Vol.2015-OS-134 No.9
2015/8/4

縮退表現に基づくシーケンス演算処理の効率化

川島 英之[1,2,a)]　建部 修見[1,2,b)]

概要：時系列イベントを複合して状態認識を行う場合，複数のイベントを効率的に複合するためにシーケンス演算子が有効である．このとき一つの厳密な態度として，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考えられる．そのような態度は skip-till-any-match 方式と呼ばれる．この方式を用いると多数の複合イベントが生成される．シーケンス演算子は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既知ではない．本論文では skip-till-any-match 方式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する．実験により縮退表現は従来手法である SASE に比べて処理時間とメモリサイズを効率化でき，その程度は最大で数千倍であることを実験的に示す．

1. はじめに

センサと機械学習を用いることで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イベントを抽出し，それらを組み合わせることで状態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 kick 状態を検出するために 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 pose1, pose2, pose3, pose4 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から構成される複合イベントを**シーケンスパタン**と表記する．必然的にセンサは誤差を含み，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限らない．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，一つの厳密な態度として，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．

イベントの前後関係からシーケンスパタンを検出する手段としてシーケンス演算子が研究されてきた．シーケンス演算子 SEQ(A,B,C) は入力としてイベントシーケンスを取る．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 が生起した場合にそれらのシーケンスパタンを出力する．本論文では入力シーケンスをアルファベットと時間の組で記述する．例えば “c5” はイベント C が時刻 5 に存在することを表す．図 1 に示されるような問合せを

```
PATTERN SEQ(A a, B b, C c)
FROM sequence
WHERE a.conf = var1 AND
      b.conf = var2 AND
      c.conf = var3
RETURN abc
```

図 1　Q1: シーケンスパタンの列挙

考える．この問合せは [11, 12] に合わせて記載している．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ンを示す．A,B,C はクラスを表し a,b,c はインスタンスを表す．FROM 句は入力シーケンスデータを表す．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．信頼度は様々な方法で与えられるだろうが仮に分類器が SVM 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．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．

シーケンスそのものを求めるよりも，その総数を数え上げることが求められる場合がある．例えば全数検索を行うと長時間を要する場合，事前にシーケンス数の数え上げを行うことがある．そのような場合は得られたシーケンスパタンを得たあとに集約処理を行えば良い．すなわち Q2(図 2) のような問合せが記述される．RETURN 句に示される COUNT がシーケンスパタン数のカウント処理を行うことを表している．

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)
FROM sequence
RETURN COUNT(*)
```

図 2 Q2: シーケンスパタン数のカウント

Q1 と Q2 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 [11] 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 [11,12] 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．

シーケンス演算子に関する研究では **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 [12] 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 [12]．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 every A→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い．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，処理時間とメモリサイズを削減することに挑戦する．我々の問題解決アプローチはイベントシーケンスを縮退表現することである．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 [12]．その一方，本研究で対象とする問合せパタン SEQ(A,B,C) に関する縮退表現はまだ扱われておらず，縮退表現の展開手法も未知である．

全てのイベントシーケンスを列挙するアプローチを採用すると，その列挙件数は指数関数的に増加する．仮に列挙アルゴリズムを embarassingly parallel にすることができ，かつ多数の計算資源が使えたとしても，指数関数的に増加する問題への効果は限定的となる．従って結果件数の増加を抑制する方式には少なからぬ効果があると我々は考える．

我々の提案する縮退表現は複数のシーケンスパタン中に共通するサブシーケンスパタンが存在する点を利用する．例えば前述の (a1 b3 c7), (a1 b7 c8), (a4 b7 c8) において a1 b7 c8 は二つのシーケンスパタンに共通して存在する．これらをまとめて表現することで出力するシーケンスパタンの数を減らす．これにより処理時間とメモリサイズを同時に削減する．

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．5 節では研究内容について議論する．最後に 6 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

あるイベントの後に生起したイベントを検出する要求は多く存在する．その例には株価分析 [1] や人間行動認識 [4,5] がある．このように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 [11]．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ても実現可能だが，結合演算は処理コストが高い．一方，処理ロジックをそのまま実現するシーケンス演算子は結合演算に比べて効率的である．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 [1,3,6–9,11,12]．本論文でデータストリームに限定せずシーケンス演算考えるためウィンドウ演算は導入しない．

2.2 関連研究

本研究ではシーケンス演算子を処理する際のイベントスキップ方式として skip-till-any-match を取り上げる．これに関連する研究をイベントスキップ方式に応じて説明す

る．イベントスキップ方式には 3 種類のものがある．最も厳しい方式は strict contiguity と呼ばれる．これは隣接するイベントのみを照合対象とする．この応用例には株価分析がある [1]．テクニカル分析と呼ばれる株価分析では株価が変動パタンが既知のパタン（例：三尊）に正確に適合することを求めるため，このイベントスキップ方式が有用だと考えられる．例えば SEQ(A,B,C) という問合せに対して a1 e2 b3 a4 f5 g6 b7 c8 が入力された場合，結果は得られない．

これを緩和したイベントスキップ方式として skip-till-next-match がある．これは照合イベントが到着するまで他のイベントを無視する方式である．前述の例における出力は (a1 b3 c8) である．SASE [11, 12] はシーケンス演算子を提案すると共に上記のイベントスキップ方針を提案した．また，NFA を用いたシーケンスパタン構築手法である AIS(Active Instance Stack) を提案し，それが単純なシーケンス構築法である SSC(Sequence Scan and Construction) やウィンドウリレーショナル結合に比べて効率的であることを示した．SASE [11] 同様に skip-till-next-match 方針をターゲットとしながらも NFA を使わずに効率化を試みた研究に ZStream [6] がある．

イベントスキップ方針として skip-till-next-match を更に緩和させる方針として skip-till-any-match がある．これは全てがノイズでありえるような場合に有効になる．入力イベントの照合を非決定的に行うことで複数の状態を同時に保持する方式である．前述の例における出力は (a1 b3 c8), (a1 b7 c8), (a4 b7 c8) である．著者が知る限り skip-till-any-match の効率化に取り組んだ研究は SASE++ [12] のみである．ただし SASE++においてはクリネ閉包すなわち SEQ(A, B+, C) というパタンの B+のみを対象として skip-till-any-match の効率化に取り組んでいる． SASE++は非クリネ閉包すなわち SEQ(A,B,C) 等の場合は対象外としている．例えば入力シーケンスが a1 b2 b3 b4 c5 である場合，出力結果は (a1 b2 b3 b4 c5), (a1 b2 b3 c5), (a1 b2 b4 c5), (a1 b3 b4 c5), (a1 b2 c5), (a1 b3 c5), (a1 b4 c5) の 7 シーケンスパタンとなるすなわち B+部分が展開された結果が出力される．この展開を省けば性能効率が向上することは自明であろう．そこで SASE++では (a1 <b2 b3 b4> c5) という結果を出力する方式を提案している．

SASE++は skip-till-any-match の効率化に取り組んでいるが，二つの問題が残されている．第一の問題は非クリネ閉包に関する処理である．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．第二の問題は効率的な展開手法である．展開により出力されるシーケンスパタンの数は SEQ$(E_1, \ldots, E_n)$ の場合に $\sum_{i=1}^{n} {}_nC_i$ である．従って展開処理は効率的であることが好ましい．

3. 提案

3.1 シーケンスパタン検出

本節では Q1 のような問合せ処理の効率化を考える．この処理はシーケンスパタン検出である．

3.1.1 縮退表現

本研究ではシーケンスパタン検出処理を効率化するために縮退表現を導入する．この縮退表現は SASE++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しては与えられていない．

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 の各状態に分配する．分配したあと，このアルゴリズム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行う．ただし探索中に葉ノードに到達した後，根に戻るまでの間に再度降下方へ移動する場合には，縮退表現を生成してその移動を行わない．縮退表現を用いることで出力結果のサイズは小さくなる．

例えば入力シーケンスが図 3 である場合を考える．この時，SASE [11,12] 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 4 となる．このとき深さ優先探索を行う SASE のイベントアクセス遷移は図 5 に示すようになる．この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．

一方，提案手法を用いる場合のイベントアクセス遷移は図 7 に示されるようになる．結果として得られるシーケンスパタンは図 8 に示す 15 件になる．ϕ と記載されている部分が縮退部である．SASE の場合よりも件数が少ないことがわかる．クリネ閉包の場合，縮退表現には ϕ などの記載は不要であり，単純に入力イベントをそのまま記載するだけでよい [12]．一方非クリネ閉包の場合には上記で見たように縮退表現は複雑になる．

縮退表現を構築するアルゴリズムを algorithm 1 に示す．まずは非縮退表現のシーケンスパタンを生成する（4 行目）．その後の探索において根に戻るまでの間に同レベルまで降りる場合があれば，降りる前に縮退表現を生成する（8 行目）．

3.1.2 展開

ユーザが結果を閲覧を求める場合には，前述の縮退表現は展開される必要がある．展開処理を効率的に実行するために，複数のシーケンスパタンを一括してコピーし，変更部分のみ修正する方式を示す．

ϕ は前方に存在するシーケンスパタンを用いて展開できる．縮退表現を展開するアルゴリズムを algorithm 2 に示

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

図 3 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 4 SEQ(A,B,C) に対応する AIS

c4→ b0 → b1 → b3 → a2 → c10 → b0 → b1 → b3 → a2 → b5 → a2 → b6 → a2 → b9 → a2 → a7 → a8 → c12 → b0 → b1 → b3 → a2 → b5 → a2 → b6 → a2 → b9 → a2 → a7 → a8 → b11 → a2 → a7 → a8 → c14 → b0 → b1 → b3 → a2 → b5 → a2 → b6 → a2 → b9 → a2 → a7 → a8 → b11 → a2 → a7 → a8 → b13 → a2 → a7 → a8

図 5 SASE のトラバース

す．まずはコピーを行う開始点 (watermark) を求める必要がある (6 行目)．それから現在のシーケンスの 1 つ手前までの全シーケンスが有する領域を全て足し合わせた領域を確保する (7 行目)．それからコピーを行う (9 行目)．コピー後に現在のシーケンスパタンのイベント情報をコピーする．これらのコピー処理により深さ優先探索処理と同等の結果を得る．

3.2 数え上げ

本節では Q2 に示されるような数え上げ処理への縮退表現を適用を提案する．数え上げは頻繁に使われる演算であるため，その効率化は重要である．SASE 同様に深さ優先探索を行う場合には探索が葉ノードに到着した時点でカウンタの値を 1 つ増やす．一方，縮退表現を用いる場合にはまず縮退表現を構築し，それから展開処理においてカウンタを足し合わせる．縮退表現構築アルゴリズムは同様であるので，展開処理アルゴリズムを algorithm 3 に示す．このアルゴリズムはシーケンスパタン検出とは異なり，空間確保を行わずに足し合わせ処理のみを実行する．

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(R) Xeon(R) CPU E5-2695 v2 @ 2.40GHz, RAM 66 GB.

(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 6 シーケンスパタンの集合

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

図 7 提案手法のトラバース

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

図 8 縮退表現

Algorithm 1 Generate Reduced Expression

```
1:  for Event ∈ AIS do
2:    Push down an event to stack for result;
3:    if Stack for result is full then
4:      Generate a sequence pattern scanning the stack for
        result;
5:    else
6:      if Direction is upward then
7:        if Already result is generated then
8:          Genenrate a reduced sequence pattern;
9:        end if
10:     end if
11:     Invoke myself;
12:   end if
13:   Pop up an event from stack for result;
14: end for
```

4.1 シーケンスパタン検出 (Q1)

本節では Q1 を対象として行った評価実験の結果を述べる．4.1.1 節と 4.1.2 節では入力パタンの生成分布は一様分布に従う．4.1.3 節では生成分布に偏りを与えた場合の結果を示す．

Algorithm 2 Expand Reduced Expression

```
 1: for s ∈ sequence patterns do
 2:   if (s ≠ reduced one) then
 3:     Allocate integrated result space for a single sequence;
 4:     Copy original result to integrated result;
 5:   else
 6:     Find the watermark (start sequence for copy);
 7:     Allocate integrated area for sequences for copy;
 8:     for c ∈ sequences for copy do
 9:       Copy from c to integrated area;
10:     end for
11:     for c ∈ sequences for copy do
12:       Replace event in integrated area;
13:     end for
14:   end if
15: end for
```

Algorithm 3 Count using Reduced Expression

```
 1: for s ∈ sequence patterns do
 2:   if (s ≠ reduced one) then
 3:     Counter of this sequence ← 1;
 4:   else
 5:     Find the watermark (start sequence for copy);
 6:     for c ∈ sequences for copy do
 7:       Increment total counter by counter of c;
 8:     end for
 9:   end if
10: end for
```

4.1.1 処理時間

クエリが SEQ(A,B,C) である時の実験結果を 9 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．Reduced は提案手法を表す．この場合，縮退表現を出力する．いずれも縦軸は処理時間を表す．実験結果は提案手法である Reduced が SASE に比べて高速であることを示している．入力シーケンス長が 1000, 2000, 3000 の時，性能比はそれぞれ 178, 380, 593 倍である．シーケンスが長い程，性能差が大きくなる結果が得られた．縮退表現を構築した後に展開処理を行った結果が Reduced+Expand により示されている．これは Reduced よりもかなり低性能だが，それも SASE よりは高性能である．入力シーケンス長が 1000，2000，3000 の時，Reduced+Expand の SASE に対する性能向上は 5.9,5.9,6.1 倍である．

次にクエリを SEQ(A,B,C,D,E,F,G,H,I,J) とした場合の実験結果を図 10 に示す．前図とは入力イベント数が 10 分の 1 である点に注意されたい．この実験結果も提案手法である Reduced が SASE に比べて高速であることを示している．入力シーケンス長が 100, 200, 300 の時，性能比はそれぞれ 141, 499, 4062 倍である．シーケンスが長い程，性能差が大きくなる結果が得られた．

提案手法である Reduced が SASE に比べて大幅に効率的になった理由は枝刈りが有効だった可能性がある．今回

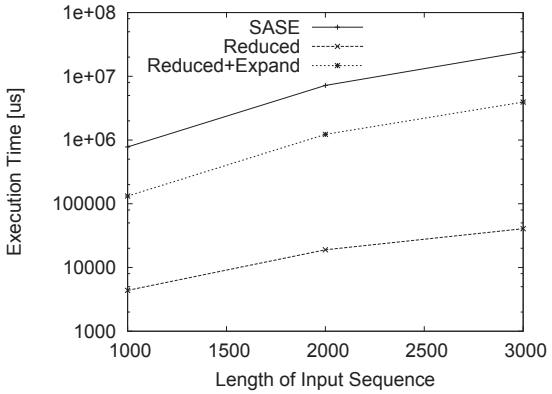

図 9 処理時間：SEQ(A,B,C)

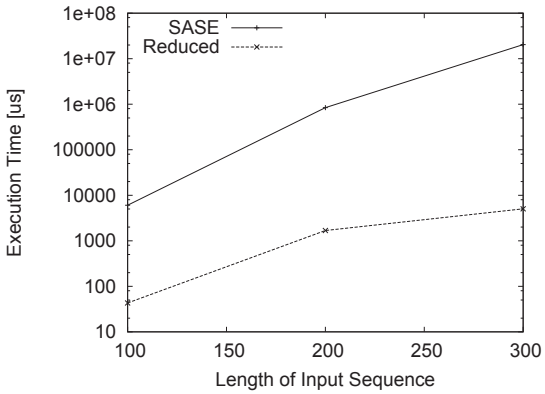

図 10 処理時間：SEQ(A,B,C,D,E,F,G,H,I,J)

の入力シーケンスにおいては全てのイベントの生起パタンを同一確率にしているが，枝刈りが殆ど効かない場合においては上述の性能差は縮まると考えられる．

4.1.2 シーケンスパタン数

出力されるシーケンスパタン数が多い程メモリを使用するため，仮想メモリ使用による性能劣化や OS によるプログラム停止を引き起こす可能性がある．データベースシステムとして考えた場合には他のクエリとも共有するバッファプールを多く使うことはシステム全体の性能劣化を引き起こす．従って出力されるシーケンスパタン数は少ない程好ましい．ここでは従来手法 (SASE) と提案手法 (Reduced) によるシーケンスパタン数を測定する．

クエリイベント数が 3 の場合の結果を図 11 に，クエリイベント数が 10 の場合の結果を図 12 に示す．いずれも最大の差がついたのは入力シーケンス数が最大の場合であり，前者の場合は 656 倍，後者の場合は 3786 倍だった．いずれの場合も 1 つのシーケンスパタンのメモリサイズは等しいため，縮退表現を用いることでメモリサイズをそれなりに削減できたと言える．

処理時間削減率はメモリサイズ削減率よりも高い．例えばメモリサイズ削減率が 3786 倍だった場合の処理時間の削減率は 4062 倍だった．この差分は探索コストの差である可能性がある．提案手法は縮退シーケンスパタンを作成

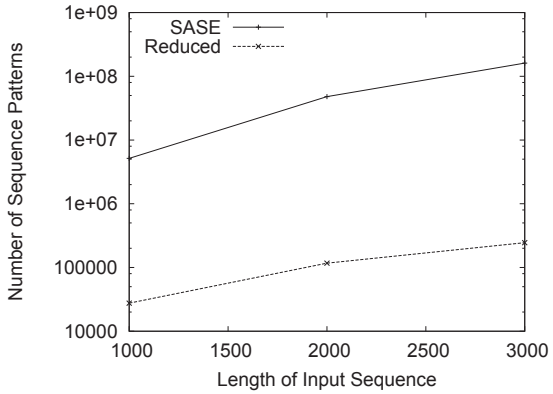

図 11 メモリサイズ：SEQ(A,B,C)

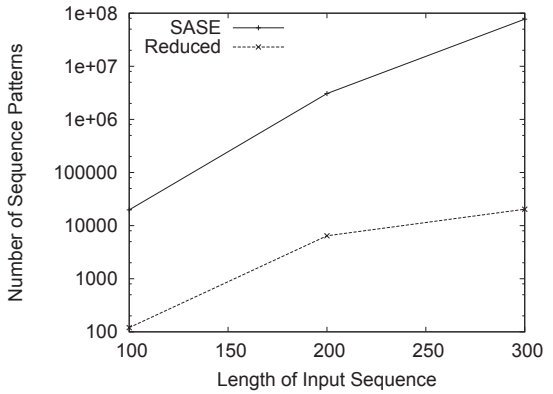

図 12 メモリサイズ：SEQ(A,B,C,D,E,F,G,H,I,J)

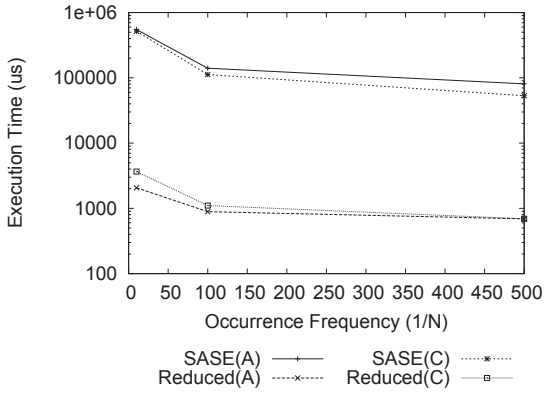

図 13 非一様分布における処理時間

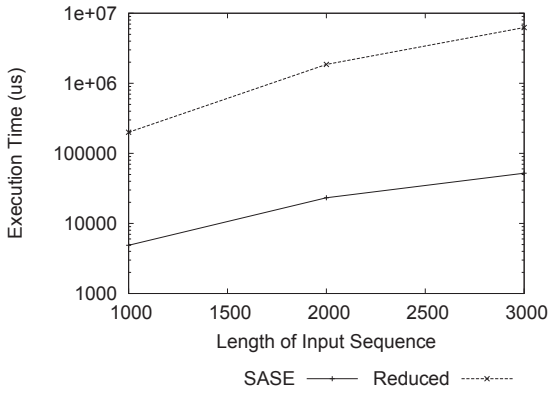

図 14 数え上げ

する際にはノード移動を省略する．この枝刈り処理が性能差に関連している可能性がある．

4.1.3 イベント生起確率分布

上記ではイベント生起確率分布を一様分布にした場合の結果を示した．イベント生起確率分布を変動させた場合の結果を図 13 に示す．SASE(A) は SASE を用いて A の生起確率を変動させたグラフを表す．例えば横軸の値が 10 の時，SASE(A) では 10 回に 1 回イベント A が生起し，残りの 9 回は B と C が等確率で生起する．他も同様である．入力イベント数はいずれも 1000 である．

実験結果より，A にしても C にしても発生頻度が下がる程（横軸が大きくなる程）処理時間が短くなることがわかる．また，SASE は C の発生頻度を下げると性能が向上し，提案手法は A の発生頻度を下げると性能が向上している．

4.2 数え上げ (Q2)

本節では Q2 を対象として行った評価実験の結果を図 14 に示す．SASE 方式では葉ノードに到達する度にカウンタを増やすアルゴリズムを用いた．提案手法は縮退表現をまず構築し，それから展開時にカウンタの値を計算するアルゴリズム（Algorithm 3）を用いた．実験結果は提案手法が SASE よりも効率的であることを示す．イベント数が 3000 の時，性能差は最大であり，その値は 120 倍である．

現在の提案手法は縮退表現を使ってから数え上げ処理に入るために無駄があると考えられるが，それでも従来手法よりも効率的であることは興味深い．枝刈りによる効果が大きいと考えられる．なお，ここで実装した従来手法はシーケンス演算子の後に集約演算子を呼び出す方式ではなく，シーケンス演算子内部の探索処理中に集約処理を行うよう最適化（演算子プッシュダウン）をしていることを述べておく．

5. 議論

5.1 Skip-till-next-match の応用

本論文ではセンサと機械学習を用いて状態認識を行うシステムを skip-till-next-match の主たる応用として取り上げた．このイベントスキップ方式の応用が現実に存在するのがここで議論を行う．既存研究では一応用として Hadoop のジョブ監視が挙げられている．そのため応用は全く存在しない訳ではない [12].

センサと機械学習を用いて状態認識を行うシステムについて議論する．既にパーベイシブコンピューティングの分野で多少の研究は存在する [4]．一方，データストリーム分野においてはストリームイベントの生起が不確実であるという前提で行われる研究が存在する [3, 10]．これは RFID などのセンシングデバイスの出力は不安定・不確実である

ためである．数年前まではストリームイベントに生起確率を与えることは現実的ではなかったが，近年の機械学習技術の向上により，それは現実的になってきている．計算の基礎として，SVM ならば超平面からの距離，k-NN ならばオブジェクト数を用いることができるだろう．一方，識別性能の低い機械学習がそもそも用いられる可能性があるかは不明である．検出要求は高い一方，求められる精度のセンサ・機械学習を用いることが困難であるような場合には，提案手法が使われる可能性があると考える．

5.2 シーケンス演算子とリレーショナル演算子の統合

シーケンス演算子の出力は順序付集合である一方，リレーショナル演算子の入出力は順序無集合であるため，両者の相性は悪い．しかしながらシーケンス演算子の出力を 1 タプルに変換するか，あるいは集合間の順序をほどけば，両者を統一した観点から処理可能になる．

リレーショナル処理系では出力結果が増大するのは結合演算子と直積演算子である．それらは 2 入力演算子であり，入力サイズが N と M の時に最大で NM 件のタプルを生成する．この巨大な結果の生成・処理を効率化するために現在でも多くの研究が行われている．一方，シーケンス演算子の場合 3000 件の入力に対して 160922926 件程度の出力が生成されるため，効率化は重要になると考えられる．リレーショナル処理系において選択演算子や射影演算子のプッシュダウンによる演算子融合が大きな効率化をもたらしている．シーケンス演算子が含まれる体系においても同様に演算子融合が重要になると我々は考える．

6. 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．実験により縮退表現は従来手法である SASE に比べて 4000 倍程度の高速であり，3780 倍程メモリサイズが小さくなることが示された．これより提案である縮退表現は処理時間とメモリサイズを同時に効率化できると結論する．

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」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] Tran, T. T. L., Peng, L., Diao, Y., McGregor, A. and Liu, A.: CLARO: modeling and processing uncertain data streams, VLDB Journal, Vol. 21, No. 5, pp. 651–676 (2012).
[11] Wu, E., Diao, Y. and Rizvi, S.: High-Performance Complex Event Processing over Streams, SIGMOD Conf. (2006).
[12] Zhang, H., Diao, Y. and Immerman, N.: On Complexity and Optimization of Expensive Queries in Complex Event Processing, SIGMOD Conf. (2014).