SK6. Atak na sieć (II)

Raport wstępny do projektu w ramach kursu "Grafy i Sieci" (GIS)

Patryk Kocielnik, Jan Kumor, 5.04.2018r.

Opiekun projektu

dr inż. Sebastian Kozłowski

Opis zadania

Dane są dwie sieci: euklidesowa i losowa (ER) o mniej więcej takiej samej liczbie wierzchołków i krawędzi. Porównać prawdopodobieństwa powodzenia ataku na losowe krawędzie tych sieci (udany atak to taki, który prowadzi do rozspójnienia sieci).

Planowane wykorzystanie narzędzi w projekcie

- Środowisko rozwiązania: stacja robocza pod kontrolą systemu GNU/Linux,
- Język implementacji rozwiązania: Python,
- Narzędzia do analizy i wizualizacji grafów:
  - *igraph* - bilbioteka języka C [1],
  - *python-igraph* interfejs programistyczny biblioteki *igraph* dla języka Python [2],

Składniki rozwiązania

1. Moduł generacji sieci: euklidesowych oraz losowych, o zadanej liczbie wierzchołków:
   - Sposób wywołania: `graph = generate_graph(graph_type, vertex_probability)`,
   - Rezultat wywołania: `graph` jako dwuwymiarowa macierz sąsiedztwa (`int * int`) opisująca wygenerowany graf,
   - Podstawą komponentu będą moduły generowania sieci z pakietu *igraph* - funkcje `Erdos_Renyi` (dla grafów losowych) oraz `GRG` (dla grafów euklidesowych).
2. Filtr usuwający z grafu losowo wybraną krawędź:

- Sposób wywołania: `new_graph = break(graph)`.
- Rezultat wywołania: `new_graph` jako graf pozbawiony losowo wybranej krawędzi.

3. Analizator spójności sieci:
   - Sposób wywołania: `is_connected(graph)`
   - Rezultat wywołania: c (`bool`) flaga przyjmująca wartość `True` jeśli graf jest spójny, przeciwnie - `False`.
4. Moduł analizy ataku na zadany graf:
   - Sposób wywołania: `analyse_attack(graph)`,
   - Rezultat wywołania: p (`float`) jako prawdopodobieństwo powodzenia ataku na losowo wybraną krawędź zadanego grafu wejściowego,
   - Moduł przeprowadza próby ataku na każdą z krawędzi grafu i na podstawie ich wyników oblicza prawdopodobieństwo powodzenia.

Interfejs aplikacji

Interfejsem aplikacji będzie konsola tekstowa. Motywacją tego podejścia jest łatwość łączenia aplikacji z interfejsem tekstowym w filtry, które później wykorzystać można do analizy bardziej złożonych struktur.

Przebieg eksperymentu

Liczbę iteracji k ustal na wartość z przedziału od 1 do 25. Liczbę wierzchołków v ustal na należącą do zbioru V_num: 10, 100, 1000, 10000, 100000, Liczbę krawędzi ustal na należącą do zbioru E_num: 10, 100, 1000, 10000, 100000.

1. Powtórz dla k przypadków:
2. Wygeneruj graf o zadanym typie, liczbie wierzchołków `v` i liczbie krawędzi `e`,
3. Usuń z grafu losowo wybraną krawędź,
4. Sprawdź, czy nastąpiło rozspójnienie grafu.
5. Oblicz iloczyn: $rozspójnień/ataków$

Generowanie grafów losowych

Generowanie grafu losowego będzie podzielone na dwa etapy.

Pierwszym etapem będzie przyjęcie zadanej liczby wierzchołków oraz zadanej gęstości grafu i obliczenie z nich docelowej liczby krawędzi $q_target$dla grafu wyjściowego. Drugi etap polegał będzie na wygenerowaniu grafu o n wierzchołkach połączonych losowo q_{target} krawędziami.

Algorytm ten przyjmuje dwa argumenty: liczbę wierzchołków n oraz współczynnik prawdopodobieństwa wystąpienia krawędzi n.

Grafy losowe ER (model Erdősa–Rényi) zostaną wygenerowane z użyciem funkcji `Erdos_Renyi` klasy `Grah` pakietu *igraph*. Metoda ta przyjmuje jako parametry:

- liczbę wierzchołków grafu n
- prawdopodobieństwo wystąpienia danej krawędzi p lub zadaną liczbę krawędzi m.

Zgodnie z dokumentacją pakietu *igraph* algorytm wykorzystywany w metodzie `Erdos_Renyi` ma złożoność obliczeniową równą $O(|V| + |E|)$

Grafy euklidesowe zostaną wygenerowane z wykorzystaniem funkcji `GRG` klasy `Graph` z pakietu *igraph*. Metoda ta przyjmuje jako parametry liczbę wierzchołków grafu n oraz promień r. Algorytm generacji grafu euklidesowego o n wierzchołkach:

1. Rozmieść n wierzchołków w kwadracie jednostkowym,
2. Połącz krawędziami te wierzchołki, które znajdują się od siebie w odległości mniejszej niż zadany promień r.

Zgodnie z dokumentacją pakietu *igraph* algorytm wykorzystywany w metodzie `GRG` ma złożoność obliczeniową nie większą niż $O(|V|^2 + |E|)$.

W celu uzyskania grafu o zadanej przybliżonej liczbie krawędzi m zostanie wykorzystane następujące podejście iteracyjne:

1. Dla wybranej wartości promienia r wygeneruj graf euklidesowy o n wierzchołkach z użyciem funkcji `GRG`,
2. Jeśli liczba krawędzi wygenerowanego grafu jest:
   - znacząco mniejsza od zadanej liczby krawędzi, zwiększ wartość promienia r,
   - znacząco większa od zadanej liczby krawędzi, zmniejsz wartość promienia r,
   - w przyliżeniu równa zadanej liczbie krawędzi, zwróć wygenerowany graf i zakończ algorytm,
3. Wróc do punktu 1.

Liczba krawędzi jest w przybliżeniu równa zadanej liczbie krawędzi gdy: $\frac{|m_{zad}-m|}{m_{zad}} < \varepsilon$. Gdzie ε jest parametrem kontrolującym dokładność przybliżenia.

Weryfikacja spójności grafu

Do weryfikacji spójności grafów zostanie wykorzystana metoda `is_connected` klasy `GraphBase` z pakietu *python-igraph*. Opiera się ona na algorytmie przeszukiwania grafu wgłąb (ang. *depth-first search*, DFS). Pseudokod algorytmu DFS [2], przedstawiony został poniżej :

1. Utwóż tablicę `visited` o `n` elementach,
2. Tablicę `visited` wypełnij wartościami `false`,
3. Utwórz pusty stos `S`,

4. Inicjuj licznik odwiedzonych wierzchołków,
5. Rozpocznij przejście DFS od wierzchołka 0,
6. Wierzchołek oznacz jako odwiedzony,
7. Przechodź przez graf dopóki stos `S` nie jest pusty, wykonując następujące kroki:
   - Pobierz wierzchołek ze stosu,
   - Pobrany wierzchołek usuń ze stosu,
   - Zwiększ licznik odwiedzonych wierzchołków,
   - Przejrzyj kolejnych sąsiadów,
     - Szukaj do sąsiadów jeszcze nie odwiedzonych,
     - Odznacz sąsiada jeśli jeszcze nie odwiedzony,
     - Umieść sąsiada na stosie. Jeśli wszystkie wierzchołki zostały odwiedzone, graf jest spójny. W przeciwnym wypadku, graf jest niespójny.

Złożoność czasowa algorytmu wynosi $O(E + V)$.

Schemat testów

Testy zostaną przeprowadzone w następujący sposób:

1. Z wcześniej zdefiniowanej listy zostanie przyjęty zestaw parametrów testu,
2. Wygenerowany zostanie zestaw k grafów testowych o przyjętych wcześniej parametrach,
3. Dla każdego z grafów:
   1. Dla każdej z krawędzi badanego grafu:
      1. Krawędź ta zostanie usunięta z grafu (zostanie przeprowadzony atak na tę krawędź),
      2. Sprawdzona zostanie spójność grafu po przeprowadzeniu ataku,
      3. Jeśli graf nie jest spójny atak zakończył się powodzeniem,
   2. Obliczone zostanie prawdopodobieństwo powodzenia ataku na losowo wybraną krawędź z badanego grafu, zgodnie ze wzorem: $p_i = \frac{n_{sukces}}{m}$
4. Zgodnie ze wzorem: $p_{sr} = \frac{\sum_{i=0}^{k} p_i}{k}$, zostanie obliczone średnie prawdopodobieństwo powodzenia ataku dla zestawu k grafów testowych o przyjętych parametrach.
5. Jeśli pozostały nieprzetestowane zestawy parametrów, nastąpi powrót do punktu 1.

Model danych

Projektowane narzędzie wykorzystywać będzie implementacje grafów nieskierowanych z biblioteki *igraph*.

Graf we wspomnianej implementacji jest reprezentowany jako wielozbiór krawędzi oraz metadane. Najważniejszymi polami zawartymi w metadanych są:

- liczba wierzchołków grafu,
- określenie czy graf jest skierowany czy nie.

Każda z krawędzi grafu nieskierowanego jest modelowana jako nieuporządkowana para (dwuelementowy zbiór) etykiet oznaczających wierzchołki grafu. Krawędzie są etykietowane, a etykiety przyjmują wartości od 0 do $|E| - 1$. Etykiety wierzchołków przyjmują wartości od 0 do $|V| - 1$.

Przykładową, uproszczoną (pominięto etykiety krawędzi) strukturę grafu nieskierowanego przedstawiono poniżej:

```
( wierzchołki: 6,
  skierowany: nie,
  krawędzie:
   {
    {0,2},
    {2},
    {2,3},
    {3},
    {3,4},
    {3,4},
    {4,1}
   }
 )
```

Należy nadmienić, iż implementacja ta dopuszcza istnienie w grafie pętli. Jednak w projektowanym narzędziu grafy takie nie będą rozpatrywane.

Testy poprawności rozwiązania

Poprawność zaproponowanego rozwiązania zostanie sprawdzona na prostych grafach o niewielkiej liczbie wierzchołków i krawędzi, a następnie na grafach o liczbie wierzchołków sięgającej 200, 1000, 2000 oraz 4000. Dla niewielkich grafów łatwo można dokonać dokładnej analizy i obliczyć dla nich prawdopodobieństwa powodzenia ataku na losowo wybraną krawędź. Wielkoskalowe grafy dają nadzieję na większą stabilność wyników.

Proponowane małe grafy testowe wraz z oczekiwanymi wynikami to:

- 2 wierzchołki, 1 krawędź - oczekiwane prawdopodobieństwo powodzenia ataku 1.0,
- 3 wierzchołki, 2 krawędzie - oczekiwane prawdopodobieństwo powodzenia ataku 1.0,
- 3 wierzchołki, 3 krawędzie - oczekiwane prawdopodobieństwo powodzenia ataku 0.0,

- 4 wierzchołki, 3 krawędzie - oczekiwane prawdopodobieństwo powodzenia ataku 0.8,
- 4 wierzchołki, 4 krawędzie - oczekiwane prawdopodobieństwo powodzenia ataku 0.2,
- 4 wierzchołki, 5 krawędzi - oczekiwane prawdopodobieństwo powodzenia ataku 0.0.