ストリーム中のアイテム頻度に関する省領域乱択近似アルゴリズム

イム　ヒジェ

平成27年 2月

修士課程
情報学専攻
九州大学大学院システム情報科学府

目次

概説

近年，大規模化するインターネットや様々なセンサーでは, 大量のデータストリームが生成されている．このようなデータストリームに対して，すべてをメモリ空間に記録してから処理することは難しい．従って，逐次的に入力されてくるデータに対して，限られたメモリ空間上で有効な統計情報を抽出できるストリームアルゴリズムが注目されている．ストリームアルゴリズムの代表例として，ストリーム中の頻出アイテムの検出, アイテム種類数の計算, これらを包括する頻度モーメントの計算などが挙げられる．

本研究では, これら問題の汎用拡張を動機として, ストリーム中の出現頻度分布を推定する問題を扱う．これは, ストリームで出現頻度ごとに何種類のアイテムがあるか調べる問題であり, ストリームの統計的特性の調査に応用できる．本研究では, 省領域で出現頻度分布を推定する乱択アルゴリズムを提案する．提案アルゴリズムは, ストリーム中の出現可能な最大のアイテム種類数を m としたときに, 簡単な条件を満たすデータ対して $\mathrm{O}(\log^2 m)$bits のメモリ空間を使う．さらに, 実際のダークネットデータに対して提案アルゴリズムの実装実験を行い, アルゴリズムの妥当性と改善方法について考察する．

Abstract

In recent years, computing for large data such as network traffic and sensor data becomes more and more important because of its applications. For example, analyzing data and computing statistics on network traffic are available to use proactive response against cyber-attacks. However, the storage of a large mount of data requires a huge memory space, as well as it takes a long time. Thus, streaming algorithms that can extract valid information from sequentially coming data on a limited memory space is attracting attention. For a data stream, there are many problems such as finding *frequent items*, estimating *the number of distinct items*, and computing *the frequency moment*.

Motivated by the extension of existing problems, this paper deals with the problem of estimating *the frequency distribution* in a data stream. This problem is to find the number of types of items for each frequency in a stream, which is available to examine some statistical properties of the stream. We propose a randomized algorithm to estimate *the frequency distribution*. This algorithm uses $\mathrm{O}(\log^2 m)$ memory space for some streams satisfying simple conditions, where m is the maximum number of distinct items of a stream. We also show experimental results on our algorithm for a real darknet data.

第 1 章

はじめに

1.1 先行研究

1.1.1 ストリームアルゴリズム

一般的にストリームアルゴリズムは，逐次的に入力されるデータ（ストリームデータ) を処理するアルゴリズムのことである. すなわち, 入力されてくるデータのすべては記憶せず, 必要な一部の情報だけを抽出して処理を行う特徴がある. 省メモリ空間を使って目的とする有効な情報を求めるために, ストリームアルゴリズムでは特集なデータ構造を使ったり，サンプリングやハッシュ関数などの確率的手法を使ったりする方法が開発されてきた.

ストリームアルゴリズムが求める有効な情報には，ストリームの頻度モーメント [1, 13], 頻出アイテム [16, 19]，アイテム種類数 [2, 4, 5, 6, 12], エントロピー [3, 14], ヒストグラム [7, 9, 11], Heavy Distinct Hitters[15, 21] などがあり，それらを省メモリ空間で求めるストリームアルゴリズムが提案されている．次からは頻度モーメント，アイテム種類数, ヒストグラムを求める問題とそれらの先行研究を紹介する.

1.1.2 頻度モーメント

ストリーム上で頻度モーメントを求める問題に関する特筆すべき最初の研究は Alon 達 [1] により始まった．ストリーム上の k 次頻度モーメント F_k とはストリーム上で出現したすべての要素の格出現頻度を k 乗して合計をとった値と定義される． 具体的に書くと，1 から m までの整数が N 個並んだストリーム $S = e_1, ..., e_N (e_i \in [1, m])$ に対して, 頻

度モーメントは

$$F_k = \sum_{e=1}^{m} n_e^k$$

となる．ここで n_e はストリーム S 上の要素 e の出現頻度とする．

頻度モーメントはストリームの様々な統計情報を包括する．0 次頻度モーメントはストリームで出現した要素の種類数であり，1 次頻度モーメントはストリームの長さと一致する．頻度モーメントを変換した $\sum_{e=1}^{m} n_e \log n_e$ の計算はストリームのエントロピーの計算に応用できる [3, 14].

頻度モーメントを素直に求める方法はストリームで出現したすべての要素ごとに出現頻度を数えることだが，$\Omega(m)$ bits のメモリ空間を必要とするので現実的ではない．Alon 達は [1] の論文で通信複雑度を用いて，$k \geq 5$ の k 次頻度モーメントを誤差 0.1 未満で $1-\delta$(δ は 0.5 未満の任意の定数) 以上の確率で求めるためには少なくとも $\Omega(m^{1-5/k})$ のメモリ空間が必要することを証明した (この証明では $N = m$ の前提を使う)．さらに，彼らは $k \geq 1$ に対して $\widetilde{O}(km^{1-1/k}\log(m+N))$ bits, $k = 2$ に対しては $\widetilde{O}(\log(m+N))$ bits のメモリ空間を使い高い確率で頻度モーメントと近い近似値を出力するアルゴリズムを提案した．($\widetilde{O}$ は $\epsilon^{-2}\log(1/\delta)$ を省略した O 記号)

$k = 2$ に対して Alon 達のアルゴリズムが空間的最適アルゴリズムだと知られており，$k \in (0, 2)$ に対してはメモリ空間 $\mathrm{O}(\log(m+N))$, 更新時間 $\mathrm{O}(\log^2(1/\epsilon))$ の Kane 達のアルゴリズム [13] が最適だと知られている．

1.1.3 アイテム種類数

ストリーム $S = e_1, ..., e_N (e_i \in [1, m])$ が与えられたときに，アイテム種類数 (0 次頻度モーメント) は $F_0 = |\{e_1, ..., e_N\}|$ と定義される．F_0 を正確に求めるためにはどの決定的アルゴリズムを用いても $\Omega(m)$ のメモリ空間を必要とする [1]. 少し誤差を許すが，省領域で F_0 を推定するアルゴリズムは 1985 年 Flajolet と Martin により提案された [4]．このアルゴリズムはハッシュ関数を使い $\mathrm{O}(\log m)$ のメモリ空間でアイテム種類数を推定する．その他に，メモリ空間と計算時間を改善したアイテム種類数を推定するストリームアルゴリズムが提案されている．2002 年, Bar-Yossef 達 [2] は $\mathrm{O}(\epsilon^{-2}\log m)$ のメモリ空間, $\mathrm{O}(\log(1/\epsilon))$ の更新時間で 2/3 以上の確率で $((1-\epsilon)F_0, (1+\epsilon)F_0)$ の範囲でアイテム種類数を推定する乱択アルゴリズム提案した．このアルゴリズムは誤差パラメータ ϵ を調節することで，トレードオフの関係である精度とメモリ空間を決めることができる．さらに，このアルゴリズムを独立並行に走らせて出力値の中央値を取ると，高い確率で $(1 \pm \epsilon)F_0$ の間でアイテム種類数を推定することができる．現在では, Kane 達の $\mathrm{O}(\epsilon^{-2} + \log m)$ の

メモリ空間, $O(1)$ の更新時間のアルゴリズム [12] が $(1 \pm \epsilon)F_0$ の範囲で F_0 を推定するアルゴリズムの中でメモリ空間と更新時間の面で最適だと知られている.

1.1.4 ヒストグラム

ストリームデータ上のヒストグラムは出現頻度分布を棒グラフで表したものである. この棒グラフの横軸はバケット (あるいはビン) と呼ばれるストリーム上で出現可能な要素の部分範囲であり，各バケットの高さ (棒グラフの縦軸) はバケットの範囲に含まれる要素の出現頻度の合計とする．具体的に例を挙げで説明する．[1,100] の間の整数が出現するストリームに対して, 10 個のバケットを使いヒストグラムを作ることを考えてみよう．まず，10 個のバケットで [1,100] の範囲を均等に分けてみると，各バケットは $B_1 = [1, 10], B_2 = [11, 20], ..., B_{10} = [91, 100]$ と定義される．この場合，ヒストグラムで各バケットの高さは各範囲の要素の出現回数となる．例えば，バケット B_1 の高さはストリームで 1 から 10 の間の整数の出現頻度の合計となる．このようにバケットの範囲を均等な長さで定義したヒストグラムを *equi-width histograms*[10, 20] とする．一方, *equi-width histograms* からいつも頻度分布に関する有効な情報が得られるとは言えない. 例えば，上記のストリームで 1 と 2 は 1000 回, 3 から 100 は 1 回ずつ出現したとする．この場合，B_1 の高さは $1000 \cdot 2 + 1 \cdot 8 = 2008$, B_2 の高さは $1 \cdot 10 = 10$ である．B_2 の高さを見て，11 から 20 までは出現頻度が小さいと推定できるが，B_1 の高より 1 から 10 までの整数のそれぞれの出現頻度に関する情報は得られない．従って，*V-optimal histograms* と呼ばれて，各バケットで含まれている要素の出現頻度の分散を最小化するように各バケットの範囲を決めるヒストグラムがよく使われている．先の例だと，$B_1 = [1, 2]$(B_1 では出現頻度が 1000 と近いの要素の集まり) とし，B_2 から B_{10} は $[3, 100]$ の範囲を順序で被らないように分けると, 各バケットの分散が最小になる (この例では分散がすべて 0). *V-optimal histograms* は各バケットごとにデータの範囲を表す始点と終点 (先の例の [1,2] では, 1 が始点，2 が終点)，バケットの高さで構成される．ここでバケットの高さはバケットに含まれる要素の出現頻度の平均値とする. 分散を最小化する性質より，各バゲットはバケットの高さに近い出現頻度を持つ要素の集まりになったと推定できる．(ただし, *V-optimal histograms* も誤差を最小化しただけなので，実際の分布との推定に難しい場合が存在する)

1998 年 Jagadish 達は [11] の論文で, バケットの数 B と要素が $[1, m]$ から出現するデータ (データの数は m より小さい) が与えれているときに，$O(mB)$ メモリ空間, $O(m^2B)$ 計算時間で *V-optimal histograms* を計算する最適アルゴリズムを提案した．ストリームアルゴリズムの観点からは，2006 年 Guha 達 [7] より m と B の線形に比例したメモリ空

間を使って *V-optimal histograms* を $(1+\epsilon)$ 近似するアルゴリズムが提案された．更に小さいメモリ空間のアルゴリズムでは 2012 年, Indyk 達 [9] よる $O(B^2 \log m)$ メモリ空間で *V-optimal histograms* を推定するアルゴリズムが存在する．

1.2 本研究の概要

本研究では既存の頻度モーメントやヒストグラムでは観測できなかった情報が得られる**出現頻度分布**を定義し，その分布を推定するストリームアルゴリズムを提案する．

まず，ネットワークデータの監視の観点から頻度モーメントとヒストグラムの問題点を述べる．ネットワークデータに対してストリームアルゴリズムを使う例は DDoS 攻撃などのサイバー攻撃による統計的異常を検知がある．具体的に 0 次頻度モーメント (ネットワーク上の異なる IP アドレスの数) やエントロピーをある時間単位で計算すると, ネットワークの頻度分布の変化が分かるので DDoS 攻撃やボットネットによる攻撃の予知に応用できる．しかし, この場合は IP の種類数やエントロピーの値だけが分かるので，変化の具体的な原因や全体的の頻度分布は分からない．いくつかの k に対して同時に k 次頻度モーメントを求めれば頻度分布が推定できるが，頻度モーメントを求めるときの k 倍のメモリ空間が必要とするので簡単ではない．一方，*equi-width histograms* や *V-optimal histograms* のようなヒストグラムは頻度モーメントより具体的に IP アドレスの出現頻度分布が分かる．しかし, これらのヒストグラムは IP アドレス (IPv4) の範囲 0.0.0.0 ~ 255.255.255.255 を B 個の順序で被らないバケットに分けて出現頻度を表すが, 意味がある結果を得るためにバケットの数 B をどう決めるかと, 連続した IP アドレスの部分範囲と出現頻度のヒストグラムが異常検知に直観的ではないことが問題である．

従って，本研究では頻度モーメントとヒストグラムの問題を解決するために出現頻度分布を新しく定義する．C_j をストリーム S の中で出現頻度が 2^j 以上の要素の集合とし, **出現頻度分布**は集合 $C_0, \ldots, C_s$ の大きさ, すなわち $|C_0|, \ldots, |C_s|$ のことを言う（s はストリームの長さの二進対数, $s = \lfloor \log_2 N \rfloor$ とする).

C_0 は, 出現頻度が $2^0 = 1$ 以上の要素の集合なので, ストリーム S で出現した要素の集合 E_S と等しい．一般的に $|C_j| - |C_{j+1}|$ は出現頻度が 2^j 以上 2^{j+1} 未満の要素の数となる．従って, 出現頻度分布 $|C_0|, ..., |C_s|$ が分かれば，出現頻度が $[1,2), [2,4), ..., [2^j, 2^{j+1}), ..., [2^{s-1}, 2^s)$ 範囲に含まれる要素の数が求められる．

ネットワークデータの例を使ってこの分布の良い点を説明する．ネットワークデータに対するヒストグラムの横軸は IP アドレスの全体範囲を順序で被らないように分けたバケットになるので，頻度ごとのユニーク IP アドレスの数が分かりにくい．一方，出現頻

度分布からは横軸を $1 \leq j \leq s$, 縦軸を $|C_j|$ としたグラフが得られる．このグラフは log スケールでの出現頻度ごとにユニーク IP アドレスの数を表す．例えば，ボットネットによるネットワーク攻撃は多数のユニーク IP が通常と違う頻度で通信を行う．出現頻度分布を単位時間で求めてグラフの変化を見てみると, 既存のヒストグラムよりボットネットの動きを直観的に推定しやすくなる．

ところが，$|C_0|$(アイテム種類数) を正確に求めることは $\Omega(m)$ bits 必要とするので [1], 出現頻度分布を正確に求めるためには少なくとも $\Omega(m)$ bits のメモリ空間を必要とする．

従って，本研究では省領域で出現頻度分布を近似する乱択アルゴリズムを提案する．本手法は簡単なサンプリング法とアイテム種類数を求めるアルゴリズム [2] を組み合わせたものである．提案アルゴリズムは簡単な条件を満たすストリームに対して $\mathrm{O}(\log^2 m)$ bits のメモリ空間を使う．

本論文の構成は, まず第 2 章でいくつかの基本準備を行う．第 3 章では出現頻度分布を近似するアルゴリズムを提案し，そのアルゴリズムの理論解析について述べる．次に第 4 章では実際のネットワークデータに対する提案アルゴリズムの実装実験を行って, アルゴリズムの妥当性と改善方法について考察する．

第 2 章

準備

2.1 問題定義

本研究で扱うストリームデータと問題を定義する．まず，ストリーム S は 1 以上 m 以下の整数からなる列だとする．列の i 番目の要素を e_i と表記し，ストリーム S の i 番目の要素と呼ぶ．N をストリームの長さと定義し，ストリーム S を $S = e_1, \ldots, e_N (e_i \in [1, m])$ と記す．

ストリーム S の中で出現した要素の集合を $E_S (\subseteq [1, m])$ とする．出現した異なる要素の数 (数要素の種類数) は集合 E_S の大きさ $|E_S|$ である．n_e をストリーム上で要素 e の出現頻度と定義する ($n_e := |\{i \in \{1, ..., N\} : e_i = e\}|$)．$E_S$ に属するすべての要素 e の出現頻度 n_e の合計はストリームの長さ N と一致する ($\sum_{e \in E_S} n_e = N$)．例えば，ストリーム $S = 3, 10, 3, 2, 1, 5, 3, 1, 1, 1$ において，出現した要素の集合は $E_S = \{3, 10, 2, 1, 5\}$ であり，要素それぞれの出現頻度は $n_3 = 3$, $n_{10} = 1$, $n_2 = 1$, $n_1 = 4$, $n_5 = 1$ である．

本研究で扱うすべての問題は入力としてストリームデータを一方で一回のみ使えると仮定する．すなわち，ストリーム S の要素を e_1 から e_N まで一回ずつ読みながら処理する．この制約の下で本研究で扱う問題を定義する．

問題 2.1. (k **次頻度モーメント計算問題**) ストリーム S の k 次頻度モーメント F_k とは，ストリームで出現したすべての要素の出現頻度を k 乗して合計をとった値と定義し，$F_k = \sum_{e=1}^{m} n_e^k$ と記す．与えられた k とストリーム S に対して，k 次頻度モーメント F_k を求める問題を k 次頻度モーメント計算問題と言う．

問題 2.2. (**非線形関数 f の計算問題**) 関数 $f : \mathbb{R} \to \mathbb{R}$ は $f(0) = 0$，任意の $x < y (\in \mathbb{R})$ に対して $f(x) < f(y)$ が満たされ実数から実数への関数とする．この関数 f と S が与えられているときに，$F_f = \sum_{e=1}^{m} f(n_e)$ を求めることを非線形関数 f の計算問題と定義する．

一般的な頻度モーメントは問題 2.1 のように定義されるが, 本研究では log 関数のような非線形関数に注目するために問題 2.2 を定義する.

問題 2.3. (**アイテム種類数の** (ϵ, δ) **近似問題**) ストリームのアイテム種類数 F_0 とはストリームで出現した異なる要素の数 $(= |E_S|)$ とする. アイテム種類数の (ϵ, δ) 近似問題は, 与えられたパラメータ $0 < \epsilon, \delta < 1$ とストリーム S に対して, $\Pr[|F_0 - \hat{F_0}| \leq \epsilon F_0] \geq 1 - \delta$ が満たす F_0 の推定値 $\hat{F_0}$ を求める問題とする.

アイテム種類数の (ϵ, δ) 近似問題の ϵ は許される誤差の範囲を表し, δ は推定値が誤差の範囲を外される確率を意味する. 次に本研究の主問題を定義する.

問題 2.4. (**出現頻度分布計算問題**) 集合 C_j を出現頻度が 2^j 以上の要素を元とする集合と定義し, $C_j = \{e \in [1, m] : n_e \geq 2^j\}$ と表記する. 長さ N のストリームに対して, $|C_0|, \ldots |C_s|$ を出現頻度分布と呼び, 出現頻度分布を求める問題を出現頻度分布計算問題とする. ただし, $s = \lfloor \log_2 N \rfloor$ とする.

2.2 頻度モーメントを求める Alon 達のアルゴリズム

本節では Alon 達の論文 [1] から頻度モーメント F_k を推定するアルゴリズムと頻度モーメント計算に関連する定理を紹介する. まず, Alon 達の頻度モーメント F_k を推定するアルゴリズムを Algorithm1 を使って説明する.

Algorithm 1 Alon 達の F_k を求めるアルゴリズム [1] の概要

Require: $S = e_1, ..., e_N (e_i \in [1, m])$

1: $[1, ..., N]$ から一様ランダムに α を決める.
2: $r \leftarrow |\{e_j : \alpha \leq j \leq N, e_j = e_\alpha\}|$
3: **return** $X = N(r^k - (r-1)^k)$

Algorithm1 は $O(\log N)$bits のメモリ空間を使って頻度モーメントの推定値 X を出力する乱択アルゴリズムである. ストリームからデータを読み始める前に α を $[1, ..., N]$ から一様ランダムに決める. データを読み始めて $(\alpha - 1)$ 番目までのデータは無視し, α 番目のデータから最後までは α 番目に出現した要素 e_α の出現頻度 r を数え上げていく. 出力 $X = N(r^k - (r-1)^k)$ の期待値は求めたい値 F_k と一致し, 分散 $V(X) \leq kF_1F_{2k-1}$ が成立する. Algorithm 1 を複数独立並行に走らせると, 高い確率で F_k と近い値が得られ

る. 具体的な空間複雑度と信頼性は誤差パラメータ ϵ と確率パラメータ δ を使って次の定理から示される.

定理 2.5. [1] 任意の $k \geq 1, \epsilon > 0, \delta > 0$ に対して, ストリーム $S = e_1, ..., e_N (e_i \in [1, m])$ から $\mathrm{O}(\frac{k \log(1/\delta)}{\epsilon^2} m^{1-1/k}(\log m + \log N))$ のメモリ空間を使い, 頻度モーメント F_k との絶対値誤差が ϵF_k 以下である推定値 X を $1 - \delta$ 以上の確率で出力する乱択ストリームアルゴリズムが存在する.

さらに, 頻度モーメントを推定するために必要なメモリ空間の下界について次の定理が存在する.

定理 2.6. [1] 固定された任意の $k > 5$ と $\delta < 1/2$ に対して, ストリーム $S = e_1, ..., e_m (e_i \in [1, m])$ から, 確率 $\Pr[|F_k - X| > 0.1F_k] < \delta$ を満たしながら X を出力する任意の乱択アルゴリズムは少なくとも $\Omega(m^{1-5/k})$bits のメモリ空間を必要とする.

2.3 アイテム種類数を求める Bar-Yossef 達のアルゴリズム

Bar-Yossef 達 [2] は $\mathrm{O}(\frac{\log(\delta^{-1})}{\epsilon^2} \log m)$ のメモリ空間, $\mathrm{O}(\log(1/\epsilon))$ の更新時間でアイテム種類数の (ϵ, δ) 近似問題を解くストリームアルゴリズム [2] を提案した. このアルゴリズムの理論的背景には次の確率性質がある. 0 以上 1 以下の範囲から独立ランダムに m 個の実数を取ってきたときに, m 個の実数の中で t 番目に小さい値 v の期待値は $t/(m+1)$ である (証明は [17] を参照). こういう性質を使って, アイテムのハッシュ値の中で t 番目に小さい値 v を覚えていれば小さいメモリ空間でアイテム種類数が推定できる.

具体的に Bar-Yossef 達のアルゴリズムを説明する. アルゴリズムはハッシュ関数 $h : [1, m] \to [1, m^3]$ とサイズ t のリスト L を使う. ハッシュ関数 h はペアワイズ独立性を満たすハッシュ族 $\mathcal{H}$ からランダムに選べられたもので, ストリーム中で出現する要素 $(\in [1, m])$ を $[1, m^3]$ の範囲でランダムにマッピングする (ハッシュ関数族のペアワイズ独立性と作り方は次の節で述べる). リスト L はストリームの中から出現した要素のハッシュ値の中で小さい順で t 個のハッシュ値を記憶する. アルゴリズムは i 番目の要素 e_i を読んで, ハッシュ値 $h(e_i)$ を計算し必要に応じてリスト L を更新する. 最後にリスト L の中から一番大きいハッシュ値 (全体で t 番目に小さいハッシュ値)v を使い $X = tm^3/v$ を出力する.

$t = \lceil 96/\epsilon^2 \rceil$ とすれば, 出力値 X に対して $\Pr[|X - F_0| > \epsilon F_0] < 1/3$ が成立する. 更に, このアルゴリズムを $\mathrm{O}(\log(\delta^{-1}))$ 個並行に走らせて v の中間値を $\hat{F}_0$ とすれば, Chernoff 不等式より $\Pr[|\hat{F}_0 - F_0| \leq \epsilon F_0] \geq \delta$ が成り立つ.

2.4 ペアワイズ独立性を満たすハッシュ関数族

本節ではハッシュ関数族のペアワイズ独立性を定義して，ペアワイズ独立性を満たすハッシュ関数族の例を挙げる．

定義 2.7. (ハッシュ関数族のペアワイズ独立性) 有限集合 A と B に対して, ハッシュ関数族 $\mathcal{H}$ はハッシュ関数 $h : A \to B$ の集合とする．ハッシュ関数族 $\mathcal{H}$ がペアワイズ独立性を持つとは, ハッシュ関数族 $\mathcal{H}$ から選べられた任意のハッシュ関数 h と $a_1 \neq a_2 \in A$ と $b_1, b_2 \in B$ に対して,

$$\Pr[h(a_1) = b_1 \text{ and } h(a_2) = b_2] = \Pr[h(a_1) = b_1] \cdot \Pr[h(a_2) = b_2]$$

が満たされるとする．

ペアワイズ独立性を満たすハッシュ関数族の例を紹介する．m を 2 以上の整数とおいて, M を m^3 より大きい素数の中で一番小さい素数とする．ハッシュ関数の族 $\mathcal{H}$ を次のようにハッシュ関数 $h_{\alpha,\beta} : [1, m] \to [1, M]$ の集合とする．

$$\mathcal{H} = \{h_{\alpha,\beta} : (\alpha, \beta) \in [1, M] \times [1, M]\}$$

$$h_{\alpha,\beta}(e) = (\alpha \cdot e + \beta \mod M) + 1$$

α と β を $[1, M]$ から独立ランダムに選ぶと（$\mathcal{H}$ からハッシュ関数のランダムに選ぶことと同値), 任意の $a \in [1, m]$, $b \in [1, M]$ に対して,

$$\Pr[h_{\alpha,\beta}(a) = b] = 1/M$$

が満たされる．さらに, 異なる a_1, $a_2 \in [1, m]$ と $b_1, b_2 \in [1, M]$ に対して,

$$\Pr[h(a_1) = b_1 \text{ and } h(a_2) = b_2] = \frac{1}{M^2} \tag{2.1}$$

が成り立つので (式 (2.1) の証明は [17] を参照), このように設計されたハッシュ族 $\mathcal{H}$ はペアワイズ独立性が満たされる．

2.5 確率変数の不等式

本論文で使う確率変数に関する不等式を紹介する．証明は参考文献 [18] を参考してほしい．

定理 2.8. (*Markov* 不等式) 非負の確率変数 X と任意の正数 $a > 0$ に対して,

$$\Pr[X \geq a] \leq \frac{E[X]}{a}.$$

定理 2.9. (*Chebyshev* 不等式) 確率変数 X の期待値が μ, 分散が σ^2 のとき, 任意の正数 $a > 0$ に対して,

$$\Pr[|X - \mu| \geq a\sigma] \leq \frac{1}{a^2}.$$

Chebshev 不等式は任意の正数 $\epsilon > 0$ に対して, $\Pr[|X - \mu| \geq \epsilon\mu] \leq \frac{\sigma^2}{(\epsilon\mu)^2}$ と変形して使われる場合が多い.

定理 2.10. (*Chernoff Bound*) $X_1, \ldots, X_n$ を 0 または 1 の値を取る独立な確率変数だとし, 確率変数 X_i が 1 の値を取る確率を $\Pr[X_i = 1] = p_i$ とする. また, 確率変数 $X = \sum_{i=1}^{n} X_i$ とすると, X の期待値が $\mu = \sum_{i=1}^{n} p_i$ である. このとき, 任意の正数 δ に対して,

$$\Pr[X > (1+\delta)\mu] < \left[\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right]^{\mu}, \tag{2.2}$$

$$\Pr[X < (1-\delta)\mu] < \exp(-\mu\delta^2/2). \tag{2.3}$$

Chernoff Bound は確率変数 $X_1, \ldots, X_n$ が独立という強い制約があるものの, Chebshev 不等式よりはもっとタイトなバウンドが得られる.

第 3 章

出現頻度分布推定問題

本章では, ストリームから出現頻度分布を省領域で推定するアルゴリズムについて述べる. まず, 3.1 節では出現頻度分布推定問題の詳細を分析する. 3.2 節では Main Algorithm を説明するための準備でサンプルリングを使った素朴なアルゴリズムを説明する. 3.3 節では $O(\log^2 m)$bits のメモリ空間を使って出現頻度分布を推定する Main Algorithm を提案する.

3.1 問題の分析とアルゴリズムのアイデア

問題 2.4 で定義したように**出現頻度分布**を求める問題は, ストリーム $S = e_1, \ldots, e_N (e_i \in [1, m])$ から $|C_0|, \ldots, |C_s|$ を計算することである. ただし, C_j は出現頻度が 2^j 以上の要素の集合で, $|C_j|$ はその集合の大きさ, 出現頻度が 2^j 以上の要素の種類数を表す. 例えば, C_0 は出現頻度が $2^0 = 1$ 以上要素の集合で, $|C_0|$ はストリーム S からの要素の種類数である. また, $C_s \subseteq C_{s-1} \subseteq \ldots \subseteq C_0$ が成立し, $C_j \setminus C_{j+1}$ は出現頻度が 2^j 以上 2^{j+1} 未満の要素の集合とする.

ストリーム S_1 の例を使って具体的に出現頻度分布を説明する. ストリーム S_1 は 500 種類の要素が出現し ($|E_{S_1}| = 500$), 出現頻度ごとの種類数が表 3.1 に表されたとする. この表の 2 番目の項目はストリーム S_1 で i 回出現要素の種類数を表す. 例えば, ストリーム S_1 で 1 回出現した要素の種類数は 100 であり, 2 回出現した要素の数は 70 である. 3 番目の項目は i 回以上出現した要素の数を表す. 1 回以上出現した要素の数は 500 で, ストリーム S_1 で出現した要素の種類数は 500 であることを意味する. これらをグラフとして表したものが図 3.1 の (a) である. 一方, $|C_j|$ は出現頻度が 2^j 以上の要素の種類数だと定義した. ストリーム S_1 で 20 回を超えて出現した要素はないので, 5 以上の j に対して 2^j 回以上出現した要素の数は $|C_j| = 0$ である. $0 \leq j \leq 5$ に対して $|C_j|$ は表 3.2 に書いてあ

表 3.1: ストリーム S_1 の出現分布

出現頻度 i	i 回出現した要素の種類数	i 回以上出現した要素の種類数
1	100	500
2	70	400
3	81	330
4	60	249
5	35	189
6	27	154
7	24	127
8	30	103
9	15	73
10	13	58
11	7	45
12	10	38
13	5	28
14	8	23
15	3	15
16	5	12
17	3	7
18	2	4
19	1	2
20	1	1

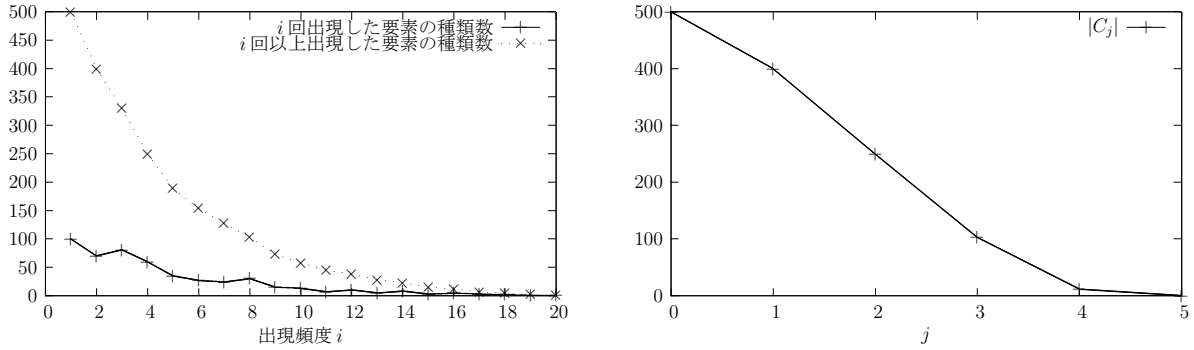
表 3.2: ストリーム S_1 の出現頻度分布

j	$\lvert C_j \rvert$
0	500
1	400
2	249
3	103
4	12
5	0

り, グラフで表したものが図 3.1 の (b) である.

出現頻度分布を求める簡単な方法は出現したすべての要素をリストに書いて要素ごとに出現頻度を数え上げることである. しかし, この方法は出現可能な要素の種類数 m に比例するメモリ空間が要るので, インターネットデータの IP アドレス (IPv4 なら $m = 2^{32}$) のように m が大きい場合は現実的ではない. そもそも $|C_0|$ を正確に求めることはアイテム種類数問題と一致するので, どのような決定的アルゴリズムを用いても $\Omega(m)$ のメモリ空間が必要とする [1]. 従って, $|C_0|, \ldots, |C_s|$ を小さいメモリ空間で正確に求めることは簡単ではない.

では $|C_j|$ の近似なら小さいメモリ空間で計算可能だろうか. 小さいメモリ空間を使う



(a) 出現頻度と要素の種類数　　(b) ストリーム S_1 の**出現頻度分布**

図 3.1: ストリーム S_1 の頻度分布に関するグラフ

ストリーム処理で $|C_j|$ の計算が難しい直観的な理由は, 過去のデータに関して記録できる情報の量が制限されているからである. 例えば, $|C_0|$ を求めることを考える. ストリームの i 番目の要素 e_i を読んだときに, この要素 e_i がすでに出現したことがある要素かどうかは e_i に対応する情報を持っていなければ判別できない. 従って, $|C_0|$ を正しく計算するためには要素の種類数（$= |C_0|$) に比例する情報を持つ必要がある [1].

それでアイテム種類数問題を省領域で近似するときによく使う方法はハッシュ関数である [2, 4, 5, 12]. 十分多数の異なる要素が出現したときに, このハッシュ関数により異なる要素は異なるハッシュ値を持ち, 全体のハッシュ値が種類数に比例して均等に配置される. このように特徴つけられたハッシュ値の一部の情報だけを持っていれば小さいメモリ空間でもアイテム種類数が推定できる. この方法の具体的な例が第 2 章で紹介した Bar-Yossef 達の空間複雑度 $\mathrm{O}(\log m)$ の近似アルゴリズムである.

本研究で省領域で $|C_j|$ を推定するアイディアは, 頻度に対してサンプリング, 種類数に対してはハッシュ関数を使うことである. ハッシュ関数を使ったら i 番目の要素 e_i が既に出現したかどうか $\mathrm{O}(\log m)$ のメモリ空間で判別できる. C_j は出現頻度が 2^j 以上の要素の集合なので, ストリーム S からサンプリングより 2^j 回以上出現する要素を選び出すことができる. その後に選ばれた要素の中からハッシュ関数より種類数が推定し, この組み合わせで $|C_j|$ を推定する.

3.2 節では, サンプリングより頻度ごとの要素を選び出す Simple Algorithm を提案する. 3.3 節では, ハッシュ関数を使って種類数を推定するアルゴリズムと組み合わせし $\mathrm{O}(\log^2 m)$ のメモリ空間で**出現頻度分布**を推定する Main Algorithm を提案する.

3.2 Simple Algorithm

Main Algorithm を説明するために, まず素朴なアルゴリズムを与えて解析を行う.

3.2.1 アルゴリズムと精度保証定理

Algorithm 2 $|C_0|,\ldots,|C_s|$ を推定する Simple Algorithm

Require: data stream $S = e_1,\ldots,e_N (e_i \in [1,m])$, パラメータ s

1: $\widetilde{C}_j \leftarrow \emptyset$ $(j = 1,\ldots,s)$.
2: **for** $i = 1$ **to** N **do** //ストリーム S から i 番目の要素 e_i を読む
3: 公平なコインを初めて表が出るまで投げ続ける (最大 (s+1) 回まで). このとき投げた回数を x とする.
4: **for** $j = 0$ **to** $(x-1)$ **do**
5: $\widetilde{C}_j \leftarrow \widetilde{C}_j \cup \{e_i\}$
6: **end for**
7: **end for**
8: **return** $|\widetilde{C}_0|,|\widetilde{C}_1|,\ldots,|\widetilde{C}_s|$

このアルゴリズムは簡単なサンプリングを行って $|C_j|$ の推定値 $|\widetilde{C}_j|$ を出力する. ストリームから i 番目の要素 e_i を読んだら, 裏が出る確率が 1/2, 表が出る確率が 1/2 の公平なコインを使って, 初めて裏が出るまで投げ続ける. $(s+1)$ 回コインを投げても裏が出てない場合はそれ以上投げることを止める. このとき, 投げた回数を x とすれば, $0 \le j \le x-1$ に対して $\widetilde{C}_j$ に要素 e_i を追加する. 既に $\widetilde{C}_j$ が e_i を含んでいれば何もしない. すべてのデータが読み終わったら, $|\widetilde{C}_0|,\ldots,|\widetilde{C}_s|$ を出力する. 疑似コードは Algorithm2 に記す.

Simple Algorithm が出力する $|\widetilde{C}_0|,\ldots,|\widetilde{C}_s|$ の期待値の下界と上界について定理 3.1 が成立する. ここで a_j, b_j はストリームの $|C_0|,\ldots,|C_s|$ より決まる定数である.

定理 3.1. *Simple Algorithm* が出力する $|\widetilde{C}_j|$ の期待値に対して,

$$a_j \le \mathrm{E}[|\widetilde{C}_j|] < b_j \tag{3.1}$$

が成り立つ. ここで a_j と b_j はそれぞれ次の値とする.

$$\begin{aligned} a_j &= \max_{0\le\alpha<(s-j)} \{(1-\exp(-2^\alpha)\cdot|C_{j+\alpha}|\}, \\ b_j &= \min_{0\le\alpha\le j} \left\{\left(1-1/4^{2^{-\alpha}}\right)\cdot|C_0| + 1/4^{2^{-\alpha}}|C_{j-\alpha}|\right\}. \end{aligned}$$

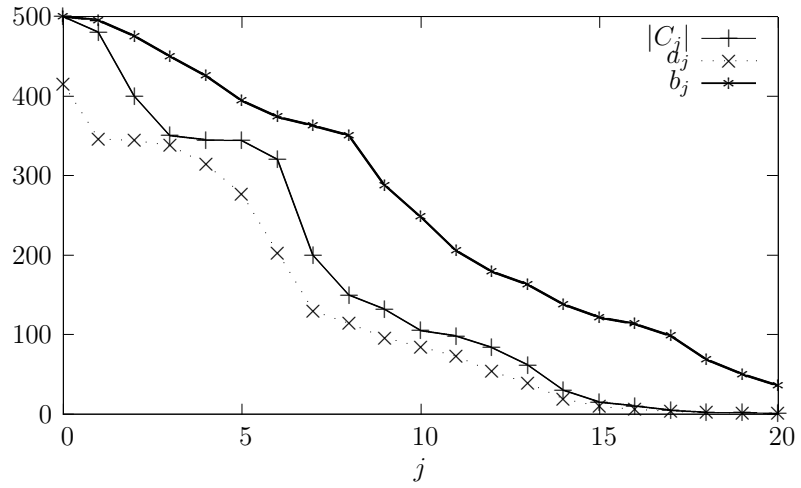


図 3.2: ストリーム S_2 の出現頻度分布と Simple Algorithm の出力の期待値の上界 b_j と下界 a_j.

表 3.3: ストリーム S_2 の出現頻度分布

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$\lvert C_j\rvert$	500	480	400	350	345	344	320	200	150	132	105	98	84	62	30	15	10	5	2	2	1

表 3.3 のストリーム S_2 のような例では Simple Algorithm が出力する期待値の下界 a_j と上界 b_j は図 3.2 に示される. 定理 3.1 から $\alpha = 0$ とすれば, 次に命題が成り立つ.

命題 3.2. *Simple Algorithm* が出力する $|\widetilde{C}_j|$ の期待値に対して,

$$0.63|C_j| \leq \mathrm{E}[|\widetilde{C}_j|] < 0.75|C_0| + 0.25|C_j|. \tag{3.2}$$

3.2.2 定理 3.1 の証明

Algorithm2 について確率変数 $X_{j,e}$ は要素 e が $\widetilde{C}_j$ に含まれば 1, そうでなければ 0 の値を持つ確率変数とする. ストリームの i 番目の要素 e_i が $\widetilde{C}_j$ に到達する確率は $1/2^j$ なので, n_e 回出現した要素 e のいずれも $\widetilde{C}_j$ に到達しない確率は $(1-1/2^j)^{n_e}$ である. したがって, 要素 e が少なくとも一つ以上 $\widetilde{C}_j$ に到達する確率 $p_{j,e}$ は

$$p_{j,e} = \Pr[X_{j,e} = 1] = 1 - (1 - 1/2^j)^{n_e} \tag{3.3}$$

である. $X_{j,e}$ の期待値と分散は

$$\mathrm{E}[X_{j,e}] = 1 \cdot p_{j,e} + 0 \cdot (1 - p_{j,e}) = p_{j,e}$$

$$\mathrm{Var}[X_{j,e}] = \mathrm{E}[X_{j,e}^2] - \mathrm{E}[X_{j,e}]^2 = p_{j,e} \cdot (1 - p_{j,e})$$

と計算できる.

確率変数 $X_j = \sum_{e \in E_S} X_{j,e}$ を定義すると, X_j は $\widetilde{C}_j$ に到達した要素の種類数を表す. $e \in E_S$ に対して $X_{j,e}$ はお互いに独立に値を持つので, X_j の期待値と分散は

$$\mathrm{E}[X_j] = \sum_{e \in E_S} \mathrm{E}[X_{j,e}] = \sum_{e \in E_S} p_{j,e} \tag{3.4}$$

$$\mathrm{Var}[X_j] = \sum_{e \in E_S} \mathrm{Var}[X_{j,e}] = \sum_{e \in E_S} p_{j,e} \cdot (1 - p_{j,e}) \tag{3.5}$$

である.

補題 3.3. e は自然対数の底, 整数 j, α はそれぞれ $j \geq 1$, $\alpha \geq 0$ とする. 出現頻度が $2^{j+\alpha}$ 以上の要素 $e \in C_{j+\alpha}$ に対して,

$$\Pr[X_{j,e} = 1] \geq 1 - \exp(-2^{\alpha}), \tag{3.6}$$

$$\Pr[X_{j,e} = 0] \leq \exp(-2^{\alpha}), \tag{3.7}$$

が成立する. また, 出現頻度が $2^{j-\alpha}$ 未満の要素 $e \in E_S \setminus C_{j-\alpha}$ に対して次の不等式が成り立つ.

$$\Pr[X_{j,e} = 1] < 1 - 1/4^{2^{-\alpha}}, \tag{3.8}$$

$$\Pr[X_{j,e} = 0] > 1/4^{2^{-\alpha}}. \tag{3.9}$$

C_j は定義より出現頻度が 2^j 以上の要素の集合なので, それの推定集合 $\widetilde{C}_j$ には出現頻度が 2^j 以上の要素は含んで, 2^j 未満の要素は含まないことが望ましい. 補題 3.3 の意味は, j を固定した $C_j = \{e \in E_S : n_e \geq 2^j\}$ に対して, 出現頻度が $2^{j+\alpha}$ より大きい要素は $\widetilde{C}_j$ に到着する可能性が高くて, 出現頻度が $2^{j-\alpha}$ より小さい要素は $\widetilde{C}_j$ に到達する確率が低いことを意味する.

補題 3.3 の証明. ストリームで n_e 回出現した要素 e が $\widetilde{C}_j$ に含まれる確率は式 (3.3) より $1 - (1 - 1/2^j)^{n_e}$ である. これは多く出現する要素ほど $\widetilde{C}_j$ に含まれる確率が高くなることを意味する. 出現頻度が $2^{j+\alpha}$ 以上の要素 $e (e \in C_{j+\alpha})$ に対して,

$$\begin{aligned} \Pr[X_{j,e} = 1] &= 1 - \Big(1 - \frac{1}{2^j}\Big)^{n_e} \\ &\geq 1 - \Big(1 - \frac{1}{2^j}\Big)^{2^{j+\alpha}} \end{aligned}$$

$$\begin{aligned} &= 1 - \left(\left(1 - \frac{1}{2^j}\right)^{2^j}\right)^{2^\alpha} \\ &\geq 1 - \mathrm{e}^{-2^\alpha} = 1 - \exp(-2^\alpha) \end{aligned} \tag{3.10}$$

が成り立つ. 不等式 (3.10) は $j \geq 1$ に対して $(1 - \frac{1}{2^j})^{2^j} \leq \mathrm{e}^{-1}$ なので成立する. 出現頻度が $n_e \geq 2^{j+\alpha}$ の要素が $|\widetilde{C}_j|$ に含まらない確率は $\exp(-2^\alpha)$ 以下である.

同様に出現頻度が $n_e < 2^{j-\alpha}$ である要素 e に対して,

$$\begin{aligned} \Pr[X_{j,e} = 1] &= 1 - \left(1 - \frac{1}{2^j}\right)^{n_e} \\ &< 1 - \left(1 - \frac{1}{2^j}\right)^{2^{j-\alpha}} \\ &= 1 - \left(\left(1 - \frac{1}{2^j}\right)^{2^j}\right)^{2^{-\alpha}} \\ &< 1 - 1/4^{2^{-\alpha}} \end{aligned} \tag{3.11}$$

が成立する. 不等式 (3.11) は $j \geq 1$ に対して $(1 - \frac{1}{2^j})^{2^j} \geq 1/4$ なので成り立つ. □

補題 3.3 を使って定理 3.1 が証明できる.

定理 3.1 の証明. Simple Algorithm が出力する $|\widetilde{C}_j|$ を確率変数 X_j だと考える. 証明のアイディアは確率変数 X_j の期待値 ($\mathrm{E}[X_j] = \sum_{e \in E_S} p_{j,e}$) を $C_{j+\alpha}$ に含まれる要素とそうでない要素に分けて計算する.

$$\begin{aligned} \mathrm{E}[X_j] &= \sum_{e \in E_S \setminus C_{j+\alpha}} p_{j,e} + \sum_{e \in C_{j+\alpha}} p_{j,e} \\ &\geq \sum_{e \in E_S \setminus C_{j+\alpha}} p_{j,e} + (1 - \mathrm{e}^{-(\alpha+1)}) \cdot |C_{j+\alpha}| \qquad (3.12) \\ &\geq (1 - \exp(-2^\alpha)) \cdot |C_{j+\alpha}| \qquad (3.13) \end{aligned}$$

式 (3.12) は補題 3.3 より成り立ち, 式 (3.13) は確率 $p_{j,e} \geq 0$ より成立する. この不等式は 0 以上, $s - j$ 未満のすべての整数 α に対して成り立つので, $\mathrm{E}[X_j]$ は $\max_{0 \leq \alpha < (s-j)}\{(1 - \exp(-2^\alpha)) \cdot |C_j|\}$ より下から抑えられる.

$\mathrm{E}[X_j]$ の上界も同様に計算できる.

$$\begin{aligned} \mathrm{E}[X_j] &= \sum_{e \in E_S \setminus C_{j-\alpha}} p_{j,e} + \sum_{e \in C_{j-\alpha}} p_{j,e} \\ &< \left(1 - 1/4^{2^{-\alpha}}\right) \cdot (|C_0| - |C_{j-\alpha}|) + \sum_{e \in C_{j-\alpha}} p_{j,e} \end{aligned} \tag{3.14}$$

$$\leq \left(1 - 1/4^{2^{-\alpha}}\right) \cdot (|C_0| - |C_{j-\alpha}|) + |C_{j-\alpha}|$$
$$= \left(1 - 1/4^{2^{-\alpha}}\right) \cdot |C_0| + 1/4^{2^{-\alpha}} |C_{j-\alpha}| \tag{3.15}$$

式 (3.14) は補題 3.3 より成立する. 任意の $0 \leq \alpha \leq j$ に対して, 不等式 (3.15) が成り立つので,

$$\mathrm{E}[X_j] < \min_{0 \leq \alpha \leq j} \left\{ \left(1 - 1/4^{2^{-\alpha}}\right) \cdot |C_0| + 1/4^{2^{-\alpha}} |C_{j-\alpha}| \right\}, \tag{3.16}$$

が成立し, $\mathrm{E}[X_j]$ を上から抑えることができる. 更に, X_j と $|C_0|$ の差の下界を表すことも出来る.

$$\mathrm{E}[|C_0| - X_j] \geq \max_{0 \leq \alpha \leq j} \left\{ (|C_0| - |C_{j-\alpha}|) \cdot \left(\frac{1}{4}\right)^{2^{-\alpha}} \right\} \tag{3.17}$$

□

3.3 Main Algorithm

Simple Algorithm では出現したアイテム ($j \in \{0, 1, \ldots, s\}, \widetilde{C}_j$ の要素) を全て覚えておく必要があった. この記憶領域の削減のため, Bar-Yossef 達 [2] のアルゴリズムを応用する.

3.3.1 アルゴリズムと精度保証定理

本研究で提案する Main Algorithm の疑似コードを Algorithm3 に記す. このアルゴリズムは入力としてストリーム S とパラメータ t を使い $|C_j|$ の推定値 $|\widetilde{C}_j|$ を出力する. ただし, 仮定としてストリーム長さ N と要素の出現範囲 $[1, m]$ は分かっているが, N と $m(> N)$ は非常に大きいとする. アルゴリズムの出力結果と真の値との差について次の定理と系が成り立つ.

定理 3.4. $0 < \epsilon < 1$ とする. *Algorithm3* の出力 $|\hat{C}_j|$ に対して次の不等式が成り立つ.

$$\Pr[|\hat{C}_j| > (1 + \epsilon) b_j] < \frac{4}{\epsilon^2 t}, \tag{3.18}$$
$$\Pr[|\hat{C}_j| < (1 - \epsilon) a_j] < \frac{2 b_j}{\epsilon^2 t a_j}. \tag{3.19}$$

a_j と b_j はそれぞれ次の値とする.

$$a_j = \max_{0 \leq \alpha < (s-j)} \{(1 - \exp(-2^{\alpha})) \cdot |C_{j+\alpha}|\},$$
$$b_j = \min_{0 \leq \alpha \leq j} \left\{ \left(1 - 1/4^{2^{-\alpha}}\right) \cdot |C_0| + 1/4^{2^{-\alpha}} |C_{j-\alpha}| \right\}.$$

Algorithm 3 Main Algorithm

Require: data stream $S = e_1, \ldots, e_N (e_i \in [1, m])$, パラメータ t, s

1: M を m^3 より大きい素数の中で一番小さい素数とする.
2: α, β を $[1, M]$ から独立ランダムに選ぶ.
3: **for** $i = 1$ **to** N **do** //ストリーム S からデータを読み始める
4: 公平なコインを初めて表が出るまで投げ続ける (最大 (s+1) 回まで). このとき投げた回数を x とする.
5: **for** $j = 0$ **to** $x - 1$ **do**
6: ハッシュ関数 $h(e) = (\alpha \cdot e + \beta \mod M) + 1$ を使って e_i のハッシュ値を計算し, 小さい順で t 個のハッシュ値を記憶するリスト L_j を更新する.
7: **end for**
8: **end for**
9: **for** $j = 0$ **to** s **do**
10: **if** L_j のサイズ $< t$ **then**
11: $|\hat{C}_j| \leftarrow L_j$ のサイズ
12: **else**
13: $|\hat{C}_j| \leftarrow \frac{tm^3}{v_j}$ (v_j は L_j の中で一番大きいハッシュ値)
14: **end if**
15: **end for**
16: **return** $|\hat{C}_0|, \ldots, |\hat{C}_s|$

系 3.5. *Algorithm3* でパラメータ $t = \frac{12}{\epsilon^2}\max(2, \frac{b_j}{a_j})$ とすれば,

$$\Pr[(1-\epsilon)a_j < |\hat{C}_j| < (1+\epsilon)b_j] \geq \frac{2}{3} \tag{3.20}$$

が成立する. さらに, *Algorithm3* を $\lceil 15\log_2(\delta^{-1})\rceil$ 個独立並行に実行させてそれらの中間値 $|\hat{C}_j|$ とすると,

$$\Pr[(1-\epsilon)a_j < |\hat{C}_j| < (1+\epsilon)b_j] \geq 1-\delta \tag{3.21}$$

が成立する.

定理 3.4 と系 3.5 の意味は, パラメータ t を調整して Algorithm3 を並行に実行することで, $|C_j|$ の推定値を高い確率で $(1-\epsilon)a_j$ 以上 $(1+\epsilon)b_j$ 以下の間で出力することである.

提案アルゴリズムの空間複雑度について述べる. Algorithm3 はハッシュ関数のために α と β を $[1, m^3]$ から選んで $\mathrm{O}(\log m)$ bits のメモリ空間を使って記憶する. 次に $\mathrm{O}(\log m)$bits のハッシュ値を最大 t 個記録するリストを $s = \mathrm{O}(\log N)$ 個使う. 従って, Algorithm 3 は $\mathrm{O}(t\log m \cdot \log N)$ bits のメモリを使う. 系 3.5 で $\Pr[(1-\epsilon)a < |\hat{C}_j| < (1+\epsilon)b] \geq 1-\delta$ を満たさせるために, $t = \frac{12}{\epsilon^2}\max(2, \frac{b_j}{a_j})$ とおいて Algorithm3 を $r = \lceil 15\log_2(1/\delta)\rceil$ 回平行に実行させる. b_j/a_j を定数として考えるなら, $t = \mathrm{O}(\epsilon^{-2})$ となり, このときの空間複雑度は, $\mathrm{O}(\epsilon^{-2}\log(\delta^{-1})\log m \cdot \log N)$ bits である. $N < m$ の前提では, 空間複雑度は $\mathrm{O}(\epsilon^{-2}\log(\delta^{-1})\log^2 m)$ bits となる.

3.3.2 理論解析

まず, Algorithm3 の詳しい動作と使うデータ構造を説明する. 次に定理 3.4 と系 3.5 を証明する.

Main Algorithm が各要素をサンプリングする方法は前節で説明した Simple Algorithm の方法と同じである. しかし, Simple Algorithm のように $\widetilde{C}_j$ に含まれる要素をすべて記録するではなく, ランダムに選べられた一つのハッシュ関数と t 個のハッシュ値を記録するリストを用いて $|\widetilde{C}_j|$ を推定する. 言い換えれば, Main Algorithm は元の問題である $|C_j|$ を推定するために, $|\widetilde{C}_j|$ を推定する値 $|\hat{C}_j|$ を出力するアルゴリズムである.

ハッシュ関数とリストについて説明する. まず, ハッシュ関数はペアワイズ独立性を満たすハッシュ関数族から独立ランダムに選べられたもので, ストリームで出現する要素 $e \in [1, m]$ を $[1, M]$ の範囲で一様ランダムにマッピングする. ただし, M は m^3 より大きい素数の中で一番小さい素数である. 具体的にハッシュ関数族 $\mathcal{H}$ を次のようにハッシュ関数 $h_{\alpha,\beta}$ の集合とする.

$$\mathcal{H} = \{h_{\alpha,\beta}(e) : (\alpha,\beta) \in [1,M]\times[1,M], h_{\alpha,\beta}(e) = (\alpha\cdot e + \beta \mod M) + 1\}$$

$[1, M]$ から一様ランダムに α,β を選ぶことで, ハッシュ族 $\mathcal{H}$ からランダムにハッシュ関数 h が選択される. 2.4 節で説明したように, ハッシュ関数族 $\mathcal{H}$ はペアワイズ独立性を満たす. このハッシュ関数よりストリームから出現した任意の要素 e のハッシュ値は $[1, M]$ の範囲で一様ランダムに値を取る.

次に, リスト L_j は $|\widetilde{C}_j|$ を推定するために使うもので, 各リストは最大 t 個のハッシュ値が記憶できる. リスト L_j のサイズとは, リスト L_j の中に記録されているハッシュ値の数とする.

Main Algorithm はリスト L_j を使い $\widetilde{C}_j$ に対して小さい順で t 個のハッシュ値を更新していく. すなわち, i 番目に読んだ要素 e_i のハッシュ値 $h(e)$ が各 L_j の中にあるハッシュ値より小さければ, $h(e_i)$ の中で最も大きいハッシュ値を除去し $h(e_i)$ を追加する. ここで注意することは, 既に同じハッシュ値がリストに記憶されていたら, 何も動作しないことである. そうしないと, ハッシュ値が小さい要素が多数出現したときに, リストにはその要素のハッシュのみが記録されるようになる. 異なる要素が同じハッシュ値を持つならば誤差が発生する問題があるが, m が非常に大きいと仮定したので, 異なる要素が同じハッシュ値を持つ確率 $M^{-2}(\simeq m^{-6})$ は異常に小さくて無視できると考える.

ストリームデータが読み終わったらリストを使って $|\widetilde{C}_j|$ の推定値として $|\hat{C}_j|$ を計算する. リスト L_j に記録されているハッシュ値の数が t より小さい場合は, そのハッシュ値の数を $|\widetilde{C}_j|$ にする. そうでなければ, リスト L_j に記録されているハッシュ値の中で一番大きい値 v_j(全体で t 番目に小さいハッシュ値) を使って $\frac{tm^3}{v_j}$ の値を $|\hat{C}_j|$ とする.

定理 3.4 の証明. 定理 3.4 の意味は, Algorithm3 の出力値 $|\hat{C}_j|$ が Simple Algorithm の定理 3.1 で示した下界 a_j と上界 b_j から離れる確率が入力パラメータ t に依存することである. まず, $\Pr[|\hat{C}_j| > (1+\epsilon)b_j] < \frac{4}{\epsilon^2 t}$ を証明する.

$$\Pr[|\hat{C}_j| > (1+\epsilon)b_j] = \Pr[v_j < \frac{tm^3}{(1+\epsilon)b_j}] \tag{3.22}$$

$$< \Pr[v_j < (1-\frac{\epsilon}{2})\frac{tm^3}{b_j}] \tag{3.23}$$

式 (3.22) は $|\hat{C}_j| = \frac{tm^3}{v_j}$ の定義より, 式 (3.23) は $0 < \epsilon < 1$ に対して, $1/(1+\epsilon) < 1 - \epsilon/2$ なので成立する. 一方, v_j は $\widetilde{C}_j$ に到達した要素のハッシュ値の中で t 番目に小さい値である. v_j がある値 α より小さいことは, $\widetilde{C}_j$ に到達した要素の中でハッシュ値が α より小さい要素が t 個を超えて存在することを意味する. 確率変数 $Y_{j,e}$ を

$$Y_{j,e} = \begin{cases} 1 & \text{if } (e \in \widetilde{C}_j \text{ and } h(e) < (1-\epsilon/2)\frac{tm^3}{b_j}), \\ 0 & \text{otherwise,} \end{cases}$$

とする. すべての要素に対して $Y_{j,e}$ の合計を $Y_j = \sum_{e \in E_S} Y_{j,e}$ と定義すると, 式 (3.23) の $\Pr[v_j < (1-\epsilon/2)\frac{tm^3}{b_j}]$ は $\Pr[Y_j > t]$ と同値である.

$\Pr[Y_{j,e} = 1]$ は, 要素 e が $\widetilde{C}_j$ に到着する事象と要素 e のハッシュ値が $(1-\epsilon/2)\frac{tm^3}{b_j}$ より小さい事象が独立なので, $\Pr[e \in \widetilde{C}_j] = p_{j,e}$ と $\Pr[h(e) < (1-\epsilon/2)\frac{tm^3}{b}]$ の積で求められる. ハッシュ関数 $h(e)$ は任意の要素 $e \in [1, m]$ に対して $[1, m^3]$ の間の一様にマッピングするので, $h(e)$ が $(1-\epsilon/2)\frac{tm^3}{b_j}$ より小さい確率は $(1-\epsilon/2)\frac{t}{b_j}$ である. 式 (3.4) と (3.16) より $\sum_{e \in E_S} p_{j,e} < b_j$ なので, Y_j の期待値について次の不等式が成り立つ.

$$\begin{aligned}
\mathrm{E}[Y_j] &= \sum_{e \in E_S} \mathrm{E}[Y_{j,e}] = \sum_{e \in E_S} \Pr[Y_{j,e} = 1] \\
&= \sum_{e \in E_S} \Pr[e \in \widetilde{C}_j] \cdot \Pr[h(e) < (1-\epsilon/2)\frac{tm^3}{b_j})] \\
&= \sum_{e \in E_S} p_{j,e}(1-\epsilon/2)\frac{t}{b_j} \\
&= (1-\epsilon/2)\frac{t}{b_j} \sum_{e \in E_S} p_{j,e} \\
&< (1-\epsilon/2)t
\end{aligned} \tag{3.24}$$

Y_j の分散についてはハッシュ関数 $h(e)$ のペアワイズ独立性より次の不等式が得られる.

$$\begin{aligned}
\mathrm{Var}[Y_j] &= \sum_{e \in E_S} \mathrm{Var}[Y_{j,e}] = \sum_{e \in E_S} \Pr[Y_{j,e} = 1] \cdot (1 - \Pr[Y_{j,e} = 1]) \\
&= \sum_{e \in E_S} \big(p_{j,e}(1-\epsilon/2)\frac{t}{b_j}\big) \cdot \big(1 - p_{j,e}(1-\epsilon/2)\frac{t}{b_j}\big) \\
&\leq \sum_{e \in E_S} \big(p_{j,e}(1-\epsilon/2)\frac{t}{b_j}\big) \\
&= (1-\epsilon/2)\frac{t}{b_j} \sum_{e \in E_S} p_{j,e} \\
&\leq (1-\epsilon/2)t \\
&< t
\end{aligned} \tag{3.25}$$

不等式 (3.24) より $t - \mathrm{E}[Y_j] > \frac{\epsilon t}{2}$ なので, 次の不等式 (3.26) が満たされる.

$$\begin{aligned}
\Pr[Y_i > t] &= \Pr[Y_j - E[Y_j] > t - E[Y_j]] \\
&\leq \Pr[Y_j - E[Y_j] > \frac{\epsilon t}{2}]
\end{aligned} \tag{3.26}$$

$$
\begin{aligned}
&\leq \Pr[|Y_j - E[Y_j]| > \frac{\epsilon t}{2}] \\
&\leq \frac{4\mathrm{Var}[Y_j]}{\epsilon^2 t^2} \\
&< \frac{4t}{\epsilon^2 t^2} \\
&= \frac{4}{\epsilon^2 t}
\end{aligned} \tag{3.27}
$$

不等式 (3.27) は Chebyshev の不等式と不等式 (3.25) より成り立つ. 以上より定理 3.4 の不等式 (3.18) が証明される.

次に定理 3.4 の不等式 (3.19),$\Pr[|\hat{C}_j| < (1-\epsilon)a_j] < \frac{2b_j}{\epsilon^2 t a_j}$ を示す.

$$
\begin{aligned}
\Pr[|\hat{C}_j| < (1-\epsilon)a_j] &= \Pr[v_j > \frac{tm^3}{(1-\epsilon)a_j}] \\
&\leq \Pr[v_j > (1+\epsilon)\frac{tm^3}{a_j}]
\end{aligned} \tag{3.28}
$$

$0 < \epsilon < 1$ の範囲で $\frac{1}{1-\epsilon} > (1+\epsilon)$ なので, 不等式 (3.28) が成り立つ.

t 番目に小さいハッシュ値 v_j が $(1+\epsilon)\frac{tm^3}{a_j}$ を超える確率は, ハッシュ値が $(1+\epsilon)\frac{tm^3}{a_j}$ 未満の要素の数が t 未満である確率と同値である. 確率変数 $Z_{j,e}$ を要素 e のハッシュ値が $(1+\epsilon)\frac{tm^3}{a_j}$ より小さければ 1, そうでなければ 0 の値を持つ確率変数と定義し Z_j は出現したすべての要素に対して $Z_{j,e}$ の合計とする. すなわち,

$$
Z_{j,e} = \begin{cases} 1 & \text{if } (e \in \widetilde{C}_j \text{ and } h(e) < (1+\epsilon)\frac{tm^3}{a_j}), \\ 0 & \text{otherwise,} \end{cases}
$$

とし,

$$
Z_j = \sum_{e \in E_S} Z_{j,e}
$$

と定義すると,

$$
\Pr[v_j > (1+\epsilon)\frac{tm^3}{a_j}] = \Pr[Z_j < t] \tag{3.29}
$$

が成立する. 次に Z_j の期待値と分散を求めてこの確率を上から抑える. Z_j の期待値は

$$
\mathrm{E}[Z_j] = \sum_{e \in E_S} \mathrm{E}[Z_{j,e}] = \sum_{e \in E_S} \Pr[Z_{j,e} = 1]
$$

と計算できる. そして $\Pr[Z_{j,e}=1] = \Pr[e \in \widetilde{C}_j] \cdot \Pr[h(e) < (1+\epsilon)\frac{tm^3}{a_j}] = p_{j,e} \cdot \frac{(1+\epsilon)t}{a_j}$ と $a_j \leq \sum_{e\in E_S} p_{j,e} < b_j$ より, Z_j の期待値について

$$(1+\epsilon)t \leq \mathrm{E}[Z_j] < (1+\epsilon)\frac{tb_j}{a_j} \tag{3.30}$$

が満たされる. 不等式 (3.30) より,

$$\mathrm{E}[Z_j] - t \geq \epsilon t \tag{3.31}$$

が成り立つ. 分散については,

$$\begin{aligned}
\mathrm{Var}[Z_j] &= \mathrm{E}[Z_j^2] - \mathrm{E}[Z_j]^2 \\
&\leq \mathrm{E}[Z_j^2] \\
&= \mathrm{E}[Z_j] \\
&< (1+\epsilon)\frac{tb_j}{a_j} \\
&< \frac{2tb_j}{a_j}
\end{aligned} \tag{3.32}$$

が成立する. 最後に不等式（(3.31),(3.32)）と Chebyshev の不等式を使うと次が成り立つ.

$$\begin{aligned}
\Pr[Z_i < t] &\leq \Pr[|Z_j - \mathrm{E}[Z_j]| > \mathrm{E}[Z_j] - t] \\
&\leq \Pr[|Z_j - \mathrm{E}[Z_j]| > \epsilon t] \\
&\leq \frac{\mathrm{Var}[Z_j]}{\epsilon^2 t^2} \\
&\leq \frac{2tb_j}{\epsilon^2 t^2 a_j} \\
&= \frac{2b_j}{\epsilon^2 t a_j}.
\end{aligned} \tag{3.33}$$

□

系 3.5 の証明. $t = \frac{12}{\epsilon^2}\max(2, \frac{b_j}{a_j})$ とすれば, 不等式 (3.18),(3.19) から次の不等式が成り立つ.

$$\Pr[|\hat{C}_j| > (1+\epsilon)b_j] < \frac{4}{\epsilon^2 t} < \frac{1}{6} \tag{3.34}$$

$$\Pr[|\hat{C}_j| < (1-\epsilon)a_j] < \frac{2b_j}{\epsilon^2 t a_j} < \frac{1}{6}. \tag{3.35}$$

従って, $\Pr[(1-\epsilon)a_j < |\hat{C}_j| < (1+\epsilon)b_j] \geq 2/3$ である. $r = \lceil 15\log_2(1/\delta) \rceil$ とおいて, r 回独立に Algorithm3 を実行させて $\hat{C}_j$ の中間値を取れば, Chernoff 不等式より $\Pr[(1-\epsilon)a < |\hat{C}_j| < (1+\epsilon)b] \geq 1-\delta$ が成立する.

□

第 4 章

実験

本章では Simple Algorithm と Main Algorithm について計算機実験を行う. 扱うデータは独立行政法人情報通信研究機構 (NICT) が管理するダークネットデータである.

4.1 実験環境

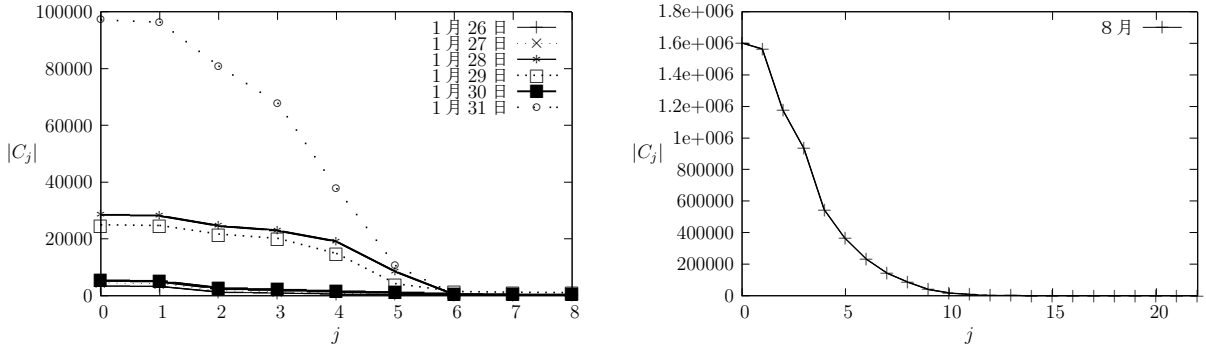
実験に用いた計算機のスペックは表 4.1 のようである. アルゴリズムの実装で乱数の生成はメルセンヌ・ツイスタを用いた.

表 4.1: 実験に用いた計算機のスペック

OS	CentOS release 5.9(Final)
CPU	Intel(R) Core(TM) i5-3330 CPU @3.00GHz
RAM	8GB
言語	Java 1.7.0_65
開発環境	Eclipse (Luna Release 4.4.0)

4.2 実験データ

独立行政法人情報通信研究機構 (NICT) が管理するダークネットデータを用いる. このデータは 2014 年 1 月 26 日から 1 月 31 日までの 1 日間及び 2014 年 8 月の 1 か月の間に, ダークネット IP アドレスにパケットを送信した IP アドレス (IPv4) の列である. IP



(a) 2014 年 1 月 26 日から 1 月 31 日まで

(b) 2014 年 8 月

図 4.1: 出現頻度分布

アドレスは $(0,0,0,0) \sim (255.255.255.255)$ の範囲なので, IP アドレスの列は $[1, 2^{32}]$ からなるストリームデータ ($m = 2^{32}$) として考えることができる. 日付ごとのパケット数 N とユニーク IP 数 $|E_S|$ は表 4.2 となる.

表 4.2: 日付ごとのパケット数とユニーク IP 数

日付	パケット数 ($= N$)	ユニーク IP 数 ($= |E_S|$)
1 月 26 日	1,631,053	3,488
1 月 27 日	1,548,938	4,597
1 月 28 日	2,059,759	28,568
1 月 29 日	3,205,750	25,026
1 月 30 日	2,115,686	5,293
1 月 31 日	3,441,658	97,058
8 月	186,042,309	1,601,355

各データに対して出現頻度分布を求めた結果は表 4.3 である. 図 4.1(a) は 2014 年 1 月 26 日から 31 日までの日ごとの出現頻度分布を, 横軸を $j = 0$ から 8 まで, 縦軸を $|C_j|$ で表したものである. 同様に 2014 年 8 月の出現頻度分布のグラフは図 4.1(b) のように表される.

表 4.3: 日付と出現頻度分布 $|C_j|$

	1 月 26 日	1 月 27 日	1 月 28 日	1 月 29 日	1 月 30 日	1 月 31 日	8 月
$\|C_0\|$	3,488	4,579	28,568	25,026	5,293	97,058	1,601,355
$\|C_1\|$	3,280	4,298	28,162	24,724	5,044	96,323	1,561,964
$\|C_2\|$	1,300	1,783	24,607	21,662	2,615	80,806	1,175,247
$\|C_3\|$	980	1,324	22,936	20,180	2,087	67,732	936,682
$\|C_4\|$	583	826	19,101	14,768	1,486	37,704	542,739
$\|C_5\|$	394	548	8,520	4,230	1,041	10,536	362,797
$\|C_6\|$	252	307	479	1,456	560	546	233,329
$\|C_7\|$	188	209	320	1,240	441	380	144,222
$\|C_8\|$	146	151	252	1,063	381	313	86,889
$\|C_9\|$	113	109	175	864	307	194	40,040
$\|C_{10}\|$	96	101	92	550	138	167	17,047
$\|C_{11}\|$	64	61	62	255	96	82	6,979
$\|C_{12}\|$	56	57	55	121	69	70	3,030
$\|C_{13}\|$	39	34	43	44	52	51	1,537
$\|C_{14}\|$	25	24	29	30	38	35	838
$\|C_{15}\|$	19	15	20	21	28	26	317
$\|C_{16}\|$	5	3	2	3	4	5	176
$\|C_{17}\|$	1	1	0	0	0	1	91
$\|C_{18}\|$	0	1	0	0	0	0	44
$\|C_{19}\|$	0	0	0	0	0	0	24
$\|C_{20}\|$	0	0	0	0	0	0	11
$\|C_{21}\|$	0	0	0	0	0	0	2
$\|C_{22}\|$	0	0	0	0	0	0	1
$\|C_{23}\|$	0	0	0	0	0	0	0

4.3 Simple Algorithm を用いた実験

本節では Simple Algorithm の実証実験を行う. 小節 4.3.1 ではアルゴリズムを実装し, 出力値と真の値の差を考察する. 小節 4.3.2 では Simple Algorithm を複数回実装させて中央値を取る (median 法) 方法を使ってアルゴリズムの改良可否を検討する.

4.3.1 Simple Algorithm の妥当性の検討実験

目的 Simple Algorithm が出力する出現頻度分布の推定値の妥当性を検討する.

データと方法 Simple Algorithm のパラメータ $s = 18$ とする. 2014 年 1 月 26 日から 1 月 31 日のダークネットデータ対して, 日ごとにメルセンヌツイスタのシードを 4357 と固定して Simple Algorithm を実装する. アルゴリズムの出力値と真の値を比較する.

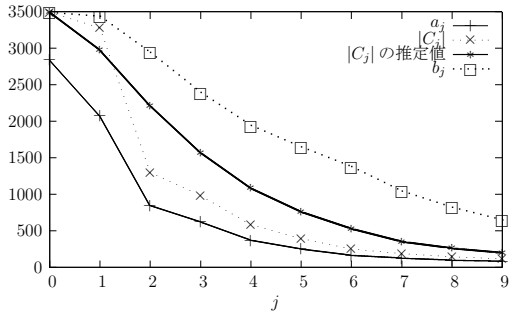
結果 日付ごとに Simple Algorithm から得られた $|C_0|, \ldots, |C_{18}|$ の推定値と真の値,a_j, b_j を図 4.2 に示す.

考察 最初に, アルゴリズムが出力する出現頻度分布の推定値と真の値の差について考察する. 次に, アルゴリズムの出力値と定理 3.1 の関係について述べる.

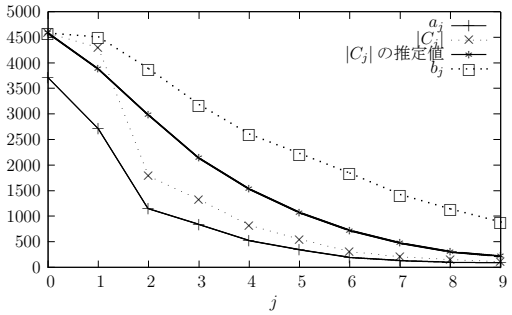
まず, 2014 年 1 月 29 日データを用いた実験結果, 図 4.2 のグラフ (d) を使って真の値とアルゴリズムによる推定値を比較する. $j = 0, \ldots, 4$ では推定値が真の値と近くて良い近似が出来ている. しかし, $j = 5$ 以上では推定値と真の値の差が大きくなることが分かる. $|C_4|$ と比べて $|C_5|$ の値が急に小さくなったので, これが誤差の増加の原因として考えられる. この傾向は他の日付でも見られる. 例えば, 26 日の $j = 1$ から 2, 27 日の $j = 1$ から 2, 28 日の $j = 4$ から 5 の間も, $|C_j|$ より $|C_{j+1}|$ が急に小さくなったときに, アルゴリズムの推定値の誤差が大きくなった. しかし, この実験は疑似乱数をを使ったので, 偶然に誤差が発生した可能性もありうる. 従って, 小節 4.3.2 で複数回アルゴリズムを実行させて中央値を取る実験を行い, アルゴリズムの妥当性について考察を続ける.

次に, アルゴリズムの出力値と定理 3.1 の関係について述べる 定理 3.1 より, Simple Algorithm の出力値の期待値 $\mathrm{E}[|\widetilde{C}_j|]$ について

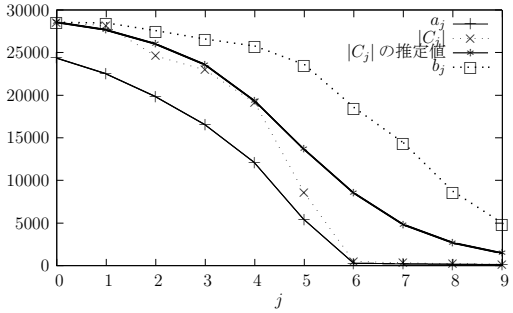
$$a_j \leq \mathrm{E}[|\widetilde{C}_j|] < b_j$$



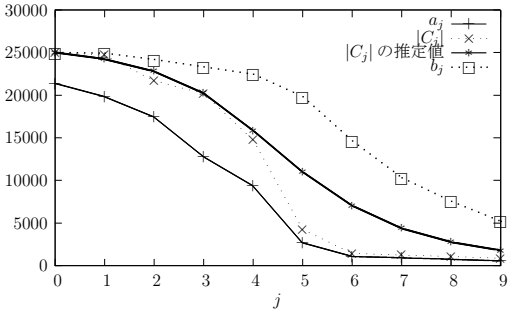
(a) 2014 年 1 月 26 日



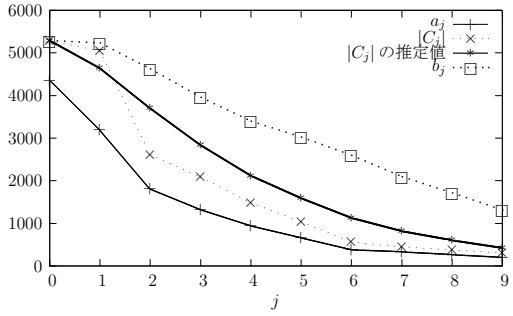
(b) 2014 年 1 月 27 日



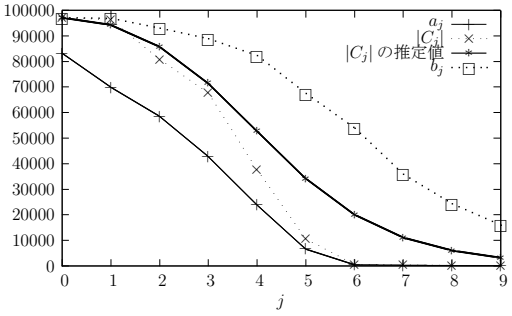
(c) 2014 年 1 月 28 日



(d) 2014 年 1 月 29 日



(e) 2014 年 1 月 30 日



(f) 2014 年 1 月 31 日

図 4.2: 2014 年 1 月 26 日から 31 日のデータを用いて Simple Algorithm の実装実験結果

が成り立つ. ここで a_j と b_j はストリーム S により決まる次の値である.

$$\begin{aligned} a_j &= \max_{0\le\alpha<(s-j)}\{(1-\exp(-2^{\alpha})\cdot|C_{j+\alpha}|\}, \\ b_j &= \min_{0\le\alpha\le j}\left\{\left(1-1/4^{2^{-\alpha}}\right)\cdot|C_0|+1/4^{2^{-\alpha}}|C_{j-\alpha}|. \right. \end{aligned}$$

図 4.2 を見ると, 全ての日付に対してアルゴルリズムが出力する $|C_j|$ の推定値は定理 3.1 のように a_j と b_j の間に抑えられていることが確認できた.

4.3.2 Median 法より Simple Algorithm の向上検討実験

目的 Median 法より Simple Algorithm の精度を上げられるか検証する.

データと方法 Simple Algorithm のパラメータ $s = 22$ とする. 入力データは 2014 年 8 月のダークネットデータを用いる. メルセンヌツイスタのシードを初めに 4357 と設定して, $k = 1, 5, 15$ 回連続で繰り返して Simple Algorithm を実装する. これらの出力値の中央値と真の値を比較する.

結果 2014 年 8 月のダークネットデータに対して, $k = 1, 5, 15$ 回連続で繰り返して Simple Algorithm を実装した結果の中央値を $|C_j|$, a_j,b_j と一緒に図 4.3 に示す. $k = 1, 5, 15$ の推定値は差が小さくグラフで区別が難しいので, $j = 0$ から 7 までの結果を表 4.4 に表示する.

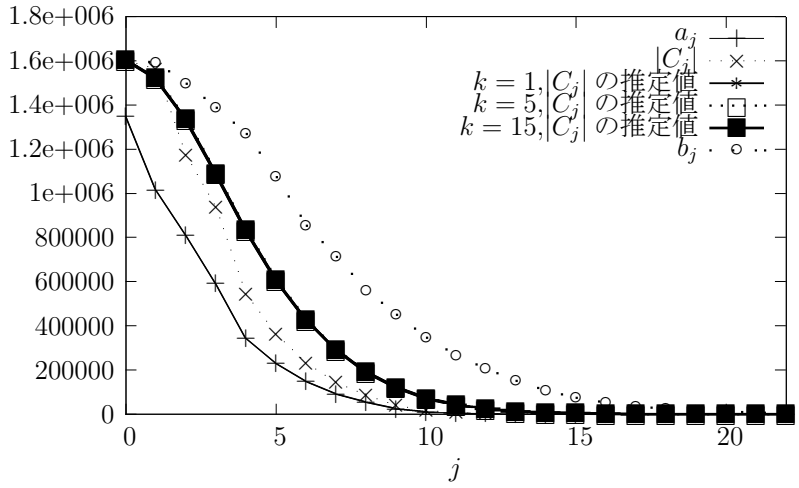


図 4.3: 2014 年 8 月のデータに対して Simple Algorithm を k 回実装して中央値を求めた実験結果のグラフ

表 4.4: 2014 年 8 月のデータに対して Simple Algorithm を k 回実装して中央値を求めた実験結果 ($j = 0, \ldots, 7$)

j	$\lvert C_j \rvert$	a_j	b_j	$k = 1$, $\lvert C_j \rvert$ の推定値	$k = 5$, $\lvert C_j \rvert$ の推定値	$k = 15$, $\lvert C_j \rvert$ の推定値
0	1,601,355	1,350,574.87	1,601,355.00	1,601,355	1,601,355	1,601,355
1	1,561,964	1,016,194.40	1,591,507.25	1,521,840	1,521,527	1,521,368
2	1,175,247	809,915.70	1,494,828.00	1,335,391	1,335,140	1,335,271
3	936,682	592,095.71	1,388,301.00	1,089,757	1,089,183	1,089,227
4	542,739	343,076.34	1,269,018.50	833,225	833,386	833,290
5	362,797	229,331.35	1,072,047.00	607,039	607,962	607,608
6	233,329	147,492.00	852,800.44	427,066	427,383	427,066
7	144,222	91,165.65	11,168.60	290,668	291,096	290,877

考察 図 4.3 と表 4.4 を見ると, $k = 1, 5, 15$ のときの推定値はわずかの差だけ存在してほぼ同じ値である. 従って, Simple Algorithm を繰り返してやって中央値を取っても $|C_j|$ との誤差は改善できないことが分かった.

誤差が発生する理由として, Simple Algorithm の出力値 $|\widetilde{C}_j|$ はかなり高い確率で真の値 $|C_{j-1}|$ に影響を受けることが考えられる. すなわち, 現在の予想は, 出力頻度が 2^{j-1} 以上, 2^j 未満の要素は $\widetilde{C}_j$ に含まれてほしくないが, 高い確率で多数の要素が $\widetilde{C}_j$ に含まれてしまい, $|C_j|$ の推定値が真の値より大きくなることである. これが正しければ, 小節 4.3.1 の考察で述べたように, $|C_{j-1}|$ より $|C_j|$ が急に小さくなるデータに対して, アルゴリズムは $|C_{j-1}|$ の影響をもらって真の値 $|C_j|$ より大きい値を推定値として出力することが説明できる. 従って, 今後の課題はこのアルゴリズムの誤差が発生する理由を検討し, 真の出現頻度分布と近い分布を出力するアルゴリズムの開発である.

4.4 Main Algorithm の実験結果

Main Algorithm の実装実験を行う.

4.4.1 パラメータ t による Main Algorithm の実装実験

目的 パラメータ t による Main Algorithm の出力値と Simple Algorithm の結果を比較する.

データと方法 Main Algorithm のパラメータ $s = 18$ とする. 入力データは 2014 年 1 月

31 日のダークネットデータを用いる. データから出現する IP アドレスは $(0,0,0,0) \sim (255.255.255.255)$ の範囲なので, 各 IP アドレスを $[1, 2^{32}]$ の間の整数とする. 従って, データの出現範囲 $m = 2^{32}$ であり, アルゴリズムの中で使う, m^3 より大きい素数の中で一番小さい素数 M は,

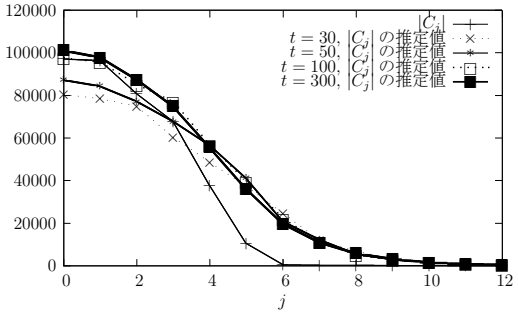
$$M = 79228162514264337593543950397$$

である. メルセンヌツイスタのシードを初めに 7343 と設定して, パラメータ $t = 30, 50, 100, 300$ と変えながら Main Algorithm を実装する. 各パラメータに対して $k = 1, 5, 15$ 回連続で繰り返してこれらの出力値の中央値と Simple Algorithm の結果を比較する.

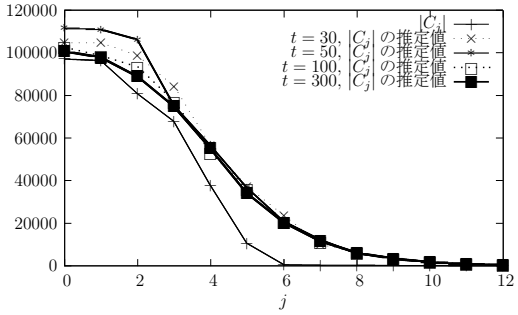
結果 2014 年 1 月 31 日のダークネットデータに対して, パラメータ $t = 30, 50, 100, 500$ ごとに $k = 1, 5, 15$ 回連続で繰り返して Main Algorithm を実装した結果の中央値を $|C_j|$ と一緒に図 4.4 に示す.

考察 Main Algorithm の実装実験結果を小節 4.3.1 で見せた Simple Algorithm による同じ日の結果, 図 4.2 の (f) と比較すると, t を 100 以上にして $k = 5$ 回実装し中央値を取れば, ほぼ同じが結果が得られることが分かる. 従って, Simple Algorithm のように $|E_S| = 97,058$ 個の IP アドレスをすべて覚える必要なく, Main Algorithm は t 個のハッシュ値を覚えるだけで同様な出現頻度分布を推定することが確認できた.

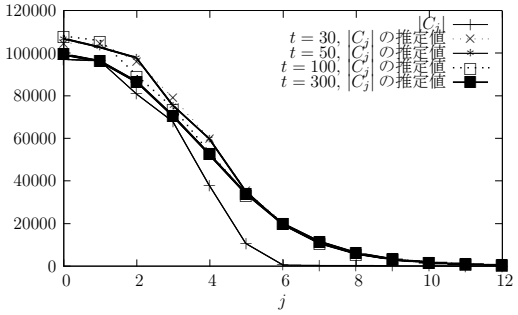
一方, Main Algorithm でも $j = 4$ から真の値 $|C_j|$ との誤差が大きくなっている. 結論的に, この誤差を減らすためにはサンプリング方法を改善するべきだと考えられる.



(a) k=1 回実装した中央値の結果



(b) k=5 回実装した中央値の結果



(c) k=15 回実装した中央値の結果

図 4.4: 2014 年 1 月 31 日ダークネットデータに対する Main Algorithm をパラメータ $t = 30, 50, 100, 300$ と変えながら, $k = 1, 5, 15$ 回実装した中央値の結果

第 5 章

おわりに

本論文ではストリーム中の出現頻度分布を定義し, 出現頻度分布を省領域で推定する乱択アルゴリズムを提案した. 提案アルゴリズムの信頼性について理論解析を行うとともに，実際のダークネットデータを用いて実験を行い提案アルゴリズムの有効性を検証した.

今後の課題は，まず提案アルゴリズムの誤差を減らす方法を検討し, 真の出現頻度分布ともっと近い分布を出力するアルゴリズムの開発である. さらに, 出現頻度分布を使ってストリームの頻度モーメントへの近似方法の提案が次の課題として残っている.

謝辞

本研究を行うにあたって，御多忙にも関わらず熱心に御指導して頂きました山下雅史教授, 来嶋秀治准教授，山内由紀子助教に深謝の意を表します。山下・来嶋研究室の皆様には様々な助言や協力をして頂き感謝致します．最後に，本研究第 4 章の実験で資料を提供して戴いた独立行政法人情報通信研究機構 (NICT) インシデント対策グループの皆様に御礼を申し上げます.

参考文献

[1] N. Alon, Y. Matias and M. Szegedy, The space complexity of approximating the frequency moments, *Proceedings of the 28th annual ACM Symposium on Theory of Computing*(STOC 1996), pp.20–29.

[2] Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan, Counting distinct elements in a data stream, *Randomization and Approximation Techniques in Computer Science*(RANDOM 2002), pp.1–10.

[3] A. Chakrabarti, G. Cormode and A. Mcgregor, A near–optimal algorithm for estimating the entropy of a stream, *ACM Transactions on Algorithm*(TALG 2010), 6(3), 51.

[4] P. Flajolet and G. N. Martin, Probabilistic counting algorithms for data base applications, *Journal of computer and system sciences*(1985), 31(2), pp.182–209.

[5] P. B. Gibbons, Distinct sampling for highly-accurate answers to distinct values queries and event reports, *Proceedings of the 27th International Conference Very Large Data Bases*(VLDB 2001), pp.541–550.

[6] P. B. Gibbons, Distinct-values estimation over data streams, *Data Stream Management: Processing High-Speed Data Streams, (M. Garofalakis, J. Gehrke, and R. Rastogi, Eds.)*(2007), Springer.

[7] S. Guha, N. Koudas, and K. Shim, Approximation and streaming algorithms for histogram construction problems, *ACM Transactions on Database Systems*(TODS 2006), 31(1), pp.396–438.

[8] G. Grimmett and D. Stirzaker, Probability and random processes, *Oxford University Press*, 2001.

[9] P. Indyk, R. Levi, and R. Rubinfeld, Approximating and testing k-histogram distributions in sub-linear time, *Proceedings of the 31st symposium on Principles of Database Systems*(PODS 2012), pp.15–22.

[10] Y. Ioannidis, The history of histograms(abridged), *Proceedings of the 29th International Conference Very Large Data Bases*(VLDB 2003), 29, pp.19–30.

[11] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. Sevcik, and T. Suel, Optimal histograms with quality guarantees, *Proceedings of the 24th International Conference on Very Large Data Bases*(VLDB 1998), pp.275–286.

[12] D. M. Kane, J. Nelson, and D. P. Woodruff, An optimal algorithm for the distinct elements problem, *Proceedings of the 29th ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*(PODS 2010), pp.41–52.

[13] D. M. Kane, J. Nelson, E. Porat, and D. P. Woodruff, Fast moment estimation in data streams in optimal space, *Proceedings of the 43d annual ACM symposium on Theory of computing*(STOC 2011), pp.745–754.

[14] A. Lall, V. Sekar, M. Ogihara, J. Xu and H. Zhang, Data streaming algorithms for estimating entropy of network traffic, *Proceedings of the joint international conference on Measurement and modeling of computer systems*(SIGMETRICS 2006), pp.145–156.

[15] T. Locher, Finding heavy distinct hitters in data streams, *Proceedings of the 23rd ACM Symposium on Parallelism in Algorithms and Architectures*(SPAA 2011), pp.299-308.

[16] G. S. Manku and R. Motwani, Approximate frequent counts over data streams, *Proceedings of the 28th International Conference on Very Large Data Bases*(VLDB 2002), pp.346–357.

[17] M. Mitzenmacher and E. Upfal, Probability and Computing, *Cambridge University Press*, 2005.

[18] R. Motwani and P. Raghavan, Randomized algorithm, *Chapman & Hall/CRC*, 2010.

[19] M. Ogata, Y. Yamauchi, S. kijima, and M. Yamashita, A randomized algorithm for finding frequent elements in streams using O(log log N) space, *Proceeding of the 22nd International Symposium on Algorithms and Computation*(ISAAC 2011), pp.514–523.

[20] R. Sebastiao, J. Gama, and T. Mendonca, Constructing fading histograms from data streams, *Progress in Artificial Intelligence*(2014) 3(1), pp15–28.

[21] S. Venkatamaran, D. Song, P.B. Gibbons, and A. Blum, New streaming algorithms for fast detection of superspreaders, *Proceedings of the 12th ISOC Symposium on Network and Distributed Systems Security*(NDSS 2005), pp.149-166.

[22] 徳山 豪, オンラインアルゴリズムとストリームアルゴリズム，共立出版，2007.