頻出時系列パターン発見アルゴリズムの実装

大谷 英行

2008年2月

北海道大学工学部情報工学科
情報知識ネットワーク研究室

要 旨

巨大なデータ集合の中から，有用な知識や情報を発見するデータマイニングは，現代では様々な分野で重要な技術である．特に，時系列データを対象とした時系列データマイニングが注目を集めている．本論文では時系列データマイニングの解析手法のひとつである頻出時系列エピソード発生を研究する．与えられた文字列データの中から，頻出エピソードと呼ばれる頻出する部分構造を効率的に取り出す手法である，右極小出現リストを用いた深さ優先探索アルゴリズムを提案する．さらに，そのアルゴリズムを実装し，計算機実験によって，他のアルゴリズムと比較し，本アルゴリズムの有効性を示す．

目次

第1章

序論

近年では，インターネットの普及やコンピュータの性能上昇に伴い，コンピュータ上で扱うデータ量が増加してきている．大きなデータ構造の中から，知識や情報，ルールを発見するデータマイニングは様々な分野で有用である．例えば医学では膨大な文字群であるDNAに対して，繰り返し出てくる文字列を得て特徴を掴んだり，そこから新しい仮説を立てることができる．また，商業分野では，購買履歴から有効な売り出し方を発見することができる．

本稿では，データマイニングの中でも系列データの中から頻出パターンを発見する問題を取り扱う．頻出パターンとはある系列データの中の，頻出な部分データ列のことである．系列データとは同質のデータ (本論文ではアルファベットの文字を用いる) を直列に並べたものを指す．系列データによっては，非常に容量の大きいデータ集合を扱いうる．大きなデータ集合を扱うにあたって注意しなくてはならないのが計算量である．頻出パターンを列挙する際の計算量は必ず系列データの大きさに比例する．データが大きくても現実的な時間で列挙するためには，高速なアルゴリズムが望ましい．

与えられたアルファベットの各要素を**イベント** (event) と呼び，そのイベントが並んだ系列データを**イベント列** (event sequence) と呼ぶ．本稿では，第2章で述べる条件を持ったイベントの集合のことを**エピソード** (episode) と呼ぶ．

イベント列の中から，頻出なエピソードを見つけるにあたって，エピソードパターンが頻出であるためには，そのエピソード中のイベントの**発生時刻** (occurrence time) が互いになるべく近くなければならない．どれくらい近ければよいかということを決定するために，**窓** (window) を与える．窓とはイベント列の一部であり，イベント列は部分

的に重なる窓の連続から成っているものと定義する．エピソード長は，この窓の長さよりも小さくなければならない．そして，エピソードが入っている窓の総数で，そのエピソードが頻出かどうかを決定する．本稿の最終的な目標は，このイベント列から，与えられた窓の長さよりも短い頻出なエピソードを，効率よく見つけることである．

関連研究として，Mannila ら [3] は幅優先探索を用いた頻出エピソード発見アルゴリズムを与えている．内田 [5] は，窓なしエピソードと窓付きエピソードに対して，深さ優先探索と極小出現リストを用いた頻出エピソード発見アルゴリズムを与えている．内田 [5] では，右極小ではない通常の極小出現リストを用いている点が本稿と異なる．

本稿では系列データから頻出直列パターンを列挙する手法を考察する．初めに，深さ優先探索と幅優先探索の両方のパターン探索法に対して，素朴な手法である DFS-Naive と BFS-Naive を与える．次に，エピソードの頻度計算に関して，右極小出現という区間を用いた手法を提案し，その効果を実装することで示す．極小出現区間の概念は Mannila ら [3] によって導入されたが，本稿ではこの条件を緩めた右極小出現区間の概念を導入し，これを用いたエピソードの頻度計算アルゴリズムを与える．これにより，簡素かつ効率よい出現区間の更新計算が可能になる．また，その他の手法として，Mannila ら [3] が提案した手法を紹介する．

第 2 章では，本論文で扱う用語の定義と問題を説明し，第 3 章では深さ優先探索と幅優先探索の Naive なアルゴリズムと，それぞれの探索で右極小出現を用いた効率的なアルゴリズム DFS-MO と BFS-MO ををを提案する．第 4 章では Mannila ら [3] のアルゴリズム BFS-MTV を説明する．また，第 5 章では，第 3 章で提案したアルゴリズムを実装し，様々な角度からの実証実験を行い，DFS-MO と BFS-MO が Naive なアルゴリズムよりも効率的であることを示す．その結果，提案手法が速度という点で効率的であることが分かった．

第2章

準備

本章では，本稿で用いる**半順序 (partial ordered)** や，イベント列，エピソードなどの基本的概念を定義し，頻出エピソード発見問題について述べる．

2.1 半順序集合

本節では，半順序という集合の概念と，それを説明するために必要な**関係 (relation)** について説明する [4].

定義 1 2つの集合 A と B の**直積 (cartesian product)** は $A \times B$ と書き，

$$A \times B = \{\ (a, b) \mid a \in A, b \in B\ \}$$

と定義する．

定義 2 2つの集合 A と B の間の関係 R とは，$A \times B$ の部分集合で定義される．

$$(a, b) \in R$$

において，a と b は関係 R を持つといい，

$$a\ R\ b$$

と書く．

R を集合 A 上の関係とし，$a, b, c \in A$ とする．A の任意の要素 a について，

$$a\ R\ a$$

が成り立つならば，R は**反射的 (reflexive)** であるという．

$$a\ R\ b \text{かつ} b\ R\ a \text{ならば} a = b$$

である時，R は**反対称的 (anti-symmetric)** であるという．

$$a\ R\ b \text{かつ} b\ R\ c \text{ならば} a\ R\ c$$

ならば，R は**推移的 (transitive)** であるという．

定義 3 集合 A 上の関係 R が，反射的で，反対称的で，かつ推移的ならば，R を半順序と呼ぶ．この時，A は**半順序集合 (partially-ordered set)** と呼ぶ．

2.2 イベント系列

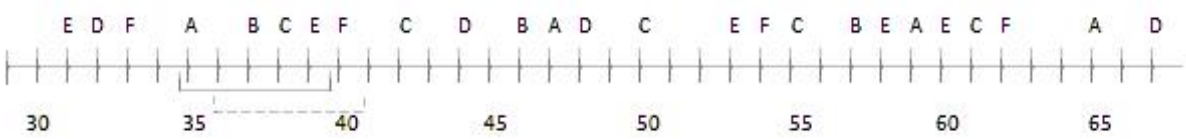

図 2.1:イベント列と窓

ある時系列に沿ったイベント列を考える．イベントの集合 E が与えられた時，ひとつのイベントを (A, t) と表す．ここで，$A \in E$ は E の要素であり，t は整数でイベントの**発生時刻 (occurrence time)** を表す．イベント列 $\mathbf{s}$ とは，3 つ組 (s, T_s, T_e) である．ここで，

$$s = \langle (A_1, t_1), (A_2, t_2), ..., (A_n, t_n) \rangle \, (t_1 < t_2 <, ..., < t_i <, ..., < t_n)$$

である．T_s と T_e は整数で，それぞれ，イベント列の**開始時刻 (starting time)** と**終了時刻 (ending time)** を表す．すなわち，任意の $1 \leq i \leq n$ について，$T_s \leq t_i < T_e$ である．

イベント列を $\mathbf{s}$= (s, T_s, T_e) とする．窓 $\mathbf{w}$ を，$\mathbf{w}$= (w, t_s, t_e) と表す．ここで，$t_s < T_e$ かつ $t_e > T_s$ で，w は，$t_s \leq t < t_e$ である，$(A, t) \in s$ の集合である．時間長 $t_e - t_s$ を，窓 $\mathbf{w}$ の**幅 (width)** といい，$width(\mathbf{w})$ と表記する．イベント列 $\mathbf{s}$ と，整数 win が与えられた時，$\mathbf{s}$ 上の全ての窓 $\mathbf{w}$ の集合の中で $width(\mathbf{w}) = win$ であるものを，$\mathcal{W}(\mathbf{s}, win)$ と書く．T_s のみを含む最初の窓と，T_e のみを含む最後の窓は，それぞれイベント列の開始地

点と終了地点のみを含み，外にはみ出したものと考える．この時，$\mathcal{W}(\mathbf{s}, win)$ の窓の数は $T_e - T_s + win - 1$ である．

図 2.1 に，$s = \langle (E, 31), (D, 32), (F, 33), (A, 35), (B, 37), (C, 38), ..., (D, 67) \rangle$ であるイベント列 $\mathbf{s}= (s, 29, 68)$ と，2 つの幅 5 の窓の例を示す．ここで，イベント列の時間は [29,68) である．時刻 35 から始まる窓は実線で，時刻 36 から始まる窓は点線でそれぞれ描かれている．時刻 35 から始まる窓は，$(\langle (A, 35), (B, 37), (C, 38), (E, 39) \rangle, 35, 40)$ である．ここで，$(F, 40)$ は窓の中には入っていないことに注意する．一方，時刻 36 から始まる窓に $(A, 35)$ は入らないが，$(F, 40)$ は入る．このイベント列における，幅 5 の窓の数は 43 であり，最初の窓は $(\emptyset, 25, 30)$ で，最後の窓は $(\langle (D, 67) \rangle, 67, 72)$ である．

2.3 エピソード

エピソードとは，発生するイベントの半順序集合で，非循環連結グラフで描くことができる．エピソードには図 2.2 のようなものがある．エピソード α のように，イベント列

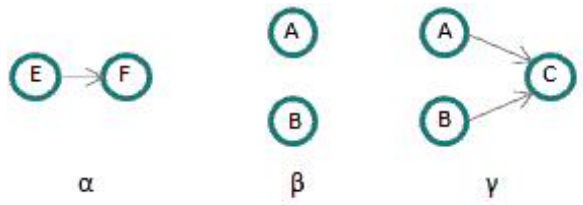

図 2.2:エピソードの種類

中で順番を持つイベント集合を，**直列エピソード (serial episode)** と呼ぶ．ただし α では，E と F の間に他のイベントが発生していてもよい．エピソード β のように，順番に制約のないイベント集合を，**並列エピソード (parallel episode)** と呼ぶ．エピソード γ は，非直列かつ非並列なエピソードの例である．本稿では，主に直列エピソードのみを考察する．

形式的には，α のノードからイベントへ $g(x) = A_h(x)$ で，$x \neq y$ 且つ $x \leq y$ である，全ての $x, y \in V$ について，$t_h(x) < t_h(y)$ であるような単射写像 $h : V \rightarrow 1, ..., n$ が存在する時，エピソードは $(V, \leq, g)$ の 3 つ組で表す．V はノード集合で，$\leq$ は V 上の半順序関係である．g は，各ノードに E のイベントを割り当てる写像で，$g : V \rightarrow E$ と表す．こ

こで各ノードに割り当てたイベントも又，ノードと同様の半順序関係を持たねばならない．あるエピソード α が与えられた時，α のサイズは $|V|$ であり，$|\alpha|$ と表記する．エピソード α が直列であるとは，全ての $x, y \in V$ について，$x \leq y$ もしくは $y \leq x$ となることをいう．エピソード α が並列であるとは，$x \neq y$ であるような，全ての $x, y \in V$ について $x \not\leq y$ となることをいう．エピソード α が単射であるとは，エピソード内に同じイベントが 2 度以上発生しないものである．例えば，図 2.2 の α について考えてみる．α について，集合 V には 2 つのノードがあり，それらを x, y と置く．写像 g はこれらのノードを $g(x) = E, g(y) = F$ とラベル付けする．この時 E は F より前に発生している，つまり $x \leq y$ である．さらにエピソード α は，同じイベントが 2 つ以上含まれていないので単射である．

次に，**部分エピソード (sub episode)** を定義する．$\beta = (V', \leq', g')$ が $\alpha = (V, \leq, g)$ の部分エピソードであるとは，全てのノード $v \in V'$ について，$g'(v) = g(f(v))$ となるような $f : V' \to V$ が存在し，$v \leq' w$ となる，$w \in V'$ に対し $f(v) \leq f(w)$ であることをいう．またこの時，α は β の**拡大エピソード (super episode)** と呼び，$\beta \preceq \alpha$ と表記する．例として，図 2.2 において β は，γ の部分グラフなので $\beta \preceq \gamma$ である．ここで，β の 2 つのノードは，それぞれ γ に同じラベル付けされたノードを持っており，β のノードは順序づけされてないので，γ の対応するノードもまた順序づけされないことに注意する．

エピソードがイベント列中で発生しているとは，エピソードである各ノードが該当するイベントと等しいイベントを持ち，そのイベント間で，ノードと同様の半順序が反映されているようなイベント集合が列中に存在していることを言う．形式的には，あるエピソード $\alpha = (V, \leq, g)$ は，

$$\mathbf{s} = (\langle (A_1, t_1), (A_2, t_2), ..., (A_n, t_n) \rangle, T_s, T_e)$$

の中に発生しているという．

2.4 頻出エピソード発見問題

イベント列 $\mathbf{s}$ と窓幅 win が与えられた時，$\mathbf{s}$ 中に発生するエピソード α の**頻度 (frequency)** を，

$$fr(\alpha, \mathbf{s}, win) = \frac{|\{\mathbf{w} \in \mathcal{W}(\mathbf{s}, win) | \alpha \text{ occurs in } \mathbf{w}\}|}{|\mathcal{W}(\mathbf{s}, win)|}.$$

と定義する．すなわち $fr(\alpha, \mathbf{s}, win)$ は，イベント列の中で，あるエピソードが出現する窓の割合である．

ある正数である**頻度のしきい値 (freqency threshold)** $0 \leq \delta \leq 1$ が与えられた時，$fr(\alpha, \mathbf{s}, win) \geq \delta$ ならば，α は頻出であるとする．$\mathcal{C}$ を，全ての並列エピソードのクラス，または全ての直列エピソードのクラスとする．ここでの問題は，与えられたエピソードのクラス $\mathcal{C}$ から，全ての頻出であるエピソードを見つけることである．$\mathbf{s}$ と win, δ によって得られる，頻出であるエピソードの集合を，**頻出エピソード集合 (frequent episode set)** と呼び，$\mathcal{F}(\mathbf{s}, win, \delta)$ と書く．

定義 4 頻出エピソード発見問題 (Frequent Itemset Mining, FIM) とは，入力としてイベント列 $\mathbf{s}$ と頻出しきい値 δ が与えられた時に，$\mathbf{s}$ 上の全ての頻出エピソードを出力する問題である．ただし，各頻出エピソードは，重複なく出力されるものとする．

ここで，本稿では直列エピソードに限った場合の発見問題に取り組む．

2.5 木の深さ優先探索と幅優先探索

ここでは，木やグラフを探索するアルゴリズムである，深さ優先探索と幅優先探索について説明する [1, 2]．木の深さ優先探索は，図 2.3 の手順で与えられた木の全ノードを訪問するアルゴリズムである．深さ優先探索を用いる利点は，使用するメモリ容量が少なくて済むことである．ループを持つようなグラフに対しては，一度訪問した地点に印をつけて，二度訪問しないようにする．

木の幅優先探索は，図 2.4 の手順で与えられた木の全ノードを訪問するアルゴリズムである．幅優先探索を用いる利点は，ループを持つようなグラフにそのまま適用できることである．

Algorithm　DFS
1: 根 (初期ノード) を決定する;
2: **while** まだ探索していない子ノード A がある **do**
3:　　子ノード A を現在のノードとする;
4: **end while**
5: 親ノードに戻る;

図 2.3：深さ優先探索アルゴリズムの概要

Algorithm　BFS
1: 根 (初期ノード) を決定し，そのノードをノード集合 **v** に入れる;
2: **while v** の中にノードが存在する **do**
3:　**v** の中から一つノードを取り出し，そのノードを現在のノード v_now とする;
4:　v_now を探索したとみなし，**v** の中から消す;
5:　$i = 0$;
6:　**while** v_now につながっていて，まだ探索していない隣接ノード $v\_not[i]$ が存在する **do**
7:　　$v\_not[i]$ を **v** の中に入れる;
8:　　$i = i + 1$;
9:　**end while**
10: **end while**
11: 親ノードに戻る;

図 2.4：幅優先探索アルゴリズムの概要

第3章

右極小出現区間を用いた頻出直列エピソード発見アルゴリズム

本章では，直列エピソードのクラスに対して，頻出エピソード発見問題を解くアルゴリズムについて説明する．エピソードの生成に関して，**幅優先探索 (BFS, Breadth-First Search)** と，**深さ優先探索 (DFS, Depth-First Search)** のそれぞれの場合について，頻度の計算を素朴な方法と**右極小出現区間 (MO, Minimal Occurrence)** を用いたアルゴリズムを紹介する．

図 3.1 は 4 つのアルゴリズムを示している．ここで，各行はエピソードの生成方法を表す．各列は頻度の計算方法を表す．例えば，素朴な幅優先探索である，**BFS-Naive** は 3.2 節で説明する．

	Naive	MO
DFS	DFS-Naive (3.1節)	DFS-MO (3.3節)
BFS	BFS-Naive (3.2節)	BFS-MO (3.4節)

図 3.1:頻出直列エピソード発見アルゴリズムの種類

3.1 Naive な深さ優先探索アルゴリズム

本節では，素朴な頻度計算を用いた，深さ優先探索アルゴリズムについて説明する．図 3.2 に，深さ優先探索を候補エピソードの生成に用いた，素朴な頻度計算を用いた頻出直列エピソード発見アルゴリズム DFS-Naive を与える．

ここに，エピソード α とイベント e が与えられた時，$\alpha \cdot e$ は α の末尾に e をつけたエピソードである．初期ノードを除く各ノードには，イベントが一つ入る．初期ノードは空である．

Algorithm DFS-Naive
入力: イベント列 **s**，初期から現在のノードまでのイベントをつなげたエピソード α(ただし初期ノードは空なので無視する)，頻出しきい値 δ，窓幅 win，**s** 上で頻出である 1 文字のエピソード集合 $First\_e$.
出力: **s** に頻度 δ 以上で出現する，全ての直列エピソード.

```
1: if |α| ≤ win then
2:   for (0 ≤ i < |First_e|; i++) do
3:     if Naive-Recognize(α・First_e[i], s)/s 上の窓の総数 > δ then
4:       α・First_e[i] を出力;
5:       DFS-Naive(α・First_e[i], s);
6:     end if
7:   end for
8: end if
```

図 3.2：深さ優先探索を頻出エピソード発見問題に用いた場合の素朴なアルゴリズム

Algorithm Naive-Recognize
入力: イベント列 $\mathbf{s} = \{s, T_s, T_e\}$，エピソード α，窓幅 win.
出力: **s** 上で win 以内で α が出てくる回数を表す整数型変数 $count$.

```
1: i = T_s, j = 0, count = 0;
2: for (T_s ≤ i < T_e; i++) do
3:   j = 1;
4:   if s[i] = α[0] then
5:     k = 1;
6:     while k < win do
7:       if s[i + k] = α[j] then
8:         if j = |α| − 1 then
```

```
 9:              count = count + 1;
10:              goto 2;
11:            end if
12:            j = j + 1;
13:          end if
14:          k = k + 1;
15:        end while
16:      end if
17: end for
```

図 3.3：イベント列上でエピソードの頻度を求める素朴なアルゴリズム

3.2 Naive な幅優先探索アルゴリズム

本節では，Naive な幅優先探索アルゴリズムについて説明する．図 3.4 に，幅優先探索を頻出直列エピソード発見問題に用いた，単純な頻出直列エピソード発見アルゴリズム BFS-Naive を与える．

Algorithm BFS-Naive
入力: E によるイベント列 $\mathbf{s}$，窓幅 win，頻出しきい値 δ.
出力: 頻出直列エピソード集合 $\mathcal{F}(\mathbf{s}, win, \delta)$.

```
 1: イベント列の中に出てくる全てのイベントを，1 文字の候補エピソード C_1 と設定
    する (ただし同じイベントのエピソードは作らない);
 2: l = 1;
 3: while C_l ≠ ∅ do
 4:   for all α ∈ C_l do
 5:     if Naive_Recognize(α, s)/s 上の窓の総数 ≥ δ then
 6:       α を出力する;
 7:       if (l = 1) then
 8:         First_e+ = α;
 9:       end if
10:     end if
11:   end for
12:   l = l + 1;
13:   Naive-Candidate(サイズ l − 1 のソートされた頻出直列エピソード F_{l−1});
14: end while
```

図 3.4：幅優先探索による Naive な頻出直列エピソード発見アルゴリズムの概要

Algorithm Naive-Candidate($\mathcal{F}_l$)
入力: ソートされたサイズ l の頻出直列エピソード集合 $\mathcal{F}_l$.
出力: ソートされたサイズ l+1 の頻出候補直列エピソード集合 $\mathcal{C}_{l+1}$.

```
1: i = 1, j = 0;
2: for (i ≤ |F_l|; i = i + 1) do
3:    if (l = 1) then
4:       First_e = C_1;
5:    end if
6:    for (j = 0; j < sizeof(First_e); j = j + 1) do
7:       F_l[i] について，First_e[j] を，自分のエピソードの末尾に付けて，サイズ l+1
         のエピソード alpha に加えていく;
8:    end for
9: end for
10: C_{l+1} を出力する;
11: if (l = 1) then
12:    First_e.clear();
13: end if
```

図 3.5：頻出候補直列エピソード生成アルゴリズム

図 3.5 に Naive な候補エピソード生成アルゴリズム Naive-Candidate を示す.

ただし，BFS-Naive の代わりに次章で紹介する Candidate-Episode アルゴリズムを使うことで，比較的簡単に，より早いアルゴリズムにすることができると考えられる.

3.3 右極小出現区間を用いた深さ優先探索アルゴリズム

本節では，右極小出現区間を用いた深さ優先探索アルゴリズムについて説明する．その前に，このアルゴリズムで用いる用語の説明を行う.

3.3.1 右極小出現リスト

定義 5 イベント出現リスト (event occurrence list)．イベント e に対して，イベント出現リスト $EOcc(e,\mathbf{s})$ とは，イベント列 $\mathbf{s}$ が与えられた時の，$\mathbf{s}$ 中に現れる e の位置 pos を順番に格納したリスト $EOcc(e,s) = \{pos_1, ..., pos_n\}$ $(n \geq 0)$ である.

定義 6 (Mannila ら [3]) エピソード α とイベント列 $\mathbf{s}$ が与えられた時，(t_s, t_e) が**極小出現区間 (minimal occurence)** であるとは，

1. α は，$\mathbf{s}$ 上の窓 $\mathbf{w} = (w, t_s, t_e)$ の中に出現している．
2. $t_s \leq t'_s$ かつ $t'_e \leq t_e$，$width(\mathbf{w}') < width(\mathbf{w})$ を満たすような，$\mathbf{w}$ の部分窓 $\mathbf{w}' = (w', t'_s, t'_e)$ の中に α が存在しない．

を満たすことを言う．

これに対して，右極小出現区間を提案する．

定義 7 エピソード α とイベント列 $\mathbf{s}$ が与えられた時，α の出現区間 $[i, j]$ が**右極小出現区間 (right-minimal occurence)** であるとは，

1. α は，$\mathbf{s}$ 上の窓 $\mathbf{w} = (w, t_s, t_e)$ の中に出現している．
2. $i' = i$ かつ $j' < j$ で，$j' - i' < j - i$ を満たすような，α の出現区間 $[i', j']$ が存在しない．

を満たすことを言う．

補題 1 (t_s, t_e) が極小出現区間ならば，右極小出現区間である．

一般に，上の補題の逆は成立しない．

定義 8 あるエピソード α と，イベント列 $\mathbf{s}$ が与えられた時，α の右極小出現区間の集合を，$MO(\alpha, \mathbf{s}) = \{ (t_s, t_e) \mid (t_s, t_e)$ は α の右極小出現区間 $\}$ と表し，これを**右極小出現リスト (right-minimal occurrence list)** と呼ぶ．例えば，図 1 において図 2 の α の右極小出現リストは $MO(\alpha, \mathbf{s}) = \{(35, 38), (47, 58)\}$ となる．

図 3.6 に，右極小出現リストを用いた頻出エピソード発見アルゴリズムを示す．WinCount 関数はサブルーチンであり，詳細は後で説明する．

Algorithm　DFS-MO($\alpha, \mathbf{s}$)
入力: イベント列 $\mathbf{s} = (s, T_s, T_e)$，エピソード α(ただし初期ノードでは空)，s に出てくるイベント集合 Σ，窓幅 win，頻出しきい値 δ.
出力: $\mathbf{s}$ に頻度 δ 以上で出現する，エピソードサイズが窓幅以内の全てのエピソード.

1: **if** $\alpha = \emptyset$ **then**
2:　**for all** $e \in \Sigma$ **do**
3:　　$EOcc(e, \mathbf{s})$ を作成する;
4:　　$MO(e, \mathbf{s}) = \emptyset$;
5:　　$MO(e, \mathbf{s}) = \cup\{[pos, pos + 1) \mid pos \in EOcc(e, \mathbf{s})\}$;
6:　　**if** WinCount($MO(e, \mathbf{s})$)/(窓総数) $\geq \delta$ **then**
7:　　　$First\_e = \cup e$;
8:　　　e を出力;
9:　　　Window-Episode-Scan($e, \mathbf{s}$);
10:　　**end if**
11:　**end for**
12: **else**
13:　**for all** $e \in First\_e$ **do**
14:　　Update($MO(\alpha, \mathbf{s}), e$);
15:　　**if** WinCount($MO(\alpha \cdot e, \mathbf{s})$)/(窓総数) $\geq \delta$ **then**
16:　　　$\alpha \cdot e$ を出力;
17:　　　**if** $|\alpha \cdot e| < win$ **then**
18:　　　　Window-Episode-Scan($\alpha \cdot e, \mathbf{s}$);
19:　　　**end if**
20:　　**end if**
21:　**end for**
22: **end if**

図 3.6：右極小出現リストを用いた頻出エピソード発見アルゴリズム

3.3.2 右極小出現リストの更新

エピソード α の末尾に，サイズ 1 の頻出イベントを付け足したエピソードの右極小出現リストを作成するアルゴリズム **Update** について説明する．右極小出現区間の定義から，ある 2 つの連続した右極小出現区間 $(MO1.s, MO1.e), (MO2.s, MO2.e)$ について考えられる順序は，

1. $MO1.s < MO2.s \; and \; MO2.s < MO1.e \; and \; MO1.e \leq MO2.e$
2. $MO1.s < MO2.s \; and \; MO1.e \leq MO2.s \; and \; MO1.e < MO2.e$

である．これより，新しくイベントを末尾に付けたときの，右極小出現区間の存在可能な様子の例を示したものを図 3.9 にまとめた．図 3.7 で，新しいイベントは，それぞれ位置 52，56，56，60 に存在しており，極小区間はそれぞれそこまで伸ばした状態である．アルゴリズム Update の基本的な考え方は，以下のとおりである．

あるエピソード α の全ての頻出区間 $\mathrm{MO}(\alpha)$ について，それぞれの出現区間が $(MO.s_i, MO.e_i)$ で表され，新しく末尾につけるイベントを c とする．更に窓幅 win を与える．この時，新しいエピソード $\alpha \cdot c$ を作るには，$MO.e_i + 1$ から $MO.s_i + win - 1$ までで c の捜索を開始する．もしあればその c の位置を pos に記憶し，$\mathrm{MO}(\alpha \cdot e)$ に $(MO.s_i, pos)$ を記録する．以上から，図 3.7 の MO2 は窓幅より長くなり，新しい頻出区間にはならないことが分かる．図 3.8 に Update アルゴリズムを示す．ただし $(start, end)$ は，ある右極小出現区間を示す．

補題 2 Update アルゴリズムの計算量は $O(|MO(\alpha, \mathbf{s})| \cdot win)$ である．

3.3.3 右極小出現リストからの頻度の計算

右極小出現リストから，エピソード出現窓数を求める WinCount 関数について説明する．基本方針としては，各右極小出現区間の最後の位置に窓の先頭を合わせ，$MO[i].start <$ 窓の先頭の位置となるまで窓を動かす．窓を動かしている間は，窓の中にエピソードが入っているので，動かした分だけ出現窓数 $count$ をインクリメントする．ただし図 3.9 の

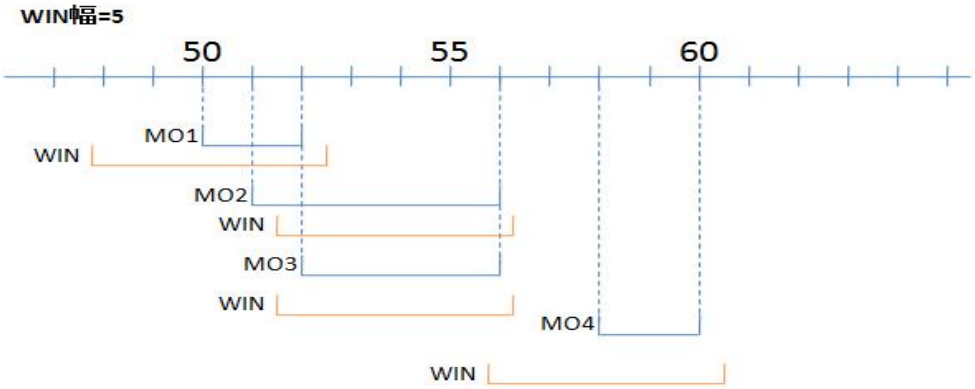

図 3.7:頻出区間

Algorithm Update($MO(\alpha, \mathbf{s})$,c)
入力: 右極小出現リスト $MO(\alpha, \mathbf{s})$，エピソード α，イベント列 $\mathbf{s}$，窓幅 win,α の末尾につけるイベント $c \in First_e$.
出力: 新しいエピソード $\alpha \cdot c$ の右極小出現リスト $MO(\alpha \cdot c, \mathbf{s})$.

```
1:  MO(α・c, s) = ∅;
2:  for (i = 0; i < |MO(α, s)|) do
3:      p = MO(α, s);
4:      for (j = p.end + 1; j < p.start + win; j++) do
5:          if (s[j] = c) then
6:              q =new pair();
7:              q.start = p.start, q.end = j;
8:              MO(α・c, s).add(q);
9:              break;
10:         end if
11:     end for
12: end for
```

図 3.8：新しい右極小出現リストを作るアルゴリズム

ような一つの窓の中に複数個エピソードが入っている場合は，このままでは，一つの窓の中で複数回 *count* がインクリメントされてしまう．よってそれを回避するために，

一つ前の右極小出現区間のスタート位置 ＜ 現在の右極小出現区間を含む窓のスタート位置

を満たす窓の場合のみ *count* をインクリメントするようにする．

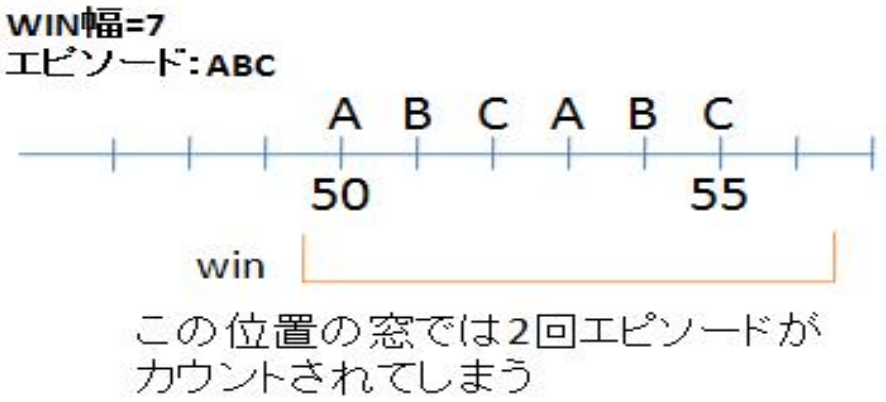

図 3.9:一つの窓内に 2 つ右極小出現区間がある場合

図 3.10 に，アルゴリズムの概要を示す．

補題 3 Update アルゴリズムの計算量は $O(|MO(\alpha, \mathbf{s})| \cdot win)$ である．

エピソードが頻出かどうかを判定する場面で，各エピソードごとに，与えられたイベント列の長さ分だけ窓を動かしてそのエピソードが入っている窓の数を数えるのが素朴なアルゴリズムである．それに対して右極小出現リストを用いる利点は，各エピソードの出現位置を覚えておくことで窓を動かす回数を減らせることにある．

3.4 右極小出現区間を用いた幅優先探索アルゴリズム

本節では，右極小出現区間を用いた幅優先探索アルゴリズム BFS-MO について説明する．3.2 節で用いた BFS-Naive 中の，Naive-Recognize の代わりに WinCount アルゴリズムを用いることで，右極小出現区間による幅優先探索を実現する．ただし，BFS-Naive では，新たな右極小出現を算出していないため，候補エピソードを作成する Naive-Candidate 内で算出する必要がある．具体的にはエピソード毎の右極小出現リストを保存することで実現する．Naive-Candidate 内で算出されたエピソードについて Update を行うことで，

Algorithm　WinCount
入力: 右極小出現リスト $MO(\alpha, \mathbf{s})$，窓幅 win.
出力: エピソード出現窓数を表す整数型変数 $count$.

```
1: last_start = 0, last_end = 0, count = 0, i = 0;
2: while i < |MO(α, s)| do
3:    p = MO[i];
4:    f_0 = p.end, w_0 = f_0 − win + 1;
5:    while (true) do
6:       if (w_0 ≤ last_start) then
7:          w_0 = w_0 + 1;
8:          continue;
9:       end if
10:      if (p.start < w_0) then
11:         break;
12:      end if
13:      count = count + 1;
14:      w_0 = w_0 + 1;
15:   end while
16:   last_start = p.start, last_end = p.end;
17:   i = i + 1;
18: end while
19: count を出力する.
```

図 3.10：右極小出現リストからエピソード出現窓数を表すアルゴリズム

幅優先探索で各右極小出現リストを得られるので，これを WinCount に用いれば，頻出な窓数を求められる.

この場合も 3.2 同様，BFS-Naive の代わりに次章で紹介する Candidate-Episode アルゴリズムを使うことで，比較的簡単に，より早いアルゴリズムにすることが期待できる.

第4章

Mannilaらの頻出直列エピソード発見アルゴリズム

本章では，頻出エピソード発見問題の中でも直列エピソード扱う問題について，効率よく解決するためのMannilaら [3] のアルゴリズムBFS-MTVについて，詳細に説明する．個のアルゴリズムは3.2節のBFS-Naiveにおいて頻度計算を，現在のパターン候補集合に対して一括して行い, 効率を上げた拡張になっている．

4.1 アルゴリズムの概要

アルゴリズムBFS-MTVの概要を図4.1に示す．

Recognizing-Episodeを候補エピソード認識アルゴリズムと呼び，Candidate-Episodeを候補エピソード生成アルゴリズムと呼ぶ．以降の節で，それぞれのアルゴリズムの詳細について述べる．

4.2 幅優先探索を用いた候補エピソードの生成

本節では，候補エピソード生成アルゴリズムについて述べる．候補エピソード生成にあたって，アルゴリズムを効率的にするために重要な補題がある．

補題 4 あるエピソードαが，イベント列$\mathbf{s}$において頻出であるならば，$\beta \preceq \alpha$を満たす部分エピソードβもまた$\mathbf{s}$中で頻出である．

Algorithm　BFS-MTV
入力: E によるイベント列 $\mathbf{s}$，窓幅 win，頻出しきい値 δ.
出力: 頻出直列エピソード集合 $\mathcal{F}(\mathbf{s}, win, \delta)$.

1: イベント列の中に出てくる全てのイベントを，サイズ 1 の候補エピソード C_1 と設定する (ただし同じイベントのエピソードは作らない);
2: $l = 1$;
3: **while** $C_l \neq \emptyset$ **do**
4:　**for all** $\alpha \in C_l$ **do**
5:　　**if** Recognizing-Episode$(\alpha, \mathbf{s}, win) \geq \delta$ **then**
6:　　α を出力する;
7:　　**end if**
8:　**end for**
9:　$l = l + 1$;
10:　Candidate-Episode(サイズ $l-1$ のソートされた頻出直列エピソード $\mathcal{F}_{l-1}$);
11: **end while**

図 4.1：Mannila らの頻出直列エピソード発見アルゴリズムの概要

この補題によって，新しい候補エピソードを作る際に，候補エピソードを構成する，全ての部分エピソードは頻出である必要がある．すなわち，頻出でないと判断されたエピソードを含むエピソード候補を削ることができるため，扱うエピソードを減らすことができる．

各エピソードはイベントの配列で，エピソード集合は辞書順に並びかえて与えられるものとする．あるエピソード α の i 番目のイベントを，$\alpha[i]$ と表記し，あるサイズ l のエピソード集合 $\mathcal{F}_l$ の i 番目のエピソードの j 番目のイベントは，$\mathcal{F}_l[i][j]$ と表記する．エピソード集合が辞書順に並べ替えられているので，1 番目のイベントが同じエピソードは，連続しているはずである．特に，サイズ l のエピソード集合 $\mathcal{F}_l[i]$ と，$\mathcal{F}_l[j]$ の最初から $l-1$ 番目までのイベントが同じである時，$i \leq k \leq j$ を満たすような全ての k においては，$\mathcal{F}_l[k]$ はサイズ l で，最初から $l-1$ 番目までのイベントもまた同じはずである．このように，サイズ l のエピソード集合において，最初から $l-1$ 番目までのイベントが同じであるような最大のエピソード列を，**ブロック (block)** と呼ぶ (図 4.2).

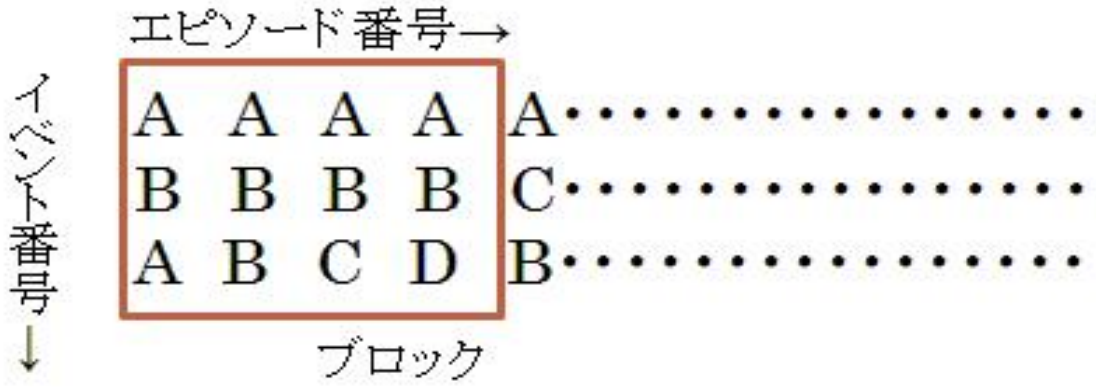

図 4.2:ブロック

アルゴリズムは図 4.3 のようになる.

Algorithm Candidate-Episode($\mathcal{F}_l$)
入力: ソートされたサイズ l の頻出直列エピソード集合 $\mathcal{F}_l$.
出力: ソートされたサイズ l+1 の頻出候補直列エピソード集合 $\mathcal{C}_{l+1}$.

```
1: i = 1, j = 0;
2: for (i ≤ |F_l|; i = i + 1) do
3:   for j = 0;F_l[i] の所属するブロックのエピソード F_l[j] が存在する;j = j + 1 do
4:     F_l[i] に，F_l[j] の末尾のイベントを，自分のエピソードの末尾に付けて，サイズ l + 1 のエピソード α とする.
5:     for (x = 1; x ≤ l − 1; x = x + 1) do
6:       if α から x 番目の文字を抜かしたサイズ l の全ての部分文字列 β が，与えられたエピソード集合 F_l の中に存在する then
7:         C_{l+1} = ∪ β;
8:       end if
9:     end for
10:  end for
11: end for
12: C_{l+1} を出力する;
```

図 4.3：頻出候補直列エピソード生成アルゴリズム

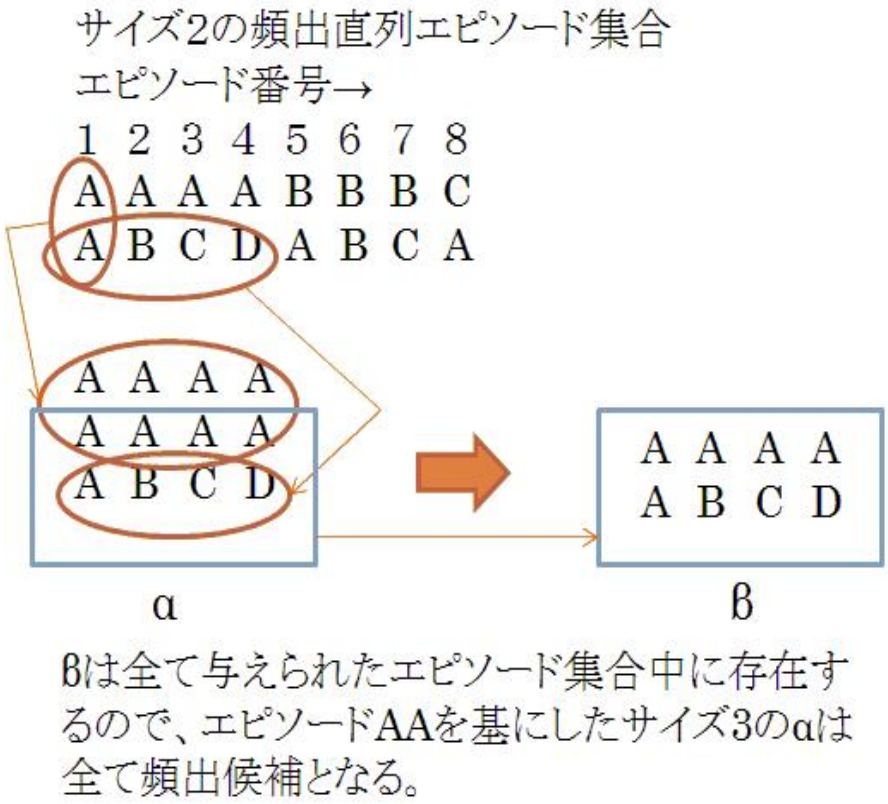

図 4.4:頻出直列候補エピソードの作成

4.3 候補エピソードの頻度計算

本節では，候補エピソードを受け取った時に，そのエピソードの，与えられたイベント列内での頻度を計算するための，効率的なアルゴリズムを説明する．初めに，アルゴリズム内で使う用語について説明する．以下は，あるエピソード α が与えられた場合である．

定義 9 $\alpha.inwindow$ は，エピソード α が窓の中に出現し，かつ窓内で位置の異なる場所に α ない時，その時の，窓の開始時刻を示す．

定義 10 $\alpha.freq\_count$ は，エピソード α が全て入っている窓の数を示す．最終的にはこの値によって，エピソード α が頻出がどうかを決定する．

定義 11 $\alpha.initialized(i)$ は，i 番目のエピソードイベントに到達したエピソードの，最新の初期化時間 (α の最初のイベントが窓に入った時間) を示す．主に，$\alpha.inwindow$ を更新する際に，同じエピソードが既に窓の中にないかを知るために用いる．

定義 12 (α, x) とは，イベント列中のある位置のエピソード α が，$x-1$ 個目までのイベントは窓の中で発見されていて，窓の中に x 個目のイベントが入ってくるのを待っている，という意味である．$waits$ は，その 2 つ組を 0 個以上含む．$waits(A)$ 内の 2 つ組は，全てエピソードイベントとして A のイベントを待っており，A が窓の中に入ると更新される．

定義 13 $beginsat(t)$ は，時間 t で初期化されるエピソードを保持する．窓の外にそのエピソードイベントが出た際に，いくつの窓の中に，そのエピソードが入っていたかを知るために用いる．

アルゴリズムは図 4.5 のようになる．

Algorithm Recognizing-Episode$(\alpha, \mathbf{s}, win)$
入力: 候補エピソード α，イベント列 $\mathbf{s} = (s, T_s, T_e)$，窓幅 win.
出力: 与えられたエピソードが窓に出現する回数を表す整数型変数 $\alpha.freq\_count$.

1: $waits(\alpha[0]) = (\alpha, 1), \alpha.initialized = 0, \alpha.freq\_count = 0, i = 1$;
2: $waits(\alpha[0])$ 以外の $waits$ とイベント列先頭以前の win 分の $beginsat$ を空にする;
3: **for** $start = T_s; start < T_e + win; start = start + 1$ //窓のスライド **do**
4: **for all** イベント $(A, start) \in s$ **do**
5: **for all** $(\alpha, i) \in waits(A)$ **do**
6: **if** $i = |\alpha|$ 且つ $\alpha.initialized[i] = 0$(他にエピソード α が窓内にない状態) **then**
7: $\alpha.inwindow = start$;
8: **end if**
9: **if** $i < |\alpha|$ **then**
10: $waits(\alpha[i+1] = \cup(\alpha, i+1))$;
11: $\alpha.initialized[i] =$ その α の最初のイベントが窓に入った時間 t;
12: $beginsat(t) = \cup(\alpha, i)$;
13: $waits(A) = waits(A) \backslash (\alpha, i)$;
14: **end if**
15: **end for**
16: **end for**
17: **for all** $(\alpha, j) \in beginsat(start - win)$ **do**
18: **if** $j = |\alpha|$ **then**
19: $\alpha.freq\_count = \alpha.freq\_count - \alpha.inwindow + start$; //$\alpha$ があった窓の数だけ増分.

```
20:       else
21:          waits(α[j + 1]) = waits(α[j + 1])\(α, j + 1)
22:       end if
23:    end for
24: end for
```

図 4.5：候補直列エピソード頻出認識アルゴリズム

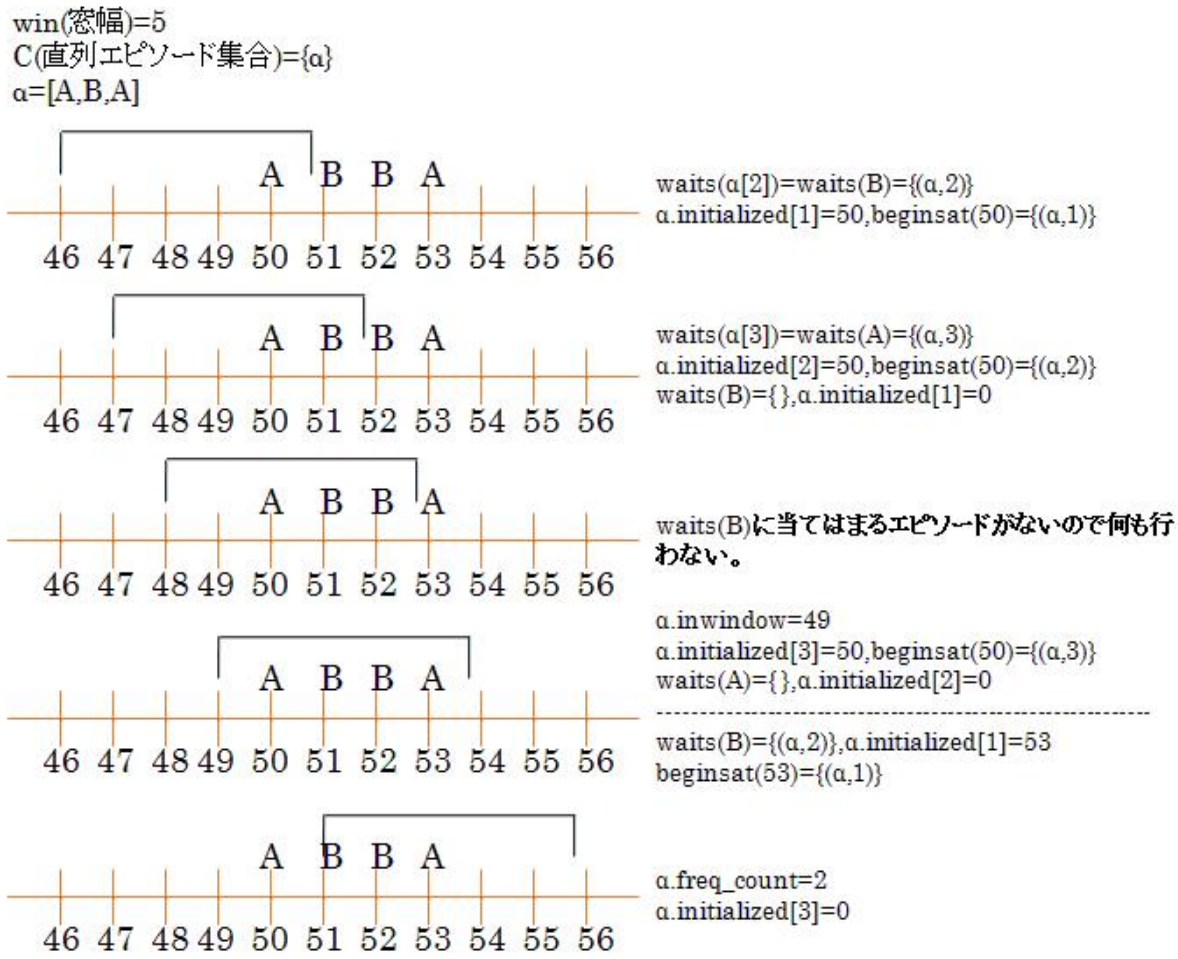

図 4.6:一つの候補エピソードに着目したときの頻度計算の流れ

4.4 計算量の解析

本節では，先程のアルゴリズムにかかる計算量について述べる．イベント列の長さと，一つの候補エピソードのサイズをそれぞれ n, l とし，サイズ l の頻出エピソード集合を $\mathcal{F}_l$，直列候補エピソード集合を C とすると，以下の 2 つの定理が得られる．

定理 1 候補エピソード生成アルゴリズムの時間計算量は $\mathcal{O}(l^2|\mathcal{F}_l|^2 log|\mathcal{F}_l|)$ である．

証明 まず，与えられたエピソード集合の各エピソードについて，ブロックの先頭を設定するため $\mathcal{O}(|\mathcal{F}_l|)$ かかる．次に，全てのエピソードから候補エピソードを作るので，$\mathcal{O}(|\mathcal{F}_l|)$ かかる．各エピソードについて，$l+1$ 文字目のイベントを作るために，所属するブロックの全てのエピソードを用いる可能性があるので，更に，$\mathcal{O}(|\mathcal{F}_l|)$ かかる．ここで一つのエピソードに着目すると，実際に $l+1$ 文字のエピソードを作り，更に，その $l+1$ 文字のエピソードから l 文字の部分文字列のエピソードを作るのに，$\mathcal{O}(l+1+(l-1)l) = \mathcal{O}(l^2)$ かかる．その部分文字列エピソードが，与えられたエピソード集合内にあるかを調べるのに 2 分探索を用いると，$\mathcal{O}(l \log |\mathcal{F}_l|)$ かかる．よって，総合時間計算量は，$\mathcal{O}(|\mathcal{F}_l| + |\mathcal{F}_l||\mathcal{F}_l|(l^2 + (l-1)l \log |\mathcal{F}_l|)) = \mathcal{O}(l^2|\mathcal{F}_l|^2 \log |\mathcal{F}_l|)$ となる．

定理 2 直列候補エピソード頻度計算アルゴリズムの時間計算量は $\mathcal{O}(n|C|l)$ である．

証明 前節の直列候補エピソード頻度計算アルゴリズムをご覧いただきたい．簡単のため与えられる候補エピソード集合中の，一つエピソードの場合について考える．1. の初期化にかかる時間は，$\mathcal{O}(l+win)$（全てのエピソードを対象にする場合は $|C|$ を乗算するが，同じイベント列を扱うので $\mathcal{O}(|C|l+win)$）となる．2. でイベント列内で窓をシフトするので，$\mathcal{O}(n)$ かかる．一つシフトする度に一つのイベントが窓に入り，一つのイベントが窓から出る．自明なことであるが，窓をシフトし終わったときに，$waits(\alpha[i])$（ただし $1 \leq i \leq l$）に該当したエピソードの総合量が多い程計算量はかかる．一つのエピソードについての総合量は，nl を超えることはないので，$\mathcal{O}(nl)$ となる．以上を全てのエピソード集合について考えて当てはめると，$\mathcal{O}(|C|l + win + n|C|l)$=$\mathcal{O}(n|C|l)$ となる．

第5章

実験

本章では，3章で示した頻出エピソード発見アルゴリズムを実装し，実験による性能評価を行った．使用したアルゴリズムは図5.1の通りである．

BFS-Naive	Naiveな幅優先探索
BFS-Naive2	候補エピソードを改良した幅優先探索
DFS-Naive	Naiveな深さ優先探索
BFS-MO1	右極小出現リストを用いた幅優先探索
BFS-MO2	候補エピソード生成を改良し，右極小出現リストを用いた幅優先探索
DFS-MO	右極小出現リストを用いた深さ優先探索

図5.1:使用したアルゴリズム

実験1と実験2では，DFS-MOについて，それぞれ頻出しきい値と窓幅を変えながら，求められる頻出なエピソードの個数を数えた．実験3と実験4では，全てのアルゴリズムについて，イベント列として与える文字列の長さを変えながら，アルゴリズムを実行してかかる時間，メモリを計測した．実験5では，BFS-MO1とDFS-MOについて，頻出しきい値を変えながらアルゴリズムを実行してかかる時間を計測した．実験6では，BFS-MO1とDFS-MOについて，窓幅を変えながらアルゴリズムを実行してかかる時間を計測した．これらのアルゴリズムは，全てC++言語で実装し，GNU g++でコンパイルした．全ての実験は，ノートPC (Genuine Intel(R) CPU U2400 1.06Ghz，RAM 1014MB，OS WindowsVista，Cygwin) 上で行った．

5.1 実験データ

GenBankが公開しているある生物種のDNA配列データを先頭から，20，50，100，250，500，1000，2000，3000，4000，5000，10000，20000，30000文字切り取ったものを元デー

タとして用いる．

5.2 実験結果

実験１．図 5.2 は，DFS-MO で長さ 5000 の文字列，窓幅 10 として，頻出しきい値を変えて得られる頻出エピソード数のグラフである．頻出しきい値が小さいほど，得られる頻出エピソード数も増えることが分かる．特に頻出しきい値 2 以下の得られるエピソード数の増加が著しい．

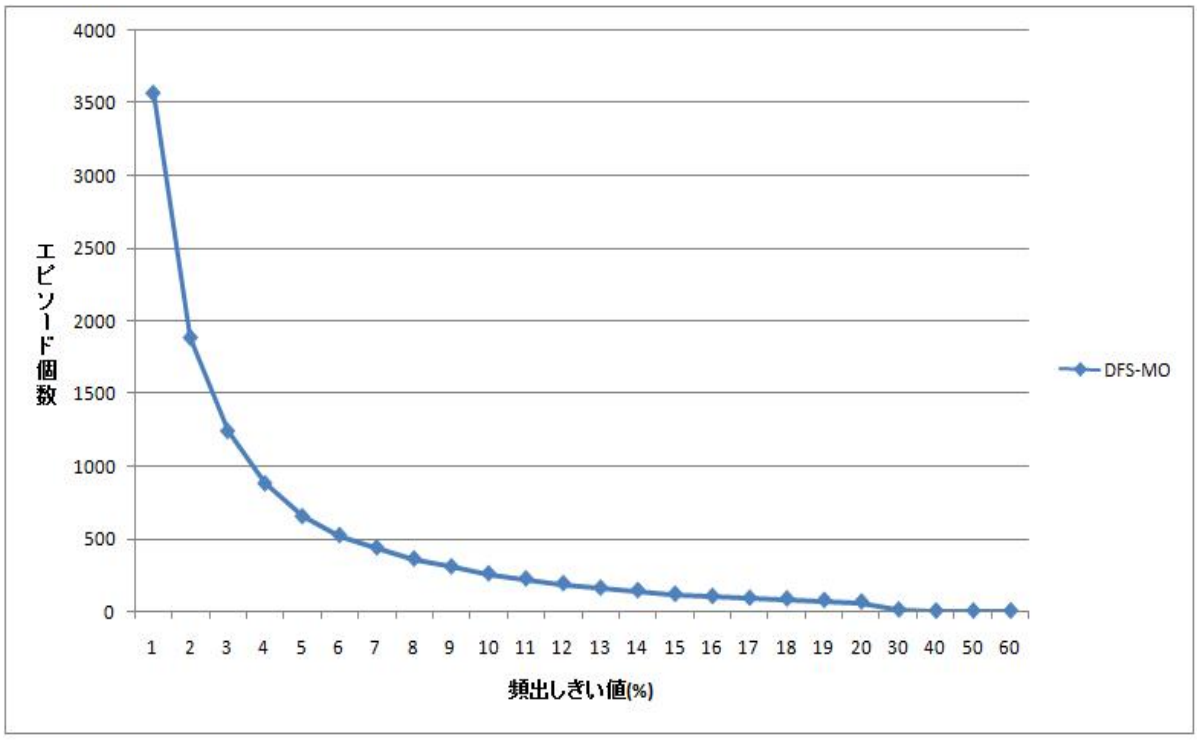

図 5.2:頻出しきい値に対して得られる頻出エピソード数

実験２．図 5.3 は，DFS-MO で長さ 5000 の文字列，頻出しきい値 5 として，窓幅を変えて得られる頻出エピソード数のグラフである．窓幅が大きいほど，得られる頻出エピソード数も増えることが分かる．特に窓幅 17 以上で得られるエピソード数の増加が著しい．

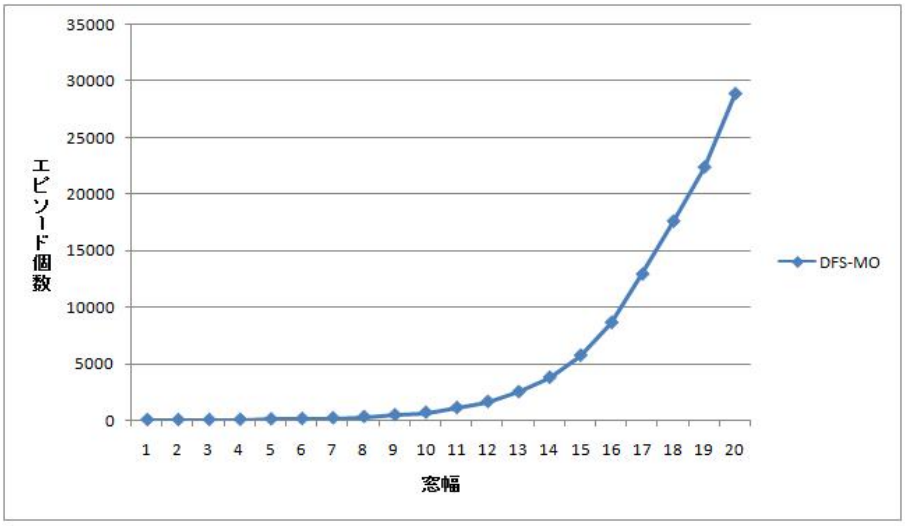

図 5.3:窓幅に対して得られる頻出エピソード数

実験 3. 図 5.4 は，全てのアルゴリズムについて頻出しきい値を 5，窓幅 10 として，テキスト長を変化させた時の計算時間のグラフである．更に図 5.5 は，その中でも BFS-MO1 と BFS-MO2，DFS-MO に絞ったものである．この実験より，テキストが長ければ長いほど，右極小出現リストを用いたアルゴリズムのほうが高速であることが分かる．さらに，右極小出現リストを用いた場合では，幅優先探索より深さ優先探索のほうが若干高速である．Naive なアルゴリズムの中では，不要な候補エピソードの頻度を計算せずに済む可能性のある BFS-Naive2 が高速である．

実験 4. 図 5.6 は，10000 文字と 1000 文字のデータサイズに対して，全てのアルゴリズムについて頻出しきい値を 5，窓幅 10 として，テキスト長を変化させた時の，使用メモリ (プライベートワーキングセット) のグラフである．Naive なアルゴリズムではあまり差は出ないが，右極小出現リストを用いたアルゴリズムは Naive よりも総じてメモリの使用量が大きい．中でもエピソードごとの右極小出現リストを持つ幅優先探索では，扱う文字列が長いほど，かなりメモリを消費していることが分かる．

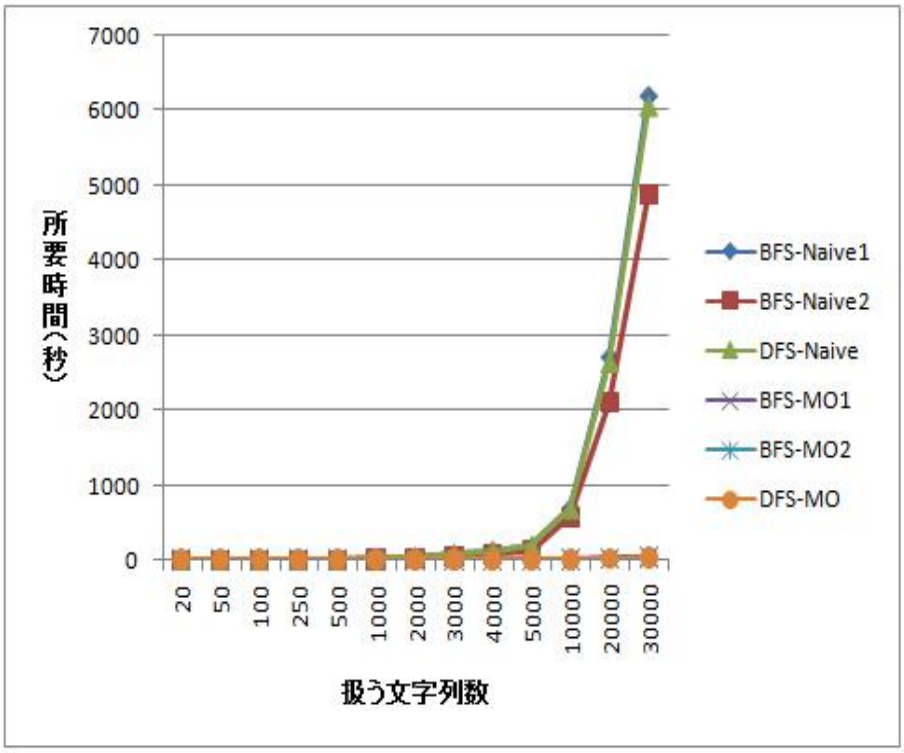

図 5.4:テキスト長に対する全てのアルゴリズムについての計算時間

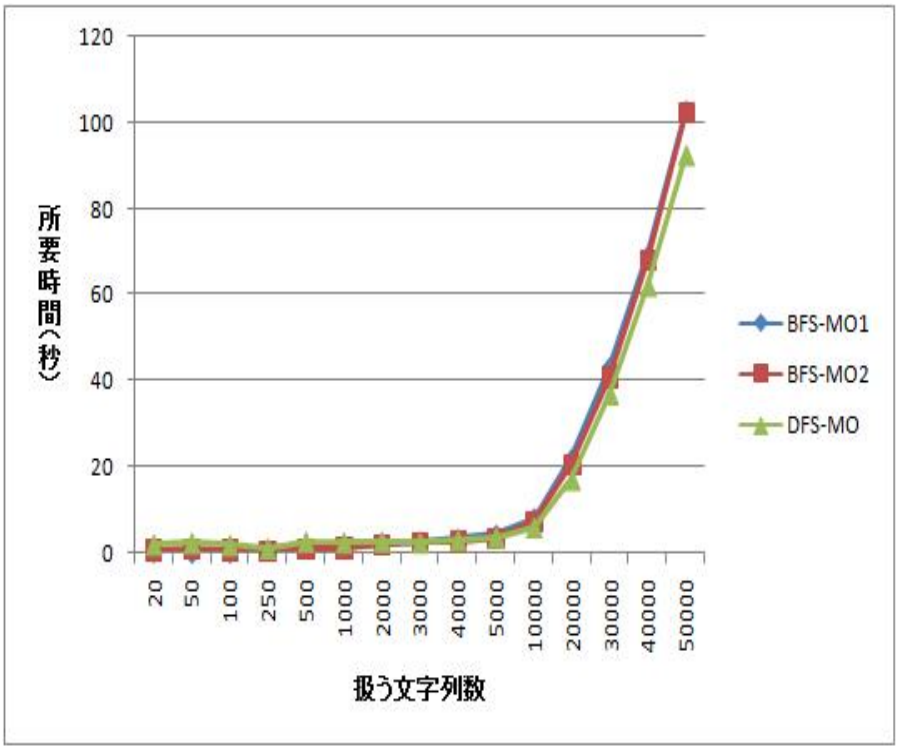

図 5.5:テキスト長に対する右極小出現リストを用いたアルゴリズムについての計算時間

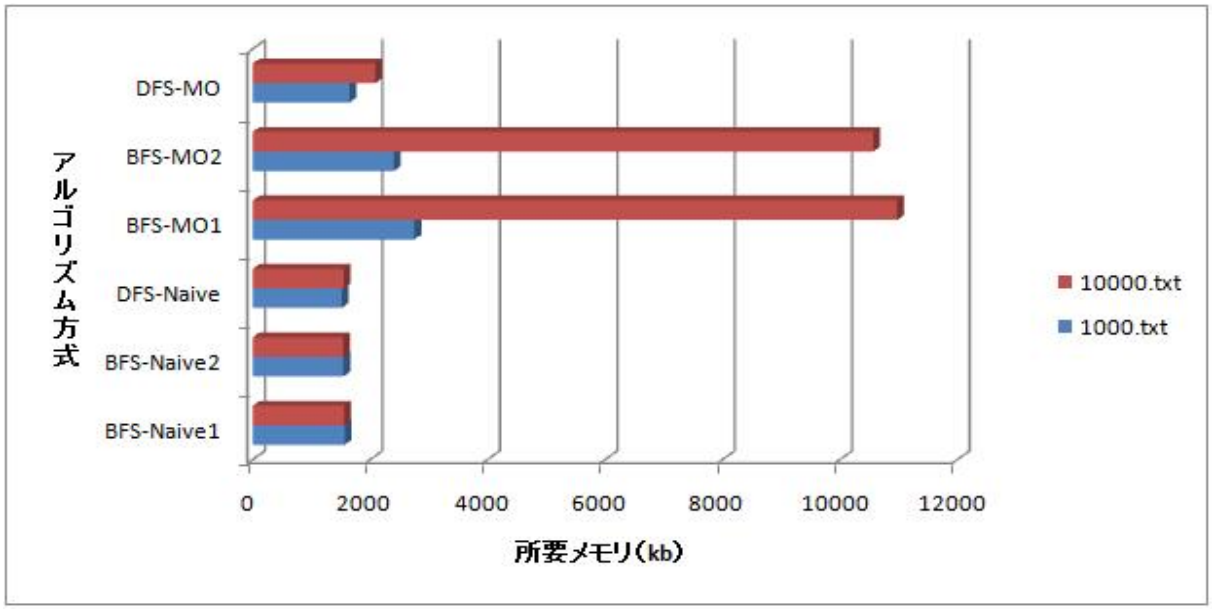

図 5.6:テキスト長に対する全てのアルゴリズムについての使用メモリ

実験 5. 図 5.7 は，BFS-MO1 と DFS-MO について長さ 5000 の文字列，窓幅 10 として，頻出しきい値を変えながらアルゴリズムを実行してかかる時間を計測したグラフである．頻出しきい値 4 以上の時は，頻出と判定されるエピソードがあまり多くなく，単純に処理を行う深さ優先探索の方が計算速度が速い．しかし，頻出エピソードが多くなるしきい値 3 以下の時は，BFS-MO1 の方が計算速度が速い．

実験 6. これに対し図 5.8 は，BFS-MO1 と DFS-MO について長さ 5000 の文字列，頻出しきい値 5 として，窓幅を変えながらアルゴリズムを実行してかかる時間を計測したグラフである．窓幅を大きくすると，BFS-MO1 と DFS-MO とでは，処理が単純な分 DFS-MO の方が高速である．

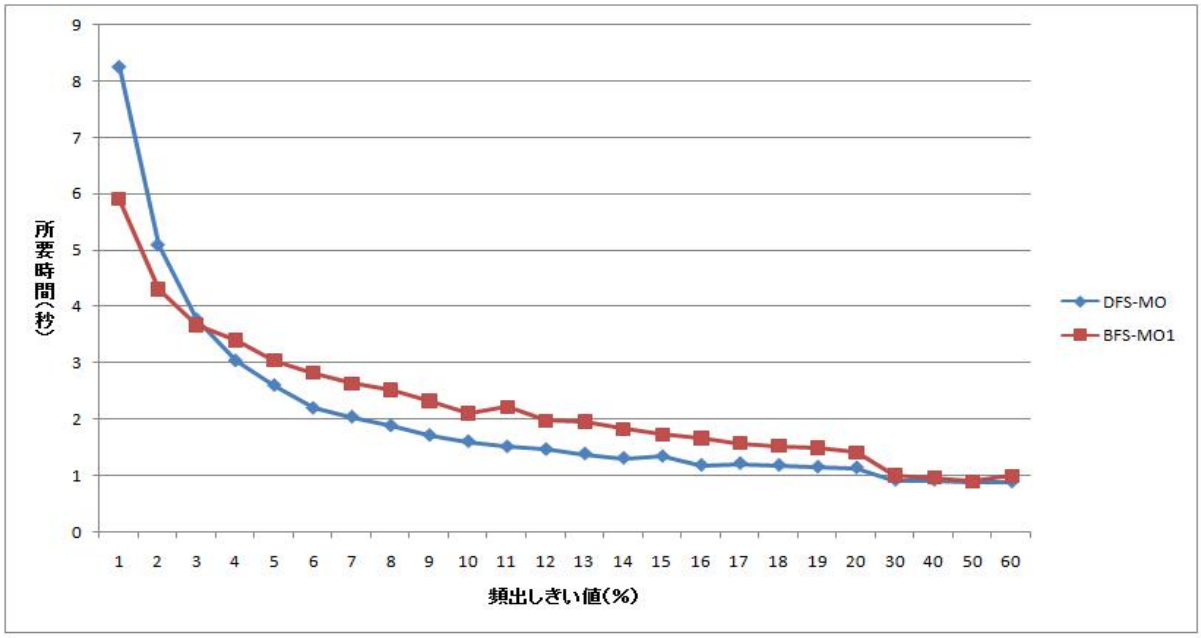

図 5.7:頻出しきい値に対する右極小出現リストを用いたアルゴリズムについての計算時間

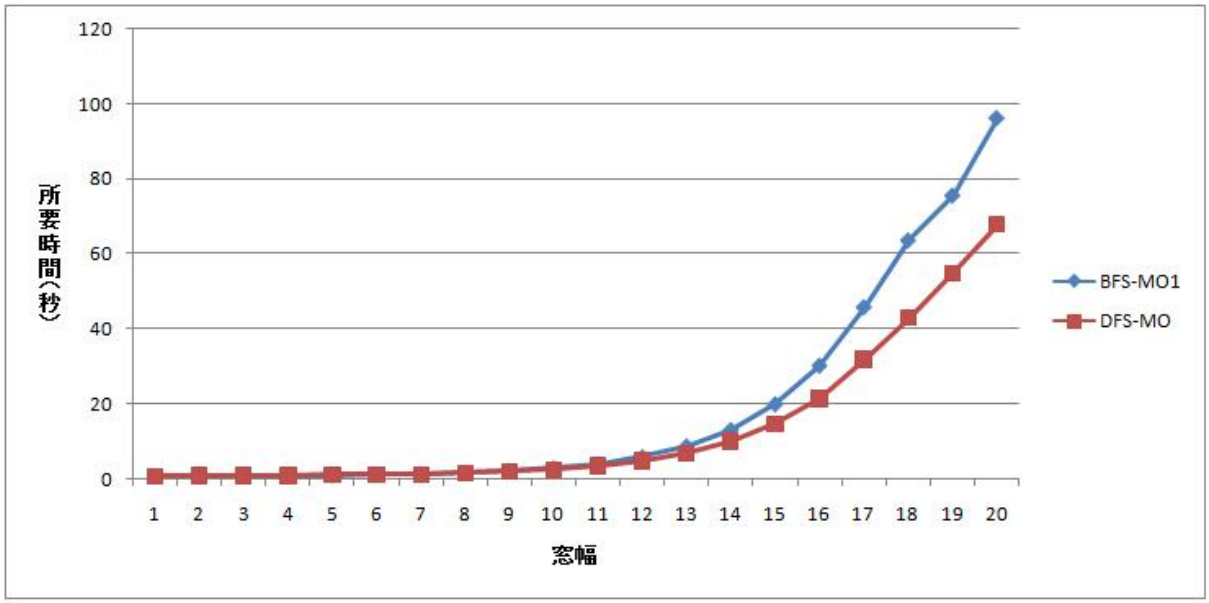

図 5.8:窓幅に対する右極小出現リストを用いたアルゴリズムについての計算時間

第6章

結論

本稿では，テキストデータから窓を用いて頻出な直列エピソードを発見するアルゴリズムについていくつか紹介した．

初めに，幅優先探索と深さ優先探索を用いて，Naive に頻出直列エピソードを見つける方法について説明した．次に，効率よく頻出エピソードを発見するために，深さ優先探索で，右極小出現リストを用いたアルゴリズム DFS-MO を与えた．そのアルゴリズムのサブルーチン Update と WinCount を用いて幅優先探索に適用することも可能である．実験により幅優先探索，深さ優先探索共に，Naive なアルゴリズムよりも，右極小出現リストを用いた方が高速であることが分かった．さらに，右極小出現を用いるアルゴリズムの中では，深さ優先探索である DFS-MO が，窓幅や頻出しきい値を変化させても，速度の面でもメモリの面でも最も安定して効率がよいことが分かった．幅優先探索である BFS-MO は，エピソードごとに右極小出現リストを持つため，頻出エピソードが多いほどメモリ使用量は増大してしまう．しかし，実験 5. の結果を見ると，速度という面では DFS-MO よりも高速になる条件があることもわかった．

次に，Mannila らが提案した幅優先探索のアルゴリズム BFS-MTV を説明した．特に，実験 3. により提案アルゴリズム中の，候補エピソード作成部分 Candidate-Episode が Naive なものよりも有効であることが分かった．

今後の課題として，より詳細な実験的評価とこのアルゴリズムを音楽等時系列テキストに用いる方法の模索，Mannila らの Recognizing-Episode による高速化が挙げられる．

謝辞

お忙しい中この研究に多大なご助力をくださった，北海道大学大学院情報学研究科有村博紀教授，喜田拓也准教授，南野香子さんに感謝します．有村教授には論文の添削や，プログラムの方針についてのご指導等大変お世話になりました．プログラム作成にあたっての方針の立て方は，とても助けになると共に感心させられるばかりです．喜田准教授には論文添削でお世話になりました．論文を書くのが初めての私にとって，完成への道筋を与えてくださったことに感謝します．またアルゴリズム作成過程でアドバイスくださったり，とてもよい雰囲気で研究に臨ませてくださった研究室の方々に感謝します．両親には故郷から私の学生生活を温かく支援していただきました．深く感謝いたします．重ねて，お世話になった皆様に厚く御礼申し上げます．ありがとうございました．

参考文献

[1] Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley Publishing Company, 1974.

[2] 平田富夫, アルゴリズムとデータ構造—改訂 C 言語版—, 森北出版株式会社, 108–112, 2002.

[3] Heikki Mannila, Hannu Toivonen, A. Inkeri Verkamo, Discovery of Frequent Episodes in Event Sequence, *Data Mining and Knowledge Discovery*, Vol. 1, 259-289, 1997.

[4] 斎藤伸自, 西関隆夫, 千葉則茂, 離散数学, 朝倉書店，27–34, 1989.

[5] 内田雄三, 窓ありエピソードと近似エピソードに対する高速な頻出系列パターン発見アルゴリズム, 九州大学大学院システム情報科学研究科, 修士論文, 2004.