時間と共に変化する多重集合に対する min-hash の高速計算

2031136 三原 寛寿

指導教員 古賀 久志 准教授

副指導教員 戸田 貴之

2022 年 1 月 28 日

目次

第 1 章

はじめに

ストリームデータとは時間経過と共に継続的に次々と生成されるデータのことを言う．近年，IoT や SNS の発展に伴いストリームデータが取り扱われる機会が増加している．例えば，IoT におけるセンサからの観測データはストリームデータである．また，特定ユーザの twitter におけるツイートやウェブページの閲覧履歴も時間と共に新データが追加されるという点でストリームデータである．こうしてストリームデータの増加に連れて，ストリームデータを対象とした類似検索も重要になっている．例えば，過去の異常パターンとの類似性に基づいた異常検知 [7] や，SNS のコンテンツが似たユーザを見つけて類似ユーザの挙動からアイテムを推薦するユーザベースの情報推薦 [8] などはストリームデータを対象とした類似検索に帰着して解ける．後者の例では，ユーザ u の SNS への投稿内容を時間経過に伴ってデータが増えるストリームデータと見なし，ストリームデータを対象とした類似検索によって u と嗜好性が似た類似ユーザを見つける．最近のストリームデータを対象とした類似検索では，ストリームデータを生成されたデータの集合として表現し集合間類似検索によって類似ストリームデータを探すアプローチが主流である．通常の集合間類似検索と比べると，新たなデータの生成により集合の要素が動的に変化するため類似検索結果を更新する必要がある点が異なる．２つの集合 A, B に対する類似度 $\mathrm{sim}(A, B)$ としては Jaccard 係数がよく用いられるが，A や B が変化する度に Jaccard 係数を計算するオーバーヘッドは大きい．そこで Min-Hash [10] を用いて A と B のコンパクトなスケッチ ms_A, ms_B を生成し，スケッチ間で Jaccard 係数を近似計算する手法がいろいろ提案されている．これらの手法はいずれも集合の変化に対するスケッチ更新を効率化するが,

- データ削除を取り扱えるか
- 同種類の要素を複数持つ多重集合に対して拡張 Jaccard 係数の近似値を計算できるか

という 2 点で機能的に異なる．データ削除に関してはストリームデータ内では新しいデータほど最新の状況を反映して価値が高いところから, 古いデータを軽視するモデルが 2 種類存在する．

1. スライディングウィンドウモデル：データストリームの直近 w 個要素をスライディングウィンドウと定義し，時刻が進むとウィンドウに到着データを追加し，ウィンドウ内の最古データを廃棄する．
2. 減衰モデル：データストリームを要素に重みが付与された重み付き集合として扱い，時間経過に伴って古いデータの重みを減衰する．

MaxLogHash [4] は Min-Hash を省メモリ化する b-bit Minhash[5] をデータ追加時に更新可能にしたがデータ削除を取り扱えない．Datar ら [1] はスライディングウィンドウモデルでデータ削除に対してもハッシュ値を更新できるアルゴリズムを考案した．さらにスケッチ更新のために保持しないといけないスライディングウィンドウ内の要素数が $O(\log W)$ となることを証明した．W はスライディングウィンドウのサイズである．しかし，Datar らの手法は多重集合を取り扱えない．Bury らは [9] はデータ削除が任意の順序で発生してもハッシュ値を更新できる手法を構築したが，やはり多重集合は取り扱えない．Histosketch[3] は多重集合に関してデータ削除を取り扱える唯一の手法であるが，減衰モデルを想定している．したがって，スライディングウィンドウモデルで多重集合を取り扱える手法は存在しない．そこで本研究では，スライディングウィンドウに対して多重集合を取り扱える手法を実現することを目的とし，Datar らの手法を多重集合を取り扱えるように拡張する．多重集合の場合，スライディングウィンドウ内の同種類の要素数に依って，同一要素に割り当てられるハッシュ値が変化するという難題があるが，本研究では Datar らの手法をハッシュ値の変化に対処できるように修正した．以下，本論文の構成を述べる．2 章で提案手法の要素技術となる多重集合，集合間類似度，Min-Hash の概念を説明する．3 章では従来手法となる Datar らの動的に変化する集合を対象としたハッシュ値更アルゴリズムを紹介する．4 章で提案手法となる時間と共に変化する多重集合に対して Min-Hash のハッシュ値を更新アルゴリズムを提案する．5 章で 4 章のアルゴリズムを複数個の要素が出入りするウインドウに対応するように拡張したアルゴリズムを提案する．6 章で提案手法を人工データ，実データを用いて評価する．提案手法がハッシュ値を完全に再計算するベースライン手法よりも Min-Hash スケッチを高速に更新できることを示す．7 章でまとめと今後の課題を述べる．

第 2 章

準備

2.1　多重集合

一般にオブジェクトの集まりを集合と呼ぶ．例えば {a, b, c} はアルファベットを要素とする集合である．一般的には要素群を表すアルファベット ϕ に対して，その要素を 0 個以上含むものが集合となる．通常，集合は同じ種類の要素を 1 つしか含まない．集合を同じ種類の要素を複数持てるようにしたものを多重集合という．つまり，{a, a, b, c, d, d, d} というような集合である．

2.2　Jaccard 係数

集合間で類似検索を行うには集合がどれだけ似ているのかを表す集合間類似度が定義されている必要がある．そのうちの 1 つとして，Jaccard 係数がある．Jaccard 係数は，ある集合 A と別の集合 B について，以下の式 2.1 で定義される．

$$\mathrm{sim}(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{2.1}$$

つまり，Jaccard 係数は 2 つの集合に含まれている要素のうち共通要素が占める割合を表している．例えば，$A = \{a, c, d, f, g\}$,$B = \{a, b, c, e, g, h, i\}$ というような 2 つの集合が存在するとすると，$A \cup B = \{a, b, c, d, e, f, g, h, i\}$,$A \cap B = \{a, c, g\}$ となり，類似度 $sim(A, B) = 0.375$ となる．

そして，Jaccard 係数を多重集合に拡張したものを拡張 Jaccard 係数と言う (式 2.2).

$$\mathrm{sim}(A, B) = \frac{\sum \min\{a_i, b_i\}}{\sum \max\{a_i, b_i\}} \tag{2.2}$$

ここで，a_i は集合 A が要素 $i \in \phi$ を含む個数である．

2.3 Min-hash

Min-hash とは，集合に対する確率的なハッシュ関数であり，Jaccard 係数を用いた集合間類似検索を高速化するための技術である [10]．クエリとデータベースかの集合間で Jaccard 係数を計算することは Jaccard 係数計算のオーバーヘッドが大きい．その問題を改善し，Jaccard 係数を高速に近似計算する方法として，Min-hash という計算方法が考えられた．Min-hash は計算されたハッシュ値が一致する確率は Jaccard 係数と一致するという性質を持ち, 類似集合ほどハッシュ値が一致しやすいという良い性質を持つ (式 2.3).

$$\Pr[\mathrm{h(A)}=\mathrm{h(B)}]=\mathrm{sim(A,B)} \tag{2.3}$$

h はハッシュ値であり，A,B は集合である．

次に，Min-hash によるハッシュ値の計算方法を紹介する．$\phi = \{x_1, x_2, \cdots, x_{|\phi|}\}$ をアルファベット集合とする．ハッシュ関数は ϕ の各アルファベットに対して，$\{1, 2, \cdots |\phi|\}$ の中から被らないようにランダムな値を割り当てることで決定される (図 2.1)．１つのある集合の中のアルファベットを見て，その中の要素に対応する割り当て値の中から最小値を選ぶ．この最小値が Min-hash によるハッシュ値となる (式 2.4).

$$h(A) = \min_{e \in \mathrm{A}} \pi(e). \tag{2.4}$$

さらに，Min-hash を用いた Jaccard 係数係数の近似計算を紹介する．多数の要素を含む巨大な集合 A のコンパクトな要約として Min-Hash のハッシュ値を k 個並べた $\{mh_1(A), mh_2(A), \cdots, mh_k(A)\}$ を使用するアプローチである．この Min-Hash のハッシュ値の集合を A のスケッチ [10] と呼ぶ．２つの集合 A と B のスケッチ間で何個ハッシュ値が一致するかで A と B のハッシュ値を近似計算する．スケッチによって割り当てられる値はランダムに違うために，ハッシュ値もスケッチによって変わる．ハッシュ値が何個一致するかの確率は Jaccard 係数と一致するが，あくまで確率であるので，Min-Hash によるハッシュ値は近似値である．

2.4 多重集合に対する Min-hash

多重集合に対して Min-hash のハッシュ値を計算する手法を紹介する．文献では，大きく２つの方法が知られている．

(1) 多重集合内の複数個のラベルに異なる値を割り当てる方法

(2)Consistent Weighted Sampling (CWS) [11]

CWS は，Min-hash の出力の確率分布を模擬して，サンプリングすることでハッシュ値を計算する手法である．明示的に割り当て値を準備する必要がなく，多重集合が整数でなくても適用可能であるという利点がある．一方で確率の理論に精通していないと拡張が難しい手法でもある．(1) の同じラベルのアルファベットに異なる値を割り当てるやり方は多重集合の重みが整数であるという制約を受けるが，拡張は比較的容易な手法である．本論文では (1) の多重集合内の同一ラベルの要素に異なる値を割り当てる手法を使用する．

(1) のハッシュ値計算手法を例を用いて説明する．まず，図 2.1 のように，アルファベット {a,b,c,d} に多重度が 2 である時の数値をランダムに割り当てる．

	a	b	c	d
1個目	3	4	7	6
2個目	5	1	8	2

図 2.1　多重集合のハッシュ値表

多重集合 $A = \{a, b, b, c, c, d, d\}$ とする．多重集合 A に対して，図 2.1 の割り当て表を用いて数値を割り当てる．そして，その中の最小値が Min-Hash によるハッシュ値となる (図 2.2).

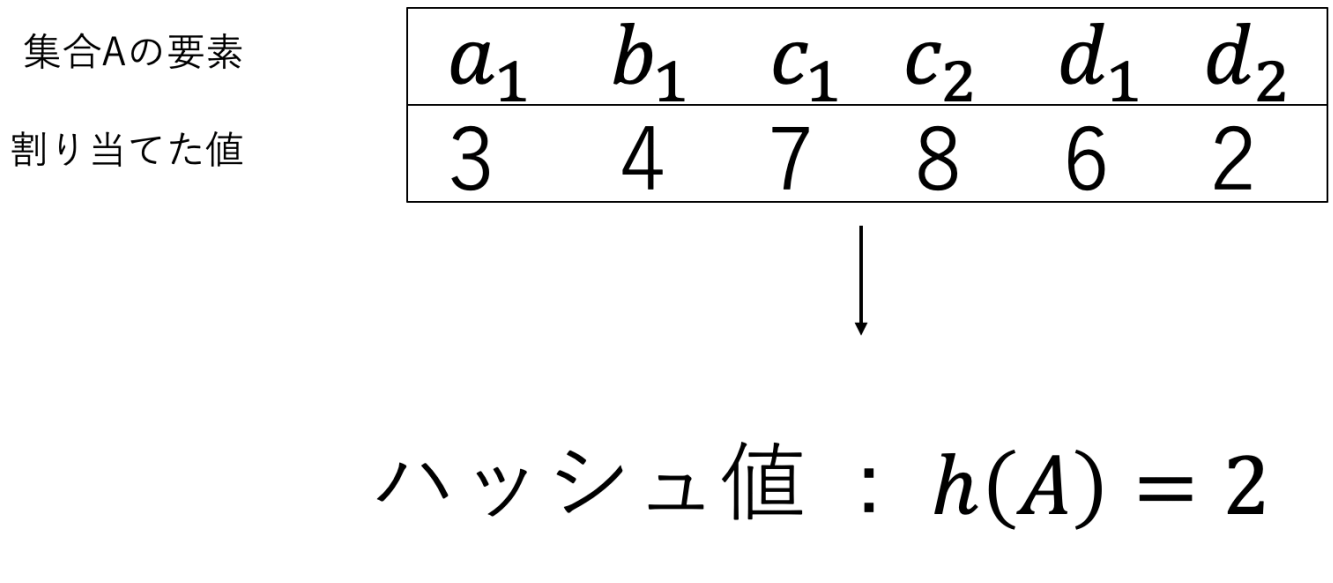

図 2.2　多重集合のハッシュ値計算

第 3 章

Datar らのアルゴリズム

本章では，Datar ら [1] によって提案されたデータストリームに対するハッシュ値更新アルゴリズムを記述する．本論文での提案手法は，このアルゴリズムを基に多重集合へ拡張したものである．データストリームとは，毎時刻要素 e が到着するデータである．Datar らの手法はデータストリームのスライディングウィンドウモデルに従って集合が変化することを仮定している．

3.1 スライディングウインドウモデル

時刻 t における集合を $A_t = \{e_{t-w+1}, e_{t-w+2}, \cdots, e_t\}$ とする．w はスライディングウインドウの幅であり，A_t は w 個の要素で構成される．また，A_t の要素は到着時間順に並んでいる．図 3.1 では，$w = 4$ の時に 3 つの時刻 t,$t+1$,$t+2$ に対する集合 A_t, A_{t+1}, A_{t+2} を図示している．そして，時刻によってスライディングウインドウは変化していく．例えば，図 3.1 のようにデータストリーム $\{a, f, h, e, k, q, o, g\}$ となっていて，ウインドウサイズ $w = 4$ とすると，時刻 t では，$A_t = \{e_{t-3}, e_{t-2}, e_{t-1}, e_t\}$ であり，時刻 $t+1$ では，$A_{t+1} = \{e_{t-2}, e_{t-1}, e_t, e_{t+1}\}$ となる．

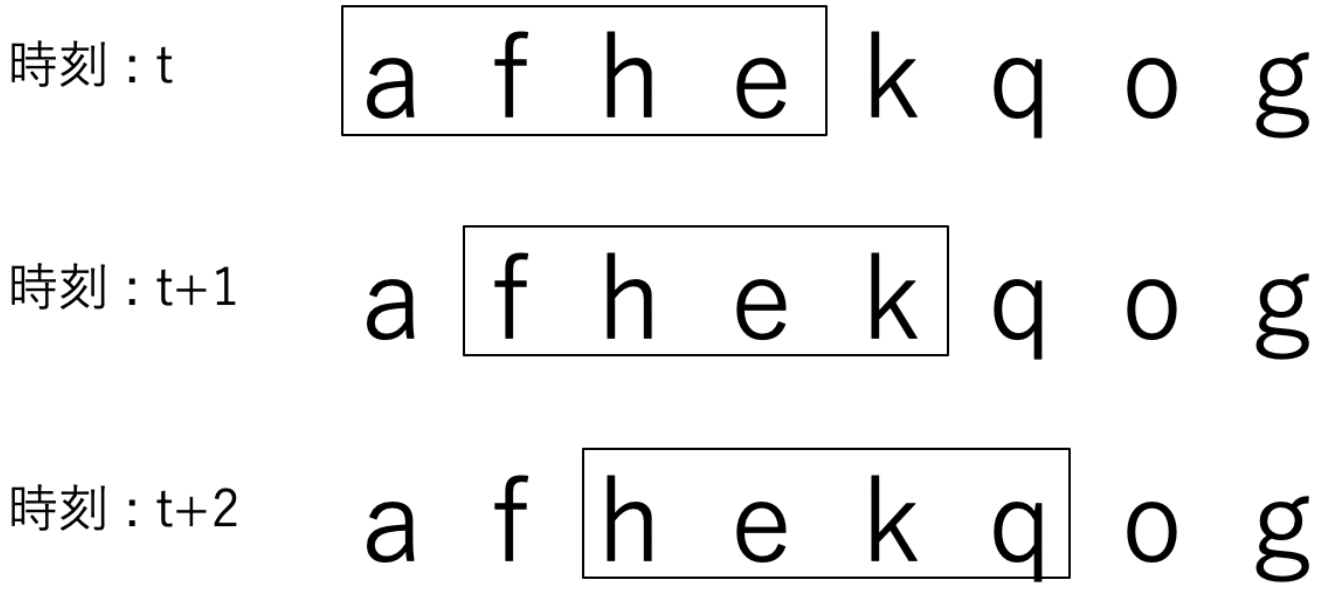

図 3.1　スライディングウインドウ

動的に変化する集合に対してハッシュ値を計算する自明な手法は，時刻経過によりウインドウがスライドするたびにハッシュ値を完全に再計算するやり方である．これはすなわち，時刻が t から $t+1$ に変化した際に，$h(A_{t+1})$ を A_t とは無関係に計算するということである．この手法では，毎時刻ウィンドウ内の全要素をスキャンする必要があるので，$h(A_{t+1})$ を求めるための時間計算量は $O(w)$ となる．

しかし、A_t と A_{t+1} は e_t と e_{t+w} 以外の $w-1$ 個の要素が共通であり，$h(A_t)$ 計算時に判明した情報を再利用することで $h(A_{t+1})$ を計算するオーバーヘッドを減らせる可能性がある．

3.2　Datar らの手法

Datar[1] らの手法の基本アイデアは

- 将来に割り当て値が最小になる可能性が絶対にない要素をあらかじめ削除することで，次の時刻のハッシュ値 $h(A_{t+1})$ を計算する時に参照する要素数を減らす

というものである．ここで割り当て値が最小になる可能性が絶対にない要素とは，「ウィンドウ内で自分より後方に割り当て値が小さい要素が存在する要素」のことである．例えば，A_t 内の 2 要素 e_i と e_j $(i<j)$ の割り当て値が $\pi(e_i)>\pi(e_j)$ を満たすとしよう．この時，e_i は自分より後方に割り当て値が小さい e_j が存在するという条件を満足する．この条件の下では e_i がウィンドウ内に存在する期間は e_j も必ずスライディングウィンドウ内に存在するので，e_i の割り当て値が最小になることはありえない (図 3.2)．Datar の手法ではこの性質を利用して，スライディングウインドウから最小値になりえない要素を削除し，将来最小値になりうる要素のみを Minlist というリストで管理する．A_t のハッシュ値は Minlist に残った要素の割り当て値の最小値となる．任意の Minlist 内の要素 e に対して，e より後方に割

り当て値 $\pi(e)$ より小さい要素は存在しないので，Minlist 内の要素群に対する割り当て値は単調増加となる．このため，$h(A_t)$ は実は Minlist の先頭要素の割り当て値と等しい．

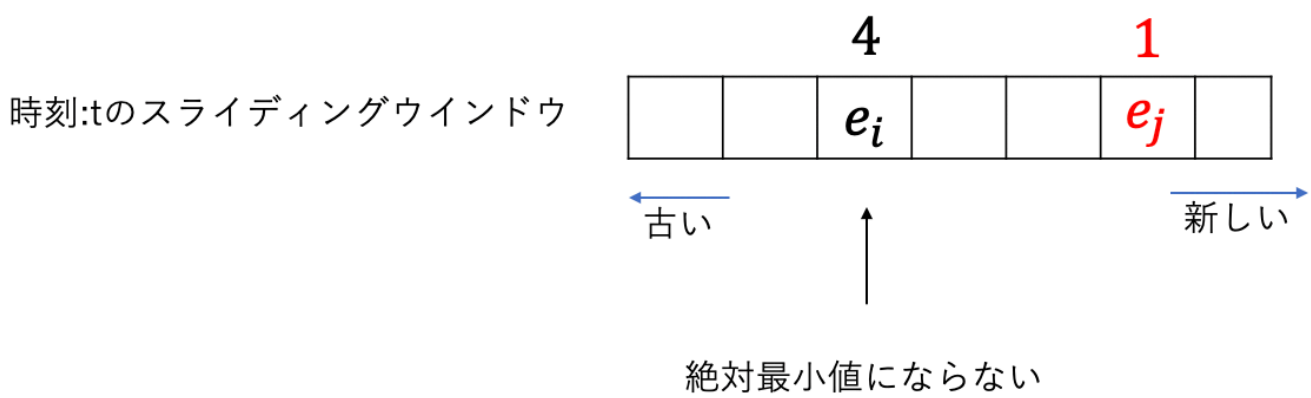

図 3.2　スライディングウインドウモデルの性質

時刻が t から $t+1$ に変化した時に Minlist を更新する手順を以下にまとめる．

- e_{t-w+1} がウィンドウから離脱した時の処理
  e_{t-w+1} が Minlist に含まれる時には e_{t-w+1} を Minlist から削除する．この時，$h(A_t) = \pi(e_{t-w+1})$ であるため，ハッシュ値を更新する必要がある．そのため，$h(A_{t+1})$ を新たに Minlist の先頭になった要素の割り当て値とする．
- e_{t+1} をウィンドウに追加した時の処理
  Minlist 内で割り当て値が $\pi(e_{t+1})$ より大きい要素は，今後，割り当て値が最小になりえないので削除する．e_{t+1} により Minlist 内の他の要素がすべて削除された結果，Minlist に e_{t+1} のみ残った場合は $h(A_{t+1}) = \pi(e_{t+1})$ とする．

このアルゴリズムの動きを図 3.3 に例示する．

時刻 t+1 の時，スライディングウインドウから一番前の要素 e_{t-w+1} を削除し，新しい要素 $e_t + 1$ のアルファベット k が入ってくる．k の割り当て値は 3 であるので，Minlist の中から 3 より大きい割り当て値を持つ要素を削除し，候補リストの一番後ろに k の割り当て値 3 を加える．

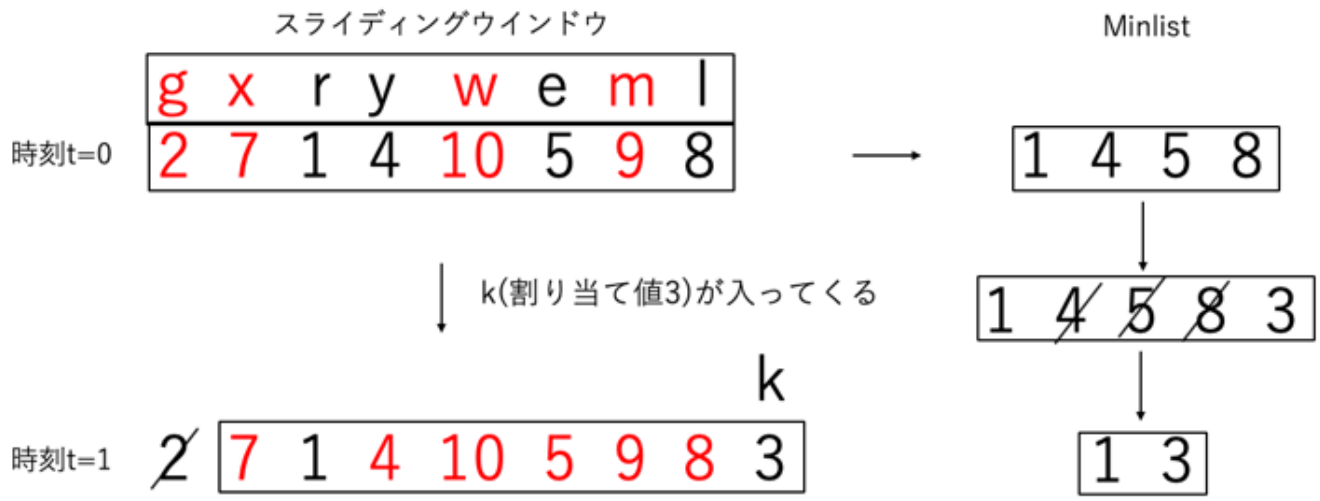

図 3.3　候補リストの作成

このように Datar らは作成した最小値の候補リストを持ちいることにより，スライディングウインドウを保持する必要がない．そのため，ウインドウサイズ W の場合, 候補リストは $\log(W)$ で保持することができ，二部探索で候補リストを探索し，実行時間は $\log\log(W)$ となることが保証される．

第 4 章

多重集合に対するハッシュ値更新アルゴリズム

本論文では，Datar らのアルゴリズムを動的に変化する多重集合に対して拡張したハッシュ値更新アルゴリズム SWMH (Sliding-Window Min-Hash) を提案する．Datar ら [1] のアルゴリズムを多重集合に拡張する場合，スライディングウインドウ内の要素 e への割り当て値が e と同じラベルを持つウインドウ内の要素数によって変化する点が難しくなる．つまり，e への割り当て値が時間経過によって変化することが起きる．スライディングウインドウ内の要素 e のラベルをアルファベット $l(e)$ とする．割り当て値 $\pi(e)$ はスライディングウインドウ内に $l(e)$ が何個あるかによって変わる．例えば，図 4.1 の場合だと，時刻 $t = 0$ の時は，スライディングウインドウ内にアルファベット b は 2 つあり，$\pi(e_i) = 4, \pi(e_j)$=1 となる．時刻 $t = 3$ でスライディングウインドウから要素 e_i が抜けた場合，e_j への割り当て値 $\pi(e_j)$ は 1 から 4 に変化する．

	a	b	c	d
1個目	3	4	7	6
2個目	5	1	8	2

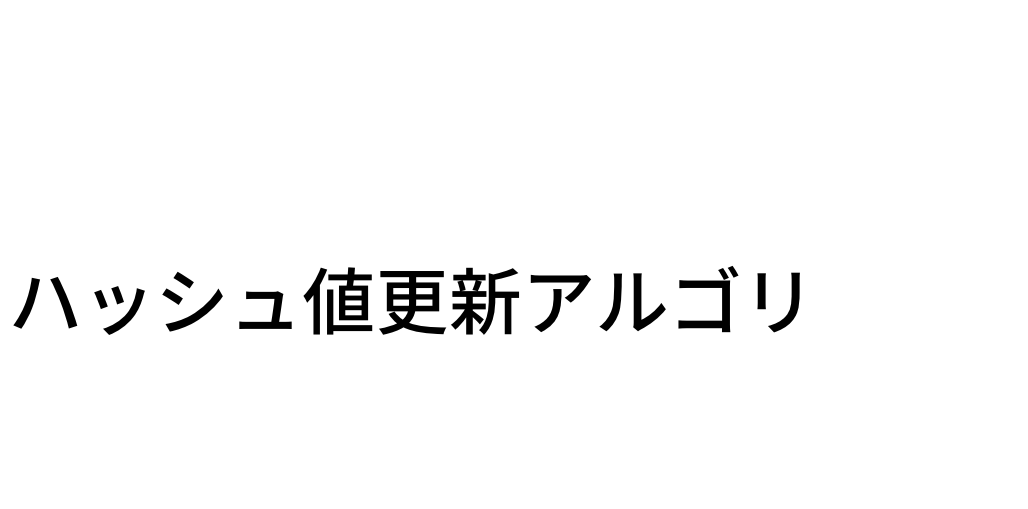

図 4.1　π (e) の変化

Datar らのアルゴリズムでは最小値になり得ない要素を Minlist から削除する．多重集合の場合，割り当て値が変化することを考慮して最小値になり得るかどうかを判定をする必要

がある．提案手法を基盤とする要素技術は以下の 2 つであり，それぞれを順に説明する．

1. 同種アルファベットへの値の割り当て
2. スライディングウィンドウ更新時の処理

4.1 同種アルファベットへの値の割り当て

多重集合に対する Min-Hash では，多重集合内に同種アルファベットが複数存在する場合それらを区別して値を割り当てる．例えば，ラベルが a のアルファベットが n 個存在する場合，それらを $\{a_1, a_2, \cdots, a_n\}$ のように区別する．a_i $(1 \leq i \leq)$ は i 番目の a という意味であり，各 a_i $(1 \leq i \leq n)$ に異なる値 $\pi(a_i)$ を割り当てる．

ここで $\pi(a_i) \leq \pi(a_{i+1})$ ならば，a_{i+1} の割り当て値は最小には絶対ならない．その理由は $i+1$ 番目の a_{i+1} が存在する条件下では，i 番目の a_i も必ず存在するからである．このように割り当て値 $\pi(a_{i+1})$ は Min Hash のハッシュ値に影響を与えないので

$$\pi(a_i) \leq \pi(a_{i+1})$$

を条件を満足する限りは別の値に変更しても構わない．そこで，$\pi(a_i) \leq \pi(a_{i+1})$ が成立する場合は $\pi(a_{i+1})$ を $\pi(a_i)$ に修正する．修正前と修正後を区別するため，修正前の割り当て値を $\pi(a_{i+1})$ とし，修正後の割り当て値を $\pi'(a_{i+1})$ とする．

次にスライディングウィンドウ内の n 個の a のインスタンスをそれぞれ何番目の a とするかを考える．つまり，n 個の a のインスタンスのインデックスをどう定めるかという問題である．通常の多重集合であれば要素間に時間順序がないのでこの問題は重要でない．しかし、スライディングウィンドウの場合，要素間に時間順序があるためインデックスの決め方によって挙動が変わる．自然な方式としては以下の 2 つが考えられる．

- 一番古い要素を a_1 とし，a のインデックスを到着時刻の昇順とする
- 一番新しい要素を a_1 とし，a のインデックスを到着時刻の降順とする

提案手法では,「一番古い要素を a_1 とし，a のインデックスを到着時刻の昇順とする」という方式を採用する．この時,

1. a_i は a_{i+1} より先にデータストリームに到着した
2. $\pi(a_i) \geq \pi(a_{i+1})$

が成立するため，a_i は a_{i+1} の存在によって Minlist から削除される．このことが任意の i に対して成立するのので，以下の lemma1 が保証される．

Lemma 1. *Minlist* の中に同種アルファベットは最新の *1* 要素しか存在しない．

Lemma 2.

$$\min\{\pi(a_1), \pi(a_2), \cdots, \pi(a_n)\} = \min\{\pi'(a_1), \pi'(a_2), \cdots, \pi'(a_n)\}$$

また，lemma 2 が成立するため割り当て値の修正によってハッシュ値は不変である．実際には多重度の上限値 n をパラメータとして π' を保持する表を事前計算し，スライディングウィンドウに到着した要素の割り当て値は表を参照して決定する．割り当て表の修正例を図 4.2 に示す．この例では多重度の上限が $n = 2$ であり，2 個目の割り当て値が 1 個目より大きい場合，割り当て値を 1 個目と同じ値に書き換えている．また，ラベルが c の要素 e_i, e_j の割り当て値の最小値は修正後も修正前から不変になっている．動的多重集合の場合は，書き換えによって，Minlist の同じアルファベットの前側の値を消せて，MInlist のサイズを小さくすることができる．

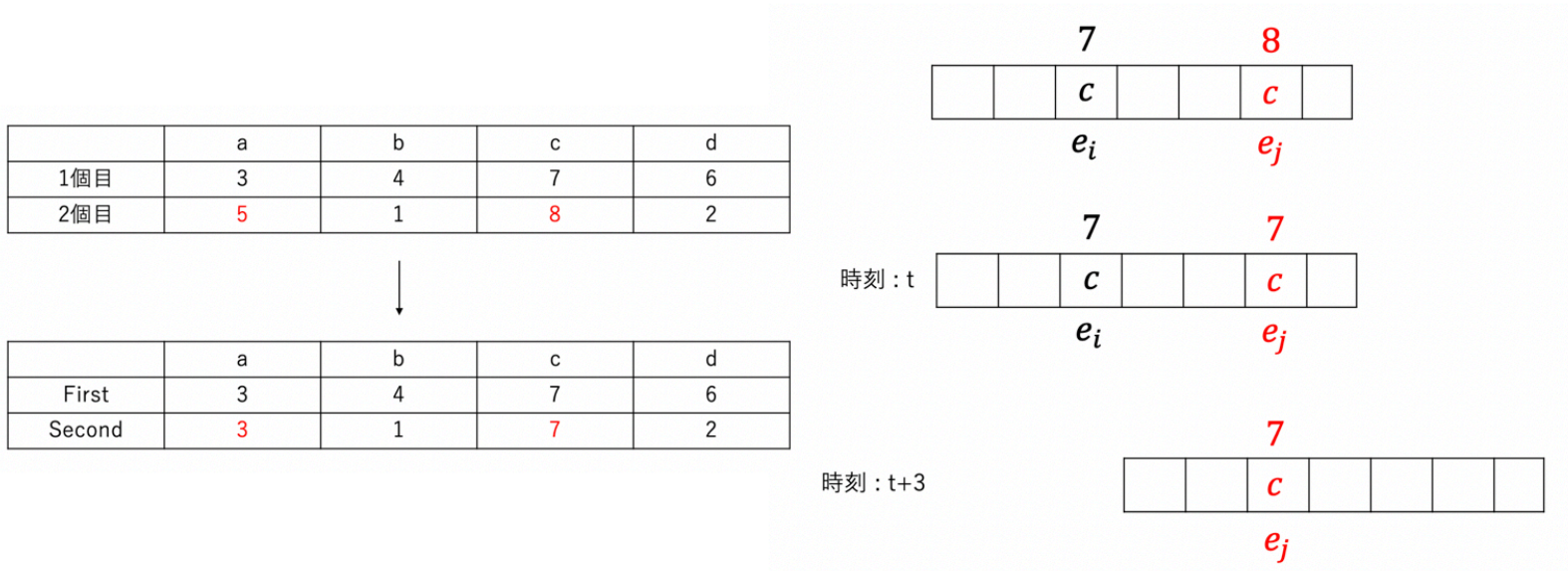

図 4.2　割り当て値の修正

以降では記述を単純化のため，修正後の値割り当て π' を単に π と記載する．

4.2　ヒストグラムの作成

スライディングウインドウが多重集合である場合，スライディングウインドウ内にいくつ同一要素が含まれているかわからないと Min-hash のハッシュ値を計算できない．そこで要素のヒストグラムを保持する．さらに各要素の到着時刻も保持するためヒストグラムのビンを到着時刻のリストとして管理し，リストのサイズにより各アルファベットのスライディングウインドウ内の個数を取得する (図 4.3)．リストは到着時刻が昇順で保持され，新たにウィンドウに到着した要素の到着時刻がエンキューされ，ウインドウから出ていく要素の到着時刻がデキューされる．

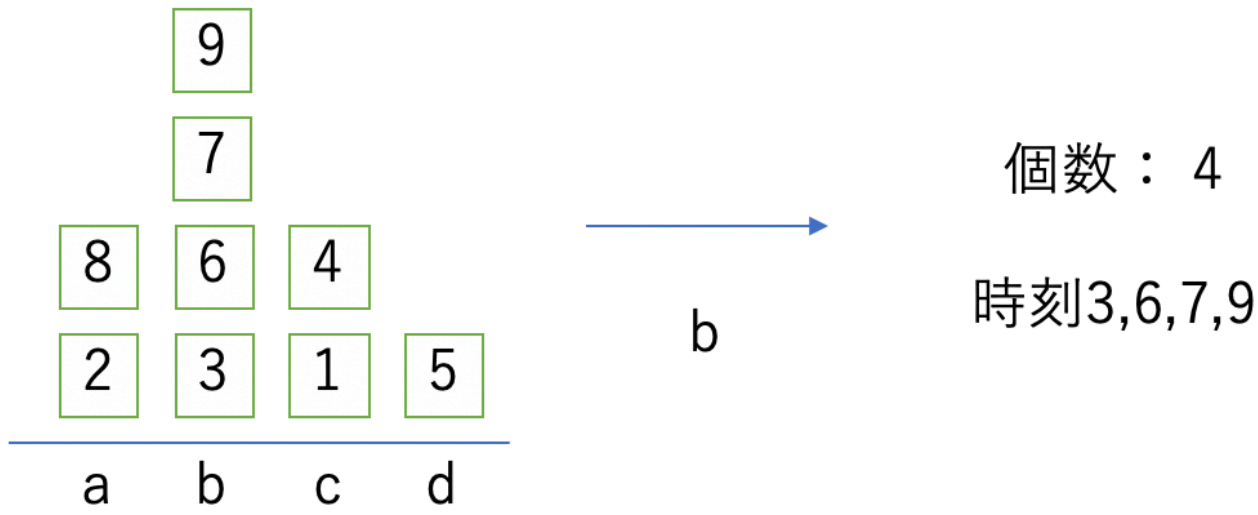

図 4.3　ヒストグラム

4.2.1　スライディングウインドウの更新

提案手法 SWMH でも Datar らの手法と同様に最小値になりうる要素のリスト Minlist を管理する．Datar らのアルゴリズムでは Minlist に割り当て値のみを保持していた．しかし多重集合の場合，割り当て値が変化が発生すると現在の割り当て値から変化後の値を計算できない．変化後の値を計算するため，Minlist は

- 現在の割り当て値：$\pi(e)$
- 要素のラベル：$l(e)$
- 要素の到着時刻：$t(e)$

の 3 つ組を要素とするリストとする．さらに SWMH では多重集合 A_t が A_{t+1} に変化する時，(1) ウィンドウから一番古い要素 e_{t-w+1} が離脱する時の処理と (2) ウインドウに新要素 e_t に入ってくる時の処理を拡張しなくてはいけない．

4.2.2　要素 e_{t-w+1} がウィンドウから出ていく処理

時刻 t に割り当て値が最小値でハッシュ値と対応する要素を α とし，そのラベルを $l(\alpha)$ とする．(Case 1): Minlist の先頭要素の時刻が $t-w+1$ ならば Minlist の先頭要素は e_{t-w+1} そのものなのでデキューする．この時，$l(e_{t-w+1}) = l(\alpha)$ であったとする．$lemmma1$ より，Minlist 内に同じラベルを持つ要素は複数存在しないので e_{t-w+1} は α と一致する．つまり，ハッシュ値と対応する要素が離脱するため，ハッシュ値の更新が必要となる．α がウィンドウから離脱することになる．そこで，新たに最小割り当て値となる要素を Minlist から探索して更新する．Minlist は割り当て値順にソートされていないので，この操作は Minlist の全要素のスキャンを伴う．

(Case 2): Minlist の先頭要素の時刻が $t-w+1$ でない場合は e_{t-w+1} は Minlist のメンバーではないため Minlist からのデキューは不要である．しかしこの場合も Minlist 内で最小割り当て値が変化する場合がある．具体的には離脱要素 e_{t-w+1} のアルファベットが $l(\alpha)$ である場合，$l(\alpha)$ の多重度が 1 減るため α の割り当て値が増加し，α の割り当て値がウィンドウ内で最小でなくなる可能性がある．この場合も Minlist をスキャンし新たな最小割り当て値を探索する．Case(2) で e_{t-w+1} のアルファベットが $l(\alpha)$ でない場合，最小割り当て値は不変である．しかし，Minlist 内にラベルが $l(e_{t-w+1})$ の要素 β が存在した場合，$\pi(\beta)$ は同じラベルの要素数が減少しすることに伴い本来増加する．しかし，最小割り当て値には影響を与えないので Minlist 内では β の割り当て値を更新しない．この結果，Minlist 内に保持されている β の割り当て値は (一時的に) 不正確になる．不正確になった割り当て値の修正は，(Case 2) で最小割り当て値を Minlist をスキャンして更新するときに同時に行う．正しい割り当て値は β のアルファベット $l(\beta)$ の多重度をヒストグラムから得ることで算出できる．

4.2.3 要素 e_{t+1} をウィンドウに入る時の処理

e_{t+1} がスライディングウインドウに入る時に，Datar らの手法では Minlist の中で $\pi(e_{t+1})$ より割り当て値が大きい要素を消すだけで十分だった．しかし，多重集合の場合は $\pi(e_{t+1})$ が将来増加する可能性があり，単に $\pi(e_{t+1})$ より割り当て値が大きい要素を消すと割り当て値が最小となる可能性がある要素も消してしまう．そこで，$\pi(e_{t+1})$ ではなく，e_{t+1} の将来の割り当て値の上限を上回る要素だけ削除する．e_{t+1} のアルファベットを $l(e_{t+1})$ とする．また，Minlist 内の要素 γ の到着時刻を t_γ として，t_γ よりあとに到着したアルファベット $l(e_{t+1})$ の個数を n とする．n の値はヒストグラムの $l(e_{t+1})$ のビンに記録された到着時刻のリストを後方から前方にスキャンすることで求められる．この時,

$$\pi(l(e_{t+1})_n) < \pi(\gamma))$$

であれば，要素 γ を Minlist から削除してよい．最後に Minlist の一番後ろに e_{t+1} を挿入し，$\pi(e_{t+1})$ が Minlist の最小値を更新するかをチェックする．アルゴリズムの動きを Algorithm 1 に示す．

Algorithm 1 要素 e_{t+1} がウインドウに入る処理

Input: 要素 e_{t+1}, 時刻 t の Minlist
Output: 時刻 t+1 の Minlist

for $0 \leq i \leq Minlist$.size()-1 **do**
  if $l(e_i)$==$l(e_{t+1})$ **then**
    Minlist[i] から e_i を削除
  end if
  n : e_i より後ろにある $l(e_{t+1})$ の数
  if $\pi(l(e_{t+1})_n) < \pi(e_i)$ **then**
    Minlist[i] から e_i を削除
  end if
end for
Minlist に e_{t+1} を加える
if π (e_{t+1}) が最小値 **then**
  最小値を π (e_{t+1}) に更新
end if

第 5 章

バッチ SWMH

前章で説明した SWMH では，データストリームに毎時刻要素が 1 つだけ到着することを仮定したが，現実のデータストリームでは，1 度に複数個の要素が到着することも普通である．そこで，そのような場合に SWMH を拡張することを目指す，本章では，データストリームに毎時刻 c 個の要素が到着するモデルを想定する．

5.1　自明な手法

データストリームに毎時刻 c 個の要素が到着するモデルにおいての自明な手法は，4 章で説明した SWMH を C 回適応する手法である．つまり，スライディングウインドウが c 回ずれる事象を，スライディングウインドウが 1 つずれる事が c 回繰り返されると判断して，SWMH を c 回適用する，1 回 SWMH を適用する度に Minlist が 1 回適用されるので，1 時刻あたり c 回スキャンすることになる．

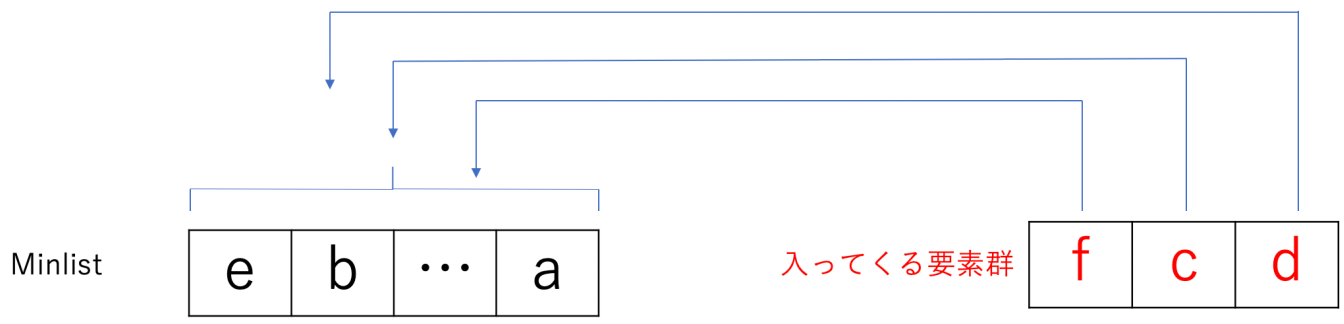

図 5.1　Minlist の更新

5.2　バッチ SWMH のアルゴリズム

本章では Minlist を c 回スキャンせず 1 回だけスキャンするバッチ SWMH を提案する．バッチ SWMH のアルゴリズムでは，ウインドウから要素が出ていく場合と入ってくる場合に処理を分けて考える．ウインドウに c 個の要素が出入りする場合，ウインド

ウから出ていく要素群を $E_{t-w/c+1}$ $=\{e^1_{t-w/c+1}, e^2_{t-w/c+1}, ..., e^c_{t-w/c+1}\}$, 入ってくる要素群を E_t $=\{e^1_t, e^2_t, ..., e^c_t\}$ とする．

5.2.1 要素群 $E_{t-w/c+1}$ が出ていく処理

バッチ SWMH の $E_{t-w/c+1}$ が出ていく処理では，4.2.2 の SWMH のウインドウから要素が出ていく時に対する処理と同じように場合分けをして処理を行う．Minlist 内で最小割り当て値となる要素 α のアルファベットを x と記述する．

(Case1):Minlist の先頭要素の時刻が $t-w/c+1$ ならば Minlist の先頭要素をデキューする．さらに，Minlist の要素が $t-w/c+1$ と違う時刻要素が出てくるまで Minlist をスキャンし，$t-w/c+1$ と同じ時刻を持つ要素を削除する．この時，削除される要素が α と同じアルファベット x を持つ場合，新たな最小割り当て値となる要素を Minlist から探索して更新する．Minlist は割り当て値順にソートされていないので，この操作は Minlist の全要素のスキャンを伴う．

(Case2):Minlist の先頭時刻が $t-w/c+1$ でない場合，Minlist からのデキューは不要である．不要である理由は，4.2.2 で説明した理由と同じである．

5.2.2 要素群 E_t が入ってくる処理

Minlist に E_t が入ってくる処理では， まず，E_t の要素すべてヒストグラムに追加する．始めにヒストグラムに追加することを利用して，以下の 3 つの手順で処理を行う．

手順 1:要素群の更新と代表アルファベットの選択

手順 1 として，E_t を後ろからスキャンし，将来最小値になり得ない要素の削除と，Minlist をスキャンする際の基準アルファベットの選択を行う．基準アルファベットとは，E_t の中で割り当て値が最小のアルファベット x と E_t の中でスライディングウインドウ内の多重度最大のアルファベット y である．c 個の要素群が入ってくることに対して，その中の 1 つのアルファベットだけで Minlist をスキャンより，2 つのアルファベットを使う方が割り当て値の変化により対応でき，Minlist を短く保つことにつなるために，基準アルファベットを 2 つ選択する．スキャンされる側の E_t 内の要素を $e^z_t(1 \leq z < c)$，E_t の一番最後尾の要素 e^c_t のラベルのアルファベットを $x = l(e^c_t)$,$y = l(e^c_t)$ として，後ろから順に以下の処理を行なっていく．

(1) e^z_t と同じアルファベットが E_t 内に複数存在するか確認

e^z_t と同じアルファベットを消すために，従来の手法では，E_t 全体をスキャンし，同じアルファベットが複数存在するか確認する必要があるため，$O(c)$ の時間を要する．しかし，私たちは先に保持させたヒストグラムの情報の 1 つである時刻を用いて，ヒストグラムの最後

尾に入っている要素の時刻と e_t^z の時刻を比較することで，同じアルファベットの中で一番最後尾の要素のみを残し，ほかの要素を削除することができる．

(2) 違うアルファベットの中で必要ない要素の削除と x,y の更新 $\pi(e_t^z)$ より $\pi(x)$ が大きい場合， e_t^z を削除し，小さい場合 x を更新する，さらに，e_t^z のアルファベットが E_t 内でヒストグラム最大の場合，y を更新する．

if ($\pi(x) \leq \pi(e_t^z)$){
　　e_t^z を E_t から削除する．
} **else** $x = l(e_t^z)$

if (histgram(y).size < histgram($l(e_t^z)$).size) $y = l(e_t^z)$

(3) 入ってきた要素群 E_t から将来最小値になり得る要素のみを残した要素群 E_t' として保持する．

手順 2:Minlist のスキャン

手順 2 では，Minlist 内で手順 1 と同じように E_t' の要素のアルファベットと同じ要素の削除と，将来最小値になり得ないアルファベットを持つ要素の削除を行なっていく．同じアルファベットの要素の削除は，手順 1 の方法により，ヒストグラムを利用して削除していく．

将来最小値になり得ないアルファベットを持つ要素の削除するために，E_t の中で，x，y の 2 つのアルファベットを用いてスキャンを行なっていく．Minlist 内の要素 λ の入ってきた時刻 t_λ とし，t_λ よりあとに到着した x， y の個数 n_x，n_y とする．時刻に応じて，x，y のうち小さい割り当て値を持つアルファベットの割り当て値を用いて，Minlist から将来の割り当て値の上限を上回る要素を削除していく．

$\pi_{\min} = \min(\pi(x_{n_x}), \pi(y_{n_y}))$
if ($\pi_{\min} < \pi(l(\lambda))$) Minlist から λ を削除

手順 3:Minlist への追加

最小値となり得ない要素を削除した Minlist に対して，E_t' を追加する．

第 6 章

実験

本章では提案手法である SWMH と，バッチ SWMH を人工データと実データを用いて実験的に評価する．6.1 節で使用したデータセットを記述する．次に，6.2 節ではデータストリームの到着レートが 1 である状況で SWMH を評価する．ここでは，時刻変化のたびにスライディングウインドウ全体をスキャンして，Min-hash のハッシュ値を再計算する手法を Baseline とし，SWMH が Baseline より高速であることを示す．6.3 節ではデータストリームの到着レートが $c > 1$ である条件でバッチ SWMH を評価する．ここでは，5.1 節で述べた SWMH を c 回適用する単純手法とバッチ SWMH の実行時間を比較する．

6.1 データセット

6.1.1 人工データセット

人工データセットは，アルファベット ϕ から，zipf 分布に従ってサンプリングを繰り返し，長さ 100,000 の文字列 st を生成する．到着レートを c として，データストリームは st の先頭から順に毎時刻 c 個取り出すことで，シミュレートする．データセットの特性を決定するパラメータは

- α：zipf 分布の偏りをコントロールするパラメータ
- ϕ :アルファベットの種類数

であり，スライディングウインドウのの長さを W とする．zipf 分布とは，α を $[0, 1]$ の範囲で指定し，アルファベットの出現頻度に偏りを持たせた分布である．アルファベットの出現順位を k,k 番目のアルファベットの個数を N_k, 一番出現頻度の高いアルファベットの個数を N_{max} とすると，zipf 分布は以下の式 6.1 で表せる．

$$N_k = N_{max} \times 1/k^{\alpha} \tag{6.1}$$

α は zipf 分布の偏りをコントロールするパラメータで α が大きいほど出現頻度の偏りが

大きい．その結果，α が大きいほどスライディングウインドウ内の要素の多重度大きくなる．$\alpha = 0$ の時は単に一様分布に従ってサンプリングがなされる．

6.1.2　実データセット

実データによる実験は，connect dataset と mushroom dataset の 2 種類のデータ [4] [6] からデータベースを作成した．まず，connect dataset とは 117 種類のラベルが出現し，約 20 の長さのラベルから成る集合が約 4,000 列連なったデータである．次に，mushroom dataset とは 127 種類のラベルが出現し，約 20 の長さのラベルから成る集合が約 60,000 列連なったデータである．どちらのデータも 1 つの集合に同じラベルは含まれないデータである．それらのデータからランダムに集合を 1 つ選ぶ処理を，選ばれた集合をつなげた長さがデータストリームの長さ $100,000+$ ウインドウサイズ W を超える長さになるまで行い，データストリーム st を生成する．人工データと同じように，到着レートを c として，データストリームは st の先頭から順に毎時刻 c 個取り出すことで，シミュレートする．

6.2　SWMH の実験評価

まず，人工データを使った評価主数を説明する．人工データのパラメータは以下の 3 つである．

- α
- ϕ
- W

$\alpha = 1$, $|\phi| = 100$,$W = 100$ という組み合わせをデフォルトパラメータとし，それぞれのパラメータを変更し，実験を行なった．

次に，実データを使った評価主数を説明する．実データのパラメータは，

- W

のみであり，$W = 100$ をデフォルトパラメータとし，実験を行なった．

それぞれの実験において，実験のばらつきを減らすために，ハッシュ関数を 10 個生成し，SWMH10 回の合計を実行時間とし，実験を行なった．

実験 1): ウインドウサイズ W を変えて実験

スライディングウインドウサイズ $W = 100, 500, 1000, 5000, 10000$ と変えて，提案手法の実行速度と最小値候補リスト Minlist の長さを計測した．人工データ, connect, mushroom いずれのデータセットにおいても，SWMH が Baseline より圧倒的に早かった (図 6.1,6.3 ,6.4)．SWMH の実行時間は，Baseline と比べて，W=100 の時，約 9 倍，W=1000 の時，

約 17 倍,W=10000 の時，約 60 倍となった．実行時間と W の関係をより詳しく調査するため人工データに対する実行時間を対数グラフで表現したものを図 6.2 に表す．Baseline では，実行時間が W に対してリニアに増加するのに対して，SWMH では実行時間が $\log(W)$ に比例して増加する．Baseline の実行時間が W に比例するのは Min-hash の計算時間が $O(W)$ であるためである．一方，SWMH の実行時間は Minlist の長さに関係があると考えられる．そこで Minlist の長さを調べた結果を図 6.5 に表す．Minlist の長さは，毎時刻ごとの Minlist の長さの 1 時刻の平均を表している．

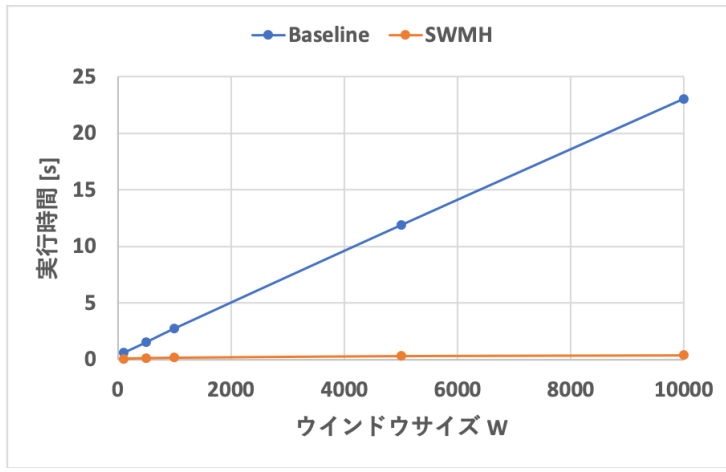

図 6.1　人工 dataset

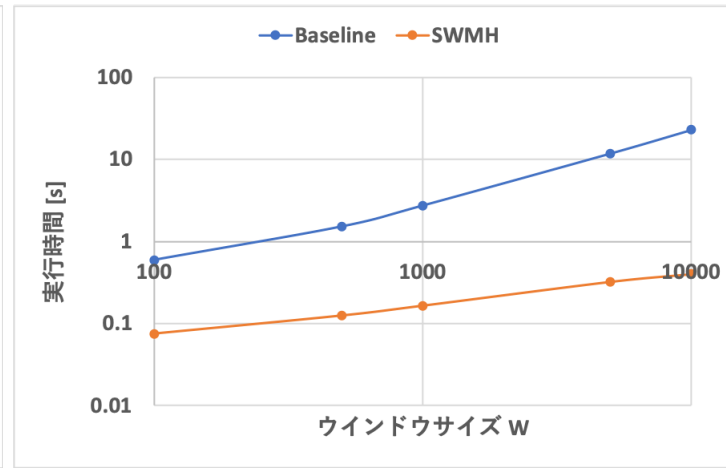

図 6.2　人工 dataset における対数グラフ

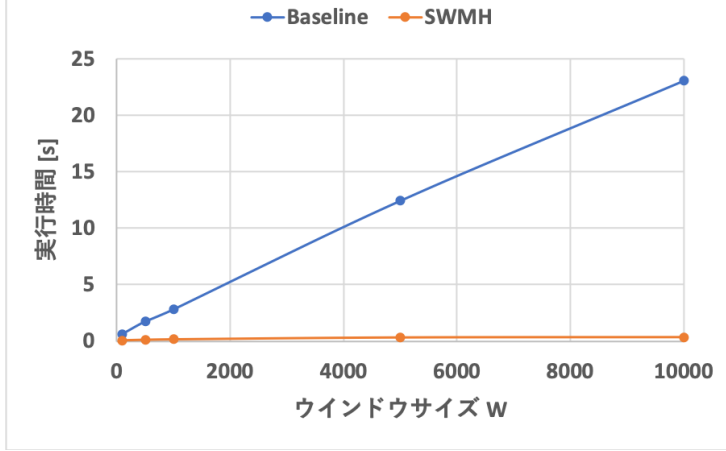

図 6.3　mushroom dataset

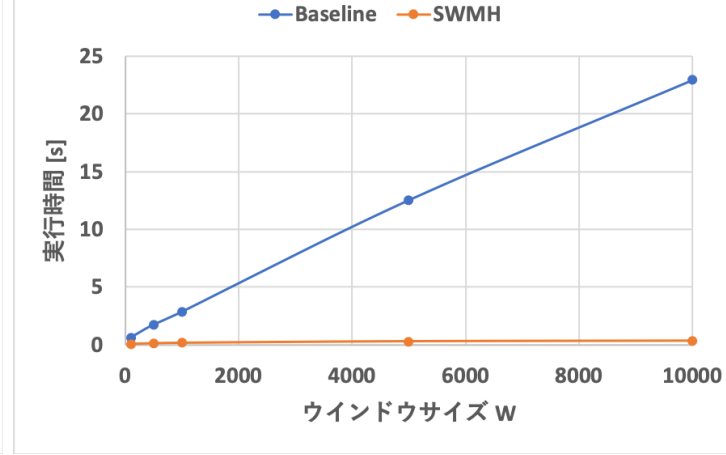

図 6.4　connect dataset

実験 2):偏りパラメータ α を変えて実験

偏り $\alpha = [0, 1]$ の幅で 0.1 ごとに変えて，SWMH の実行時間と Minlist の長さを計測した．

人工データ, connect, mushroom いずれのデータセットにおいても，SWMH が Baseline より早かった (図 6.6)．Baseline では，偏りが大きくなるにつれて実行時間が短くなることに対して，SWMH では実行時間が長くなっていく．これは，偏りが大きいほど要素の出現に偏りができ，ハッシュ値計算時の割り当て値表へのアクセスが局所的になり，キャッシュ効率が改善したため Baseline は実行時間が短くなったと考えられる．しかし，SWMH で

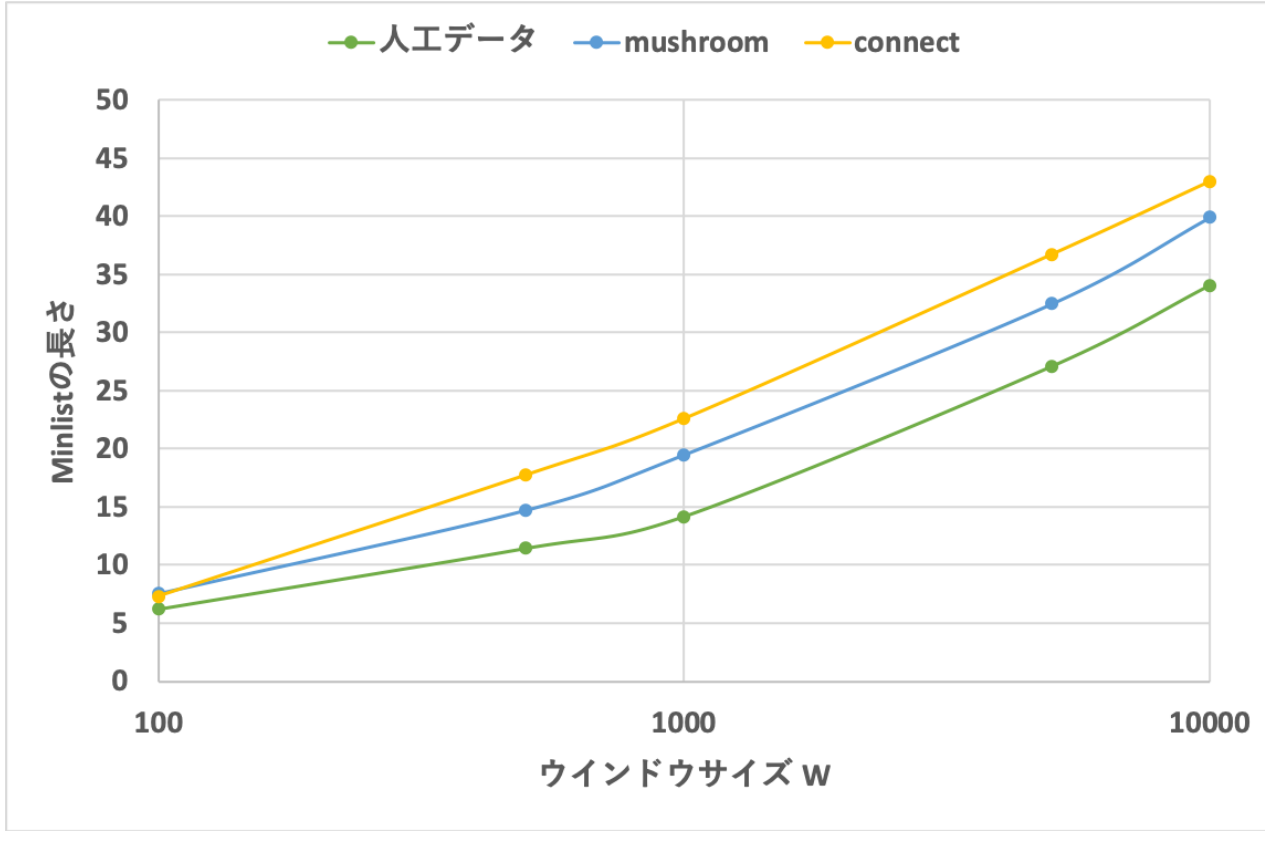

図 6.5　W に対する Minlist の長さ

は，偏りが大きくなるごとに同じ要素が多く出やすくなる影響から 4.2.3 のアルゴリズムにより Minlist スキャンの際に，ヒストグラムや割り当て表を深く参照する回数が増える一方で，Minlist の長さに大きな影響を与えないため実行時間が長くなっていると考えられる．表 6.1, 図 6.7 より，Minlist の長さは，α が増えるごと途中までは，増加するが，途中から減少する．これは，α が大きくなるにつれて，Minlist 内の要素が違うラベルによって消される減少量より，同じラベルに消される増加量が大きくなっていくことにより，起きていると考えられる．

表 6.1　傾きと実行時間，Minlist の長さとの関係

傾き	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
実行時間 [s]	0.06311	0.0631	0.06227	0.06238	0.06454	0.0668	0.6732	0.06885	0.0711	0.07347	0.07577
Minlist の長さ	6.44605	6.46051	6.47262	6.52681	6.59502	6.66685	6.75032	6.81968	6.62904	6.54369	6.29902

実験 3):ラベル種類数 $|\phi|$ の変更

要素の種類数 $|\phi| = 10, 50, 100, 500$ と変えて，提案手法の実行速度と最小値候補リスト Minlist の長さを計測した．図 6.8 より，Baseline では，$|\phi|$ が増加すると実行時間が増加する．この理由は，ラベルの種類数が多いほど，割り当て値表の広範囲にアクセスし，アクセスが局所的でなくなった結果，キャッシュ効率が限定的にないためと思われる．一方で，SWMH では，ラベルの種類数が増えるほど，Minlist 更新の際に，同じラベルを持つ要素同士の削除が減少し，実行時間が減っていくと考えられる．

実験 4): ヒストグラム参照上限固定実験

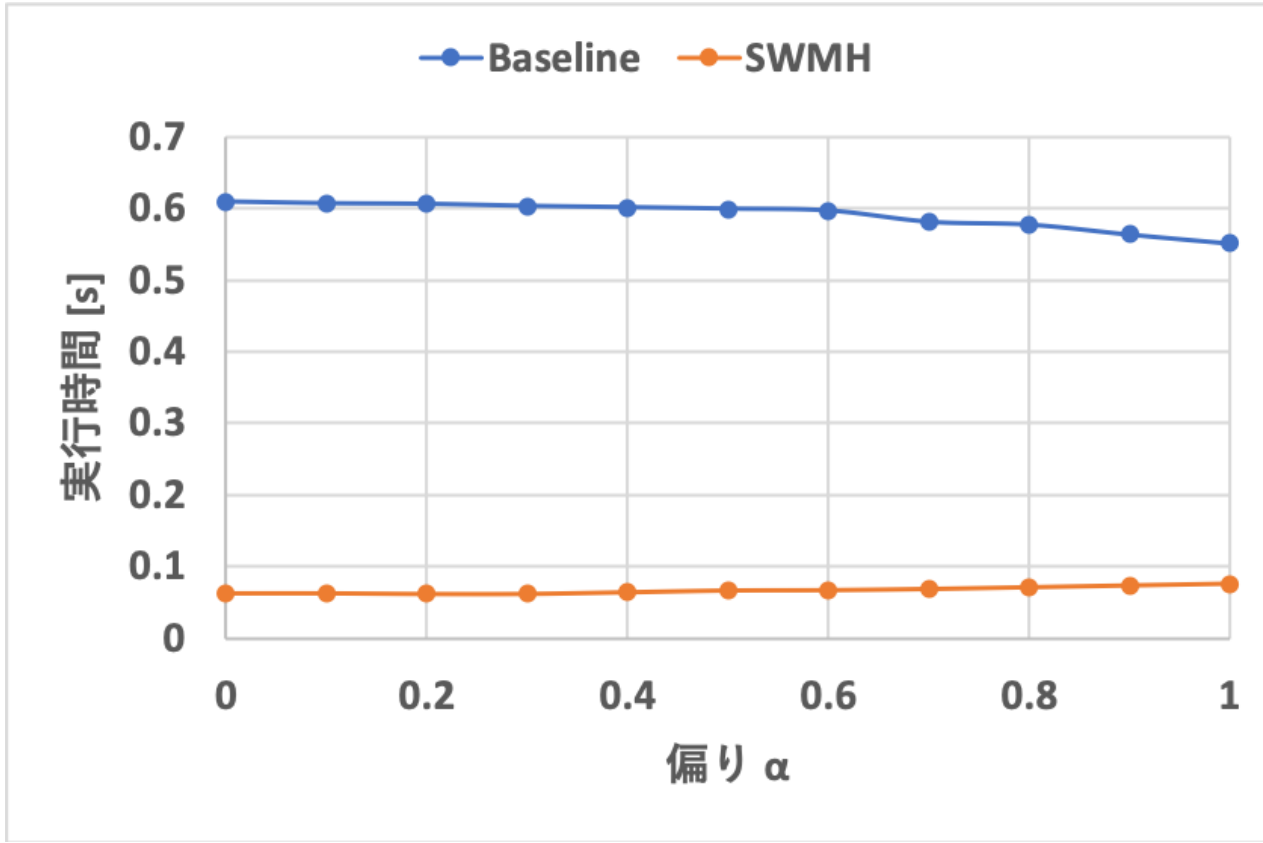

図 6.6　傾き α を変えた実行時間

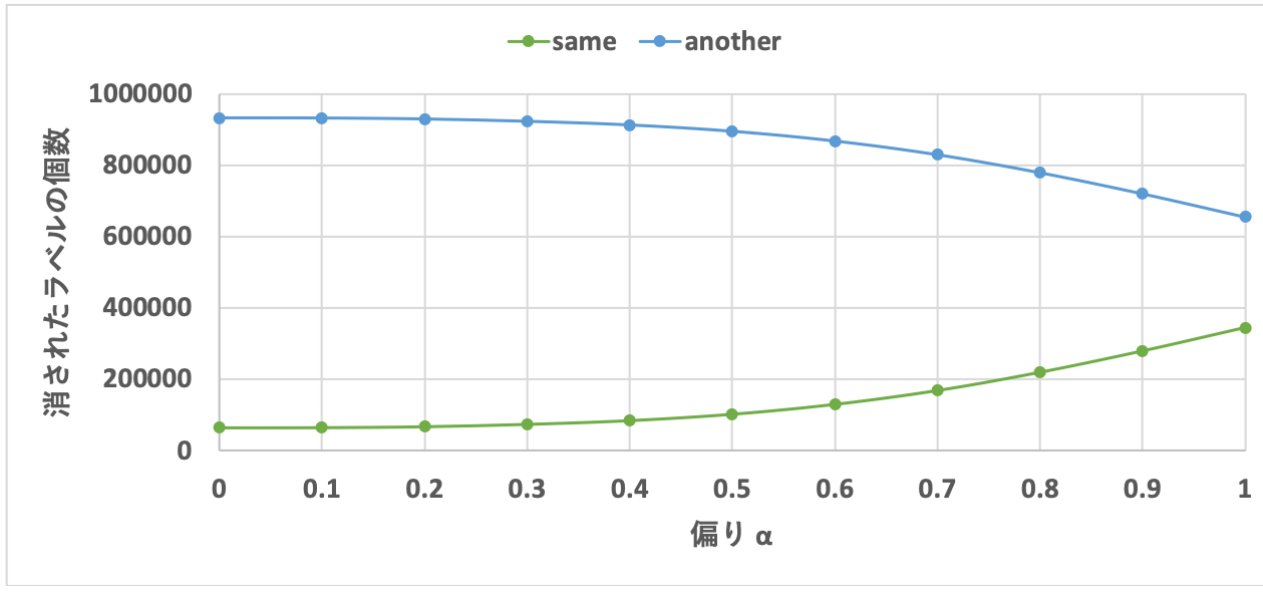

図 6.7　Minlist の消去されたラベルの種類

SWMH のアルゴリズムでは，Minlist をスキャンする際に，スキャンされる要素 e' よりウインドウ内で後ろにある $l(e_{t+1})$ の個数に応じた $l(e_{t+1})$ の割り当て値を使用し，将来必要ない要素の削除を行なっているが，ヒストグラムを参照する個数の上限を固定して Minlist をスキャンする実験を行う．例えば，上限 2 の場合，e' の後ろに $l(e_{t+1})$ が 5 個あったとしても，$\pi(l(e_{t+1})_5)$ ではなく，$\pi(l(e_{t+1})_2)$ と比べて，e' を削除するか判定する．

参照するヒストグラムの個数を固定し，ラベルの種類数 $|\phi| = 10, 30, 100$ と変えて，実行時間を計測した．図 6.9 より，$|\phi| = 10, 30$ の時は，ヒストグラム参照上限 2 で実行時間が一番短くなり，$|\phi| = 100$ の時は，上限 1 の実行時間が一番短くなった．ヒストグラムを多

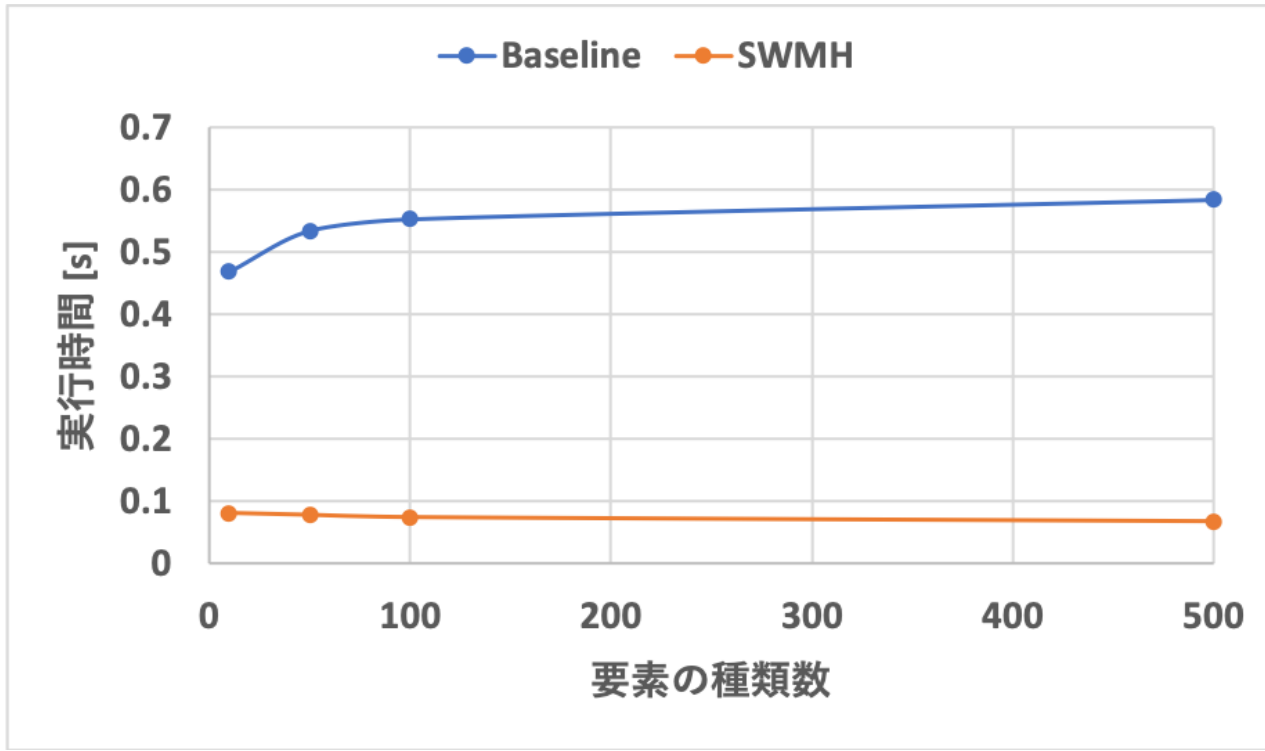

図 6.8　ラベルの種類数 $|\phi|$ を変えた実行時間

く参照しても，Minlist の長さへ与える影響は少なくなり (図 6.10)，逆に何回も割り当て表やヒストグラムを参照することに時間をかけていると考えられる．

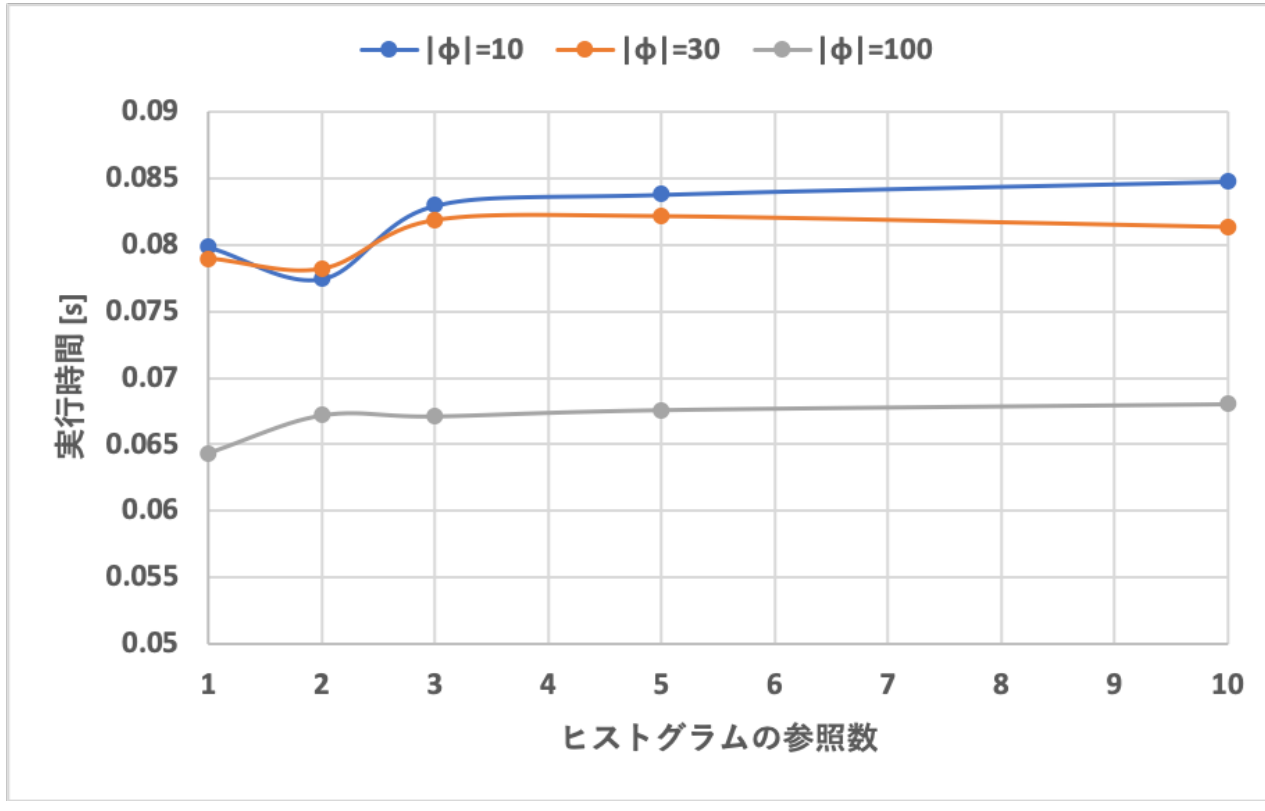

図 6.9　ヒストグラムの参照数を変えた実行時間

6.3　バッチ SWMH の実験評価

本実験では，人工データと connect，mushroom の 3 つのデータセットを使用した．まず，人工データを使った評価主数を説明する．人工データのパラメータは到着レート $c = 5$,

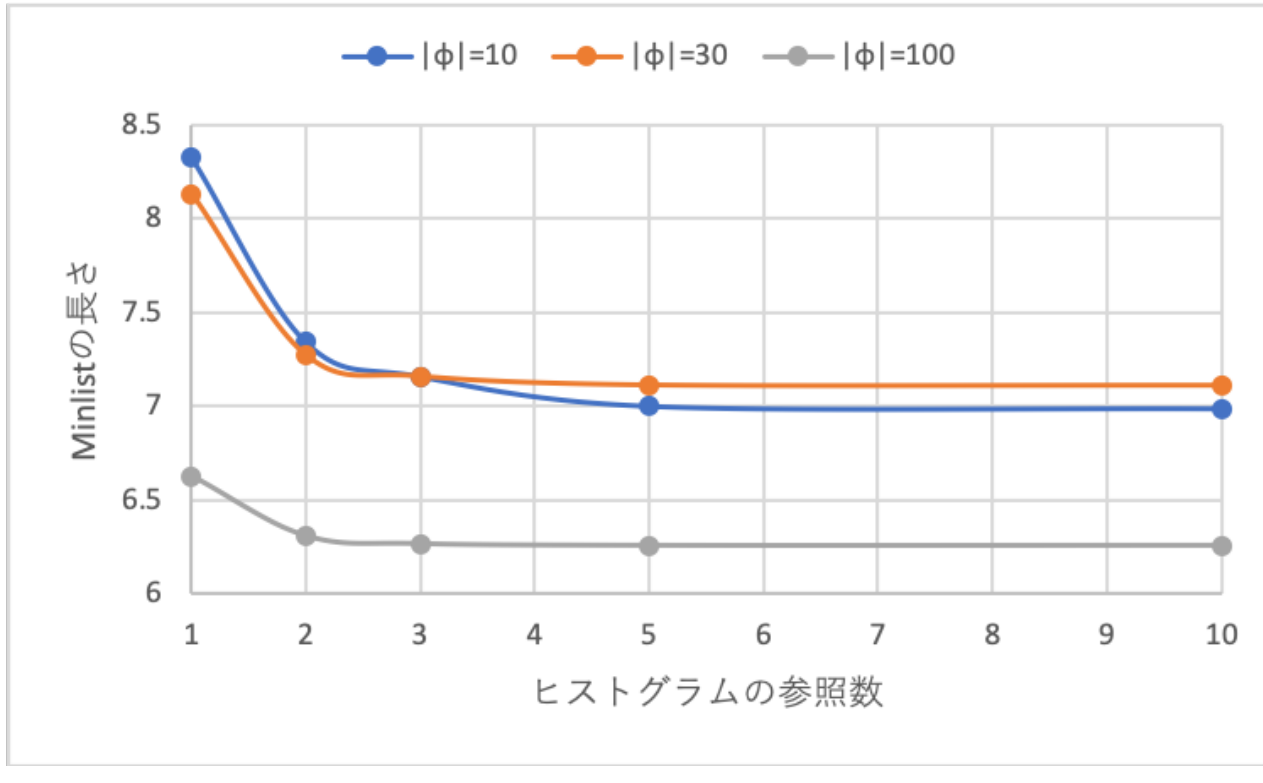

図 6.10　ヒストグラムの参照数を変えた Minlist

$\alpha = 1$, $|\phi| = 100$,$W = 100$ という組み合わせをデフォルトパラメータとし，実験を行なった．次に，実データを使った評価主数を説明する．実データのパラメータは，$c = 5$, $W = 100$ をデフォルトパラメータとし，実験を行なった．実験において，実験のばらつきを減らすためにハッシュ関数を 10 個生成し，バッチ SWMH の合計を実行時間として，実験を行なった.

人工データ, connect, mushroom いずれのデータセットにおいても，SWMH を c 回適応する方法より，バッチ SWMH の方が約 2 倍早かった (表 6.2). 図 6.3 より，Minlist の長さはバッチ SWMH の方が長くなった．SWMH では，c 回スキャンするため，c 個の要素のアルファベットを使用するが，バッチ SWMH では c 個の要素のうち 2 個の要素のアルファベットのみを使用し，削除するために Minlist が長くなっていると考えられる．しかし，入ってきた要素すべてと Minlist 内の要素を比較して候補とならない要素を消していくのではなく，入ってきた要素の中から最小値となり得る要素を厳選し，その要素を用いて削除していくことにより Minlist を更新する回数を減らし，実行時間を短縮することができたと考えられる.

表 6.2　バッチ SWMH の実行時間

データセット	人工	connect	mushroom
SWMH	0.0742	0.0743	0.073
バッチ SWMH	0.042	0.0365	0.037

表 6.3　バッチ SWMH の Minlist の長さ

データセット	人工	connect	mushroom
SWMH	6.2883	6.2082	6.296
バッチ SWMH	6.5501	7.5545	7.4214

第 7 章

まとめ

本研究ではデータストリームに対する類似検索の高速化を念頭に，データストリームに対するハッシュ値の更新アルゴリズムを取り扱った．とくにスライディングウィンドウを動的に変化する多重集合と見なして，スライディングウィンドウに対する Min-Hash のハッシュ値更新アルゴリズム SWMH を提案した．SWMH はスライディングウィンドウで集合を取り扱った Datar らの手法を多重集合に拡張したものであり，スライディングウインドウモデルで多重集合を取り扱える初の手法である．SWMH の特筆すべき点は，動的多重集合に対して Min-hash を計算する場合，１つの要素への割り当て値が多重度に影響されて動的に変化するという難点に対応したことである．さらに要素への割り当て値をハッシュ値が変化しない範囲で修正することにより，SWMH が管理する必要がある要素数を削減させて，実行時間を短縮した．さらに，SWMH を拡張し，一度に複数個の要素がスライディングウインドウに到着するモデルに対応した Min-hash 計算手法であるバッチ SWMH を提案した．バッチ SWMH では，要素 1 つ１つに対して Minlist を更新するのではなく，ウインドウに入ってきた要素群の中で，最小の割り当て値を保持する要素と最大多重度のラベルを保持する要素の 2 つに絞り，Minlist へアクセスする回数を減らすことで実行時間の短縮を狙った．人工 dataset と実データセットを用いて Min-hash 計算時間を測る実験を行い，提案手法を評価した．まず，SWMH は，毎時刻 Min-hash のハッシュ値を再計算するベースラインより，高速にハッシュ値を算出できることを示せた．さらに，バッチ SWMH は，複数個の要素がウインドウを出入りする場合に SWMH より短い時間で Min-hash を高速計算できることを示した．最後に，提案手法 SWMH では，データを保持するためにヒストグラムを多用しているため，多くのメモリを消費している．従って，今後の研究課題としては，メモリ使用量の削減のために近似ヒストグラムを用いてハッシュ値を計算する手法の実現が望まれる．

参考文献

[1] Mayur Datar and S Muthukrishnan "Estimating Rarity and Similarity over Data Stream Window" AT&T Research, Florham Park NJ, USA.

[2] X, Xu,C. Gao, J. Pei, K.Wang, and A. Al-Barakati,"Continuous similarity search for evolving queries," Knowledge and Information Systems, vol.48(3),pp.649-678, September 2016.

[3] Yang D, Li B, Rettig L, Cudre-Mauroux P. "HistoSketch : fast similarity preserving sketching of streaming histograms with concept drift", 2017 IEEE international conference on data mining (ICDM); 2017. p. 545-54.

[4] Wang P, Qi Y, Zhang Y, Zhai Q, Wang C, Lui J, Guan X. "A Memory-Efficient Sketch Method for Estimating High Similarities in Streaming Sets", 2019.

[5] Ping Li, Arnd Christian Konig, "b-Bit Minwise Hashing", 2010 Nineteenth International World Wide Web Conference (WWW 2010).

[6] Michael M, Rasmus P, Ninh, "Efficient Estimation for High Similarities using Odd Sketches" , In WWW, pages 109-118, 2014.

[7] W PM Rowe, A Paola Carrieri, C Alcon-Giner, S Caim, A Shaw, K Sim, J Kroll, L j.Hall, E O.Pyzer-Knapp, M D.Winn, "Streaming histogram sketching for rapid microbiome analytics" Rowe et al.Microbiome 2019.

[8] D. Yang, B. Li, and P. Cudre-Mauroux, "Poisketch: Semantic place labeling over user activity streams," in Proc. of IJCAI, 2016, pp. 2697-2703.

[9] M Bury, C Schwiegelshohn, M Sorella, "Efficient Similarity Search in Dynamic Data Streams", 2021.

[10] A Z Broder, M Charikar, A M Frieze, M Mitzenmacher, "Min-Wise Independent Permutations",Journal of Computer and System Sciences Volume 60, Issue 3, June 2000, Pages 630-659.

[11] M Manasse, F McSherry, K Talwar, "Consistent weighted sampling", 2010.