Aplikacja do Podziału Grafu

Dokumentacja Końcowa

Gniewko Wasilewski, Jan Szulc

kwiecień 2025

Spis treści

1 Cel projektu

Celem aplikacji jest podział grafu na zadan liczbę części w sposób minimalizujący liczbę przeciętych krawędzi. Podział musi zapewniać równomierność liczby wierzchołków w częściach w ramach określonego marginesu procentowego.

2 Środowisko pracy

Projekt realizowany w Visual Studio Code, w języku C, kompilowany za pomocą GCC na systemie Windows. Zarządzanie wersjami poprzez GitHub.

3 Funkcjonalność programu

Program umoźliwia:

- Wczytywanie grafu z pliku `.csrrg`
- Podział grafu na określon liczbę części
- Kontrolę równowagi liczby wierzchołków
- Zapis wyników w formacie tekstowym lub binarnym
- Konfigurację poprzez argumenty wiersza poleceń
- Obsługę błędów we/wy

4 Argumenty wiersza poleceń

Przykład wywołania:

```
./graph_partition input.csrrg output.txt -p 3 -m 10 -b
```

Argumenty:

- `input.csrrg` – plik wejściowy grafu
- `output.txt` – plik wyjściowy
- `-p <liczba>` – liczba części (domyślna: 2)
- `-m <liczba>` – margines procentowy (domyślny: 10%)
- `-b` – zapis binarny

5 Format pliku wejściowego

Plik `.csrrg` opisuje graf poprzez:

1. Maksymalną liczbę węzłów
2. Listę indeksów węzłów
3. Wskaźniki na początkowe indeksy
4. Listy sąsiedztwa

6 Obsługiwane błędy i komunikaty

Table 1: Obsługiwane błędy i ich kody powrotu

Błęd	**Komunikat**	**Kod powrotu**
Brak pliku wejściowego	Błąd: Nie podano pliku wejściowego	1
Błędny format pliku	Błąd: Niepoprawny format pliku	2
Błęd odczytu pliku	Błąd: Nie można otworzyć pliku	3
Nieprawidłowa liczba części	Błąd: Liczba części musi być >1	4

7 Przykłady użycia

7.1 Podział na 3 części, margines 5%

```
./graph_partition graf.csrrg wynik.txt -p 3 -m 5
```

7.2 Podział na 4 części, zapis binarny

```
./graph_partition graf.csrrg wynik.bin -p 4 -b
```

8 Przykładowy graf

9 Algorytm podziału grafu

1. Wczytanie grafu z pliku `.csrrg`
2. Obliczenie początkowego podziału (Spectral Clustering + k-średnich)
3. Optymalizacja przez minimalizację przeciętych krawędzi
4. Sprawdzenie równowagi liczby wierzchołków
5. Zapis wyników

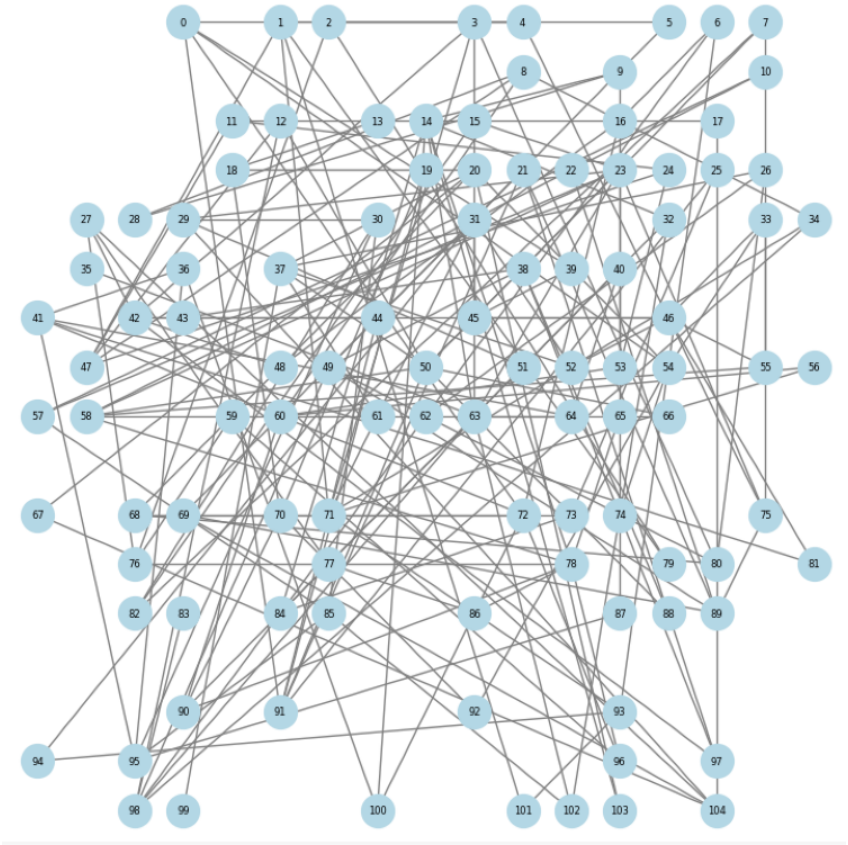

Figure 1: Przykładowy graf podlegający podziałowi

10 Opis funkcji

- `liczba_wierzcholkow()` – liczba wierzchołków w grafie
- `oblicz_macierz_Laplasjana()` – wyznaczenie macierzy Laplasjana
- `spectral_clustering()` – podział przez clustering
- `optymalizuj_podzial()` – minimalizacja przeciętych krawędzi
- `sprawdz_rownowage()` – kontrola marginesu
- `zapisz_wyniki()` – zapis danych do pliku

11 Pseudokod

```
podziel_graf(graf, liczba_czesci, margines)
  liczba_wierzcholkow <- liczba_wierzcholkow(graf)
  L <- oblicz_macierz_Laplasjana(graf)
  wektory_wlasne <- oblicz_wektory_wlasne(L)
  podzial <- k_srednich(wektory_wlasne, liczba_czesci)

  repeat:
    optymalizuj_podzial()
```

```
    sprawdz_rownowage()
  until margines OK

  zapisz_wyniki(podzial)
```

12 Struktura programu

- **main.c**: funkcja główna i obsługa argumentów
- **graph.c**: operacje na grafach
- **utils.c**: funkcje pomocnicze
- **partition.c**: algorytmy podziału

13 Wynik programu

Po zakończeniu działania programu w pliku wynikowym zapisany jest podział grafu w formacie:

`[indeks węzła];[pozycja x];[pozycja y];[klaster];[sąsiad 1],[sąsiad 2],...`

Przykład:

```
0;3;0;4;72,39,91,4,54
1;5;0;0;47,4,79,101,71
2;6;0;3;76,79,5
3;9;0;2;63,71,80,42
4;10;0;0;75
5;13;0;2;49
6;14;0;3;93,16,98
```

Każda linia odpowiada jednemu węzłowi: jego indeks, pozycje (x, y), przypisany klaster oraz lista indeksów sąsiadów.

14 Repozytorium aplikacji

https://github.com/JanSzulc/jimp2

15 Podsumowanie

Aplikacja do podziału grafów umożliwia skuteczną analizę struktur sieciowych, z elastyczną konfiguracją parametrów i solidną obsługę błędów. Potencjalne kierunki rozwoju obejmują zaawansowane techniki optymalizacji i integrację z systemami wizualizacji.