並列カウンタ配列による近似頻出値問題のための高速な要約データ構造

金田　悠作† 　有村　博紀†† 　宇野　毅明†††

† 楽天株式会社 楽天技術研究所　〒 158–0094 東京都世田谷区玉川一丁目 14–1 楽天クリムゾンハウス
†† 北海道大学 大学院情報科学研究科　〒 060–0814 北海道札幌市北区北 14 条西 9
††† 国立情報学研究所　〒 101–8430 東京都千代田区一ツ橋 2–1–2
E-mail: †yusaku.kaneta@rakuten.com, ††arim@ist.hokudai.ac.jp, †††uno@nii.jp

あらまし　要素の追加と削除を許したデータストリーム上の近似頻出値問題を効率良く解くための要約データ構造を提案する．要素の全体集合を $U = [0, u)$ で表し，要素 $x \in U$ の入力データストリーム中の頻度を f_x で表すとき，近似頻出値問題は，事前に定めた実数 $\phi \in (0, 1]$ と $\varepsilon \in (0, \phi]$ に対して，$f_x > \phi n$ を満たす要素 x を全て含む近似頻出値集合 $\hat{\mathcal{F}} \subseteq U$ を返す問題である．ここで，n は全要素の合計頻度（追加した要素数と削除した要素数の差）を表し，任意の要素 $x \in \hat{\mathcal{F}}$ は $f_x > (\phi - \varepsilon)n$ を満たす．提案データ構造は，全要素の頻度情報を $O(\log(k/\delta)\log(u)/\varepsilon)$ 領域で近似的し，要素の追加と削除を償却 $O(\log(k/\delta))$ 時間で，近似頻出値集合の出力を $O(\log^2(k/\delta)/\varepsilon)$ 時間でサポートする．ここで，$\delta \in (0, 1]$ は出力の失敗確率を定める任意の実数であり，$k = \lceil 1/\phi \rceil$ とする．同問題を解く Cormode と Muthukrishnan の要約データ構造 (ACM Trans. Database Syst., 2005) と比較した場合，提案データ構造は，同等の計算領域と失敗確率を達成しつつ，全操作を理論的に高速化している．また，この高速化の実現のために，計算機ワード内並列性を利用して複数カウンタ上の更新と比較を高速に実現するデータ構造である並列カウンタ配列を提案する．

キーワード　頻出値問題，要約データ構造，スケッチ

1. はじめに

入力データストリーム S 上の頻出値問題 (frequent elements problem) は，事前に定めた任意の実数 $\phi \in (0, 1]$ に対し，要素の集合 $\mathcal{F} = \{x \in U \mid f_x > \phi n\}$ を求める問題である．ここで，$U = \{0, \ldots, u-1\}$ で要素の全体集合を表す．また，f_x で要素 $x \in U$ の S 中の頻度を表し，$n = \sum_{x \in U} f_x$ で全要素の合計頻度を表す．この問題は，ネットワーク異常の検知，インターネット広告の最適化，検索クエリ傾向の解析などの幅広い分野に応用な基盤技術であり，盛んに研究されてきた [3, 6–10, 14–17]．

一方，この問題を厳密に解くアルゴリズムの計算領域の下限は $\Omega(u)$ ビットである [7]．そのため，理論と実用の両方の興味から，既存研究の多くは，次のデータストリーム上の近似頻出値問題を解く $o(u)$ ビット領域の要約データ構造あるいはスケッチの開発に焦点を当ててきた [7–9, 14, 15, 17]：

問題 (**データストリーム上の近似頻出値問題** [6]) 長さ n の入力データストリーム S 上の近似頻出値問題とは，事前に定めた任意の実数 $\phi \in (0, 1]$ と $\varepsilon \in (0, \phi]$ に対し，近似頻出値集合 $\hat{\mathcal{F}} \subseteq U$ を出力する問題である．ここで，任意の要素 $x \in \hat{\mathcal{F}}$ に対して $f_x > (\phi - \varepsilon)n$ であり，また，$\mathcal{F} \subseteq \hat{\mathcal{F}}$ を満たす．

Cormode と Muthukrishnan [7] は，この近似頻出値問題を確率的に解く要約データ構造 Combinatorial Group Testing (CGT) を提案している．CGT は，全要素の頻度情報を管理する要約データ構造 Count-Min Sketch (CMS) [8] をグループ検査のアイデアで拡張することで構成され，基本操作である要素の追加と近似頻出値集合出力だけでなく，要素の削除と，近似頻度計算，近似内積計算など多くの操作を効率良くサポートする．

具体的には，要約データ構造 CGT は，$O(\log(k/\delta)\log(u)/\varepsilon)$ 領域を用いて，要素の追加と削除を $O(\log(k/\delta)\log(u))$ 時間でサポートし，任意の時点における近似頻出値集合 $\hat{\mathcal{F}}$ を $O(\log(k/\delta)(\log(u) + \log(k/\delta))/\varepsilon)$ 時間で出力する [7]．ここで，$k = \lceil 1/\phi \rceil$ は解の最大個数を表す整数であり，$\delta \in (0, 1]$ は出力の失敗確率を定める任意の実数である．

この要約データ構造のボトルネックは，全ての操作の計算時間に出現する要素の表現長 $\log u$ の項である．実際，Cormode と Muthukrishnan [7] は，このボトルネックを部分的に改善するために更新時間と計算領域のトレードオフを与えており，$O(\tau \log(k/\delta)\log_\tau(u)/\varepsilon)$ 領域を許すことで，要素の追加と削除を $O(\log(k/\delta)\log_\tau(u))$ 時間に高速化できることを示した．ここで，$\tau \geq 2$ は任意の整数である．一方，近似頻出値集合の出力は $O(\log(k/\delta)(\tau \log_\tau(u) + \log(k/\delta))/\varepsilon)$ に低速化する．

しかし，この Cormode と Muthukrishnan のトレードオフには，以下の二つの課題がある：

(1) 計算領域を $o(u)$ ビットに保ったまま，要素の追加と削除を $O(\log u)$ 倍に高速化できない．

(2) 計算領域と成功確率を犠牲にすることなく，更新操作と出力操作を同時に高速化できない．

本稿では，既存の要約データ構造 CGT [7] と同じデータ構造サイズと成功確率を達成しつつ，要素の追加と削除を $O(\log u)$ 倍に，近似頻出値出力を $O(\log(u)/\log(k/\delta))$ 倍に同時に高速化した要約データ構造 Bit-Parallel Combinatorial Group Testing (BP-CGT) を与えることで，上述の二つの課題を同時に解決できることを示す．本稿の主定理を以下に示す：

定理 1 近似頻出値問題に対して，要素の追加と削除を償却 $O(\log(k/\delta))$ 時間でサポートし，任意の時点における近似頻出値集合を $O(\log^2(k/\delta)/\varepsilon)$ 時間で出力する $O(\log(k/\delta)\log(u)/\varepsilon)$ 領域のデータ構造が存在する．ここで，$k = \lceil 1/\phi \rceil$ であり，任意の実数 $\delta \in (0, 1]$ に対して出力の成功確率は $1 - \delta$ である．

本稿では，Cormode と Muthukrishnan の結果 [7] と同様に，計算モデルとしてワード長 w の unit-cost word RAM を仮定する．また，各要素の表現長は $w \geq \log u$ を満たすと仮定する．

提案データ構造の高速化のアイデアは，**ビット並列カウンタ配列**によるサイズ $m = O(w)$ のカウンタ配列 $C[0..m-1]$ の高速な実装である．並列カウンタ配列は，主に近似文字列照合の高速化に利用されてきた，計算機ワード内並列性を利用して複数個のカウンタを高速に処理するデータ構造である．以降では，ビット並列カウンタ配列を単に並列カウンタ配列と呼ぶ．

Baeza-Yates と Gonnet [1] は，k 文字の異なりを許す近似文字列照合の高速化のために，単一の計算機ワードに詰め込まれた $O(w/\log k)$ 個の $O(\log k)$ ビット整数カウンタに対する定数値の加減算を定数時間で実現する並列カウンタ配列を与えた．

Grabowski と Fredriksson [12] は，$O(\log k)$ ビット整数カウンタと $O(\log\log k)$ ビット整数カウンタを併用することで，$O(w/\log\log k)$ 個の $O(\log k)$ ビット整数カウンタに対する定数値の加減算を償却定数時間で実現する並列カウンタ配列を与えている．また，これを拡張し，$O(w)$ 個の任意ビット長の整数カウンタに対する定数値の加算を償却定数時間で実現する並列カウンタ配列を与えている．

最後に，Bille と Thorup [2] は，正規表現照合の文字クラスの繰り返しを高速に扱うために，$O(w)$ 個の任意ビット長の整数カウンタに対する定数値の減算と，各カウンタごとに異なる整数への初期化，正数判定を償却定数時間で実現する並列カウンタ配列を与えている．

本稿では，$O(w)$ 個の整数カウンタに対する定数値の加算と減算の両方向の更新を償却定数時間でサポートする並列カウンタ配列を与える．これに対し，既存の並列カウンタ配列は，(償却) 定数時間で処理可能なカウンタの個数が少ない，あるいは，定数値の加算と減算の片方向の更新だけをサポートしていた．

1.1. 関 連 研 究

データストリーム上の（近似）頻出値問題に関する既存研究は，データ構造のサポートする操作の種類に従って分類できる [6]．以下では，(近似) 頻出値出力と要素の追加と削除をサポートするデータ構造に関する既存研究について述べる：

Frandsen と Skyum [10] は，$\phi = 0.5$ に対する頻出値問題である過半数値問題を解く $O(n)$ 領域のデータ構造を与えた．このデータ構造は，過半数値出力と要素の追加と削除を定数時間でサポートする．第 4 節で与える要約データ構造は，過半数値出力に近似を許すことで，このデータ構造と同等の計算時間を実現しつつ，データ構造サイズを $O(\log u)$ 領域に改善する．

Cormode と Muthukrishnan [7] は，$\phi = \varepsilon = 0.5$ の場合の近似頻出値問題である近似過半数値問題を確率的に解く $O(\log u)$ 領域の要約データ構造を与えた．このデータ構造は，近似過半数値出力と要素の追加と削除を $O(\log u)$ 時間でサポートする．第 4 節で与える要約データ構造は，このデータ構造と同等の計算領域を実現しつつ，近似過半数値出力を定数時間に要素の追加と削除を償却定数時間に改善する．また，Cormode と Muthukrishnan [7] は，近似頻度計算と要素の追加と削除をサポートする要約データ構造を近似頻出値問題に拡張するアイデアを与え，Cormode と Muthukrishnan [8] は，このアイデアを利用することで，CMS を近似頻出値問題に拡張した要約データ構造を示した．近年，CMS と CGT を含むその拡張は，データストリーム処理だけでなく，圧縮センシング，次元削減，疎なフーリエ変換などに応用可能な大規模データ処理の基盤技術としてますます注目されている [13].

Cormode と Hadjieleftheriou [6] は，近似頻出値問題を解く要約データ構造に関する実験的性能評価の結果を与えた．要素の追加と削除をサポートする要約データ構造内の比較において，トレードオフ変数を $b = 16$ とした CGT は，最も高速かつ高精度であった．詳細な結果は，文献 [6] を参照されたい．

1.2. 本稿の構成

第 2 節で基本的な定義と表記を与える．また，以降の節で用いる整数の二進表現に関する性質を述べる．

第 3 節で並列カウンタ配列を与える．このデータ構造は，計算機ワード内並列性を利用することで，サイズ $O(w)$ のカウンタ配列の任意の要素カウンタに対し，定数増減を償却定数時間で，符号判定を定数時間でサポートする．ここで，定数増減の対象と符号判定の結果は，共に要素カウンタ位置を指定するビット列で示される．

第 4 節で $\phi = \varepsilon = 0.5$ の場合の近似頻出値問題である近似過半数値問題を解く効率良い要約データ構造を与える．第 3 節の並列カウンタ配列を既存の要約データ構造 [7] のアイデアと組み合わせることで，追加と削除を償却定数時間で，過半数値出力を定数時間で実現できることを示す．

第 5 節で任意の近似頻出値問題を解く効率良い要約データ構造を与える．第 4 節のデータ構造を既存の要約データ構造 [7] のアイデアと組み合わせることで，追加と削除を高速化できることを示す．また，既存データ構造 [7] を修正することで，同等の成功確率のまま，頻出値出力を高速化できることを示す．

2. 準　　備

整数全体の集合を $\mathbb{Z} = \{0, \pm 1, \ldots\}$ と書き，自然数全体の集合を $\mathbb{N} = \{0, 1, \ldots\}$ と書く．整数 i, j $(i \leq j)$ に対する区間を $[i..j] = \{i, i+1, \ldots, j\} \subseteq \mathbb{Z}$ で表す．また，任意の正数 p に対し，$[p] = [0..p-1] \subseteq \mathbb{N}$ と書く．

計算モデル 計算モデルとして，ワード長 $w \geq \log u$ ビットの unit-cost word-RAM を仮定する．このモデルでは，w ビット整数に対する論理演算と乗算と除算を含む四則演算，および，メモリ中の任意の連続する w ビットの読み書きを定数時間で実行できる．また，複数の m ビット整数を単一のワードに詰め込むことで，サイズ $\lfloor w/m \rfloor$ の m ビット整数配列に対する要素ごとの加減算と大小比較を定数時間で実行できる．

整数の二進表現 任意の整数 $x \in \mathbb{N}$ に対し，$\mathsf{bit}(x, i) \in \{0, 1\}$

で x の i 番目のビットを表す．任意の集合 $S \subseteq [m]$ に対し，$\mathsf{int}(S) \in [0..2^m-1]$ で $\mathsf{bit}(\mathsf{int}(S), i) = 1 \iff i \in S$ を満たす整数を表す．また，任意の整数 $x \in \mathbb{Z}$ $(x \neq 0)$ に対し，$\mathsf{lsb}(x) = \min\{i \in \mathbb{N} \mid x \bmod 2^i = 0\}$ と定める．言い換えると，$\mathsf{lsb}(x)$ は，x の二進表現における最下位 1 の位置を表す．以降では，便宜上，$\mathsf{lsb}(0) = \infty$ と定める．二値カウンタの桁上げ回数と同様の議論により，次の補題が成り立つ：

補題 1 任意の正数 n に対して，$\mathsf{lsb}(1), \ldots, \mathsf{lsb}(n)$ の各要素は償却定数時間で計算可能．

本稿で利用する関数 lsb の性質を与える．整数の加算に関し，次の補題が成り立つ：

補題 2 任意の整数 x, y に対し，以下が成り立つ：

(1) $\mathsf{lsb}(x+y) = \mathsf{lsb}(x) \iff \mathsf{lsb}(x) < \mathsf{lsb}(y)$.

(2) $\mathsf{lsb}(x+y) > \mathsf{lsb}(x) \iff \mathsf{lsb}(x) = \mathsf{lsb}(y)$.

(3) $\mathsf{lsb}(x+y) < \mathsf{lsb}(x) \iff \mathsf{lsb}(x) > \mathsf{lsb}(y)$.

3. 並列カウンタ配列

本節では，サイズ $O(w)$ のカウンタ配列上の定数更新と定数比較を並列にサポートするデータ構造を与える．カウンタ配列は，近似頻出値問題を解く要約データ構造を実現する際に重要になる．本節の以降では，m でカウンタ配列のサイズを表し，n で各要素カウンタのビット長を表す．また，$m = O(w)$ と $n = O(w)$ を仮定する．

カウンタ配列 $C[0..m-1]$ は，m 個の符号付き整数カウンタ $C[0], \ldots, C[m-1] \in \mathbb{Z}$ を管理し，次の基本操作をサポートするデータ構造である．ここで，$x \in [0..2^m-1]$ とする：

定義 1 (**カウンタ配列の基本操作**)

$\mathsf{increment}(C, x)$: 各 $i \in [m]$ に対して，$C[i] \leftarrow C[i] + \mathsf{bit}(x, i)$.

$\mathsf{decrement}(C, x)$: 各 $i \in [m]$ に対して，$C[i] \leftarrow C[i] - \mathsf{bit}(x, i)$.

$\mathsf{iszero}(C)$: 整数 $\mathsf{int}(\{i \in [m] \mid C[i] = 0\})$ を返す．

以下では，基本操作 increment, decrement, iszero をそれぞれ単位増加，単位増減，ゼロ判定と呼ぶ．また，単位増加と単位減少を合わせて単位増減と呼ぶ．

カウンタ配列 $C[0..m-1]$ は単に整数配列で実装できる．この場合，配列を走査することで各基本操作を $\Theta(m)$ 時間でサポートできる．実際，近似頻出値問題を解く要約データ構造 [7] は，カウンタ配列をサイズ $O(\log u)$ の整数配列で実装している．

本節の以降では，通常の論理演算と算術演算だけを用いて単位増減とゼロ判定をそれぞれ償却定数時間と定数時間で実現できることを示す．本節の主結果として次の補題を示す：

定理 2 (**並列カウンタ配列の計算量**) 語長 w の word RAM を仮定したとき，サイズ $O(w)$ カウンタ配列上の単位増減とゼロ判定をそれぞれ償却定数時間と定数時間でサポートする $O(n)$ 領域のデータ構造が存在する．

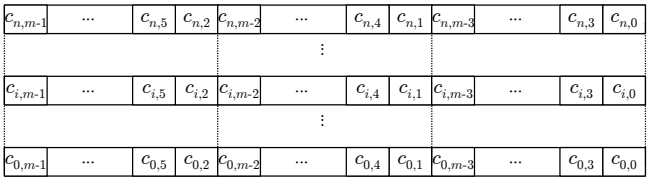

図 1 カウンタ配列 $C[0..m-1]$ を表す小カウンタの配置．ただし，簡単化のために $m \bmod 3 = 0$ を仮定．

3.1. 提案データ構造

並列カウンタ配列 $C[0..m-1]$ の各要素カウンタ $C[i] \in \mathbb{Z}$ を n 個の小カウンタ $c_{i,0}, \ldots, c_{i,n-1}$ で表す．ここで，小カウンタは $C[i] = \sum_{j=0}^{n-1} c_{i,j} 2^j$ を満たす．このとき，単一の要素カウンタ $C[i]$ は，t 回目の更新操作において $c_{i,0}, \ldots, c_{i,\mathsf{lsb}(t)}$ だけを変更することで，両方向の更新操作 increment と decrement を共に償却定数時間でサポートできる．

以下では，単一のカウンタ C を表す小カウンタ $c_0, \ldots, c_{n-1}$ に対して基本アイデアを示す．サイズ m のカウンタ配列を扱うためには，各要素カウンタ $C[i]j$ を $i \bmod 3$ の値に従ってグループに分け，各グループ内の要素カウンタを番号 i の小さい順に下位から上位に向かって連続して配置し，単一のカウンタに対する更新式を論理演算と算術演算を組み合わせて並列に実行すれば良い．図 1 に配列サイズ m が 3 で割り切れる場合の配置を示す．

並列カウンタ配列の各要素カウンタは，冗長二進カウンタの一種と見なせる．通常の冗長二進カウンタは，高速な単位増減の実現のために各桁を $\{0, \pm 1\}$ 中の値で表す．これに対し，並列カウンタ構造の各要素カウンタは，桁上げを遅延するために各桁を $\{0, \pm 1, \pm 2\}$ 中の値で表す．

単位増減 並列カウンタ配列の償却定数時間 increment/decrement 手続きについて述べる．両方向の更新を償却定数時間でサポートするために，更新回数 $t \geq 0$ に対して下位 $\min\{\mathsf{lsb}(t), n-1\}$ 個の小カウンタのみを更新する．

任意の時刻 t の更新操作後，各 $0 \leq \ell < \mathsf{lsb}(t)$ に対する小カウンタは，通常の冗長二進カウンタと同様に $c_\ell \in \{0, \pm 1\}$ に設定する．一方で，$\mathsf{lsb}(t) \leq \ell < n$ に対する小カウンタは，下位の小カウンタからの桁上げを格納するため $c_\ell \in \{0, \pm 1, \pm 2\}$ に設定する．桁上げ計算は，以下の式で計算される：

$$(c_i, x) \leftarrow \begin{cases} (c_i + x - 2, +1) & \text{if } c_i + x \geq +2. \\ (c_i + x + 2, -1) & \text{if } c_i + x \leq -2. \\ (c_i + x, 0) & \text{otherwise.} \end{cases}$$

任意の時刻 t に対し，$\mathsf{pre}(t) = \max\{t - 2^{\mathsf{lsb}(t)}, 0\}$ で小カウンタ $c_{\mathsf{lsb}(t)}$ を最後に参照した時刻を表す．このとき，小カウンタの値に関する次の補題が成り立つ：

補題 3 時刻 t の更新の開始において，小カウンタは次を満たす：

(1) 各 $0 \leq i < \mathsf{lsb}(t)$ に対し，$c_i \in \{0, \pm 1, \pm 2\}$.

(2) 各 $\mathsf{lsb}(t) \leq i < \mathsf{lsb}(\mathsf{pre}(t))$ に対し，$c_i \in \{0, \pm 1\}$.

(3) 各 $\mathsf{lsb}(\mathsf{pre}(t)) \le i < n$ に対し，$c_i \in \{0, \pm 1, \pm 2\}$.

証明. 更新回数 t に関する帰納法で示す.

初めに，更新回数 $t = 0$ の場合に主張が成立することを示す：このとき，$\mathsf{lsb}(t) = n$ であり，任意の $0 \le \ell < n$ に対して $c_\ell = 0$ に初期化されてため成り立つ.

次に，任意の更新回数 $t \ge 1$ 以前に主張が成立すると仮定し，更新回数 $t+1$ で同様に成立することを示す：

1. $\mathsf{lsb}(t+1) > \mathsf{lsb}(t)$ の場合：補題 2 (2) より，$\mathsf{lsb}(t) = 0$ である．時刻 $\mathsf{pre}(t+1)$ の桁上げ計算により，$c_{\mathsf{lsb}(t+1)} \in \{0, \pm 1\}$ である．また，時刻 $\mathsf{pre}(t+1) < t' < t$ の桁上げ計算により，任意の $0 \le i < \mathsf{lsb}(t)$ に対して $c_i \in \{0, \pm 1, \pm 2\}$ である．以上より，$\mathsf{lsb}(t+1) > \mathsf{lsb}(t)$ の場合に主張は成立する.

2. $\mathsf{lsb}(t+1) < \mathsf{lsb}(t)$ の場合：補題 2 (3) より，$\mathsf{lsb}(t+1) = 0$ である．したがって，任意の $\mathsf{lsb}(t+1) = 0 \le i < n$ に対し，$|c_i| \le 2$ を示せば良い．時刻 t に関する帰納法の仮定より，時刻 $t+1$ の更新後に $|c_0| \le 2$ を満たす．また，他の全ての小カウンタは時刻 t の更新後の値のままである．以上より，$\mathsf{lsb}(t+1) < \mathsf{lsb}(t)$ の場合に主張は成立する.

□

以上の補題より，任意の時刻 t で桁上げ計算は正しく計算される．また，任意の時刻 t の更新操作後，下位 $\mathsf{lsb}(t)+1$ 個の小カウンタの値 $L_{\mathsf{lsb}(t)} = \sum_{i=0}^{\mathsf{lsb}(t)} c_i 2^i$ は，$|L_{\mathsf{lsb}(t)}| \le 3 \cdot 2^{\mathsf{lsb}(t)} - 1$ を満たす.

ゼロ判定 並列カウンタ配列の定数時間 iszero 手続きに関して述べる．基本アイデアは，全ての $0 \le i < n$ に対して上位の小カウンタの値 $U_i = \sum_{\ell=i}^{n} c_\ell 2^\ell$ を格納することである．各 U_i を陽に格納するには n ビット必要だが，定数ビットの情報を格納することで，任意の時刻 t において，各 U_i を更新し，$C = U_0 = 0$ を判定できることを示す.

各 U_i は，ある整数 $u_i \in \mathbb{Z}$ が存在し，$U_i = u_i 2^i$ と書ける．また，時刻 t の更新後，$L_{\mathsf{lsb}(t)} = 3 \cdot 2^i - 1$ である．したがって，$-1 \le u_{\mathsf{lsb}(t)+1} \le +1$ の場合に陽に u_i を格納し，それ以外の場合に特別な値 $\perp$ を格納することで，U_i を定数ビットで表現しつつ，任意の時刻 t で $C = U_{\mathsf{lsb}(t)+1} + L_{\mathsf{lsb}(t)} = 0$ を判定できる．ここで，任意の整数 $x \in \mathbb{Z}$ と $\perp$ との算術演算の結果は $\perp$ であり，また $\perp$ といかなる整数との比較結果は偽であるとする．このとき，iszero 操作は，単に $u_0 = 0$ を検査することで定数時間でサポートできる.

任意の時刻 t において更新される小カウンタは $c_0, \ldots, c_{\mathsf{lsb}(t)}$ であるため，$u_0, \ldots, u_{\mathsf{lsb}(t)}$ を更新すれば良い．任意の時刻 t に対し，各 u_i $(0 \le i \le \mathsf{lsb}(t))$ は，次の通り更新できる：

$$u_i \leftarrow \begin{cases} \perp & \text{if } |2u_{i+1} + c_i| > 2. \\ 2u_{i+1} + c_i & \text{otherwise.} \end{cases}$$

カウンタ配列 $C[0..m-1]$ 上の符号判定は，次の通り計算できる：正の要素カウンタ位置を格納する整数 p をデータ構造に追加する．データ構造の構築時に $p \leftarrow 0$ と初期化した後，p は，

Algorithm 1 並列カウンタ更新手続き.

procedure update$(c_0, \ldots, c_{n-1}, u_0, \ldots, u_{n-1}, x_t)$:
  $x \leftarrow x_t$
  for $i = 0$ **to** $\min\{\mathsf{lsb}(t), n-1\}$ **do**
    $$(c_i, x) \leftarrow \begin{cases} (c_i + x - 2, +1) & \text{if } c_i + x \ge +2. \\ (c_i + x + 2, -1) & \text{if } c_i + x \le -2. \\ (c_i + x, 0) & \text{otherwise.} \end{cases}$$
  for $i = \min\{\mathsf{lsb}(t), n-1\}$ **to** 0 **do**
    $$u_i \leftarrow \begin{cases} \perp & \text{if } (u_{i+1} = \perp) \vee (|2u_{i+1} + c_i| > 2). \\ 2u_{i+1} + c_i & \text{otherwise.} \end{cases}$$

単位増加の開始時に $p \leftarrow p \mid (\mathsf{iszero}(C)\ \&\ x)$ と，単位減少の終了時に $p \leftarrow p\ \&\ {\sim}(\mathsf{iszero}(C)\ \&\ x)$ と更新できる．負の要素カウンタ位置を格納する整数も同様に更新できる．以上により，要素カウンタの符号判定は，定数時間で計算できる.

カウンタ配列 $C[0..m-1]$ 上の定数比較は，次の通り計算できる：各要素カウンタ $C[i]$ と比較したい定数を $\alpha[i]$ と書く．並列カウンタ配列の初期化の際に，各要素カウンタ $C[i]$ を定数 $-\alpha[i]$ に設定する．このとき，等価判定 $C[i] = \alpha[i]$ はゼロ判定に帰着でき，大小判定 $C[i] > \alpha[i]$ と $C[i] < \alpha[i]$ は符号判定に帰着できる．以上により，要素カウンタと定数の比較は，$O(\max_{i \in [m]} \alpha[i])$ 時間の前処理の後，定数時間で計算できる.

4. 近似過半数値問題に対する要約データ構造

本節では，$\phi = \varepsilon = 0.5$ に対する近似頻出値問題である近似過半数値問題を解く高速な要約データ構造を与える．この要約データ構造は，$O(\log u)$ 領域を用いて，過半数値の出力と要素の追加と削除をそれぞれ定数時間と償却定数時間でサポートする．ただし，過半数値が存在しない場合は，任意の値を出力する．以下では，$m = \log u$ で各要素の表現長を表す．本節で与える要約データ構造の基本操作を次に示す：

定義 2 (**近似過半数値問題を解く要約データ構造の基本操作**)

$\mathsf{insert}_2(S, x)$: x を要約データ構造 S に追加する：$f_x \leftarrow f_x + 1$.

$\mathsf{delete}_2(S, x)$: x を要約データ構造 S に削除する：$f_x \leftarrow f_x - 1$.

$\mathsf{output}_2(S)$: $\phi = \varepsilon = 0.5$ に対する単元集合 $\hat{\mathcal{F}}$ を返す.

4.1. Cormode と Muthukrishnan の既存データ構造

近似過半数値問題に対する Cormode と Muthukrishnan の要約データ構造 [7] は，近似過半数値出力と要素の追加と削除を $O(\log u)$ 時間でサポートする.

このデータ構造は，カウンタ配列 $C[0..m-1]$ と整数 $n \in \mathbb{N}$ として表せ，$C[i] = \sum_{x \in U} f_x \times \mathsf{bit}(x, i)$ と $n = \sum_{x \in U} f_x$ を満たすように更新される．したがって，要素の追加と削除は，各要素カウンタ $C[i]$ の単位増減で実現できる．また，過半数値 x が存在するならば，任意の $0 \le i < m$ に対して，$\mathsf{bit}(x, i) = 1$ のとき，またそのときに限り $C[i] > \lceil 0.5n \rceil$ である．したがって，過半数値の出力は，各要素カウンタと整数値の比較で実現できる.

Algorithm 2 近似過半数値問題に対する要素の追加操作.

```
1: procedure insert_2(S = (C[0..m − 1], n), x)
2:     n ← n + 1
3:     increment(C, x)
4:     if n is odd then
5:         increment(C, 2^m − 1)
```

Algorithm 3 近似過半数値問題に対する要素の削除操作.

```
1: procedure delete_2(S = (C[0..m − 1], n), x)
2:     n ← n − 1
3:     decrement(C, x)
4:     if n is even then
5:         decrement(C, 2^m − 1)
```

Algorithm 4 近似過半数値問題に対する出力操作.

```
1: procedure output_2(S = (C[0..m − 1], n))
2:     return check_{>0}(C)
```

4.2. 提案データ構造

本稿の近似過半数値問題に対する要約データ構造の基本アイデアは，前節の並列カウンタ配列を既存の要約データ構造 [7] と組み合わせることである．

カウンタ配列 $C[0..m-1]$ を前節のデータ構造で実現することで，要素の追加と削除は，ただちに償却定数時間に高速化できる．一方，近似過半数値出力は，各要素カウンタと実行時に決まる $O(w)$ ビット整数の比較 $C[i] > \lceil 0.5n \rceil$ を伴うため，$O(w)$ 時間のままである．

提案データ構造は，近似過半数値出力を高速化するために，各要素カウンタ $C[i]$ に格納する値を以下の通り変更する：

$$C[i] = \sum_{x \in U} f_x \times \mathsf{bit}(x, i) - \lceil 0.5n \rceil.$$

この変更により，近似過半数値出力は，カウンタ配列に対する正数判定 $C[i] > 0$ に帰着できる．また，右辺の第一項と第二項は，それぞれ高々± 1 しか変化せず，定数時間で求まる．

本稿で提案する近似過半数値問題を解く要約データ構造の各操作を Algorithm 2–4 に示す．ここで，$\mathsf{check}_{>0}(C[0..m-1])$ は，符号判定 $C[i] > 0$ を満たすとき，そのときに限り $\mathsf{bit}(x, i) = 1$ を満たす整数 x を返す．本節の議論と定理 2 より，近似過半数値問題を解く要約データ構造に関する次の定理を得る：

定理 3 (**近似過半数値問題の計算量**) 要素の追加と削除を償却定数時間で，近似過半数値出力を定数時間でサポートする $O(\log u)$ 領域のデータ構造が存在する．

5. 近似頻出値問題に対する要約データ構造

本節では，任意の実数 $0 < \phi \leq 1$ に対する近似頻出値問題を確率的に解く要約データ構造 BP-CGT を与える．このデータ構造は，$O(\log(k/\delta)\log(u)/\varepsilon)$ 領域を用いて，要素の追加と削除を償却 $O(\log(k/\delta))$ 時間で，近似頻出値出力を $O(\log^2(k/\delta)/\varepsilon)$ 時間でサポートする．ここで，δ は，出力の失敗確率を定める任意の定数である．本節で与える要約データ構造の基本操作を次に示す：

定義 3 (**近似頻出値問題を解く要約データ構造の基本操作**)

$\mathsf{insert}(S, x)$: x を要約データ構造 S に追加する：$f_x \leftarrow f_x + 1$.

$\mathsf{delete}(S, x)$: x を要約データ構造 S に削除する：$f_x \leftarrow f_x - 1$.

$\mathsf{output}(S)$: 任意の ϕ, ε に対する集合 $\hat{\mathcal{F}}$ を返す.

5.1. Cormode と Muthukrishnan の既存データ構造

近似頻出値問題に対する Cormode と Muthukrishnan の要約データ構造 CGT [7] は，任意の整数 $b > 0$ に対し，要素の追加と削除を $O(\log(k/\delta)\log_\tau(u))$ 時間で，近似頻出値出力を $O(\log(k/\delta)(\tau \log_\tau(u) + \log(k/\delta))/\varepsilon)$ 時間でサポートする．ただし，出力の成功確率は $1 - \delta$ である．

要約データ構造 CGT [7] は，次の要素で構成される．ここで，整数 r と c の具体的な値は，以降の説明で決定する：

- $n \in \mathbb{N}$: 総頻度カウンタ.
- $N[1..r][1..c]$: 部分頻度カウンタ配列.
- $C[1..r][1..c][0..m-1]$: 部分ビット頻度カウンタ配列.
- $h_1, \ldots, h_r$: r 個の汎用ハッシュ関数 $h_i : U \to [1..c]$.

ここで，総頻度カウンタは $n = \sum_{x \in U} f_x$ を格納し，部分頻度カウンタ配列は $N[i][j] = \sum_{x \in U} f_x \times [h_i(x) = j]$ を格納し，部分ビット頻度カウンタ配列の各要素は $C[i][j][\ell] = \sum_{x \in U} f_x \times [h_i(x) = j] \times \mathsf{bit}(x, \ell)$ を格納する．

要約データ構造 CGT は，要素 $x \in U$ の追加と削除を r 個のカウンタ配列上の単位増減に帰着する．また，出力を $r \times c$ 個のカウンタ配列上の大小比較と高々$r^2 \times c$ 回のハッシュ関数計算に帰着する．要素の追加と削除の実装方法は，データ構造を構成する各要素の定義より明らかである．以下では，出力の実装方法を説明する：各 i において，要素 $x \in \mathcal{F}$ と衝突する要素の合計頻度 $f_{\neq x}$ が，$f_{\neq x} \leq \phi n$ を満たすとき，近似過半数値出力と同じアイデアで要素 x を計算できる．汎用ハッシュ関数の性質より，要素 x と衝突する要素の合計頻度の期待値は，

$$\mathbb{E}[f_{\neq x}] \leq \frac{n}{c}$$

であり，Markov の不等式より，以下の関係が成り立つ：

$$\mathbb{P}[f_{\neq x} \geq \phi n] \leq \frac{\mathbb{E}[f_{\neq x}]}{\phi n} = \frac{1}{c\phi}.$$

このとき，$c \geq 2/\phi$ と設定することで，$\mathbb{E}[f_{\neq x}] \leq \phi n/2$ と $\mathbb{P}[f_{\neq x} \geq \phi n] \leq 1/2$ を満たす．言い換えると，確率 $1/2$ 未満で $f_{\neq x} < \phi n$ である．全 r 回の試行で $f_{\neq x} \geq \phi n$ となる確率は $(1/2)^r = \delta/k$ 以下である．したがって，Boole の不等式より，$\mathcal{F}$ に含まれる高々$k = \lceil 1/\phi \rceil$ 個の要素は，成功確率 $1 - \delta$ で全て出力される．

各要素 $x \in \hat{\mathcal{F}}$ の頻度が $f_x > (\phi - \varepsilon)n$ を満たすためには，x を $\hat{\mathcal{F}}$ に追加する前に頻度検査を行う．この検査は，全ての $i \in [1..r]$ で $N[i][h_i(x)] > \phi n$ を満たすことを確認する．任意

Algorithm 5 要約データ構造 BP-CGT の要素の追加操作.

```
1: procedure insert(S=(C[1..r][1..c][0..m−1], N[1..r][1..c], n), x)
2:     n ← n + 1
3:     for i = 1 to r do
4:         insert_2((C[i][h_i(x)], N[i][h_i(x)]), x)
```

Algorithm 6 要約データ構造 BP-CGT の要素の削除操作.

```
1: procedure delete(S=(C[1..r][1..c][0..m−1], N[1..r][1..c], n), x)
2:     n ← n − 1
3:     for i = 1 to r do
4:         delete_2((C[i][h_i(x)], N[i][h_i(x)]), x)
```

Algorithm 7 要約データ構造 BP-CGT の出力操作.

```
1: procedure output(S=(C[1..r][1..c][0..m−1], N[1..r][1..c], n))
2:     F̂ ← ∅
3:     for (i, j) = (1, 1) to (r, c) do
4:         x ← output_2((C[i][j], N[i][j]))
5:         for k = 1 to r do
6:             if N[k][h_k(x)] ≤ φn then
7:                 break
8:             F̂ ← F̂ ∪ {x}
9:     return F̂
```

の候補値 $x \in U$ の頻度を $f_x < (\phi - \varepsilon)n$ と仮定する．候補値 x が，第 i 回目の頻度検査をパスするとき，i 回目の試行で要素 x と衝突する要素の合計頻度は，$f_{\neq x} \geq \varepsilon n$ を満たす．任意の誤差上限 $\varepsilon < \phi$ に対し，$c \geq 2/\varepsilon$ と設定すると，期待値の線形性と Markov の不等式より，$\mathbb{E}[f_{\neq x}] \leq \varepsilon n/2$ と $\mathbb{P}[f_{\neq x} \geq \varepsilon n] \leq 1/2$ であり，要素 x がパスする確率は 1/2 以下である．独立した r 回の試行により，最終的に要素 $x \notin \hat{\mathcal{F}}$ を出力する確率は $(1/2)^r = \delta/k$ 以下である．以上により，確率 $1 - \delta$ 以上で任意の $x \in \hat{\mathcal{F}}$ に対して，$f_x \leq (\phi - \varepsilon)n$ を満たす.

5.2. 提案データ構造

本稿の近似頻出値問題を解く要約データ構造 BP-CGT の基本アイデアは，既存データ構造 CGT [7] の各サイズ $m = O(w)$ のカウンタ配列 $C[i][j][0..m-1]$ を第 3 節の並列カウンタ配列で置き換えることである．この置き換えにより，要素の追加と削除は，ただちに償却 $O(\log(k/\delta))$ 時間に高速化できる.

一方，近似過半数値問題の場合と同様に，CGT の近似頻出値出力は，単純な並列カウンタの置き換えでは高速化できない．また，近似過半数値問題の場合と異なり，カウンタ値の変更による解決は難しい．近似過半数値問題の場合，カウンタ配列と合計頻度は同時に更新されるため，$\lceil \phi n \rceil$ の変化をカウンタ配列に反映できた．これに対し，近似頻出値問題の場合，追加あるいは削除される要素 x に関連する r 個のカウンタ配列だけ更新されるため，$\lceil \phi n \rceil$ の変化を x に関連しない $r \times (c-1)$ 個のカウンタ配列に反映できない.

提案データ構造 BP-CGT の各操作を Algorithm 5–7 に示す．次の補題は，近似頻出値出力の候補計算を単に近似過半数値出力で実現した場合でも，CGT の出力に関する理論的に保証は変化しないことを示す：

補題 4 BP-CGT は，確率 $1 - \delta$ 以上で $\hat{\mathcal{F}}$ を返す.

証明. 初めに，BP-CGT の出力 $\hat{\mathcal{F}}$ は，確率 $1-\delta$ 以上で $\mathcal{F} \subseteq \hat{\mathcal{F}}$ を満たすこと示す：独立した各試行において，任意の頻出値 $x \in \mathcal{F}$ の頻度 f_x は $f_x > \phi n$ を満たし，x と衝突する要素の合計頻度 $f_{\neq x}$ は，Markov の不等式より確率 1/2 以下で $f_{\neq x} \geq \phi n$ を満たす．したがって，確率 1/2 未満で $f_x > N[i][h_i(x)]/2 = (f_x + f_{\neq x})/2$ であり，$\mathsf{output}_2(C[i][h_i(x)], N[i][h_i(x)])$ は正しく $x \in \mathsf{output}$ を返す.

次に，任意の出力値 $x \in \hat{\mathcal{F}}$ の頻度 f_x は，確率 $1 - \delta$ 以上で $f_x \geq (\phi - \varepsilon)n$ を満たすことを示す：BP-CGT は，頻度検査を変更しないため，CGT と同様の議論により示される． □

6. おわりに

本稿では，近似頻出値問題を効率良く解くための要約データ構造 BP-CGT を提案した．この要約データ構造は，集合 $U = [0, u)$ に含まれる全要素の頻度情報を $O(\log(k/\delta)\log(u)/\varepsilon)$ 領域で管理し，要素の追加と削除を償却 $O(\log(k/\delta))$ 時間でサポートし，任意の時点での近似頻出値出力を $O(\log^2(k/\delta)/\varepsilon)$ 時間でサポートする．ここで，近似頻出値出力の失敗確率は，$\delta \in (0, 1]$ である．この要約データ構造により，長さ n のデータストリーム上の近似頻出値問題は，$O((n\log(k/\delta)\log u + \log^2(k/\delta))/\varepsilon)$ 時間で解ける．提案データ構造は，同じ操作をサポートする Cormode と Muthukrishnan のデータ構造 CGT [7] と同等の計算領域と失敗確率を達成すると同時に，要素の追加と削除を $O(\log u)$ 倍に高速化し，近似頻出値出力を $O(\log(u)/\log(k/\delta))$ 倍に高速化している．このような高速化は，CGT に関する既存の時間と領域のトレードオフでは実現できなかった.

また，提案データ構造 BP-CGT を実現するために，$m = O(w)$ 個のカウンタ $C[0], \ldots, C[m-1]$ を効率良く管理するデータ構造である並列カウンタ配列を提案した．このデータ構造は，各 $i \in [0, m)$ に対して，カウンタの更新操作 $C[i] \leftarrow C[i] + \mathsf{bit}(x, i)$ と $C[i] \leftarrow C[i] - \mathsf{bit}(x, i)$ を償却定数時間でサポートし，カウンタの比較操作 $C[i] = 0$ を定数時間でサポートする．ここで，$x \in [0, 2^m)$ は，更新操作を行うカウンタを指定する m ビット整数である．また，比較操作の結果は，m ビット整数 $\sum_{i=0}^{m-1} [C[i] = 0] \cdot 2^i$ として計算される.

文　献

[1] R. A. Baeza-Yates, G. H. Gonnet: A new approach to text searching. *Commun. ACM*, 35(10), pp. 74—82, 1992.

[2] P. Bille, M. Thorup: Regular expression matching with multi-strings and intervals. In *Proc. 21st SODA*, pp. 1297–1308, 2010.

[3] R. S. Boyer, J. S. Moore: MJRTY: a fast majority vote algorithm. automated reasoning. *Essays in Honor of Woody Bledsoe*, pp. 105–118, 1991

[4] J. L. Carter, M. N. Wegman: Universal classes of hash functions. *J. Comput. Syst. Sci.*, 18(2), pp. 143–154, 1979.

[5] M. Charikar, K. C. Chen, M. Farach-Colton: Finding frequent items in data streams. *Theor. Comput. Sci.*, 312(1), pp. 3–15, 2004.

[6] G. Cormode, M. Hadjieleftheriou: Methods for finding frequent items in data streams. *VLDB J.*, 19(1), pp. 3–20,

2010.

[7] G. Cormode, S. Muthukrishnan: What's hot and what's not: tracking most frequent items dynamically. *ACM Trans. Database Syst.*, 30(1), pp 249–278, 2005.

[8] G. Cormode, S. Muthukrishnan: An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1), pp. 58–75, 2005.

[9] E. D. Demaine, A. López-Ortiz, J. I. Munro: Frequency estimation of internet packet streams with limited space. In *Proc. 10th ESA*, pp. 348–360, 2002.

[10] G. S. Frandsen, S. Skyum: Dynamic maintenance of majority information in constant time per update. *Inf. Process. Lett.*, 63(2), pp. 75–78, 1997.

[11] K. Fredriksson, S. Grabowski: Nested counters in bit-parallel string matching. In *Proc. 3rd LATA*, pp. 338–349, 2009

[12] S. Grabowski, K. Fredriksson: Bit-parallel string matching under Hamming distance in $O(n\lceil m/w \rceil)$ worst case time. *Inf. Process. Lett.*, 105(5), pp. 182–187, 2008.

[13] P. Indyk: Sketching via hashing: from heavy hitters to compressed sensing to sparse fourier transform. In *Proc. 32nd PODS*, pp. 87–90, 2013.

[14] R. M. Karp, S. Shenker, C. H. Papadimitriou: A simple algorithm for finding frequent elements in streams and bags. *ACM Trans. Database Syst.* , 28(1), pp. 51–55, 2003.

[15] A. Metwally, D. Agrawal, A. E. Abbadi: An integrated efficient solution for computing frequent and top-k elements in data streams. *ACM Trans. Database Syst.* , 31(3), pp. 1095–1133, 2006.

[16] J. Misra, D. Gries: Finding repeated elements. *Sci. Comput. Program*, 2(2), pp. 143–152, 1982.

[17] G. S. Manku, R. Motwani: Approximate frequency counts over data streams. In *Proc. 28th VLDB*, pp. 346–357, 2002.

[18] A. Pagh, R. Pagh, S. S. Rao: An optimal Bloom filter replacement. In *Proc. 16th SODA*, pp. 823–829, 2005.

[19] 有村博紀, 喜田拓也: データストリームのためのマイニング技術. 情報処理学会 情報処理, 46(1), pp. 4–11, 2005.