Vol.2016-OS-136 No.13
2016/3/1

縮退表現に基づくシーケンスパタン集合の圧縮

川島 英之[1,2,a)]　建部 修見[1,2,b)]

概要：センサデバイスと機械学習技術の発展に伴い，時系列イベントを複合して時系列状態認識が可能になりつつある．複数のイベントを効率的に複合するためにはシーケンス演算子が有効であることが知られている．センサデータと機械学習出力が内在する不確実性に対処するには，時系列イベントは全て偽陽性であるとみなし，成立しうる全ての複合イベントを構築し，それらを精査する方式が求められる．この方式を用いると爆発的に多数の複合イベントが生成される．シーケンス演算子は一般的にクリネ閉包と非クリネ閉包を提供する．クリネ閉包に関しては効率的手法が既知である一方，非クリネ閉包に関する効率的手法は筆者らの既存研究を除けば既知ではなく，その既存手法は最適ではない．そこで本論文では非クリネ閉包を対象にしてシーケンスパタンを効率的に圧縮する縮退表現を提案する．提案手法は本研究の評価環境において，汎用ライブラリ GZIP2 に対して時間効率で 8,210 倍，空間効率で 9,982 倍の改善を示す．

Kawashima Hideyuki[1,2,a)]　Tatebe Osamu[1,2,b)]

1. はじめに

近年のセンサデバイスと機械学習技術の急激な発展を鑑みれば，センサと機械学習を用いることで時系列状態認識が容易に行えるようになりつつある [4,5]．そのような研究の中には小規模な部分的イベントを検出したあと，それらを組み合わせることで意味粒度の大きなイベントを構築する手法がある．例えば人間の行動認識に関する研究 [4] ではキックを行う状態（これを kick 状態と表記する）を検出するために 4 つのイベントが使われることが述べられている．すなわち pose-A, pose-B, pose-C, pose-D を組み合わせることで kick 状態が検出される．以降ではこの状態のように複数の時系列イベント (例：<pose-A>) から構成される複合イベントを**シーケンスパタン** (例：<pose-A, pose-B, pose-C, pose-D>) と表記する．ノイズを含まないセンサデータは存在せず，100%の精度を示す機械学習システムは存在しない．従って本来的にセンサデータは誤差を含み，機械学習は誤結果を出力する可能性がある．それゆえ機械学習が出力する各イベントは正しいとはあまり考えられない．例えば p1 という結果が出力された場合，それは p1 かもしれないがそうでないかもしれない．従って全てのイベントは偽陽性 (false positive) である可能性がある．そのようなイベントからシーケンスパタンを構築する際，成立しうる全てのシーケンスパタンを調査する方式が考えられる．例えば kick パタンを pose-A1, pose-A2, pose-B3, pose-C4, pose-D5 という入力シーケンスから検出することを考える．二つの pose-A を区別するために終わりに時刻印を付与してある．このとき kick パタンを検出するためには “pose-A1, pose-B3, pose-C4, pose-D5” と “pose-A2, pose-B3, pose-C4, pose-D5” が共に出力される必要がある．

このようにイベントの前後関係からシーケンスパタンを出力することは本論文の目的とは別の目的で研究が行われてきた．それらの研究ではこの処理を行う演算子をシーケンス演算子と呼称している [1,3,6,7,9–11]．シーケンス演算子は入力としてイベントシーケンスを取る．例えばシーケンス演算子 SEQ(A,B,C) は，イベント A の後にイベント B が生起し，その後にイベント C が生起した場合を検出する．検出後，それらのシーケンスパタンが出力される．シーケンス演算子の挙動解析に集中するために，本論文では入力シーケンスを簡略的に表現することにし，アルファベットと時間の組で記述することとする．例えば “c5” をイベント C が時刻 5 に存在することを表すことにする．

さて，いま図 1 に示されるような問合せを考える．この問合せは [10,11] に合わせて記載してある．PATTERN 句に記載されている SEQ は検出したい時系列イベントパタ

1 筑波大学 システム情報系
2 国立研究開発法人 JST CREST
a) kawasima@cs.tsukuba.ac.jp
b) tatebe@cs.tsukuba.ac.jp

```
PATTERN SEQ(A a, B b, C c, D d)
FROM sequence
WHERE a.conf > 0.8 AND b.conf > 0.6 AND
      c.conf > 0.4 AND d.conf > 0.3 AND
RETURN abcd
```

図 1 Q1: シーケンスパタンの列挙

ンを示す．A,B,C はイベントのクラスを表し a,b,c はイベントのインスタンスを表す．FROM 句は入力シーケンスイベントの情報源を表す．これはリレーショナルデータベースにおけるそれと同様の意味を表す．各イベントには信頼度情報が付与されており，それが閾値を越えたものだけが処理されることが WHERE 句に示されている．この信頼度は様々な方法で与えることができる．仮に機械学習モジュールが分類器であり，その実装が SVM であるならば，この信頼度は超平面からの距離に基づいて算出しうる．他にも多数の手法が考えられようが，いずれにしても閾値を与える必要がある．最後に，RETURN 句は問合せパタンに適合したシーケンスパタンの返却を示す．

Q1 に示される SEQ は**シーケンス演算子**と呼ばれる演算子として実現される．シーケンス演算子は主としてストリームデータ処理の文脈で研究が行われてきた．その応用例としては RFID データストリームを利用したリアルタイム万引き検出 [10] や株価ストリームにおける特定パタン（例：三尊/逆三尊）のリアルタイム検出 [1] がある．シーケンス演算子自体の効率的な処理技法としてはオートマトンを用いるアプローチ [10,11] とリレーショナル結合を用いるアプローチ [6] の 2 種類が存在する．並列化による高性能化技法 [1] も存在する．XML ストリームに処理に関する高速化技法 [7] も存在する．不確実ストリームに対する処理技法も存在する [3]．既存のシーケンス演算子 [6,10,11] に関する研究では，RETURN 句において何らかの集約を必須としている．RETURN 句において列挙されたシーケンスパタンの集合を返却する問題は筆者の従来研究 [3,8] を除けばこれまで扱われてこなかった．

シーケンス演算子に関する殆どの既存研究 [6,10,11] では **skip-till-next-match** と呼ばれるイベントスキップ方式を主眼としている．イベントスキップ方式とは，シーケンス演算子がイベントをスキップする方式である．シーケンス演算子を最初に提案した [10] では 3 つのイベントスキップ方式（pattern-contiguity, skip-till-next-match, skip-till-any-match）が示されている．その中で skip-till-next-match はノイズ，即ちシーケンス演算子の対象外であるイベントを無視する方式だと述べられている．例えば問合せが SEQ(A,B,C) であり入力イベントシーケンスが (a1 e2 b3 a4 f5 g6 b7 c8) である場合，この方式は途中のノイズを無視して (a1 b3 c8) を出力する．

本研究の対象であるセンサデータの機械学習による状態認識というタスクにおいては skip-till-next-match 方式の適用は不適切である．なぜなら全てのイベントが偽陽性でありえるからである．本研究の対象においては前述の例において出力されるべきは (a1 b3 c8), (a1 b7 c8), (a4 b7 c8) の 3 つのシーケンスパタンである．そのような出力を得るイベントスキップ方式は学術的には **skip-till-any-match** と呼ばれる [11]．実用的システムとしては ESPER [2] と呼ばれるストリーム処理システムにおいて everyA→every B というイベントスキップ方式が同じ意味を有する．この方式は多数のシーケンスパタン (n イベントに対して $\sum_{i=1}^{n} {}_nC_i$) を出力するために処理時間とメモリサイズのいずれもが高価になる．データベースシステムとしてこの問合せを扱うことを考えるとき，処理時間もさることながらメモリサイズが問題となる．

本研究ではイベントスキップ方式を skip-till-any-match にした場合におけるシーケンス演算子の処理について，利用する空間サイズを削減することに挑戦する．問合せパタンが SEQ(A,B+,C) であり B+のみに skip-till-any-match を用いる場合の縮退表現は既に提案されている [11]．その一方，本研究で対象とする問合せパタン SEQ(A,B,C) に関する処理は一部の既存研究において扱われてきた [3,8,11]．しかしながらこれらはいずれも処理時間の削減のみに焦点を当てており，空間サイズの削減は扱っていない．空間サイズの削減を扱う研究は筆者が知る限り筆者らの先行研究 [12] が初めての試みであり，本論文ではそれよりも優れた方式を示す．

本論文で我々はシーケンスパタン集合を縮退表現する技法を提案する．縮退表現により出力されるシーケンスパタン集合の空間コストは汎用圧縮ライブラリに比べて削減される．さらにその縮退化に要する処理時間は汎用圧縮ライブラリに比べて高速である．本論文は縮減化を提案するにとどまらず，縮減表現を展開してシーケンスパタン集合を生成する技法も提案する．縮退表現による利点を述べる．まず，膨大量のシーケンスパタン集合によりストレージ容量が圧迫される現象を縮退表現は回避可能にする．次に，縮退表現はシーケンスパタン集合生成処理の並列化を容易にする．なぜなら縮退化されたシーケンスパタン集合は容量が小さいため，それを他ノードへ移送することは容易になるからである．

本論文の構成は以下の通りである．2 節では研究背景を述べる．3 節では提案手法を述べる．4 節では評価を述べる．最後に 5 節では本論文をまとめる．

2. 研究背景

2.1 シーケンス演算子

人間は時間の流れと共に生きる．そのため，あるイベン

トの後に生起したイベントといった，イベントのシーケンスに価値を見出すことが多い．それゆえそのようなシーケンスパタンを検出する要求は多く存在する．具体的な例としては株価分析 [1] や人間行動認識 [4,5] が挙げられる．株価分析 [1] では三尊のような時系列パタンをシーケンス演算子を用いて検出する例が示されている．人間行動認識 [4,5] ではシーケンス演算子は使われていないが，時系列信号データからシンボルのシーケンスを検出する試みが述べられている．

このような時間的依存性のある複数のイベントから一連のシーケンスを効率的に取り出すためにシーケンス演算子が考案された [10]．この処理はリレーショナルデータベースにおける自己結合を用いれば実現可能だが，結合演算は処理コストが高いためにその利用は避けることが好ましい．その代わりにシーケンスパタンを検出するロジックを直接的に演算子としてシーケンス演算子がある．これは結合演算に比べて効率的であることが知られている [10]．シーケンス演算子は出力がシーケンス，すなわち順序付集合となるため，順序無集合を扱うリレーショナルモデルとは不適合であり，CEP エンジン等と呼称されて別システムとして扱われてきた．また，このような検出はリアルタイム性を伴う場合が多いと考えられておりシーケンス演算子はデータストリーム処理の文脈でウィンドウ演算と組み合わせて研究がされてきた [1,3,6–11]．本論文でデータストリームに限定せずにシーケンス演算を考えるためウィンドウ演算は導入しない．

2.2 課題

本研究ではイベントスキップ方式を skip-till-any-match にした場合におけるシーケンス演算子の処理について，利用する空間サイズを削減することに挑戦する．空間サイズの削減には利点がある．まず，膨大量のシーケンスパタン集合によるストレージ容量の圧迫を回避できる．次に，処理の分散並列化を容易にする．なぜならデータの容量が小さければ，そのデータを他ノードへ移送するコストは低減するからである．

問合せパタンが SEQ(A,B+,C) であり B+のみに skip-till-any-match を用いる場合の縮退表現は既に提案されている [11]．その一方，本研究で対象とする問合せパタン SEQ(A,B,C) に関する処理は一部の既存研究において扱われてきた [3,8,11]．しかしながらこれらはいずれも処理時間の削減のみに焦点を当てており，空間サイズの削減は扱っていない．空間サイズの削減を扱う研究は筆者が知る限り，筆者の先行論文 [12] 以外には存在しない．

SASE++ [11] は skip-till-any-match の効率化に取り組んでいるが非クリネ閉包に関する処理を扱っていない．前述の機械学習を用いた状態認識などの応用を考えると，非クリネ閉包に対する skip-till-any-match の適用が求められる．本論文では ski-till-any-match により生成される膨大量のシーケンスパタンを高速かつ効率的に圧縮する技法を提案する．

3. 提案

3.1 縮退表現

本研究では検出されるシーケンスパタンを高速に圧縮するために縮退表現に基づく方式を導入する．この縮退表現は SASE++ [11] ではクリネ閉包を対象として提案されているが，その手法は非クリネ閉包に対する手法に比べて実に単純であり，まるで異なる手法である．すなわち非クリネ閉包に対する縮退表現は，我々の先行研究 [12] を除けば他に存在しない．

提案手法はまず入力シーケンスをシーケンシャルスキャンして NFA(Non Deterministic Automata) の各状態に分配する．分配後，提案手法は受理状態に関するイベントから開始状態イベントへと深さ優先探索をしながら縮退表現を生成する．提案手法について述べる前に，その基礎となり単純な技法である SASE の方式を説明する．例えば入力シーケンスが図 2 である場合を考える．この時，SASE [10,11] ではまず AIS(Active Instance Stack) と呼ばれるデータ構造を構築する．これは NFA の下部にイベントを格納するスタックである．この場合 AIS の内容は図 3 となる．下方への遷移を行う度に遷移先のイベントが結果出力用スタックにプッシュされ，上方への遷移を行うとイベントがスタックからポップされる．このイベントアクセスに A へ遷移したあと，B へ戻る前にシーケンスパタンを 1 つ出力する．総計 28 件のシーケンスパタンがこの場合出力され，それは図 4 に示されている．SASE の方式を algorithm 1 に示す．

上述した SASE の方式では膨大量の結果が生成される．そこで本研究では縮退表現を用いて圧縮を行うことを考える．上記と同じ入力から生成した縮退表現を図 5 に示す．ここでは 13 個のイベントが示されている．SASE が生成する非縮退表現におけるイベントは 84 個であるため，必要な空間サイズがおよそ 6.5 分の 1 に削減されていることがわかる．縮退表現を構築するアルゴリズム compress を algorithm 2 に示す．このアルゴリズムは再帰的に動作する．29 行目で concise 自身を呼び出している．状況を受理状態，開始状態，それ以外の状態，の 3 種類に分類する．それぞれ入力イベントを結果用スタック (result-stack)) にプッシュするが，そこで適切な処理ができない場合には (例：その先で新しいイベントをプッシュできなかった等) 自分をポップアップする．再帰処理から戻った後でも状態が不適切ならば自分をポップアップする．

3.2 展開

シーケンスパタンの内容を分析するためには，前述の縮

b0 b1 a2 b3 c4 b5 b6 a7 a8 b9 c10
b11 c12 b13 c14 b15 a16 a17 b18 b19

図 2 入力シーケンス例

Depth	A	B	C
1	a17	b19	c14
2	a16	b18	c12
3	a8	b15	c10
4	a7	b13	c4
5	a2	b11	
6		b9	
7		b6	
8		b5	
9		b3	
10		b1	
11		b0	

図 3 SEQ(A,B,C) に対応する AIS

(a2 b3 c4)	(a2 b3 c10)	(a2 b5 c10)
(a2 b6 c10)	(a2 b9 c10)	(a7 b9 c10)
(a8 b9 c10)	(a2 b3 c12)	(a2 b5 c12)
(a2 b6 c12)	(a2 b9 c12)	(a7 b9 c12)
(a8 b9 c12)	(a2 b11 c12)	(a7 b11 c12)
(a8 b11 c12)	(a2 b3 c14)	(a2 b5 c14)
(a2 b6 c14)	(a2 b9 c14)	(a7 b9 c14)
(a8 b9 c14)	(a2 b11 c14)	(a7 b11 c14)
(a8 b11 c14)	(a2 b13 c14)	(a7 b13 c14)
(a8 b13 c14)		

図 4 シーケンスパタンの集合

b13 c14 b11 c12 a8 a7 b9 b6 b5 c10 a2 b3 c4

図 5 縮退表現

退表現は展開される必要がある．縮退表現を展開するアルゴリズム expand を algorithm 3 に示す．まず，N 個のイベントを result-stack から取り出す．N はイベント種類の数である．例えば SEQ(A,B,C) ならば N は 3 となる（1 行目）．次に展開作業を繰り返し的に実行する．まずはコピー開始地点を発見する．コピー開始地点は 3 行目の event と同一タイプのイベントであり，かつその前に生起したイベントを含む全てのシーケンスの最初である．次にコピー開始地点からコピー終了地点までをまとめてコピーする．最後にコピーした範囲で時刻印の修正を行う．

4. 評価

本節では提案手法を実験的に評価した結果を示す．実験に用いた計算機の仕様は次の通りである：CPU:Intel Core i5 3.2GHz, Memory 16GB．データは一様乱数に従い生成した．プログラムは C++言語により作成した．

Algorithm 1 SASE の方式：genSeq(current-ais)

```
1: if current-ais is for starting state then
2:     generate a result-sequence by temporal-stack
3:     return
4: end if
5: for event in current-ais do
6:     push an event to temporal-stack
7:     invoke genSeq(next-ais, event)
8:     pop an event from temporal-stack
9: end for
```

Algorithm 2 提案する圧縮方式: compress(current-ais, previous-event)

```
 1: if current-ais is φ then
 2:     return;
 3: end if
 4: for current-event ∈ current-ais do
 5:     if time of current-event > time of previous-event then
 6:         break;
 7:     end if
 8:     Push current-event to result-stack
 9:     if current ais is for acceptance state then
10:         do nothing
11:     else if current ais is for starting state then
12:         if MaxTime for current ais is not initialized then
13:             update maxTime(current-event);
14:         else if time of current-event <= MaxTime for
            current-ais then
15:             Pop an event from result-stack
16:             continue this loop
17:         else
18:             update maxTime(current-event);
19:         end if
20:     else
21:         if MaxTime for current ais is not initialized then
22:             update maxTime(current-event);
23:         else if current-event is not the latest one then
24:             Pop an event from result-stack
25:             continue this loop
26:         else
27:             update maxTime(current-event);
28:         end if
29:         invoke compress(next ais, current-event);
30:     end if
31:     if current-ais is an internal one or current ais is for ac-
        ceptance state then
32:         if top of result-stack is current-event then
33:             if no result-sequence is generated so far then
34:                 Pop an event from result-stack
35:                 continue this loop
36:             end if
37:         end if
38:     end if
39: end for
```

いずれの実験でも入力イベント数の最大数を 4,000 とした．この理由はイベント数が 5000 の場合に急激な性能劣化が見られたからである．96〜100%を示していた CPU 利用率が 0.1%前後になったことから仮想記憶が使われた可能性がある．いずれにしても本節で行う評価は出力される

Algorithm 3 提案する展開方式: expand

```
1: Pop N events (N is the number of sequence types specified
   by a query) and construct a sequence
2: Add the sequence to result-list
3: for event ∈ result-stack (from bottom to top) do
4:     Find the start point of copy
5:     copy them and rewrite the timestamp to that of current-
       event
6: end for
```

シーケンスパタン集合がメモリに乗る場合を対象とする．

4.1 処理時間

クエリが SEQ(A,B,C) である時の処理時間に関する実験結果を 6 に示す．“SASE” は従来手法を表す．これは受理状態のイベントから深さ優先探索を初期状態へ向けて行い，初期状態へ到達した時点で結果を出力する．“Proposed(Compress)” は縮退表現を出力する提案手法を表す．“Proposed(Compress+Expand)” は縮退化した後にそれを展開した場合を表す．“BZIP2 Compression” は bzip2 コマンドにより全出力結果を圧縮した場合を表す．図の縦軸は対数軸での処理時間を表す．従って値が小さいほど好ましい．

出力結果の整合性について考えると，提案手法である Proposed(Compress+Expand) と SASE は同一結果を出力する．一方性能について考えると，SASE は Proposed(Compress+Expand) よりも高い性能を示しており，イベント数が 4,000 の場合に約 23%の性能向上を示している．

Proposed(Compress) は縮約表現を出力するため，SASE よりも少ない容量の結果を生成する．一方，その性能は SASE よりも高い．イベント数が 4,000 の場合に，Proposed(Compress) は SASE よりも 843 倍程度高速である．

データを縮約表現する一般的な技法に圧縮がある．著名な圧縮ライブラリである GZIP2 による圧縮処理時間も同図に示されている．GZIP2 は図に示されるグラフの中で最も遅い．イベント数が 4,000 の場合における GZIP2 の性能劣化率は, Proposed(Compress+Expand) に比べて 9.6 倍, SASE に比べて 11.9 倍，Proposed(Compress) に比べて 9,982 倍である．GZIP2 の行うタスクが Proposed(Compress) の行うタスクと同一であることを考えると，9,982 倍の差は顕著だと言っても差し支えないように思われる．

4.2 圧縮率

提案手法による縮退表現はどの程度の情報圧縮を可能にするのか本節で述べる．クエリイベント数が 3 の場合の結果を図 7 に示す．縦軸は出力結果のサイズ（バイト単位）を対数で表し，横軸は入力イベントの数を表す．この図には 3 つのグラフが示されている．Proposed(Compress) は提案手法により縮約を行ったシーケンスパタン集合のサイズを表す．SASE は既存手法である SASE が出力したシーケンスパタン集合のサイズを表す．GZIP2 は SASE の出力を GZIP2 により圧縮した結果のサイズを表す．この図に Proposed(Compress+Expand) が示されていない理由は，その結果は SASE と完全に一致するからである．

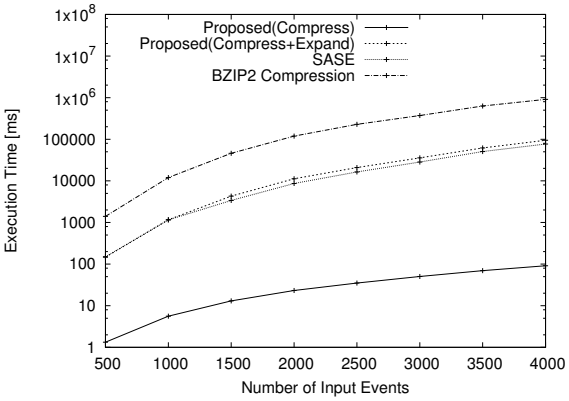

図 6 処理時間：SEQ(A,B,C)

この図からは, Proposed(Compress) が最良の結果を示す一方，SASE が最悪の結果を示していることがわかる．両者の中間に GZIP2 が存在している．Proposed(Compress) と SASE について最大の差がついたのは入力シーケンスパタン数が 4,000 の場合である．この時の差は 295,211 倍である．即ち Proposed(Compress) は入力データのサイズをロスレスで 295,211 分の 1 に削減できたと言える．汎用圧縮ライブラリである GZIP2 も圧縮を実現しているが，その性能は最良の場合に高々 37 分の 1 程度である．圧縮率の差を図 8 に示す．この図においてイベント数が 4,000 の時，Proposed(Compress) は GZIP2 に比べて 8,210 倍程度の圧縮率を示している．以上より，提案手法は汎用圧縮ライブラリよりも本データセットにおいては圧縮率が高いと言えよう．

4.3 評価のまとめ

上述した評価の結果をここでまとめる．

(1) GZIP2 に対する提案手法の評価 Proposed(Compress) は GZIP2 に比べて最大で 8,210 倍程度の空間の削減を示す．Proposed(Compress) は GZIP2 に比べて最大で 9,982 倍程度の処理時間の削減を示す．以上より提案手法は汎用ライブラリを凌駕すると言える．

(2) SASE に対する提案手法の評価 Proposed(Compress) は SASE に比べて最大で 295,211 倍の空間効率性を示す．一方，Proposed(Compress+Expand) は SASE に比べて最大で 23%の性能劣化を示す．以上より提案手法は SASE に比べて処理時間を若干犠牲にしながら顕著な空間削減を実現すると言える．

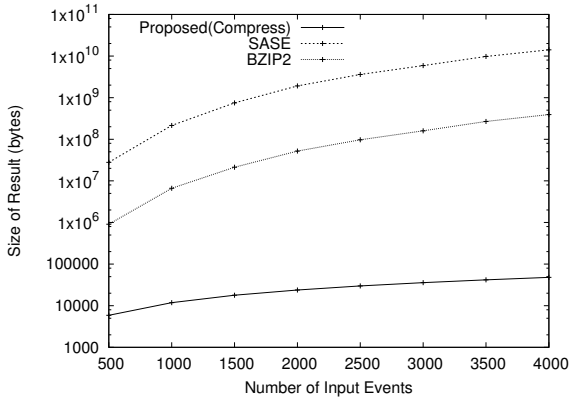

図 7 メモリサイズ：SEQ(A,B,C)

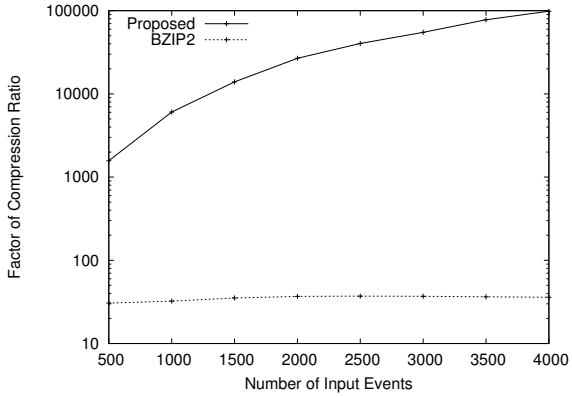

図 8 圧縮率：SEQ(A,B,C)

5. まとめ

5.1 結論

本論文では skip-till-any-match におけるシーケンス演算子処理を効率化するために縮退表現を提案した．我々は文献 [12] において本論文で提案した技法の前身である萌芽的技法を提案した．本論文の成果はそれを発展させたものである．縮退表現についてはこれ以上の空間効率化は難しいと考える．

提案手法をプログラムで実装し，計算機上で SASE ならびに GZIP と性能・空間効率について比較実験を行った．その結果，次の実験結果を得た．Proposed(Compress) は GZIP2 に比べて最大で 8210 倍程度の空間の削減を示した．Proposed(Compress) は GZIP2 に比べて最大で 9982 倍程度の処理時間の削減を示した．Proposed(Compress) は SASE に比べて最大で 98304 倍の空間効率性を示した．一方，Proposed(Compress) は SASE に比べて最大で 23

以上より本研究で扱ったデータセットと実験環境においては，提案手法は汎用ライブラリ GZIP2 を処理時間と圧縮効率で凌駕すると同時に，提案手法は SASE に比べて処理時間を若干犠牲にしながら顕著な空間削減を実現すると結論する．

5.2 今後の課題

今後の課題は，展開処理の高速化とストレージアクセスを要する膨大量のシーケンスパタン集合への対処である．

謝辞 本研究の一部は，JST CREST「ポストペタスケールデータインテンシブサイエンスのためのシステムソフトウェア」，JST CREST「EBD：次世代の年ヨッタバイト処理に向けたエクストリームビッグデータの基盤技術」，JST CREST「広域撮像探査観測のビッグデータ分析による統計計算宇宙物理学」，科研費「#25280043HA」，JST A-STEP「#AS262Z02895H」による．

参考文献

[1] Balkesen, C., Dindar, N., Wetter, M. and Tatbul, N.: RIP: Run-based Intra-query Parallelism for Scalable Complex Event Processing, DEBS (2013).
[2] ESPER Reference: http://www.espertech.com.
[3] Kawashima, H., Kitagawa, H. and Li, X.: Complex Event Processing over Uncertain Data Streams, 3PGCIC (2010).
[4] Kim Eunju and Sumi Helal and Diane Cook: Human Activity Recognition and Pattern Discovery, IEEE pervasive computing (2010).
[5] Matsubara, Y., Sakurai, Y. and Faloutsos, C.: AutoPlait: automatic mining of co-evolving time sequences, SIGMOD Conf. (2014).
[6] Mei, Y. and Madden, S.: ZStream: a cost-based query processor for adaptively detecting composite events, SIGMOD Conf. (2009).
[7] Mozafari, B., Zeng, K. and Zaniolo, C.: High-Performance Complex Event Processing over XML Streams, SIGMOD Conf. (2012).
[8] Nishimura, N., Kawashima, H. and Kitagawa, H.: A High Throughput Complex Event Detection Technique with Bulk Evaluation, 3PGCIC (2013).
[9] Oge, Y., Miyoshi, T., Kawashima, H. and Yoshinaga, T.: A fast handshake join implementation on FPGA with adaptive merging network, SSDBM (2013).
[10] Wu, E., Diao, Y. and Rizvi, S.: High-Performance Complex Event Processing over Streams, SIGMOD Conf. (2006).
[11] Zhang, H., Diao, Y. and Immerman, N.: On Complexity and Optimization of Expensive Queries in Complex Event Processing, SIGMOD Conf. (2014).
[12] 川島英之，建部修見：縮退表現によるシーケンス演算処理の効率化，情報処理学会研究報告システムソフトウェアとオペレーティング・システム，Vol. 2015-OS-134, No. 9, pp. 1–7 (2015).