Grafy i Sieci - Projekt

Aproksymacyjne wykrywanie motywów w sieciach

Łukasz Brzozowski, Jakub Janaszkiewicz, Kacper Siemaszko

29 maja 2020

Spis treści

1 Przedstawienie problemu

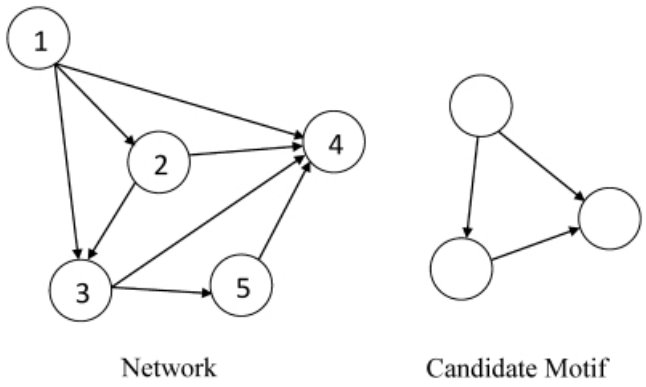
Podczas analizy sieci o dużych rozmiarach, chcielibyśmy dowiedzieć się, czy posiada ona nieprzypadkową strukturę. Jednym z pytań, które sobie zadamy podczas takiej analizy jest to, czy w grafie często pojawia się jakiś wzorzec. Żeby odpowiedzieć na to pytanie, możemy przyjrzeć się podgrafom o małej liczbie wierzchołków i pogrupować je, porównując ich struktury. Pojawienie się zbioru, którego liczność jest istotnie duża, sugeruje nam występowanie w sieci wzorca. Taki wzorzec nazywamy motywem sieci. Ogólne przedstawienie problemu pozostawia wiele niewiadomych - pojawia się zatem potrzeba doprecyzowania definicji zagadnienia. Możemy to zrobić poprzez zadanie sobie kilku pytań odnośnie rozwiązywanego przez nas problemu wyszukiwania motywu sieci.

- Czy podgrafy mogą się pokrywać?
- Podgrafy indukowane czy nieindukowane?
- Jak grupować wygenerowane podgrafy?
- Kiedy liczność zbioru jest istotnie duża?

Kiedy odpowiemy na te pytania, otrzymamy poprawnie zdefiniowany problem, który możemy zacząć rozwiązywać.

1.1 Pokrywanie się i typy podgrafów

Na samym początku decydujemy o założeniach problemu. Na tym etapie potrzebna jest wiedza ekspercka, która pozwoli nam określić sposób generowania podgrafów i ich wzajemne oddziaływanie.

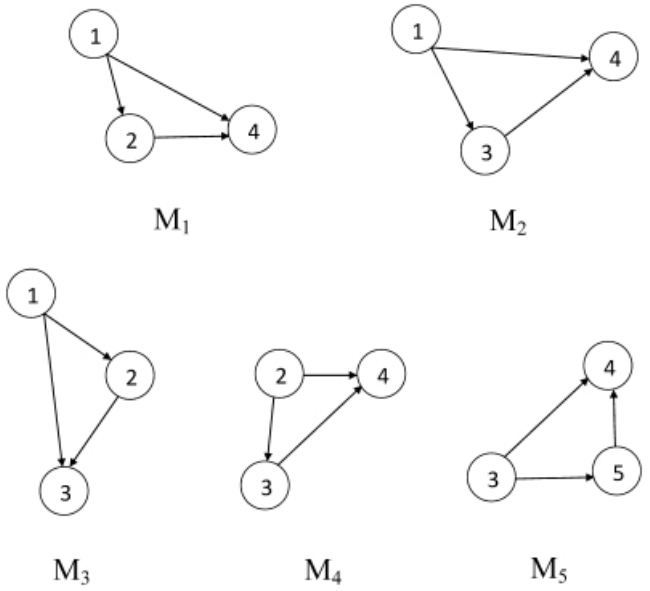


Rysunek 1: Graf i testowany wzorzec
Źródło: [1]

Na bazie przykładu z rysunku 1 możemy zobaczyć w jaki sposób założenia o dopuszczeniu pokrywających się podgrafów wpłyną na liczbę wystąpień wzorca w grafie. Rozróżniamy trzy modele [1]:

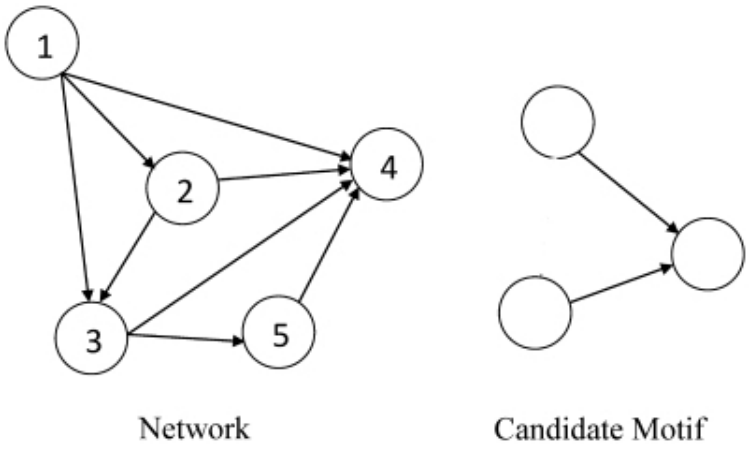
- F_1 - wierzchołki i krawędzie mogą się pokrywać,
- F_2 - wierzchołki mogą się pokrywać, krawędzie nie,
- F_3 - ani wierzchołki, ani krawędzie nie mogą się pokrywać.

W zależności od wybranego modelu wyniki mogą znacząco się różnić. W najmniej ograniczającym modelu - F_1 - znajdziemy pięć dopasowań. Sytuacja robi się już bardziej skomplikowana w modelu F_2. Rozwiązania są niejednoznaczne - zarówno pod względem dopasowanych podgrafów, jak i ich liczności. W modelu F_2 moglibyśmy dopasować - $\{M_4\}$ lub $\{M_1, M_5\}$, lub $\{M_3, M_5\}$. W modelu F_3 dopasujemy tylko jeden z podgrafów - po wykluczeniu trzech z pięciu wierzchołków nie jesteśmy już w stanie znaleźć kolejnego wystąpienia.



Rysunek 2: Dopasowania podgrafów w modelu F_1
Źródło: [1]

Kolejnym założeniem, które powinniśmy przyjąć, jest sposób w jaki generujemy podgrafy. Wyróżniane są dwa podejścia - indukowane i nieindukowane.



Rysunek 3: Testowanie uproszczonego wzorca
Źródło: własne

Różnicę między nimi możemy zobaczyć na przykładzie z rysunku 3. Jeśli sprawdzamy podgrafy nieindukowane, dopasujemy 7 podgrafów. Jeśli będziemy sprawdzać wyłącznie podgrafy indukowane, dopasujemy tylko podgrafy $\{2, 4, 5\}$ i $\{1, 4, 5\}$.

1.2 Grupowanie grafów i liczność zbioru

W zależności od przyjętych założeń, grupowanie może przyjąć różny poziom zaawansowania. Najluźniejsze założenie - wierzchołki i krawędzie mogą się pokrywać - sprowadza proces grupowania do znalezienia klas abstrakcji relacji izomorfizmu określonej na przeglądanych podgrafach. Pozostałe założenia komplikują grupowanie przez wykluczanie i zależność od kolejności dodawania podgrafu do zbioru.

Dochodzimy do momentu, w którym pojawia się w problemie wyszukiwania motywów wątek statystyczny. To, czy możemy nazwać strukturę motywem, rozstrzygamy poprzez przetestowanie hipotezy zerowej $H_0 : N_G \leqslant \overline{N}$, gdzie N_G to liczba wystąpień struktury w grafie G, a $\overline{N}$ to średnia liczba wystąpień struktury w grafach porównywalnych do G (pod względem liczby wierzchołków, liczby krawędzi, rozkładu stopni wierzchołków). Odrzucenie hipotezy zerowej pozwala nam wierzyć, że testowana struktura jest motywem sieci.

1.3 Przyjęte założenia

Przedstawiamy listę założeń, z którymi będziemy przeprowadzać naszą analizę.

- Krawędzie i wierzchołki grafów mogą się pokrywać,
- poszukujemy dowolnych grafów, nie tylko indukowanych,
- poszukiwane grafy i przeszukiowana sieć są nieskierowane,
- poszukiwane grafy są spójne
- w sieci nie występują krawędzie wielokrotne ani pętle.

Ostatnie trzy założenia z powyższej listy wynikają z ograniczeń analizowanych algorytmów.

2 Typowe podejścia

Algorytmy rozwiązujące problem wykrywania motywów w sieciach dzielą się na dwa główne podejścia – network-centric oraz motif-centric.

2.1 Algorytmy *network-centric*

W algorytmach typu *network-centric*, w pierwszej kolejności, na podstawie sieci wejściowej, generowana jest lista podgrafów o małej liczbie wierzchołków. Następnie przetwarzamy listę w celu znalezienia w niej motywu. Podstawową zaletą takiego podejścia jest mniejsza liczba kandydatów na motywy – rozpatrywane są tylko struktury występujące w sieci. Różnice między algorytmami pojawiają się na dwóch wcześniej wymienionych etapach:

- Wygenerowanie listy podgrafów sieci,
- przetwarzanie listy podgrafów w celu znalezienia motywów.

Przykładami algorytmów *network-centric*, które prezentują różne podejścia do generowania i przetwarzania podgrafów, są algorytmy MFinder i Kavosh [1]. MFinder buduje listę podgrafów poprzez tzw. *edge extension*, czyli dokładanie do podgrafów kolejnych krawędzi, aż do osiągnięcia podgrafów o zadanym rozmiarze. Następnie wybiera tylko część krawędzi z sieci (*edge sampling*) i szuka wzorców wyłącznie w podgrafach, które je zawierają. Jeżeli w sieci istnieje motyw, powinien pojawić się również w losowo wybranych podgrafach. Drugi algorytm - Kavosh, buduje listę podgrafów poprzez *node extension*, dokładając do podgrafów kolejne wierzchołki. Następnie dzieli wszystkie podgrafy na klasy izomorfizmu i klasyfikuje je jako motyw w zależności od ich liczności.

2.2 Algorytmy *motif-centric*

Algorytmy typu *motif-centric* starają się odpowiedzieć na następujące pytanie - czy wzorzec S jest motywem grafu G? Żeby uzyskać funkcjonalność algorytmów typu *network-centric* musimy najpierw wygenerować zbiór wzorców, które będziemy testować pod kątem bycia motywem sieci. Oprócz użycia w problemie wykrywania motywu, algorytmy typu *motif-centric* mają potencjał w heurystykach opartych o informacje na temat częstotliwości typowych wzorców w sieci wejściowej. Ten aspekt eksplorujemy w dalszej części dokumentacji i tam też zamieściliśmy opis przykładowych algorytmów typu *motif-centric*.

3 Rozwiązanie aproksymacyjne

Odmiennie od przedstawionych do tej pory rozwiązań, *color coding*, czyli algorytm zaproponowany przez Alona et al. [2], zwraca szacowaną liczbę wystąpień zadanego wzorca w sieci. Do głównych zalet tego algorytmu należą:

- możliwość uzyskania szacowania o dowolnej precyzji,
- dużo mniejsza złożoność obliczeniowa.

W naszych rozważaniach rozpatrujemy jego modyfikację - algorytm FASCIA [3], czyli wersję algorytmu Alona zoptymalizowaną pod kątem obliczeń równoległych i sprytnym użyciem różnych struktur danych w celu oszczędzania pamięci.

Główną wadą algorytmu jest przystosowanie wyłącznie do wzorców o strukturze drzewiastej. W problemie wyszukiwania motywów w sieci jest to niestety istotne ograniczenie. W dalszej części zaproponujemy wykorzystanie algorytmu FASCIA do optymalizacji działania innych algorytmów, w których to ograniczenie nie będzie dla nas problemem.

3.1 Złożoności algorytmów liczących wzorce

Algorytmy liczące występowanie wzorca S w grafie G w sposób wyczerpujący, poprzez dokładanie sąsiadów, mają złożoność $O(N^k)$, gdzie N to liczba wierzchołków w grafie G, a k to liczba wierzchołków wzorca. Najlepsza modyfikacja tego typu algorytmów dochodzi do złożoności $O(N^{\frac{\omega k}{3}})$ [3], gdzie ω to wykładnik w szybkim mnożeniu macierzy. FASCIA jest zoptymalizowaną pod kątem zużycia pamięci i zrównolegnienia obliczeń wersją algorytmu *color coding* Alona et al. [2], który pozwala na znalezienie wyniku w złożoności $m \cdot 2^{O(k)} \cdot O(e^k \cdot \frac{\log 1/\delta}{\epsilon^2})$, gdzie m to liczba krawędzi w grafie G, $1 - 2\delta$ - współczynnik ufności, ϵ promień przedziału ufności. W algorytmie szacującym widzimy znacznie mniejszą zależność złożoności obliczeniowej od wielkości.

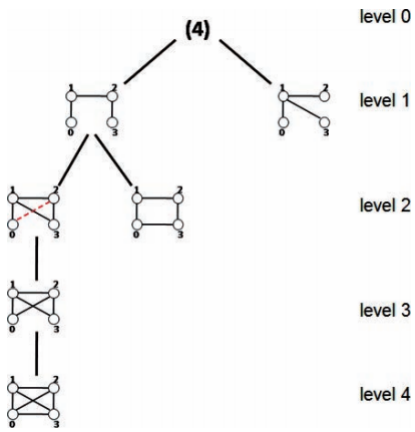
4 Opis algorytmów

Przed przedstawieniem naszej idei musimy najpierw wyjaśnić działanie algorytmów, które będziemy analizować i modyfikować.

4.1 Algorytm MODA

MODA - MOtif Detection Algorithm [4] to jeden z najwydajniejszych obecnie znanych algorytmów służących do wykrywania podstruktur w sieciach biologicznych. Ideą algorytmu jest wykorzystanie informacji o już wykrytych motywach do szybszego wykrywania następnych. Kluczowym obiektem algorytmu MODA jest *expansion tree*, abstrakcyjne drzewo nadające strukturę podgrafom, których częstość występowania należy zweryfikować. Z założenia MODA zaprojektowana jest tak, aby przyjmować liczbę wierzchołków i dla każdego grafu o tej liczbie wierzchołków sprawdzać, czy jest to motyw we wcześniej ustalonej sieci; możliwe jest jednak wykorzystanie tego algorytmu do weryfikacji dowolnego zbioru grafów wejściowych.

Expansion tree to drzewo, która w każdym wierzchołku poza korzeniem przechowuje podgraf, którego liczbę wystąpień w sieci chcemy policzyć. W korzeniu znajduje się po prostu liczba wierzchołków tych podgrafów, stała dla pojedynczego *expansion tree*. Dziećmi każdego węzła zawierającego graf G są grafy $G'_1, \ldots, G'_i$ takie, że z dowolnego z nich możemy otrzymać G przez operację usuwania krawędzi. Możemy łatwo postawić prostą obserwację: jeśli G nie jest motywem w wybranej sieci, żaden jego potomek w *expansion tree* także nie jest motywem w wybranej sieci. Pozwala to znacznie przyspieszyć operację wykrywania wszystkich motywów w sieci, ponieważ ograniczamy znacznie liczność sprawdzanego zbioru. Ponadto, znając położenia wystąpień grafu G w sieci (a dokładniej podgrafów izomorficznych z G), możemy każde z nich weryfikować pod kątem wystąpień każdego z $G'_1, \ldots, G'_i$.



Rysunek 4: T_4 - *Expansion tree* dla struktur o 4 wierzchołkach
Źródło: [4]

Na powyższym rysunku widzimy T_4, czyli *expansion tree* dla podgrafów czterowierzchołkowych. Łatwo zauważyć zachodzenie wcześniej opisanej relacji oraz dodatkowo to, że na poziomie pierwszym każdego *expansion tree* muszą znajdować się drzewa, o ile znajdują się w zbiorze zliczanych grafów (MODA ogranicza się do wyszukiwania motywów będących grafami spójnymi). Można zatem wywnioskować, że drzewa to szczególne przypadki wykrywanych struktur i należy zaaplikować do nich inny sposób zliczania, niż do

pozostałych grafów. W algorytmie MODA zliczanie drzew to *mapping module*, a zliczanie grafów niebędących drzewami to *enumerating module*. Poniżej przedstawiamy pseudokod algorytmu MODA:

input : G: sieć, k: liczba wierzchołków grafów, Δ: threshold
output: FSL: lista grafów będących motywami w G
note : F_G: zbiór wystąpień grafu w sieci

`fetch(`T_k`);`
begin
  G' = Get-Next-BFS(T_k);
  if $|E(G')| = (k-1)$ **then call** *Mapping-Module(G', G)*;
  else call *Enumerating-Module(G', G)*;
  save F_G;
  if $|F_G| > \Delta$ **then add** G' **into** FSL;
end

Algorytm 1: Pseudokod algorytmu MODA
Źródło: [4]

Widzimy zatem, że dwa główne komponenty algorytmu MODA to wymienione wcześniej moduły. Ze względu na fakt, że w naszej pracy skupiamy się na wykrywaniu drzew, szczególną uwagę należy poświęcić modułowi *mapping module*. Udowodniono, że wykrywanie dokładnej liczby wystąpień izomorfizmów w grafach jest problemem NP-zupełnym, zatem działanie dokładnego algorytmu *mapping module* byłoby bardzo kosztowne obliczeniowo. Z tego powodu, twórcy algorytmu MODA postanowili wykorzystać w tym miejscu algorytm Grochowa-Kellisa, opisany w następnym punkcie.

4.2 *Mapping module* - algorytm GK

Algorytm Grochowa-Kellisa [5] służy do zliczania liczby wystąpień danego grafu w sieci. Dzięki zastosowaniu *symmetry-breaking*, czyli uwzględnieniu przy zliczaniu automorfizmów grafu podczas zliczania, pozwala na szybsze niż naiwne otrzymanie finalnej wartości.

Powyższy pseudokod obrazuje wyszukiwanie liczby wystąpień grafu H w sieci G. W nawiasach kwadratowych zaznaczono fragmenty algorytmu odpowiadające za zastosowanie mechanizku *symmetry-breaking*, ponieważ także bez nich algorytm działa poprawnie. W powyższym kodzie $Aut(H)$ oznacza grupę automorfizmów grafu H, a stwierdzenie g **supports** h sygnalizuje, że może istnieć izomorfizm z H w podgraf G taki, że h jest mapowane na g. Sprawdzenie warunku "wspierania" h przez g sprowadza się do sprawdzenia, czy g ma odpowiedni stopień w porównaniu do stopnia h oraz czy stopnie sąsiadów g zgadzają się ze stopniami wierzchołków h. W ten sposób przechodzimy po wszystkich wierzchołkach G i wszystkich wierzchołkach H i sprawdzamy, czy istnieje izomorfizm. Warunki C gwarantują nam unikalność wykrycia tego izomorfizmu spośród wszystkich automorfizmów H. Jeśli wykryjemy izomorfizm, zostaje on dodany do zwracanej listy izomorfizmów H w podgrafy G. Są one wyznaczane przy pomocy odpowiednich mapowań etykiet wierzchołków H i G na liczby naturalne - autorzy pracy generują takie etykietowanie, że żaden nietożsamościowy automorfizm nie spełnia warunków C - dzięki temu każde wystąpienie H w G jest zliczane jednokrotnie.

FindSubgraphInstances(H,G):
Finds all instances of query graph H in network G

Start with an empty set of instances.
[Find Aut(H). Let H_E be the equivalence representatives of H.]
[Find symmetry-breaking conditions C for H given H_E and Aut(H).]
Order the nodes of G by increasing degree and
 then by increasing neighbor degree sequence.
For each node g of G
 For each node h of H [H_E] such that g can support h
  Let f be the partial map associating $f(h) = g$.
  Find all isomorphic extensions of f [up to symmetry]
   i.e. call IsomorphicExtensions(f,H,G[,$C(h)$]).
  Add the images of these maps to the set of all instances.
 Remove g from G.
Return the set of all instances.

Rysunek 5: Pseudokod algorytmu GK
Źródło: [5]

Należy tutaj podkreślić, że gdy ograniczymy się do przechodzenia iteracyjnie przez podzbiór wierzchołków G zamiast przez wszystkie, algorytm nadal może znaleźć wszystkie wystąpienia motywów - tracimy jednak w tym momencie deterministyczność wyniku. W standardowej implementacji algorytmu MODA twórcy przechodzą iteracyjnie przez $\frac{1}{3}$ wszystkich wierzchołków sieci.

4.3 Algorytm FASCIA i *color coding*

Algorytm FASCIA [3], czyli Fast Approximate Subgraph Counting and Enumeration, jest wersją algorytmu *color coding* Alona et al. [2] zoptymalizowaną pod kątem paralelizacji obliczeń i zużycia pamięci. W projekcie używamy algorytmu FASCIA, ale do przedstawienia idei działania algorytmu wystarczy opisanie pierwotnej wersji algorytmu. Choć algorytm *color coding* działa dla wszysktich grafów o ograniczonej *treewidth*, tak twórcy FASCIA potwierdzają jej działanie jedynie dla drzew oraz grafów "drzewopodobnych" z trójkątami. To ograniczenie prawdopodobnie wynika bezpośrednio z zastosowania nowej metody indeksacji w celu uproszczenia procesu zliczania grafów, choć autorzy nie wyjaśniają tej kwestii bezpośrednio.

Color coding jest algorytmem wykorzystującym programowanie dynamiczne - rozbijamy problem na mniejsze zagadnienia i korzystając z wcześniejszych wyników aktualizujemy wyniki na wyższych poziomach. Chcemy więc znaleźć szacowaną liczbę wystąpień wzorca T w grafie G. W tym celu powtórzymy wielokrotnie eksperyment polegający na zliczeniu wystąpień poprzez losowe pokolorowanie grafu G. W algorytmie możemy wyróżnić trzy kluczowe etapy:

- wygenerowanie zbioru podwzorców $\mathcal{S}$ na bazie wzorca T,
- losowe pokolorowanie grafu G,
- zliczenie liczby wystąpień wzorca przy pomocy programowania dynamicznego.

Pierwszy etap będzie miał miejsce tylko raz, dwa pozostałe będą wykonywane wielokrotnie wewnątrz eksperymentów.

Algorithm 1 Subgraph counting using color coding.

1: Partition input template T (k vertices) into subtemplates using *single edge cuts*.
2: Determine $Niter \approx \frac{e^k \log 1/\delta}{\epsilon^2}$, the number of iterations to execute. δ and ϵ are input parameters that control approximation quality.
3: **for** $i = 1$ to $Niter$ **do**
4: Randomly assign to each vertex v in graph G a color between 0 and $k-1$.
5: Use a dynamic programming scheme to count *colorful* non-induced occurrences of T.
6: **end for**
7: Take average of all $Niter$ counts to be final count.

Rysunek 6: Idea algorytmu *color coding*
Źródło: [3]

4.3.1 Generowanie podwzorców

Do poprawnego przejścia przez etap generowania zbioru podwzorców $\mathcal{S}$, potrzebujemy założenia o drzewiastej strukturze wzorca. Inicjalizujemy $S := \{T\}$. Wybieramy z drzewa T jego korzeń v i którąś z jego krawędzi. Następnie usuwamy wybraną krawędź w wyniku czego otrzymujemy dwa nowe podwzorce - podwzorzec zawierający oryginalny korzeń oznaczamy jako *active child* i korzeń pozostaje bez zmian, a w drugim korzeniem staje się wierzchołek na drugim końcu krawędzi i podwzorzec oznaczamy jako *passive child*. Oba wzorce dodajemy do zbioru S. Procedurę dzielenia na podwzorce przeprowadzamy tak długo aż nie dojdziemy do korzeni niezawierających krawędzi. Zbiór S przechowujemy w formie *partitioning tree* [Rysunek 7], w którym trzymamy historię podziału wzorca T.

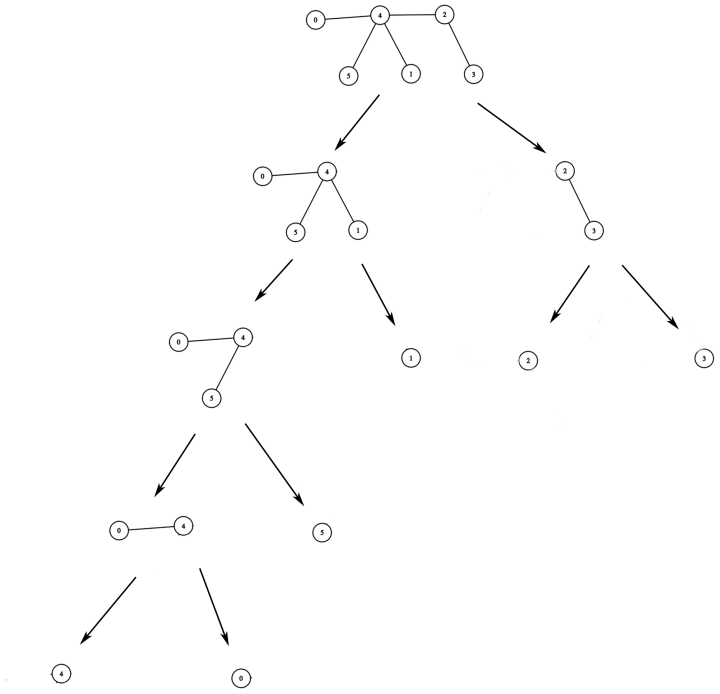
4.3.2 Sekwencja programowania dynamicznego

W pierwszej kolejności, dla każdego wierzchołka $v \in G$ przypisuje kolor pomiędzy 0 i $k-1$, przy czym k jest maksymalną liczbą kolorów. Do poprawnego działania algorytmu wymagamy, żeby k było większe lub równe liczbie wierzchołków wzorca T. Dla uproszczenia, przyjmujemy $k := |V(T)|$. Rozpatrzmy trójwymiarową tablicę counts$[\mathcal{S}][V(G)][\mathcal{C}]$ o następujących indeksach:

- zbiór podwzorców $\mathcal{S}$,
- zbiór wierzchołków G,
- zbiór możliwych kolorowań podwzorców $\mathcal{C}$.

Dla danego podwzorca S o rozmiarze h, kolorowanie podwzorca definiujemy jako mapowanie h unikalnych kolorów na każdy wierzchołek S. Rysunek 8 w szczegółowy sposób przedstawia proces zliczania w postaci zagnieżdżonych pętli. Wystąpienia poszczególnych podwzorców obliczane są w podejściu bottom-up. Pętla na zewnątrz przechodzi przez podwzorce w kolejności odwrotnej do kolejności ich powstawania - tzn. *postorder traversal* po elementach $\mathcal{S}$.

Wewnątrz, przechodzimy po wszystkich wierzchołkach $v \in G$. Jeśli podwzorzec S jest pojedynczym wierzchołkiem, wiemy, że counts$[S][v][C]$ wynosi 0 dla wszystkich k kolorowań C podwzorca S, oprócz koloru przypisanego do v, gdzie wartość wynosi 1.



Rysunek 7: Przykład struktury *partitioning tree*. Na lewo *active*, na prawo *passive*
Źródło: własne

W przeciwnym wypadku, jeśli podwzorzec S ma wielkość $h > 1$, wiemy, że posiada on $a :=$ *active child* i $p :=$ *passive child*. Z kolejności przechodzenia po podwzorcach (*postorder traversal*) wynika, że liczności wystąpień podwzorców a i p zostały już policzone i możemy z nich korzystać. Następnie przechodzimy przez każde kolorowanie $C \in \mathcal{C}$ odpowiadającemu podwzorcowi S.

Ustalamy *count* := 0. Wierzchołek v staje się korzeniem S, co oznacza, że jest też korzeniem a. Oprócz tego, kandydatem na korzeń p jest każdy wierzchołek u należący do sąsiedztwa v, ozn. $N(v)$. Kolorowanie C możemy podzielić na C_a i C_p, które są kolorowaniami i odpowiednia *active child* i *passive child*. Stąd, wartość count$[S][v][C]$ jest sumą, po wszystkich u oraz wszystkich podziałów C_a i C_p, iloczynu count$[a][v][C_a] \cdot$ count$[p][u][C_p]$.

Finalnie, sumujemy wartości count$[T][V(G)][\mathcal{C}]$ i szacujemy liczbę wystąpień T w grafie G skalując sumę przez liczbę automorfizmów T i prawdopodobieństwo, że wzorzec jest unikalnie pokolorowany.

4.3.3 Uśrednianie wyniku

FASCIA zwraca szereg szacowań, które możemy wykorzystać w różny sposób. Podstawowym jest uśrednienie tych wartości i zwrócenie ich jako szacowanej liczbie wystąpień wzorca T w grafie G.

Algorithm 2 The dynamic programming routine in FASCIA.

for all sub-templates S created from partitioning T, in reverse order they were created during partitioning **do**
  for all vertices $v \in G$ **do**
    if S consists of a single vertex **then**
      Set table$[S][v][color\ of\ v]$:= 1
    else
      S consists of active child a and passive child p
      for all colorsets C of unique values mapped to S **do**
        Set $count$:= 0
        for all $u \in N(v)$, where $N(v)$ is the neighborhood of v **do**
          for all possible combinations C_a and C_p created by splitting C and mapping onto a and p **do**
            $count$ +=
            table$[a][v][C_a]$·table$[p][u][C_p]$
          end for
        end for
        Set table$[S][v][C]$:= $count$
      end for
    end if
  end for
end for
$templateCount$ = $\sum_v \sum_C$ table$[T][v][C]$
P = probability that the template is colorful
α = number of automorphisms of T
$finalCount = \frac{1}{P \cdot \alpha} \cdot templateCount$

Rysunek 8: Pseudokod części programowania dynamicznego FASCIA
Źródło: [3]

5 Przedstawienie pomysłu

Jak przedstawiliśmy w poprzedniej sekcji, moduł mapowania w algorytmie MODA to jego słaby punkt - złożoność obliczeniowa zliczania wystąpień pojedynczego drzewa w sieci jest ponadwielomianowa, a liczba drzew mających n wierzchołków to w przybliżeniu $0.5349^{-\frac{5}{2}} \cdot 0.3383^{-n}$ [6], co oznacza jej eksponencjalny wzrost wraz ze zwiększaniem liczby wierzchołków. Krok ten jest jednak konieczny, by umożliwić wygenerowanie kandydatów na motywy niebędące drzewami. Nasz pomysł to zastosowanie algorytmu FASCIA w celu aproksymacyjnego decydowania, że dane drzewo jest motywem. Do tego celu proponujemy dwie hipotezy, które użytkownik może wykorzystać podczas zliczania wystąpień drzewa T w sieci G. Niech T_G oznacza liczbę wystąpień T w G, a N oznacza średnią liczbę wystąpień uzyskaną na zbiorze losowogenerowanych sieci.

5.1 Wariant I

$$H_0 : T_G \leqslant N$$

Dla powyższej hipotezy możemy przeprowadzić test statystyczny po każdej iteracji

algorytmu FASCIA na poziomie istotności α. Odrzucenie hipotezy zerowej będzie wskazywało, że dane drzewo **jest** motywem z dokładnością do poziomu istotności. Istnieje jednak ryzyko odrzucenia przez tę hipotezę drzew, które są motywami w sieci, ale popełniliśmy błąd statystyczny. Lista drzew, dla których odrzuciliśmy hipotezę, jest zatem zawarta w liście motywów.

5.2 Wariant II

$$H_0 : T_G \geqslant N$$

W tym wariancie odrzucenie hipotezy będzie mówiło nam, że dane drzewo **nie jest** motywem z dokładnością do poziomu istotności. Jeśli wygenerujemy listę drzew, dla których nie było podstaw do odrzucenia hipotezy, będzie to lista kandydatów na motywy - lista motywów jest w niej zawarta.

5.3 Główna idea

Najistotniejszym problemem algorytmu FASCIA jest jego niezdolność do zwracania listy wystąpień danego drzewa w sieci. W związku z tym, o ile może stanowić poważną konkurencję dla algorytmu GK do zliczania drzew, o tyle nie można go efektywnie wykorzystać do decydowania, czy dowolny graf jest motywem w sieci, ponieważ nie można bezpośrednio przekazać wyniku algorytmu FASCIA do algorytmu MODA w zastępstwie za *mapping module*. Proponujemy jednak następujące rozwiązanie, które może okazać się szybsze, niż standardowa implementacja algorytmu MODA:

1. Wygenerowanie populacji losowych sieci podobnych do G,
2. Dla każdego drzewa T w zbiorze wszystkich drzew o zadanym rozmiarze:
    - 2.1. Uzyskanie próby zliczeń występowania T w populacji losowo wygenerowanych sieci poprzez przybliżenie faktycznej próby liczbami zwracanymi przez algorytm FASCIA po niewielkiej liczbie iteracji na każdej z tych sieci,
    - 2.2. Iteracyjne wyznaczanie liczby wystąpień T w G przy pomocy algorytmu FASCIA,
    - 2.3. Jednoczesne z punktem 2.2 wykonywanie testów statystycznych i dodanie drzewa T do listy w przypadku spełnienia warunku z wariantu I lub II,
3. (Przekazanie otrzymanej listy do *mapping module* w zastępstwie za listę wszystkich drzew.)

W zależności od wybranego wariantu zmienia się narzut obliczeniowy na *mapping-module*. W przypadku pierwszego wariantu hipotezy, do algorytmu GK w algorytmie MODA przekazujemy tylko te drzewa, które są motywami według wartości statystyki testowej wyznaczonej przez algorytm FASCIA. Nie możemy po prostu przekazać tych drzew od razu do dalszego etapu algorytmu MODA - *enumerating module* - ponieważ FASCIA nie zwróci nam dokładnej listy z wystąpieniami podgrafu w sieci tak, jak robi to algorytm GK. Znacznie ograniczamy tym samym liczbę wywołań algorytmu GK, ryzykując jednak, że

nie wszystkie motywy zostaną wykryte. Jeśli zdecydujemy się na wariant drugi, do algorytmu GK przekażemy dłuższą listę drzew, maksymalizując pewność znalezienia wszystkich motywów - zwiększa to jednak koszt obliczeniowy w stosunku do wariantu pierwszego.

W trzecim punkcie znajdujemy się zatem w sytuacji, gdzie zamiast listy wszystkich drzew k-wierzchołkowych, mamy listę tylko tych z nich, które po filtracji algorytmem FASCIA zostały kandydatami na motywy. Pozwala nam to na przekazanie do algorytmu MODA ograniczonej listy drzew, tym samym zmniejszając narzut obliczeniowy na *mapping module* (algorytm GK). Nie mamy co prawda pewności, co do faktycznej poprawy działania - gdyby w naszej sieci wejściowej wszystkie drzewa były motywami, zwiększamy niepotrzebnie koszt obliczeniowy. Wydaje się jednak, że taka sytuacja wystąpi z bardzo niskim prawdopodobieństwem - w wielu przypadkach tylko część zbioru wszystkich drzew będzie motywami. Ten punkt powyższej listy jest w nawiasie, ponieważ główną częścią naszej analizy będzie porównanie dwóch wariantów *mapping module* - standardowego zliczenia wszystkich drzew algorytmem GK oraz zliczenia za pomocą algorytmu GK wyłącznie drzew przefiltrowanych algorytmem FASCIA. W ten sposób sprawdzimy, czy zaproponowane rozwiązanie faktycznie osiąga lepsze czasy wykonania operacji zliczania drzew. W przypadku potwierdzenia tych przypuszczeń można rozważać rozwiązanie z trzecim punktem w kontekście realnej próby poprawy algorytmu MODA, który obecnie jest algorytmem *state-of-the-art* wśród algorytmów wyszukujących motywy w sieciach.

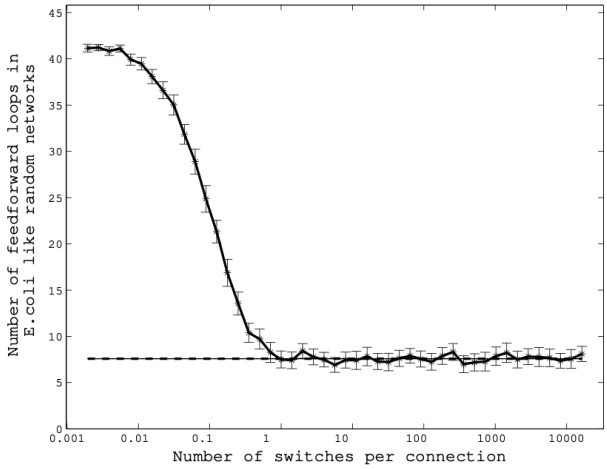
6 Generowanie sieci losowych

W celu uzyskania próby częstości występowania badanego kandydata na motyw w sieciach losowych, istotnym staje się algorytm stosowany do ich generowania. Nie może on generować grafów w pełni losowych - właściwości sieci wejściowej takie, jak np. średnica czy rozkład stopni wierzchołków, powinny być w jak największym stopniu zachowane. Na podstawie dostępnej literatury możemy wskazać kilka stosowanych zazwyczaj algorytmów:

1. Najpopularniejszą techniką generowania sieci losowych na podstawie wzorca jest "metoda zamieniania". Podstawowy krok polega na wyborze dwóch krawędzi – $(a \rightarrow b)$ i $(c \rightarrow d)$ – a następnie zamienieniu ich wierzchołków docelowych tak, by otrzymać krawędzie $(a \rightarrow d)$ i $(c \rightarrow b)$. Operacja ta zachowuje stopnie wchodzace i wychodzące wierzchołków a także liczbę krawędzi. Aby być pewnym, że wygenerowana sieć jest wystarczająco losowa, należy powtórzyć tę operację odpowiednio wiele razy.

2. Inną techniką jest "metoda dopasowywania". W tej metodzie każdy wierzchołek sieci wejściowej ma określoną ilość wchodzących i wychodzących "wypustek", o których możemy myśleć jak o "odciętych" krawędziach. W kolejnych krokach łączymy losową wypustkę wchodzącą i wychodzącą tworząc krawędź między nimi. Metoda ta zazwyczaj poprawnie generuje sieci o wymaganych własnościach, choć pewne poprawki w algorytmie są konieczne aby odwzorować fakt, że w realistycznych sieciach ilość wierzchołków o wysokim stopniu jest stosunkowo niewielka.

Na potrzeby naszego projektu zdecydowaliśmy się generować sieci losowe przy użyciu "metody zamieniania" – dotychczasowe badania nad różnicami w wynikach osiąganych za pomocą tych algorytmów [7] wskazują, że algorytm oparty na dopasowywaniu nie zawsze generuje wszystkie losowe grafy o danym rozmiarze z tym samym prawdopodobieństwem. Poza tym zakłada on, że jeśli wygenerowana krawędź spowoduje utworzenie multigrafu,



Rysunek 9: Liczba wystąpień motywu *feed-forward loop* w sieciach podobnych do sieci transkrypcyjnej E. coli w zależności od ilości wykonanych zamian na krawędź
Źródło: [7]

cała sieć jest odrzucana i algorytm zaczyna od nowa. W porównaniu - przy stosowaniu algorytmiu zamieniania nie ma pojęcia porażki, ale za to nie można być pewnym ile razy należy wykonać zamianę by graf był odpowiednio zrandomizowany. Autorzy pracy [7] wykonali jednak numeryczne testy pokazujące, że dla ok. $100 \times E$ razy, gdzie E - ilość krawędzi, rezultaty są wystarczająco dobre. Obrazowym przykładem może być przedstawiony na rysunku 9 zanik motywów w sieciach podobnych do wyjściowej, w zależności od liczby wykonanych zamian. Zaletą algorytmu jest też złożoność czasowa zależna wyłącznie od wybranej ilości zamian.

7 Metody statystyczne

Przedstawiamy kluczowe techniki statystyczne wykorzystane w naszych analizach.

7.1 Testowanie statystyczne

Na mocy centralnego twierdzenia granicznego możemy stwierdzić, że obliczone liczby wystąpień zwracane dla grafów z populacji sieci podobnych do G, jako średnie ze wyników iteracji *color codingu*, rozkładają się zgodnie z rozkładem normalnym. W kontekście naszego problemu mamy więc próbę statystyczną, której średnią chcemy porównać z określoną wartością - wynikiem iteracji algorytmem FASCIA na oryginalnej, badanej sieci. Odpowiednim testem na potrzeby sprawdzenia naszej hipotezy zerowej jest więc test t-Studenta. Test t-Studenta stosuje następującą statystykę t:

$$t = \frac{\bar{X} - m_0}{s} \cdot \sqrt{n}$$

gdzie $\bar{X}$ to średnia badanej próby, m_0 - średnia rozkładu z którego pochodzi, przy założeniu hipotezy zerowej, s - odchylenie standardowe próby oraz n - jej liczność.

Po obliczeniu wartości t można, stosując rozkład t-Studenta liczbie stopni swobody $\nu = n-1$ znaleźć prawdopodobieństwo hipotezy zerowej. Aby ją odrzucić, będziemy ocze-

kiwać, że nasza wartość t leży poniżej 0.05 kwantyla otrzymanego rozkładu (lub powyżej 0.95, zależnie od wariantu).

7.2 Przedziały ufności

Jako pomocnicze narzędzie do testowania statystycznego przedstawionego powyżej będziemy korzystać z przedziałów ufności liczby wystąpień danego grafu w sieci jawnie otrzymywanego z implementacji algorytmu FASCIA. Alon et. al. [2] w pracy o *color coding* udowodnili, że jeśli c to faktyczna liczba wystąpień grafu w sieci, a wartość zwracana przez algorytm oparty na *color coding* po N_{iter} iteracji algorytmu znajduje się w przedziale $[c(1-\varepsilon), c(1+\varepsilon)]$ z prawdopodobieństwem $1-2\delta$, to

$$\varepsilon = \sqrt{\frac{e^k \log \frac{1}{\delta}}{N_{iter}}}$$

W powyższym wzorze k to liczba wierzchołków zliczanego grafu. O ile duża odległość przedziału ufności od $\bar{N}$ naturalnie koreluje z wartościami powyżej przedstawionej statystyki, o tyle możemy wykorzystać powyższy wzór do określenia wartości maksymalnej liczby iteracji dla przypadku, gdy hipoteza nie zostanie odrzucona. Możemy heurystycznie przyjąć, że oczekujemy, aby

$$\varepsilon \leqslant q \cdot \widehat{\mathbb{E}(N)},$$

gdzie q jest pewną niewielką stałą, np. $q = \delta$, a $\widehat{\mathbb{E}(N)}$ to pewien estymator oczekiwanej liczby wystąpień grafu w sieci. W naturalny sposób możemy znaleźć co najmniej dwa estymatory:

- estymator pierwszy - średnia liczba wystąpień obliczona na wcześniej wygenerowanych sieciach losowych,
- estymator drugi - oczekiwana liczba wystąpień w grafie losowym Erdősa–Rényi'ego o tej samej liczbie wierzchołków i proporcjonalnej liczbie krawędzi co wejściowa sieć.

Estymator pierwszy mamy dany w sposób jawny, natomiast w pracy [8] możemy znaleźć, że w modelu Erdősa–Rényi'ego:

$$\mathbb{E}(N) \sim n^{k-g}(pn)^g,$$

gdzie n to liczba wierzchołków w sieci, k to liczba wierzchołków liczonego podgrafu, p to prawdopodobieństwo wystąpienia krawędzi w grafie losowym, a g to liczba krawędzi w zliczanym podgrafie. Relacja $\sim$ to relacja proporcjonalności. Z powyższego wzoru, zakładając stałą gęstość grafu losowego i wiedząc, że zliczamy drzewa, możemy wyprowadzić:

$$\mathbb{E}(N) \sim \left(\frac{2|E(G)|}{n-1}\right)^{k-1},$$

gdzie $|E(G)|$ to liczba krawędzi w sieci losowej. Możemy za tę liczbę przyjąć liczbę krawędzi w naszej sieci wejściowej, ponieważ sieci losowe są generowane na jej podobieństwo. Nie znaleźliśmy wzoru na współczynnik proporcjonalności, jednak dotychczasowe obliczenia wskazują, że jest on nieznacznie mniejszy od 1. Daje nam to zatem bezpieczny zapas i pozwala ustalić:

$$N_{iter} = \left\lceil \frac{e^k \log \frac{1}{\delta}}{\delta^2 \widehat{\mathbb{E}(N)}^2} \right\rceil$$

8 Spodziewane wyniki

Na podstawie przedstawionej wstępnej analizy algorytmów przewidujemy, że algorytm FASCIA osiągnie znacznie lepsze czasy od algorytmu GK przy zliczaniu wystąpień grafów w sieci. Choć autorzy algorytmu GK nie podają teoretycznej złożoności swojego algorytmu, a jedynie twierdzą, że jest "eksponencjalnie lepszy" od naiwnej metody, złożoność ta nadal jest znacznie ponadwielomianowa. Wstępnie przeprowadzone testy wskazują, że przy zliczaniu wystąpień nawet pojedynczego grafu o 8 wierzchołkach w zredukowanej sieci *h.pylori* - 714 wierzchołków, 1118 krawędzi - czas wykonania algorytmu przekracza godzinę. Jednocześnie, jedenaście tysięcy iteracji algorytmu FASCIA, czyli liczba gwarantująca $\varepsilon < 1$, zatem przyzwoity przedział ufności mogący stanowić pewien punkt wyjścia, wykonuje się mniej niż minutę. Tym samym liczymy, że weryfikacja motywów algorytmem FASCIA przed przekazaniem ich do algorytmu GK w celu wykorzystania w MODA może przynieść poprawę czasu obliczeń, prawdopodobnie kosztem niewielkiego zmniejszenia precyzji wykrywania motywów.

Literatura

[1] E. Wong, B. Baur, S. Quader, and C.-H. Huang, "Biological network motif detection: principles and practice," *Briefings in Bioinformatics*, vol. 13, pp. 202–215, 06 2011.

[2] N. Alon, R. Yuster, and U. Zwick, "Color-coding," *Journal of the ACM (JACM)*, vol. 42, no. 4, pp. 844–856, 1995.

[3] G. M. Slota and K. Madduri, "Fast approximate subgraph counting and enumeration," in *2013 42nd International Conference on Parallel Processing*, pp. 210–219, 2013.

[4] S. Omidi, F. Schreiber, and A. Masoudi-Nejad, "Moda: An efficient algorithm for network motif discovery in biological networks," *Genes & Genetic Systems*, vol. 84, no. 5, pp. 385–395, 2009.

[5] J. A. Grochow and M. Kellis, "Network motif discovery using subgraph enumeration and symmetry-breaking," 2007.

[6] B. Richmond, A. Odlyzko, and B. McKay, "Constant time generation of free trees," *SIAM Journal on Computing*, vol. 15, pp. 540–548, 05 1986.

[7] R. Milo, N. Kashtan, S. Itzkovitz, M. E. J. Newman, and U. Alon, "On the uniform generation of random graphs with prescribed degree sequences," 2003.

[8] S. Itzkovitz and U. Alon, "Subgraphs and network motifs in geometric networks," *Physical review. E, Statistical, nonlinear, and soft matter physics*, vol. 71, p. 026117, 03 2005.