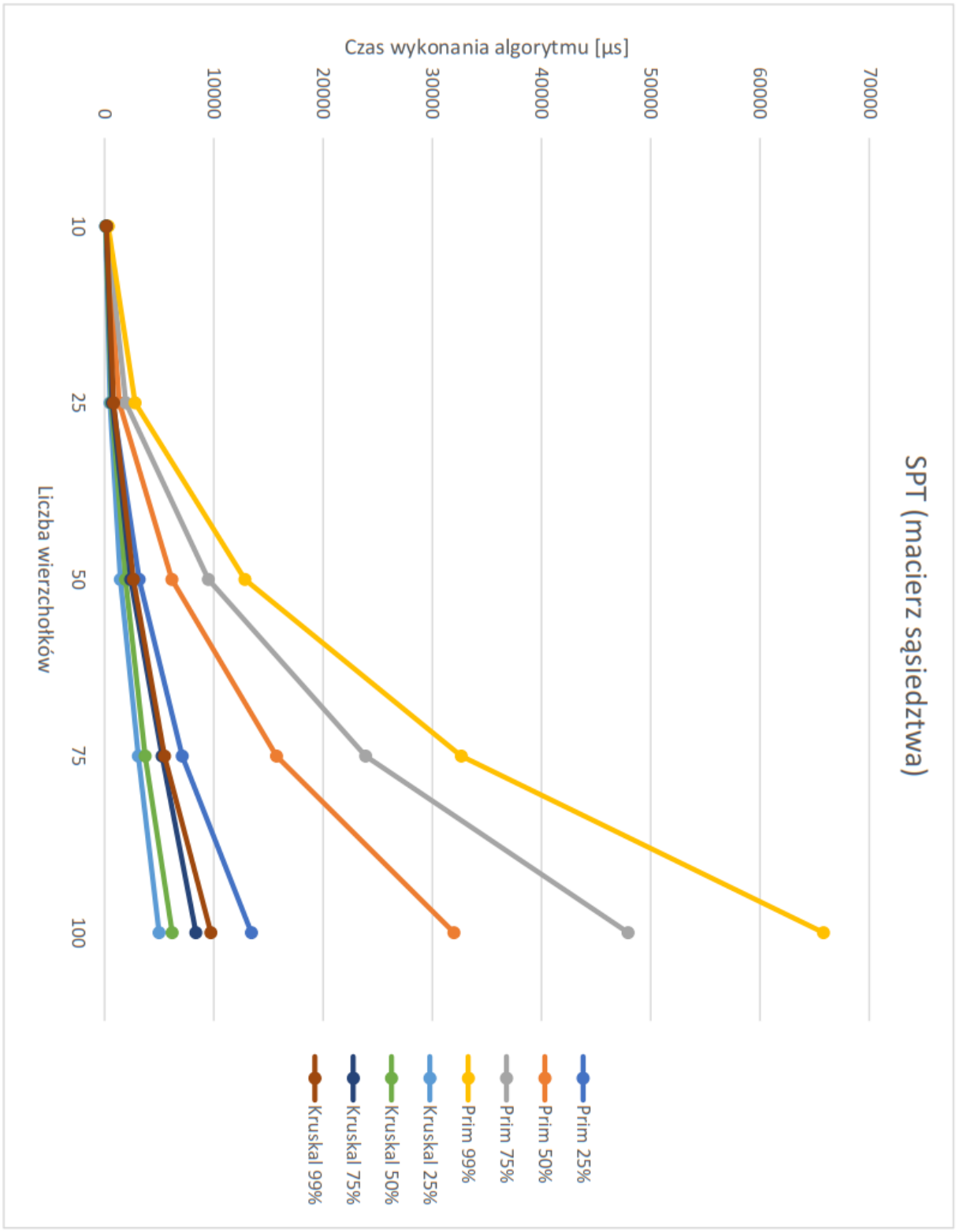
Implementacja macierzowa

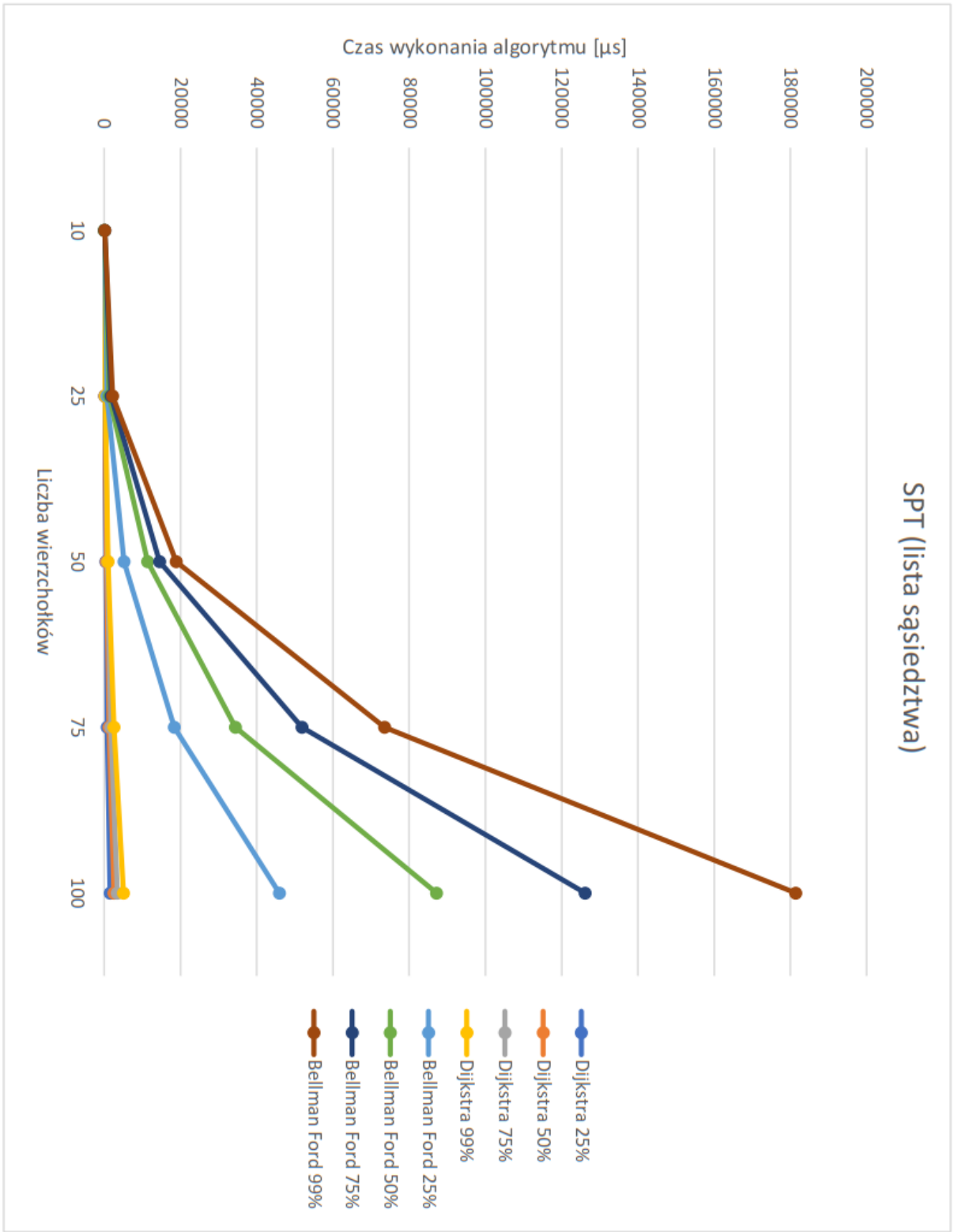
	Liczba wierzchołków				
Gęstość	10	25	50	75	100
25	80.119	1045.582	6763.686	22559.986	49081.009
50	107.903	1507.567	11064.276	39122.752	89486.488
75	158.527	1956.617	15105.559	54991.740	116037.973
99	172.803	2557.564	21828.061	59952.227	149188.883

Implementacja listowa

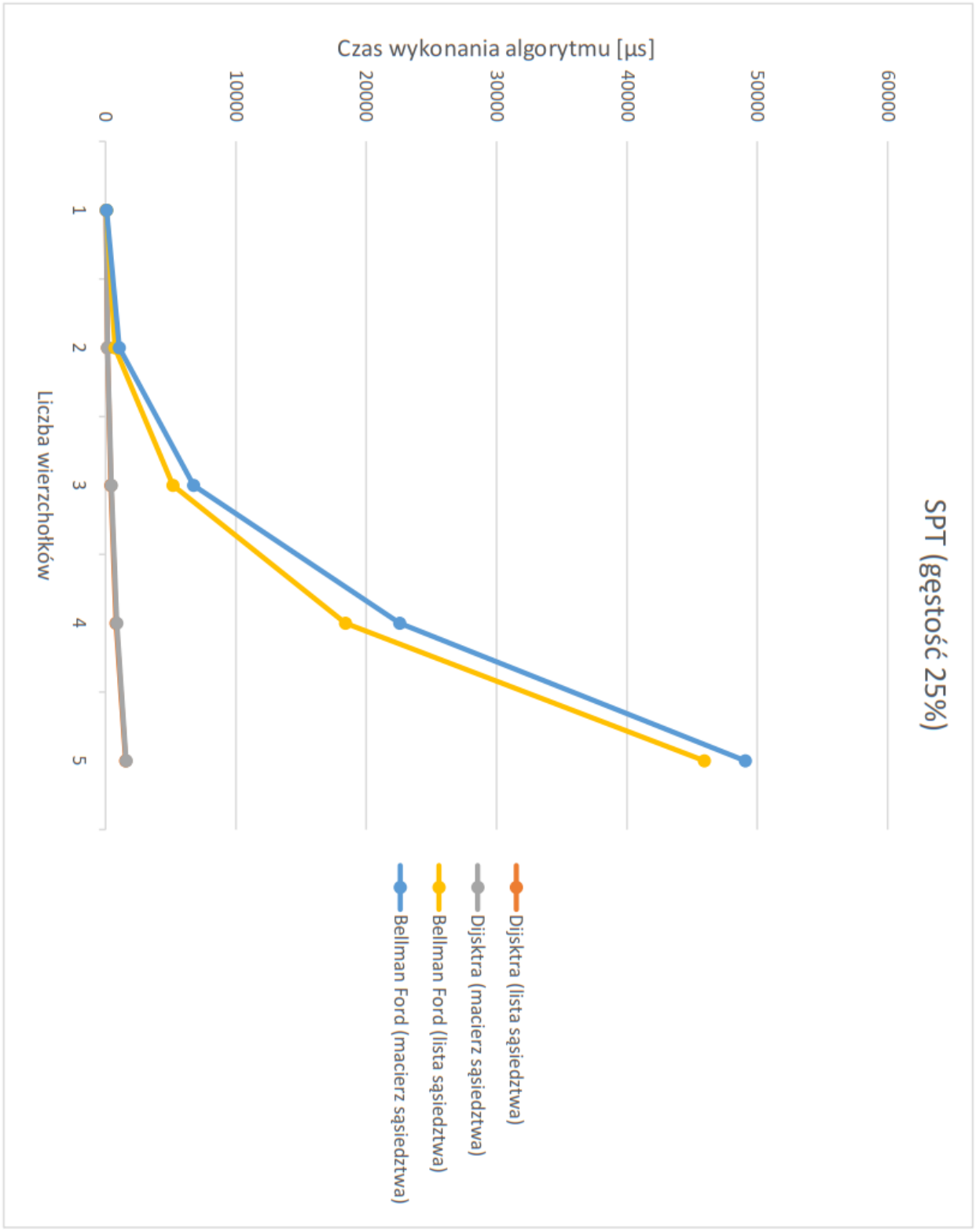
	Liczba wierzchołków				
Gęstość	10	25	50	75	100
25	77.161	715.049	5175.207	18413.758	45941.560
50	96.404	1287.440	11367.856	34396.838	87206.784
75	132.020	1767.974	14505.883	51903.355	126170.960
99	155.789	2222.841	18893.881	73563.443	181436.477

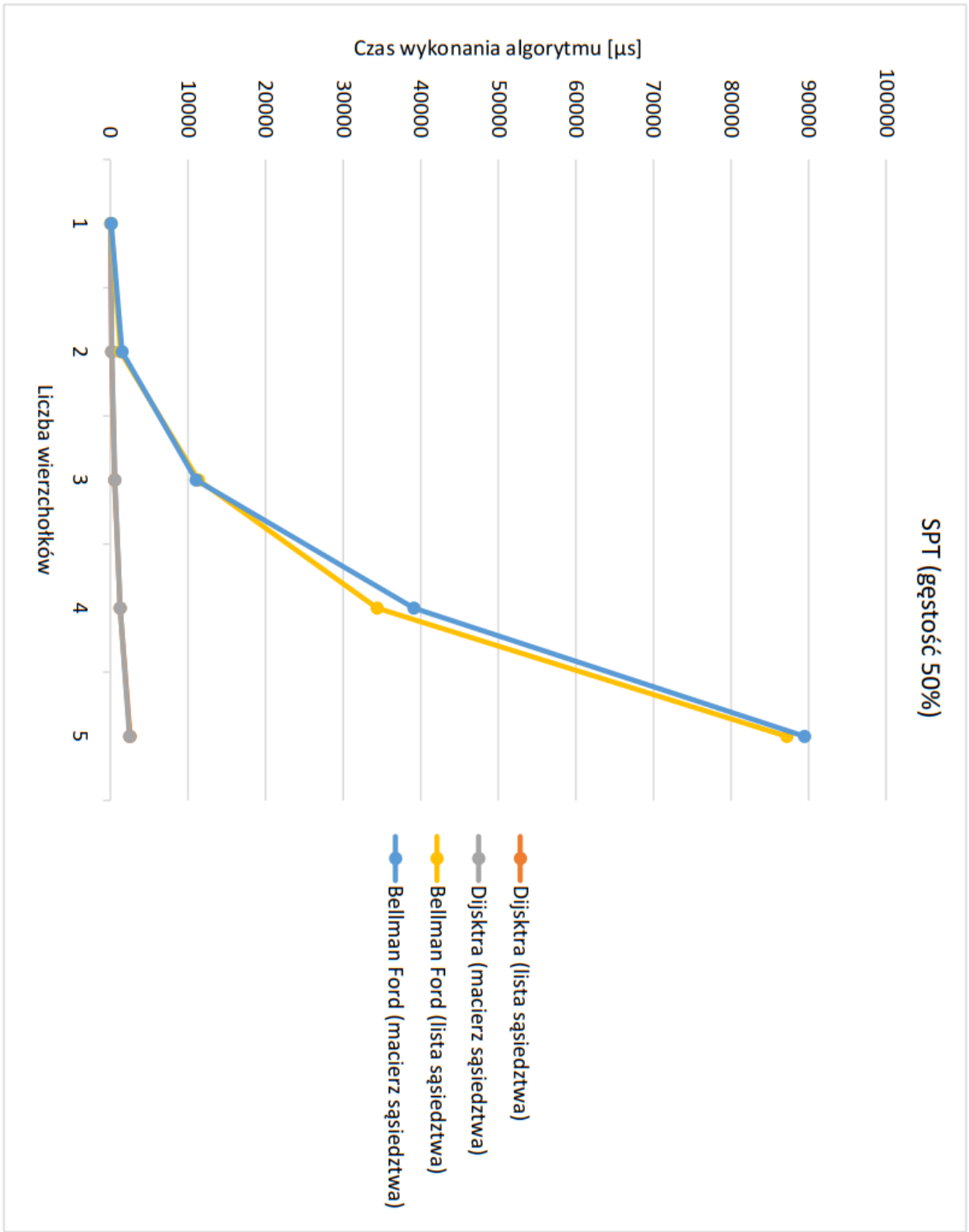
3.3 Wykresy typu I

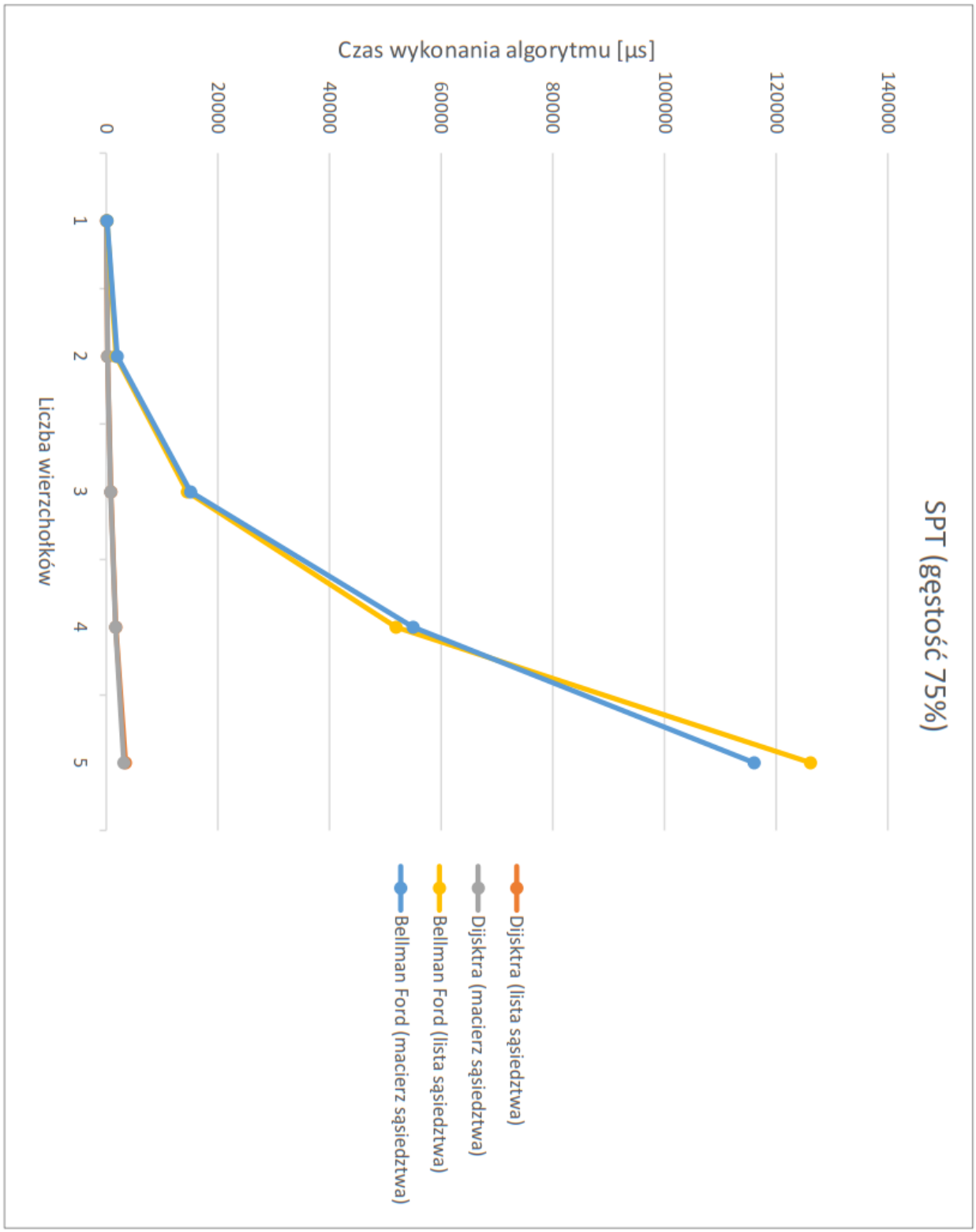


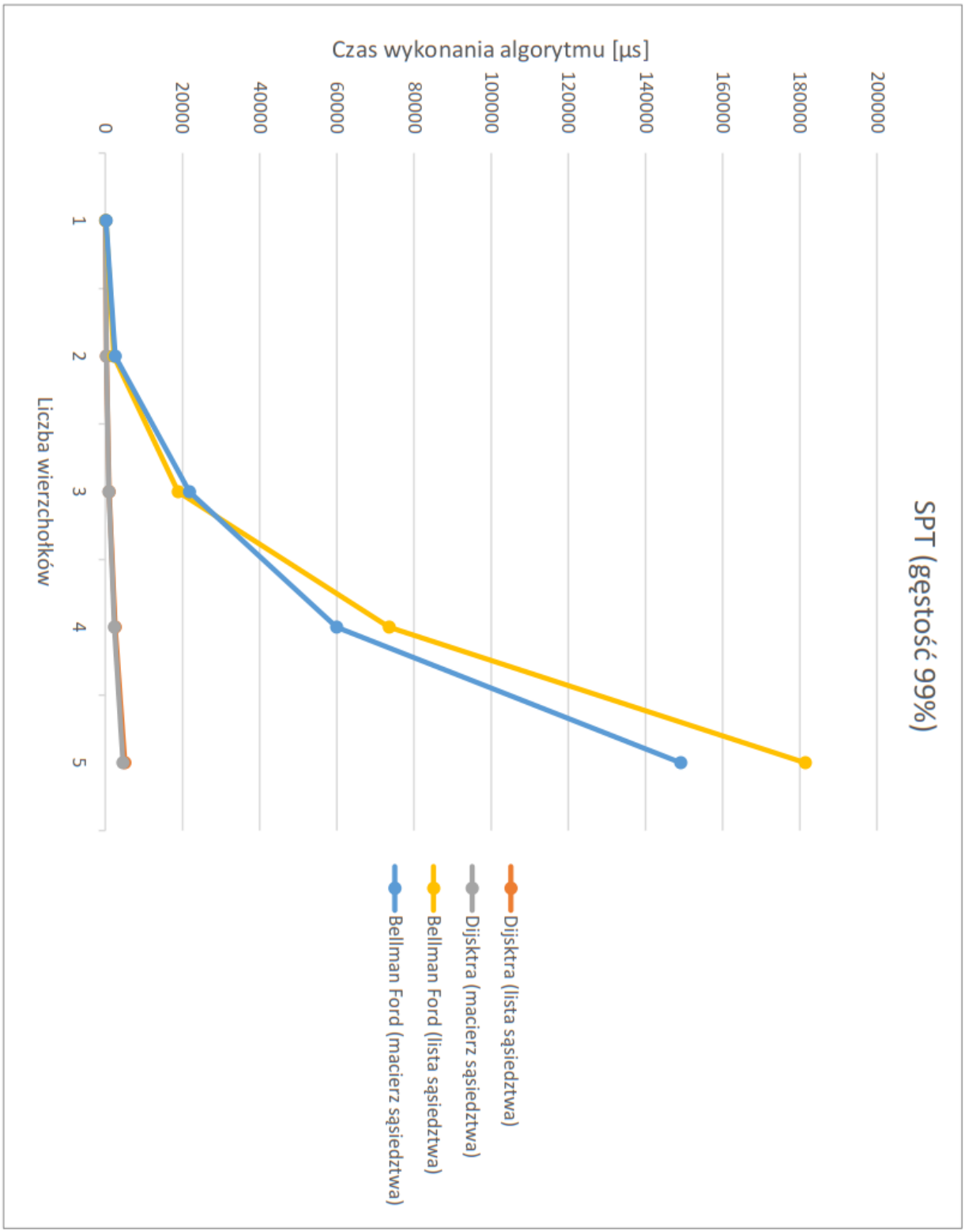

SPT (lista sąsiedztwa)
Czas wykonania algorytmu [µs]
200000
180000
160000
140000
120000
100000
80000
60000
40000
20000
0
10
25
50
75
100
Liczba wierzchołków
Dijkstra 25%
Dijkstra 50%
Dijkstra 75%
Dijkstra 99%
Bellman Ford 25%
Bellman Ford 50%
Bellman Ford 75%
Bellman Ford 99%

3.4 Wykresy typu II




SPT (gęstość 50%)
Czas wykonania algorytmu [µs]
100000
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
1
2
3
4
5
Liczba wierzchołków
Dijsktra (lista sąsiedztwa)
Dijsktra (macierz sąsiedztwa)
Bellman Ford (lista sąsiedztwa)
Bellman Ford (macierz sąsiedztwa)


SPT (gęstość 75%)
Czas wykonania algorytmu [µs]
140000
120000
100000
80000
60000
40000
20000
0
1
2
3
4
5
Liczba wierzchołków
Dijsktra (lista sąsiedztwa)
Dijsktra (macierz sąsiedztwa)
Bellman Ford (lista sąsiedztwa)
Bellman Ford (macierz sąsiedztwa)


SPT (gęstość 99%)
Czas wykonania algorytmu [µs]
200000
180000
160000
140000
120000
100000
80000
60000
40000
20000
0
1
2
3
4
5
Liczba wierzchołków
Dijsktra (lista sąsiedztwa)
Dijsktra (macierz sąsiedztwa)
Bellman Ford (lista sąsiedztwa)
Bellman Ford (macierz sąsiedztwa)

Wnioski

4.1 Minimalne drzewo rozpinające

W przypadku problemu minimalnego drzewa rozpinającego, algorytm Kruskala okazał się być szybszym dla każdego zestawu parametrów grafu.
Sposób przechowywania grafu w pamięci komputera zauważalnie wpłynął na czas wykonania obu algorytmów.
Reprezentacja listowa jest mniej korzystna pod względem czasu, niż reprezentacja macierzowa (macierz sąsiedztwa). To zjawisko ma nieznaczny wpływ na czas wykonania algorytmu Kruskala. Nie można tego samego powiedzieć o algorytmie Prima, gdyż czas jego wykonania dla reprezentacji listowej, był średnio prawie dwukrotnie większy, niż dla reprezentacji macierzowej.

4.2 Problem najkrótszej ściezki

Z badań czasu wykonania algorytmów rozwiązujących problem najkrótszej ściezki wynika, że algorytm Dijkstry jest znacznie lepszym, szybszym narzędziem, niż algorytm Bellmana-Forda, gdy graf wejściowy nie posiada cykli ujemnych. W przypadku, gdy występują cykle ujemne, należy zastosować taki algorytm, który potrafi znaleźć rozwiązanie optymalne, a w kontekście zaimplementowanych w tym projekcie algorytmów, jest to tylko alg. Bellmana-Forda.
W przeciwieństwie do wniosków wynikających z badań algorytmów grafowych, sposób reprezentacji grafu nie miał tutaj wiekszęgo znaczenia dla czasu wykonania. Wyniki otrzymane z pomiarów czasu wykonania algorytmów, były podobne dla zadanego zestawu parametrów (liczba wierzchołków, gęstość), niezależnie od sposobu reprezentacji grafu w pamięci komputera.

4.3 Ogólne

Rezultaty badania czasu wykonania algorytmów grafowych wykazują spójność z oczekiwaniami, rozumianymi jako asymptotyczna złożoność obliczeniowa.

Literatura

- Wikipedia - Category:Graph algorithms, dostęp 16.05.2019
- algorytmika.wikidot.com - "Problem Find-Union", dostęp 21.05.2019
- www.cise.ufl.edu - "Union-Find Problem, lecture 222" , dostęp 17.05.2019
- University of Minnesota - George Karypis "lectures, Graph Algorithms" , dostęp 21.05.2019
- Zespół Inżynierii Oprogramowania i Inteligencji Obliczeniowej, prof. dr hab. inż. Magott Jan - "Struktury Danych i Złożoność Obliczeniowa", dostęp 30.05.2019