Algorytmy Macierzowe

Sprawozdanie IV

Grupa wtorek 13:00b

Michał Kuszewski i Michał Nożkiewicz

9 stycznia 2024

1 Opis zadania i użyte narzędzia

Naszym zadaniem było zaimplementowanie algorytmów permutacji macierzy i przetestowanie ich na macierzach opisujących topologię siatki trójwymiarowej.

1. Algorytm Minimum Degree
2. Algorytm Cuthill-McKee
3. Algorytm Reversed-Cuthill McKee

Do realizacji zadania użyliśmy języka Python. Korzystaliśmy z bibliotek numpy, matplotlib i scipy.

2 Pseudokody algorytmów

2.1 Minimum Degree

Algorytm 1: Minimum Degree

Data: $G = (V, E)$
Result: $V^{'}$ = permutation of V

$order = []$
$\forall\ v \in V, adj(v) = \{w : \{v, w\} \in E\}$
for $i \leftarrow 1$ **to** $|V|$ **do**
    $p = \underset{\forall v \in V}{\operatorname{argmin}} |adj(v)|$
    for $v \in adj(p)$ **do**
        $adj(v) = (adj(v) \cup adj(p)) \setminus \{v, p\}$
    $V = V \setminus \{p\}$
    $order.add(p)$
return $order$

2.2 Cuthill-Mckee

Algorytm 2: Cuthill-Mckee

Data: $G = (V, E)$
Result: $V^{'}$ = permutation of V

$order = []$
$\forall\ v \in V, adj(v) = [w : \{v, w\} \in E]$
for $v \in |V|$ **do**
    if *not* $v.visit$ **then**
        $Queue = []$
        $Queue.push(v)$
        $v.visit = True$
        $order.add(v)$
        while *not* $Queue.empty()$ **do**
            $v = Queue.pop()$
            for $s \in sorted(adj(p),\ key = len(adj(s)))$ **do**
                if *not* $s.visit$ **then**
                    $s.visit = True$
                    $order.add(s)$
                    $Queue.push(s)$
return $order$

2.3 Reversed Cuthill-Mckee

Algorytm 3: Reversed Cuthill-Mckee

Data: $G = (V, E)$
Result: $V^{'}$ = permutation of V

return $reverse(Cuthill - Mckee(order))$

3 Ważne fragmenty kodu

3.1 Funkcja do utworzenia grafu z podanej macierzy

```python
def get_adjacency(matrix):
    m, n = matrix.shape

    adjacency = {i : set() for i in range(m)}

    for i in range(m):
        for j in range(n):
            if i != j and matrix[i][j] > 0:
                adjacency[i].add(j)

    return adjacency
```

Rysunek 1: Funkcja get_adjacency()

3.2 Funkcja realizująca algorytm Minimum Degree

```python
def minimum_degree(matrix):
    m, n = matrix.shape
    result = []
    adjacency = get_adjacency(matrix)

    for i in range(m):
        best_vertex = -1
        best_value = n + 1

        for vertex, neighbors in adjacency.items():
            if len(neighbors) < best_value:
                best_value = len(neighbors)
                best_vertex = vertex

        for vertex in adjacency:
            adjacency[vertex].discard(best_vertex)

        for neighbor in adjacency[best_vertex]:
            adjacency[neighbor] |= adjacency[best_vertex].difference({neighbor})

        adjacency.pop(best_vertex)
        result.append(best_vertex)

    return result
```

Rysunek 2: Funkcja minimum_degree

3.3 Funkcja realizująca algorytm Cuthill-Mckee

```python
def cuthill_mckee(matrix):
    def bfs():
        while len(q) > 0:
            vertex = q.popleft()

            if visited[vertex]:
                continue

            visited[vertex] = True
            result.append(vertex)

            neighbors = sorted(list(adjacency[vertex]), key=lambda x: len(adjacency[x]))

            for neighbor in neighbors:
                if not visited[neighbor]:
                    q.append(neighbor)


    m, n = matrix.shape

    adjacency = get_adjacency(matrix)
    degrees = [(vertex, len(neighbors)) for vertex, neighbors in adjacency.items()]
    sorted_vertices = sorted(degrees, key=lambda x: x[1])

    for i in range(len(sorted_vertices)):
        sorted_vertices[i] = sorted_vertices[i][0]

    result = []

    q = deque()
    visited = [False for _ in range(m)]

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

    return result
```

Rysunek 3: Funkcja cuthill_mckee

3.4 Funkcja realizująca algorytm Reversed Cuthill-Mckee

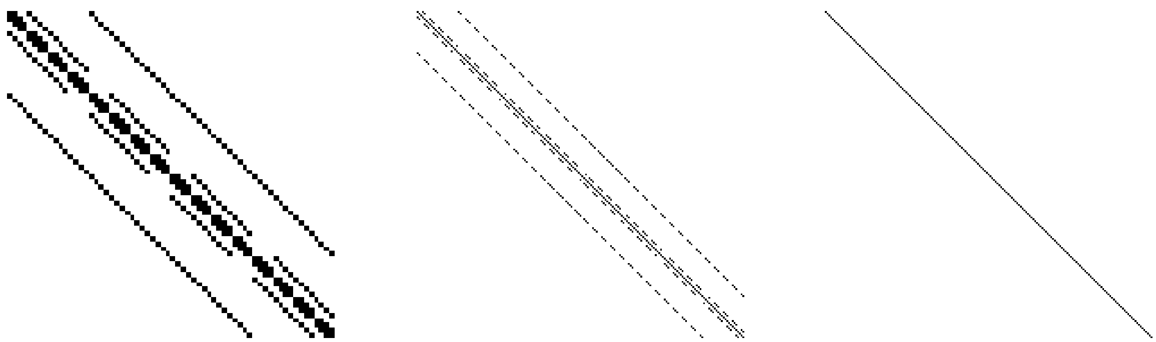
```
def reversed_cuthill_mckee(matrix):
    return cuthill_mckee(matrix)[::-1]
```

Rysunek 4: Funkcja reversed_cuthill_mckee

4 Wizualizacja wyników algorytmów

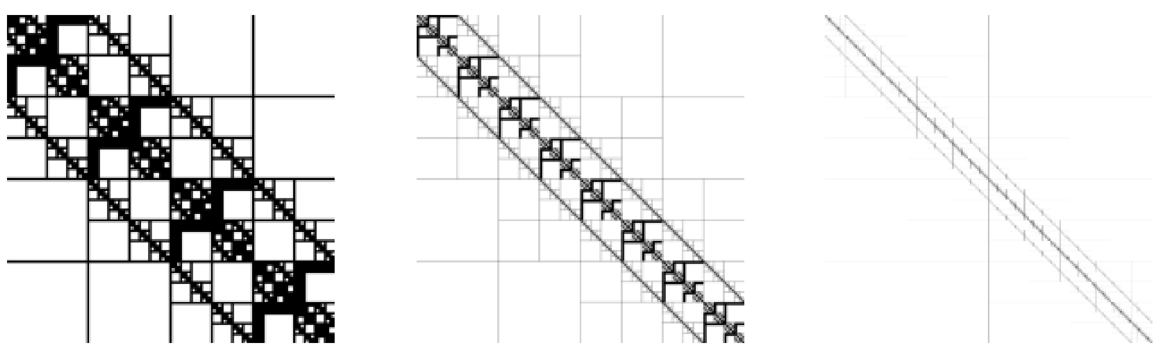
Do testów algorytmu użyliśmy macierzy opisujących topologię siatki trójwymiarowej. Macierze miały wymiary 64, 512, 4096. Na każdym z poniższych obrazków macierze są pokazane w tej kolejności.

4.1 Wzorzec rzadkości(przed permutacją)



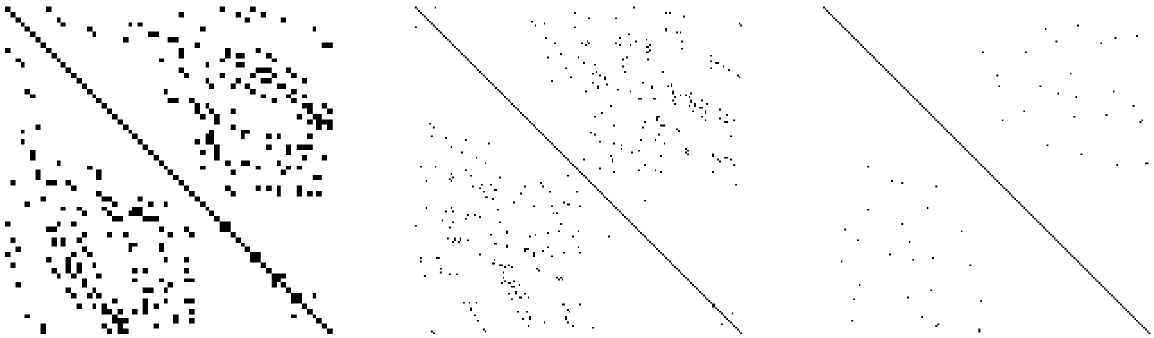
Rysunek 5: Wzorce rzadkości przed permutacją

4.2 Macierz skompresowana(bez permutacji)

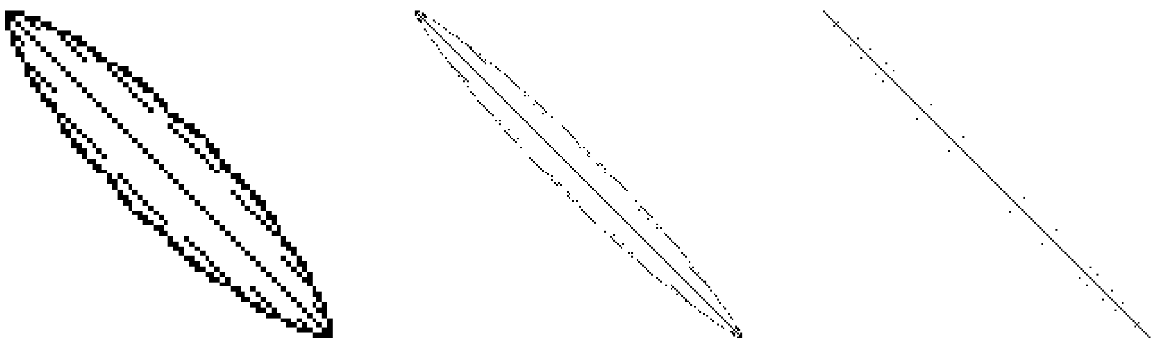


Rysunek 6: Wzorce rzadkości przed permutacją

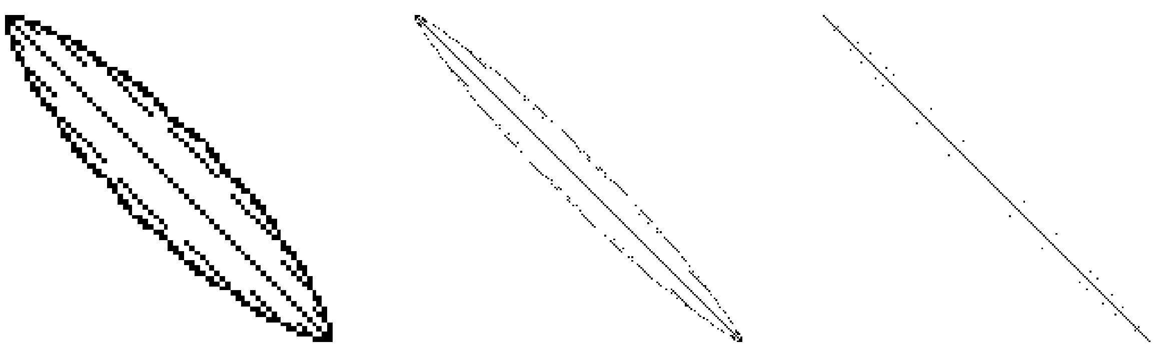
4.3 Wzorce rzadkości po permutacji



Rysunek 7: Wzorce po permutacji Minimum-Degree

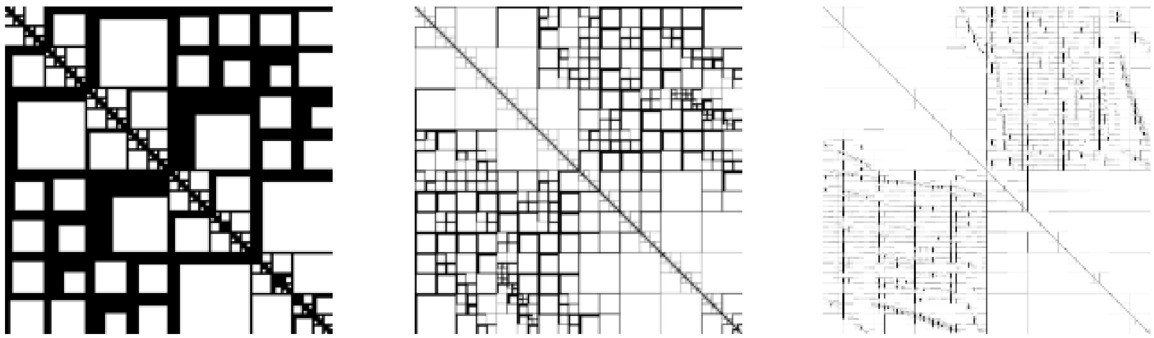


Rysunek 8: Wzorce po permutacji Cuthill_Mckee

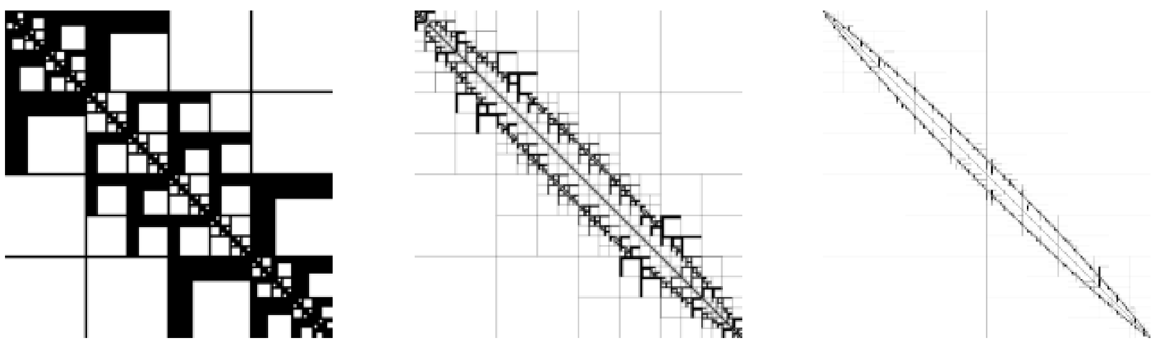


Rysunek 9: Wzorce po permutacji Reversed Cuthill_Mckee

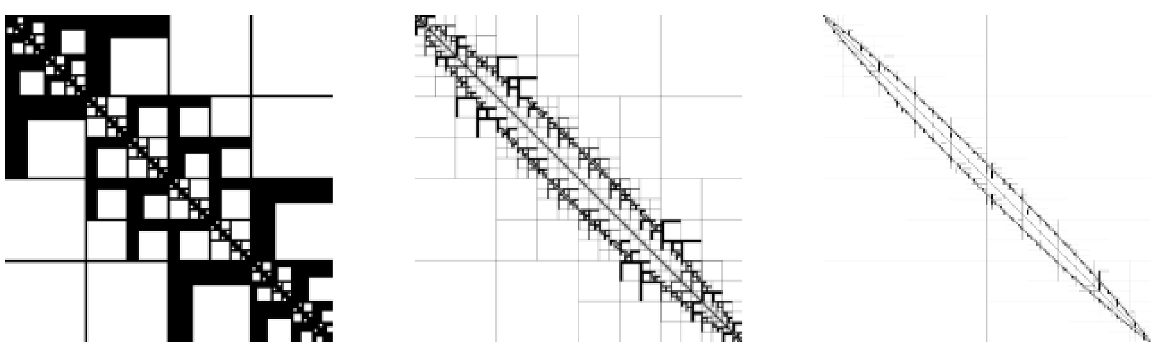
4.4 Macierz skompresowana po permutacji



Rysunek 10: Kompresja po permutacji Minimum-Degree



Rysunek 11: Kompresja po permutacji Cuthill_Mckee



Rysunek 12: Kompresja po permutacji Reversed Cuthill_Mckee

5 Porówananie stopnia kompresji

Dla każdego rozmiaru macierzy policzyliśmy ile zajmuje miejsca w bajtach, a następnie porównaliśmy to z ilością pamięci, które zajmują struktury do przechowywania macierzy hierarchicznych i tak dla każdego algorytmu permutacji poza odwróconym Cuthillem, gdyż wyniki były dla niego takie same jak dla zwykłego.

Wymiar Macierzy	Macierz	Po kompresji	Minimum-Degree	Cuthill-Mckee
64	32768	71128	46248	40808
256	2097152	746160	786464	508032
4096	134217728	6429512	11911104	4959672

Tabela 1: Tabela dla porównania stopnia kompresji

6 Wnioski

Widać, że lepszym algorytmem permutacji okazał się Cuthill-Mckeel, daje on najlepsze wyniki jeśli chodzi o kompresję, a także jest znacznie szybszy niż Minimum Degree, gdyż w trakcie obliczania permutacji nie dodaje nowych wierzchołków do grafu.