時間と共に変化する多重集合に対する min-hash の高速計算

1610615 三原寛寿

指導教員

古賀　久志　准教授

2020 年 1 月 31 日

目次

第 1 章

はじめに

集合は情報科学における基本データ構造であり，様々な応用分野で使用される．例えば，文書処理ではテキスト文書を単語の集合として表現して処理することが頻繁になされる．また，グラフ処理においてもグラフを部分グラフの集合として処理する技術が多く存在する．通常，これらの応用においては，集合は静的であり，その要素は不変である．一方で，近年，ストリームデータ処理の分野で要素が動的に変化する集合が取り扱われるようになって来ている．ストリームデータの特性は時間経過ともに新しいデータが到着することであり，代表例としては IoT におけるセンサからの観測データや心電図データなどがある．また，特定のユーザの 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．これらはユーザの嗜好性を表しているため情報推薦に活用される．ここで，1 データストリームを到着したデータの集りと捉えると，1 データストリームはユーザ嗜好性を表す動的に要素が追加される集合ということになる．従って，情報推薦は，動的に要素が追加される集合に対する集合を対象とした類似検索に帰着できる．さらに，1 データストリーム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以下の 2 種類存在する．

- 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．
- 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．

このような背景の下、Xu[2] らは動的に変化する集合をクエリとして，静的な集合のデータベース D から毎時刻上位 K 個の類似集合を検索する Continuous Similarity Search for Evolving Queries 問題を提唱とした．この問題設定では，クエリ集合 Q_t をスライディングウィンドウで定義し，クエリ集合の変化に対応して検索結果を更新する必要がある．ここで t は時刻であり，クエリ集合が t によって変わることを表す．この問題は，Q_t とデー

タベース D の全集合間で，類似度 (Jaccard 係数) を毎時刻計算し，上位 k 個の集合を返すことで自明に厳密に解ける．しかし，この厳密解法では 2 つの集合 A と B 間で類似度 $sim(A,B)$ を計算するオーバーヘッドが大きい．そこで，Xu らは，Min-Hash[6] を用いて A と B のコンパクトなスケッチ (Min-Hash スケッチ)ms_A, ms_B[4] を生成し，スケッチ間で Jaccard 係数を近似計算する近似解法も提案している．

本論文では，スライディングウィンドウモデルで動的に変化する集合に対する効率的な Min-Hash スケッチの生成方法を研究対象とする．Xu らの近似解法ではクエリ集合が Q_t から Q_{t+1} に変化する度に，Min-Hash スケッチ $ms_{Q_{t+1}}$ を完全に再計算する．しかし，Q_t と Q_{t+1} は多数の要素を共有するにもかかわらず，Min-Hash スケッチを完全に再計算するのは非効率である．$ms_{Q_{t+1}}$ を Min-Hash スケッチ $ms_{Q_t)}$ を再利用して計算する手法が望まれるが，そのためには Q_{t+1} に対する Min-Hash のハッシュ値 $h(Q_{t+1})$ を，Q_t に対するハッシュ値 $h(Q_t)$ を更新して計算出来なければならない．幸いそのような技術は既存であり，Datar[1] らがスライディングモデルに動的で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考案している．

しかし，Datar[1] らのハッシュ値更新アルゴリズムは同種類の要素を複数持つ多重集合を取り扱えない．そこで本研究では，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できるように Datar らの手法を拡張した．実験により，提案手法を用いて多重集合に対する Min-Hash スケッチを高速に計算できることを，集合間類似検索が高速に完了することから示した．集合間類似検索問題としては，データベースに n 個のデータストリームのスライディングウィンドウにより定まる n 個の動的に変化する集合 $\{S_1^t, S_2^t, \cdots, S_n^t\}$ が登録され，クエリ集合 Q が静的な Continuous Similarity Search for Evolving Database 問題 [2] を取り扱った．

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，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．3 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．4 章で提案手法を Continuous Similarity Search for Evolving Database 問題に適用して実験的に評価し，時刻経過の度に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集合間類似検索を高速に完了できることを示す．5 章でまとめと今後の課題を述べる．

第 2 章

本研究の関連

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 { みかん, いちご, なし } は果物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，集合は同一数の要素を 1 つしか含まない．集合を同一集合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，{ いちご, いちご, みかん, なし, ばなな, ばなな, ばなな } というような集合である．そして，これらのような集合に対しての集合間類似度を検索する方法が存在する．集合に対する類似度には，さまざまな定義があるが，本論文は Jaccard 係数を用いる．

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，以下の式で定義される．

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

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．

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

ここで，a_i は集合 A が要素 $i \in \pi$ を含む個数である．しかし，このようにクエリとデータベースかの前集合間で 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．

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 [5]．集合間類似検索とは，

- クエリ集合 Q
- n の集合から構成されるデータベース $D = \{S_1, S_2, \cdots, S_n\}$

が与えられて，データベース D から Q と類似した集合を見つける問題である．

そして，Min-hash によって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．Min-Hash はハッシュテーブルを用いて Jaccard 係数の計算回数を減らすことを目的としている．

$$Pr[h(A) = h(B)] = sim(A, B)$$

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．$\lambda = \{x_1, x_2, \cdots, x_{|\lambda|}\}$ をアルファベット集合とする．ハッシュ関数は λ の各アルファベットに対して，$\{1, 2, \cdots |\lambda|\}$ 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．

	a	b	c	d
割り当て値	3	15	7	2

図 2.1　集合のハッシュ値表

$$h(A) = \min_{e \in \mathrm{A}} \pi(e). \tag{2.1}$$

さらに，Min-hash を用いた Jaccard 係数係数の近似計算を紹介する．元々,Min-Hash はハッシュテーブルを用いて，集合間類似検索を高速化することを目指して設計された一方で，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もある．この Min-Hash のハッシュ値の集合を A のスケッチ [4] 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．

2.3.1 多重集合に対する Min-hash

多重集合における Jaccard 係数の計算を Min-hash を用いて近似計算する方法を紹介する．多重集合において，集合の中に２つ以上同じアルファベットが存在することがある．多重集合に対する Min-Hash では同一アルファベットに何個目かということに応じた異なる数値を割り当てる．例として，図 2.2 のように，アルファベット {a,b,c} 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	12	15	7	2
2個目	3	10	13	11

図 2.2　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 2.2 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．

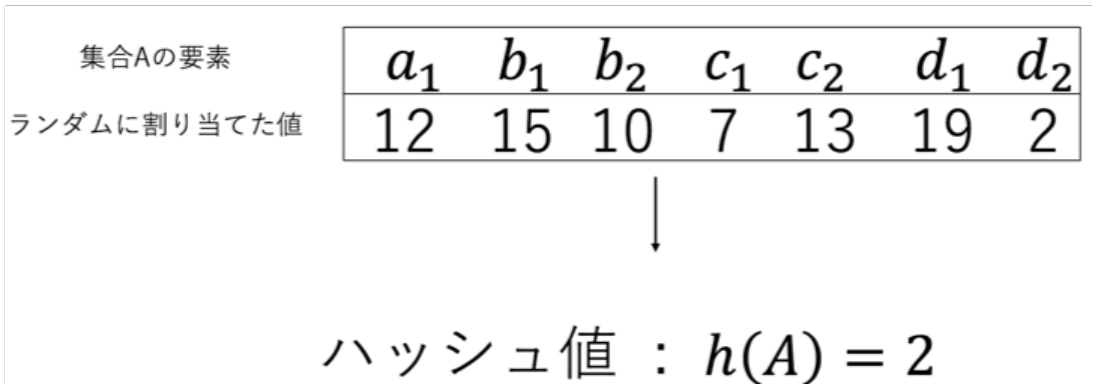

図 2.3　多重集合のハッシュ値計算

2.4　時間と共に変化する集合に対するハッシュ値更新

本節では，Datar ら [1] によって提案された時間と共に変化する集合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．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．

時刻 t における集合を $A_t = \{e_1, e_2, \cdots, e_w\}$ 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 2.4 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．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 2.3 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，$A_t = \{e_1, e_2, e_3, e_4\}$ であり，時刻 $t+1$ では，$A_{t+1} = \{e_2, e_3, e_4, e_5\}$ となる．

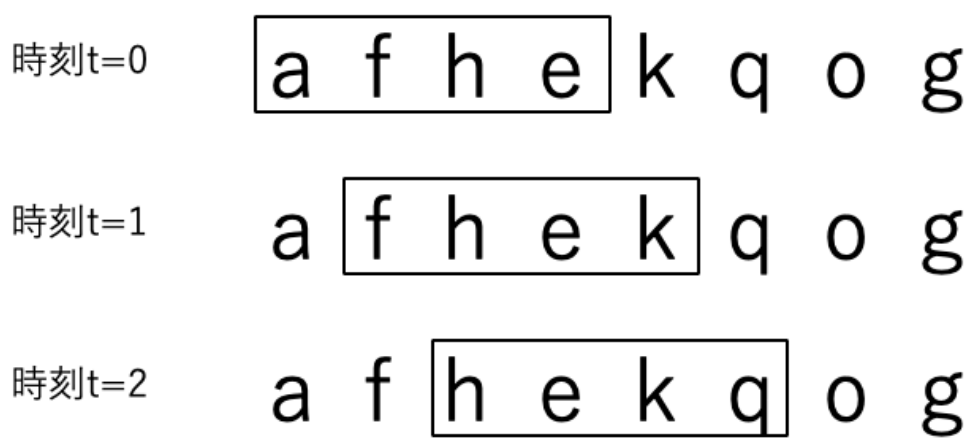

図 2.4　スライディングウインドウ

動的に変化する集合に対してハッシュ値を計算するの自明なやり方は，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．このやり方では，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_1 と e_{w+1} 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．Datar[1] らの手法の基本アイデアは

- $h(A_t)$ を計算した時点に，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発見し削除することで，次の時刻のハッシュ値 $h(A_{t+1})$ を計算する時にチェック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．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内で将来最小値になりうる要素のリスト Minlist を作成して管理する．A_t のハッシュ値は Minlist に残っている要素の割り当て値の最小値となり，式 (2.2) で定義される．

$$h(A_t) = \min_{e \in \text{Minlist}} \pi(e). \tag{2.2}$$

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．

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_1 がウィンドウから離脱した時の処理

  e_1 が Minlist に含まれる時には e_1 を Minlist から削除する．この時，$h(A_t) = \pi(e_1)$ であるため，$h(A_{t+1})$ を新たに Minlist の先頭になった要素の割り当て値とする．

- e_{w+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 $\pi(e_{w+1})$ 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_{w+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_{w+1} のみ残った場合は $h(A_{t+1}) = \pi e_{w+1}$ とする．

このアルゴリズムの動きを図 2.5 に例示する．

一番最初にスライディングウインドウから Minlist を作成する時は，前の要素から一つずつ見ていって，候補リストに加えていく．その過程で，加えた要素より前に自分より大きい要素があれば削除する．その工程を一番後ろの要素まで行うと単調増加の候補リストが作成される．これは候補リストの中で一番前の要素が一番小さく，Min-hash によるハッシュ値の値となるということである．そして，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を削除し，新しい要素 k が入ってくる．そこで，候補リストを見て 2 は入っていないので，一番古い要素は入ってなく，一番後ろに 3 が入ってくるので，候補リストの中から 3 より大きい要素を削除し，候補リストの一番後ろに 3 を加える．つまり，スライディングウインドウ更新の際は，一番古い要素が候補リストに入って入れば削除し，新しい要素の割り当て値より大きい値が候補リストに存在する場合は，その中から削除する．そして，候補リストの一番後ろに新しい要素の割り当て値を加える．

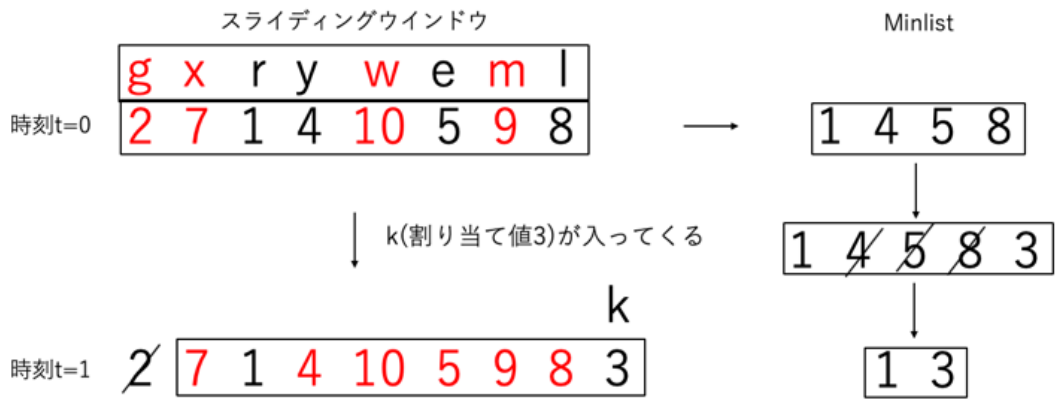

図 2.5　候補リストの作成

第 3 章

本研究の目的

本論文では，動的に変化する多重集合に対するハッシュ値の更新を論じる．データストリームのスライディングウインドウ内の集合 A_k が時刻経過により A_{k+1} に変化した状況を考える．一番単純な手法は，A_t を考慮しないでハッシュ値 $h(A_{t+1})$ を完全に再計算することである．しかし，多重集合に対して，Min-Hash を用いた単純計算をすることは多くの時間を必要とする．

そして，A_t，A_{t+1} が多重集合ではなく集合であれば，2.1.2 項で述べた Datar[1] らによる動的に変化する集合に対してハッシュ値の更新をするアルゴリズムにより，$h(A_t)$ を $h(A_{t+1})$ に更新することによって効率よくできる．しかし，このアルゴリズムは多重集合には対応していない．そこで，本研究では，集合に対応している Datar らのアルゴリズムを多重集合に拡張することを目的とする．

第 4 章

多重集合に対するハッシュ値更新アルゴリズム

本章では，Datar[1] らの提案したアルゴリズムを拡張したアルゴリズムを述べる．

4.1　多重集合になったことの難しさ

Datar[1] らの提案した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アルファベットの個数によって，割り当て値 π が変化するという難しさがある．

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 α とする．$\pi\ (e)$ はスライディングウインドウ内に α 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，$\pi\ (e_i)$=15, $\pi\ (e_j)$=10 であるが，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，$\pi\ (e_j)$=15 となる．

	a	b	c	d
1 個目	12	15	7	2
2 個目	3	10	13	11

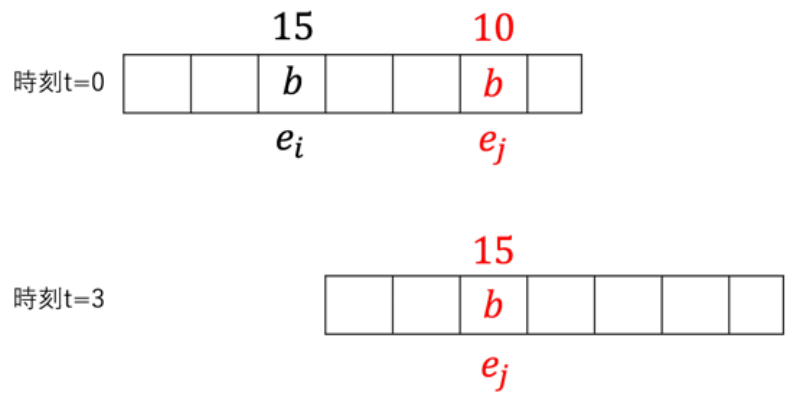

図 4.1　π (e) の変化

4.2　提案手法のアルゴリズム

Datar らのアルゴリズムでは最小値になり得ない要素を削除して計算量を削除する．多重集合の場合，最小値になり得るかどうかの判定を割り当て値が変化することを考慮して実施する必要がある．

提案手法ではスライディングウインドウの中に同一種のアルファベットが複数存在する場合は到着時刻順に順番を決めている．

4.2.1　割り当て値の保持の仕方

多重集合では，同一種アルファベットに対して，割り当て値 π が複数必要であるので π を保持する表を用いる．この割り当て表に修正を加える (図 4.2)．具体的には多重度の上限が 2 で，割り当て値が 1 個目より大きい場合，2 個目の要素は絶対に最小値にならないので，割り当て値を 1 個目と同じ値に書き換える．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．

	a	b	c	d
1個目	12	15	7	2
2個目	3	10	13	11

↓

	a	b	c	d
1個目	12	15	7	2
2個目	3	10	7	2

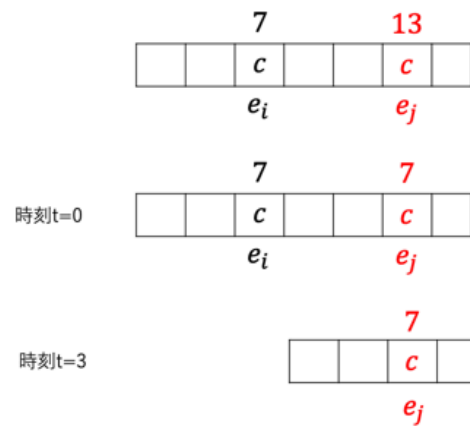

図 4.2　割り当て値の修正

アルファベットの多重度の上限値が一般に $n(\geq 2)$ の場合，割り当て表は以下のように 1 つずつ各アルファベットに対して更新する．

$$\text{for } (i = 1; i \leq n; i++)\{ \text{ if } (\pi\ (a_i) < \pi\ (a_{i+1}))\ \pi\ (a_{i+1}) = \pi\ (a_i)\ \}$$

この修正により同一アルファベットに対する割り当て値は多重度に対して単調減少となる．そして Minlist の中に同一アルファベットを持つ要素は 1 つしか存在しない事を保証できる．e_i は e_j より後ろに存在し，かつ割り当て値が小さい Minlist の中で e_j は e_i で削除できる．

4.2.2 スライディングウインドウの更新

集合の場合と同じように，最小値になりうる候補リスト Minlist も準備する．しかし，集合の時のように単調増加のリストにはならないため，更新の際に，手間は多くなる．そして，多重集合で行う場合，多重集合 A_t が A_{t+1} に変化する時のスライディングウインドウに新しい要素が入ってくる処理と一番古い要素が出ていく処理の 2 つの処理を拡張しなくてはいけない．

(1) 要素 e_1 がウィンドウから出ていく処理

アルファベット x を出ていく要素 e_x の x とする．アルファベット x がスライディングウインドウ内にいくつあるかにより処理は場合分けされる．

- アルファベット x がスライディングウインドウ W に 1 個だけの場合
  まず，割り当て値 $\pi\ (x_1)$ が最小値であるなら，最小値の候補リストである Minlist から x を削除し，最小値を更新後の Minlist の最小値に更新する．また，$\pi\ (x_1)$ が最小値ではないが,Minlist に存在する場合は，Minlist から x を削除する．Minlist に x が存在しない場合は，Minlist を更新しない．
- アルファベット x がスライディングウインドウ W に n 個存在する場合 $(n \geq 2)$
  もし，$\pi\ (x_n)$ が Minlist に含まれるのであれば，Minlist の $\pi\ (x_n)$ を $\pi\ (x_{n-1})$ に更新する．さらに，$\pi\ (x_n)$ が最小値であったのであれば，最小値を更新後の Minlist の最小値に更新する．

(2) 要素 e_{w+1} をウィンドウに追加した時の処理

新しい要素がスライディングウインドウに入る時に，集合の場合には，Minlist の中で新しい要素の割り当て値より大きい値を消す処理しか必要ではなかった．しかし，多重集合の場合は，e_{w+1} の割り当て値が将来増加する可能性があるため，現在の e_{w+1} の割り当て値より大きいという条件で削除を行うと，候補になりうる要素も消してしまう．Minlist の更新の際に，e_{w+1} の現在の割り当て値ではなく将来の e_{w+1} の割り当て値の上限を上回る要素だけ削除する．将来の e_{w+1} の割り当て値の上限は，e_{w+1} のアルファベットを y とすると $\pi\ (y_1)$ となる．

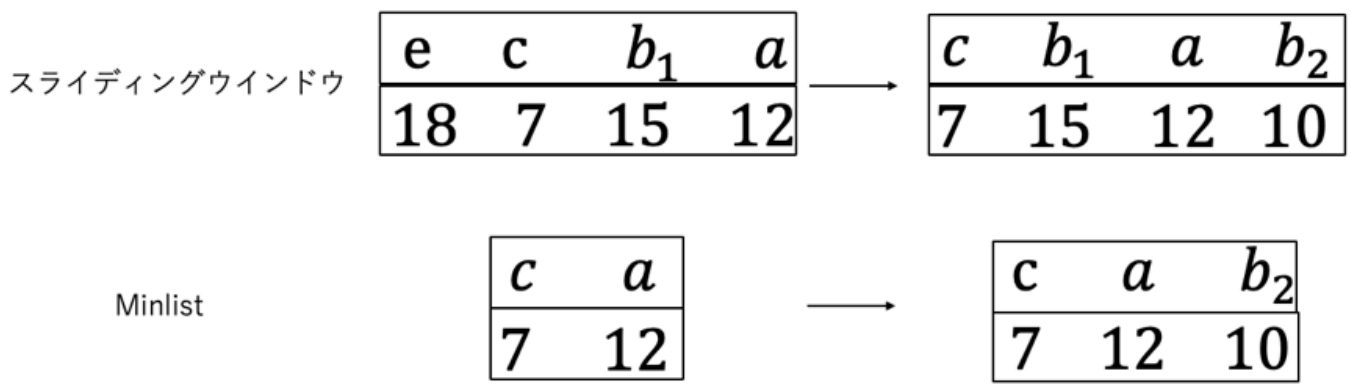

図 4.3　多重集合の MInlist の更新

スライディングウインドウ W にアルファベット y である要素 e_{w+1} が入ってきて，y が W の中に存在する数 m を調べる. 以下の 2 つの処理を行う.

- $\pi\ (y_1)$ と Minlist の中の値を 1 つずつ比べていき，Minlist の中の値 $\geq \pi\ (y_1)$ であれば，その値を Minlist から削除する.
- Minlist[i] のアルファベットが e_{w+1} と同一である場合は Minlist[i] を削除する.

最後に，Minlist の一番後ろに $\pi\ (y_m)$ を登録し，$\pi\ (y_m)$ が Minlist の中で一番小さいなら，最小値を更新する.

第 5 章

実験

この節では，提案した多重集合に対するハッシュ値更新アルゴリズムがハッシュ値を完全に再計算する単純手法より高速に動作するか検証する．このため，提案手法を動的に変化する多重集合を対象として類似検索アルゴリズムに組み込み，単純手法を組み込んだ類似検索アルゴリズムと実行時間を比較する実証を行った．本実験で用いた問題設定は，クエリは静的で変化せず，データベースが動的に変化する問題設定 [3] である．この問題設定の定義を説明する．

$\lambda = \{x_1, x_2, \cdots, x_{|\lambda|}\}$ をアルファベット集合とする．各要素 $e \in \lambda$ であるデータストリーム $X_1, X_2, \cdots$ には，毎時刻新しい要素が 1 つ到着する．データ S_i は X_i の直近 W 個の要素である．データベース D は $\{S_1, S_2, \cdots, S_n\}$
を含む集合である．クエリ Q は静的であり，変化しない．
そして，本実験では，静的で変化しないクエリと動的で変化する 1000 個のデータを持ったデータベースを用いて実験を行った．

- クエリ Q
- データベース $D = \{S_1, S_2, \cdots, S_{1000}\}$

クエリとデータベースは人工データを用いた．

5.1 データの作成

人工データは IBM Quest data generator を用いて作成した．IBM Quest data generator では，まず，以下の 3 種類のパラメータを指定する．

- 集合の数 $ntrans$
- 集合の中の要素数 len
- 要素の種類数 $|\lambda|$

そして，平均が len の長さであり，その中の要素の種類が $|\lambda|$ である集合が $ntran$ 個生成される．例えば,$len = 30$, $|\lambda| = 300$,$ntran = 1000$ で指定すると,300 種類の要素で，長さ約 20 から約 40 までの集合が 1000 個生成される．

クエリは IBM Quest data generator で作成した $ntran$ 個の集合のうちから 1 つがランダムに選ばれた集合とした．データストリームの作成は，まず，クエリと同じように作成した $ntran$ 個の集合の中からランダムに集合を 1 つ選ぶ処理を，選ばれた集合を繋げた長さが時刻 $t+$ スライディングウインドウ W を超える長さになるまで行う．つまり，ランダムな複数の集合を繋げていくということである．しかし，このままでは，スライディングウインドウ内の要素の上限が定まらないため，データストリーム内のスライディングウインドウの要素が上限を超えないようにデータ作成をしなくてはいけない．そのため，データ作成の時に，スライディングウインドウ内の要素の上限を超える場合は，要素を別グループの要素に置き換えるという処理を行った．例えば, 要素の上限が 2 の時に，500 種類要素を用意して，うち 300 種類でデータ作成を基本的に行っていくが，要素がスライディングウインドウ内に 3 つになってしまった場合，3 つ目の要素を残りの 200 種類の中の要素に置き換えるという処理を行った．

Algorithm 3 スライディングウインドウに要素を入れる

Input: IBM で作成したデータ A
Output: データストリーム S

```
    x=301
    for S の長さ < t + W do
 3:   l =rand()%1000
      for j = 0;j < A[l].size();j++ do
        スライディングウインドウ dt の中の数
        whist[A[l][j]] + 1
 6:     if dt.size()¿W then
          whist[dt[0]] − 1;
          dt.erase(dt.begin())
 9:     end if
        if whist[A[l]] が上限値を超えている then
          whist[A[l]] − 1
12:       x+1
          dt に x を加える
          S に A[l] を加える
15:     end if
        if whist[A[l]] が上限値を超えていない then
          dt に A[l] を加える
18:       S に A[l] を加える
        end if
      end for
21: end for
```

このようにデータストリームを n=1000 個作成し，その集合をデータベースとした．

5.2　人工データを用いた実験

実験では，以下のパラメータを用いて行った．

・アルファベットの数の上限値 $L = \{2, 3, 4, 5\}$

・スライディングウインドウ W

この人工データの元で，各時刻 t において, クエリ Q と類似度が高い k 個の集合を検索する実験を行った．時刻を $t = 1$ から 1000 まで進め，1000 回の top-k 検索にかかる合計の処理時間を測定した．top-k のサイズ $k = 10$ として，データ生成のパラメータ $n = 1000$,$W = 100$ を元にして処理時間について評価した．

5.2.1　アルファベットの数の多重度の上限値を変化させて行った実験

データベース内のアルファベットの数の多重度の上限値 $L = 2, 3, 4, 5$ と変化させた場合の実行時間を図 5.1 に示す．上限値を変化させる場合，要素の種類数によって，データベース内に出現する要素の種類数が変わってくるので，L=2 の時は $|\lambda| = 300$,L=3 の時は $|\lambda| = 200$,L=4 の時は $|\lambda| = 150$,L=5 の時は $|\lambda| = 120$ と要素の種類数も変化させて実験を行った．

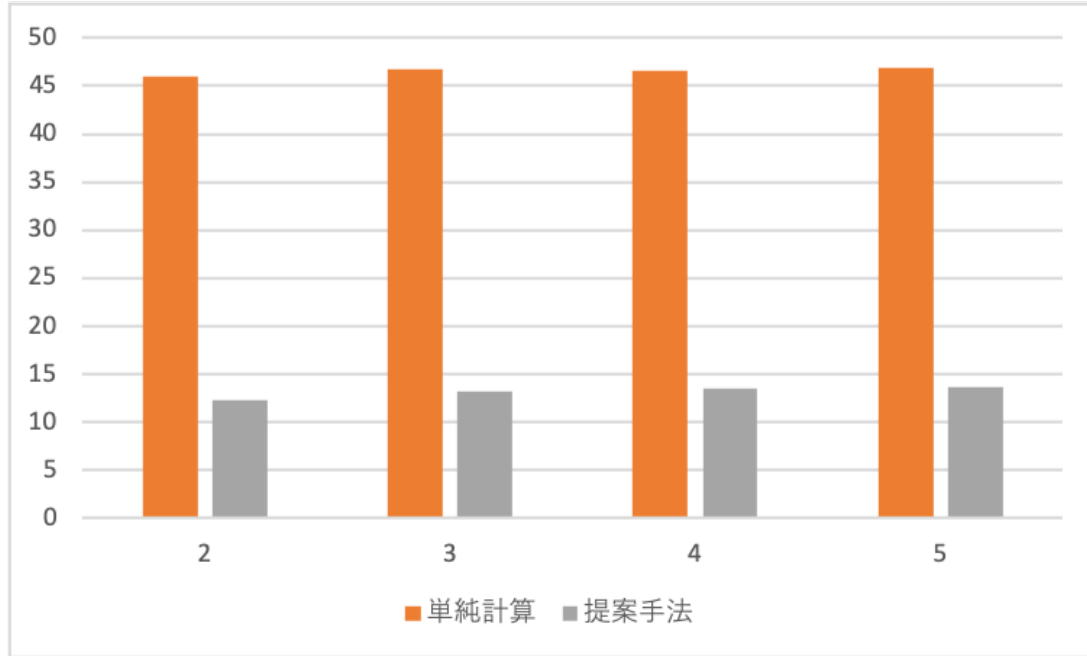

図 5.1　L を変えた時の処理速度

5.2.2 結果

L-2,3,4,5 と変えても，単純手法も提案手法も処理速度に変化はなかった．そして，提案手法は単純手法より処理速度が約 3 倍速かった．

5.2.3 考察

Datar の手法が集合を扱う場合，アルファベットの多重度は必ず 1 なので，要素 x に対する割り当て値 $\pi\ (x)$ は 1 つしか存在せず不変である，このことから x が到着した場合には Minlist 内の $\pi\ (x)$ より大きい要素をすべて削除でき，Minlist が単調増加になる．

しかし，多重集合の場合の要素 x のアルファベットを α とすると，α の多重度により $\pi\ (x)$ の値は動的に変化する．この事から x が到着した場合には Minlist 内では $\pi\ (x)$ ではなく $\pi\ (\alpha_1)$ より大きい要素しか削除できなく，Minlist が単調増加でなくなる．多重度が大きくなるほど $\pi\ (x)$ と $\pi\ (a_1)$ のギャップが大きくなるので，Minlist 内で消されずに残る要素が増える．

以上の要因から実験前は，処理速度は上限値が上がるほど低下することを予想していた．しかし，実際に実行してみると，あまり処理速度に変化がなかった．原因として考えられることは，データを作る際に，データベース内のアルファベットの上限を設定しても，上限に達しているアルファベットの数が少ないことが考えられる．もしくは，アルファベットの要素の種類数が減ることによって，候補となる数も減るために，あまり処理速度が変わらなかったと考えられる．

第 6 章

まとめ

本研究では，動的に変化する集合に対して，Min-Hash 値を更新するアルゴリズムを多重集合へ対応するように拡張することを試みた．多重集合へ対応させることには難しさがあり，その点に関する部分を改良する必要があった．工夫を加えたのは以下の 2 点である．

- スライディングウインドウ内の要素の個数によって，割り当て値 π が変化する点への応用
- Min-Hash の最小値になる候補リスト Minlist が単調増加にならないため，最小値の選び方やデータの保持させ方

割り当て値 π が変化することに対応するために，π を保持する表を作り，その表からハッシュ値になり得ない値を省き，π の保持の仕方に工夫を加えた．そして，Minlist の更新際は，値の変化が多いために，第 4 章のようにアルゴリズムを応用した．よって，Datar らの集合に対するアルゴリズムを多重集合へ対応させ，拡張することができた．
そして，多重集合へ拡張させた提案手法を用いて，人工データによる検索実験を行い，提案手法の評価を行った．この実験では，従来の多重集合に対する類似度計算の手法より，多重集合に対して，高速な類似検索が行えることを示せた．
最後に，今回の研究では，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，多くのメモリを使用している．今後の研究では，ヒストグタムを用いずに類似度計算を行い，処理速度を落とさずに，メモリを削減させることが課題である．そして，実験の元となるデータを本研究で使ったデータよりも上限値まで達する要素が多いデータを使用して，処理速度が低下するかどうか評価する必要がある．

謝辞

本研究を進めるにあたり，ご指導を頂いた卒業論文指導教員の古賀准教授に感謝致します.

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] 野口 大樹 ”集合間類似度を用いたストリームデータの top-k 類似検索における枝刈アルゴリズムの改善”

[4]Dingqi Yang, Bin Li, Philippe Cudre-Mauroux ”POIsketch: Semantic Place Labelong over User Activity Streams”

[5] 岡野原 大輔, “MinHash による高速な類似検索”, https://research.preferred.jp/2011/02/minhash/