Sprawozdanie nr 3, GIS

minimalizacja liczby central w grafie

Michał Pasieka
Igor Sawczuk

Realizacja projektu

Program został zaimplementowany i przetestowany. Jako, że zaproponowane przez nas rozwiązanie z użyciem solwera LP okazało się nieakceptowalne - zmodyfikowaliśmy algorytm projektu tak, by był on heurystyczny. Poza tym program został zaimplementowany zgodnie ze specyfikacją zawartą w sprawozdaniu nr 2.

Kod programu

Kod oraz historia zmian dostępna jest w serwisie github.com pod adresem https://github.com/igos/gis, gdzie jest publicznie dostępna. Ponadto do sprawozdania dostarczone jest archiwum ze spakowanymi źródłami oraz przykładowymi grafami.

Wykorzystanie zewnętrznych bibliotek

Podczas implementacji projektu korzystaliśmy m.in. z biblioteki JGrapht (https://github.com/jgrapht/jgrapht). Okazało się, że implementacja algorytmu Floyda-Warshalla była zaimplementowana niepoprawnie, nieprawidłowo obliczana jest długość ścieżki w grafach, gdzie krawędzie mają wagę różną od 1. Dokonaliśmy odpowiedniej poprawki (https://github.com/plprofetes/jgrapht/commit/7b97b6d936026ab3119fdc6b39b41308364e04ca) i zgłosiliśmy ją do autorów biblioteki JGrapht. Oczekujemy na odpowiedź od nich.
Ponadto wykorzystaliśmy importer plików GML z biblioteki Blueprints (https://github.com/tinkerpop/blueprints/wiki). Tu też odnaleźliśmy błąd, biblioteka nie wczytywała poprawnie grafu jeśli etykiety krawędzi lub wierzchołków były parsowalne na typy numeryczne. Aby zapobiec występowaniu tego problemu w odpowiedni sposób generujemy grafy.

Generator grafów

Aby umożliwić testowanie naszego rozwiązania w sposób wygodny - zaimplementowaliśmy generator grafów nieskierowanych, które następnie są zapisywane w formacie GML i mogą posłużyć jako dane wejściowe dla właściwego programu.
Generator przyjmuje następujące parametry wejściowe:

- -v <liczba wierzchołków>
- -p <prawdopodobieństwo wystąpienia krawędzi>
- -w <maksymalna waga krawędzi>
- -f <plik wyjściowy>, jeśli nie podano - wyniki wypisywane są na ekran.

Uruchomienie programu bez parametrów, bądź z nieprawidłowymi, prezentuje powyższe informacje.

Uruchomienie programu

Wygenerowanie dowolnego grafu do pliku GML:

```
JAVA –JAR GRAPHGENERATOR.JAR -V 1000 -P 0.03 -W 100 -F GENERATED-LAST.GML
```

Poszukiwanie central w wygenerowanym grafie:

```
JAVA –JAR GIS.JAR -R 20 -F GENERATED-LAST.GML -T -L 90 –X
```

Uruchomienie powyższych programów bez podania opcji, powoduje wyświetlenie wszystkich opcji wraz z opisem.

Przy analizie dużych grafów zalecamy użycie opcji -Xmx3584m

Wyniki testów poprawności i jakości algorytmów

Sposób testowania

Pamięć, profilowanie i samplowanie - jvisualvm z JDK 7.

Aspekt poprawności

Program posiada wbudowane mechanizmy sprawdzania poprawności rozwiązania. Zgodnie z tym, co napisano w sprawozdaniu 2, przeprowadzane jest sprawdzenie o poprawności heurystyki, tj. każdemu wierzchołkowi przyporządkowywana jest najbliższa centrala, a sprawdzany jest warunek, że da odległość jest nie większa niż zadane r. Jeśli wszystkie wierzchołki posiadają centralę w okolicy, bądź same są centralą - rozwiązanie jest uznawane za poprawne.

Odporność

Sytuacje opisane w sprawozdaniu 2 zostały obsłużone, tj.:

- plik wejściowy z niepoprawnie zapisanym grafem - program powinien zwrócić błąd.
- plik wejściowy z grafem pustym
- r mniejsze od wszystkich wartości krawędzi

Obsługa sytuacji awaryjnych

Sytuacje opisane w sprawozdaniu 2 zostały obsłużone, tj.::

- brak pamięci
- nieprawidłowy zbiór danych wejściowych
- nieprawidłowe parametry wejściowe

Ekstremalne testy obciążenia

Dla grafu o 2000 wierzchołków i o prawdopodobieństwie wystąpienia krawędzi = 0.3 (prawie 2 mln krawędzi) plik opisujący ten graf zajmuje ponad 40MB. W czasie obliczeń zużywa nieco ponad 2GB pamięci na stercie. Czas poszukiwania rozwiązania z wykorzystaniem procesora Intel Core i5 pierwszej generacji o taktowaniu 2.4GHz to nieco ponad 40 minut.
Dla takiej samej konfiguracji komputera wczytanie 5000 wierzchołków i 12.5 mln krawędzi przepełniało stertę o rozmiarze 4GB. Sam plik opisujący graf to ponad 200MB.

Aspekt jakości

Zaimplementowany heurystyczny algorytm pokrycia nie daje wyników optymalnych. Jednak dla małych grafów wykonano ręczne sprawdzenie, które pozwoliło stwierdzić, że uzyskane wyniki są wystarczającymi rozwiązaniami.

Podsumowanie

Podczas realizacji projektu odkryliśmy błędy w dostępnych bibliotekach operacji na grafach oraz poczyniliśmy odpowiednie kroki by wcielić nasze poprawki.

Projekt nie jest zoptymalizowany pod kątem wykorzystania zasobów oraz procesora. Jednym z rozwiązań było by użycie wielu wątków przy algorytmie Floyda-Warshalla oraz heurystycznym algorytmie pokrycia. Podejrzewamy, że stworzenie własnego importera GML pozwoliłoby również na zmniejszenie zużycia pamięci o 10%. Projekt jest dostępny otwarcie na GitHub https://github.com/igos/gis/ , umożliwiając jego dalszy rozwój przez inne osoby.

Podczas projektu odkryliśmy również, że specyfikacja GML narzuca wiele ograniczeń, na przykład wagę krawędzi musieliśmy przechowywać jako label.

Niektóre wyniki dla małych grafów zostały sprawdzone ręcznie, potwierdzając tym samym poprawność działania programu i zawartych w nim algorytmów.