Algorytmy macierzowe

Laboratorium 4

Sprawozdanie

Łukasz Stępień, Szymon Urbański

1. Temat zadania

Laboratorium polegało na zaimplementowaniu algorytmów permutacji macierzy:

- minimum degree
- Cuthill-McKee
- reversed Cuthill-McKee

Należało także narysować:

- wzorzec rzadkości macierzy rzadkiej przed kompresją i permutacją
- macierz rzadką przed permutacją ale po kompresji
- wzorzec rzadkości macierzy rzadkiej po permutacji ale przed kompresją
- macierz rzadką po permutacji i po kompresji

2. Rozwiązanie

Zaimplementowano następujące algorytmy permutacji wierszy macierzy:

- **minimum degree**

a) Pseudokod:

```
minimum_degree(G):
    for i=1 to n:
        wybierz węzeł p o najmniejszej liczbie sąsiadów
        połącz wszystkie węzły które sąsiadują z p ze sobą
        usuń węzeł p
        zaktualizuj ilość sąsiadów
    zwróć kolejność usuwania wierzchołków
```

b) Kod:

```python
def minimumDegree(self):
    degrees = [sum(row) for row in self.matrix]
    notVisited = [i for i in range(self.n)]
    matrixCopy = copy.deepcopy(self.matrix)
    res = []

    for k in range(self.n):
        notVisited.sort(key=lambda x: (degrees[x], x))
        p = notVisited.pop(0)
        res.append(p)
        for i in range(self.n):
            if matrixCopy[p][i] != 0:
                matrixCopy[p][i] = 0
                matrixCopy[i][p] = 0
                degrees[i] -= 1
                for j in range(i + 1, self.n):
                    if matrixCopy[p][j] != 0 and matrixCopy[i][j] == 0:
                        matrixCopy[i][j] = 1
                        matrixCopy[j][i] = 1
                        degrees[i] += 1
                        degrees[j] += 1
    return res
```

Kod 2.1 Implementacja algorytmu minimum degree

- **Cuthill-McKee**

a) Pseudokod:

```
CM(G):
    znajdź węzeł v z najmniejszą liczbą sąsiadów i wstaw go do kolejki
    while kolejka zawiera elementy:
        wyciągnij v z kolejki
        for u sąsiedzi v (posortowani według liczby sąsiadów):
            if u nie był odwiedzony:
                wstaw u do kolejki
    zwróć węzły w kolejności wyciągania ich z kolejki
```

b) Kod:

```
def CuthillMckee(self):
    degrees = [sum(row) for row in self.matrix]

    Q = Queue()
    R = []

    notVisited = [(i, degrees[i]) for i in range(len(degrees))]

    while len(notVisited):

        minNodeIndex = min(range(len(notVisited)),
                           key=lambda i: notVisited[i][1])

        Q.append(notVisited[minNodeIndex][0])

        notVisited.pop(findIndex(notVisited, notVisited[Q[0]][0]))

        while Q:
            toSort = []
            v = Q.popleft()
            for i in range(self.n):
                if (i != v and self.matrix[v][i] != 0 and findIndex(notVisited, i) != -1):
                    toSort.append(i)
                    notVisited.pop(findIndex(notVisited, i))

            toSort.sort(key=lambda x: degrees[x])
            Q.extend(toSort)
            R.append(v)

    return R
```

Kod 2.2 Implementacja algorytmu Cuthill-McKee

```
def findIndex(a, x):
    return next((i for i, item in enumerate(a) if item[0] == x), -1)
```

Kod 2.3 Znalezienie indeksu (wykorzystane w kodzie 2.2)

```
def CM(self):
    cuthill = self.CuthillMckee()
    return cuthill
```

Kod 2.4 Wywołanie funkcji z kodu 2.2

- **reversed Cuthill-McKee**

a) Pseudokod:

```
RCM(G):
      znajdź węzeł v z najmniejszą liczbą sąsiadów i wstaw go do kolejki
      while kolejka zawiera elementy:
            wyciągnij v z kolejki
            for u sąsiedzi v (posortowani według liczby sąsiadów):
                  if u nie był odwiedzony:
                        wstaw u do kolejki
      zwróć węzły w odwrotnej kolejności niż wyciąganie ich z kolejki
```

b) Kod:

```python
def RCM(self):
    cuthill = self.CuthillMckee()
    return cuthill[::-1]
```

Kod 2.5 Wywołanie funkcji z wykorzystaniem kodu 2.2

Zaimplementowano także generator macierzy o strukturze opisującej topologię trójwymiarowej siatki zbudowanej z elementów sześciennych.

```python
def generate_3d_grid_matrix(k):
    size = 2**(3*k)
    matrix = [[0] * size for _ in range(size)]
    eps = np.finfo(float).eps

    for i in range(size):
        x, y, z = np.unravel_index(i, (2**k, 2**k, 2**k))
        neighbors = [
            np.ravel_multi_index((x + dx, y + dy, z + dz),
                                 (2**k, 2**k, 2**k), mode='wrap')
            for dx in [-1, 0, 1]
            for dy in [-1, 0, 1]
            for dz in [-1, 0, 1]
            if (dx != 0 or dy != 0 or dz != 0) and 0 <= x + dx < 2**k and 0 <= y + dy < 2**k and 0 <= z + dz < 2**k
        ]
        for neighbor in neighbors:
            matrix[i][neighbor] = random.uniform(10 ** -8 + eps, 1.0 - eps)

    return np.array(matrix)
```

Kod 2.6 Generowanie macierzy o zadanym wymiarze

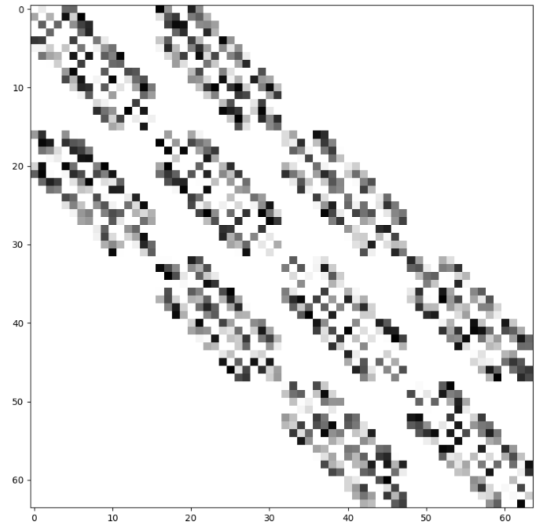
W celu kompresji i rysowania macierzy hierarchicznej użyto funkcji zaimplementowanych podczas laboratorium 3.

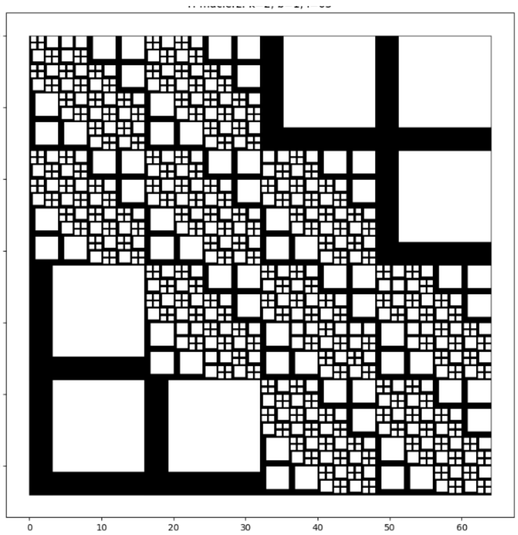
3. Otrzymane wyniki

Dla macierzy o rozmiarze 2^{3k}, gdzie $k \in \{2, 3, 4\}$, wykonano zadanie używając trzech algorytmów permutacji wierszy.

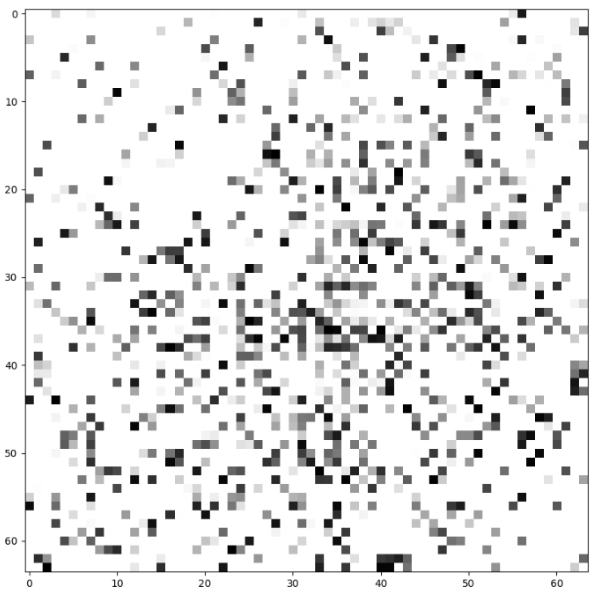
1) k = 2

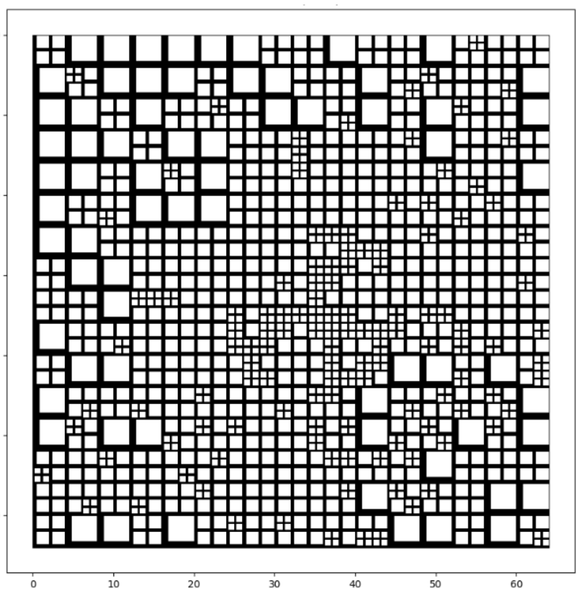
a) Brak permutacji



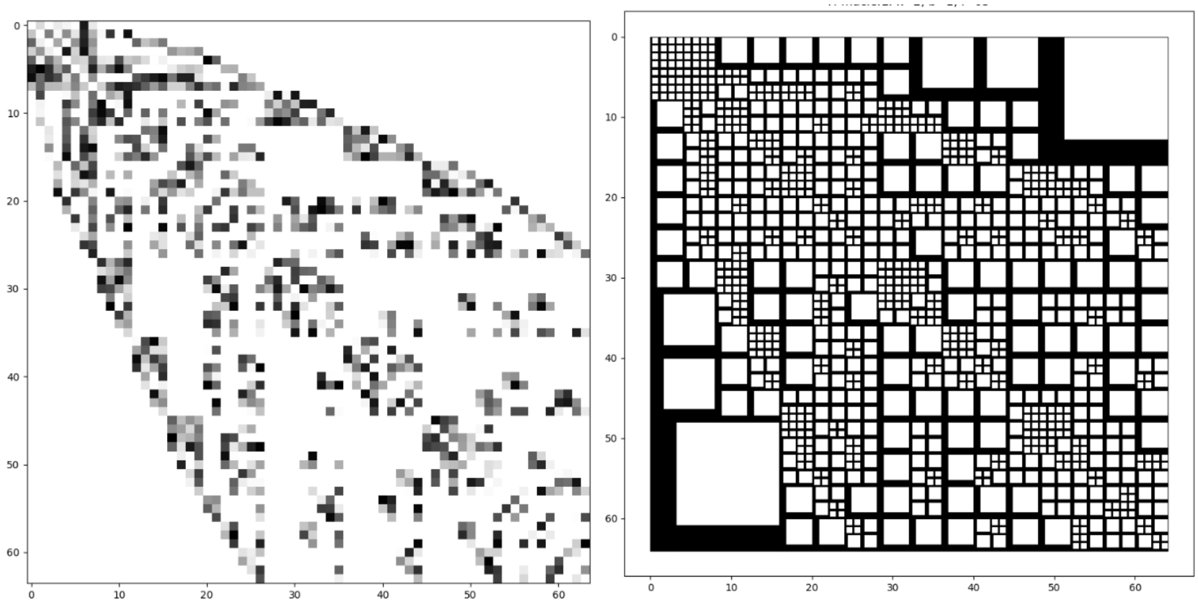


b) Permutacja minimum degree

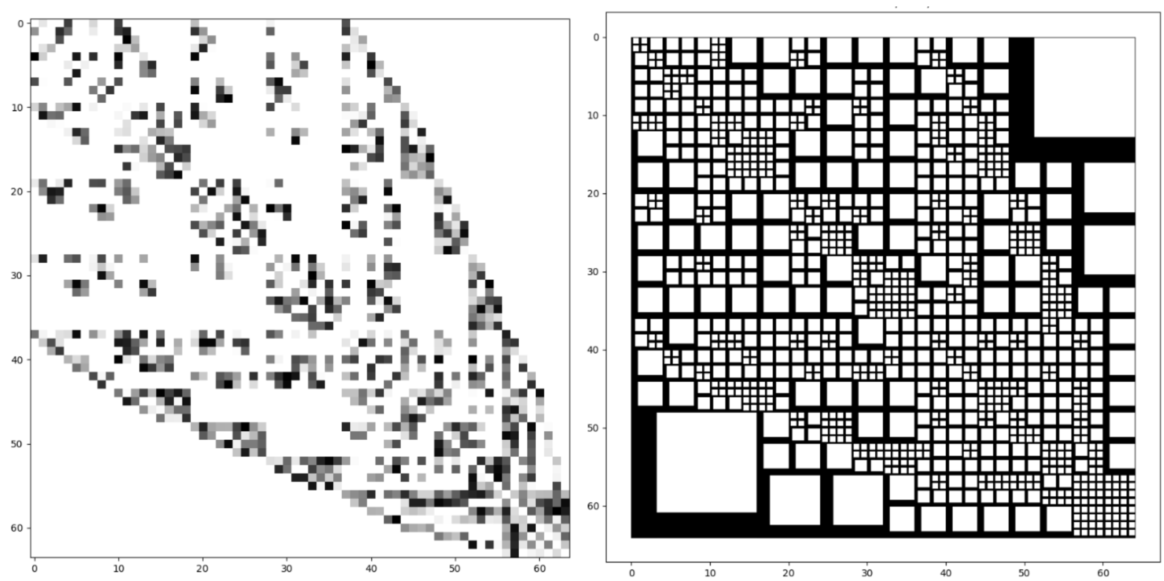




c) Permutacja Cuthill-McKee

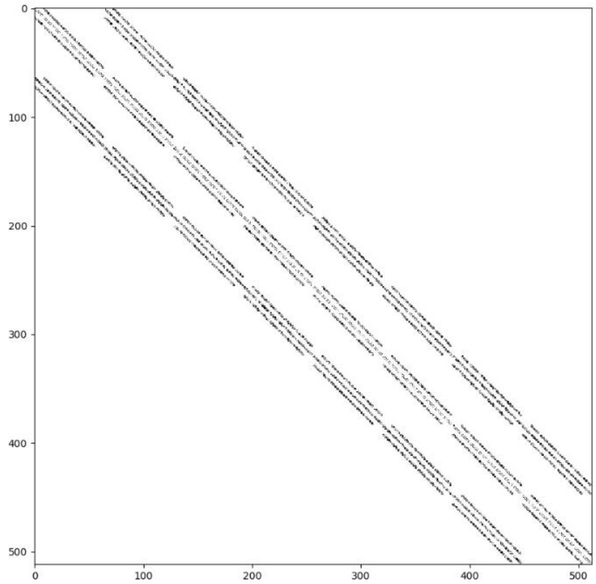


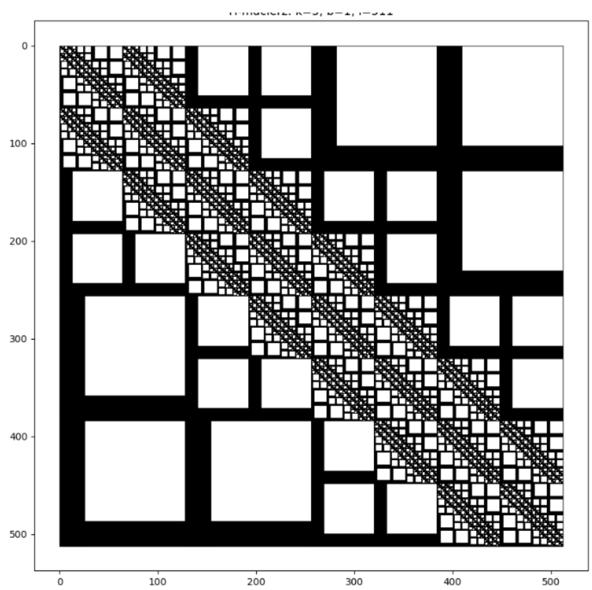
d) Permutacja Reversed Cuthill-McKee



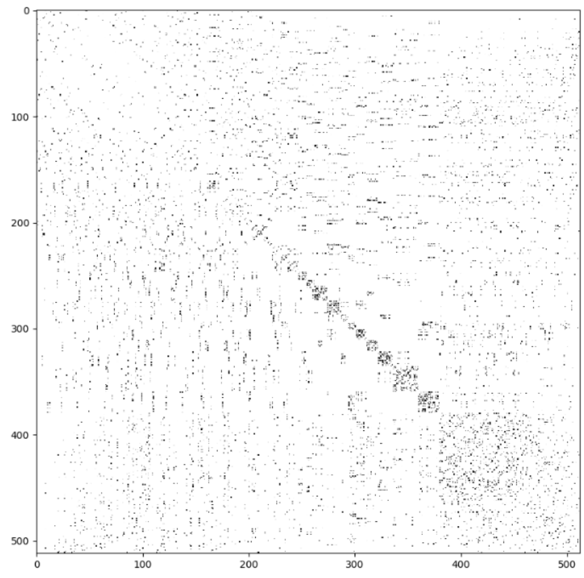
2) k = 3

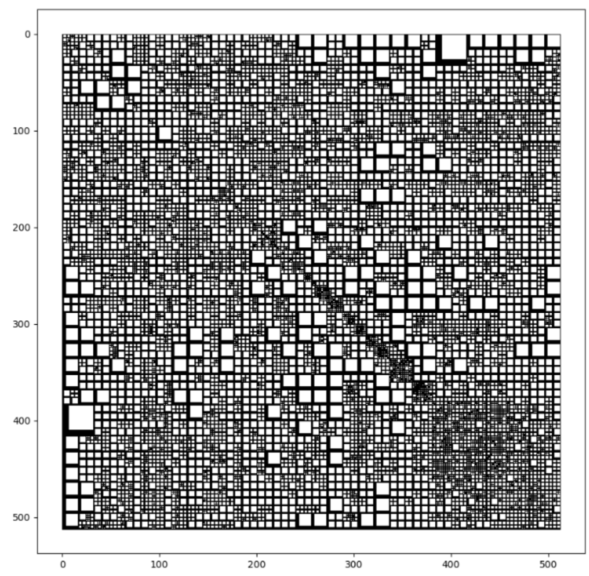
a) Brak permutacji



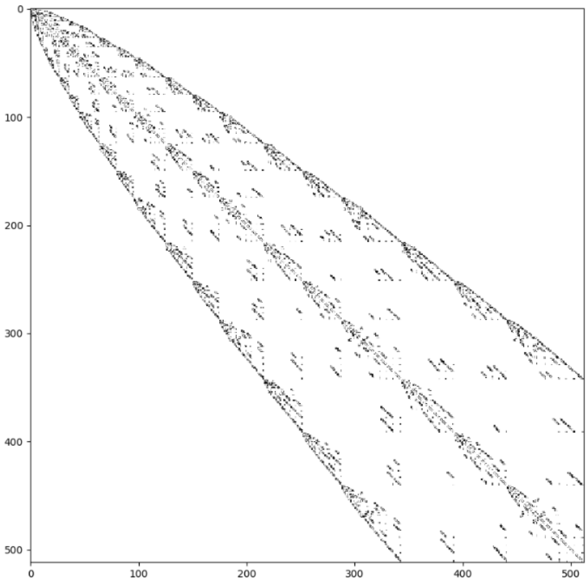


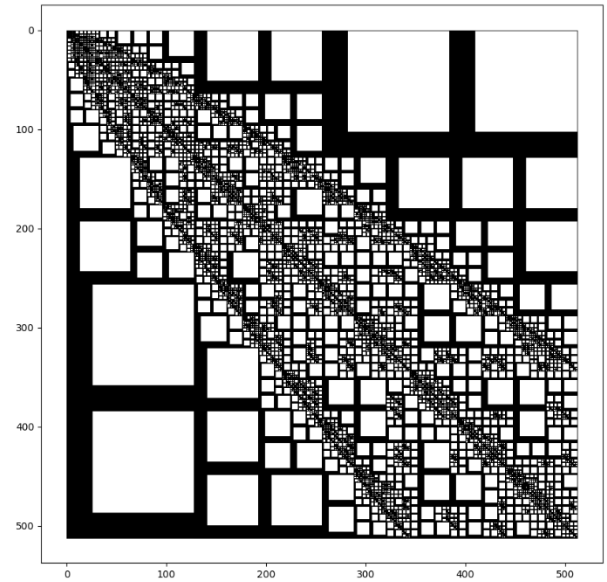
b) Permutacja minimum degree



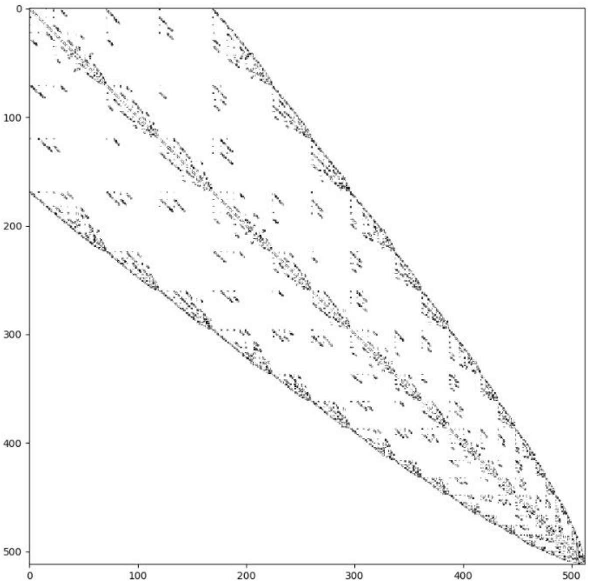


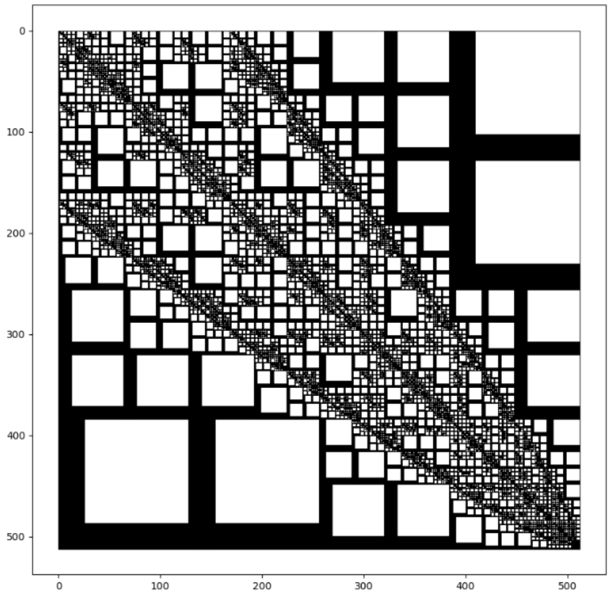
c) Permutacja Cuthill-McKee





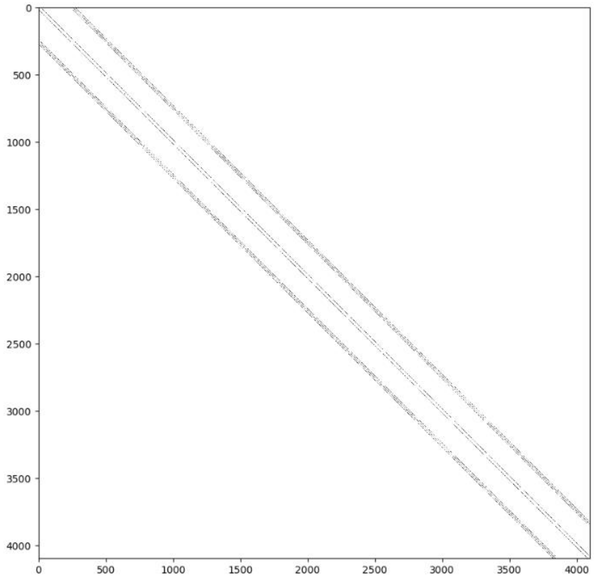
d) Permutacja Reversed Cuthill-McKee

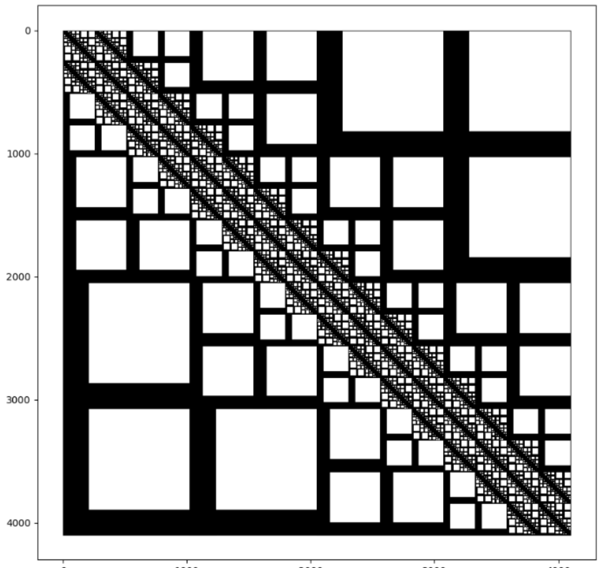




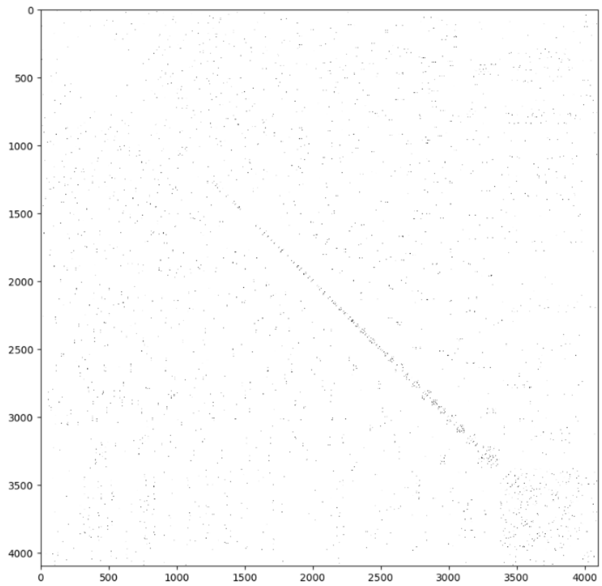
3) k = 4

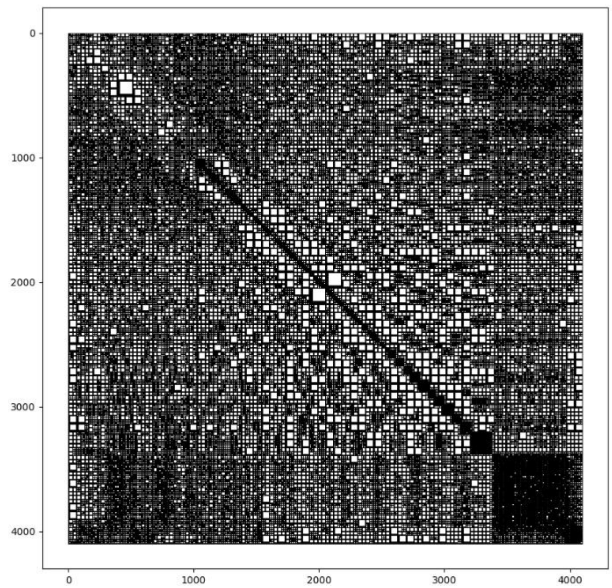
a) Brak permutacji



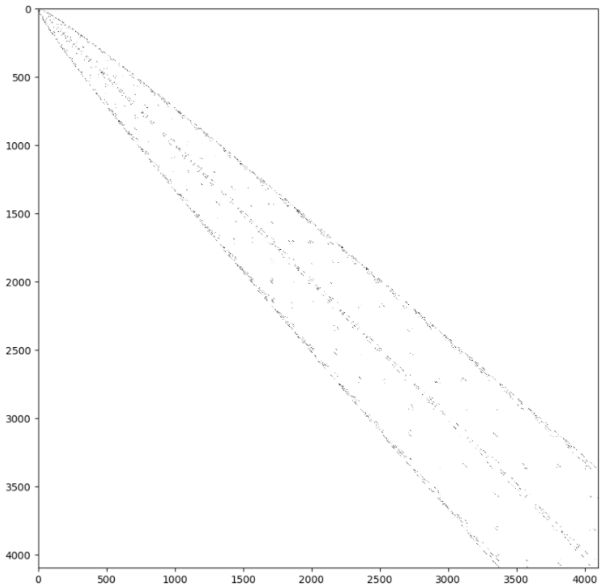


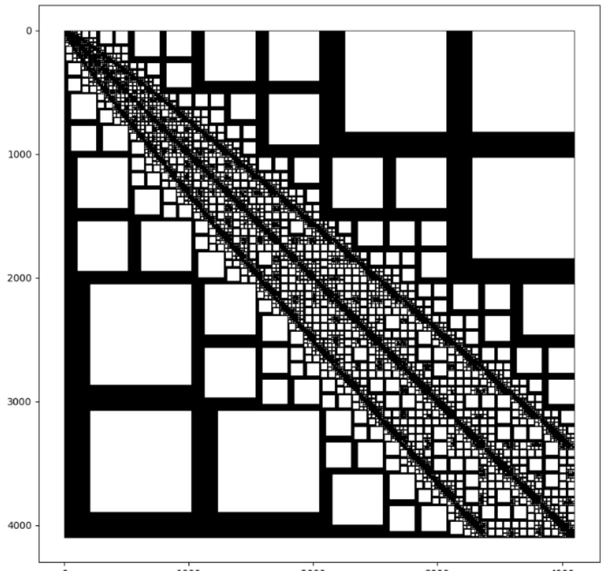
b) Permutacja minimum degree



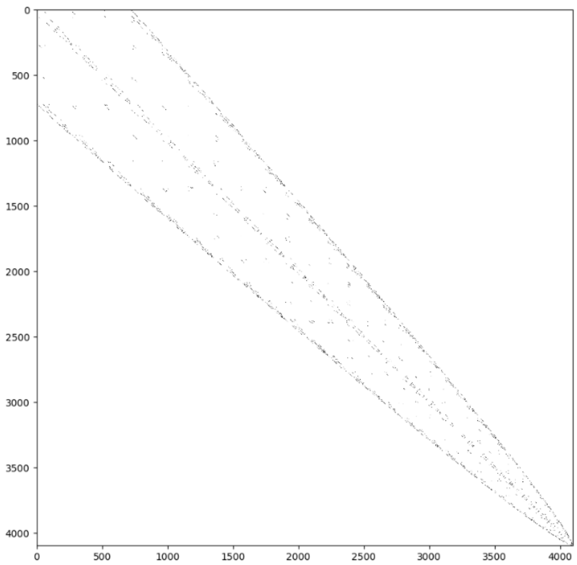


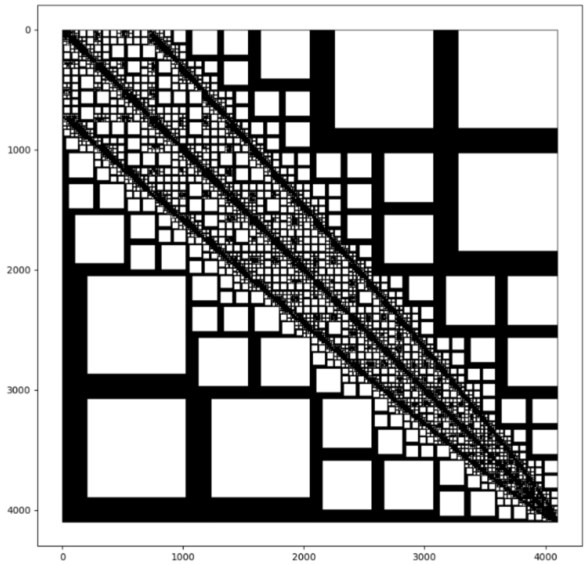
c) Permutacja Cuthill-McKee





d) Permutacja Reversed Cuthill-McKee





4. Wnioski

Na podstawie powyższych wykresów można stwierdzić, że można dokonać permutacji wierszy macierzy symetrycznej rzadkiej na podstawie grafu. Warto zwrócić uwagę, że w zależności od wyboru algorytmu permutacja wierszy jest inna i inaczej wygląda też macierz hierarchiczna. Dodatkowo powyższe algorytmy można uogólnić na przypadki, gdy permutowana macierz nie jest symetryczna.