Wydział Podstawowych Problemów Techniki
Politechnika Wrocławska

Analiza strumieni danych online

Adam Wilczak
Nr indeksu: 204409

Praca magisterska napisana
pod kierunkiem
dra Jakuba Lemiesza

Politechnika Wrocławska

Wrocław 2017

Spis treści

Wstęp

W tej pracy podejmiemy temat przybliżonego zliczania unikalnych elementów w strumieniu danych w oparciu o efektywne szkice danych, a także możliwości wykorzystania tych szkiców do szacowania wyników operacji teoriomnogościowych na odpowiadających im danych.

Problem przybliżonego zliczania unikalnych elementów coraz częściej pojawia się w systemach przetwarzania danych i środowiskach OLAP (Online Analytical Data Processing). W systemach tego typu bardzo często mamy do czynienia z masywnymi strumieniami danych, których dokładna analiza wymaga dużych zasobów pamięciowych i obliczeniowych. W szczególności próba dokładnego zliczania unikalnych elementów w strumieniu jest związana z dużymi wymaganiami pamięciowymi, gdyż wymaga przechowywania wszystkich napotkanych dotychczas różnych elementów. Takie podejście wydaje się bardzo niewydajne, zwłaszcza biorąc pod uwagę liczbę urządzeń oraz aplikacji generujących dziś dane oraz to, w jak dużych ilościach i z jaką częstotliwością te dane napływają (np. logi napływające w tysiącach wpisów na sekundę). Często przechowywanie takich danych wymagałoby setek terabajtów pamięci, a przestrzeń ta będzie szybko rosła z każdym kolejnym napływającym wpisem.

W odpowiedzi na powyższy problem zaproponowano wykorzystanie algorytmów probabilistycznych, które pozawalają oszacować liczbę unikalne elementy z pewnym kontrolowanym błędem, ale mają istotnie mniejsze wymagania pamięciowe. Algorytmy te operują na tak zwanych szkicach danych będących ich skrótową reprezentacją (np. w postaci haszy wybranych elementów). Jednym z pierwowzorów tego typu algorytmów był, oparty na idei liczników probabilistycznych, algorytm *Probabilistic Counting*. Jego idea została rozwinięta w **JL**: cytowanie
algorytmie *LogLog*, a ostatecznie znalazła zastosowanie w dobrze dziś znanym algorytmie *HyperLogLog*. Al- **JL**: cytowanie
gorytm *HyperLogLog* jest obecnie wykorzystywany i rozwijany m.in. przez firmy takie jak Google czy Oracle. **JL**: cytowanie

Inna, popularna rodzina algorytmów przybliżonego zliczania jest oparta na pomyśle związanym ze staty- **JL**: cytowanie
stykami pozycyjnymi. Do tej rodziny należy m.in. znany pod kilkoma różnymi nazwami algorytm *MinCount*, **JL**: cytowanie
który szczegółowo omówimy w tej pracy. Oprócz tego istnieje jeszcze wiele innych algorytmów zliczania takich
jak np. *Multiresolution Bitmap, S-Bitmap czy MaxCount* . **JL**: jakies cytwo

Niniejsza praca ma na celu omówieniem najbardziej popularnych algorytmów przybliżonego zliczania i rozważenie możliwości wykorzystania szkiców danych na jakich się opierają do do przeprowadzania operacji teoriomnogościowych na zbiorach danych, z którymi są związane. Możliwość wykonywanie operacji teoriomnogościowych na szkicach danych jest ostatnio przedmiotem wielu badań . Operacje teoriomnogościowych **JL**: cytowania
na szkicach ma wiele zastosowań praktycznych związanych z agregacją danych i wykonywaniem zapytań na wielu szkicach. Jako przykład rozważmy problem zliczania unikatowych użytkowników odwiedzających stronę internetową. Załóżmy, że średnio stronę odwiedza około 100 osób na sekundę i dokładne zliczanie unikalnych adresów IP jest zbyt kosztowne pamięciowo. Możemy użyć jednej z metod aproksymacyjnych do stworzenia szkiców odpowiadających unikalnym użytkownikom odwiedzających stronę w ciągu każdej godziny. W przyszłości moglibyśmy być jednak zainteresowani innymi informacjami, np. ilu było unikalnych użytkowników jednego dnia, jednego mieniąca lub ilu jest użytkowników, którzy odwiedzają stronę rano i wieczorem. Wówczas możliwość wykonywania operacji sumowania, przekroju czy różnicy na szkicach umożliwiałaby wykorzystanie istniejących szkiców o małej granulacji czasowej i nie byłoby potrzeby tworzenia wielu szkiców o różnej granulacji czasowej.

Zauważmy ponadto, że możliwość wykonywania operacji teoriomnogościowych na szkicach umożliwiałaby również łatwe zliczanie unikalnych elementów w środowisku rozproszonym. Przykładowo, moglibyśmy podzielić masywny strumień danych na wiele podstrumieni przetwarzanych niezależnie na różnych węzłach klastra (osobne szkice) a następnie zagregować informacje wykorzystując operacje sumy.

Struktura pracy

Praca jest podzielona na pięć rozdziałów. W rozdziale pierwszym omówimy popularne algorytmy przybliżonego zliczania: *MinCount* oraz *HyperLogLog*, a także związane z nimi szkice danych. W rozdziale drugim omówimy podstawowe techniki umożliwiające wykonywanie teoriomnogościowych na szkicach danych. W rozdziale trzecim omówimy metodę *estymatora ważonego* przedstawioną w [11] i umożliwiającą wykonywanie operacji sumy i przekroju na szkicach związanych z algorytmem *MinCout*. Pokażemy także, jak można tę metodę uogólnić dla szkiców związanych z algorytmu *HyperLogLog* oraz jak można ją zastosować do oszacowania rozmiaru różnicy szkiców. W rozdziale czwartym przedstawimy i omówimy wyniki eksperymentów, a także przyjrzymy się problemowi wyboru optymalnej kolejności wykonywania operacji na wielu szkicach.

c czy powyz-
tury pracy jest
em faktycznym

Algorytmy przybliżonego zliczania

W tym rozdziale opiszemy problem zliczania oraz trzy algorytmy przybliżonego zliczania, na których będzie się opierała dalsza cześć pracy: `MinCount`, `Streaming MinCount` oraz `HyperLogLog`. Omówimy ich działanie, związane z nimi szkice danych, a także obciążenie i koncentrację opartych na tych szkicach estymatorów. Przedstawimy również pełny kod algorytmów.

2.1 Problem zliczania

Najpierw zdefiniujmy czym jest problem zliczania, którego dotyczy nasza praca. Zacznijmy od zdefiniowania pojęcia *multizbioru*. Multizbiór $\mathfrak{M}$ definiujmy jako parę (S, m), gdzie S jest zbiorem nazywanym *zbiorem fundamentalnym* , natomiast m jest funkcją postaci $m : S \to \mathbb{N}_{\geqslant 1}$. Wartość $m(s)$ nazywamy mnogością elementu $s \in S$. Problem zliczania możemy zatem sformułować w taki sposób: mając multizbiór $\mathfrak{M}$, znaleźć moc n zbioru fundamentalnego S.

W ogólnym przypadku, nie posiadając żadnych informacji na temat danych, aby znaleźć dokładne rozwiązanie tego problemu potrzebujemy liniowej pamięci $O(n)$. Zatem, próba zmniejszenia zapotrzebowania pamięci algorytmu poniżej tego poziomu musi mieć wpływ na jego dokładność i skutkuje zmniejszeniem dokładności ostatecznego wyniku. Na szczęście jeśli jesteśmy w stanie kontrolować ten błąd i jest on stosunkowo niewielki - wówczas taki algorytm jest wystarczający w większości praktycznych zastosowań i daje nam satysfakcjonunjący wynik.

2.2 Algorytm MinCount

Pierwszym algorytmem przybliżonego zliczania, którym się zajmiemy jest `MinCount` znany również pod innymi nazwami, np. `K-th Minimum Value` (KMV) [5]. Algorytm ten opiera się na statystykach pozycyjnych.

Statystyki pozycyjne

Rozważmy zmienne losowe $X_1, X_2, \ldots, X_n$. Statystykami pozycyjnymi będziemy nazywać zmienne losowe $X_{(1)}, X_{(2)}, \ldots, X_{(n)}$ powstałe przez posortowanie realizacji zmiennych $X_1, X_2, \ldots, X_n$ rosnąco. Zmienną $X_{(k)}$ nazywamy k-tą statystyką pozycyjną. W szczególności $X_{(1)} = \min\{X_1, X_2, \ldots, X_n\}$. W dalszej części pracy będziemy zakładać, że zmienne $X_1, X_2, \ldots X_n$ są niezależne i każda ma rozkład jednostajny na odcinku (0,1).

Przyjmijmy, że istnieje funkcja haszująca

$$h \colon \mathfrak{M} \to (0, 1) \tag{2.1}$$

taka, że jeśli $\mathfrak{M}$ posiada n unikalnych elementów $a_1, a_2, \ldots a_n$ to przyjmując, że $U_i = h(a_i)$ otrzymamy ciąg niezależnych zmiennych losowych o rozkładzie jednostajnym $U_1, U_2, \ldots U_n \sim U(0,1)$. Zauważmy również, że jeśli element a_i pojawia się w $\mathfrak{M}$ wielokrotnie, to zawsze będzie zhaszowany do tej samej wartości.

Rozważmy statystyki pozycyjne $U_{(1)}, U_{(2)}, \ldots, U_{(n)}$ powstałe przez posortowanie realizacji $U_1, U_2, \ldots U_n$. O zmiennej losowej $U_{(k)}$ wiemy, że pochodzi z rozkładu $Beta(\alpha, \beta)$, gdzie $\alpha = k, \beta = n + 1 - k$ i jest zdefiniowana przez następującą funkcję rozkładu prawdopodobieństwa [4]:

$$f(x; \alpha, \beta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha, \beta)}, \tag{2.2}$$

gdzie

$$B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha + \beta)}$$

przy czym funkcja $\Gamma(z)$ jest nazywana *gammą Eulera* i definiuje się ją jako:

$$\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} dt$$

Łatwo pokazać, że wartość oczekiwana zmiennej losowej X pochodzącej z rozkładu $Beta(\alpha, \beta)$ jest funkcją stosunku parametrów α i β:

$$E[X] = \int_0^1 x f(x; \alpha, \beta) dx = \int_0^1 x \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha, \beta)} dx = \frac{\alpha}{\alpha + \beta}. \tag{2.3}$$

Stąd mamy

$$E[\frac{1}{U_{(k)}}] = \int_0^1 \frac{1}{x} f(x; k, n+1-k) dx = \frac{n}{k-1}. \tag{2.4}$$

Estymator liczności

Wykorzystując formułę (2.4) oraz metodę momentów, możemy zdefiniować estymator liczności $\hat{n}$ naszego zbioru wejściowego $a_1, a_2, \ldots a_n$ jako:

$$\hat{n} = \frac{k-1}{U_{(k)}} \tag{2.5}$$

Pokażemy teraz, że estymator $\hat{n}$ jest estymatorem nieobciążonym, gdy $k \geqslant 2$:

$$\begin{aligned} E[\hat{n}] &= \int_0^1 \frac{k-1}{x} f(x; k, n+1-k) dx && (2.6) \\ &= (k-1) \int_0^1 \frac{1}{x} f(x; k, n+1-k) dx && (2.7) \\ &= (k-1) \frac{n}{k-1} = n && (2.8) \end{aligned}$$

Podobnie, możemy policzyć wariancję estymatora [4]:

$$\begin{aligned} Var[\hat{n}] &= E[\hat{n}^2] - E[\hat{n}]^2 && (2.9) \\ &= \frac{k-1}{k-2} n(n-1) - n^2 && (2.10) \\ &= \frac{(k-1)n(n-1) - n^2(k-2)}{k-2} && (2.11) \\ &= \frac{n(n-k+1)}{k-2} && (2.12) \end{aligned}$$

Przy czym, dla $n \to \infty$ możemy zapisać:

$$Var[\hat{n}] \approx \frac{n^2}{k-2} \tag{2.13}$$

Szkic danych

Zdefiniujemy szkic danych algorytmu `MinCount`. Szkic określamy jako parę (S, τ), gdzie S to zbiór k najmniejszych (i unikalnych) haszy spośród wszystkich wartości $h(\mathfrak{M})$. Natomiast τ to wartość k-tej statystyki pozycyjnej, czyli największa wartość w zbiorze haszy S.

Algorytm `MinCount`

Algorytm `MinCount` opisany w poniższym pseudokodzie działa w następujący sposób. Początkowo w szkicu mamy $S = \emptyset$ i $\tau = 0$. Jeśli w chwili pojawienia się nowego elementu wejściowego x ze strumienia $\mathfrak{M}$ rozmiar $|S| < k$ wówczas dodajemy x do S i ustalamy τ jako największą wartość z S. Jeśli rozmiar $|S| \geqslant k$ to porównujemy hasz $h(x)$ z aktualną wartością τ szkicu. Jeżeli jest mniejsza, to dodajemy $h(x)$ do zbioru S, usuwamy z niego dotychczasowe τ i ustalamy nowe.

W oparciu o szkic (S, τ) jesteśmy w stanie w dowolnym momencie wyliczyć wartość estymatora liczności

$$\hat{n} := \frac{k-1}{\tau}.$$

Algorithm 1 Algorytm `MinCount`

```
k - liczba przechowywanych haszy
S - zbiór przechowywanych haszy
τ - największy przechowywany hasz
h - funkcja haszująca elementy 𝔐 w odcinek (0, 1)

function Add(x ∈ 𝔐)
    e ← h(x)
    if e ∉ S then
        if S.size < k then
            S.add(e)
            τ ← max{S}
        else if e < τ then
            S.remove(τ)
            S.add(e)
            τ ← max{S}
        end if
    end if
end function

function Estimate
    if S.size < k then
        return S.size
    else
        return (k − 1)/τ
    end if
end function
```

2.3 Algorytm Streaming MinCount

Jednym z mniej znanych algorytmów opartych na statystykach pozycyjnych, który będziemy rozważać w dalszej części pracy jest algorytm `Streaming MinCount` [10]. Algorytm ten różni się od klasycznego `MinCounta` sposobem estymacji liczności. Wykorzystuje on taki sam szkic, ale wykonuje tzw. *estymację kroczącą*. Tzn. wykorzystuje do estymacji również informacje o zmianach w szkicu, a nie tylko końcową postać szkicu:

$$\hat{n} = \sum_{t \in T} \frac{Z_t}{\tau_t} \tag{2.14}$$

gdzie τ_t to wartość największego spośród k przechowywanych haszy po przetworzeniu t elementów $\mathfrak{M}$, a Z_t jest zmienna binarną przyjmującą wartość 1 jeśli szkic zmienił się w momencie t lub 0 w p.p. Jak widać ten estymator operuje na przestrzeni czasu t, tzn. każdy kolejny napotkany element ze strumienia wejściowego inkrementuje t. Dzięki takiemu podejściu, estymator zmienia wartość tylko wtedy, gdy hasz nowo napotkanego elementu modyfikuje szkic. Wykorzystanie dodatkowych informacji o zmianach w szkicu powoduję zmniejszenie wariancji estymatora o połowę względem klasycznego algorytmu `MinCount` [11]. Analiza tego estymatora wraz ze szczegółowym opisem i dowodem własności znajduje się w pracy [10]. Poniżej przedstawiamy pseudokod dla algorytmu `Streaming MinCount`:

Algorithm 2 Algorytm `Streaming MinCount`

k - liczba przechowywanych haszy
S - zbiór przechowywanych haszy
τ - największy przechowywany hasz
h - funkcja haszująca elementy $\mathfrak{M}$ w odcinek $(0, 1)$
$\hat{n} \leftarrow 0$ - estymator liczności

function ADD($x \in \mathfrak{M}$)
    $e \leftarrow h(x)$
    if $e \notin S$ **then**
        if $S.size < k$ **then**
            $S.add(e)$
            $\tau \leftarrow max\{S\}$
            $\hat{n} \leftarrow \hat{n} + (1/\tau)$
        else if $e < \tau$ **then**
            $S.remove(\tau)$
            $S.add(e)$
            $\tau \leftarrow max\{S\}$
            $\hat{n} \leftarrow \hat{n} + (1/\tau)$
        end if
    end if
end function

function ESTIMATE
    return $round(\hat{n})$
end function

2.4 Algorytm HyperLogLog

Trzecim algorytmem przybliżonego zliczania, który omówimy w tej pracy jest `HyperLogLog` zaprezentowany w pracy [8]. Podobnie jak w algorytmie `MinCount` haszuje on elementy wejściowego multizbioru $\mathfrak{M}$ w odcinek (0,1). Jednak tym razem hasze są traktowane nie jak liczby w zapisie binarnym, a jak ciągi złożone z zer i jedynek. Algorytm na bieżąco śledzi maksymalną liczbę zer wiodących wśród wszystkich haszy. Idea algorytmu jest następująca: przy założeniu, że na każdej pozycji zero i jedynka pojawia się z jednakowym prawdopodobieństwem, hasze które zawierają więcej zer wiodących są rzadziej spotykane. Wskazują zatem na większą liczność zbioru wejściowego. Innymi słowy, jeśli ciąg bitów postaci $0^{q-1}1$ pojawi się na początku hasza, wówczas przy założeniu, że funkcja h zwraca wartości zgodnie z rozkładem jednostajnym, możemy estymować liczbę różnych elementów multizbioru wejściowego na 2^q.

Takie podejście oparte na pojedynczym eksperymencie posiada jednak dużą wariancję. Aby zmniejszyć warinację stosuję się technikę znaną jako stochastyczne uśrednianie (ang. *stochastic averaging*). Strumień wejściowy $\mathfrak{M}$ dzielony jest na m mniejszych podstrumieni $\mathfrak{M}_i$ podobnych rozmiarów. O tym, do którego podstrumienia należeć będzie dany element decyduje p początkowych bitów z jego hasza, gdzie $m = 2^p$. Następnie te p początkowych bitów jest usuwane z $h(x)$ i dla każdego $x \in \mathfrak{M}_i$ wyznaczana jest pozycja q pierwszej jedynki z tak powstałego $h(x)$. Największe ze znalezionych pozycji q w każdym podstrumieniu przetrzymywane są w tablicy rejestrów M, tzn. w $M[i]$ jest największą wartość q dla podstrumienia $\mathfrak{M}_i$:

$$M[i] = \max_{x \in \mathfrak{M}_i} Q(x) \tag{2.15}$$

gdzie $Q(x)$ jest funkcją zwracającą pozycję pierwszej jedynki w haszu elementu x. Korzystając z tych rejestrów algorytm wylicza estymację liczby różnych elementów $\hat{n}$ jako średnią harmoniczną, skorygowaną o współczynniki α_m usuwający obciążenie estymatora:

JL: sprawdzi wszystko dobrze q+1, Q w całyn pseudokodzie

$$\hat{n} = \alpha_m m^2 (\sum_{i=1}^{m} 2^{-M[i]}), \tag{2.16}$$

gdzie

$$\alpha_m = (m \int_0^\infty (\log_2(\frac{2+u}{1+u}))^m du)^{-1}. \tag{2.17}$$

Można pokazać, że dla $n \to \infty$, wartość oczekiwana dla powyższego estymatora ma postać (zobacz [8]):

$$E[\hat{n}] = n(1 + \delta_1(n) + o(1)), \tag{2.18}$$

gdzie $|\delta_1(n)| < 5 \times 10^{-5}$ dla $m \geqslant 16$, wariancja natomiast wynosi:

$$Var[\hat{n}] = (n(\frac{b_m}{\sqrt{m}} + \delta_2(n) + o(1)))^2, \tag{2.19}$$

gdzie $|\delta_2(n)| < 5 \times 10^{-4}$ dla $m \geqslant 16$, a stała b_m jest ograniczona i wynosi odpowiednio dla:

- $b_{16} \approx 1.106$,
- $b_{32} \approx 1.070$,
- $b_{64} \approx 1.054$,
- $b_{128} \approx 1.046$,
- $b_\infty = \sqrt{3 \log 2 - 1} \approx 1.03896$.

Funkcje $\delta_1(n)$ oraz $\delta_1(n)$ są funkcjami oscylującymi o małej amplitudzie i mogą zostać bezpiecznie pominięte w zastosowaniach praktycznych.

Wyniki eksperymentalne pokazały jednak, że przedstawiona powyżej wersja algorytmu nie sprawdza się dla wszystkich zakresów wartości n, dlatego twórców wprowadzili do algorytmu pewne poprawki:

1. Poprawka dla małych liczności - symulacje przeprowadzone przez autorów algorytmu wykazały, że jeśli liczba rożnych elementów $n \leqslant \frac{5}{2}m$, to pojawiają się istotne zaburzenia. Dlatego dla tego zakresu liczności n autorzy sugerują użycie algorytmu `Linear Counting` [6].

2. Poprawka dla dużych liczności - gdy wartość n zbliża się do $2^{32} \approx 4 \times 10^9$, kolizje haszy stają się coraz bardziej prawdopodobne (jeśli używamy standardowo 32-bitowej funkcji haszującej). Aby temu zapobiec zastosowano również stosowną korekcję. Jeśli wartość estymatora $\hat{n}$ jest większa niż $\frac{2^{32}}{30}$ wówczas jego ostateczna wartość zostaje zastąpiona przez:

$$\hat{n} = -2^{32} log(1 - \frac{\hat{n}}{2^{32}}) \tag{2.20}$$

Szczegółowo algorytm oraz powyższe korekcje zostały opisane przez twórców w [8] wraz ze wszystkimi własnościami.

Szkic danych

Szkicem danych algorytmu `HyperLogLog` jest kolekcja rejestrów $M[i]$ przechowujących największą napotkaną pozycję pierwszej jedynki w haszach podstrumienia $\mathfrak{M}_i$, gdzie $i = 1...m$, $m = 2^p$ jest parametrem wyznaczającym liczbę rejestrów, a więc sterującym dokładnością algorytmu. Szkic ten możemy w naturalny sposób wykorzystać do wyznaczenia estymatora `Linear Counting` potrzebnego do zastosowania korekcji 1. Wystarczy, że zliczymy ile spośród rejestrów jest równych 0 (oznaczmy liczbę pustych rejestrów przez V) i zastosujemy wzór:

$$\hat{n} = mlog(\frac{m}{V}) \tag{2.21}$$

Poniżej przedstawiamy pseudokod algorytmu `HyperLogLog` z zastosowaniem opisanych wcześniej korekcji:

Algorithm 3 Algorytm `HyperLogLog`

$m \leftarrow 2^p$ - liczba rejestrów, gdzie $p \in \mathbb{N}_+$
α_m - współczynnik korygujący obciążenie
$q(s)$ - funkcja zwracając pozycję pierwszej jedynki w ciągu bitów s
$h(x)$ - funkcja haszująca $\mathfrak{M}$ w $(0, 1)$
M - kolekcja m rejestrów

for $i = 1..m$ **do**
  $M[i] \leftarrow 0$
end for

function ADD($x \in \mathfrak{M}$)
  $e \leftarrow h(x)$
  $idx \leftarrow 1 + \langle e_1, e_2, ..., e_b \rangle_2$
  $v \leftarrow e_{b+1}, e_{b+2}, ...$
  $M[idx] \leftarrow max\{M[idx], q(v)\}$
end function

function ESTIMATE
  $\hat{n} \leftarrow \alpha_m m^2 (\sum_{j=0}^{m} 2^{(-M[j])})^{-1}$
  if $\hat{n} \leqslant \frac{5}{2} m$ **then**
    $V \leftarrow$ liczba rejestrów $M[i]$ równych 0
    if $V > 0$ **then**
      $\hat{n} \leftarrow m \log(\frac{m}{V})$
    end if
  else if $\hat{n} > \frac{1}{30} 2^{32}$ **then**
    $\hat{n} \leftarrow -2^{32} \log(1 - \frac{\hat{n}}{2^{32}})$
  end if
  return $\hat{n}$
end function

Operacje teoriomnogościowe

W tym rozdziale wprowadzimy pojęcie *podobieństwa Jaccarda* zbiorów oraz przedstawimy algorytm `MinHash` służący do .... Następnie omówimy podstawowe metody estymacji liczności zbiorów powstałych w wyniku wykonania operacji teoriomnogościowych w przypadku, gdy nie są znane bezpośrednio zbiory na jakich wykonywane są te operacje, a jedynie ich szkice. **JL**: uzupełnić

Podobieństwo Jaccarda

Podobieństwo Jaccarda definiujemy jako:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}. \tag{3.1}$$

Definicję tę możemy także uogólnić na n zbiorów [7]:

$$J(A_1, A_2, ..., A_n) = \frac{|A_1 \cap A_2 \cap ... \cap A_n|}{|A_1 \cup A_2 \cup ... \cup A_n|}. \tag{3.2}$$

W tej pracy zaproponujemy wykorzystanie do tego, jako narzędzia pomocniczego - algorytmu `MinHash`, który wykorzystywany jest m.in. do estymacji podobieństwa *Jaccarda* dwóch zbiorów danych.

Algorytm MinHash

W tym podrozdziale opiszemy działanie algorytmu `MinHash` oraz wyjaśnimy w jaki sposób możemy go zastosować do ... . Algorytm `MinHash` korzysta z techniki tzw. *min-wise hashing*. Schemat ten został po raz pierwszy zaproponowany przez Andrei Brodera [2], jako narzędzie do określania podobieństwa stron internetowych. **JL**: uzupełnić

Niech h będzie funkcją haszującą, która mapuje elementy ze zbiorów na liczby. Dla dowolnego zbioru X zdefiniujmy $h_{min}(X)$ jako minimalny element z X względem funkcji h , to jest taki element $x \in X$ dla którego $h(x)$ osiąga najmniejszą wartość. **JL**: czy to ma i nie, ze calkowite

Zastosujemy funkcję h_{min} na zbiorach A i B. Zakładając że nie wystąpią żadne kolizje haszy, otrzymamy dokładnie tę samą wartość, jeśli element x należący do sumy zbiorów $A \cup B$ osiągający minimalną wartość $h(x)$ znajduje się także w przecięciu tych zbiorów $A \cap B$. Prawdopodobieństwo zajścia takiego zdarzenia jest równe właśnie indeksowi *Jaccarda* tych zbiorów:

$$Pr(h_{min}(A) = h_{min}(B)) = J(A, B) \tag{3.3}$$

Łatwo zatem pokazać, że jeśli I jest zmienną losową przyjmującą wartość 1 gdy $h_{min}(A) = h_{min}(B)$ i 0 w przeciwnym przypadku, wówczas I jest nieobciążonym estymatorem dla $J(A, B)$ [9].

Ponieważ zmienna I ma zbyt dużą wariancję, aby być przydatnym estymatorem indeksu *Jaccarda*, główną ideą algorytmu `MinHash` jest zmniejszenie tej wariancji poprzez użycie estymatora będącego średnią z wielu niezależnych eksperymentów tego typu. Najprostsza wersja algorytmu zakłada użycie k różnych funkcji haszujących, gdzie k to ustalony parametr. Innymi słowy, każdy zbiór X jest reprezentowany przez k wartości $h^{(1)}_{min}(X), h^{(2)}_{min}(X), \ldots, h^{(k)}_{min}(X)$ policzonych przez k funkcji haszujących. Estymator $J(A, B)$ w tej wersji algorytmu wygląda następująco:

$$\hat{J}(A, B) = \frac{1}{k} \sum_{i=1}^{k} [h^{(i)}_{min}(A) = h^{(i)}_{min}(B)], \tag{3.4}$$

gdzie $[x]$ jest notacją *Iversona* dla zdarzenia x, zdefiniowaną jako

$$[x] = ...$$

Tak zdefiniowany estymator jest nieobciążonym estymatorem $J(A, B)$ [9], a jego wariancja wynosi:

$$Var(\hat{J}(A,B)) = \frac{J(A,B)(1 - J(A,B))}{k} \tag{3.5}$$

zamto wyprowa-

sie czy wtedy

e jest potrzebne

najpierw przed-

dla dwóch zbio-

ogołnił

Ten wzór można łatwo wyprowadzić:... w podobny sposób jak wzór (4.24). Estymator ten jest zmienną losową będącą sumą k niezależnych prób *Bernoulliego* postaci $[h^{(i)}_{min}(A) = h^{(i)}_{min}(B)]$, a jak wiemy ze wzoru (4.38) prawdopodobieństwo sukcesu takiej jednej próby jest równe $J(A, B)$.

Istnieje również drugi, mniej znany wariant algorytmu `MinHash`, który będziemy wykorzystywać w dalszej części tego rozdziału. Przedstawimy go od razu dla indeksu *Jaccarda* dla wielu zbiorów (4.38). Używa on analogicznego estymatora jak (4.39), ale korzysta z tylko jednej funkcji haszującej. Zamiast generować k różnych funkcji haszujących, przechowujemy k najmniejszych wartości dla każdego z porównywanych zbiorów $A_1, A_2, ..., A_n$ i korzystamy z tylko jednej funkcji haszującej (analogicznie jak w algorytmie `MinCount`) [7]. Posiadamy zatem zbiór haszy, który możemy traktować jako losową próbkę z $\bigcup A_i$. Następnie dla k najmniejszych wartości z tej próbki zliczamy ile z nich znajduje się również wśród k najmniejszych wartości w próbkach poszczególnych zbiorów. Całość dzielimy przez k i otrzymujemy alternatywną wersję estymatora indeksu *Jaccarda* [7]:

$$\hat{J}(A_1, A_2, ..., A_n) = \frac{|min_k\{\bigcup min_k\{h(A_i)\}\} \cap (\bigcap min_k\{h(A_i)\})|}{k} \tag{3.6}$$

duże H we wzo-

gdzie $h(A_i)$ oznacza zbiór zhaszowanych elementów zbioru A_i, a $min_k\{S\}$ jest funkcją zwracającą k najmniejszych elementów ze zbioru S (w naszym przypadku są to hasze).

3.1 Naiwna estymacja operacji teoriomnogościowych

Omówimy teraz naiwne metody wyników estymacji operacji teoriomnogościowych na szkicach, skupimy się na operacji sumy i przekroju. Dla niektóre szkiców danych operacja sumy może być zdefiniowana w stosunkowo naturalny sposób. Dla przykładu, łatwo zauważyć, że oszacowanie liczności dla sumy dwóch zbiorów A_1 i A_2 przy użyciu szkiców M_1 i M_2 związanych z algorytmem `HyperLogLog` sprowadza się do stworzenia nowego szkicu $M_{A_1 \cup A_2}$, takiego, że

$$M_{A_1 \cup A_2}[i] = \max ....$$

wzor

Najważniejszą własnością powyższej operacji jest to, że powstały szkic jest identyczny ze szkicem, który powstałby z sumy zbiorów wejściowych $A_1 \cup A_2$ oraz fakt, że suma dwóch szkiców daje w wyniku nowy szkic. Oznacza to że operacja sumy jest zamknięta i pozawala m.in. na sumowanie ze sobą sekwencyjnie większej liczby szkiców. Bardziej formalnie, możemy zdefiniować taką operację $\dot{\cup}$, że dla dwóch dowolnych zbiorów A, B:

$$S(A_1) \dot{\cup} S(A_2) = S(A_1 \cup A_2), \tag{3.7}$$

znaczek $\hat{\cup}$ na $\dot{\cup}$

gdzie S to funkcja generująca szkic danych dla zbioru.

W przeciwieństwie do operacji sumy, dla szkicach rozważanych przez nas algorytmów nie istnieje naturalna operacja przekroju. Istnieją metody umożliwiające oszacowanie mocy przekroju, ale nie zwracają one wyniku nowego szkicu, tak jak operacja sumy. Jedną z takich podstawowych metod jest zastosowanie zasady *włączeń i wyłączeń*:

$$|A_1 \cap A_2| = |A_1| + |A_2| - |A_1 \cup A_2| \tag{3.8}$$

i skorzystanie z wcześniej wyznaczonej estymacji sumy do policzenia estymacji przekroju. Metodę tę można również wykorzystać do oszacowania różnicy zbiorów:

$$|A_1 \setminus A_2| = |A_1 \cup A_2| - |A_2|. \tag{3.9}$$

Estymatę przekroju można również policzyć korzystając z podobieństwa *Jaccarda*. Poniżej podajemy zestawienie estymatorów operacji sum i przekroju zdefiniowanych zgodnie z naiwnym podejściem opisanym powyżej:

$$\hat{N}(S_1 \hat{\cup} S_2) = \hat{N}(S(S_1 \cup S_2)) \tag{3.10}$$

$$\hat{N}(S_1 \hat{\cap} S_2) = \hat{N}(S(A_1)) + \hat{N}(S(A_2)) - \hat{N}(S(S_1 \cup S_2)) \tag{3.11}$$

$$\tag{3.12}$$

a także z wykorzystaniem podobieństwa *Jaccarda*:

$$\hat{N}_1(S_1 \hat{\cap} S_2) = \hat{J}(S(A_1), S(A_2))\hat{N}(S(S_1 \cup S_2)) \tag{3.13}$$

$$\hat{N}_2(S_1 \hat{\cap} S_2) = \frac{\hat{J}(S(A_1), S(A_2))}{1 + \hat{J}(S(A_1), S(A_2))}(\hat{N}(S(A_1)) + \hat{N}(S(A_2))) \tag{3.14}$$

Powyższe wzory można uzasadnić następująco:

Powyższe metody estymacji nie są jednak zbyt dokładne . Zwłaszcza w przypadku przekroju, gdzie błąd jest z mniej więcej proporcjonalny do wielkości sumy lub większego zbioru, a oczekiwalibyśmy błędu ograniczony rozmiarem mniejszego zbioru. W skutek użycia powyższych formuł często dochodzi również do anomalii w wyniku których oszacowanie liczności jest ujemne. Dlatego w dalszej części pracy zajmiemy się analizą innych metod szacowania wyników operacji teoriomnogościowych, które dają dokładniejsze wyniki i pozbawione są tego rodzaju anomalii. Ponadto, przynajmniej w przypadku algorytmu `MinCount`, pozawalają na zdefiniowanie zamkniętej operacji przekroju.

JL: uzupełnić, rami

JL: cytowanie

3.2 HyperLogLog

3.2.1 Operacja sumy

W poprzedniej sekcji wyjaśniliśmy, że algorytm `HyperLogLog` posiada naturalną, zamkniętą operację sumy teoriomnogościowej na swoich szkicach. Jest ona wyjątkowo nieskomplikowana i sprowadza się do znalezienia maksimum na każdym z rejestrów spośród sumowanych szkiców [3]. Mając dwa szkice `HyperLogLog-a` rozmiaru m: $M_1 = (M_{11}, ..., M_{1m})$ oraz $M_2 = (M_{21}, ..., M_{2m})$ reprezentujące dwa zbiory A_1 oraz odpowiednio A_2, procedura tworząca szkic $M_u = (M_{u1}, ..., M_{um})$ reprezentujący sumę $A_1 \cup A_2$ wygląda następująco:

$$M_{ui} = max(M_{1i}, M_{2i}) \quad \textbf{dla} \quad i = 1,...,m. \tag{3.15}$$

Pamiętajmy, że w ten sposób możemy sumować szkice o tych samych parametrach p i q (patrz rozdział:...). Parametr p kontroluje błąd względny, natomiast q określa zakres wartości dla rejestrów. Suma $p+q$ określa liczbę używanych bitów haszu i definiuje tym samym maksymalna liczność jaką możemy wyznaczyć. Warto zauważyć, że jeśli liczność zbioru zbliża się do 2^{p+q} kolizje haszy stają się coraz częstsze i błąd znacznie wzrasta. Istnieje jednak możliwość sumowania dwóch szkiców o różnych parametrach parach (p, q) oraz (p',q'), albowiem każdy szkic `HyperLogLog` opisany przez parę (p, q) może zostać zredukowany do szkicu opisanego przez (p', q'), jeśli spełniony jest warunek $p' \leqslant p$ oraz $p' + q' \leqslant p + q$. Taka transformacja jest bezstratna, tj. powstały szkic jest taki sam jak szkic który powstałby przez dodawanie tych wszystkich elementów od początku do szkicu opisanego przez (p', q') [3].

JL: ujednolicić ów z tym co w p cji

JL: uzupelnic ro te parametry op

JL: opisac dokla dukacja sie odby

3.2.2 Operacja przekroju i różnicy

O ile w przypadku operacji sumy na szkicach `HyperLogLog` sprawdza się całkiem dobrze i jest łatwa w implementacji, to niestety nie są znane naturalne i zamknięte operacje przekroju oraz różnicy na tych szkicach. Oczywiście stosunkowo łatwo możemy estymować liczność przekroju zbiorów korzystając z zasady włączeń i wyłączeń lub podobieństwa Jaccarda. O ile metoda oparta na zasadzie włączeń i wyłączeń nie sprawdza się w tym przypadku i prowadzi do anomalii, to metoda wykorzystująca podobieństwo *Jaccarda* (2.19) do estymacji przekroju jest stosunkowo dokładna . Wymaga ona jednak dodatkowej struktury danych pozwalającej na estymację indeksu *Jaccarda* zbiorów. Częstą praktyką jest wykorzystanie do tego algorytmu `MinHash` [7]. Wyniki związane z tym podejściem zostaną jeszcze omówione w kolejnym rozdziale.

nczyłem spraw-

JL: to nie jest do
wolywania sie d

JL: cytowanie,
sie na eksperym

3.3 MinCount

3.3.1 Operacja sumy

Przyjrzyjmy się teraz operacji sumy dla algorytmu `MinCount`. Załóżmy że posiadamy dwa szkice danych (S_1, τ_1) oraz (S_2, τ_2). Chcemy otrzymac szkic (S_u, τ_u) będący sumą tych dwóch szkiców. Naturalną intuicją jest wykonanie sumy zbiorów k najmniejszych haszy S_1 oraz S_2 i utworzenie z nich nowego zbioru S_u zawierającego k najmniejszych haszy z sumy S_1 i S_2 oraz wyznaczenie nowego τ_u. Takie podejście jednak odrzuca dużą ilość istotnych informacji wśród zbiorów haszy ze względu na ograniczenie w postaci parametru k. Posiadając $2k$ haszy ze zbiorów S_1 i S_2 odrzucamy połowę z nich - prowadzi to nawet do dwukrotnego zwiększenia wariancji estymatora. Weźmy jako przykład dwa zbiory A_1 oraz A_2, które są rozłączne i tej samej liczności. Najlepszym estymatorem sumy jest po prostu $\hat{N}(S_1 \cup S_2) = \hat{N}(S_1) + \hat{N}(S_2)$. Jego wariancja wynosi $\frac{|A_1|^2+|A_2|^2}{k} = \frac{|A_1 \cup A_2|^2}{2k}$. Jednak nasza operacja sumy odrzuca k haszy co powoduje dwukrotny wzrost wariancji do $\frac{|A_1 \cup A_2|^2}{k}$ [11].

W pracy [11] zaproponowana została prosta zmiana dla tej operacji, która w rezultacie zamiast ograniczać nowo powstały szkic do k wartości, tworzy największy możliwy szkic, przez co nie tracimy żadnych istotnych informacji. Oznaczmy przez $\tau(S)$ największy przechowywany hasz w szkicu S, przez $h(S)$ zbiór haszy przechowywanych w szkicu S oraz $h(S, \tau)$ niech będzie zbiorem haszy w szkicu S których wartości są mniejsze bądź równe τ. Nowy operator sumy na szkicach `MinCount` wygląda następująco:

$$\tau_{min} = \tau(S_1 \cup S_2) = min(\tau_1, \tau_2) \tag{3.16}$$

$$h(S_1 \cup S_2) = h(S_1, \tau_{min}) \cup h(S_2, \tau_{min}) \tag{3.17}$$

W tej modyfikacji algorytmu odrzucamy te wartości które są większe niż wartość graniczna τ_{min}, a szkice są następnie łączone poprzez sumę teoriomnogościową na zbiorach pozostałych haszy.

Powstały szkic jest identyczny ze szkicem wielkości $|h(S_1 \cup S_2)|$ skonstruowanym z elementów ze zbioru $A_1 \cup A_2$. Estymator liczności dla tak stworzonego szkicu zdefiniowany jest następująco:

$$\hat{N}_{impr}(S_1 \cup S_2) = \frac{|h(S_1 \cup S_2)| - 1}{\tau_{min}} \tag{3.18}$$

3.3.2 Operacja przekroju

Rozpatrzmy teraz operację przekroju dla algorytmu `MinCount`. W naiwnym podejściu wykorzystujemy zasadę włączeń i wyłączeń, jak zostało to pokrótce opisane we Wprowadzeniu, ale ta metoda nie pozawala nam na zdefiniowanie zamkniętego operatora przekroju, a jedynie na wyznaczenie estymacji liczności przekroju i jest podatna na anomalie oraz posiada duży błąd.

Możemy jednak zdefiniować operator przekroju w podobny sposób, jak zdefiniowaliśmy ulepszony operator sumy w poprzednim podrozdziale. Tak jak poprzednio tworzymy - zamiast szkicu ograniczonego do k wartości - największy możliwy szkic. Nowy operator przekroju na szkicach `MinCount` wygląda następująco [11]:

$$\tau_{min} = \tau(S_1 \cup S_2) = \tau(S_1 \cap S_2) = min(\tau_1, \tau_2) \tag{3.19}$$

$$h(S_1 \cap S_2) = h(S_1, \tau_{min}) \cap h(S_2, \tau_{min}) \tag{3.20}$$

Jedną z największych zalet takiej definicji jest fakt, że otrzymujemy tutaj operator zamknięty, czyli taki który w wyniku operacji przekroju na dwóch szkicach daje w wyniku nowy szkic, a nie tylko estymatę liczności. Estymator liczności dla tak zdefiniowanej operacji przekroju wygląda tak:

$$\hat{N}_{impr}(S_1 \cap S_2) = \frac{|h(S_1 \cap S_2)| - \alpha(S_1, S_2)}{\tau_{min}} \tag{3.21}$$

gdzie

$$\alpha(S_1, S_2) = \begin{cases} 1 & \text{jeśli } \tau_{min} \in h(S_1 \cap S_2) \\ 0 & \text{w p.p.} \end{cases}$$

3.4 Estymacja metodą Największej Wiarygodności

Estymacja liczności zbioru jest problemem estymacji parametru - dzięki takiemu sformułowaniu problemu, możemy ustalić dwa bardzo ważne fakty: przydatne informacje w szkicu są zakodowane przez *wystarczające statystyki*, a estymator *największej wiarygodności* (maximum likelihood estimator) jest asymptotycznie wydajnym estymatorem [11]. Mimo, iż metoda estymacji z użyciem *estymatora największej wiarygodności* jest metodą optymalną, nie będziemy się nią zajmować w tej pracy. Ze wszystkich metod omawianych przez nas i przedstawionych w pracy [11] jest ona najtrudniejsza do efektywnego policzenia i implementacji w praktyce. Metoda omówiona i przetestowana przez nas jest dużo prostsza w implementacji oraz pozwala na łatwiejsze rozszerzenie jej na inne rodzaje szkiców oraz na operacje teoriomnogościowe na większej ilości zbiorów. Jednocześnie estymatory wyprowadzone tą metodą są niemal tak dokładne jak estymatory wyprowadzone metodą *estymatora największej wiarygodności* [11]. Mowa tutaj o metodzie *estymatora ważonego*, która została szczegółowo omówiona w rozdziale 4.

Estymacja metodą *estymatora największej wiarygodności* może być również zastosowana w kontekście algorytmu `HyperLogLog`. Taki pomysł został przedstawiony i szczegółowo omówiony w pracy [3]. Jednak podobnie jak w przypadku algorytmu `MinCount` jest on trudny do efektywnego zaimplementowania, między innymi ze względu na konieczność maksymalizacji wielowymiarowej funkcji. W dalszej części naszej pracy przyjrzymy się generalizacji metody *estymatora ważonego* dla algorytmu `HyperLogLog`, która jest szybsza i dużo łatwiejsza oraz omówimy jej efektywność.

Metoda estymatora ważonego

W tym rozdziale zajmiemy się inną metodą estymacji operacji teoriomnogościowych opisaną w [11]. Jest to metoda w której tworzone są *re-weighted estimators*, które dalej w tej pracy będziemy nazywać *estymatorami ważonymi*. Jest to metoda łatwiejsza w implementacji niż metody korzystające z *pseudo-likelihood* oraz jest ona łatwiejsza w generalizacji na inne rodzaje szkiców, m.in. szkice algorytmu `HyperLogLog`, jak również dla operacji na większej liczbie zbiorów. Te estymatory są tworzone poprzez wyznaczenie średniej ważonej kilku nieobciążonych estymatorów liczności (nazywanych estymatorami składowymi). Opiszemy w jaki sposób należy zdefiniować estymatory składowe aby metoda sprawdzała się prawie tak dobrze jak metoda oparta na estymatorze *największej wiarygodności*, oraz jak należy optymalnie *ważyć* estymatory. W dalszej części tego rozdziału dotyczącej szkiców algorytmu `MinCount` zakładamy użycie wersji algorytmu `Streaming MinCount` opisanej w rozdziale 2.3.

4.1 Optymalne ważenie estymatorów

Na początku wyjaśnimy na czym polega ważenie estymatorów. W tym celu przypomnimy czym jest *macierz kowariancji*. Macierz ta jest uogólnieniem pojęcia wariancji na przypadek wielowymiarowy. Dla wektora zmiennych losowych $(X_1, X_2, ..., X_n)$ ma ona postać:

$$\Sigma = \begin{bmatrix} {\sigma_1}^2 & \sigma_{12} & \cdots & \sigma_{1n} \\ \sigma_2 1 & {\sigma_2}^2 & \cdots & \sigma_{2n} \\ \vdots & \cdots & \ddots & \vdots \\ \sigma_n 1 & \sigma_{n2} & \cdots & \sigma_n^2 \end{bmatrix}$$

gdzie:

1. ${\sigma_i}^2 = Var(X_i)$
2. $\sigma_{ij} = Cov(X_i, X_j)$

Warto dodać, że macierz kowariancji Σ jest macierzą symetryczną, a jej wyznacznik macierzy jest nieujemny (jeżeli wektor zmiennych losowych jest niezdegenerowany).

Zgodne z lematem 3 [11], posiadając ciąg zgodnych estymatorów $\hat{N}_1, ..., \hat{N}_n$ o nieosobliwej macierzy kowariancji Σ, optymalne wagi które sumują się do 1 i minimalizują wariancję są proporcjonalne do $\Sigma^{-1} 1_n$, gdzie 1_n to wektor jednostkowy długości n. Powstały estymator jest nieobciążony (bądź zgodny) o wariancji ${1_n}^T \Sigma^{-1} {1_n}^{-1}$. W praktyce jednak macierz kowariancji Σ rzadko jest znana, dlatego potrzebna nam jest jakaś jej aproksymacja. Inną metodą ważenia jest potraktowanie estymatorów jako zmiennych losowych niezależnych i użycie przekątnej macierzy kowariancji, czyli wariancji tych zmiennych losowych. Wówczas, *estymator ważony* ma następującą postać:

$$\hat{N}_w = \sum_{i=1}^{n} z \frac{\hat{N}_i(S)}{Var(\hat{N}_i(S)} \tag{4.1}$$

gdzie z to stała normalizacyjna, zapewniająca, że wagi sumują się do jedynki [11].

4.2 Estymatory składowe

Omówimy teraz metodę na tworzenie estymatorów składowych. Naszym celem jest stworzenie estymatorów składowych, które są jak najmniej nawzajem ze sobą skorelowane, oraz których wariancję jesteśmy w stanie zaproksymować. Jednym z pomysłów, który został przedstawiony w [11] jest zdefiniowanie estymatorów, z których każdy z nich wykorzystuje tylko jeden estymator liczności. Tym sposobem, końcowy estymator ważony jest tak naprawdę liniową kombinacją estymatorów liczności `MinCount`.

Aby zdefiniować estymatory składowe zauważmy że zarówno liczność sumy dwóch zbiorów jak ich przekroju możemy zapisać jako stosunek liczności sumy (analogicznie - przekroju) do liczności jednego z tych zbiorów razy liczność tegoż zbioru. Zdefiniujmy:

$$\alpha_i = \frac{|A_1 \cap A_2|}{|A_i|} \tag{4.2}$$

$$\beta_i = \frac{|A_1 \cup A_2|}{|A_i|} \tag{4.3}$$

Wówczas tak jak zauważyliśmy, liczności sumy oraz przekroju możemy zapisać jako:

$$|A_1 \cap A_2| = \alpha_i|A_i| = \frac{|A_1 \cap A_2|}{|A_i|}|A_i| \tag{4.4}$$

$$|A_1 \cup A_2| = \beta_i|A_i| = \frac{|A_1 \cup A_2|}{|A_i|}|A_i| \tag{4.5}$$

Zdefiniujmy teraz estymatory dla α_i oraz β_i. Najprostszymi estymatorami będą:

$$\hat{\alpha_i} = \frac{|h(S_1 \cap S_2)|}{|h(S_i, \tau_{min}|} \tag{4.6}$$

$$\hat{\beta_i} = \frac{|h(S_1 \cup S_2)|}{|h(S_i, \tau_{min}|} \tag{4.7}$$

Zgodnie z lematem 4. [11] tak zdefiniowane estymatory są estymatorami zgodnymi. Estymator nazywamy zgodnym jeśli prawdopodobieństwo, że jego wartość będzie bliska wartości szacowanego przez niego parametru, wzrasta wraz z podniesieniem liczności próby [1].

Posiadając estymatory dla α_i i β_i oraz estymatory liczności pojedynczych szkiców $\hat{N}(S_i)$ możemy zdefiniować estymatory składowe w następującej postaci:

$$\hat{N}_i(S_1 \cap S_2) = \hat{\alpha_i}\hat{N}(S_i) \tag{4.8}$$

$$\hat{N}_i(S_1 \cup S_2) = \hat{\beta_i}\hat{N}(S_i) \tag{4.9}$$

Są to zgodne estymatory dla, odpowiednio, $|A_1 \cap A_2|$ oraz $|A_1 \cup A_2|$ [11].

4.3 Wariancja estymatorów składowych

Żeby można było zastosować metodę *estymatorów ważonych* potrzebujemy znać ich wariancje, a przynajmniej ich przybliżone wartości. W tym celu korzystamy z *Centralnego Twierdzenia Granicznego* i zakładamy że zachodzi ono dla $\hat{N}(S_i)$ [11]. Przypomnimy teraz w skrócie treść tego twierdzenia.

Centralne Twierdzenie Graniczne

Centralne Twierdzenie Graniczne jest twierdzeniem rachunku prawdopodobieństwa, za pomocą którego możemy w wielu sytuacjach założyć, że zmienna losowa, którą modelujemy dane zjawisko, ma rozkład mocno zbliżony do rozkładu normalnego.
Niech $X_1, ..., X_n$ będzie losowa próbą rozmiaru n - t.j. sekwencją niezależnych zmiennych losowych o jednakowym rozkładzie prawdopodobieństwa o wartości oczekiwanej μ i skończonej wariancji σ^2. Ustalmy

$S_n = \frac{X_1+...+X_n}{n}$ jako średnią naszej próby. Wówczas gdy $n \to \infty$, zmienna losowa postaci $Z = \sqrt{n}(S_n - \mu)$ jest zbieżna według rozkładu do rozkładu normalnego $N(0, \sigma^2)$. Przydatność twierdzenia wynika z faktu, iż zmienna Z dąży do rozkładu normalnego niezależnie od rozkładu poszczególnych zmiennych losowych X_i. Stosując *Centralne Twierdzenie Graniczne* dla naszego estymatora $\hat{\alpha}_i\hat{N}(S_i)$ (oraz analogicznie dla $\hat{\beta}_i\hat{N}(S_i)$) otrzymamy [11]:

$$\sqrt{k_i}(\hat{\alpha}_i\hat{N}(S_i) - n_i\alpha_i) \to N(0, \sigma^2) \tag{4.10}$$

Dokonajmy teraz dekompozycji estymatora składowego, tak aby mnożnik ($\hat{\alpha}$ lub $\hat{\beta}$) oraz estymatę liczności $\hat{N}$ można było traktować jako zmienne losowe praktycznie niezależne. Możemy zapisać:

$$\sqrt{k_i}(\hat{\alpha}_i\hat{N}(S_i) - n_i\alpha_i) = \sqrt{k_i}\hat{\alpha}_i(\hat{N}(S_i) - n_i) + \sqrt{k_i}(\hat{\alpha}_i - \alpha_i)n_i \tag{4.11}$$

Teraz możemy znaleźć wariancję estymatora $\hat{N}(S_i)$ oraz mnożnika $\hat{\alpha}_i$. Skorzystamy z dobrze znanego wzoru $Var(X+Y) = Var(X)+Var(Y)+2Cov(X,Y)$. Ponieważ zmienne te są ujemnie skorelowane [11], a składnik $2Cov(X,Y)$ jest pomijalnie mały, możemy zapisać to równanie w formie nierówności $Var(X+Y) \leqslant Var(X)+Var(Y)$:

$$Var(\sqrt{k_i}(\hat{\alpha}_i\hat{N}(S_i) - n_i\alpha_i)) \leqslant \tag{4.12}$$

$$Var(\sqrt{k_i}\hat{\alpha}_i(\hat{N}(S_i) - n_i) + \sqrt{k_i}(\hat{\alpha}_i - \alpha_i)n_i) \leqslant \tag{4.13}$$

$$Var(\sqrt{k_i}\hat{\alpha}_i(\hat{N}(S_i) - n_i)) + Var(\sqrt{k_i}(\hat{\alpha}_i - \alpha_i)n_i) \leqslant \tag{4.14}$$

$$k_i\hat{\alpha}_i{}^2Var(\hat{N}(S_i) - n_i) + k_i\hat{n}_i{}^2Var(\hat{\alpha}_i - \alpha_i) \leqslant \tag{4.15}$$

$$k_i\hat{\alpha}_i{}^2Var((\hat{N}(S_i)) + k_i\hat{n}_i{}^2Var(\hat{\alpha}_i) \tag{4.16}$$

Ponieważ wariancja dla $\hat{N}(S_i)$ jest nam znana [10], musimy jedynie znaleźć wariancje dla $Var(\hat{\alpha}_i)$ oraz $Var(\hat{\beta}_i)$. możemy je przybliżyć przy pomocy takich formuł [11]:

$$Var(\hat{\alpha}_i) \approx \frac{\alpha_i(1-\alpha_i)}{|h(S_i, \tau_{min})|} \tag{4.17}$$

$$Var(\hat{\beta}_i) \approx \frac{\beta_i(\beta_i - 1)}{|h(S_i, \tau_{min})|} \tag{4.18}$$

Wzór na $Var(\hat{\alpha}_i)$ można intuicyjnie wytłumaczyć w następujący sposób. Estymator $\hat{\alpha}_i$ jest tak naprawdę serią $|h(S_i, \tau_{min})|$ niezależnych prób *Bernoulliego*, gdzie z prawdopodobieństwem α_i losujemy element należący do przekroju $A_1 \cap A_2$. Prawdopodobieństwo zdarzenia przeciwnego wynosi $1 - \alpha_i$. Oznaczając pojedynczą próbę jako X_i, oraz przyjmując dla uproszczenia zapisu $z = |h(S_i, \tau_{min})|$ możemy zapisać:

$$\hat{\alpha}_i = \frac{1}{z}\sum_{i=1}^{z} X_i \tag{4.19}$$

Nakładając wariancję otrzymujemy:

$$Var(\frac{1}{z}\sum_{i=1}^{z} X_i) = \tag{4.20}$$

$$\frac{1}{z^2}Var(\sum_{i=1}^{z} X_i) = \tag{4.21}$$

$$\frac{1}{z^2}\sum_{i=1}^{z} Var(X_i) = \tag{4.22}$$

$$\frac{1}{z^2}\sum_{i=1}^{z} \alpha_i(1-\alpha_i) = \tag{4.23}$$

$$\frac{\alpha_i(1-\alpha_i)}{z} \tag{4.24}$$

W analogiczny sposób możemy wyznaczyć wariancję dla β_i

Tak wyliczone przybliżenia wariancji możemy teraz zastosować do wzoru (4.16) co daje nam takie przybliżenie wariancji dla $\hat{N}_i(S_1 \cap S_2)$, które możemy następnie użyć do *ważenia* tego estymatora.

$$Var(\sqrt{k_i}(\hat{\alpha_i}\hat{N}(S_i) - n_i\alpha_i)) \leqslant k_i\hat{\alpha_i}^2 Var((\hat{N}(S_i)) + k_i\hat{n_i}^2 Var(\hat{\alpha_i}) \tag{4.25}$$

$$k_i Var((\hat{\alpha_i}\hat{N}(S_i) - n_i\alpha_i)) \leqslant k_i\hat{\alpha_i}^2 Var((\hat{N}(S_i)) + k_i\hat{n_i}^2 Var(\hat{\alpha_i}) \tag{4.26}$$

$$Var((\hat{\alpha_i}\hat{N}(S_i))) \leqslant \hat{\alpha_i}^2 Var((\hat{N}(S_i)) + \hat{n_i}^2 Var(\hat{\alpha_i}) \tag{4.27}$$

$$Var((\hat{\alpha_i}\hat{N}(S_i))) \leqslant \hat{\alpha_i}^2 \frac{\hat{n_i}^2}{2k_i} + \hat{n_i}^2 \frac{\hat{\alpha_i}(1-\hat{\alpha_i})}{|h(S_i, \tau_{min})|} = \tag{4.28}$$

$$\hat{n_i}^2(\hat{\alpha_i}^2 \frac{1}{2k_i} + \frac{\hat{\alpha_i}^2(1-\hat{\alpha_i})}{\hat{\alpha_i}|h(S_i, \tau_{min})|}) \tag{4.29}$$

$$\hat{n_i}^2\hat{\alpha_i}^2(\frac{1}{2k_i} + \frac{(1-\hat{\alpha_i})}{\hat{\alpha_i}|h(S_i, \tau_{min})|}) \tag{4.30}$$

$$\hat{Var}(\hat{N}_i(S_1 \cap S_2)) \approx \hat{N}_i(S_1 \cap S_2)^2(\frac{1}{2k_i} + \frac{(1-\hat{\alpha_i})}{\hat{\alpha_i}|h(S_i, \tau_{min}|}) \tag{4.31}$$

Wyprowadzenie dla $\hat{N}_i(S_1 \cup S_2)$ wygląda analogicznie:

$$\hat{Var}(\hat{N}_i(S_1 \cup S_2)) \approx \hat{N}_i(S_1 \cup S_2)^2(\frac{(\hat{\beta_i} - 1)}{\hat{\beta_i}|h(S_i, \tau_{min}|} + \frac{1}{2k_i}) \tag{4.32}$$

4.4 Aproksymacja różnicy zbiorów metodą estymatora ważonego

Jednym z otwartych tematów w pracy [11] jest wyprowadzenie estymatora składowego dla operacji różnicy zbiorów. W tym krótkim podrozdziale zdefiniujemy taki estymator, który następnie możemy wykorzystać w metodzie *estymatora ważonego* do aproksymacji różnicy dwóch szkiców $\hat{N}_i(S_1 \setminus S_2)$.
Zauważmy, że różnicę dwóch zbiorów A_1 i A_2 możemy przedstawić jako przekrój pierwszego zbioru z dopełnieniem drugiego z nich:

$$A_1 \setminus A_2 = A_1 \cap {A_2}^c \tag{4.33}$$

A co za tym idzie, moc różnicy dwóch zbiorów możemy przedstawić w taki sposób:

$$|A_1 \setminus A_2| = \frac{|A_1 \setminus A_2|}{|A_i|}|A_i| = \frac{|A_1 \cap {A_2}^c|}{|A_i|}|A_i| \tag{4.34}$$

Zdefiniujmy zatem $\gamma_i = \frac{|A_1 \cap {A_2}^c|}{|A_i|} = \frac{|A_1 \setminus A_2|}{|A_i|}$ oraz jego estymator:

$$\hat{\gamma_i} = \frac{|h(S_1 \setminus S_2)|}{|h(S_i, \tau_{min}|} \tag{4.35}$$

Zauważmy że całą nasza przestrzenią Ω jest po prostu suma haszy z S_1 oraz S_2, więc aby wyznaczyć $|h(S_1 \setminus S_2)|$ wystarczy że odrzucimy ze zbioru haszy $h(S_1)$ wszystkie hasze znajdujące się w $h(S_2)$. Korzystając z faktu (4.34) możemy przybliżyć $Var(\hat{\gamma_i})$ takim samym wzorem jak dla $\hat{\alpha_i}$, czyli:

$$Var(\hat{\gamma_i}) \approx \frac{\gamma_i(1-\gamma_i)}{|h(S_i, \tau_{min})|} \tag{4.36}$$

Ostatecznie, przeprowadzając analogiczne obliczenia jak w (4.31), otrzymujemy wzór na wariancję estymatora różnicy postaci:

$$\hat{Var}(\hat{N}_i(S_1 \setminus S_2)) \approx \hat{N}_i(S_1 \setminus S_2)^2(\frac{1}{2k_i} + \frac{(1-\hat{\gamma_i})}{\hat{\gamma_i}|h(S_i, \tau_{min}|}) \tag{4.37}$$

4.5 Metoda estymatora ważonego dla algorytmu HyperLogLog

Jak dotąd omówiliśmy zastosowanie metody *estymatora ważonego* w kontekście algorytmu `MinCount`. W tym podrozdziale przedstawimy pomysł generalizacji tej metody na szkice algorytmu `HyperLogLog`. Potrzebne będą nam do tego estymatory mnożników α_i oraz β_i. Niestety nie jesteśmy w stanie policzyć ich korzystając bezpośrednio z informacji przechowywanych w szkicu HyperLogLog-a. Zauważmy jednak że zarówno α_i jak i β_i możemy wyrazić za pomocą podobieństwa *Jaccarda*:

$$\alpha_i = \frac{|A_1 \cap A_2|}{|A_i|} = J(A_1 \cap A_2, A_i) \tag{4.38}$$

$$\beta_i = \frac{|A_1 \cup A_2|}{|A_i|} = J(A_i, A_1 \cup A_2,)^{-1} \tag{4.39}$$

Zatem aby znaleźć estymatory dla α_i i β_i, możemy posłużyć się estymatorami dla odpowiadających im indeksów *Jaccarda*.

JL: zakomentow i minhash i prze czatek

Zastosowanie w algorytmie HyperLogLog

Pomysłem zaproponowanym w tej pracy jest wykorzystanie algorytmu `MinHash` do policzenia estymatorów mnożników α_i oraz β_i, t.j:

$$\hat{\alpha_i} = \hat{J}(A_1 \cap A_2, A_i), \tag{4.40}$$

$$\hat{\beta_i} = \hat{J}(A_i, A_1 \cup A_2,)^{-1}. \tag{4.41}$$

Ich wariancję możemy przybliżyć używając następujących wzorów. Dla $\hat{\alpha_i}$ będzie to po prostu wariancja estymatora $\hat{J}(A_1 \cap A_2, A_i)$, którą znamy dla algorytmu `MinHash`:

$$Var(\hat{\alpha_i}) = \frac{\alpha_i(1-\alpha_i)}{k} \approx \frac{\hat{J}(A_1 \cap A_2, A_i)(1 - \hat{J}(A_1 \cap A_2, A_i))}{k} \tag{4.42}$$

Wyznaczenie wariancji dla $\hat{\beta_i}$ nie jest już zupełnie trywialne, ponieważ estymator ten jest odwrotnością indeksu *Jaccarda*. Skorzystamy zatem z techniki znanej czasem jako *delta metoda* , pozwalającej na oszacowanie wariancji funkcji zmiennej losowej, przy odpowiednich założeniach.

JL: (zobacz: cy ”On Delta-Me ments and Prob J.Cichonia)

$$Var(Y) \approx Var(f(X)) = (f'(E[X]))^2 Var(X) \tag{4.43}$$

Oznaczmy przez $b_i = \frac{1}{\beta_i} = J(A_i, A_1 \cup A_2)$, oraz niech $f(x) = \frac{1}{x}$. Wówczas $\beta_i = f(b_i)$. Podstawiając do wzoru (4.43) otrzymujemy:

$$Var(\hat{\beta_i}) \approx Var(\frac{1}{b_i}) = ((\frac{1}{b_i})')^2 Var(b_i) = \tag{4.44}$$

$$(-\frac{1}{{b_i}^2})^2 \frac{b_i(1-b_i)}{k} = \tag{4.45}$$

$$\frac{1}{{b_i}^3} \frac{(1-b_i)}{k} = \tag{4.46}$$

$${\beta_i}^3 \frac{(1-\frac{1}{\beta_i})}{k} = \tag{4.47}$$

$$\frac{{\beta_i}^2(\beta_i - 1)}{k}, \tag{4.48}$$

$$Var(\hat{\beta_i}) \approx \frac{{\beta_i}^2(\beta_i - 1)}{k} \approx \frac{\hat{J}(A_i, A_1 \cup A_2)^2(\hat{J}(A_i, A_1 \cup A_2) - 1)}{k}. \tag{4.49}$$

Aby zastosować schemat *estymatora ważonego* potrzebna nam będzie jeszcze wariancja estymatora $\hat{N}_i^{HLL}(S)$ wyliczonego przez algorytm `HyperLogLog`. Jak przedstawiliśmy we wzorze (2.12), wariancja estymatora wynosi:

$$Var[\hat{N}_i^{HLL}(S)] = (n(\frac{b_m}{\sqrt{m}} + \delta_2(n) + o(1)))^2 \tag{4.50}$$

Ponieważ $\hat{N}_i^{HLL}(S)$ jest estymatorem asymptotycznie nieobciążonym [8], a czynnik $\delta_2(n)$ jest pomijalnie mały - wariancję możemy przybliżyć takim wzorem:

$$\hat{Var}[\hat{N}_i^{HLL}(S)] \approx (\hat{N}_i^{HLL}(S)(\frac{b_m}{\sqrt{m}}))^2 \tag{4.51}$$

Aby policzyć wariancję dla estymatorów składowych $\hat{N}_i^{HLL}(S_1 \cap S_2) = \hat{\alpha}_i \hat{N}_i^{HLL}(S_i)$ (i analogicznych dla sumy oraz różnicy), wykorzystamy wzór (4.11). Obliczenia będą przebiegały analogicznie jak dla algorytmu `MinCount`, z tą różnicą, że teraz rozmiarem naszej próby nie jest już liczba przechowywanych haszy k_i, ale liczba rejestrów m_i:

$$Var((\hat{\alpha}_i \hat{N}^{HLL}(S_i))) \leqslant \hat{\alpha_i}^2(\hat{n_i}\frac{b_m}{\sqrt{m}})^2 + \hat{n_i}^2\frac{\hat{\alpha}_i(1-\hat{\alpha}_i)}{k} = \tag{4.52}$$

$$\hat{n_i}^2(\hat{\alpha_i}^2(\frac{b_m}{\sqrt{m}})^2 + \frac{\hat{\alpha_i}^2(1-\hat{\alpha}_i)}{\hat{\alpha}_i k}) \tag{4.53}$$

$$\hat{n_i}^2\hat{\alpha_i}^2(\frac{b_m^2}{m} + \frac{(1-\hat{\alpha}_i)}{\hat{\alpha}_i k}) \tag{4.54}$$

$$\hat{Var}(\hat{N_i}^{HLL}(S_1 \cap S_2)) \approx \hat{N}_i(S_1 \cap S_2)^2(\frac{b_m^2}{m} + \frac{(1-\hat{\alpha}_i)}{\hat{\alpha}_i k}) \tag{4.55}$$

Obliczenia dla $\hat{N}_i^{HLL}(S_1 \cup S_2) = \hat{\beta}_i \hat{N}_i^{HLL}(S_i)$ wyglądają analogicznie. Ostateczny wzór ma postać:

$$\hat{Var}(\hat{N_i}^{HLL}(S_1 \cup S_2)) \approx \hat{N}_i(S_1 \cup S_2)^2(\frac{b_m^2}{m} + \frac{(\hat{\beta}_i - 1)}{k}) \tag{4.56}$$

Tak wyznaczone przybliżenia wariancji możemy następnie użyć we wzorze na *estymator ważony* (4.1), analogicznie jak w przypadku algorytmu `MinCount`.

Wyniki eksperymentów

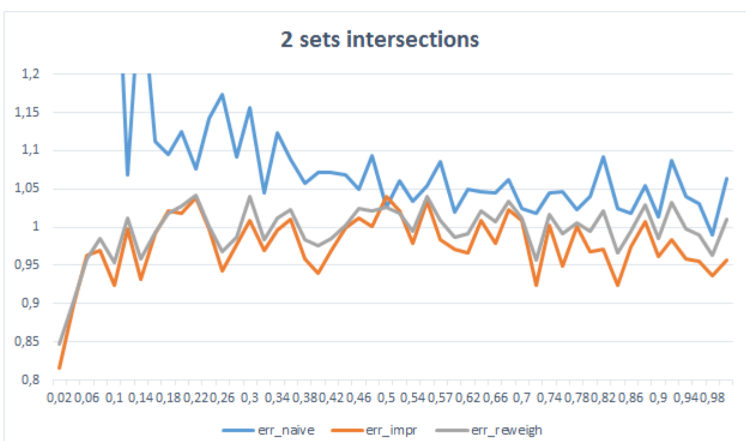
W tym rozdziale przyjrzymy się wynikom przeprowadzonych eksperymentów dla omawianych wcześniej algorytmów i ich ulepszeń. Na początku rozważymy algorytm `MinCount` - porównamy podejście naiwne, ulepszenie zaproponowane w rozdziale 2 oraz metodę *estymatora ważonego* dla tego algorytmu. Następnie dla algorytmu `HyperLogLog`, skupimy się głównie na operacji przekroju - porównamy podejście naiwne, korzystające z indeksu *Jaccarda* i metodę estymatora *ważonego* dla tegoż algorytmu. Na koniec porównamy efektywność obu algorytmów w kontekście operacji teoriomnogościowych i postaramy się określić które z nich sprawdzą się najefektywniej w praktyce. Eksperymenty przeprowadzone w tym rozdziale badają dokładność algorytmów na przestrzeni różnych indeksów *Jaccarda*. Dokładność algorytmu mierzymy przez $\frac{\hat{n}}{n}$, czyli stosunek liczności zbioru wyznaczonej przez estymator do prawdziwej liczności zbioru. We wszystkich eksperymentach dla algorytmu `MinCount`, ustaliliśmy parametr $k = 100$, czyli szkice przetrzymywały 100 najmniejszych haszy.

5.1 Wyniki dla algorytmu MinCount

Na początek przedstawiamy wyniki dla trzech metod estymacji dla algorytmu `MinCount`:

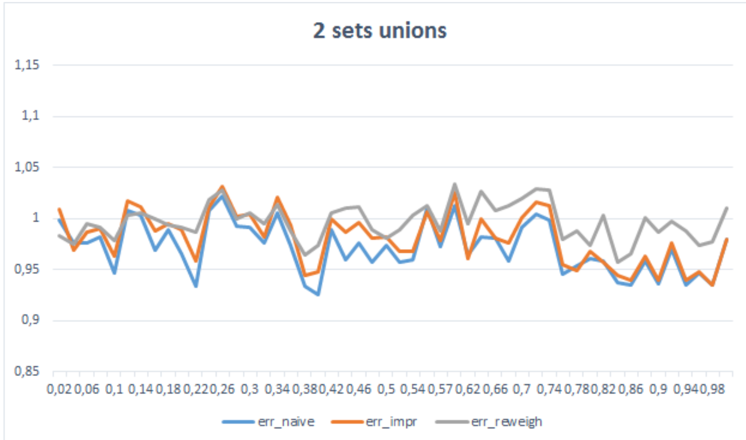
1. metoda naiwna (z *zasady włączeń i wyłączeń*)
2. ulepszenie zaproponowane w rozdziale 2
3. metoda *estymatora ważonego*

Wykresy 5.1 oraz 5.2 przedstawiają wyniki eksperymentów dla sum i przekrojów dwóch zbiorów.



Rysunek 5.1: Porównanie metod dla przekrojów 2 zbiorów przy $n = 10^5$

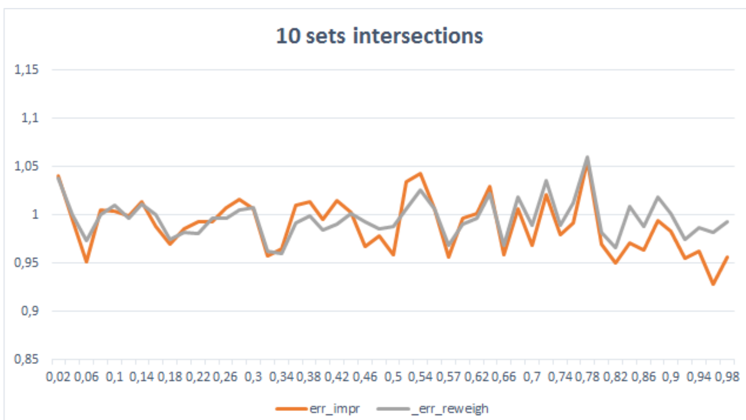
Zauważmy, że dla operacji sumy - praktycznie na całym przedziale indeksów *Jaccarda* zbiorów - najmniejszy błąd posiada metoda *estymatora ważonego*. W przypadku przekroju metody 2. oraz 3. wypadają o wiele lepiej niż metoda naiwna. Na wykresie możemy zauważyć, że metoda *estymatora ważonego* posiada podobną tendencję jak metoda 2., ale dla większości indeksów *Jaccarda* posiada mniejszy błąd.



Rysunek 5.2: Porównanie metod dla sumy 2 zbiorów przy $n = 10^5$

Na kolejnych wykresach: 5.3, 5.4, 5.5 oraz 5.6, zestawiliśmy wyniki dla sum i przekrojów większej liczby zbiorów. Wykonaliśmy eksperymenty dla 10 oraz 100 zbiorów o mocach $n = 10^5$, porównując metody 2. oraz 3. Metodę naiwną pominęliśmy ze względu na jej znaczące wady opisane w rozdziale 3.1.

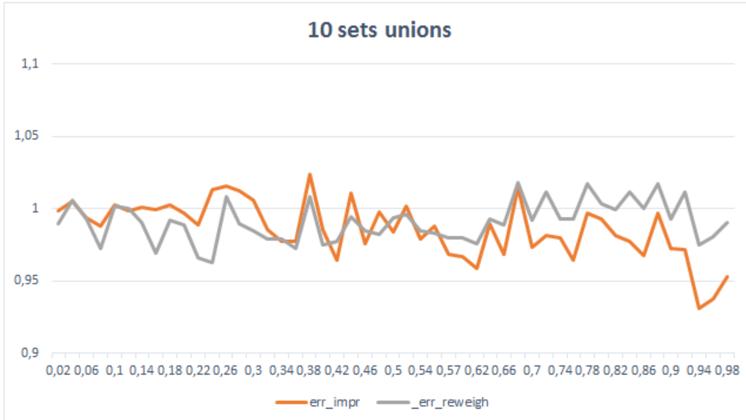
CDN ...



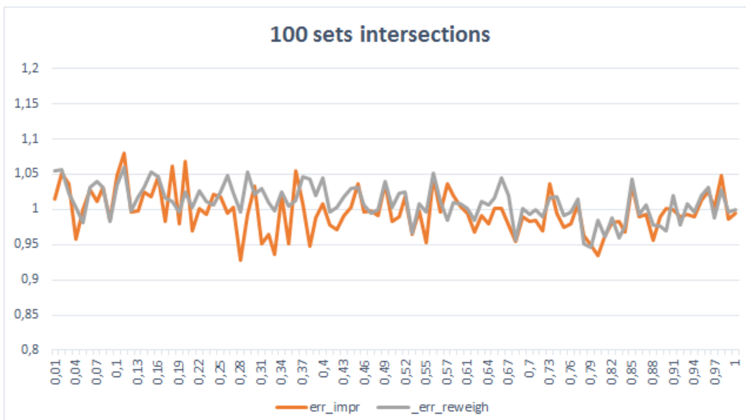
Rysunek 5.3: Porównanie metod dla przekroju 10 zbiorów

Ostatnimi eksperymentami przeprowadzonymi dla algorytmu `MinCount` było sprawdzenie metody *estymatora ważonego* dla różnicy zbiorów, omówionej w rozdziale 4.4. Na wykresie 5.7 przedstawiamy wyniki dla różnicy dwóch zbiorów, porównując podejście naiwne oraz podejście wykorzystujące *estymator ważony*.

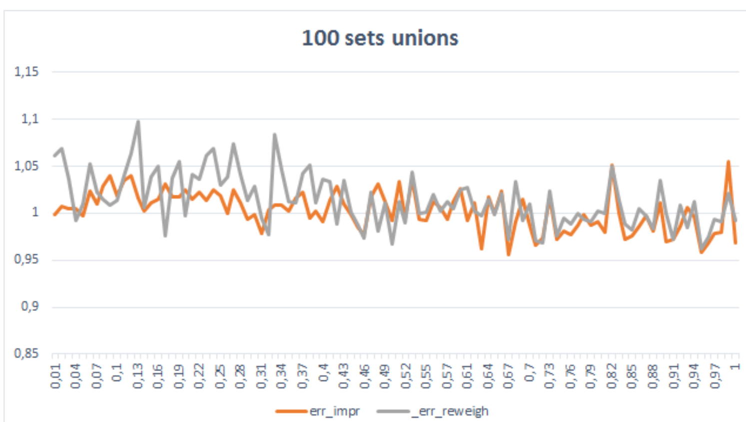
Zauważmy ze metody 2. oraz 3. po raz kolejny wypadają znacznie lepiej od metody naiwnej, a metoda *estymatora ważonego*, podobnie jak w przypadku sumy i przekroju, pokrywa się z tendencją metody 2., ale w większości przypadków posiada mniejszy błąd.



Rysunek 5.4: Porównanie metod dla sumy 10 zbiorów



Rysunek 5.5: Porównanie metod dla przekroju 100 zbiorów

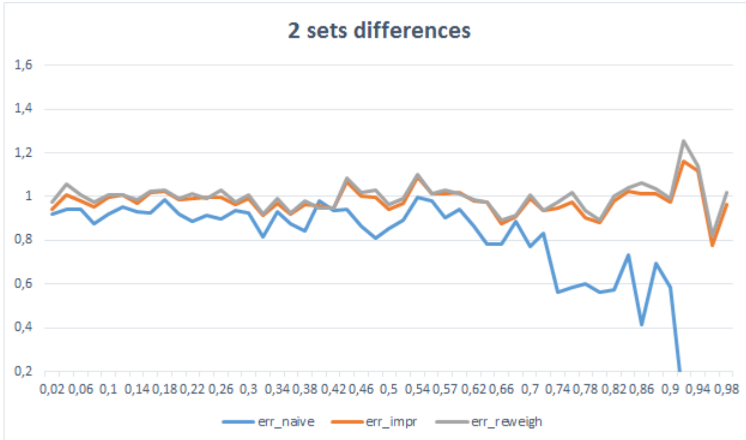


Rysunek 5.6: Porównanie metod dla sumy 100 zbiorów

5.2 Wyniki dla algorytmu HyperLogLog

W tym podrozdziale przedstawimy wyniki dla algorytmu `HyperLogLog`. Zestawiliśmy ze sobą dwie metody:

1. metoda naiwna z użyciem indeksu *Jaccarda* (wzór (2.19))

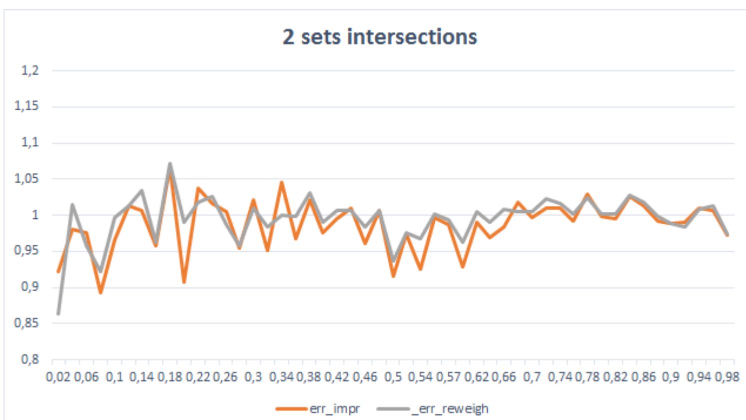


Rysunek 5.7: Porównanie metod dla różnicy 2 zbiorów

2. metoda *estymatora ważonego*

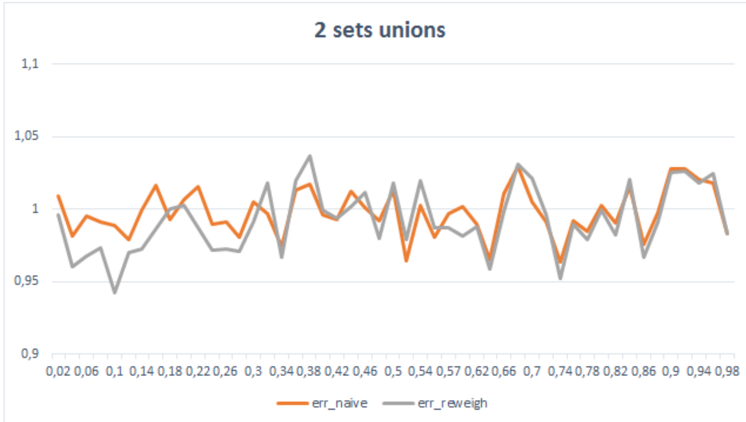
Obydwie metody korzystają z pomocniczej struktury dla każdego ze szkiców, przechowującej k najmniejszych wartości (czyli zasadniczo z podstawowej wersji szkicu `MinCount`), pozwalającej na estymację indeksu *Jaccarda* [7], potrzebnego zarówno do estymacji metodą naiwną jak i metodą *estymatora ważonego*. W przeprowadzanych eksperymentach ustaliliśmy parametry $k = 80$ oraz $b = 8$.

Na wykresach 5.9, 5.9, 5.11 oraz 5.10 przedstawiamy porównanie powyższych metod w kontekście operacji sumy oraz przekroju. Eksperymenty przeprowadziliśmy dla 2 oraz 10 zbiorów o mocach $n = 10^5$.

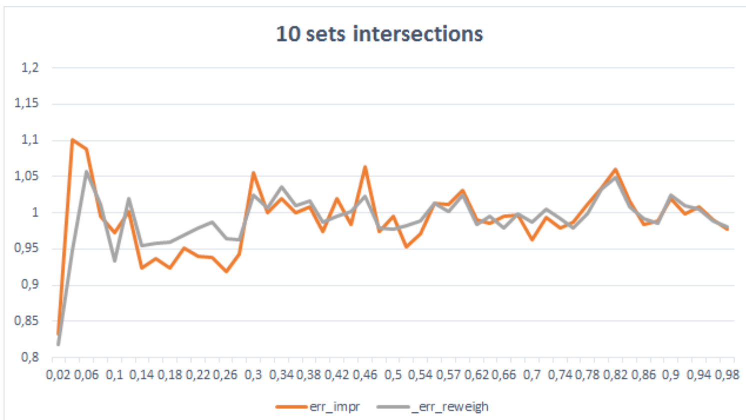


Rysunek 5.8: Porównanie metod dla przekroju 2 zbiorów przy $n = 10^5$

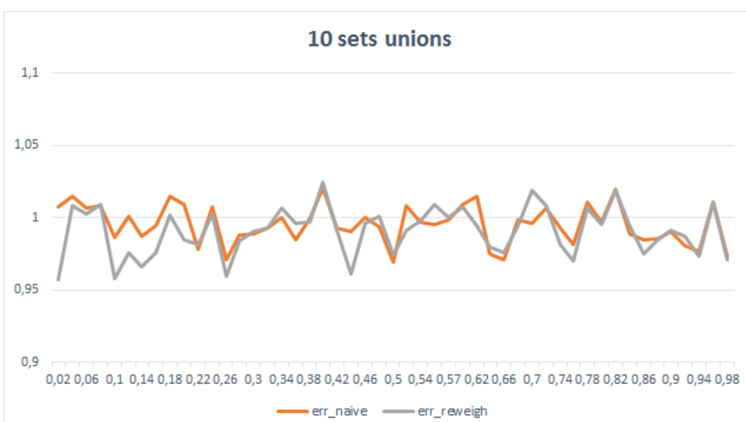
Wnioski wynikające z tych eksperymentów sugerują, że metoda *estymatora ważonego* w kontekście algorytmu `HyperLogLog` dla operacji sumy niestety nie dorównuje pod względem dokładności podejściu naiwnemu. W przypadku sumy jest to dosyć oczywiste, ponieważ jak wspominaliśmy w rozdziale 2 - `HyperLogLog` posiada naturalną operację sumy. W przypadku operacji przekroju metoda naiwna wykorzystująca indeks *Jaccarda* i wzór (2.19) posiada większy błąd niż estymacja z użyciem *estymatora ważonego*, zwłaszcza dla zbiorów o niskim podobieństwie. Zauważmy że obie metody wykorzystują tyle samo pamięci, bowiem obie wymagają dodatkowej struktury pozwalającej na estymację indeksu *Jaccarda* przy wykorzystaniu algorytmu `MinHash`.



Rysunek 5.9: Porównanie metod dla sumy 2 zbiorów przy $n = 10^5$



Rysunek 5.10: Porównanie metod dla przekroju 10 zbiorów przy $n = 10^5$



Rysunek 5.11: Porównanie metod dla sumy 10 zbiorów przy $n = 10^5$

Podsumowanie

W naszej pracy zajęliśmy się problemem zliczania unikalnych elementów w strumieniach danych, a konkretnie metodami efektywnego wykonywania operacji teoriomnogościowych na szkichach danych algorytmów `MinCount` oraz `HyperLogLog`.

CDN ...

Bibliografia

[1] *Estimators, Mean Square Error, and Consistency.* 2006.

[2] A. Z. Broder. *On the resemblance and containment of documents.* 1997.

[3] O. Ertl. *New cardinality estimation algorithms for HyperLogLog sketches.* 2017.

[4] F. Giroire. *Order statistics and estimating cardinalities of massive data sets.* 2009.

[5] P. J. H. B. R. Y. S. Kevin Beyer, Rainer Gemulla. *Distinct-Value Synopses for Multiset Operations.* 2009.

[6] H. M. T. Kyu-Young Whang, Brad T. Vander-Zanden. *A Linear-Time Probabilistic Counting Algorithm for Database Applications.* 1989.

[7] A. Pascoe. *HyperLogLog and MinHash, A Union for Intersections.* 2013.

[8] O. G.-F. M. Philippe Flajolet, Éric Fusy. *HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm.* 2007.

[9] D. P. W. Rasmus Pagh, Morten Stöckel. *Is Min-Wise Hashing Optimal for Summarizing Set Intersection?* 2014.

[10] D. Ting. *Streamed Approximate Counting of Distinct Elements.* 2014.

[11] D. Ting. *Towards Optimal Cardinality Estimation of Unions and Intersections with Sketches.* 2016.

Zawartość płyty CD

W tym rozdziale należy krótko omówić zawartość dołączonej płyty CD.