Laboratorium 4.

Permutacje macierzy

Bartosz Hanc

1 Wstęp

Celem ćwiczenia było napisanie algorytmów permutacji macierzy: Minimum Degree, Cuthill–MacKee oraz Reversed Cuthill-MacKee oraz zbadanie ich wpływu na kompresje macierzy rzadkich opisujących topologię siatki trójwymiarowej zbudowanej z sześciennych komórek dla siatek o rozmiarach $4 \times 4 \times 4$, $8 \times 8 \times 8$ oraz $16 \times 16 \times 16$. Wygenerowano również wizualizacje uzyskanych po permutacji i kompresji macierzy hierarchicznych.

2 Kod rozwiązania

Poniżej zamieszczono kod rozwiązania w języku Python. Nieskierowany graf macierzy symetrycznej był reprezentowany przez strukturę `Graph`, szczególności graf został opisany przez listę sąsiedztwa (pole `Graph.adj`). Funkcje `MinimumDegree(G: Graph)`, `CuthillMacKee(G: Graph)` oraz `ReversedCuthillMcKee(G: Graph)` implementujące odpowiednio algorytmy zwracają listę opisującą permutację macierzy wejściowej reprezentowanej przez graf `G`. Macierze były reprezentowane jako dwuwymiarowe tablice z biblioteki Numpy.

```
class Graph:
    def __init__(self, A: np.ndarray):
        n = A.shape[0]
        self.adj = {u: {v for v in range(n) if A[u, v] > 0 and u != v}
                    for u in range(n)}
```

```
def MinimumDegree(G: Graph) -> np.ndarray:
    n = len(G.adj)
    order = []

    for _ in range(n):
        p = min(G.adj, key=lambda v: len(G.adj[v]))
        order.append(p)
        for u in G.adj[p]:
            G.adj[u] |= G.adj[p]
            G.adj[u] -= {p, u}
        del G.adj[p]

    return np.array(order)
```

```
from collections import deque

def CuthillMcKee(G: Graph) -> np.ndarray:
    queue, order, visit = deque(), [], {v: False for v in G.adj}
    p = min(G.adj, key=lambda x: len(G.adj[x]))
    order.append(p)
    visit[p] = True

    for u in sorted(G.adj[p], key=lambda x: len(G.adj[x])):
        queue.append(u)

    while len(queue) > 0:
        u = queue.popleft()

        if not visit[u]:
            order.append(u)
            visit[u] = True

            for v in sorted(G.adj[u], key=lambda x: len(G.adj[x])):
                if not visit[v]: queue.append(v)

    return np.array(order)
```

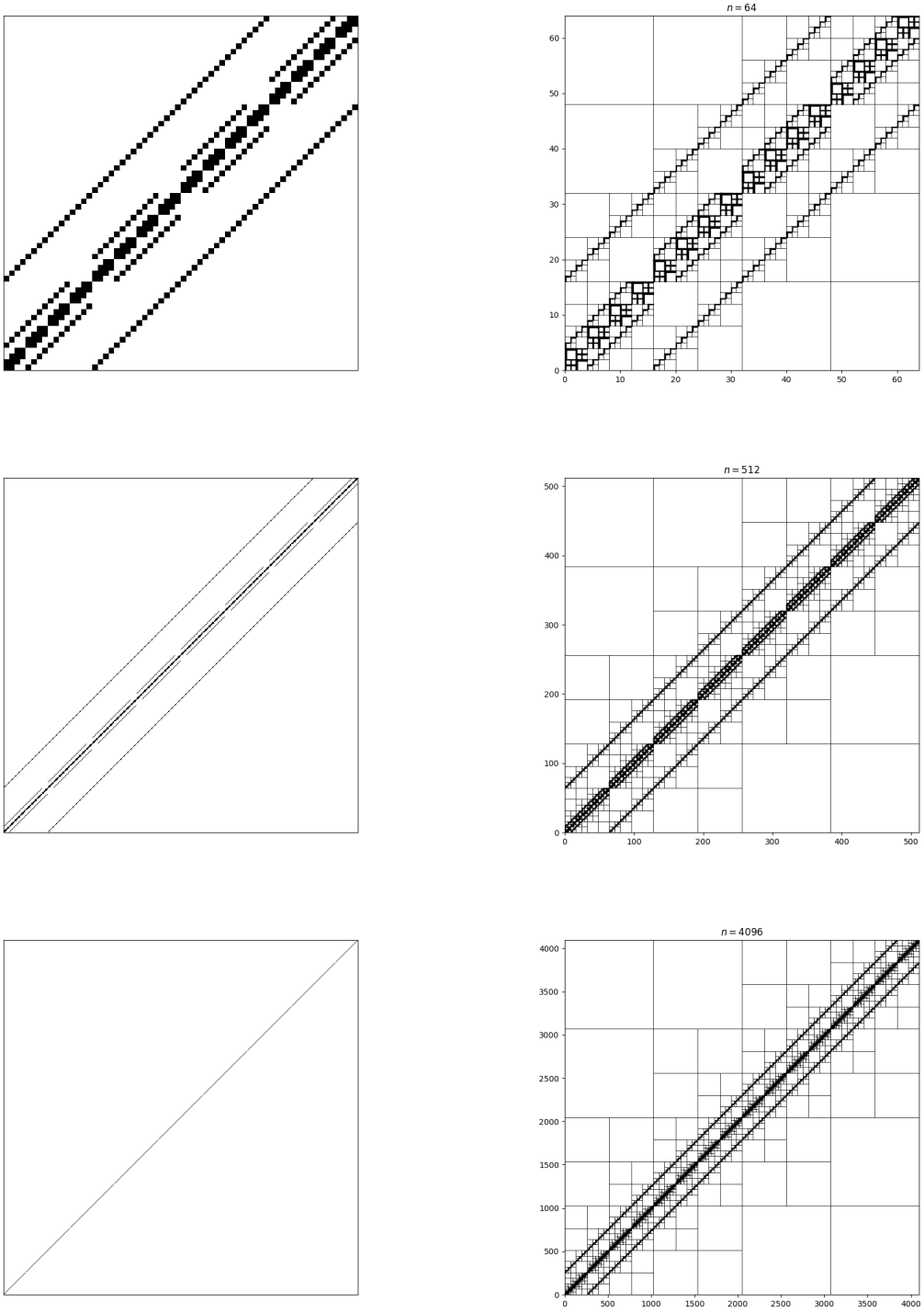
```
def ReversedCuthillMcKee(G: Graph) -> np.ndarray:
    return CuthillMcKee(G)[::-1]
```

Mając daną listę `order` opisującą permutację macierzy zwróconą przez jeden z powyższych algorytmów samą permutację macierzy `A` można wykonać jako

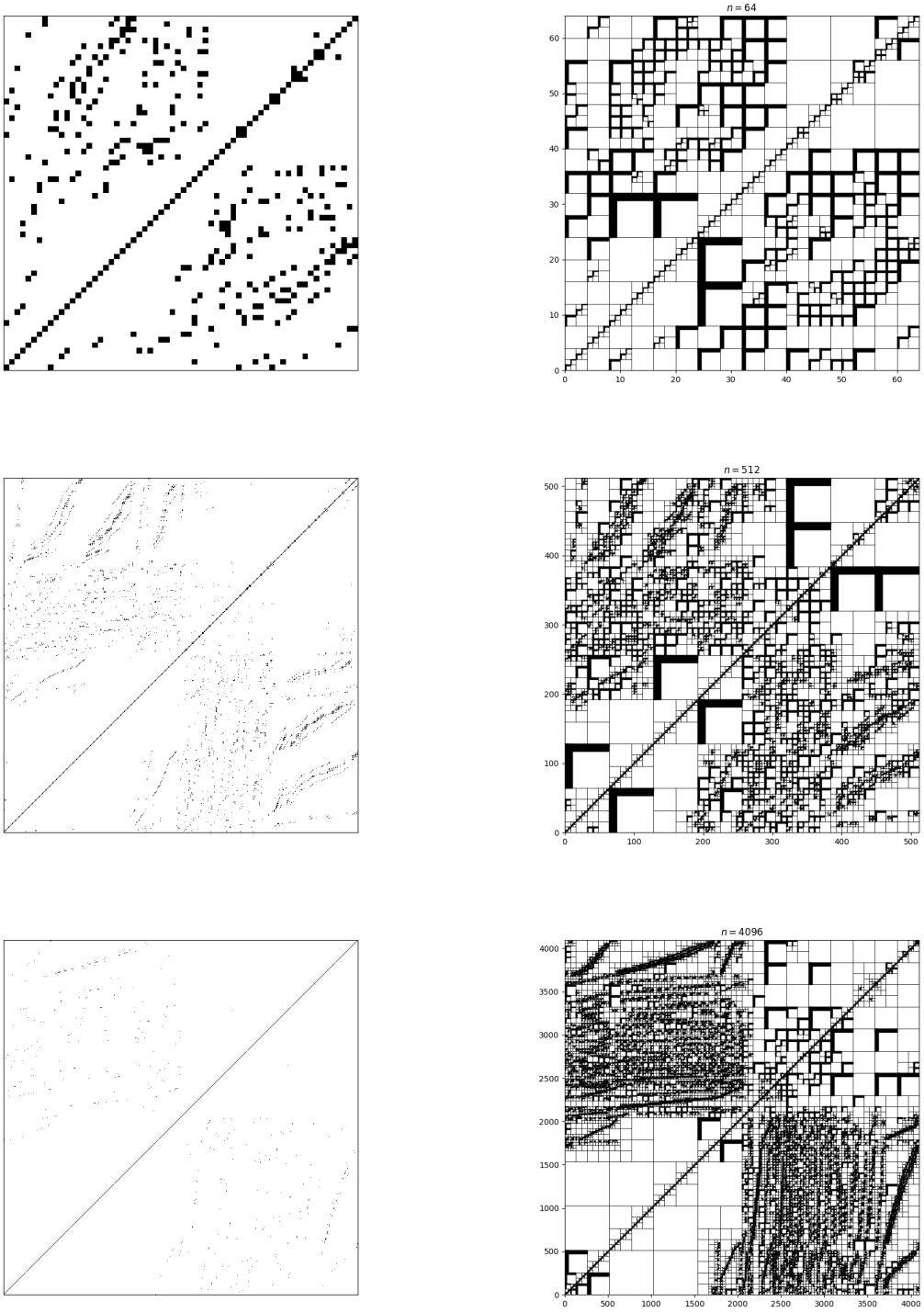
```
A = A[order, :][:, order]
```

3 Wyniki

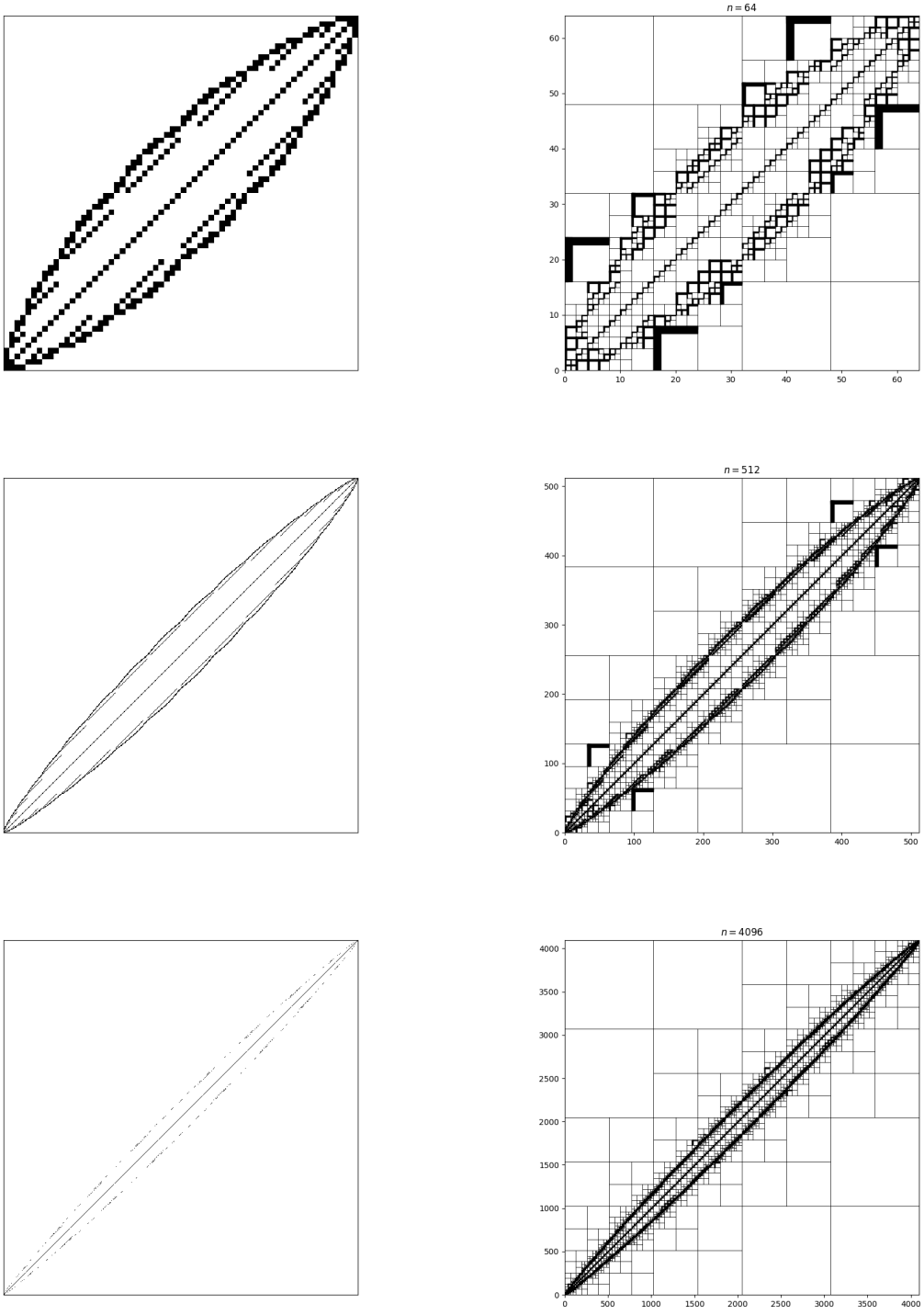
Zgodnie z poleceniem wygenerowano losowe macierze o strukturze opisującej topologię siatek trójwymiarowych o komórkach sześciennych rozmiarów $4 \times 4 \times 4$, $8 \times 8 \times 8$ oraz $16 \times 16 \times 16$. Poniżej na Rysunku 1 zamieszczono wzorce rzadkości tych macierzy oraz uzyskaną przez kompresję macierz hierarchiczną przed zastosowaniem permutacji.



Rysunek 1: Wzorce rzadkości oraz uzyskane macierze hierarchiczne dla macierzy wejściowych przed permutacją



Rysunek 2: Wzorce rzadkości oraz uzyskane macierze hierarchiczne dla macierzy wejściowych po permutacji korzystającej z algorytmu Minimum Degree



Rysunek 3: Wzorce rzadkości oraz uzyskane macierze hierarchiczne dla macierzy wejściowych po permutacji korzystającej z algorytmu Cuthill–MacKee