Dokumentacja Implementacyjna - Aplikacja do Podziału Grafu

Gniewko Wasilewski, Jan Szulc

31 marca 2025

1 Wstęp

Dokumentacja ta przedstawia implementację aplikacji w języku C, której zadaniem jest podział grafu na zadaną liczbę części, przy minimalizacji liczby przeciętych krawędzi i zapewnieniu równomiernego podziału wierzchołków w tych częściach. Program realizuje algorytm oparty na heurystyce optymalizacyjnej, który pozwala na efektywne dzielenie dużych grafów.

2 Opis aplikacji

Aplikacja została zaimplementowana w języku C i działa w trybie terminalowym. Program umożliwia podział grafu na zadane przez użytkownika liczby części, a także zapis wyników w formacie tekstowym lub binarnym. Użytkownik może dostosować parametry podziału, takie jak liczba części oraz margines różnicy liczby wierzchołków w częściach. Aplikacja obsługuje dane wejściowe w formacie `.csrrg`, który jest skompresowaną reprezentacją grafu.

3 Algorytm podziału grafu

Algorytm realizujący podział grafu jest oparty na podejściu optymalizacyjnym, w którym celem jest zminimalizowanie liczby przeciętych krawędzi przy zachowaniu równomiernego podziału wierzchołków. Algorytm można opisać w kilku etapach:

3.1 Etap 1: Wczytanie grafu

Pierwszym krokiem jest wczytanie grafu z pliku `.csrrg`. Program odczytuje dane z pliku, takie jak lista wierzchołków oraz ich połączenia. Na tej podstawie tworzymy macierz sąsiedztwa lub listy sąsiedztwa.

3.2 Etap 2: Obliczenie początkowego podziału

Aby uzyskać początkowy podział, graf jest dzielony na `p` części (gdzie `p` jest liczbą podaną przez użytkownika lub domyślnie wynosi 2). Początkowy podział może być wykonany przy użyciu algorytmu Spectral Clustering, który bazuje na wartościach i wektorach własnych macierzy Laplasjana, a następnie na algorytmie k-średnich, który przydziela wierzchołki do odpowiednich części.

3.3 Etap 3: Heurystyka minimalizacji przeciętych krawędzi

Po dokonaniu początkowego podziału, program przechodzi do optymalizacji, starając się zmniejszyć liczbę przeciętych krawędzi. Algorytm iteracyjnie przesuwa wierzchołki pomiędzy częściami, minimalizując liczbę przeciętych krawędzi. Przesunięcia są wykonywane, jeśli prowadzą do zmniejszenia liczby przecięć, przy jednoczesnym zachowaniu równowagi w liczbie wierzchołków.

3.4 Etap 4: Sprawdzenie równowagi liczby wierzchołków

Po dokonaniu optymalizacji, program sprawdza, czy liczba wierzchołków w każdej części nie różni się od siebie o więcej niż zadany margines procentowy. Jeśli margines jest przekroczony, algorytm ponownie podejmuje próbę optymalizacji.

3.5 Etap 5: Zakończenie i zapis wyników

Gdy podział jest gotowy, wyniki są zapisywane do pliku wyjściowego w wybranym formacie. Program może zapisać dane w formacie tekstowym lub binarnym, w zależności od wyboru użytkownika.

4 Opis funkcji

Aplikacja zawiera następujące główne funkcje:

4.1 Funkcja `liczba_wierzcholkow()`

Funkcja ta zwraca liczbę wierzchołków w grafie. Zależnie od reprezentacji grafu, może to być długość listy sąsiedztwa lub rozmiar macierzy sąsiedztwa.

4.2 Funkcja `oblicz_macierz_Laplasjana()`

Funkcja oblicza macierz Laplasjana grafu. Jest to różnica między macierzą stopni a macierzą sąsiedztwa.

4.3 Funkcja `spectral_clustering()`

Funkcja ta realizuje algorytm Spectral Clustering. Używa macierzy Laplasjana, oblicza wartości i wektory własne, a następnie przypisuje wierzchołki do poszczególnych części grafu na podstawie algorytmu k-średnich.

4.4 Funkcja `optymalizuj_podzial()`

Funkcja ta odpowiada za optymalizację podziału grafu, minimalizując liczbę przeciętych krawędzi. Jest to iteracyjny proces, w którym wierzchołki są przesuwane między częściami, jeśli zmniejsza to liczbę przecięć.

4.5 Funkcja `sprawdz_rownowage()`

Funkcja sprawdza, czy różnica w liczbie wierzchołków pomiędzy częściami nie przekracza zadanego marginesu procentowego. Jeśli różnica jest zbyt duża, algorytm ponownie optymalizuje podział.

4.6 Funkcja `zapisz_wyniki()`

Funkcja zapisuje wyniki podziału grafu do pliku w formacie tekstowym lub binarnym, w zależności od wyboru użytkownika.

5 Pseudokod

Poniżej zamieszczono pełny pseudokod algorytmu, który jest realizowany przez aplikację:

```
Funkcja podziel_graf(graf, liczba_czesci, margines_procentowy):
    liczba_wierzcholkow <- liczba_wierzcholkow(graf)
    liczba_wierzcholkow_na_czesc <- liczba_wierzcholkow / liczba_czesci
    L <- oblicz_macierz_Laplasjana(graf)
    wektory_wlasne <- oblicz_wektory_wlasne(L)
    wierzcholki_zredukowane <- wybierz_wektory(wektory_wlasne, liczba_czesci)
    podzial_poczatkowy <- k_srednich(wierzcholki_zredukowane, liczba_czesci)

    powtarzaj_dopóki:
        liczba_wierzcholkow_w_czesci <- [0, 0, ..., 0]
        dla i = 1 do liczba_wierzcholkow:
            czesc <- podzial_poczatkowy[i]
            liczba_wierzcholkow_w_czesci[czesc] <- liczba_wierzcholkow_w_czesci[

        maksymalna_roznica <- max(liczba_wierzcholkow_w_czesci) - min(liczba_wie
        jeśli maksymalna_roznica > (margines_procentowy * liczba_wierzcholkow):
            wykonaj_dodatkowa_optymalizacje_podzialu(podzial_poczatkowy, liczba_
        koniec jeśli
    dopóki maksymalna_roznica <= (margines_procentowy * liczba_wierzcholkow)

    liczba_przecieci <- oblicz_przeciecia_krawedzi(graf, podzial_poczatkowy)
    powtarzaj_dopóki:
        zmieniono <- fałsz
        dla i = 1 do liczba_wierzcholkow:
            czesc_stara <- podzial_poczatkowy[i]
            czesc_nowa <- znajdz_najlepsza_czesc_dla_wierzcholka(i, graf, podzia
            jeśli czesc_nowa != czesc_stara:
                podzial_poczatkowy[i] <- czesc_nowa
                zmieniono <- prawda

        liczba_przecieci_nowa <- oblicz_przeciecia_krawedzi(graf, podzial_poczat
        jeśli liczba_przecieci_nowa < liczba_przecieci:
            liczba_przecieci <- liczba_przecieci_nowa
        koniec jeśli
    dopóki zmieniono = prawda

    zwróć podzial_poczatkowy


Funkcja liczba_wierzcholkow(graf):
    zwróć długość(graf)
```

```
Funkcja oblicz_macierz_Laplasjana(graf):
    D <- oblicz_macierz_stopni(graf)
    A <- oblicz_macierz_sasiedztwa(graf)
    L <- D - A
    zwróć L

Funkcja oblicz_wektory_wlasne(L):
    wektory_wlasne, wartosci_wlasne <- oblicz_wartosci_i_wektory_wlasne(L)
    zwróć wektory_wlasne

Funkcja k_srednich(wierzcholki, k):
    inicjalizuj_centroidy <- losowe_wybieranie_centroidow(wierzcholki, k)
    powtarzaj_dopóki:
        przypisanie_czesci <- przypisz_wierzcholki_do_centroidow(wierzcholki, ce
        centroidy <- oblicz_nowe_centroidy(wierzcholki, przypisanie_czesci)
    dopóki zmiana_centroidow < próg_zbieżności
    zwróć przypisanie_czesci

Funkcja wykonaj_dodatkowa_optymalizacje_podzialu(podzial, liczba_wierzcholkow_na
    dla i = 1 do liczba_wierzcholkow:
        czesc <- podzial[i]
        jeśli liczba_wierzcholkow_w_czesci[czesc] > liczba_wierzcholkow_na_czesc
            czesc_nowa <- znajdz_najlepsza_czesc_dla_wierzcholka(i, graf, podzia
            podzial[i] <- czesc_nowa
    koniec dla

Funkcja oblicz_przeciecia_krawedzi(graf, podzial):
    liczba_przecieci <- 0
    dla i = 1 do liczba_wierzcholkow:
        dla sąsiad wierzcholka_i w grafie:
            jeśli podzial[i] != podzial[sąsiad]:
                liczba_przecieci <- liczba_przecieci + 1
    zwróć liczba_przecieci

Funkcja znajdz_najlepsza_czesc_dla_wierzcholka(i, graf, podzial):
    najlepsza_czesc <- podzial[i]
    minimalna_liczba_przecieci <- liczba_przecieci_dla_czesci(i, podzial)
    dla czesc = 1 do liczba_czesci:
        liczba_przecieci_nowa <- liczba_przecieci_dla_czesci_po_przeniesieniu(i,
        jeśli liczba_przecieci_nowa < minimalna_liczba_przecieci:
```

```
        najlepsza_czesc <- czesc
        minimalna_liczba_przecieci <- liczba_przecieci_nowa
koniec dla
zwróć najlepsza_czesc
```

6 Struktura programu

Program składa się z kilku plików źródłowych:

6.1 main.c

Plik `main.c` zawiera funkcję `main()` i jest odpowiedzialny za przetwarzanie argumentów wiersza poleceń, wywołanie odpowiednich funkcji i zarządzanie całym procesem podziału grafu.

6.2 graph.c

Plik `graph.c` zawiera funkcje związane z reprezentacją i manipulacją grafem, w tym wczytywanie grafu z pliku, podział grafu i optymalizację.

6.3 utils.c

Plik `utils.c` zawiera funkcje pomocnicze, takie jak obsługa argumentów wiersza poleceń, zapis do pliku oraz zarządzanie błędami.

6.4 partition.c

Plik `partition.c` zawiera algorytmy służące do realizacji podziału grafu i optymalizacji, w tym heurystyki minimalizacji przeciętych krawędzi.

7 Podsumowanie

Aplikacja do podziału grafu jest narzędziem do analizy dużych sieci i struktur grafowych. Dalsza optymalizacja algorytmu może obejmować implementację bardziej zaawansowanych metod optymalizacji, takich jak algorytmy genetyczne czy algorytmy przeszukiwania lokalnego.