Grafy i Sieci - Projekt

Aproksymacyjne wykrywanie motywów w sieciach

Łukasz Brzozowski, Jakub Janaszkiewicz, Kacper Siemaszko

29 maja 2020

Spis treści

1 Opis użytych algorytmów i modyfikacje

Działanie algorytmów, którymi zajmowaliśmy się w projekcie opisaliśmy już w dokumentacji wstępnej. W celu przygotowania finalnego rozwiązania, skorzystaliśmy z istniejących implementacji, które zmodyfikowaliśmy pod własne potrzeby.

1.1 Algorytm MODA

Jako implementację algorytmu MODA, wybraliśmy wersję ParaMODA[1][2] pod licencją MIT. Jest to standardowa MODA rozszerzona o obsługę wielowątkowości. Jedyną modyfikacją zastosowaną w algorytmie MODA jest zmiana domyślnej liczby iteracji - autorzy proponują liczbę iteracji równą $\frac{|V|}{3}$, gdzie V to zbiór wierzchołków w sieci. Sprawia to jednak, że algorytm nie działa w sposób deterministyczny, a jedynie dobrze przybliża faktyczną liczbę wystąpień danego podgrafu w sieci. Ze względu jednak na niedeterministyczny charakter drugiego z porównywanych algorytmów, tj. algorytmu FASCIA, zdecydowaliśmy się na zmianę domyślnej liczby iteracji na wartość, która gwarantuje nam dokładny wynik, czyli $|V|$. Zapewnia nam to większe bezpieczeństwo porównań, szczególnie, że wyniki zliczeń mogą być bardzo zbliżone. Wymagało to jednak tylko pojedynczej zmiany w kodzie:

```
1 if (numberOfSamples <= 0) numberOfSamples = inputGraph.VertexCount (/ 3);
```

Warto także zauważyć, że algorytm MODA nie należy do głównej części naszego rozwiązania, a służy jedynie jako punkt odniesienia do wyników uzyskanych algorytmem FASCIA. Choć stanowił on jeden z trzonów naszej analizy, jego obsługa nie jest częścią aplikacji, a użytkownik ma możliwość zweryfikowania wyników na własną rękę.

1.2 Color Coding - FASCIA

Jako implementację Color Codingu wykorzystaliśmy open-source'owy algorytm FASCIA[3]. W standardowej wersji, zwracany jest uśredniony wynik z wielu iteracji. W przypadku naszej implementacji, opartej o strumienie, zwracamy szacowanie po każdej iteracji. Dzięki temu, możemy testować hipotezę w czasie rzeczywistym. W wielu przypadkach pozwala to na zakończenie działania algorytmu dużo wcześniej niż obliczona maksymalna liczba iteracji.

Jako że nie musieliśmy wykonywać bezpośrednich zmian w ciele algorytmu, ponieważ FASCIA po każdej iteracji zwraca obecną szacunkową liczbę wystąpień danego motywu w sieci, postanowiliśmy do nowej obsługi problemu wykorzystać proste skrypty w językach bash i Python.

Składowymi naszego rozwiązania są odpowiednio:

- Skrypt w języku Python, który oblicza maksymalną liczbę iteracji dla algorytmu FASCIA na podstawie przyjmowanych sieci, motywu i argumentu podanego przez użytkownika określającego wariant stopowania (warianty są opisane w dokumentacji wstępnej),

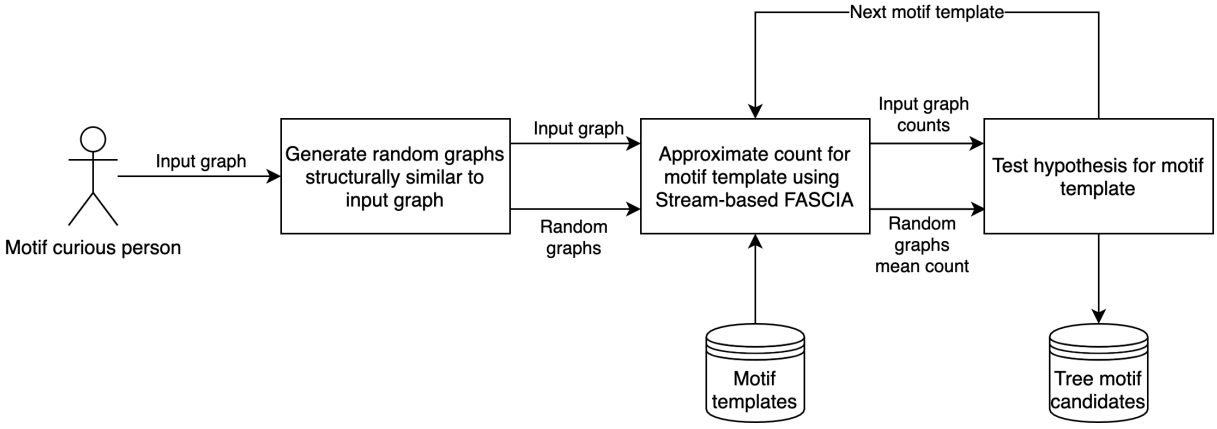
[1] Repozytorium `ParaMODA`: `https://github.com/smbadiwe/ParaMODA`.

[2] Wszystkie źródła informacji zawartych w dokumentacji końcowej pokrywają się ze źródłami w dokumentacji wstępnej - w razie problemów odsyłamy do zamieszczonych tam artykułów po bardziej szczegółowe informacje

[3] Repozytorium `FASCIA`: `https://github.com/francktcheng/fascia`.

- Prosty skrypt w języku bash, który uruchamia algorytm FASCIA na pojedynczym grafie,
- Skrypt w języku bash, który obsługuje skrypt pythonowy do obliczania wartości testu statystycznego oraz pipeline pomiędzy skryptami. Kontroluje on maksymalną liczbę iteracji oraz kończy działanie algorytmu, gdy wyniki testu stają się jednoznaczne. Stanowi on tym samym główne ciało rozwiązania,
- Skrypt `./filter.sh` - jedyny skrypt przewidziany do bezpośredniego użycia, służy do obsługi programu oraz wywołuje pozostałe części kodu. Przyjmuje jako argumenty główne sieć w postaci pliku tekstowego zapisanego z rozszerzeniem `.graph` oraz maksymalną liczbę wierzchołków drzew, które zostaną zweryfikowane co do bycia motywami. Plik sieci powinien w pierwszych dwóch linijkach zawierać odpowiednio liczbę wierzchołków i liczbę krawędzi sieci, a w następnych liniach pary wierzchołków oddzielane spacją, które oznaczają obecne w sieci krawędzie. Wierzchołki indeksowane są od zera. Dodatkowo, skrypt przyjmuje preferencję użytkownika co do zwracanej listy grafów podejrzanych o bycie motywami zgodnie z opisem w dokumentacji wstępnej, oraz jedną z dwóch wartości: `meanRandom` lub `expER` wskazujące, w jaki sposób ma zostać obliczona maksymalna liczba iteracji. Argument `meanRandom` wskazuje, że powinna ona zostać obliczona na podstawie średniej liczby wystąpień danego motywu w sieciach generowanych losowo, a argument `expER` oznacza obliczanie maksymalnej liczby iteracji na podstawie oczekiwanej liczby wystąpień w modelu Erdősa-Renyi'ego.

Wynikiem wywołania powyższych algorytmów jest lista nazw plików `.graph` odpowiadających drzewom, które zostały oznaczone jako kandydaci na motywy. Ich źródłem jest katalog `motif` zawierający wszystkie drzewa o ilości wierzchołków nie przekraczającej 13 i podzielony na podfoldery grupujące drzewa o tym samym rozmiarze. Rysunek 1 prezentuje schemat działania całego rozwiązania, zgodnie z powyższym opisem.



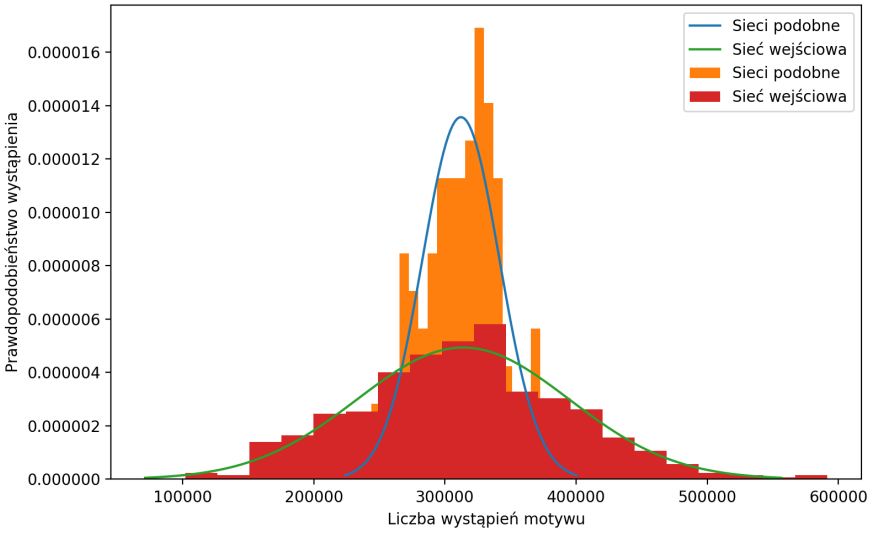
Rysunek 1: Schemat działania proponowanego rozwiązania

1.3 Pozostałe modyfikacje

Choć w naszym rozwiązaniu nie nastąpiły drastyczne zmiany w stosunku do dokumentacji wstępnej, częściowo przez fakt, że głównym celem naszego rozwiązania pozostaje

analiza, a nie dostarczenie aplikacji, tak musieliśmy wprowadzić pomniejsze zmiany wraz z pojawianiem się nowych wyników w trakcie analiz.

Przede wszystkim, zmianie uległ przeprowadzany po każdej iteracji FASCII test statystyczny. O ile w dokumentacji wstępnej zakładaliśmy wykorzystanie jednostronnych testów t-Studenta, o tyle zdecydowaliśmy się zmienić to podejście na rzecz testów dwustronnych. Dzięki temu możemy przy odpowiednio niskiej p-wartości zdecydować nie tylko, czy dane poddrzewo występuje w wejściowej sieci statystycznie częściej, niż w zbiorze sieci losowych, ale także czy występuje w niej statystycznie rzadziej. To pozwala na jednoznaczne odrzucanie drzew, które wg. FASCII motywami nie są, oraz na wybór ze strony użytkownika, czy chce skupić się jedynie na drzewach dla których test statystyczny wykazał odpowiednią częstość występowania, czy także na tych, dla których test statystyczny nie był rozstrzygający. Zrezygnowaliśmy także z pomysłu porównywania średniej rozkładu liczby wystąpień szukanego poddrzewa w sieciach losowych ze średnią otrzymaną z kolejnych iteracji algorytmu FASCIA na sieci wejściowej. Zamiast tego bierzemy pod uwagę cały rozkład poszczególnych zliczeń otrzymanych z iteracji color-codingu, otrzymując drugi rozkład. Wykorzystujemy więc nie test t-Studenta, a test Welcha dla rozkładów o nierównych wariancjach. Rysunek 2 przedstawia jak wyglądają rozkłady zliczeń pewnego poddrzewa uzyskanych w pewnej sieci oraz sieciach do niej podobnych.



Rysunek 2: Przykładowe rozkłady przybliżonych liczb wystąpień poddrzewa

Druga istotna zmiana nastąpiła na etapie kontroli liczby iteracji działania algorytmu. Zakładaliśmy korzystanie z dwóch metod obliczania odpowiedniej liczbą iteracji - jedna korzystająca z estymatora zbudowanego na grafach losowych i druga z estymatora opartego na modelu Erdősa–Rényi'ego. Zwracane wartości były bardzo podobne, ale doszliśmy do wniosku, że bardziej właściwe w naszym problemie będzie korzystanie z estymatora wynikającego ze średniej otrzymanej na losowych sieciach. Pozostawiamy jednak użytkownikowi możliwość porównania obu metod i próby dostrzeżenia ewentualnych różnic. Ponadto, poprzednie założenia nie uwzględniały sytuacji, gdy "dość szybko" jesteśmy w stanie zweryfikować, czy dany graf jest lub nie jest motywem. W naszej modyfikacji zamiast czekać do końca działania algorytmu, postanowiliśmy zaimplementować warunek wczesnego stopu oraz odpowiednie warunki graniczne:

1. Jeśli test statystyczny dwadzieścia razy z rzędu postanowi, że dany graf jest motywem (lub odpowiednio tyle samo razy, że nie jest motywem), działanie algorytmu zostaje przerwane i przechodzi on do następnego drzewa.

2. Maksymalna liczba iteracji została z dołu ograniczona przez 100

3. Maksymalna liczba iteracji została z góry ograniczona przez 500

Modyfikacja z punktu drugiego uodparnia algorytm na sytuacje, gdy w przypadku małych sieci zamierzony przedział ufność jest zbyt szeroki, przez co algorytm osiąga go po kilku iteracjach. Pojedyncza iteracja algorytmu FASCIA jest zaś na tyle mało kosztowna, że bardziej opłacalne było wymuszenie na algorytmie większej dokładności. Z drugiej strony, dla niektórych sieci maksymalna liczba iteracji sięga dziesiątek tysięcy, co dla celów analizy jest zbędne, ponieważ próbka badawcza wielkości 500 obserwacji w zupełności wykorzystuje potencjał pomysłu zastosowania testu statystycznego - jeśli po 500 iteracjach test nie odrzucił ani nie zaakceptował danego podgrafu, prawdopodobieństwo, że stanie się to wskutek jego dalszego działania, jest znikome.

2 Kierunek projektu

Decydując się na nasze rozwiązanie zwróciliśmy uwagę na probabilistyczny charakter problemu wykrywania motywów. Nie ma jednoznacznej zasady, według której ustalamy czy podgraf jest motywem. Dzięki temu problem jest otwarty na rozwiązania aproksymacyjne, niewymagające deterministycznego wyniku. Ich użycie wymaga tylko niewielkiej modyfikacji hipotezy statystycznej sprawdzającej czy podgraf jest motywem.

W problemie wyszukiwania motywu, użytkownika nie interesują wyłącznie podgrafy będące drzewami, przez co FASCIA nie jest zazwyczaj brana pod uwagę jako jego rozwiązanie. Przygotowane przez nas rozwiązanie stworzyliśmy z myślą o modyfikacji algorytmu, który rozwiązuje problem całościowo. Zdecydowaliśmy się na MODĘ, ponieważ tak jak FASCIA reprezentuje podejście motif-centric. W algorytmie MODA możemy wyróżnić etap zajmujący się wyłącznie drzewami, co jest idealną okazją na użycie w nim FASCII w celu potencjalnej poprawy szybkości wyszukiwania motywów.

Celem projektu było porównanie algorytmu aproksymacyjnego z rozwiązaniem deterministycznym w celu zdecydowania, czy użycie FASCII na etapie wyszukiwania drzew będzie wydajniejsze od podstawowej wersji, w której wykorzystywany jest algorytm Grochowa-Kellisa. Uzyskane przez nas wyniki pozwolą porównać te dwa algorytmy i zdecydować, czy warto modyfikować algorytm MODA.

3 Dane testowe

Problem wyszukiwania motywu w grafach jest ważnym zagadnieniem w biologii, dlatego szczególnie istotne były w naszych testach właśnie sieci biologiczne. Ponadto wykorzystaliśmy dane testowe pochodzące z naszego zespołu na MS Teams, aby umożliwić porównanie wyników pomiędzy grupami. Zamieszczamy dokładny opis sieci, które wkorzystaliśmy:

- Sieć Filmweb z zespołu MS Teams - sieć powiązań pomiędzy aktorami rozumianymi jako wspólne granie w danym filmie. Niestety oryginalna sieć była zbyt duża,

przez co algorytm GK nie zwrócił żadnego wyniku ze względu na braki pamięci. Postanowiliśmy jednak wziąć próbki tej sieci różnych wielkości, gdzie każda próbka jest podgrafem większej. Umożliwi nam to analizę złożoności algorytmu na rosnących zbiorach, jednak nie jest to zbiór, który jest wiarygodny pod kątem realnie istniejących sieci. Odpowiednie warianty sieci Filmweb są następujące:

  - Small - 2577 wierzchołków, 3136 krawędzi,
  - Medium - 3395 wierzchołków, 6048 krawędzi,
  - Big - 3984 wierzchołków, 10288 krawędzi,
  - Large - 4627 wierzchołków, 18965 krawędzi,

- Sieć NBA z zespołu MS Teams - rozmiar sieci ponownie przerósł nasze możliwości obliczeniowe, jednak udało nam się wykonać jeden pomiar algorytmem GK dla ścieżki trzywierzchołkowej. Zabieg obcięcia sieci do mniejszych rozmiarów tak, jak w przypadku sieci Filmweb, nie wydawał się konieczny, ponieważ ponownie nie dostalibyśmy wiarygodnej reprezentacji rzeczywistych sieci.

- Sieć H. pylori - sieć biologiczna bakterii *Helicobacter pylori*. Ma 714 wierzchołków i 1121 krawędzi.

- Sieć S. cerevisciae - sieć biologiczna grzyba *Saccharomyces cerevisiae*. Ma 690 wierzchołków i 1093 krawędzie.

- Sieć E. coli - sieć biologiczna pałeczki okrężnicy *Escherichia coli*. Ma 423 wierzchołki i 578 krawędzi.

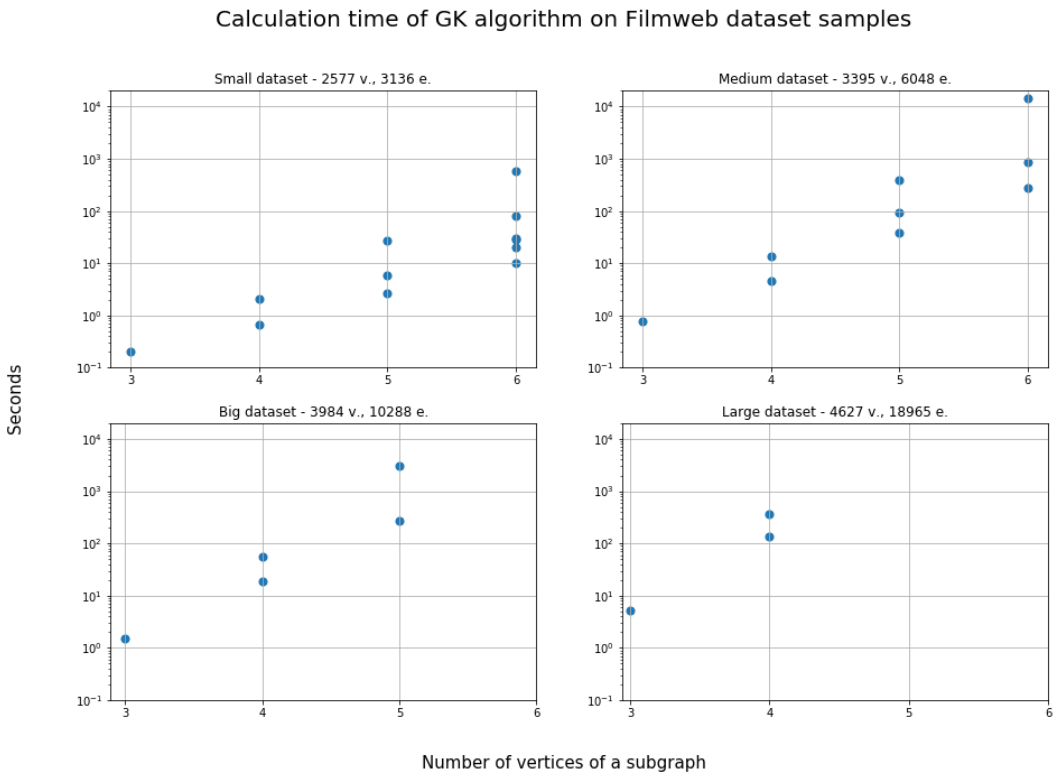
Na powyższych sieciach wykonaliśmy wykrywanie motywów naszą implementacją algorytmu FASCIA i algorytmem MODA (a właściwie częścią odpowiadającą za algorytm GK) dla wszystkich drzew od trzech do sześciu wierzchołków. Autorzy algorytmu GK sami wskazują, że ich rozwiązanie nie jest przystosowane do rozwiązywania problemu wyszukiwania większych motywów; nam także nie udało się wykonać żadnego wywołania algorytmu dla motywów siedmiowierzchołkowych. W przedstawionych wynikach będą pojawiały się braki - wynika to z faktu, że już na niektórych z przedstawionych sieci, dla określonych motywów, zwracany był błąd wynikający z braku dostępnej pamięci. Przedstawiamy zatem wszystkie rezultaty, wynikające z obliczeń, jakie udało się osiągnąć.

4 Uzyskane wyniki

Analizę wyników rozpoczniemy od zbioru Filmweb, który stanowi dobry punkt wyjścia do zweryfikowania poprawności działania implementacji. Jako że nie reprezentuje on jednak odpowiednio rzeczywistych sieci ze względu na próbkowanie grafu, weryfikacja motywów została przeprowadzona jedynie na sieciach biologicznych.

4.1 Filmweb

Rysunek 3 przedstawia czas potrzebny algorytmowi GK na wykrycie motywów w wariantach sieci Filmweb. Interesujące było dla nas, jak zmienia się czas wykonywania się algorytmu GK, ponieważ autorzy nie podają dokładnej złożoności swojego rozwiązania, zatem w celach analizy kluczowa była odpowiedź na pytanie, od czego ten czas zależy.

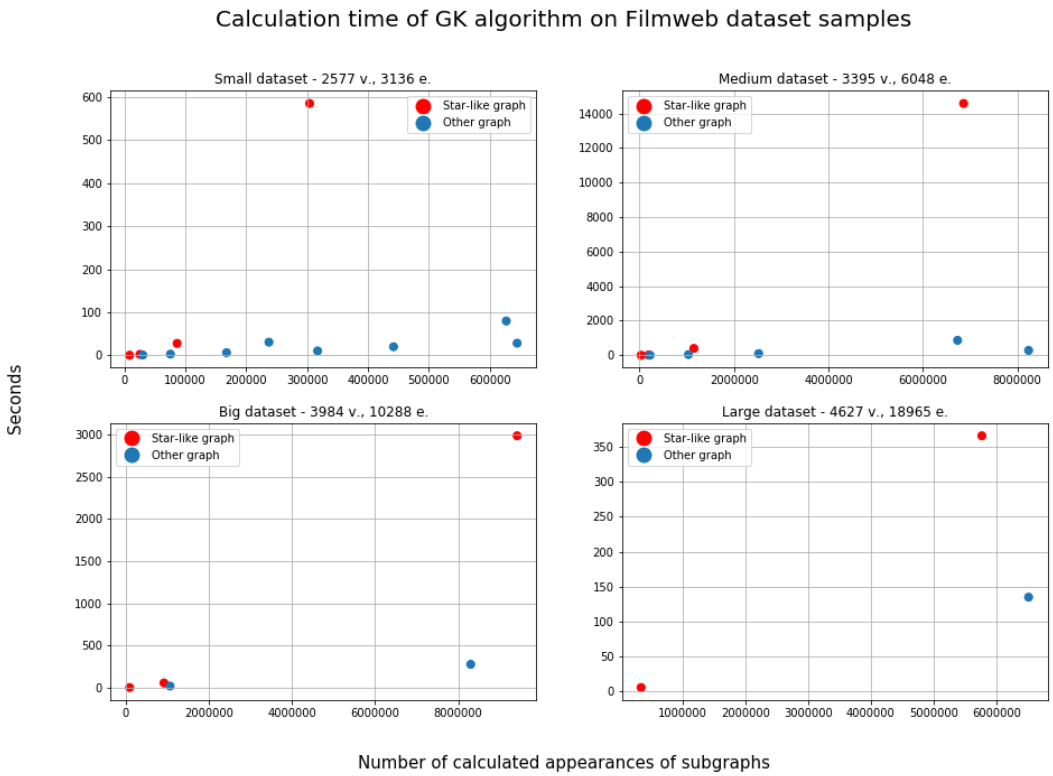


Rysunek 3: Porównanie czasu wykonywania algorytmów dla różnych wielkości motywu

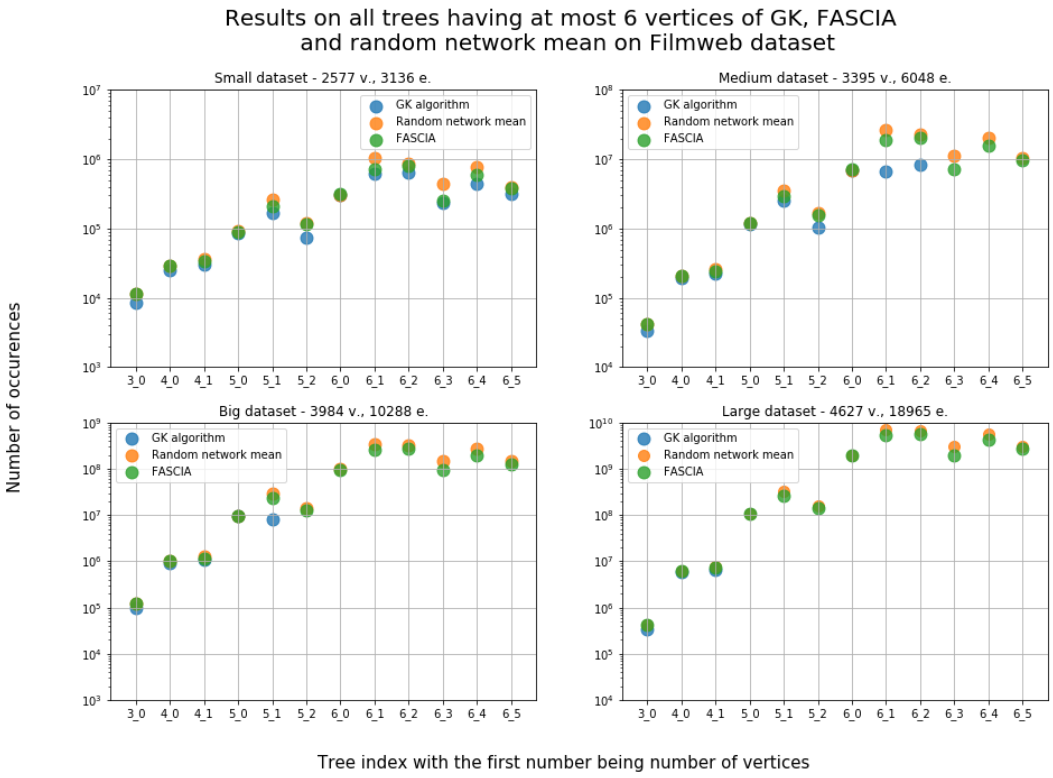
Rysunek 3 jednoznacznie wskazuje, że czas działania rośnie wykładniczo wraz ze wzrostem rozmiaru motywu, co potwierdza stawiane w dokumentacji wstępnej przypuszczenia. Zauważmy jednak, że czasy obliczeń są dość silnie rozrzucone dla ustalonej wielkości motywu - nawet na skali logarytmicznej możemy zauważyć dość dużą wariancję. Warto zatem postawić pytanie, czy pewne konkretne motywy wymagają dłuższego czasu obliczeń, niż inne. Takie przypuszczenie musi jednak zostać skonfrontowane z obliczoną liczbą wystąpień danego motywu w sieci, ponieważ może to być istotny czynnik także mający wpływ na obliczenia, a niewidoczny na rysunku 3. Odpowiedzi na postawione pytanie udziela rysunek 4.

Na rysunku 4 pomimo niewielkiej liczby próbek możemy dość dobrze zaobserwować znacznie szybszy przyrost potrzebnego czasu do wykonania algorytmu, gdy kandydatem na motyw jest gwiazda, niż w pozostałych przypadkach. Jest to o tyle interesujące, że problem zliczania gwiazd można bardzo szybko rozwiązać, gdy znane są stopnie wierzchołków w sieci. O ile nie jest to prawdopodobnie istotne, gdy użytkownik jest zainteresowany analizą wszystkich grafów, zamiast tylko drzew, lub nawet gdy ma dostęp do narzędzi umożliwiających prowadzenie obliczeń na większych motywach, o tyle w naszym przypadku osobne rozważenie gwiazd w algorytmie GK mogłoby przyspieszyć proces obliczeń ponad pięciokrotnie. Ponadto, rysunek 4 obrazuje także co najmniej liniową - choć podejrzewamy wykładniczą - zależność czasu wykonania od liczby znalezionych wystąpień danego podgrafu.

Po przedstawionej powyżej analizie złożoności algorytmu GK możemy przejść do porównania wyników porównywanych algorytmów. Odpowiednie rezultaty są przedstawione na rysunku 5.



Rysunek 4: Zależność czasu obliczeń od liczby wykrytych wystąpień motywu w sieci

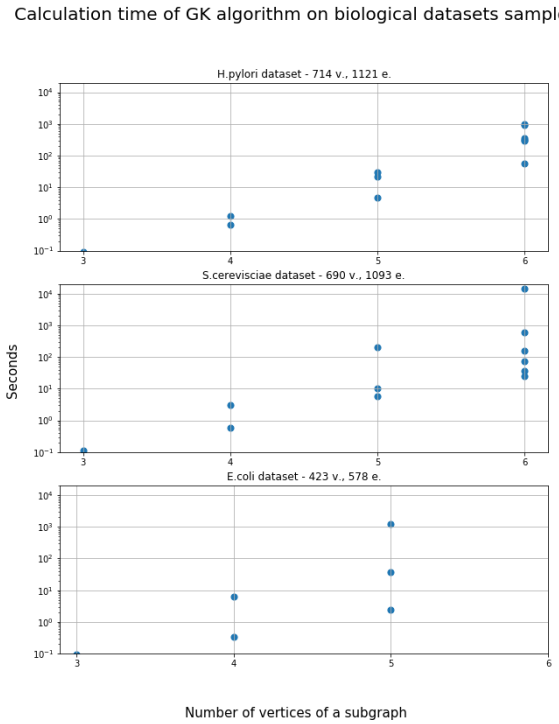


Rysunek 5: Szacowane liczby wystąpień motywów. W niektórych przypadkach wyniki pokrywają się, przez co widoczność niektórych wartości może być ograniczona.

Podstawową obserwacją, jaką należy wykonać na podstawie rysunku 5, jest zweryfikowanie poprawności naszej implementacji algorytmu FASCIA do wykrywania motywów. Należy pamiętać, że w naszej implementacji nie jest istotna ostatnia wartość zwrócona przez algorytm FASCIA, ponieważ celem nie jest zliczanie wystąpień danego grafu, a testowanie otrzymanych wartości pod kątem wystąpienia istotnej różnicy ze średnią. Z tego powodu wyniki algorytmu FASCIA najczęściej znajdują się blisko średniej, ponieważ w trakcie ich zbiegania do wartości otrzymanej algorytmem GK nastąpiło przerwanie - dokładniej opisane w sekcji 1.3. Jeśli zatem wynik algorytmu FASCIA znajduje się pomiędzy wynikiem algorytmu GK, a wartością na sieciach losowych, możemy wnioskować o poprawności obliczeń. W przypadku wariantów sieci Filmweb we wszystkich przypadkach, dla których było możliwe policzenie GK, zaszła opisana sytuacja, co potwierdza poprawność implementacji algorytmu.
Na zbiorze Filmweb nie wykonujemy wykrywania motywów, ponieważ sieć nie odpowiada rzeczywistemu obiektowi i służy jedynie do analizy poprawności i złożoności algorytmów. Wyniki skuteczności algorytmu FASCIA zostaną przedstawione w następnej sekcji.

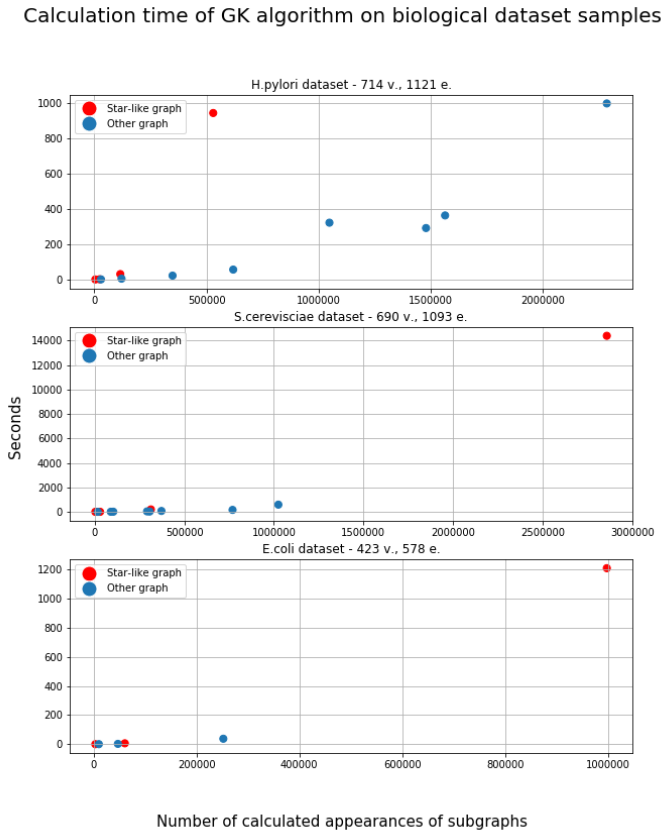
4.2 Sieci biologiczne

Po zweryfikowaniu poprawności implementacji algorytmu i zaprezentowaniu częściowych rezultatów, możemy przejść do głównej części analizy, czyli oceny skuteczności naszej implementacji w wykrywaniu motywów. Ze względu jednak na nierealistyczny charakter sieci Filmweb, na sieciach biologicznych także wykonaliśmy analizy złożoności algorytmu GK, aby ostatecznie potwierdzić prawdziwość wcześniej postawionych obserwacji.



Rysunek 6: Zależność czasu obliczeń od wielkości motywu w sieciach biologicznych

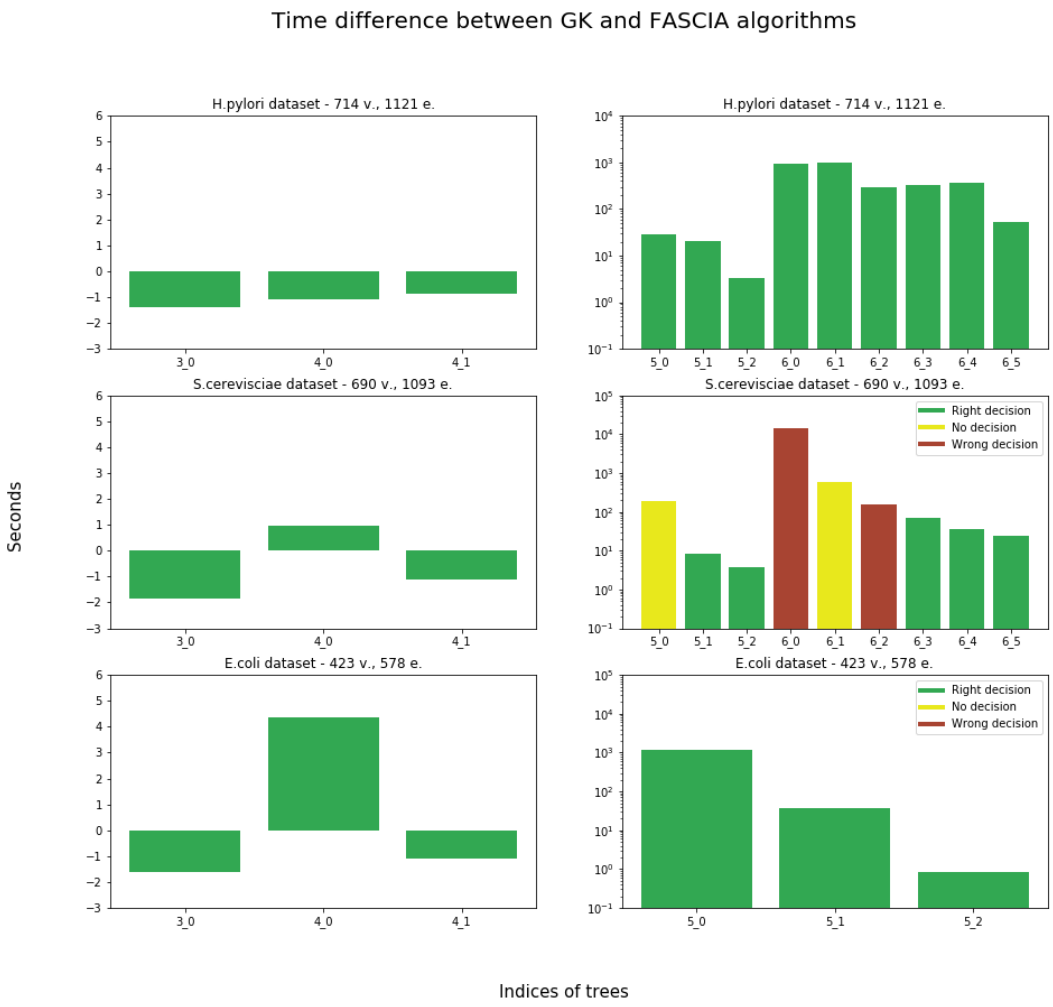
Rysunki 6 i 7 przedstawiają wyniki obliczeń wykonanych wcześniej na sieci Filmweb powtórzone teraz na sieciach biologicznych. O ile nie ma wątpliwości, co do wykładniczej



Rysunek 7: Zależność czasu obliczeń od liczby znalezionych wystąpień w sieciach biologicznych

zależności czasu od wielkości motywu (rysunek 6), o tyle nowe wyniki zdają się potwierdzać nasze poprzednie przypuszczenia, że zależność czasu od liczby znalezionych wystąpień także jest wykładnicza (rysunek 7), a nie - jak obserwowaliśmy wcześniej - liniowa. Trudno nam jednak ze stuprocentową pewnością stwierdzić prawdziwość tej obserwacji ze względu na niewielką liczbę próbek. Analogicznie, wszystkie obserwacje postawione w poprzedniej sekcji zostają dodatkowo potwierdzone.

Rysunek 8 prezentuje kluczowe wyniki przeprowadzonej przez nas analizy. Ze względu na brak jasnej procedury klasyfikowania grafu jako motyw w algorytmie MODA zakładamy, że dany graf jest motywem, gdy liczba jego wystąpień wynosi co najmniej 101% średniej liczby wystąpień w sieciach losowych. Przypomnijmy z dokumentacji wstępnej, że celem naszego programu jest zwrócenie listy kandydatów na motywy wynikającą z naszej implementacji algorytmu FASCIA, aby umożliwić zmniejszenie narzutu obliczeniowego na algorytm GK w implementacji algorytmu MODA. Postępujemy tak, ponieważ nie otrzymujemy z algorytmu FASCIA listy wystąpień danego grafu w sieci, a jedynie ich liczbę - listę tych wystąpień musimy zatem wygenerować algorytmem GK. Sprawia to zatem, że jesteśmy narażeni na dwa rodzaje błędów - zwrócenie danego grafu jako motyw, mimo że nim nie jest, lub niezwrócenie danego grafu, pomimo że jest motywem. Zauważmy, że w potencjalnej implementacji algorytmu MODA popełnienie błędu pierwszego typu nie wpływa na całościową skuteczność wykrycia motywów - jeśli dany graf został fałszywie zaklysfikowany jako motyw przez algorytm FASCIA, zostanie to ponownie zweryfikowane algorytmem GK, po którym graf ten zostanie odrzucony. Jedynym negatywnym skutkiem takiego błędu jest wydłużenie się wykonania algorytmu. Popełnienie błędu drugiego typu jest bardziej kosztowne, ponieważ nie otrzymujemy pełnej informacji o motywach w



Rysunek 8: Różnice w czasie działania naszej implementacji algorytmu FASCIA i algorytmu GK oraz skuteczność algorytmu FASCIA

grafie. Należy zatem zaznaczyć, że wszystkie błędy algorytmu FASCIA, jakie zdarzyły się podczas obliczeń, są błędami pierwszego typu - nie wpływają na całościową skuteczność wykrywania motywu.

Na podstawie uzyskanych wyników otrzymujemy, że zastosowanie algorytmu FASCIA przed algorytmem GK ma szansę istotnie poprawić koszt obliczeniowy, ponieważ algorytm FASCIA wykonuje się znacznie szybciej od algorytmu GK, gdy wykrywamy motywy o co najmniej pięciu wierzchołkach. Dla przykładu, w sieci H. pylori nie wystąpił żaden motyw - wszystkie analizowane drzewa zostały odrzucone przez test statystyczny już po kilku sekundach. Sprawia to, że analiza motywów przy pomocy algorytmu MODA staje się zupełnie niepotrzebna i oszczędzamy czas równy sumie wysokości słupków na wykresach w pierwszym rzędzie na rysunku 8. Jedyne błędy, jakie zanotowaliśmy, to niepoprawne zaklasyfikowanie dwóch grafów jako motywy - pomimo że nimi nie były - w sieci S. cerevisciae. W zależności od preferencji użytkownika może on zatem przekazać do algorytmu MODA jedynie grafy zaklasyfikowane jako motyw, lub również grafy, co do których FASCIA nie podjęła jednoznacznej decyzji. W obu przypadkach jednak osiągamy znaczną poprawę czasową. Hipotetyczną sytuacją, w której nasz algorytm powoduje zwiększenie kosztu obliczeniowego algorytmu MODA, jest wystąpienie dużej liczby ana-

lizowanych grafów jako motywy w pewnej sieci. Wtedy algorytm GK musi wykonać się w większości standardowych przypadków, a dochodzi do tego koszt obliczeniowy naszej implementacji. Dla motywów od pięciu wierzchołków wzwyż względny dodatkowy narzut obliczeniowy wydaje się on jednak znikomy.

4.3 NBA

Na koniec naszej analizy możemy przedstawić wyniki osiągnięte na sieci NBA. Udało nam się wykonać jedynie jeden pomiar algorytmem GK, ponieważ pozostałe zwracały błędy pamięci. Osiągnięte wartości zostały jednak przedstawione w tabeli.

Średnia liczba wystąpień	Wynik algorytmu GK	Wynik algorytmu FASCIA
14 312 205	6 573 909	14 226 974

Tabela 1: Wyniki dla ścieżki trzywierzchołkowej na zbiorze NBA

Algorytm FASCIA poprawnie zidentyfikował analizowany graf jako nie-motyw. Konkluduje to tym samym naszą analizę, stanowiąc dodatkowy punkt potwierdzający skuteczność i poprawność naszej implementacji.

5 Wnioski

- Pierwszym, podstawowym wnioskiem jest stwierdzenie poprawności działania naszej implementacji algorytmu. Przedstawione wyniki potwierdzają właściwe działanie rozwiązania, a format zwracanych wyników - listy nazw plików z grafami - umożliwia łatwe zaimplementowanie dalszej modyfikacji algorytmu MODA. Ponadto, podczas procesu implementacji zaszło stosunkowo niewiele zmian w porównaniu do konceptu z dokumentacji wstępnej.

- Na podstawie osiągniętych wyników byliśmy w stanie zaobserwować złożoności czasowe algorytmu GK, co pozwoliło nam lepiej poznać ten algorytm, jako że twórcy nie opisują dokładnie złożoności. Zaobserwowaliśmy wykładniczą zależność od wielkości motywu i prawdopodobnie wykładniczą zależność od liczby wystąpień, choć niewielka liczba próbek nie pozwoliła nam się stuprocentowo przekonać do drugiej z tych obserwacji.

- Dzięki wykorzystaniu gotowych implementacji algorytmów MODA i FASCIA czas, który musielibyśmy poświęcić na implementację, mogliśmy przeznaczyć na dodatkową analizę. Ponadto, formaty wyników, w których algorytmy zwracają rezultaty, były na tyle przystępne, że nie musieliśmy dokonywać dużych zmian w kodzie, a jedynie odpowiednio przekierowywać wyniki do własnych skryptów.

- Zaskakuje wyjątkowo wolne działanie algorytmu GK dla gwiazd. Choć w szerszych analizach gwiazdy stanowią ułamek branych pod uwagę grafów, w naszym przypadku stanowiły zdecydowanie największe wyzwanie w obliczeniach. W przypadku użytkowania algorytmu GK dla drzew, w takich analizach, jak nasza, warto będzie poświęcić czas na rozważenie zliczania gwiazd jako osobnego przypadku, bo da się to zrobić w czasie liniowym w dowolnej sieci.

- Nasza implementacja algorytmu FASCIA daje satysfakcjonujące rezultaty, których skuteczność prawdopodobnie da się jeszcze poprawić, przeprowadzając odpowiedni proces optymalizacji warunków stopu. Wydaje się, że ma on największy potencjał, jeśli chodzi o dalsze usprawnienia algorytmu.

- Na podstawie przedstawionych wyników można spróbować wysnuć obserwację, że w pierwszych iteracjach FASCIA lekko przeszacowuje liczbę, do której zbiega. Nie znamy dokładnego wyjaśnienia tego zjawiska, jednak jest dla nas bardzo korzystne, ponieważ znacznie zmniejsza ryzyko popełnienia błędu drugiego typu (akapit pod rysunkiem 7, sekcja 4.2).

- Zaobserwowane rezultaty jednoznacznie wskazują na możliwość bardzo znaczącego zmniejszenia narzutu obliczeniowego przy wykonywaniu algorytmu MODA. Otrzymane wyniki sugerują, że narzut na algorytm GK może spaść od ok. 70% do nawet 100%, gdzie jego wykonanie w ogóle nie jest konieczne. Ma to szczególne znaczenie także dlatego, że algorytmy *motif-centric* w ogólności wykazują bardzo dużą nietolerancję na motywy o większej liczbie wierzchołków. Być może także w innych rozwiązaniach będzie możliwe dodanie analogicznego usprawnienia.

6 Podział prac

- Dokumentacja wstępna - Łukasz, Jakub, Kacper
- Prezentacja wstępne - Łukasz, Jakub, Kacper
- Zagadnienia związane z algorytmem FASCIA - Jakub, Kacper
- Zagadnienia związane z algorytmem MODA - Łukasz
- Potoki przetwarzania, skrypty w bashu - Jakub
- Przygotowanie danych do testów - Łukasz, Kacper
- Przygotowanie wykresów - Łukasz
- Dokumentacja końcowa - Łukasz, Jakub, Kacper
- Prezentacja końcowa - Łukasz, Jakub, Kacper