**Przemysław Rola, Juliusz Wasieleski**
**Informatyka, III rok, grupa 6**
**grudzień 2023**

# Algorytmy macierzowe – permutacja macierzy i jej wpływ na kompresję – sprawozdanie

## 1. Opis ćwiczenia

Naszym zadaniem było , po wybraniu naszego ulubionego języka, wygenerowanie losowych macierzy o rozmiarach $2^{3*k}$ gdzie k=2,3,4. Macierze te miały symulować siatki 3d. Następnie mieliśmy zaimplementować algorytmy permutacji macierzy: Minimum degree, Cuthill-McKee, Reversed Cuthill-McKee (fajny algorytm).

Potem musieliśmy przetestować działanie powyższych argorytmów sprawdzając czy permutacja rzeczywiście poprawiła kompresję.

## 2. Środowisko, biblioteki, założenia oraz użyte narzędzia

Ćwiczenie wykonaliśmy w języku Python przy użyciu Jupyer Notebooka. Do obliczeń, przechowywania danych użyliśmy bibliotek *numpy, pandas, scipy.*

Do rysowania wykresów użyliśmy biblioteki *matplotlib.*

Wszystkie obliczenia prowadziliśmy na komputerze Lenovo Y50-70 z systemem Windows 10 Pro w wersji 10.0.19045, procesor Intel Core i7-4720HQ 2.60GHz, 2601 MHz, rdzenie: 4, procesory logiczne: 8.

# 3. Implementacje

## 3.1 Minimal degree permutation

### 3.1.1 Pseudokod

Minimal degree permutation(matrix)*:*

*G(A) = {V, E}*

*Dla każdego* $v_i \in V$, $t_i$ *=#* $adj_{Go}(v_i)$

***Dla*** *k=1 do n-1:*

Wybierz węzeł p∈V o minimalnej liczbie sąsiadów

Dla każdego wierchołka $v_i \in adj_{Gk-1}(p)$:

$Adj_{Gk-1}(v_i) = \{adj_{Gk-1}(v_i)$ suma $adj_{Gk-1}(p)\} \backslash \{v_i, p\}$

$T_i$ =# $adj_{Gk}(v_i)$

$V_k = V_k \backslash \{p\}$

Kod 1 Pseudokod funkcji minimal degree permutation

### 3.1.2 Istotne fragmenty implementacji

```
def minimum_degree_permutation(matrix):
    n,m = matrix.shape
    ADJ = {i:set() for i in range(n)}
    for i in range(n):
        for j in range(m):
            if matrix[i][j] != 0 and i != j:
                ADJ[i].add(j)
    permutation = []
    for i in range(n):
        deg_min = m+1
        for v, adj in ADJ.items():
            if len(adj) < deg_min:
                v_min = v
                deg_min = len(adj)
        for v in ADJ:
            ADJ[v] = ADJ[v].difference([v_min])
        for u in ADJ[v_min]:
            ADJ[u] = (ADJ[u].union(ADJ[v_min].difference([u])))
        ADJ.pop(v_min)
        permutation.append(v_min)
    return permutation
```

Kod 2 implementacja minimum degree permutation

Nasza implementacja w dość dużym stopniu pokrywa się z pseudokodem. Warto zwrócić uwagę że krawędzie wychodzące z wierzchołków reprezentujemy jako listę zbiorów (zbiory to jedna z wbudowanych w pythona struktur), dzięki czemu możemy w łatwy sposób

obliczać różnice zbiorów (przy użyciu metody difference) albo unię zbiorów (przy użyciu metody union).

## 3.2 Cuthill & McKee

### 3.2.1Pseudokod

Cuthill & Mckee permutation(matrix)*:* # BFS(v)

*wstaw v do kolejki*

*dopóki kolejka nie jest pusta:*

*wyciągnij v z kolejki*

*wypisz v*

*for u sąsiedzi v (posortowani rosnąco według stopnia)*

*if u nie był odwiedzony*

*wstaw u do kolejki*

Kod 3 Pseudokod permutacji Cuthill & McKee

### 3.2.2 Istotne fragmenty implementacji

```
def cuthill_mckee(matrix):
    def BFS():
        while Q:
            v = Q.popleft()
            if Visited[v]:
                continue
            permutation.append(v)
            Visited[v] = True
            for u in sorted(nx.neighbors(G, v), key = lambda x : G.degree(x)):
                if not Visited[u]:
                    Q.append(u)
    n = len(matrix)

    G = nx.Graph()
    G.add_nodes_from(range(n))

    for i in range(n):
        for j in range(n):
            if matrix[i][j] != 0 and i != j:
                G.add_edge(i, j)

    permutation = []
    Visited = [False for i in range(n)]
    sorted_nodes = sorted([x for x in G.degree()], key = lambda x : x[1])
    sorted_nodes = list(map(lambda x : x[0], sorted_nodes))
    Q = deque()
    for s in sorted_nodes:
        if not Visited[s]:
            Q.append(s)
            BFS()

    return permutation
```

Kod 4 Nasza implementacja permutacji Cuthill & McKee

Już w pseudokodzie widać, że ten algorytm permutacji pokrywa się z BFSem. Jedynie dodane jest wypisywanie wierzchołków a nie tylko przechodzenie po nich. Jednakże z kwestii czysto implementacyjnych można przyjrzeć się reprezentacji grafu którą wybraliśmy. Zdecydowaliśmy się skorzystać z biblioteki NetworkX (w kodzie jako nx) w której jest klasa reprezentująca Grafy. Ciekawa może też być linijka w której sortujemy wierzchołki grafu po ich stopniu, działa to tak, że metoda degree na grafie zwraca DegreeViev, które jest reprezentacją stopni w ramach tej biblioteki, ale po którym można iterować. Dzięki temu oraz mechanizmowi list comprehension tworzymy listę krotek w których pierwszym elementem jest numer wierzchołka a drugim jego stopnień. Następnie do funkcji sortującej przekazujemy informację, że ma sortować po drugim elemencie każdej z krotek (czyli o indeksie 1) ponieważ w nim jest stopień wierzchołka. Następnie, już w kolejnej linijce, z każdego elementu tej posortowanej listy wyciągamy pierwszy element korzystając do tego z funkcji map, która na każdy z elementów mapuje zadaną operację (w naszym wypadku wyciągnięcie pierwszego elementu)

## 3.3 Fajny algorytm – Odwrócony Cuthill & McKee

### 3.3.1 Pseudokod

To co zwraca zwykły Cuthill & McKee wypisz w odwrotnej kolejności

Kod 5 skomplikowany pseudokod odwróconego algorytmu permutacji Cuthill & McKee

### 3.3.2 Istotne fragmenty implementacji

```
def reversed_cuthill_mckee(matrix):
    return cuthill_mckee(matrix)[::-1]
```

Kod 6 Nasza implementacja odwróconej permutacji Cuthill & McKee

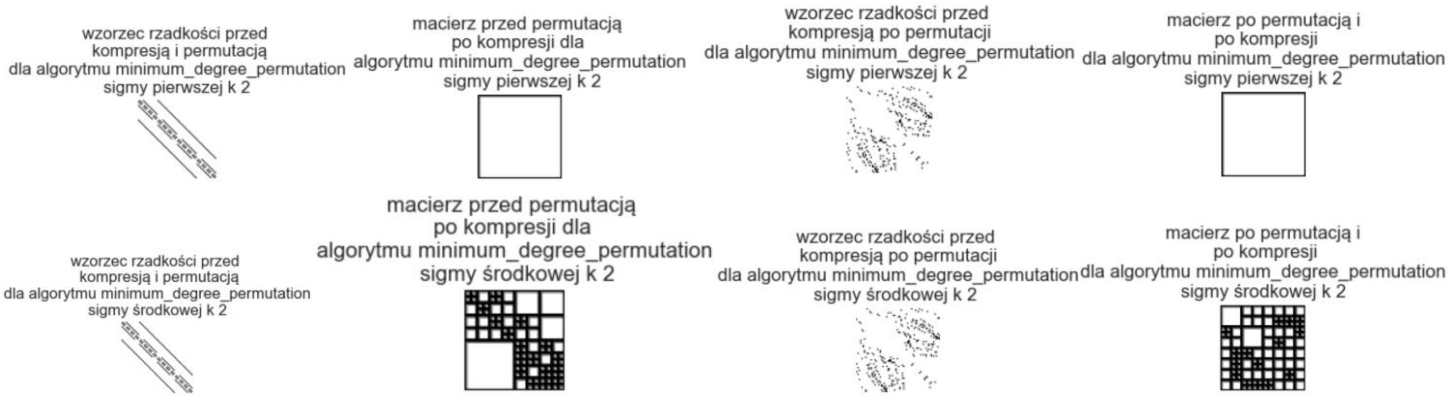
Gdybyśmy pisali rozprawę matematyczną moglibyśmy powiedzieć, że nasza implementacja jest trywialna i jej zrozumienie pozostawiamy czytelnikowi (i w przeciwieństwie do wspominanych rozpraw byłoby to prawdą).

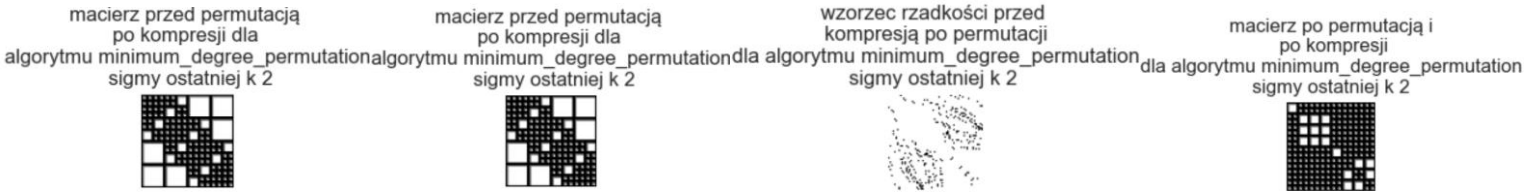
# 4 Pomiary działania permutacji i ich analiza

## 4.1 Pomiary

### 4.1.1 Pomiary dla macierzy rozmiaru $2^6$

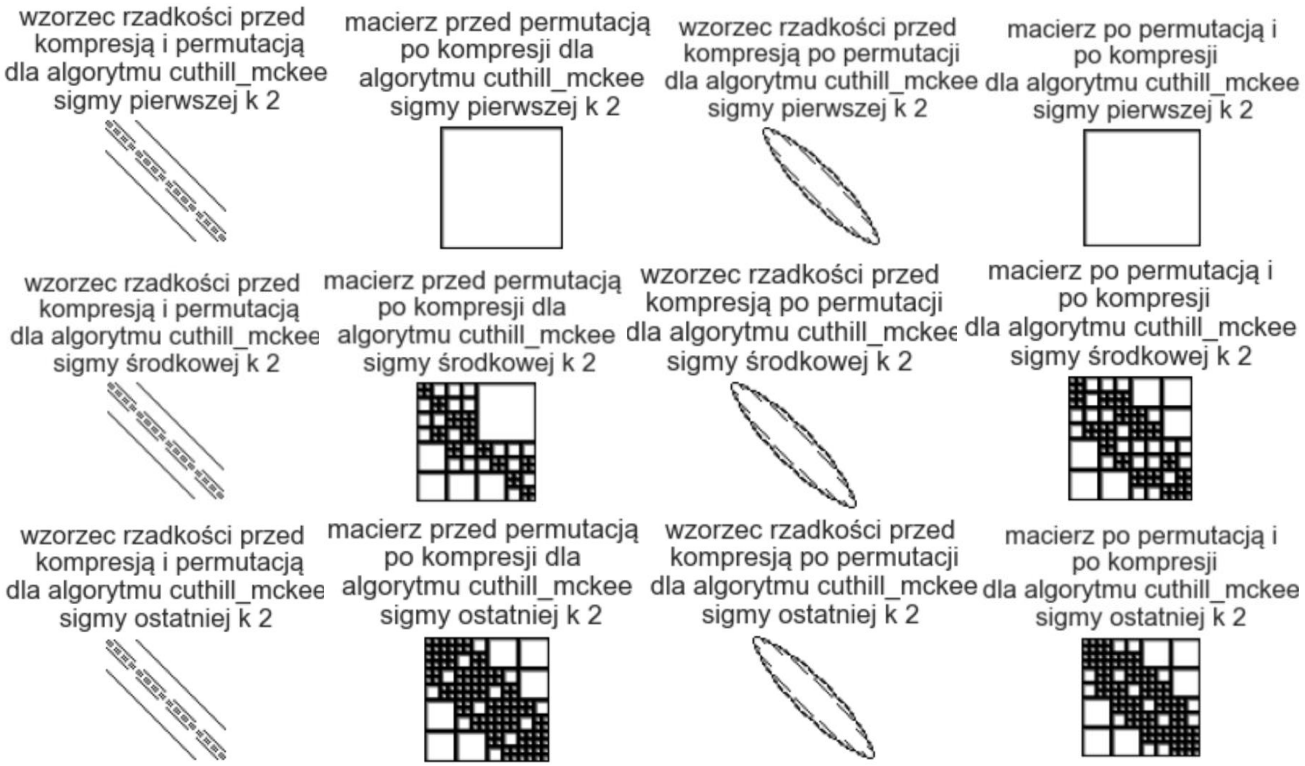
*Pomiary algorytmu minimum degree permutation*





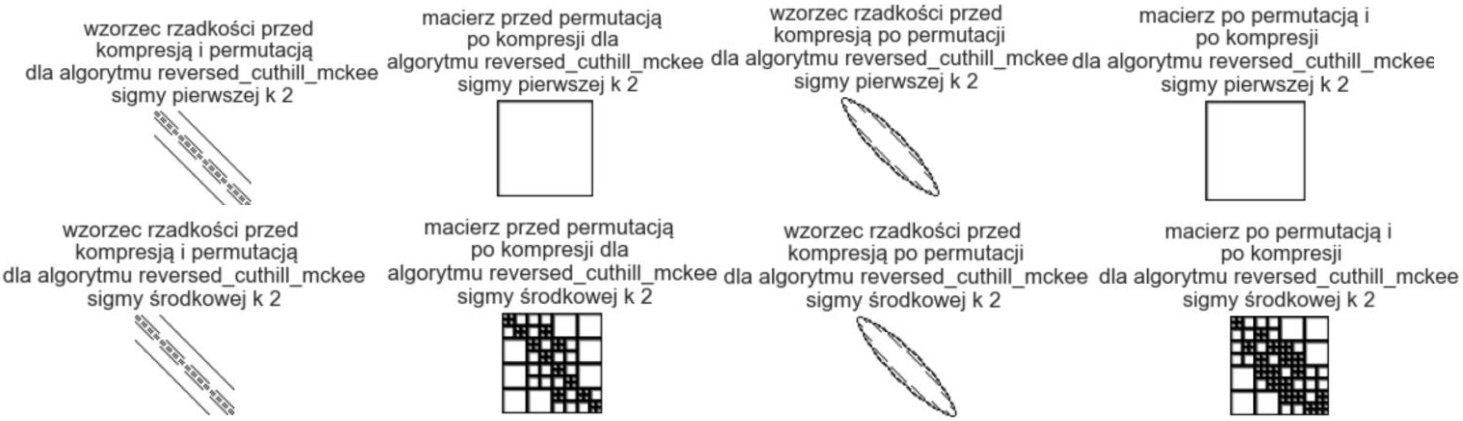

Rys. 1 Zebrane pomiary dla macierzy rozmiarów $2^6$ dla algorytmu permutacji minimum degree permutation w rozróżnieniu na różne sigmy

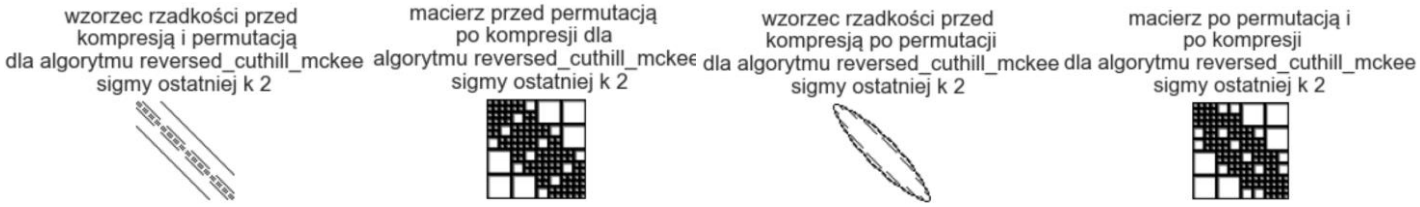
### *Pomiary Cuthill & McKee*




Rys. 2 Zebrane pomiary dla macierzy rozmiarów $2^6$ dla algorytmu permutacji Cuthill & McKee w rozróżnieniu na różne sigmy

### *Pomiary odwróconego Cuthill & McKee*

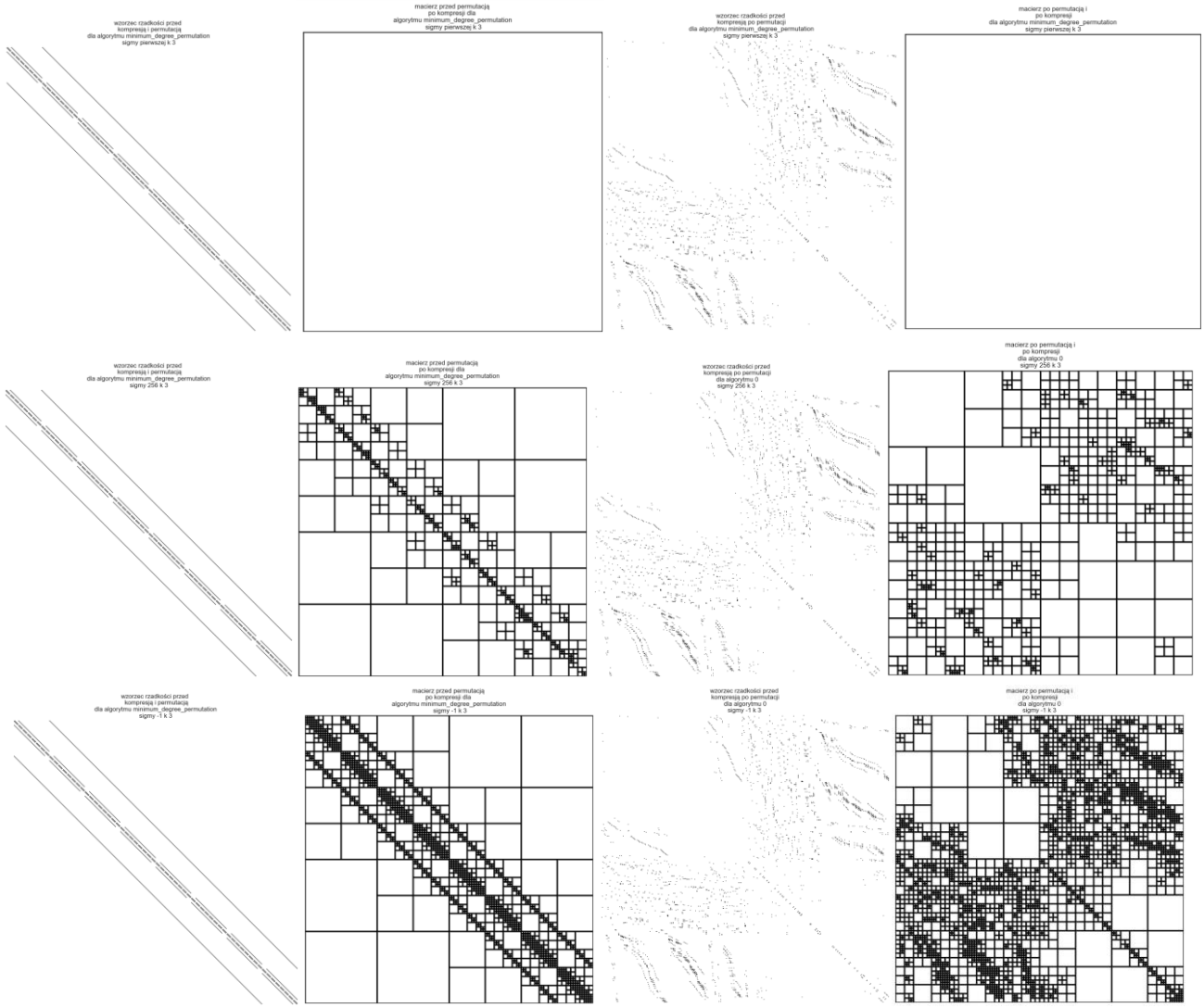





Rys. 3 Zebrane pomiary dla macierzy rozmiarów $2^6$ dla odwróconego algorytmu permutacji Cuthill & McKee w rozróżnieniu na różne sigmy

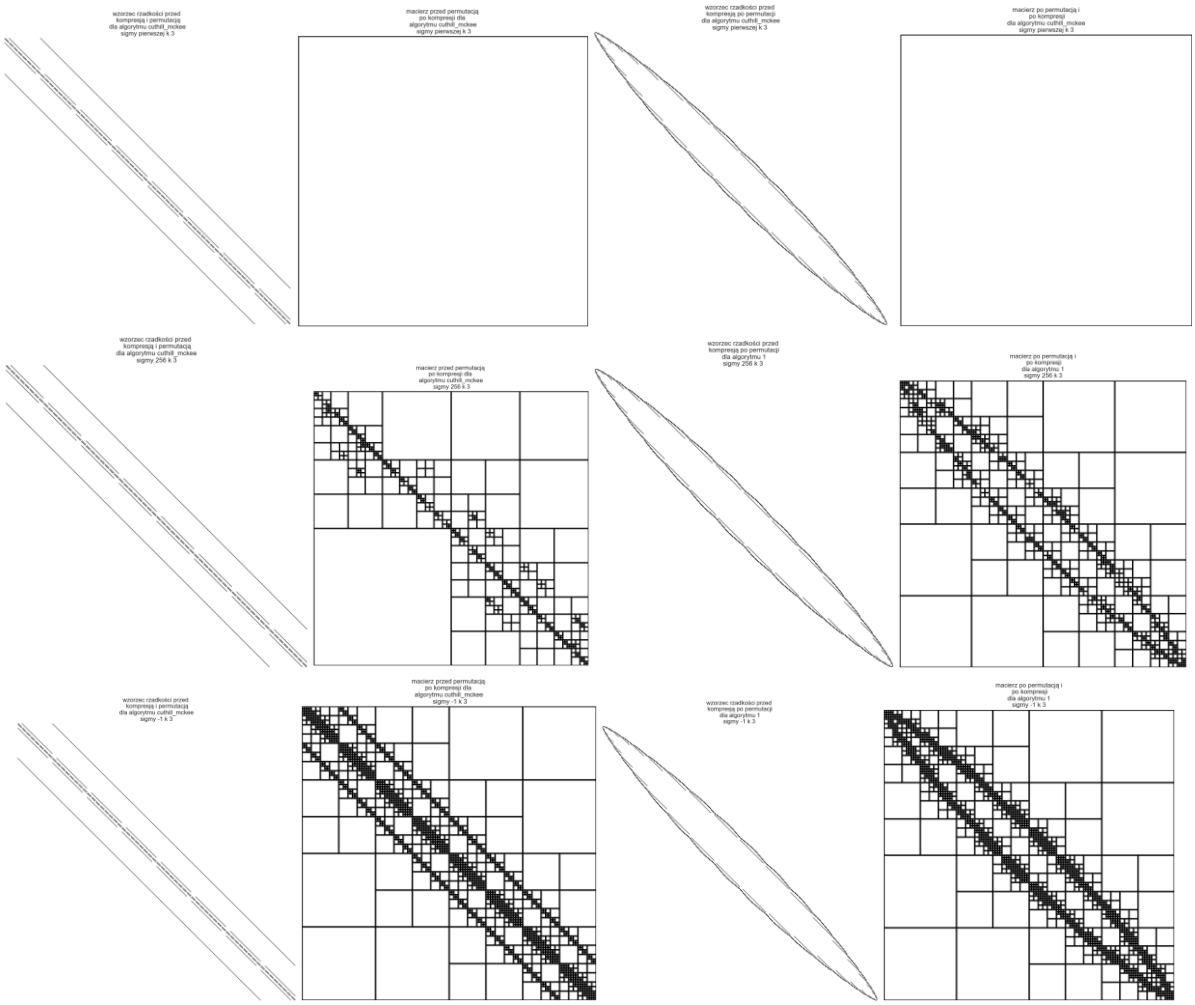
### 4.1.2 Pomiary dla macierzy rozmiaru $2^9$

*Pomiary minimal degree permutation*



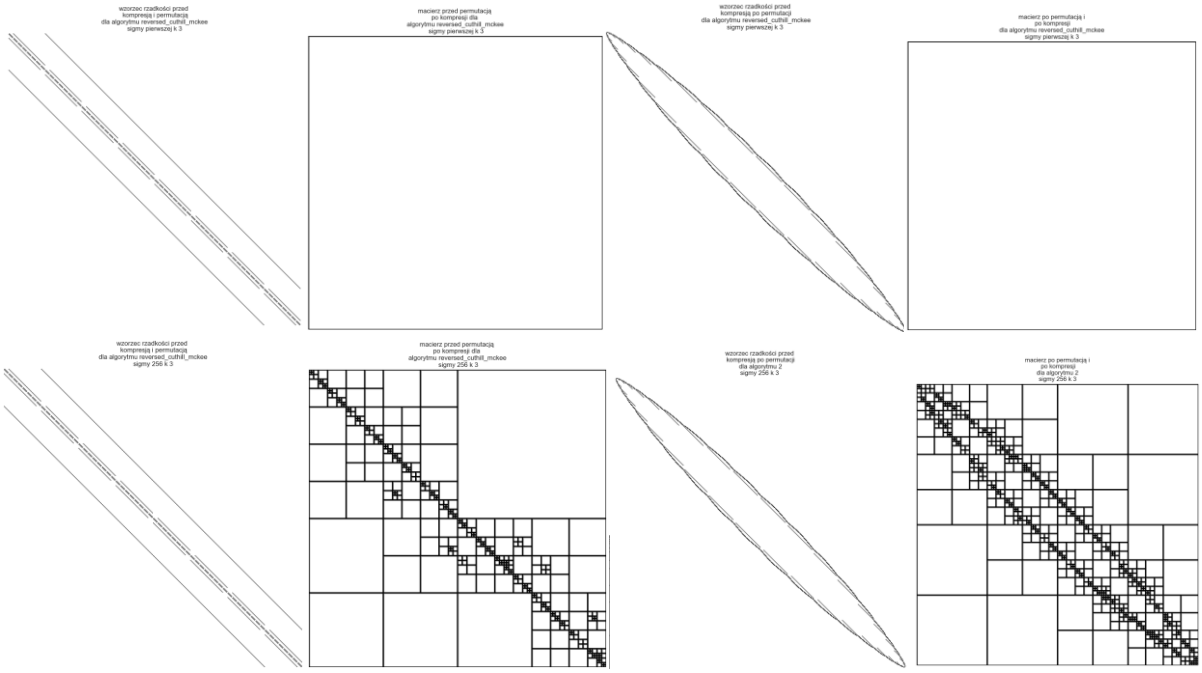
Rys. 4 Zebrane pomiary dla macierzy rozmiarów $2^9$ dla algorytmu permutacji minimum degree permutation w rozróżnieniu na różne sigmy

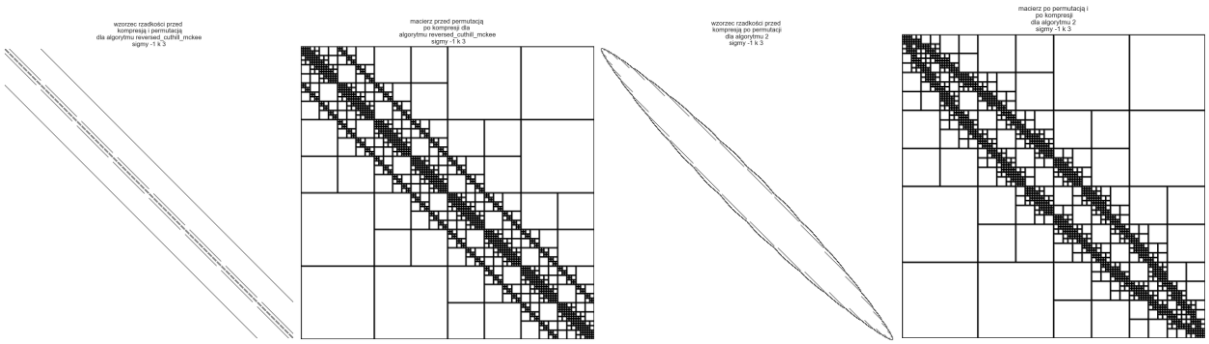
## *Pomiary Cuthill & McKee*



Rys. 5 Zebrane pomiary dla macierzy rozmiarów $2^9$ dla algorytmu permutacji Cuthill & McKee w rozróżnieniu na różne sigmy

## *Pomiary odwróconego Cuthill & McKee*

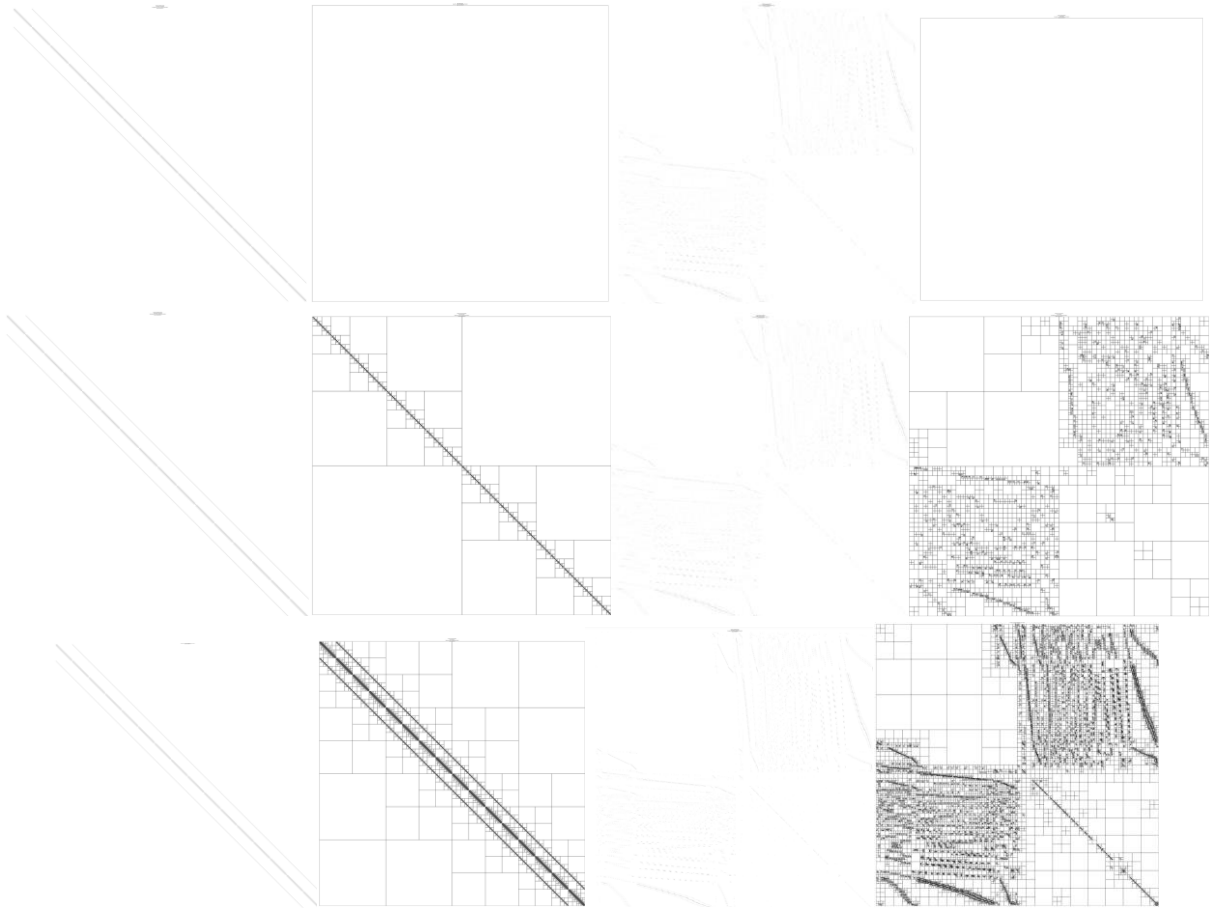




Rys. 6 Zebrane pomiary dla macierzy rozmiarów $2^9$ dla odwróconego algorytmu permutacji Cuthill & McKee w rozróżnieniu na różne sigmy

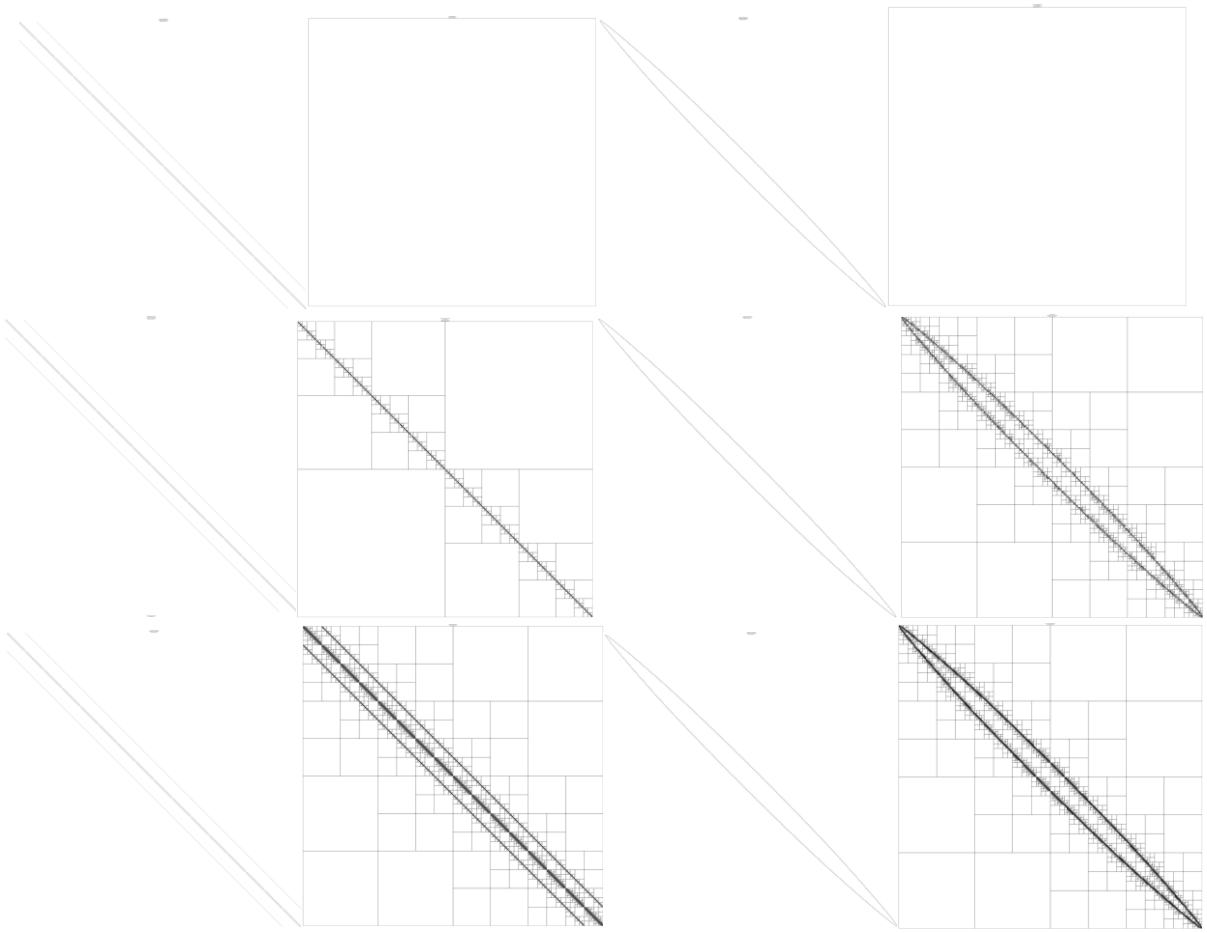
### 4.1.3 Pomiary dla macierzy rozmiaru $2^{12}$

*Pomiary minimal degree permutation*



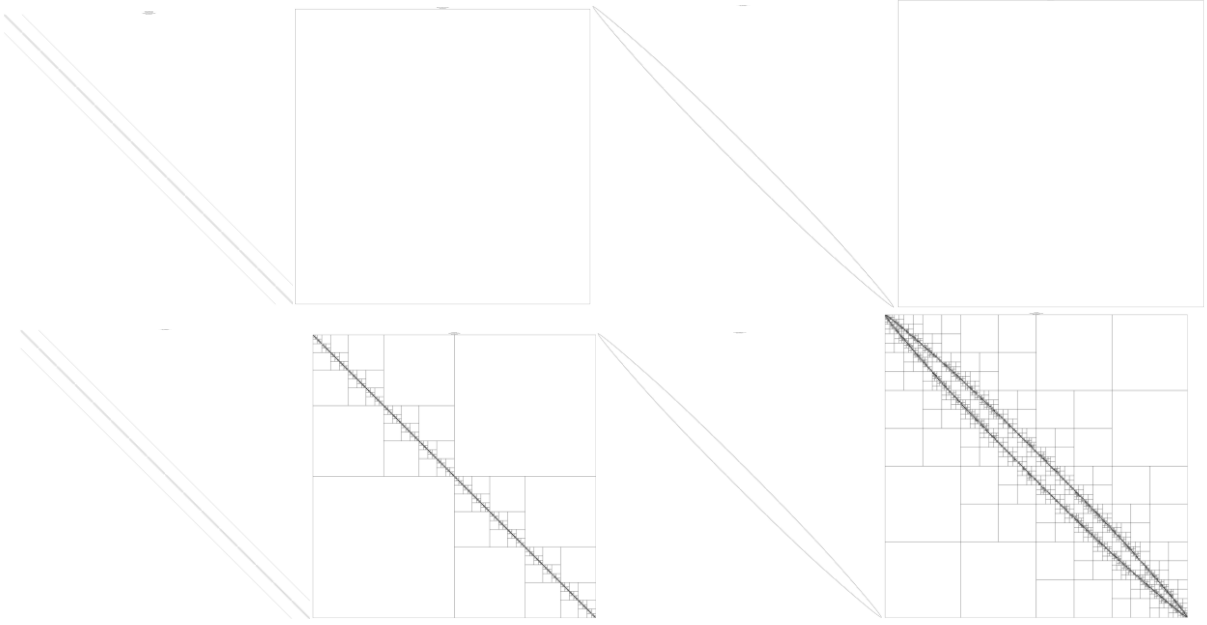
Rys. 7 Zebrane pomiary dla macierzy rozmiarów $2^{12}$ dla algorytmu permutacji minimum degree permutation w rozróżnieniu na różne sigmy

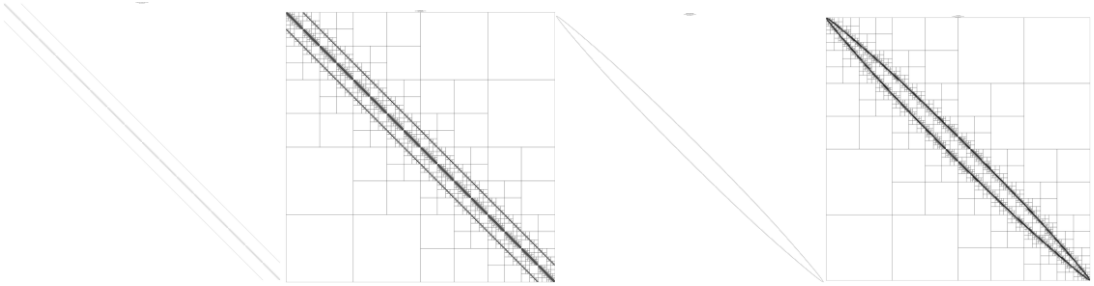
*Pomiary Cuthill & McKee*



Rys. 8 Zebrane pomiary dla macierzy rozmiarów $2^{12}$ dla algorytmu permutacji Cuthill & McKee w rozróżnieniu na różne sigmy

*Pomiary odwróconego Cuthill & McKee*





Rys. 9 Zebrane pomiary dla macierzy rozmiarów $2^{12}$ dla odwróconego algorytmu permutacji Cuthill & McKee w rozróżnieniu na różne sigmy

### 4.1.3 Pomiary stopnia kompresji macierzy

| k | sigma | Algorytm | Kompresja bez perutacji | Kompresja z permutacją |
|---|---|---|---|---|
| 2 | pierwsza | Minimum degree | 0.015747 | 0.015747 |
| 2 | środkowa | Minimum degree | 0.128784 | 0.150146 |
| 2 | ostatnia | Minimum degree | 0.172119 | 0.241821 |
| 2 | pierwsza | Cuthill & McKee | 0.015747 | 0.015747 |
| 2 | środkowa | Cuthill & McKee | 0.128784 | 0.144287 |
| 2 | ostatnia | Cuthill & McKee | 0.172119 | 0.167480 |
| 2 | pierwsza | Odwrócony Cuthill & McKee | 0.015747 | 0.015747 |
| 2 | środkowa | Odwrócony Cuthill & McKee | 0.126465 | 0.141968 |
| 2 | ostatnia | Odwrócony Cuthill & McKee | 0.172119 | 0.167480 |
| 3 | pierwsza | Minimum degree | 0.001955 | 0.001955 |
| 3 | środkowa | Minimum degree | 0.016668 | 0.039307 |
| 3 | ostatnia | Minimum degree | 0.045464 | 0.094868 |
| 3 | pierwsza | Cuthill & McKee | 0.001955 | 0.001955 |
| 3 | środkowa | Cuthill & McKee | 0.022736 | 0.034847 |
| 3 | ostatnia | Cuthill & McKee | 0.045464 | 0.045097 |
| 3 | pierwsza | Odwrócony Cuthill & McKee | 0.001955 | 0.001955 |
| 3 | środkowa | Odwrócony Cuthill & McKee | 0.018101 | 0.034460 |
| 3 | ostatnia | Odwrócony Cuthill & McKee | 0.045464 | 0.045097 |
| 4 | pierwsza | Minimum degree | 0.000244 | 0.000244 |
| 4 | środkowa | Minimum degree | 0.002662 | 0.011795 |
| 4 | ostatnia | Minimum degree | 0.008846 | 0.027671 |
| 4 | pierwsza | Cuthill & McKee | 0.000244 | 0.000244 |
| 4 | środkowa | Cuthill & McKee | 0.002673 | 0.006460 |
| 4 | ostatnia | Cuthill & McKee | 0.008846 | 0.008956 |
| 4 | pierwsza | Odwrócony Cuthill & McKee | 0.000244 | 0.000244 |
| 4 | środkowa | Odwrócony Cuthill & McKee | 0.002679 | 0.006534 |
| 4 | ostatnia | Odwrócony Cuthill & McKee | 0.008846 | 0.008956 |

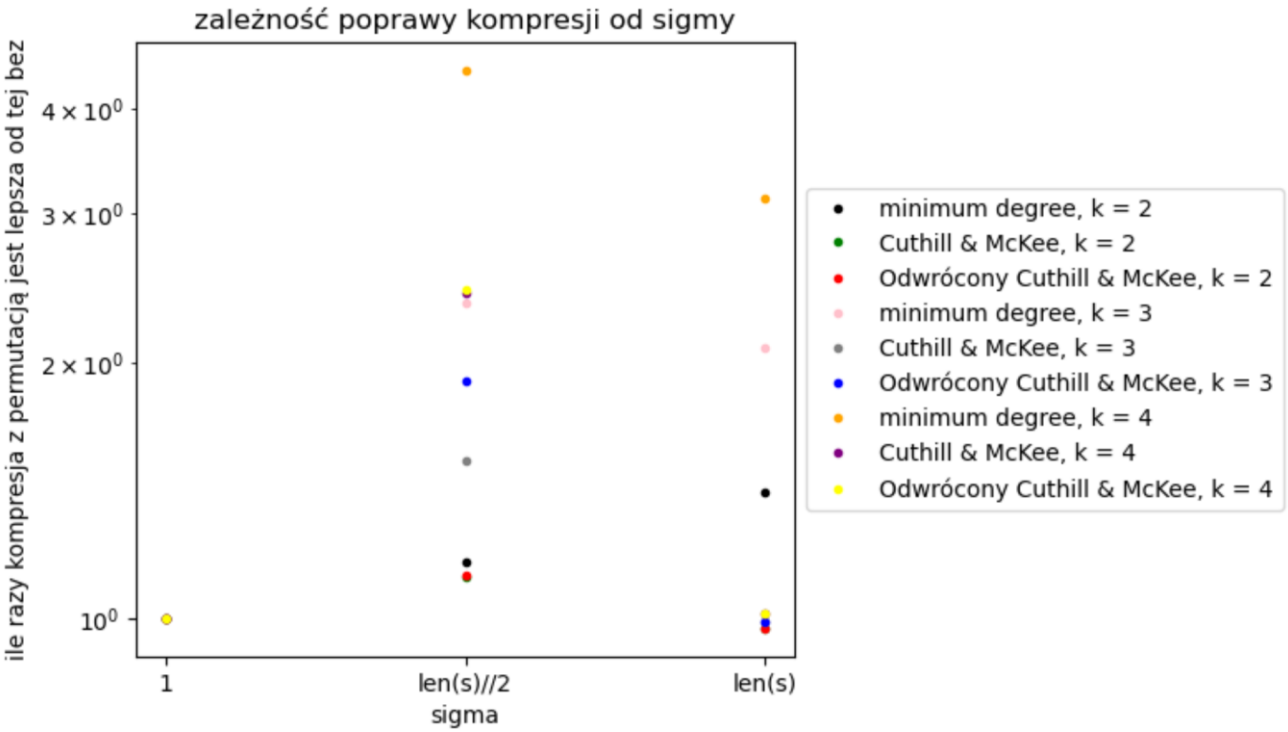
Tab 1 Poziom kompresji oznaczony kolorami szary – bez zmian, zielony – permutacja poprawia, czerwony – permutacja pogarsza

## 4.2 Analiza pomiarów

Zacznijmy od rozważenia rysunków 1-9 na każdym z nich analizując każdy wiersz osobno. Widać od razu, że permutacja jakoś działa, bo macierze wzorce gęstości macierzy spermutowanych wyglądają inaczej niż oryginalnych siatek oraz różnią się między algorytmami minimal degree oraz Cuthill & McKee. Dla tej pierwszej z permutacji wartości rozpraszają się po całej macierzy przypominając graficznie opiłki żelaza wysypane na kwadratową kartkę do której prawego górnego i lewego dolnego wierzchołka przyłożone zostały jednoimienne bieguny magnesu. W przeciwieństwie do niej, graficzne przedstawienie permutacji Cuthill& McKee oraz jej odwróconej wersji przypomina wspomniane opiłki żelaza gdybyśmy do lewego górnego i prawego dolnego wierzchołka macierzy przyłożyli różnoimienne magnesy.

Mimo, że jest to pewnego rodzaju powtórzenie obserwacji z poprzedniego sprawozdania to powyższy wpływ jest widoczny na rysunkach, więc warto zwrócić na nie uwagę. Dla pierwszej sigmy kompresja nie zachodzi, ani dla oryginalnej macierzy, ani tej spermutowanej. Porównując środkową sigmę z tą ostatnią widzimy, że bez permutacji dla środkowej sigmy mamy mnieszy stopień kompresji, a dla już spermutowanej macierzy algorytmem minimum degree, przy ostatniej sigmie uwydatnia się przekątna prawej dolnej podmacierzy.

Patrząc na tabelę nr 1 oraz rysunek nr 10 można dostrzec kilka faktów, po pierwsze dla sigmy nr 1 permutacja nic nie zmienia, bo kompresja prawie nie występuje więc permutacja nie ma możliwości wykazania się. Co więcej dla sigmy środkowej i końcowej, algorytm minimum degree zawsze poprawiał poziom kompresji. W przeciwieństwie do niego, algorytmy Cuthil & McKee oraz jego odwrócona wersja, dla ostatniej sigmy zachowywały się różnie w zależności od rozmiaru tablicy. Dla $2^6$, permutacja zdecydowanie pogarszała kompresję, dla $2^9$ był to dość porównywalny poziom z delikatną przewagą macierzy niespermutowanej, a dla $2^{12}$ permutacja tymi algorytmami rzeczywiście poprawiała poziom kompresji. Opisane pwoyżej zależności można zauważyć na poniższym wykresie nr 1.



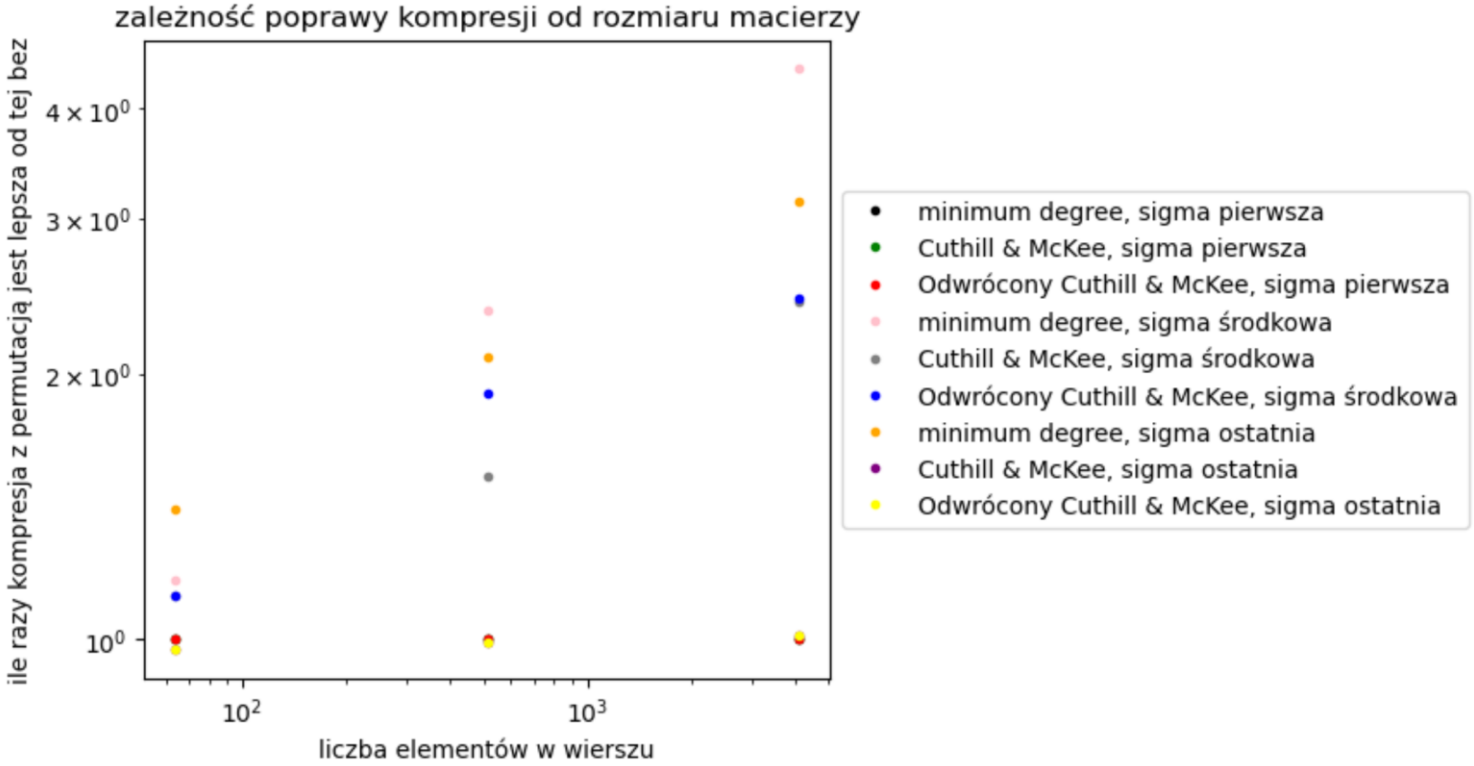

Wykres 1 zależność poprawy kompresji od sigmy

Potwierdzają się na wykresie nasze obserwacje z tabeli, mianowicie, dla sigmy 1 widać brak czegokolwiek, bo kompresja nie ma miejsca, z kolei najbardziej zjawiskowe poprawy obserwujemy dla środkowej sigmy, a dla ostatniej to zależy od algorytmu, bo minimum degree sobie nadal dobrze radzi, a oba Cuthill & McKee niezbyt.

Zauważmy, że dla ostatniej sigmy Cuthil & McKee oraz jego odwrócona wersja dawały dokładnie taki sam poziom kompresji, ale dlaśrodkowej sigmy to zwykły Cuthill & McKee dawał lepsze wyniki, także samo odwrócenie kolejności też może mieć już znaczenie.

Dla środkowej sigmy najlepiej widać tendencję, że największą poprawę poziomu kompresji mają po kolei algorytmy: Minimum degree, Cuthill & McKee, odwrócony Cuthill & McKee.

Kolejną obserwowaną przez nas tendencją jest fakt, że wraz ze wzrostem rozmiaru macierzy wzrastają benefity permutacji, bo dla przykładu dla permutacji minimal degree i ostatniej sigmy, dla macierzy rozmiaru $2^6$ mamy poprawę 1.4 raza, dla $2^9$ 2.1 raza a dla $2^{12}$ 3.13 raza. Te obserwacje można dostrzec na wykresie nr 2 znajdującym się poniżej.




Wykres 2 zależność poprawy kompresji od rozmiaru macierzy

Niestety takie zjawiskowe zależności mamy jedynie dla środkowej sigmy, dla pierwszej (bez zaskoczenia) nie mamy poprawy, a dla ostatniej widzimy, że poprawa jest wolniejsza, oraz, że dla mniejszych macierzy poprawa jest w rzeczywistości pogorszeniem.

Pięknie widać na tym wykresie, że najlepiej radzi sobie z kompresją algorytm minimal degree niezależnie od sigmy a potem nadal znośnie Cuthill & McKee ale tylko dla środkowej sigmy

# 5 Wnioski

- Dla pierwszej sigmy permutacja nie ma sensu, bo kompresja i tak nic nie zrobi

- Algorytm minimal degree najlepiej permutuje macierze tzn można po jego użyciu uzyskać najlepszy współczynnik kompresji
- Algorytmy Cuthill & McKee działają dobrze tylko dla środkowej sigmy, dla ostatniejnie dają zbyt dużej poprawy lub wręcz pogarszają
- Wraz ze wzrostem rozmiaru macierzy permutacja ma coraz większy sens, ponieważ rośnie krotność poprawy kompresji