Znajdowanie gęstych podgrafów w sieciach społecznych oraz inne algorytmy grafowe

Paweł Kacprzak

Instytut Informatyki Uniwersytetu Wrocławskiego

5 czerwca 2012

Krótki plan

1. Wstęp
2. Definicja problemu
3. Algorytm (dość ogólnie)
4. Szczegóły
5. Inne algorytmy
6. Podsumowanie

Dlaczego taki temat

Dlaczego taki temat

- Fajnie byłoby wiedzieć coś więcej o obliczeniach grafowych w MapReduce poza klasycznym `PageRank` i naiwnym `BFS`

Dlaczego taki temat

- Fajnie byłoby wiedzieć coś więcej o obliczeniach grafowych w MapReduce poza klasycznym `PageRank` i naiwnym `BFS`
- Trzeba myśleć inaczej niż w sekwencyjnych algorytmach grafowych

Dlaczego taki temat

- Fajnie byłoby wiedzieć coś więcej o obliczeniach grafowych w MapReduce poza klasycznym `PageRank` i naiwnym `BFS`
- Trzeba myśleć inaczej niż w sekwencyjnych algorytmach grafowych
- Wykorzystuje się raczej proste, kilkufazowe procedury i składa się je w całość obliczeń

Założenia

Założenia

- Fazy Map i Reduce są nierozłączne

Założenia

- Fazy Map i Reduce są nierozłączne
- Struktura grafu to lista krawędzi

Założenia

- Fazy Map i Reduce są nierozłączne
- Struktura grafu to lista krawędzi
- $deg(G)$ nie jest zbyt duży

Definicja problemu

Na przykładzie sieci mailowej

Na przykładzie sieci mailowej

- Dostajemy log serwera mailowego. Wpis (s, r, m) oznacza, że s wysyłał do r wiadomość m

Na przykładzie sieci mailowej

- Dostajemy log serwera mailowego. Wpis (s, r, m) oznacza, że s wysyłał do r wiadomość m
- Myślimy o tym jak o grafie

Na przykładzie sieci mailowej

- Dostajemy log serwera mailowego. Wpis (s, r, m) oznacza, że s wysyłał do r wiadomość m
- Myślimy o tym jak o grafie
- Dla ustalonego **k**, chcielibyśmy znaleźć takie grupy osób, że osoby z każdej grupy wysyłały do siebie co najmniej k maili

Na przykładzie sieci mailowej

- Dostajemy log serwera mailowego. Wpis (s, r, m) oznacza, że s wysyłał do r wiadomość m
- Myślimy o tym jak o grafie
- Dla ustalonego **k**, chcielibyśmy znaleźć takie grupy osób, że osoby z każdej grupy wysyłały do siebie co najmniej k maili
- Wygląda jak szukanie kliki?

Na przykładzie sieci mailowej

- Dostajemy log serwera mailowego. Wpis (s, r, m) oznacza, że s wysyłał do r wiadomość m
- Myślimy o tym jak o grafie
- Dla ustalonego **k**, chcielibyśmy znaleźć takie grupy osób, że osoby z każdej grupy wysyłały do siebie co najmniej k maili
- Wygląda jak szukanie kliki?
- Oczywiście rozwiązuje się to w sposób przybliżony

Algorytm wysokiego poziomu

Algorytm wysokiego poziomu

1. Lista logów $\longrightarrow$ graf skierowany z wagami

Algorytm wysokiego poziomu

1. Lista logów $\longrightarrow$ graf skierowany z wagami
2. Graf skierowany z wagami $\longrightarrow$ graf (zachowujemy tylko znaczące krawędzie)

Algorytm wysokiego poziomu

1. Lista logów $\longrightarrow$ graf skierowany z wagami
2. Graf skierowany z wagami $\longrightarrow$ graf (zachowujemy tylko znaczące krawędzie)
3. Wzbogacamy strukture grafu o stopnie wierzcholków

Algorytm wysokiego poziomu

1. Lista logów $\longrightarrow$ graf skierowany z wagami
2. Graf skierowany z wagami $\longrightarrow$ graf (zachowujemy tylko znaczące krawędzie)
3. Wzbogacamy strukture grafu o stopnie wierzcholków
4. Szukamy trójkątów w grafie (będzie pomocne dalej)

Algorytm wysokiego poziomu

1. Lista logów $\longrightarrow$ graf skierowany z wagami
2. Graf skierowany z wagami $\longrightarrow$ graf (zachowujemy tylko znaczące krawędzie)
3. Wzbogacamy strukture grafu o stopnie wierzcholków
4. Szukamy trójkątów w grafie (będzie pomocne dalej)
5. Zamiast szukać k-klik, szukamy k-trusses (definicja)

Szczegóły algorytmów

Proste - word count

Nic więcej

Też łatwe

Map

- Ignorujemy krawędzie, które mają wagę $< k$
- Dla każdej krawędzi (v_1, v_2, w), gdzie $w \geqslant k$, emitujemy parę $(min(v_1, v_2), max(v_1, v_2))$

Też łatwe

Map

- Ignorujemy krawędzie, które mają wagę $< k$
- Dla każdej krawędzi (v_1, v_2, w), gdzie $w \geqslant k$, emitujemy parę $(min(v_1, v_2), max(v_1, v_2))$

Reduce

- Zliczamy krawędzie o tym samym kluczu (są takie co najwyżej 2), jeśli są **dokładnie** 2 to emitujemy tą krawędź. W.p.p. ignorujemy.

Co chcemy osiągnąć?

Co chcemy osiągnąć?

Dla każdej krawędzi wejściowej (v, w) chcemy mieć krawędź postaci $(v, w, deg(v), deg(w))$

Co chcemy osiągnąć?

Dla każdej krawędzi wejściowej (v, w) chcemy mieć krawędź postaci $(v, w, deg(v), deg(w))$

Jak to zrobić?

Co chcemy osiągnąć?

Dla każdej krawędzi wejściowej (v, w) chcemy mieć krawędź postaci $(v, w, deg(v), deg(w))$

Jak to zrobić?

Będą potrzebne 2 fazy (drugi map bedzie identycznoscią)

Faza 1

Map

- Dla każdej krawędzi (v, w) emitujemy 2 rekordy (kluczem jest wierzchołek)
  1. $v, (v, w)$
  2. $w, (v, w)$
- Do jednego reducera trafią wszystkie wszystkie krawędzie incydentne z wierzchołkiem v

Faza 1

Map

- Dla każdej krawędzi (v, w) emitujemy 2 rekordy (kluczem jest wierzchołek)
  1. $v, (v, w)$
  2. $w, (v, w)$
- Do jednego reducera trafią wszystkie wszystkie krawędzie incydentne z wierzchołkiem v

Reduce

- Dla każdego rekordu o kluczu v, zliczamy najpierw stopień v, a później dla każdej krawędzi, z którą przyszedł, emitujemy $((v, w), deg(v), null)$ lub $((w, v), null, deg(v))$, gdzie kluczem jest krawędź

Faza 2

Map

- Identyczność

Faza 2

Map

- Identyczność

Reduce

- Reducer otrzyma dwa rekordy pod tym samym kluczem (v, w). W jednym będzie $deg(v)$, w drugim $deg(w)$. Wystarczy je złączyć ze sobą i wyemitować jedna

Triada

Triada

- Co to jest triada?

Triada

- Co to jest triada?
- Jak szukać trójkątów przy pomocy triad?

Triada

- Co to jest triada?
- Jak szukać trójkątów przy pomocy triad?
- Nie trzeba znajdować wszystkich triad, wystarczy **jedna** dla każdego trójkąta - o tym zaraz

Jedna triada = jeden trójkąt

Jedna triada = jeden trójkąt

- Mamy porządek na wierzchołkach (na razie załóżmy, ze to porządek po nazwie)

Jedna triada = jeden trójkąt

- Mamy porządek na wierzchołkach (na razie załóżmy, ze to porządek po nazwie)
- Mapujemy każdą krawędź na jej **mniejszy** wierzchołek

Jedna triada = jeden trójkąt

- Mamy porządek na wierzchołkach (na razie załóżmy, ze to porządek po nazwie)
- Mapujemy każdą krawędź na jej **mniejszy** wierzchołek
- Wtedy w każdym trójkącie istnieje **dokładnie** jeden wierzchołek, który dostanie 2 krawędzie, nazwijmy go **apexem**

Jedna triada = jeden trójkąt

- Mamy porządek na wierzchołkach (na razie załóżmy, ze to porządek po nazwie)
- Mapujemy każdą krawędź na jej **mniejszy** wierzchołek
- Wtedy w każdym trójkącie istnieje **dokładnie** jeden wierzchołek, który dostanie 2 krawędzie, nazwijmy go **apexem**
- Możemy już zapisać algorytm

Faza 1

Map

- Mapujemy każdą krawędź na jej mniejszy wierzchołek

Faza 1

Map

- Mapujemy każdą krawędź na jej mniejszy wierzchołek

Reduce

- Dostajemy wszystkie rekordy postaci (v, e) dla każdej e incydentnej z v.
- Wypluwamy wszystkie pary takich krawędzi (tworzą one triady)
- Kluczem jest para zewnętrznych wierzchołków triady.

Faza 2

Map

- 2 rodzaje plików wejściowych
  1. Wyjście Reduce1
  2. Oryginalny graf
- Identyczność (kluczem obu jest krawędź)

Faza 2

Map

- 2 rodzaje plików wejściowych
  1. Wyjście Reduce1
  2. Oryginalny graf
- Identyczność (kluczem obu jest krawędź)

Reduce

- Dostajemy wszystkie rekordy o kluczu e. Jeden z tych rekordów będzie z oryginalnego grafu - jest to potencjalna krawędź zamykająca każdą z triad, który są zmapowane na e.
- Emitujemy wszystkie trójkąty (kluczem są wszystkie 3 krawędzie w ustalonym porządku)

Może pojawić się problem

Może pojawić się problem

- Znajdowanie wszystkich par w Reduce1 może być kosztowne

Może pojawić się problem

- Znajdowanie wszystkich par w Reduce1 może być kosztowne
- Uniknąć dużej liczby par się nie da **(klika)**, ale można to poprawić

Ulepszenie

Ulepszenie

- Heurystyka: Zamiast porządku na wierzchołkach po nazwach, wybieramy porządek **(deg, name)**

Ulepszenie

- Heurystyka: Zamiast porządku na wierzchołkach po nazwach, wybieramy porządek **(deg, name)**
- Obserwacja: Map1 może nie emitować, jeśli $deg(v) = 1$

Co to jest?

K-truss

- To maksymalny podgraf, w którym każda krawędź jest w co najmniej $k-2$ trójkątach

Przykład

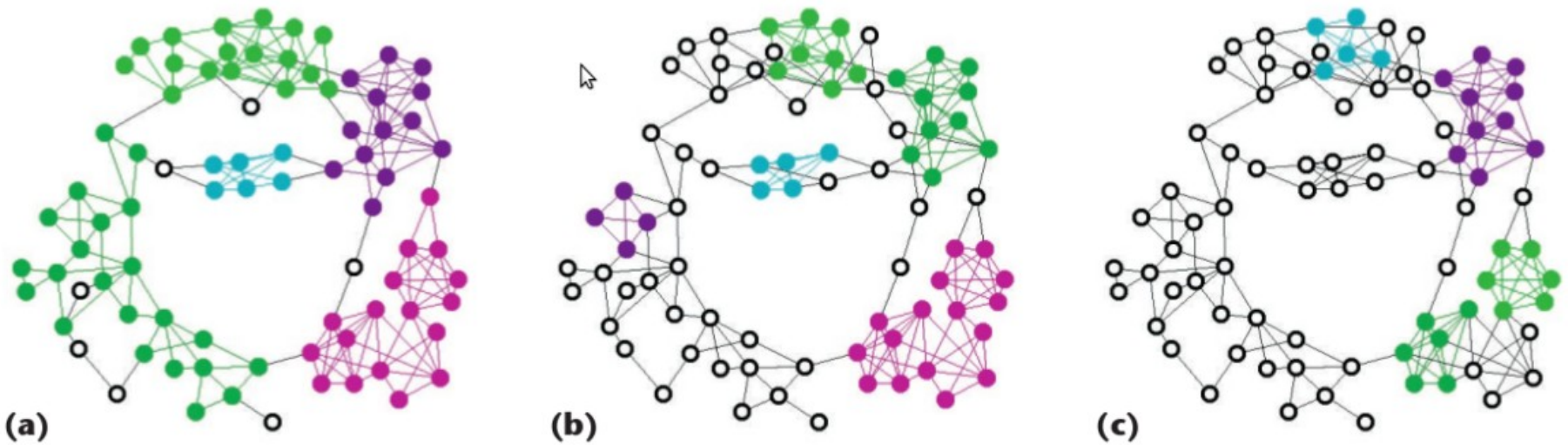

Figure 8. Trusses of a graph. Each truss has a randomly assigned color: (a) 3-trusses, (b) 4-trusses, and (c) 5-trusses. Vertices and edges not in trusses are black; such vertices are also hollow.

K-truss

Co to daje?

K-truss

Co to daje?

- Dobrze symulują kliki, ale są łatwe obliczeniowo

K-truss

Co to daje?

- Dobrze symulują kliki, ale są łatwe obliczeniowo
- Kliki występują bardzo rzadko w sieciach społecznościowych

Idea algorytmu

Idea algorytmu

- Szukamy trójkątów i zachowujemy tylko te krawędzie, które należą do co najmniej $k-2$ trójkątów

Idea algorytmu

- Szukamy trójkątów i zachowujemy tylko te krawędzie, które należą do co najmniej $k-2$ trójkątów
- Problematyczne jest to, że usuwając krawędzie, które są w mniej niż $k-2$ trójkątach, możemy zabrać trójkąty "wspierające"dla innych

Algorytm

Algorytm

1. Wzbogacamy krawędzie o stopnie

Algorytm

1. Wzbogacamy krawędzie o stopnie
2. Wyszukujemy trójkąty

Algorytm

1. Wzbogacamy krawędzie o stopnie
2. Wyszukujemy trójkąty
3. Dla każdej krawędzi liczmy w ilu trójkątach się zawiera

Algorytm

1. Wzbogacamy krawędzie o stopnie
2. Wyszukujemy trójkąty
3. Dla każdej krawędzi liczmy w ilu trójkątach się zawiera
4. Emitujemy tylko te, które są w $\geqslant k-2$

Algorytm

1. Wzbogacamy krawędzie o stopnie
2. Wyszukujemy trójkąty
3. Dla każdej krawędzi liczmy w ilu trójkątach się zawiera
4. Emitujemy tylko te, które są w $\geqslant k-2$
5. Jeśli krok 4 odrzucił jakieś krawędzie, wróc do kroku 1

Algorytm

1. Wzbogacamy krawędzie o stopnie
2. Wyszukujemy trójkąty
3. Dla każdej krawędzi liczmy w ilu trójkątach się zawiera
4. Emitujemy tylko te, które są w $\geqslant k-2$
5. Jeśli krok 4 odrzucił jakieś krawędzie, wróc do kroku 1
6. Znajdź spójne składowe w grafie, każda z nich jest **k-truss**

Inne algorytmy

Spójne składowe

Naiwne podejście

Naiwne podejście

- Używamy `BFSa` podobnego jak pisaliśmy na pracownii

Naiwne podejście

- Używamy `BFSa` podobnego jak pisaliśmy na pracownii
- Jeśli graf ma > 1 składowych, to trzeba uruchomić kilka razy, za każdym razem ze źródła, które jest poza znalezionymi składowymi

Naiwne podejście

- Używamy `BFSa` podobnego jak pisaliśmy na pracownii
- Jeśli graf ma > 1 składowych, to trzeba uruchomić kilka razy, za każdym razem ze źródła, które jest poza znalezionymi składowymi
- Nawet gdy G spójny, wymaga $diam(G)$ iteracji

Lepsza metoda Zones

Idea

Lepsza metoda Zones

Idea

- Budujemy wiele składowych na raz. Każda składowa ma wierzchołek centralny. Jeśli dwie składowe mogą być połączone, to ta, której centrum jest mniejsze, wchłania tą drugą i ustawia jej centrum na swoje.

Lepsza metoda Zones

Idea

- Budujemy wiele składowych na raz. Każda składowa ma wierzchołek centralny. Jeśli dwie składowe mogą być połączone, to ta, której centrum jest mniejsze, wchłania tą drugą i ustawia jej centrum na swoje.

- Na wejściu mamy 2 pliki:

Lepsza metoda Zones

Idea

- Budujemy wiele składowych na raz. Każda składowa ma wierzchołek centralny. Jeśli dwie składowe mogą być połączone, to ta, której centrum jest mniejsze, wchłania tą drugą i ustawia jej centrum na swoje.

- Na wejściu mamy 2 pliki:
  1. Edges postaci **key:** (v, w)**, value: dowolna**

Lepsza metoda Zones

Idea

- Budujemy wiele składowych na raz. Każda składowa ma wierzchołek centralny. Jeśli dwie składowe mogą być połączone, to ta, której centrum jest mniejsze, wchłania tą drugą i ustawia jej centrum na swoje.

- Na wejściu mamy 2 pliki:
  1. Edges postaci **key:** (v, w)**, value: dowolna**
  2. Zones postaci **key:** v**, value:** z, gdzie z jest strefą v.

Lepsza metoda Zones

Idea

- Budujemy wiele składowych na raz. Każda składowa ma wierzchołek centralny. Jeśli dwie składowe mogą być połączone, to ta, której centrum jest mniejsze, wchłania tą drugą i ustawia jej centrum na swoje.

- Na wejściu mamy 2 pliki:
  1. Edges postaci **key:** (v, w)**, value: dowolna**
  2. Zones postaci **key:** v**, value:** z, gdzie z jest strefą v.
- Na początku plik zones to (v, v)

Lepsza metoda Zones

Idea

- Budujemy wiele składowych na raz. Każda składowa ma wierzchołek centralny. Jeśli dwie składowe mogą być połączone, to ta, której centrum jest mniejsze, wchłania tą drugą i ustawia jej centrum na swoje.

- Na wejściu mamy 2 pliki:
  1. Edges postaci **key:** (v, w)**, value: dowolna**
  2. Zones postaci **key:** v**, value:** z, gdzie z jest strefą v.
- Na początku plik zones to (v, v)
- Algorytm będzie miał 3 fazy MapReduce

Algorytm - faza 1

Map

- Dla każdej krawędzi (v, w) emituj $(v, (v, w))$ oraz $(w, (v, w))$. Dla strefy (v, z) identyczność

Algorytm - faza 1

Map

- Dla każdej krawędzi (v, w) emituj $(v, (v, w))$ oraz $(w, (v, w))$. Dla strefy (v, z) identyczność

Reduce

- Mamy zbindowane na v wszystkie krawędzie e z nim incydentne oraz jego strefe z
- Dla każdej krawędzi emituj (e, z)

Algorytm - faza 2

Map

- Identyczność

Algorytm - faza 2

Map

- Identyczność

Reduce

- Mamy zbindowane do e wszystkie strefy, do której należą jej końce - co najwyżej 2 strefy
- Niech Z to zbiór stref zbindowanych do e
- Jeśli $|Z| = 1$ to pomijamy ten kubełek - to krawędź wewnątrz strefy
- Niech $z_m = min(Z)$
- Dla każdego $z \in Z \setminus \{z_m\}$ emitujemy (z, z_m) - para (stara strefa, nowa strefa)

Algorytm - faza 3

Map

- Na wejściu 2 pliki
  1. stary Zone File
  2. wynik z Fazy 2 - pary (z, z_m)
- Dla rekordu z Zone File (v, z) emituj (z, v)
- Dla rekordu z Fazy 2 emituj (z, z_m)

Algorytm - faza 3

Map

- Na wejściu 2 pliki
  1. stary Zone File
  2. wynik z Fazy 2 - pary (z, z_m)
- Dla rekordu z Zone File (v, z) emituj (z, v)
- Dla rekordu z Fazy 2 emituj (z, z_m)

Reduce

- Dostaje zbindowane na z zbiór wierzchołków V, które mają dotyczczas strefe z oraz zbiór Z nowych stref dla z
- Niech $m = min(Z)$
- Dla każdego $v \in V$ emituj (v, m) - nowa strefa dla v
- Mamy nowe strefy

Spójne składowe - podsumowanie

Spójne składowe - podsumowanie

- Kiedy się zatrzymać?

Spójne składowe - podsumowanie

- Kiedy się zatrzymać?
- W fazie 2 możemy sprawdzać czy strefy będą jeszcze zmieniane

Bonus: Wyszukiwanie czworokątów

Idea algorytmu

Idea algorytmu

- Podobnie jak przy trójkątach

Idea algorytmu

- Podobnie jak przy trójkątach
- Będziemy chcieli łączyć triady ze sobą

Idea algorytmu

- Podobnie jak przy trójkątach
- Będziemy chcieli łączyć triady ze sobą
- Przy ustalonym porządku wierzchołków jest możliwych **więcej niż** 1 **para** triad tworząca czworokąt

Idea algorytmu

- Podobnie jak przy trójkątach
- Będziemy chcieli łączyć triady ze sobą
- Przy ustalonym porządku wierzchołków jest możliwych **więcej niż** 1 **para** triad tworząca czworokąt
- Liczymy ile ich jest $\longrightarrow$ wychodzi 3

Idea algorytmu

- Podobnie jak przy trójkątach
- Będziemy chcieli łączyć triady ze sobą
- Przy ustalonym porządku wierzchołków jest możliwych **więcej niż** 1 **para** triad tworząca czworokąt
- Liczymy ile ich jest $\longrightarrow$ wychodzi 3
- Będziemy chcieli parować triady, aby uzyskać czworokąty

Triady

Triady

- Low, high, mixed triads

Triady

- Low, high, mixed triads
- Dlaczego wysokie triady są **złe**?

Algorytm - Faza 1

Map

- Dla każdej krawędzi (v, w) emituj $(v, (v, w))$ oraz $(w, (v, w))$ zaznaczając czy jest to większy (high) czy mniejszy (low) wierzchołek

Algorytm - Faza 1

Map

- Dla każdej krawędzi (v, w) emituj $(v, (v, w))$ oraz $(w, (v, w))$ zaznaczając czy jest to większy (high) czy mniejszy (low) wierzchołek

Reduce

- Dostajemy wszystkie krawędzie incydentne z v
- Parujemy każde 2 krawędzie typu low - tworzą **triadę low**
- Parujemy każdą krawędź typu low z każdą krawędzią typu high - tworzą **triadę mixed**
- Emitujemy krawędzie połączone w triadę, gdzie kluczem są zewnętrzene wierzchołki triady w porządku leksykograficznym

Algorytm - Faza 2

Map

- Identyczność

Algorytm - Faza 2

Map

- Identyczność

Reduce

- Dostajemy triady zmapowane do tej samej pary zewnętrzych wierzchołków
- Dla każdej pary triad emitujemy ją jako czworokąt

Przemyślenia

Przemyślenia

- Każdy z tych algorytmów można łatwo przerobić tak aby działał na hipergrafach (krawędź łączy dowolną liczbę wierzchołków) bez dodatkowych faz MapReduce

Przemyślenia

- Każdy z tych algorytmów można łatwo przerobić tak aby działał na hipergrafach (krawędź łączy dowolną liczbę wierzchołków) bez dodatkowych faz MapReduce
- W zaprezentowanych algorytmach, niektóre z operacji były robione w fazie Reduce, a następujący po niej Map był identycznością. Trzeba rozważyć z techinicznego punktu widzenia implementacji Hadoopa, gdzie lepiej wykonywać te operacje

Przemyślenia

- Każdy z tych algorytmów można łatwo przerobić tak aby działał na hipergrafach (krawędź łączy dowolną liczbę wierzchołków) bez dodatkowych faz MapReduce
- W zaprezentowanych algorytmach, niektóre z operacji były robione w fazie Reduce, a następujący po niej Map był identycznością. Trzeba rozważyć z techinicznego punktu widzenia implementacji Hadoopa, gdzie lepiej wykonywać te operacje
- Intuicja jest taka, że każdy algorytm wymagający `DFS`, np. szukanie mostów w grafie, będzie miał słabą implementacje w modelu MapReduce

Dziękuję za uwagę.