Algorytmy macierzowe

Raport 4

Maciej Pięta, Michał Kobiera

1. Pseudokod algorytmu permutacji macierzy

1. Minimum degree permutation

Input: matrix - macierz sąsiedztwa grafu nieskierowanego

1. *Utwórz pustą permutację o długości n: permutation = [None, None, ..., None]*

2. *Utwórz pustą listę sąsiedztwa dla każdego wierzchołka: adjacent = {0: set(), 1: set(), ..., n-1: set()}*

3. *Dla każdej pary wierzchołków (i, j), jeśli matrix[i][j] != 0, dodaj j do sąsiedztwa i i dodaj i do sąsiedztwa j.*

4. *Dla i od 0 do n-1:*

a. Znajdź wierzchołek v o najmniejszym stopniu w grafie.

b. Ustaw permutation[i] na v.

c. Usuń v z sąsiedztwa wszystkich innych wierzchołków.

d. Zaktualizuj sąsiedztwo wszystkich sąsiadów v, usuwając v i dodając sąsiedztwo v - {v}.

e. Usuń v z listy sąsiedztwa.

5. *Zwróć permutację.*

2. Cuthil – McKee permutation

Input: matrix - macierz sąsiedztwa grafu nieskierowanego

1. Utwórz pustą permutację o długości n: permutation = []

2. Utwórz pustą listę sąsiedztwa dla każdego wierzchołka: adjacent = {0: set(), 1: set(), ..., n-1: set()}

3. Utwórz pustą listę odwiedzonych wierzchołków: visited = [False, False, ..., False]

4. Dla każdej pary wierzchołków (i, j), jeśli matrix[i][j] != 0, dodaj j do sąsiedztwa i i dodaj i do sąsiedztwa j.

5. Dla każdego wierzchołka v w posortowanej liście według stopnia:

a. Jeśli v nie został odwiedzony:

i. Wywołaj BFS(v)

6. Zwróć permutację.

Funkcja BFS(vertex):

1. Utwórz pustą kolejkę: queue = [vertex]

2. Oznacz wierzchołek vertex jako odwiedzony: visited[vertex] = True

3. Dopóki kolejka nie jest pusta:

a. Pobierz aktualny wierzchołek z kolejki: current_vertex = queue.pop(0)

b. Dodaj current_vertex do permutacji: permutation.append(current_vertex)

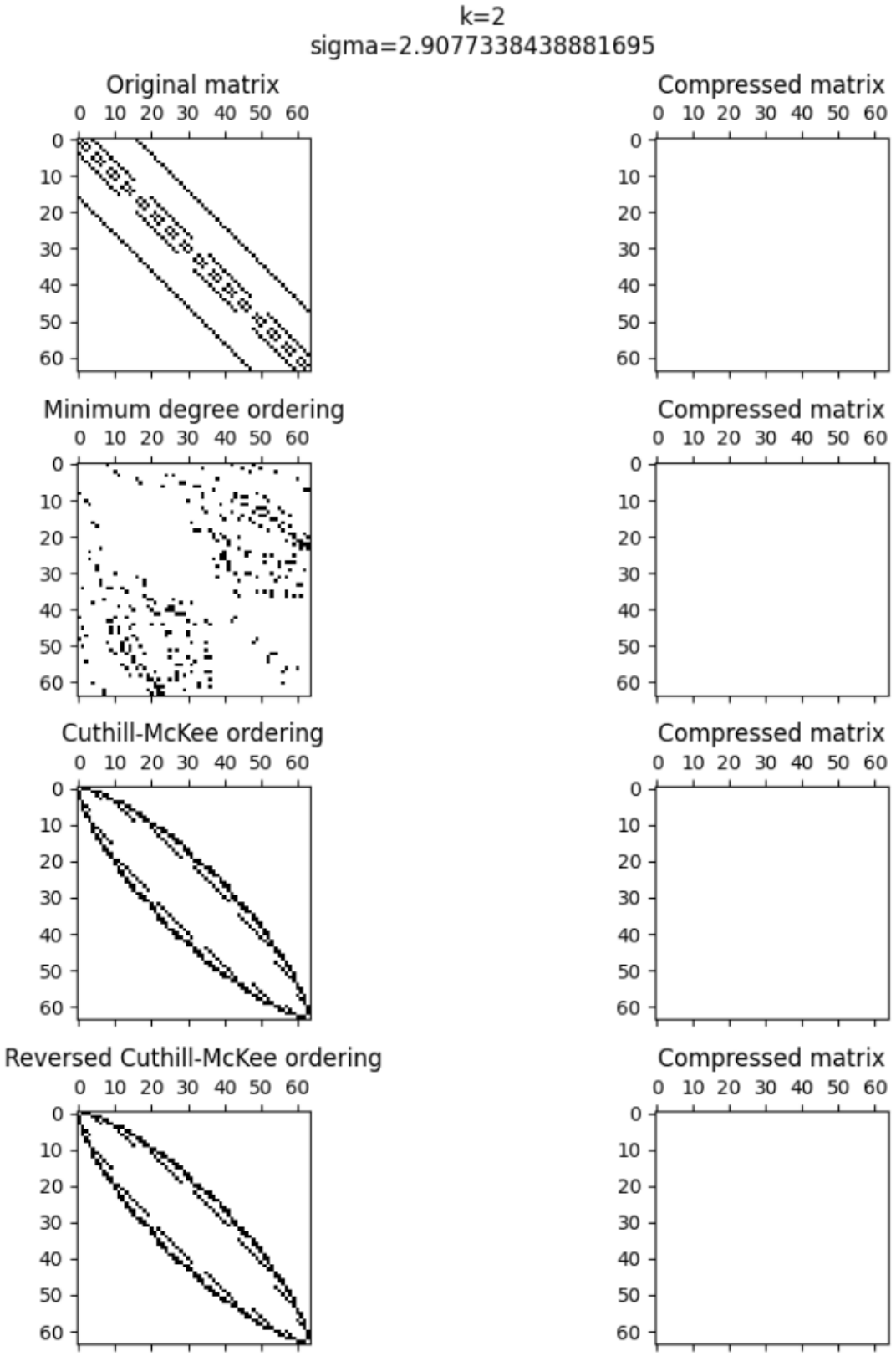
c. Dla każdego sąsiada v wierzchołka current_vertex, posortowanego według stopnia:

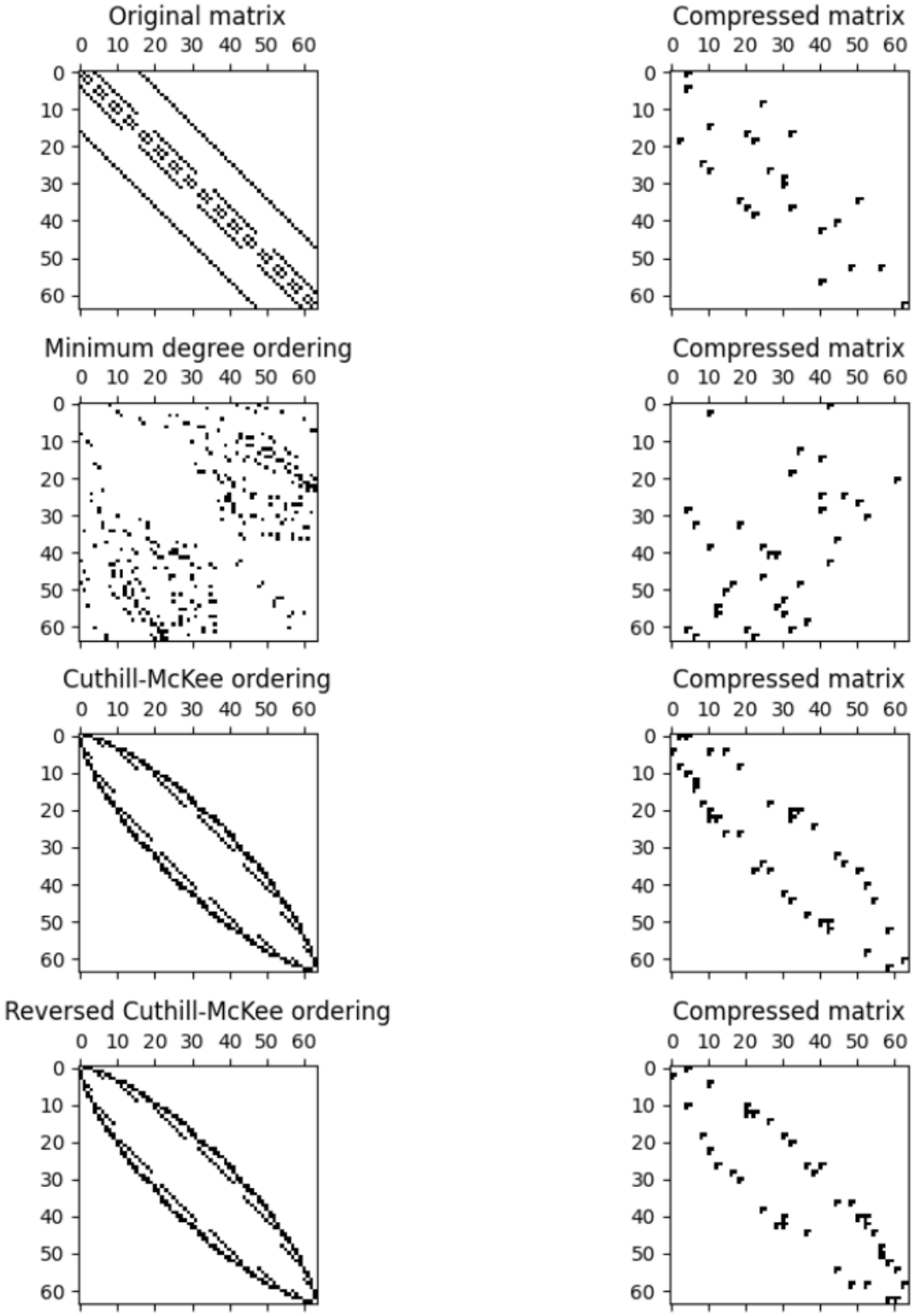
Jeśli v nie został odwiedzony:

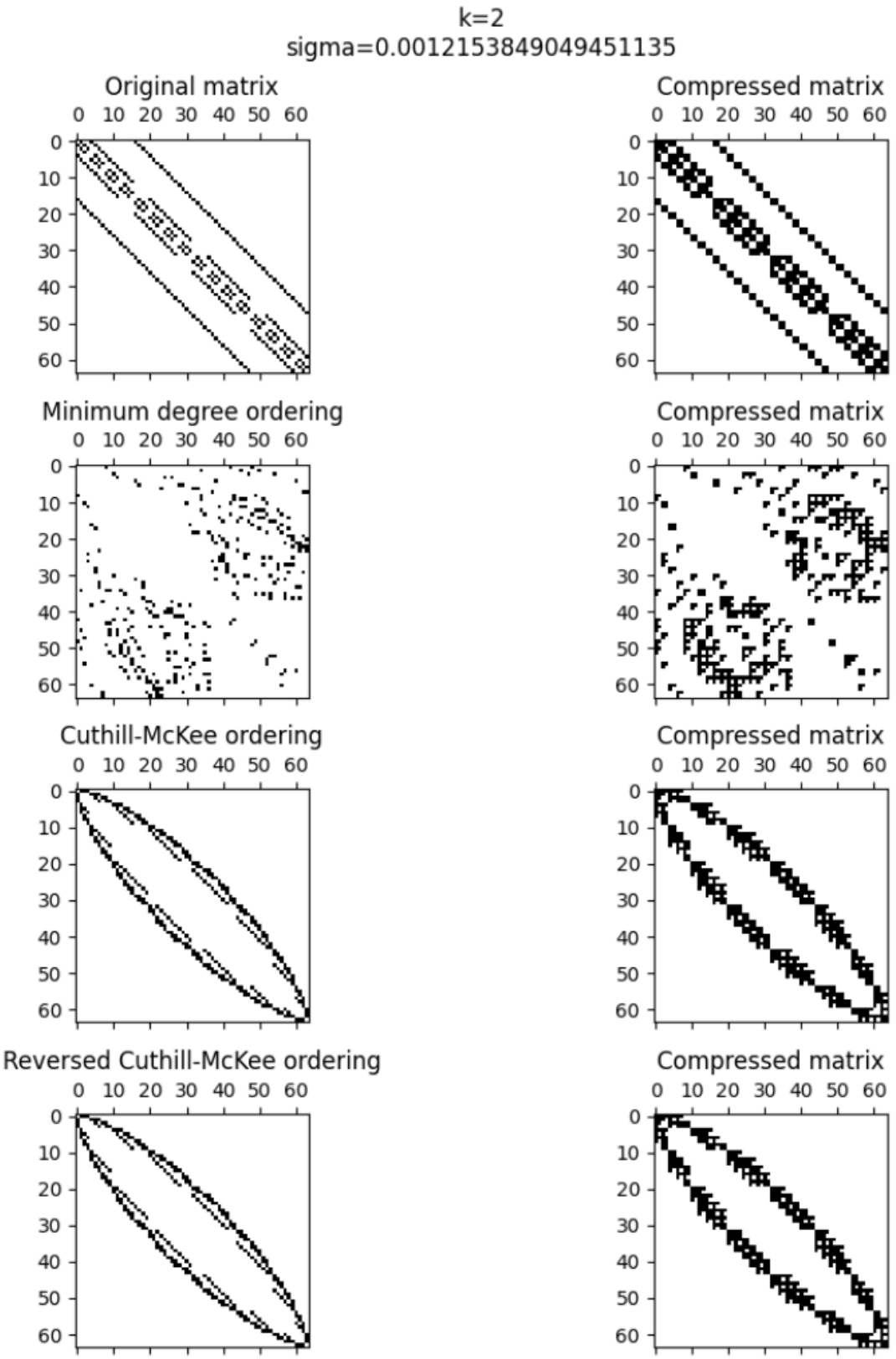
- Dodaj v do kolejki

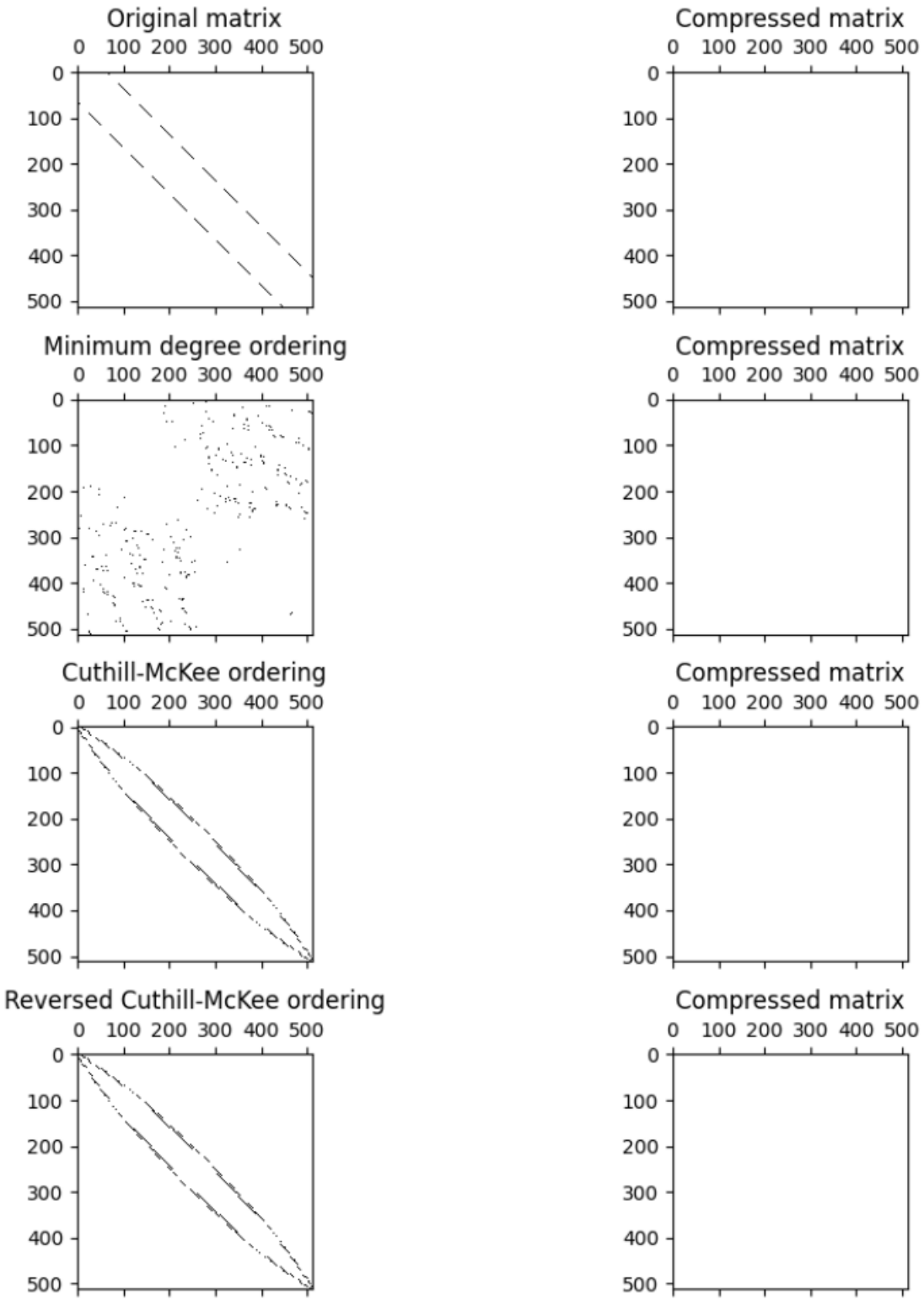
- Oznacz v jako odwiedzony: visited[v] = True

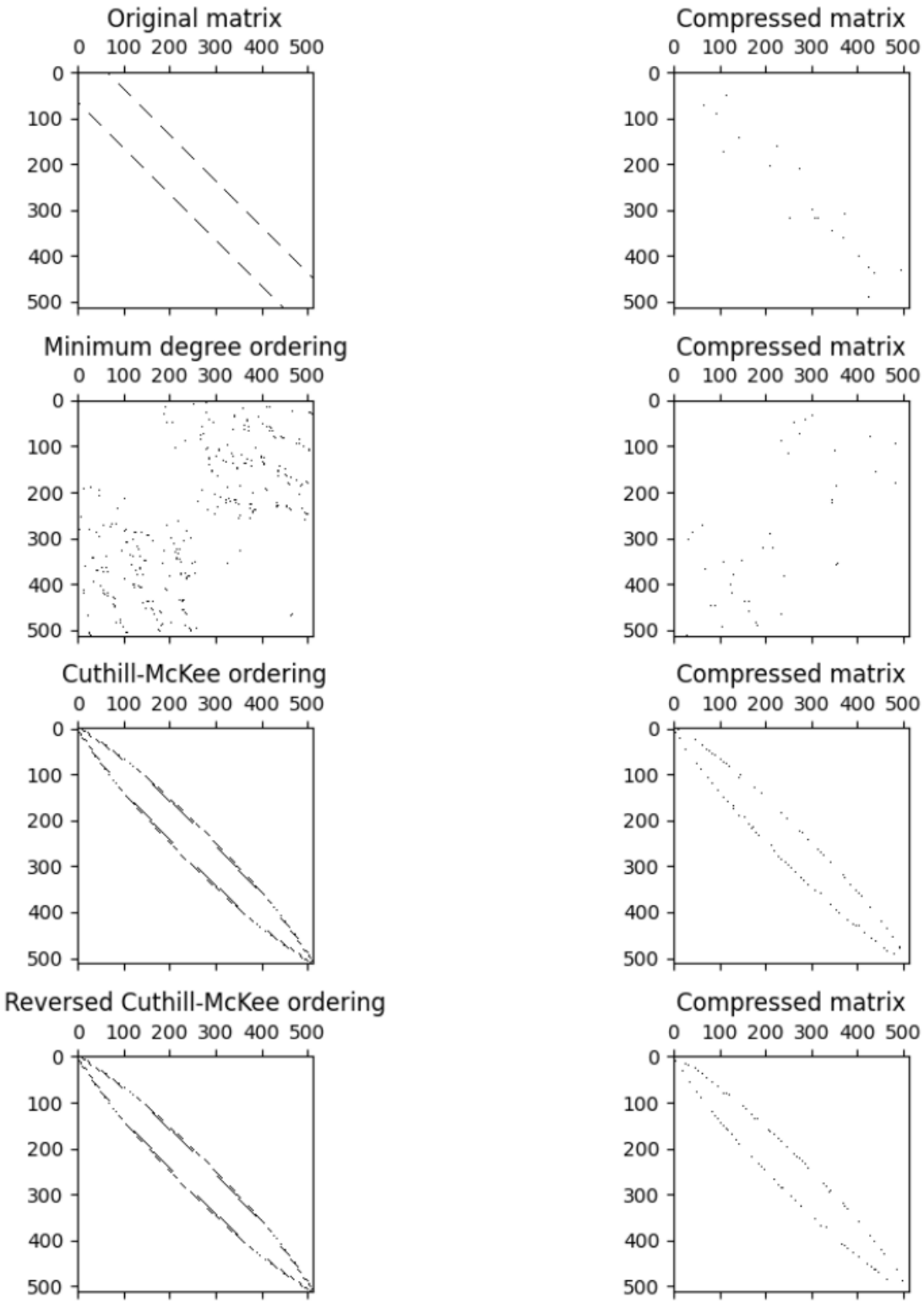
2. Porównanie wyników

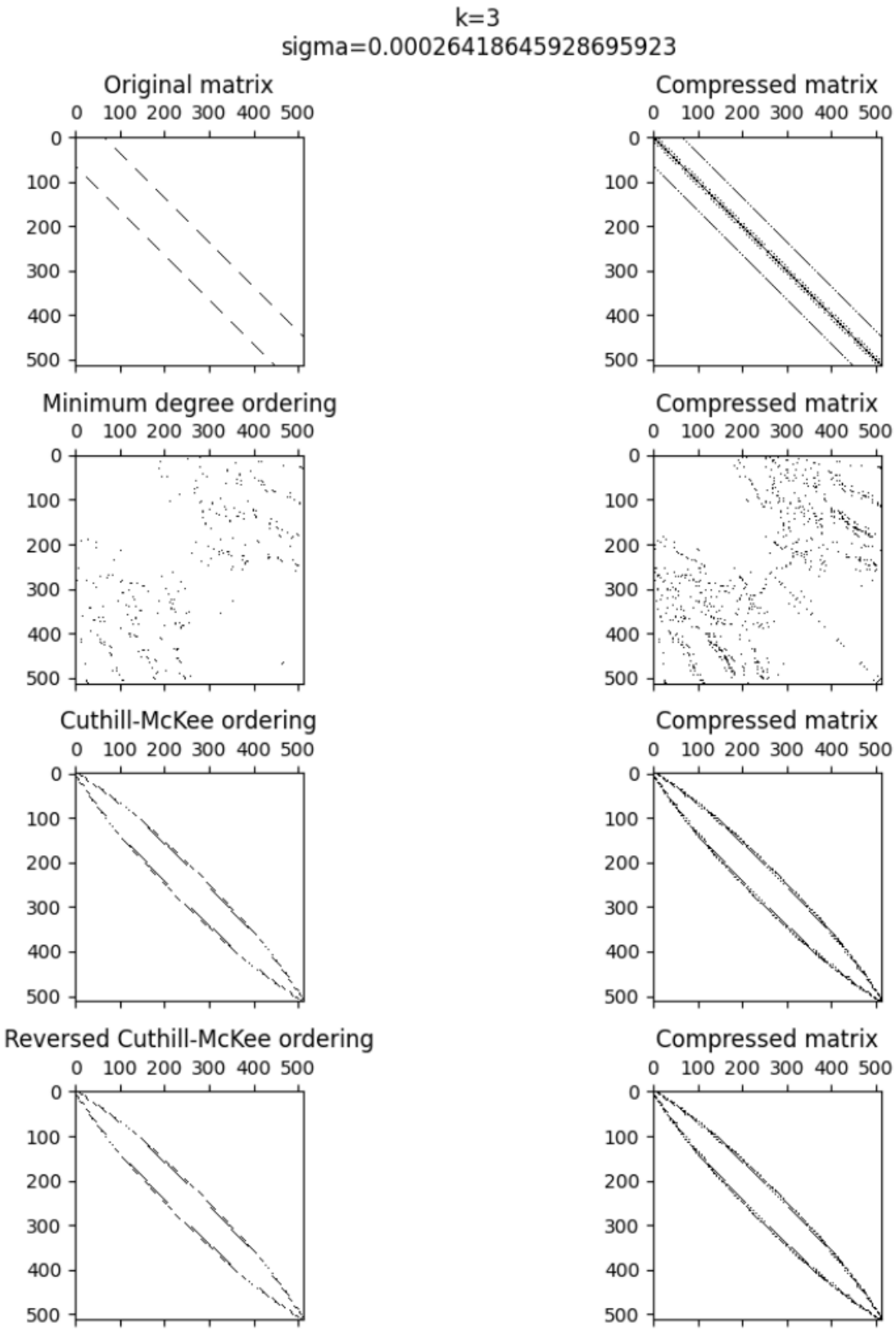


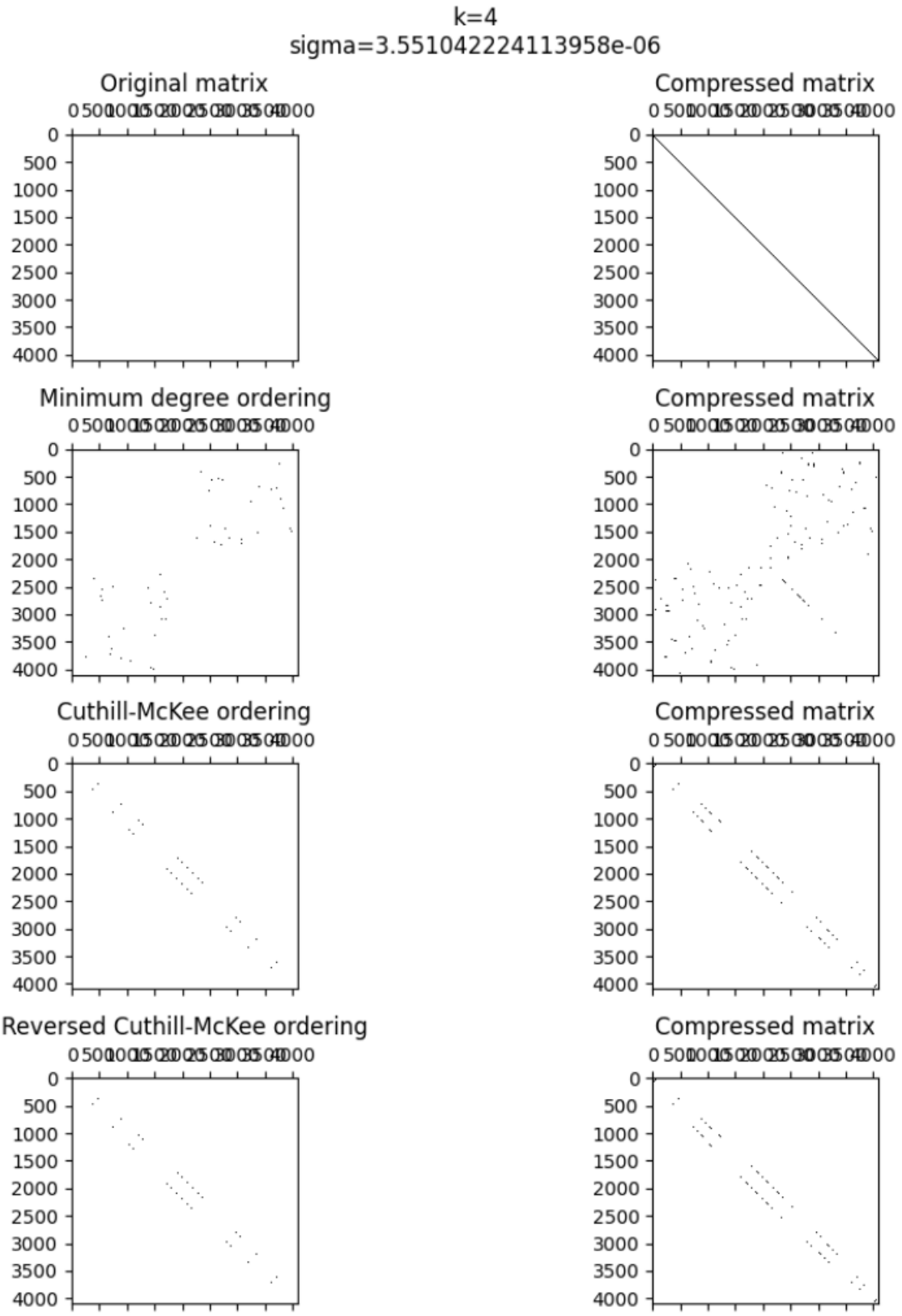

k=2
sigma=0.9034272951322354
Original matrix
Compressed matrix
Minimum degree ordering
Compressed matrix
Cuthill-McKee ordering
Compressed matrix
Reversed Cuthill-McKee ordering
Compressed matrix
0 10 20 30 40 50 60


k=2
sigma=0.0012153849049451135
Original matrix
Compressed matrix
Minimum degree ordering
Compressed matrix
Cuthill-McKee ordering
Compressed matrix
Reversed Cuthill-McKee ordering
Compressed matrix
0 10 20 30 40 50 60

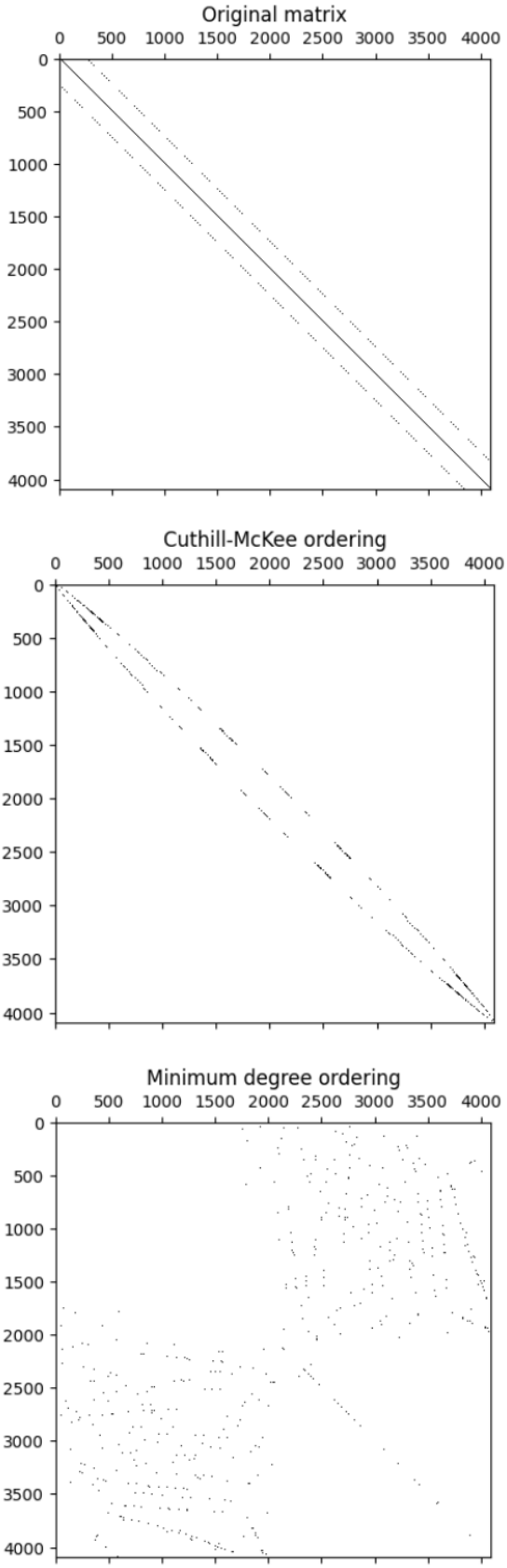

k=3
sigma=3.1042773022422088
Original matrix
Compressed matrix
Minimum degree ordering
Compressed matrix
Cuthill-McKee ordering
Compressed matrix
Reversed Cuthill-McKee ordering
Compressed matrix
0 100 200 300 400 500


k=3
sigma=0.9704135425417307
Original matrix
Compressed matrix
Minimum degree ordering
Compressed matrix
Cuthill-McKee ordering
Compressed matrix
Reversed Cuthill-McKee ordering
Compressed matrix
0 100 200 300 400 500


k=3
sigma=0.00026418645928695923
Original matrix
Compressed matrix
Minimum degree ordering
Compressed matrix
Cuthill-McKee ordering
Compressed matrix
Reversed Cuthill-McKee ordering
Compressed matrix
0 100 200 300 400 500



W celu poprawienia czytelności wykresów dla dużej macierzy, zostały dodatkowo wygenerowane figury dla k = 4


Original matrix
0 500 1000 1500 2000 2500 3000 3500 4000
0 500 1000 1500 2000 2500 3000 3500 4000
Cuthill-McKee ordering
0 500 1000 1500 2000 2500 3000 3500 4000
0 500 1000 1500 2000 2500 3000 3500 4000
Minimum degree ordering
0 500 1000 1500 2000 2500 3000 3500 4000
0 500 1000 1500 2000 2500 3000 3500 4000

3. Kluczowe fragmenty kodu

1. Generowanie macierzy

```python
def generate_3d_grid_matrix(k):
    size = 2**(3*k)
    grid_size = 2**k

    # Inicjalizacja rzadkiej macierzy
    matrix = sp.lil_matrix((size, size), dtype=float)

    # Iteracja przez elementy sześcienne siatki
    for i in range(grid_size):
        for j in range(grid_size):
            for l in range(grid_size):
                # Indeks aktualnego wierzchołka
                current_vertex = i * grid_size**2 + j * grid_size + l

                # Iteracja przez sąsiadów
                for di, dj, dl in [(-1, 0, 0), (1, 0, 0), (0, -1, 0), (0, 1, 0), (0, 0, -1), (0, 0, 1)]:
                    ni, nj, nl = i + di, j + dj, l + dl

                    # Sprawdzenie, czy sąsiad mieści się w siatce
                    if 0 <= ni < grid_size and 0 <= nj < grid_size and 0 <= nl < grid_size:
                        neighbor_vertex = ni * grid_size**2 + nj * grid_size + nl

                        # Przypisanie losowej wartości sąsiadowi
                        matrix[current_vertex, neighbor_vertex] = np.random.rand()

    return matrix
```

2. Permutacja

```python
def permutate_matrix(matrix, permutation):
    size = len(matrix)
    new_matrix = np.zeros((size, size))
    for i in range(size):
        new_matrix[i] = matrix[permutation[i]]

    new_matrix_cols = np.zeros((size, size))
    for i in range(size):
        new_matrix_cols[:, i] = new_matrix[:, permutation[i]]

    return new_matrix_cols
```

3. Algorytmy permutacji

```python
def minimum_degree_ordering(matrix):
    n = len(matrix)
    permutation = [None for _ in range(n)]
    adjacent = {i:set() for i in range(n)}

    for i in range(n):
        for j in range(n):
            if matrix[i][j] != 0:
                adjacent[i].add(j)

    for i in range(n):
        min_deg = n + 1
        min_idx = None
        for idx, deg in adjacent.items():
            if len(deg) < min_deg:
                min_idx = idx
                min_deg = len(deg)

        permutation[i] = min_idx


        for j in adjacent:
            adjacent[j].discard(min_idx)
        for v in adjacent[min_idx]:
            adjacent[v].update(adjacent[min_idx] - {v})
        adjacent.pop(min_idx)

    return permutation
```

```python
def cuthill_mckee_ordering(matrix):
    n = len(matrix)
    permutation = []
    adjacent = {i: set() for i in range(n)}
    visited = [False for _ in range(n)]

    def bfs(vertex):
        nonlocal visited, permutation, adjacent
        queue = [vertex]
        visited[vertex] = True

        while queue:
            current_vertex = queue.pop(0)
            permutation.append(current_vertex)

            for v in sorted(adjacent[current_vertex], key=lambda x: len(adjacent[x])):
                if not visited[v]:
                    queue.append(v)
                    visited[v] = True

    for i in range(n):
        for j in range(n):
            if matrix[i][j] != 0:
                adjacent[i].add(j)

    sorted_vertices = sorted(range(n), key=lambda x: len(adjacent[x]))

    for vertex in sorted_vertices:
        if not visited[vertex]:
            bfs(vertex)

    return permutation
```

Kompresja macierzy i rysowacz zostały napisane w poprzednim laboratorium.