Dokumentacja Implementacyjna - Aplikacja do Podziału Grafu

Gniewko Wasilewski, Jan Szulc

24 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 zadany 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 losowo lub heurystycznie, aby zapewnić równomierny rozkład wierzchołków.

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 `load_graph()`

Funkcja ta jest odpowiedzialna za wczytanie grafu z pliku `.csrrg`. Zawiera kod do parsowania danych wejściowych, takich jak liczba wierzchołków, lista sąsiedztwa i wskaźniki wierszy.

```
void load_graph(char *filename) {
    // Funkcja wczytuje graf z pliku i zapisuje go do struktury grafu
```

```
}
```

4.2 Funkcja `partition_graph()`

Funkcja ta realizuje podstawowy podział grafu na zadaną liczbę części. Początkowy podział jest wykonywany na podstawie prostych heurystyk, np. losowego przydzielania wierzchołków do części.

```
void partition_graph(int p) {
    // Funkcja dzieli graf na p części
}
```

4.3 Funkcja `optimize_partition()`

Funkcja ta optymalizuje podział grafu, starając się zmniejszyć liczbę przeciętych krawędzi. Używa algorytmu iteracyjnego, przesuwając wierzchołki pomiędzy częściami, jeśli zmniejsza to liczbę przecięć.

```
void optimize_partition() {
    // Funkcja optymalizuje podział grafu, minimalizując liczbę przeciętych kraw
}
```

4.4 Funkcja `check_balance()`

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

```
void check_balance() {
    // Funkcja sprawdza, czy liczba wierzchołków w częściach nie różni się zbytn
}
```

4.5 Funkcja `save_results()`

Funkcja zapisująca wyniki do pliku w formacie tekstowym lub binarnym, w zależności od wyboru użytkownika.

```
void save_results(char *filename, int binary_format) {
    // Funkcja zapisuje wyniki do pliku w formacie tekstowym lub binarnym
}
```

5 Struktura programu

Program składa się z kilku plików źródłowych:

5.1 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.

5.2 graph.c

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

5.3 utils.c

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

5.4 partition.c

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

6 Zakończenie

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.