Politechnika Wrocławska
Wydział Informatyki i Telekomunikacji

Kierunek: **Informatyka algorytmiczna (INA)**

PRACA DYPLOMOWA MAGISTERSKA

Zastosowanie szkiców danych w analizie dużych grafów

Application of data sketches in the analysis of large graphs

Paweł Polerowicz

Opiekun pracy
dr inż. Jakub Lemiesz

Słowa kluczowe: TODO, TODO, TODO

WROCŁAW, 2024

Streszczenie

Polski

Słowa kluczowe: TODO, TODO, TODO

Abstract

English

Keywords: TODO, TODO, TODO

Spis treści

Spis rysunków

Spis tabel

Spis listingów

Skróty

TODO (ang. *Uzupełnić lub usunąć ten wykaz, zależnie od potrzeb*)

Rozdział 1

Wstęp

TODO: Tutaj będzie bardzo ogólne wprowadzenie do pracy.

- *Motywacja*
- *Krótki i "wysokopoziomowy"opis problemu*
- *Podsumowanie osiągnięć pracy*

1.1. Struktura pracy

TODO: Kilka(dziesiąt) słów o strukturze i zawartości pracy. Omówienie po kolei rozdziałów i ewentualnych dodatków. Bardzo skrótowo, bo wszystko będziemy potem i tak rozwijać. Coś w jak niżej (do dopracowania).

Pierwszy rozdział stanowi niniejszy wstęp, przedstawiający ogólny zarys problematyki pracy i skrótowo podsumowujący jej wkład badawczy. W drugim rozdziale znajduje się opis problemu wraz z formalną definicją i przedstawieniem różnych jego wariantów. Przedmiotem trzeciego rozdziału jest przegląd literatury związanej z analizą wielkich grafów, z podziałem na zastosowane metodyki oraz tabelą ilustrującą porównanie znanych struktur i algorytmów. W czwartym rozdziale szczegółowo omówione zostały szkice danych, od ich formalnej definicji do bardziej praktycznych przykładów ich wykorzystania, także w kontekście niniejszej pracy. Piąty rozdział zawiera właściwy opis tego, co zostało zrobione *[TODO: przepisać nieco bardziej szczegółowo]*. W rozdziale szóstym opisane zostały przeprowadzone eksperymenty, wraz z prezentacją wyników oraz wnioskami z nich płynącymi. Ostatni, siódmy rozdział, stanowi podsumowanie pracy. Zawarte zostały w nim ogólne konkluzje na temat pracy oraz możliwe kierunki dalszych badań. Pracy towarzyszy wykaz literatury oraz dodatek, zawierający opis dołączonej płyty CD *[TODO: Na dalszym etapie sprawdzić, czy aktualne]* i instrukcję użytkowania części implementacyjnej.

Rozdział 2

Opis problemu

2.1. Analiza grafów

Analiza danych jest dynamicznie rozwijającą się dziedziną informatyki, znajdującą zastosowania w wielu gałęziach przemysłu i badaniach naukowych. Wielka różnorodność rozważanych zbiorów danych, pochodzących z odmiennych źródeł, indukuje potrzebę znajdowania wszechstronnych i efektywnych struktur danych i algorytmów, które mogą służyć do ich reprezentacji i przetwarzania. Grafy doskonale nadają się jako narzędzie do tego typu zadań ze względu na ich wrodzoną zdolność do modelowania złożonych relacji i struktur, od sieci społecznościowych i topologii Internetu po systemy biologiczne i sieci transportowe. Dzięki tej wszechstronności algorytmy grafowe znajdują dziś zastosowania w praktyce, napędzając innowacje i wspomagając przetwarzanie coraz bardziej obszernych zestawów informacji. Pomimo że grafy towarzyszą informatyce niemal od samych jej początków, to jednak rozwój tej dziedziny nie ustaje, zwłaszcza że ilość i złożoność danych stale rośnie. W dzisiejszej erze, w której rozmiary danych często osiągają ogromne rozmiary, istnieje potrzeba dostosowania metodologii opartych na grafach do bardziej efektywnego przetwarzania informacji.

W niniejszej pracy będziemy posługiwać się głównie pojęciem grafu prostego, określanego po prostu jako graf. Będziemy go oznaczać przez $G = (V, E)$ - graf, gdzie V - zbiór wierzchołków i $E \subseteq V \times V$ - zbiór krawędzi. W domyśle będziemy skupiać się na grafach ważonych.

2.2. Główne sposoby modelowania problemu

W niniejszej pracy pochylamy się nad kwestią analizy wielkich zbiorów danych, przedstawionych w postaci grafów. Jednak przed przystąpieniem do omawiania istniejących lub konstrukcji nowych rozwiązań, należy zastanowić się nad istotą problemu, z którym się mierzymy oraz wymaganiami i ograniczeniami, które proponowane algorytmy powinny spełniać. Kluczową kwestią jest więc wybór sposobu modelowania problemu. W kontekście analizy grafów możemy wyróżnić kilka ważnych i użytecznych modeli.

2.2.1. Model klasyczny

W tradycyjnej analizie grafów przyjmuje się dość prosty model, gdzie cały graf reprezentujący zbiór danych jest nam dany na wejściu do algorytmu. W praktycznych zastosowaniach jest on zazwyczaj reprezentowany przez macierz sąsiedztwa lub listę sąsiedztwa, choć istnieją również alternatywne reprezentacje, jak macierz incydencji [19]. Charakterystyczną cechą tego modelu, odróżniającą go od omawianych dalej, jest fakt, że dostępna wiedza o grafie jest pełna i dostępna w dowolnym momencie działania algorytmu. Zazwyczaj zakładamy również, że jest on

niezmienny w czasie. Jest on niewątpliwie najprostszym i jednocześnie potężnym modelem, stąd też przez dekady to na nim opierały się badania w zakresie analizy grafów. Jednak zapamiętanie całego grafu wiąże się ze sporym narzutem pamięciowym, dla macierzy i listy sąsiedztwa odpowiednio rzędu $O(|V|^2)$ i $O(|E|)$. W świecie ogromnych grafów, gdzie rozmiary analizowanych zbiorów krawędzi mogą sięgać rzędu miliardów, taka złożoność może być nieakceptowalna.

2.2.2. Strumień grafowy

W odpowiedzi na charakterystykę problemu przetwarzania ogromnych zbiorów danych powstał model strumieniowy. Graf jest w nim reprezentowany przez strumień krawędzi, napływających stopniowo. Zakładamy, że zapisanie tego grafu w klasyczny sposób jest niepraktyczne lub niemożliwe ze względu na ograniczoną pamięć. Algorytmy oparte na tym modelu powinny więc działać on-line, na bieżąco aktualizując swój stan i będąc gotowe na obsługę zapytań w dowolnym momencie. Z uwagi na rozmiar, dynamikę i często nieznaną charakterystykę danych, takie metody muszą często pomijać niektóre, mniej istotne w danym kontekście informacje o grafie, ograniczając się do tych kluczowych. W związku z tym często dopuszcza się przybliżone odpowiedzi na zapytania, jednak najlepiej z rozsądnym ograniczeniem na możliwy błąd.

Dodatkowo możemy wyróżnić kilka podkategorii w strumieniach grafu. Jeden z najważniejszych podziałów dotyczy, tego, jakiego typu zmiany mogą zachodzić w strukturze grafu. Z uwagi na tę kwestię będziemy wyróżniać dwa typy grafów. Graf nazwiemy statycznym, jeśli do grafu krawędzie są jedynie dodawane i raz ustanowione, nigdy nie znikną. W uproszczeniu możemy założyć, że graf, który badamy, jest stały i niezmienny, ale o kolejnych krawędziach dowiadujemy się stopniowo, gdy pojawiają się one w strumieniu. Z kolei grafy dynamiczne to takie, które dopuszczają szerszą gamę operacji, przede wszystkim usuwanie wcześniej istniejących krawędzi. Może to być przydatne przy reprezentowaniu szybko zmieniających się zbiorów danych, takich jak np. informacje o ruchu samochodowym czy podejrzanych aktywnościach na kontach bankowych. Strumienie grafowe będą głównym modelem rozważanym w ramach niniejszej pracy.

Definicja formalna

Niech $G = (V, E)$ - graf. Strumieniem grafowym nazywamy ciągłą sekwencję elementów, z których każdy ma postać trójki $e_i = (< s_i, d_i >; w_i, t_i)$, gdzie s_i, d_i wierzchołki grafu G i przez parę $< s_i, d_i >$ oznaczamy krawędź pomiędzy nimi. Z kolei w_i i t_i to odpowiednio waga tej krawędzi i moment jej wystąpienia. Określona krawędź może powtarzać się w różnych momentach czasowych z różnymi wagami. Zazwyczaj przyjmujemy, że wagi kolejnych wystąpień krawędzi są akumulowane. W literaturze można również spotkać nieco inne definicje, głównie różniące się dokładną postacią strumieniowanej krotki np. dla grafów dynamicznych może przybrać postać czwórki $e_i = (< s_i, d_i >; w_i, t_i, op)$, gdzie $op \in \{+, -\}$ indykuje typ operacji, a więc czy dana krawędź jest dodawana, czy usuwana z grafu[16].

2.2.3. Model półstrumieniowy TODO: Potwierdzić nazwę

Model półstrumieniowy[8] (ang. *semi-streaming model*) różni się modelu strumieniowego w dwóch głównych kwestiach. Po pierwsze, narzuca on konkretne ograniczenia na pamięć wykorzystywaną przez algorytm, najczęściej $O(|V|polylog(|V|))$, a więc dla gęstych grafów znacznie mniejszą niż rozmiar grafu. Po drugie, wejście może być skanowane wielokrotnie, zwykle stałą lub logarytmiczną liczbę razy. Model ten można uznać więc za rodzaj pomostu między klasyczną analizą grafów, w których dane znane są od początku i nie istnieją ograniczenia na dostęp do nich, a modelem strumieniowym, który nie pozwala na wielokrotne przeglądanie wcześniejszych krawędzi. Model ten jest często wybierany przez badaczy analizujących konkretne, złożone pro-

blemy grafowe takie, jak np. wyznaczanie najkrótszych ścieżek [7] lub minimalnego drzewa rozpinającego [1] przy rygorystycznych ograniczeniach pamięciowych. Podobnie jak w przypadku strumieni grafowych, możemy w tym modelu rozważać grafy statyczne i dynamiczne.

2.2.4. Model rozproszony

W wielu praktycznych zastosowaniach takich, jak analiza sieci społecznościowych, dane napływają z różnych źródeł – np. serwerów rozsianych po świecie i obsługujących różne obszary. Kolejne paczki danych są często relatywnie niezależne od siebie i mogą być rozpatrywane oddzielnie. W takich przypadkach wygodnie jest rozważać model rozproszony analizy grafów. W tym modelu dane są dzielone pomiędzy wiele węzłów obliczeniowych. Takie podejście umożliwia przetwarzanie równoległe, skracając czas obliczeń i ograniczając wielkość przesyłanych danych. Większość obliczeń jest wykonywana lokalnie, bez konieczności angażowania jednej centralnej jednostki. Komunikacja między węzłami ogranicza się do niezbędnych w danym przypadku aktualizacji, zamiast obejmować wszystkie dane. Należy pamiętać o wyzwaniach wynikających z często niepełnej wiedzy węzłów, która może utrudniać rozwiązywanie bardziej złożonych problemów. Obszar rozproszonej analizy grafów znalazł szerokie zastosowania w praktyce, czego dobrym przykładem są zaawansowane platformy ułatwiające pracę w tym modelu, takie jak Google Pregel[14], czy Apache Spark GraphX[20].

Rozdział 3

Przegląd literatury

Analiza wielkich grafów, zwłaszcza w ostatnich latach, przeżywa ogromny rozwój, budząc zainteresowanie grup badaczy z całego świata. Postępy w tej dziedzinie są naturalną odpowiedzią na potrzebę przetwarzania coraz większych zbiorów danych. Badanie interakcji w sieciach społecznościowych, zarządzanie ruchem internetowym, czy monitorowanie ruchu samochodowego to tylko niektóre z kluczowych w dzisiejszej rzeczywistości zastosowań. Grafy dobrze sprawdzają się jako modele do reprezentowania złożonych relacji między encjami, dzięki czemu odgrywają kluczową rolę w przetwarzaniu i wydobywaniu skondensowanych informacji ze strumieniowanych danych. Wybrane do tych celów algorytmy i metodologie w znacznym stopniu zależą od struktury badanych grafów, a także charakteru zapytań, które chcemy rozpatrywać. W zależności od wymagań dotyczących złożoności czasowej i pamięciowej, dokładności odpowiedzi, a także konkretnych informacji, na których zachowaniu nam zależy, inne metody mogą okazać się najlepszym wyborem. Przykładowo, odpowiedź na pytania o najkrótsze ścieżki między wierzchołkami może wymagać zapamiętania dodatkowych informacji o strukturze grafu, a więc potencjalnie użycia bardziej wyrafinowanego podejścia niż w przypadku zapytań wyłącznie o istnienie danej krawędzi.

W niniejszym przeglądzie literatury zagłębiamy się w sferę analizy dużych grafów ze szczególnym uwzględnieniem metod opartych na szkicach danych. Badamy ewoluujący krajobraz technik, algorytmów i aplikacji w tej dziedzinie, rzucając światło na metodologie stosowane w celu sprostania nieodłącznym wyzwaniom stawianym przez strumieniowe przesyłanie danych grafowych. Poprzez analizę najnowszych osiągnięć, staramy się zapewnić wgląd w znaczenie i potencjał analizy strumieni grafów w rozwiązywaniu złożonych zadań analitycznych w różnych dziedzinach, a także sformułować ogólne wnioski i wskazówki co do wyboru odpowiedniej metody do danego zastosowania. Dla lepszego ustrukturyzowania wiedzy omawiane algorytmy i struktury podzielone zostały na kilka kategorii, odpisanych dokładnie w dalszej części niniejszego rozdziału. Należy jednak pamiętać, że w niektórych przypadkach podział ten jest nieco umowny, gdyż różne podejścia i pomysły nierzadko przenikają się i inspirują wzajemnie, prowadząc do syntetycznych rozwiązań.

3.1. Streszczenia danych/Sketch synopses TODO: Potwierdzić nazewnictwo

Analiza strumieni danych jest szeroką dziedziną, nieograniczającą się oczywiście wyłącznie do danych grafowych. Istnieje wiele bardziej ogólnych, uniwersalnych metod, na których podstawie można budować rozwiązania bardziej wyspecjalizowane do konkretnych zadań. Doskonałym przykładem są streszczenia danych (*ang. Sketch synopses*). Są to kompaktowe struktury

zaprojektowane z myślą o wykorzystaniu ograniczonej ilości pamięci, umożliwiając jednocześnie aproksymację różnych statystyk i zapytań dotyczących strumienia danych, takich jak zliczanie elementów, wyznaczanie mediany, czy wykrywanie wartości odstających. Utrzymując mały szkic o stałym rozmiarze, struktury te umożliwiają analizę strumieni danych w czasie rzeczywistym bez konieczności przechowywania wszystkich elementów strumienia, co zapewnia wysoką wydajność nawet przy analizowaniu ogromnych strumieni.

Istotną w kontekście budowy dalszych rozwiązań strukturą danych jest szkic Count-Min[6]. W przeciwieństwie do wielu proponowanych wcześniej streszczeń zaprojekotwanych do badania konkretnej statystyki stanowi on dość uniwersalną strukturę, oferując wsparcie dla zapytań o częstość występowania elementu, zakresu elementów oraz iloczyn skalarny dwóch strumieni. Z kolei na ich podstawie można budować bardziej skomplikowane zapytania. Co ważne, Count-Min, choć zwraca przybliżone wyniki, to gwarantuje spełnienie pewnych założeń odnośnie dokładności. Konkretnie, wyraża się ją zazwyczaj w kontekście definiowanych przez użytkownika parametrów ϵ, δ, a więc wymagamy, aby błąd względny w odpowiedzi na zapytanie mieścił się w zakresie współczynnika ϵ z prawdopodobieństwem δ. Opiera się ona na zastosowaniu dwuwymiarowej tablicy o wymiarach $w \times d$, gdzie $w = \lceil \frac{e}{\epsilon} \rceil$ oraz $d = \lceil \ln \frac{1}{\delta} \rceil$. Nietrudno zauważyć, że zależą one od wymaganej dokładności. Struktura dalej wykorzystuje d niezależnych funkcji haszujących, mapujących elementy ze strumienia na kolumny tablicy. Każdy z napływających elementów wiąże się więc z aktualizacją jednej komórki w każdym wierszu tablicy. Wtedy, przykładowo częstotliwość elementu możemy aproksymować jako minimum z wartości odpowiadających mu komórek.

Do innych wartych wzmianki metod możemy zaliczyć np. Lossy Counting[15], ukierunkowany na wskazywanie szczególnie często występujących elementów, czy AMS[2], aproksymujący momenty częstotliwości (*ang. momenty częstotliwości*).

gSketch[24] stanowi jedną z pierwszych prób przeniesienia idei tych ogólnych metod do świata grafów. Obsługuje on proste zapytania dotyczących grafu, takie jak częstość występowania danej krawędzi w strumieniu lub gęstość wybranego podgrafu. Autorzy wychodzą od metody Count-Min, wykorzystując jej dwuwymiarową tablicę do składowania częstotliwości krawędzi. Dodatkowo, algorytm wykorzystuje próbkę testową do podziału zbioru krawędzi na podzbiory, bazując na ich częstotliwości w taki sposób, aby efektywnie wykorzystać dostępną pamięć, a następnie przetwarza właściwy strumień danych. To przetwarzanie wstępne daje cenny wgląd w charakterystykę grafu, ale należy pamiętać, że wyznaczanie reprezentacyjnej dla danego zbioru próbki nie zawsze jest trywialne. gSketch, podobnie jak Count-Min jest metodą stratną, a praktycznym wyzwaniem jest odpowiednie wyważenie parametrów tak, aby zachować balans między zużytą pamięcią a dokładnością wyników.

Choć metody takie jak gSketch mogą efektywnie zapamiętywać informacje dotyczące częstotliwości występowania krawędzi, to tracą przy tym wiedzę o strukturze grafu. Przykładowo, trudno za ich pomocą odpowiedzieć, czy istnieje ścieżka między danymi dwoma wierzchołkami. Jedną z prób rozwiązania tego problemu jest struktura gMatrix[11]. Wykorzystuje ona, podobnie jak gSketch, zasadę działania Min-Count. Jednak w tym wypadku tablica zliczająca elementy wzbogaciła się o trzeci wymiar. Konkretnie, długość i szerokość tablicy odpowiadają haszom wierzchołków, a jej głębokość związana jest ponownie z liczbą funkcji haszujących. Struktura może więc wspierać zapytania związane zarówno z krawędziami, jak i wierzchołkami. gMatrix wspierać również wykrywanie szczególnie często występujących krawędzi i wierzchołków, a więc tych, których częstotliwość przekracza dany parametr F. Wymaga to jednak, aby wybrane funkcje haszujące były odwracalne. Wtedy wystarczy wybrać komórki o odpowiednio dużych wartościach i obliczyć odwrotności haszy, aby odzyskać informacje o wierzchołkach. Podobnie, można rozważać osiągalność między wierzchołkami, wybierając krawędzie o częstotliwości występowania przekraczającej pewne F i przetwarzając je używając tradycyjnych algorytmów. Zwłaszcza w przypadku grafów o nierównej gęstości, możemy w ten sposób znacznie

ograniczyć liczbę badanych krawędzi, zachowując wciąż odpowiednie ograniczenia na prawdopodobieństwo błędu.

3.2. MDL – minimalna długość opisu

Kluczowym problemem w analizie wielkich grafów jest rozmiar danych. Zasadne wydaje się więc pytanie, czy sposób zapisu analizowanych grafów jest efektywny. W wielu przypadkach może się okazać, że sama próba zmiany modelu opisującego dane przynosi znaczne oszczędności w kwestii wykorzystanej pamięci. Metoda minimalnej długości opisu (MDL - emphang. Minimum Description Length) koncentruje się na znalezieniu najprostszego modelu, który najlepiej opisuje strukturę grafu. Techniki oparte na MDL identyfikują wzorce i kompresują graf, wybierając model, który minimalizuje całkowitą długość opisu modelu i danych przy danym modelu, ułatwiając w ten sposób wydajne przechowywanie, przesyłanie i analizę dużych grafów. Bardziej formalnie, dla danych D i rodziny dostępnych modeli MF szukamy takiego modelu $M \in MF$, który minimalizuje $L(M) + L(D|M)$, gdzie $L(M)$ i $L(D|M)$ oznaczają odpowiednio długość opisu modelu M oraz zakodowanych w nim danych D. Wiele metod opartych na MDL kompresuje dane w sposób bezstratny, co jest niewątpliwą zaletą tego modelu.

Istnieje wiele bezstratnych metod kompresujących grafy do reprezentacji o mniejszym narzucie pamięciowym. Jednak większość z nich zakładała działanie na tradycyjnej postaci grafu, gdzie dane są skończone i znane na wejściu. Rzeczywistość analizy strumieni grafowych wymaga jednak bardziej elastycznego podejścia. Jedną z pierwszych inkrementacyjnych metod kompresji grafu jest MoSSo[12]. Reprezentacja wyjściowa tej metody składa się ze zbioru superwęzłów, a więc zbiorów wierzchołków oraz superkrawędzi. Każda taka superkrawędź oznacza połączenie wszystkich wierzchołków z danego superwęzła z wierzchołkami drugiego superwęzła. Dodatkowo, częścią zapisu jest także zbiór korekt krawędzi. Mają one postać pary zbiorów $C = (C^+, C^-)$, oznaczających krawędzie, które należy dodać i usunąć, aby otrzymać prawdziwe dane. W ogólności aktualizowanie struktury dla nowych krawędzi sprowadza się do przemieszczania wierzchołków między superwęzłami w taki sposób, aby zminimalizować długość zapisu. Oczywiście sprawdzenie wszystkich możliwości byłoby kosztowne, dlatego autorzy zakładają sprawdzanie za każdym razem pewnego losowego zbioru potencjalnych wierzchołków do przemieszczania, co pozwala na znaczną oszczędność czasu, przy zachowaniu zadowalającego zużycia pamięci.

TODO: np. , SGS[13], GS4[3]

3.3. Metody oparte na modyfikacji macierzy sąsiedztwa

Jednym z najbardziej popularnych i być może najprostszym koncepcyjnie sposobem reprezentacji grafu jest macierz sąsiedztwa. Jej wiersze i kolumny odpowiadają poszczególnym wierzchołkom, a w komórkach przechowywane są wagi krawędzi pomiędzy nimi, o ile takowe krawędzie istnieją. W przypadku grafów nieważonych może to być np. wartość logiczna indykująca istnienie krawędzi lub ustalona stała. Ten sposób reprezentacji ma niewątpliwe zalety takie jak prostota implementacji i, przede wszystkim, stały czas dostępu do wag krawędzi. Z tego powodu niezaskakujący jest pomysł zachowania ogólnej zasady działania macierzy sąsiedztwa, przy jednoczesnej próbie zmniejszenia jej rozmiaru.

Jedną z pierwszych realizacji tej idei jest struktura TCM[17]. Ma ona postać macierzy o boku długości m, gdzie m jest pewną stałą. Podobnie jak w klasycznej macierzy sąsiedztwa, w jej komórkach składowane są wagi krawędzi. Zasadniczą różnicą jest natomiast sposób wyznaczania rzędu i kolumny odpowiadających danej parze wierzchołków. Są one bowiem wyznaczane przez wynik funkcji haszującej $H : V \to [1..m]$. Czas obliczania hasza jest stały, a co za tym

idzie, złożoność czasowa zapytań i dodawania nowych krawędzi również. Teoretyczna złożoność pamięciowa także jest stała i wynosi $O(m^2)$. W praktycznych zastosowaniach wybór m zależy jednak często od liczby krawędzi i przejmuje się najczęściej m rzędu $O(\sqrt{|V|})$. Dokładność rezultatów zależy od rozmiaru macierzy i może być niska ze względu na kolizje haszy. Łatwo zauważyć, że jeśli m jest istotnie mniejsze od $|V|$ to może do nich dochodzić często, co powoduje traktowanie różnych krawędzi jako kolejnych instancji tego samego połączenia. Autorzy, świadomi tego ograniczenia, proponują zastosowanie kilku parami niezależnych funkcji haszujących i stworzenie na ich podstawie wielu szkiców grafu. Przykładowo, jeśli badaną zmienną jest suma wag kolejnych instancji krawędzi między danymi dwoma wierzchołkami, to algorytm może sprawdzić odpowiednie komórki dla wszystkich szkiców, a następnie zwrócić minimalną wartość. Podejście to pozwala na analizę większych grafów niż w przypadku pojedynczego szkicu, ale ostatecznie nie rozwiązuje całkowicie problemu. Użyteczność struktury TCM w bazowej formie jest dyskusyjna, stanowi ona jednak punkt wyjściowy dla bardziej zaawansowanych rozwiązań.

Strukturą opartą na koncepcie podobnym do TCM jest *Graph Stream Sketch* (GSS)[9]. Celem autorów było stworzenie metody oferującej lepszą skalowalność dla wielkich grafów. Podobnie jak w TCM, funkcja haszująca mapuje zbiór wierzchołków na pewien mniejszy zbiór M-elementowy. Rozmiar macierzy jest natomiast równy $m, m < M$. Główną zmianą jest wprowadzenie dodatkowych cech opisujących wierzchołki. Na podstawie hasza $H(v)$ wyznaczany jest podpis wierzchołka $f(v)(0 \leq f(v) < F)$, gdzie $M = m \times F$ i $f(v) = H(v)\%F$, a także adres $h(v) = \lfloor \frac{H(v)}{F} \rfloor$. Adresy służą do wyznaczania rzędu i kolumny komórek. Komórki te mają postać krotki lub, bardziej obrazowo, kubełka, w którym przechowywana jest para podpisów wierzchołków tworzących krawędź oraz kumulatywna waga krawędzi. Przechowywanie podpisów w komórkach pozwala zredukować ryzyko kolizji haszy. Łatwo bowiem zauważyć, że nawet jeśli dwa różne wierzchołki mają taki sam adres, to istnieje duża szansa, że ich podpisy są różne. Z tego względu nowa krawędź jest dodawana do kubełka tylko w wypadku, gdy jest on pusty lub gdy istniejące w nim podpisy są zgodne z podpisami wierzchołków krawędź tą tworzących. W przeciwnym przypadku jest ona zapisywana w dodatkowym buforze, mającym postać listy sąsiedztwa pełnych haszy. Pozwala on na dodawanie nowych krawędzi z niskim ryzykiem kolizji, nawet jeśli sama macierz jest już zapełniona. Należy natomiast zauważyć, że część macierzowa struktury jest bardziej efektywna czasowo, oferując stały czas odpowiedzi na zapytanie, podczas gdy dla bufora jest on liniowy względem liczby wierzchołków. Dokładność odpowiedzi w części macierzowej zależy od długości podpisów. Potencjalnym problemem GSS jest niskie wykorzystanie pamięci w macierzy. Przy kolizji adresów nowe krawędzie mogą trafiać do bufora, mimo, że w samej macierzy pozostaje wiele pustych komórek. Aby temu zaradzić, autorzy proponują haszowanie krzyżowe (*ang. square-hashing*). Zakłada ono obliczanie dla każdego wierzchołka sekwencji niezależnych adresów. Podczas wstawiania nowych krawędzi algorytm sprawdza nie jedną komórkę macierzy, a kilka, zgodnie z sekwencją adresów i wybiera pierwszą spełniającą wymagania co do zgodności podpisów. Istnieje probabilistyczne ograniczenie na błąd względny zapytań postaci $Pr(\tilde{f}(s,d) - f(s,d)/\bar{w} > \delta) \leq \frac{|E|}{\delta m^2 4^f}$, gdzie $\tilde{f}(s,d)$ jest zwróconą sumą wag krawędzi (s,d), $f(s,d)$ jej rzeczywistą wartością, $\bar{w}$ średnią wagą krawędzi, a f długością podpisu.

Większość struktur służących podsumowujących strumieniowane grafy nie przechowuje informacji o czasie wystąpienia krawędzi. Nie wspierają one więc zapytań z zakresem czasowym, a więc np., czy dana krawędź wystąpiła w zakresie $[t, t + L)$. Tego typu zapytania mogą być kluczowe np. w przypadku analizy danych dotyczących rozprzestrzeniania się wirusów (TODO: Citation needed). Problem ten podejmuje praca proponująca strukturę Horae[5]. W jej wypadku krawędź $e_i = (< s_i, d_i >, w_i, t_i)$ jest wstawiana do komórki o adresie $(h(s_i|\gamma(t_i)), h(d_i|\gamma(t_i)))$, gdzie $\gamma(t_i) = \lfloor \frac{t_i}{gl} \rfloor$ i gl jest długością przedziałów czasowych. In-

tuicyjnie, zapytanie o pojawienie się krawędzi w zakresie czasowym $[T_b, T_e]$ moze być transformowane w sekwencję zapytań o pojedyncze zakresy, których wyniki są sumowane, a więc $Q([T_b, T_e]) = Q([T_b]), Q([T_{b+1}]), \ldots Q([T_e])$. Jednak dla takiego algorytmu złożoność czasowa jest liniowa względem liczby zakresów. Autorzy starają sie poprawić ten aspekt, zauważajac, iż przedział długości L może zostać zdekomponowany do co najwyżej $2 \log L$ podprzedziałów podsiadających dwie szczególne cechy. Po pierwsze, wszystkie zakresy czasowe w danym podprzedziale mają wspólny prefiks binarny. Po drugie, prefiksy różnych podprzedziałów mają różne długości. Z tego względu Horae zapamiętuje $O(\log(T))$ identycznych skompresowanych macierzy, gdzie T jest liczbą rozróżnialnych zakresów czasowych. Każda z nich jest utożsamiana z jedną warstwą struktury. Warstwy odpowiadają z kolei różnym długościom prefiksów. Dzięki temu zamiast wykonywać liniową względem długości przedziału czasowego liczbę zapytań, wystarczy zdekomponować przedział na podprzedziały i na ich podstawie wykonać co najwyżej jedno zapytanie na warstwę.

Metody oparte na macierzach w większości przypadków nie czynią założeń co do struktury grafu. Takie ogólne podejście oczywiście zapewnia wysoką uniwersalność, jednak w niektórych przypadkach może być nieefektywne. Przykładowo, jeśli wierzchołki w grafie są mocno zróżnicowane pod względem stopnia, a więc bardziej obrazowo, da się wyróżnić obszary gęste i rzadkie w grafie, to kolizje haszy mogą zdarzać się często. Struktura Scube[4] używa probabilistycznego zliczania do identyfikacji wierzchołków wysokiego stopnia. Przeznaczane jest dla nich więcej kubełków w macierzy niż dla wierzchołków o niskich stopniach, co pozwala bardziej efektywnie zarządzać zapełnieniem macierzy.

Metody takie jak GSS czy Horae, choć często dają przyzwoite wyniki przy odpowiednim dobraniu parametrów do badanego grafu, to ostatecznie cierpią z uwagi na ograniczoną skalowalność. Jedną z prób odpowiedzi na ten problem jest struktura AUXO[10]. Korzysta ona z macierzy przechowujących podpisy wierzchołków, podobnie jak GSS. Jednak, zamiast wstawiać nadmiarowe krawędzie do bufora o liniowym czasie dostępu, AUXO wykorzystuje wiele macierzy ustawionych w strukturę drzewa. Konkretnie, jest to binarne lub czwórkowe drzewo prefiksowe, w którego strukturę zaszyte zostały prefiksy podpisów wierzchołków. W ten sposób na każdym kolejnym poziomie drzewa podpisy przechowywane w komórkach mogą być coraz krótsze, gdyż informacja ta jest wbudowana w kształt struktury. Pozwala to osiągnąć logarytmiczny względem liczby krawędzi czas odpowiedzi na zapytania. Warto zauważyć, że złożoność pamięciowa jest ograniczona przez długość podpisów, która wyznacza maksymalną głębokość drzewa. Niemniej jednak liczba możliwych do przetworzenia krawędzi jest eksponencjalna w stosunku do liczby bitów podpisu, więc stosunkowo łatwo można dobrać wystarczające wartości. W praktycznych zastosowaniach autorzy wskazują, nieco niefortunnie, na złożoność pamięciową zbliżoną asymptotycznie do $O(|E|(1 - \log(E)))$. Jak widać, AUXO osiąga efektywność pamięciową i skalowalność kosztem zwiększenia złożoności czasowej, co może być potencjalną wadą tego rozwiązania.

3.4. Graph Spanners

TODO: np. [7], QbS [18]

3.5. Embeddings

TODO: Podział na podstawie: [21]

3.5.1. Faktoryzacja

TODO: np. GraRep, HOPE, NetMF, ProNE

3.5.2. Próbkowanie

TODO: np. DeepWalk, Node2Vec, LINE, VERSE

3.5.3. Metody oparte na sieciach neuronowych

TODO: np. DNE, DVNE, GCN, and GraphS-AGE

3.5.4. Szkice

TODO: np. NH-MF, NetHash, #GNN, NodeSketch[23], SGSketch[22].

3.6. Porównanie wspieranych operacji i ich złożoności

TODO: Tabela będzie uzupełniona i rozbudowana

Tab. 3.1: Your Table Title Here

Method	**Lossless**	**Node query**	**Edge query**	**Etc.**
Matrix-based				
Method 1	Yes/No	O(|V|)	O(|V|)	...
Method 2	Yes/No	O(|V|)	O(|V|)	...
Embeddings				
Method 3	Yes/No	X	O(|V|)	...
Method 4	Yes/No	X	X	...
MDL				
Method 5	Yes/No	O(|V|)	O(|V|)	...
Method 6	Yes/No	O(|V|)	O(|V|)	...

Rozdział 4

Szkice danych

4.1. Definicja

Rozdział 5

Główny przedmiot pracy

5.1. Motywacja

TODO

5.2. Idea

TODO

5.2.1. Przykład

TODO

5.3. Implementacja

TODO

Pseudokod

5.4. Analiza złożoności

5.4.1. Złożoność czasowa

TODO

5.4.2. Złożoność pamięciowa

TODO

5.4.3. Analiza dokładności - jeśli metoda stratna

TODO

Rozdział 6

Analiza wyników

6.1. Architektura eksperymentów

6.2. Eksperyment 1 - TODO

6.2.1. Wyniki

6.2.2. Wnioski

6.3. Eksperyment 2 - TODO

6.3.1. Wyniki

6.3.2. Wnioski

6.4. Eksperyment 3 - TODO

6.4.1. Wyniki

6.4.2. Wnioski

Literatura

[1] K. J. Ahn, S. Guha, A. McGregor. Analyzing graph structure via linear measurements. *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, Jan 2012.

[2] N. Alon, Y. Matias, M. Szegedy. The space complexity of approximating the frequency moments. *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing - STOC '96*, 1996.

[3] N. Ashrafi-Payaman, M. R. Kangavari, S. Hosseini, A. M. Fander. Gs4: Graph stream summarization based on both the structure and semantics. *The Journal of Supercomputing*, 77(3):2713–2733, 2020.

[4] M. Chen, R. Zhou, H. Chen, H. Jin. Scube: Efficient summarization for skewed graph streams. *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*, 2022.

[5] M. Chen, R. Zhou, H. Chen, J. Xiao, H. Jin, B. Li. Horae: A graph stream summarization structure for efficient temporal range query. *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, 2022.

[6] G. Cormode, S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, Apr 2005.

[7] M. Elkin, C. Trehan. Brief announcement: (1+ϵ)-approximate shortest paths in dynamic streams. *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, 2022.

[8] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, J. Zhang. On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348(2–3):207–216, 2005.

[9] X. Gou, L. Zou, C. Zhao, T. Yang. Fast and accurate graph stream summarization. *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, 2019.

[10] Z. Jiang, H. Chen, H. Jin. Auxo: A scalable and efficient graph stream summarization structure. *Proceedings of the VLDB Endowment*, 16(6):1386–1398, 2023.

[11] A. Khan, C. Aggarwal. Query-friendly compression of graph streams. *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, 2016.

[12] J. Ko, Y. Kook, K. Shin. Incremental lossless graph summarization. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, Aug 2020.

[13] Z. Ma, Y. Liu, Z. Yang, J. Yang, K. Li. A parameter-free approach to lossless summarization of fully dynamic graphs. *Information Sciences*, 589:376–394, Apr 2022.

[14] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, G. Czajkowski. Pregel. *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, Jun 2010.

[15] G. S. Manku, R. Motwani. Approximate frequency counts over data streams. *Proceedings of the VLDB Endowment*, 5(12):1699–1699, Aug 2012.

[16] A. Pacaci, A. Bonifati, M. T. Özsu. Regular path query evaluation on streaming graphs. *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, May 2020.

[17] N. Tang, Q. Chen, P. Mitra. Graph stream summarization. *Proceedings of the 2016 International Conference on Management of Data*, 2016.

[18] Y. Wang, Q. Wang, H. Koehler, Y. Lin. Query-by-sketch: Scaling shortest path graph queries on very large networks. *Proceedings of the 2021 International Conference on Management of Data*, 2021.

[19] R. J. Wilson. *Introduction to graph theory*. Prentice Hall, 2015.

[20] R. S. Xin, J. E. Gonzalez, M. J. Franklin, I. Stoica. Graphx. *First International Workshop on Graph Data Management Experiences and Systems*, Jun 2013.

[21] D. Yang, B. Qu, R. Hussein, P. Rosso, P. Cudré-Mauroux, J. Liu. Revisiting embedding based graph analyses: Hyperparameters matter! *IEEE Transactions on Knowledge and Data Engineering*, 35(11):11830–11845, Nov 2023.

[22] D. Yang, B. Qu, J. Yang, L. Wang, P. Cudre-Mauroux. Streaming graph embeddings via incremental neighborhood sketching. *IEEE Transactions on Knowledge and Data Engineering*, strona 1–1, 2022.

[23] D. Yang, P. Rosso, B. Li, P. Cudre-Mauroux. Nodesketch. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, Jul 2019.

[24] P. Zhao, C. C. Aggarwal, M. Wang. gsketch: On query estimation in graph streams. *Proceedings of the VLDB Endowment*, 5(3):193–204, 2011.