Modified PrefixSpan 法を用いた頻出正規パターンの抽出をめざして

塔野 薫隆† 北上 始‡ 田村 慶一‡ 森 康真‡ 黒木 進‡

†広島市立大学大学院 情報科学研究科 〒731-3194 広島市安佐南区大塚東三丁目４番１号
‡広島市立大学 情報科学部 〒731-3194 広島市安佐南区大塚東三丁目４番１号
E-mail: {tono,kitakami,ktamura,mori,kuroki}@db.its.hiroshima-cu.ac.jp

あらまし 著者らは,ある配列データベースから可変長ワイルドカード領域が含まれている頻出パターンを抽出するために，新 *Modified PrefixSpan* 法を提案する．この新しい方式は，既に提案した *Modified PrefixSpan* 法に，最大誤差数と呼ばれる入力パラメータを追加することによって開発されている．新方式の有効性を調べるために，PROSITE から Leucine Zipper と Zinc Finger をそれぞれ含む 2 種類のデータセットを取り出し，可変長ワイルドカード領域が含まれる頻出パターンを抽出する能力の評価を行った．その結果，本提案方式が従来方式に比べて 8〜9 倍の優れた抽出能力を持っている事を確認した.

キーワード データマイニング，バイオインフォマティックス，知識発見，知識処理，性能評価

Mining of Sequential Patterns with Variable Wildcard Regions using *Modified PrefixSpan* Method

Shigetaka TONO† Hajime KITAKAMI‡ Keiichi TAMURA‡ Yasuma MORI‡ Susumu KUROKI‡

†Graduate School of Information Sciences,

‡Faculty of Information Sciences,

Hiroshima City University, 3-4-1 Ozuka Higashi, Asaminami-ku, Hiroshima 731-3194 Japan
E-mail: {tono,kitakami,ktamura,mori,kuroki}@db.its.hiroshima-cu.ac.jp

Abstract In order to extract frequent patterns with a “variable wildcard region” from a sequence database, we propose the new *Modified PrefixSpan* method. The new method is enabled to develop by adding a new input parameter, called a maximum error count, to the former *Modified PrefixSpan* method. We verify this new method with 2 kinds of datasets, Leucine Zipper and Zinc Finger that are included in PROSITE. The results show that this new method has 8 to 9 times superior capacity for extraction of frequent patterns.

Keyword Data Mining，Bioinformatics，Knowledge Discovery，Knowledge Management，Performance Evaluation

1. はじめに

配列データベースから頻出パターンを抽出する方法を用いると，多くの問題に答えることができる．この方法は, DNA やアミノ酸の配列データだけではなく，顧客の購買履歴，Web アクセス履歴，科学的な実験データ，病気の治療履歴，自然災害の履歴などを分析するのに有用であるといわれている．我々は，これまでに分子生物学の分野におけるモチーフ抽出の問題に着目してきた．現在，さまざまなモチーフが見つかっている．各モチーフはあるアミノ酸配列データベース中に存在する特別な配列パターンであり，多様な蛋白質が持つ機能の 1 つに関係し，生物の進化の過程で保存されてきたものであると考えられている．

このようなモチーフを配列データベースから見つけるのを支援するために，古くから，マルチプルアラインメントを計算機上で行う方法の研究が行われている．これは，1970 年の Needlman and Wunsch から始まるアラインメント処理の研究として知られている[1][2]. 多くの研究は，配列データベース中から最適な類似領域をもとめるマルチプルアラインメントに焦点が当てられた[3]. しかし，アラインメント処理は，計算量が非常に大きいことや，配列間で似たような場所にある類似領域だけを相手にしているなどの問題点が指摘された.

その問題点を解決するために，MEME[4]では，アラインメントされていない n 本の配列集合からおおまかな情報としてモチーフの長さ W と各配列中の位置を利用者に指定させ，その部分に対してマルチプルアラインメントを行った後，統計的手法である期待値最大化アルゴリズムを M 回繰り返すことにより，複数の頻出パターンを抽出している．これにより，MEME では，時間計算量を従来の方法に比べ大幅に抑えているが，抽出される頻出パターン中にワイルドカードを含まないため，PROSITE[5][6]や Pfam[7]などで見られるモチーフとは直接結びつきにくいという問題がある．

本論文では，モチーフ発見を支援するために，アミノ酸の配列データベースからさまざまなワイルドカード領域をもつ頻出パターンの抽出方法を提案する．ワイルドカード領域には，固定長と可変長のワイルドカード領域の 2 種類があるが，著者らは *PrefixSpan* 法を拡張し，両者を含む頻出パターンを抽出する． Zinc Finger モチーフと Leucine Zipper モチーフのそれぞれを含む2種類のデータセットを PROSITE から取り出し，それぞれのデータセットを配列データベースとして利用し，固定長ワイルドカード領域を抽出する処理と可変長ワイルドカード領域を抽出する処理の性能比較をおこなう．

以下，2 章では，本提案方式に関する関連研究について述べる．3 章では，配列データベースから頻出パターンを抽出する問題およびそこで使用する記号の説明を行う．4 章では，従来，提案されていた固定長ワイルドカード領域を抽出する *Modified PrefixSpan* 法について紹介する．5 章では，可変長ワイルドカード領域が含まれる頻出パターンを抽出する新方式を提案する．6 章では，新方式の性能評価を行う．最後の 7 章で，本論文のまとめを行う．

2. 関連研究

固定長ワイルドカード領域を考慮し，配列データベース内に頻繁に現れるパターンの抽出に関する研究については数多く行われた[8]．しかし，当初提案されたアルゴリズムの多くは考えられる全パターンを生成し検証する方法が採用されたため効率がわるかった．その効率を改善するため，ヒューリスティクスとして頻出パターンに構造的な制限を与える方法が提案された．頻出パターンとは，あるパターンを含む配列データの件数が支持数 *min_sup* 以上のとき，そのパターンをさす．構造的な制限を与えるヒューリスティクスとして，利用者に頻出パターンの最大長[9]を与えさせる例がある．しかし，それを超える長いパターンを見逃してしまうという問題点がある．

その問題点を解決するために，TEIRESIAS アルゴリズムでは，アルファベット文字数 L とパターン長 W（$L \leqq W$）をもつ L-頻出パターンを一旦すべて求め，それらから固定長ワイルドカード領域を含む最長の頻出パターンを全て構成する方法がとられている[10][11][12]．このアルゴリズムでは，長さが W の基本頻出パターンの中にワイルドカードの総数が最大 W-L 個までの混在を許している．しかしながら，TEIRESIAS は任意の 2 つのアルファベット文字間に許されるワイルドカード数の最大値を直接指定する機能を持たない．さらに，アルファベット文字数 L の選び方に指導原理的なものがなく，最長の頻出パターン以外の頻出パターンをすべて見つけることが計算の中心になっていない．このような事情により，必ずしも使いやすいとは言えない．

PrefixSpan 法[13]は，配列データベースから頻出パターンを抽出するアルゴリズムとして知られているが，抽出過程では配列の軸位置から最左端位置までの広い範囲を参照しなければならないため，多大な計算時間を要し，無意味な頻出パターンが数多く抽出されてしまうという問題がある．また，ワイルドカード x が明記された頻出パターンを抽出する機能を持たないため，正規表現で表されるモチーフを表現できないという問題がある．例えば，頻出パターンの抽出過程において，<AC>, <A*x*C>と<A*xx*C>は，単に<AC>と解釈されてしまうので，モチーフ抽出問題には不向きである．

これらの問題を解決するために，*PrefixSpan* 法を改良した *Modified PrefixSpan* 法[14]では，頻出パターン中に複数存在する固定長ワイルドカード領域の最大長 V の設定を可能にし，頻出パターンの抽出過程において，<AC>, <A*x*C>と<A*xx*C>を異なるパターンとして区別する機能を持たせた．この方法は，L-頻出パターンを作成しないため，アルファベット文字数 L の指定が不要である．以上により，*Modified PrefixSpan* 法では利便性が向上している．

しかしながら，PROSITE に登録されているモチーフには，固定長ワイルドカード領域の他に可変長ワイルドカード領域をもつモチーフが数多く存在するにもかかわらず，可変長ワイルドカード領域を自動的に抽出する研究が行われていなかった．本論文では，*Modified PrefixSpan* 法を拡張し，可変長ワイルドカード領域を含む頻出パターンの抽出する方法について提案している．

3. 問題の形式

配列データベース $S=\{S_1,S_2,S_3,\ldots,S_n\}$ の各配列 S_i は，あるアルファベット Σ 上で定義されるとする($1 \leqq i \leqq n$)．例えば，DNA の場合は，以下の 4 文字のアルファベット Σ_{DNA} で定義される配列である．

Σ_{DNA}={a,t,c,g}

蛋白質の場合は，以下の 20 文字のアルファベット $\Sigma_{protein}$ 上で定義される配列である．

$\Sigma_{protein}$={A,C,D,E,F,G,H,I,K,L,M,N,P,Q,R,S,T,V,W,Y}

Π を Σ 上で定義された曖昧文字の集合，X をワイルドカード文字列の集合，{*}を任意長のワイルドカードを表す文字列を要素とする集合とする．PROSITE のパターンは，$(\Sigma \cup \Pi \cup X \cup \{*\})$上の正規表現として書かれている[9][15]．我々は抽出するパターン表記として，PROSITE のパターン表記を用いる．

ここでは，あるパターンが利用者によって与えられた最小支持数 *min_sup* 以上の配列にマッチするとき，そのパターンを頻出パターンと呼ぶ．頻出パターンを構成する文字要素の数が k 個のとき，その頻出パターンを k-頻出パターンと呼び，pat^k で表現する．配列データベース S から抽出対象とする k-頻出パターン pat^k は，以下の正規表現とする．

$$pat^k = <A_1\text{-}x(i_1,j_1)\text{-}A_2\text{-}x(i_2,j_2)\text{-}\ldots\text{-}x(i_{k-1},j_{k-1})\text{-}A_k>$$

ここで，<>内のハイフン(-)は，隣同士がお互いに連続していることを表現するための記号であり，その記号を省略することがある．A_i を文字要素または単に要素と呼ぶ．また，文字要素 A_i が，1 文字で表現されるとき単一要素，2 文字以上で表現（たとえば[ILVF]など）されるとき曖昧要素と呼ぶ．$x(i,j)$の領域において，$0 \leqq i \leqq j$ のとき，その領域を可変長ワイルドカード領域と呼ぶ．$i=j$ のとき，それを固定長ワイルドカード領域と呼び，この領域を $x(i)$で簡略表現することがある．以下では，固定長ワイルドカード領域だけを含むパターンを基本パターンと呼ぶ．また，$x(0,\infty)$は，任意長のワイルドカード文字列(*)と同じ意味である．パターン pat^k にマッチする部分配列の最大長を $L(pat^k)$で表現すると，$L(pat^k)$は，$L(pat^k)=k+\sum j_n$ $[1 \leqq n \leqq k\text{-}1]$ で表される．

本論文では，配列データベース S から可変長ワイルドカード領域が含まれる頻出パターンの集合 P を抽出する方法に着目する．これによって抽出された各頻出パターンの形式は複数の基本パターンを 1 つにまとめられた形式を持つ．例えば抽出された頻出パターンの形式が<F-x(2,5)-A>の様であれば，これは，<F-x(2)-A>，<F-x(3)-A>，<F-x(4)-A>，<F-x(5)-A>の 4 つの基本パターンが含まれている事を表す．頻出パターン P を抽出するためには，最小支持数 *min_sup*，最大誤差数 ε_{max}，最大ワイルドカード数 *max_wc* を必要とする．支持数 cnt_rを持つ k-頻出パターン pat_r^kを$<pat_r^k>: cnt_r$で表現し，m 個の k-頻出パターンが配列データベース S から抽出されたとすると，それらの集まりとする集合 P_k は以下のように表現される．

$$P_k = \{<pat_1^k>:cnt_1, <pat_2^k>:cnt_2, \ldots, <pat_m^k>:cnt_m\}$$

S から抽出される頻出パターンの最大要素数を q とすると，P は$\{P_1, P_2, \ldots, P_q\}$で表現される．$(k+1)$-パターンの最右端文字は，$k$-頻出パターンに対して，$\Sigma$ 上の 1 文字を追加することにより構成され得る．その 1 文字は，S_i において，k-頻出パターン pat_r^k の最右端文字よりも右側の位置に存在する可能性がある．その右側の位置を表現するために，以下の $PDB(pat_r^k)$を導入する．

$PDB(pat_r^k)$ = { (i, j) | S_i において，pat_r^k の最右端文字の右隣位置を j とする }. ただし, $1 \leqq j \leqq \| S_i \|$ である．以後，$PDB(pat_r^k)$を pat_r^k の射影データベース(Projected Database)と呼ぶ．

4. *Modified PrefixSpan* 法

PrefixSpan 法[13]を拡張した *Modified PrefixSpan* 法[14]は，固定長ワイルドカード領域が含まれる頻出パターンを抽出する方法である．

表 1 に示される配列データベース S から固定長ワイルドカード領域が含まれる頻出パターンの抽出方法を示そう．ただし，最小支持数 *min_sup* および最大ワイルドカード数 *max_wc* を各々3 とする．また，表 1 は，$S=\{S_1,S_2,S_3,S_4,S_5\}$であり，$S_1$=FKYAKWL，$S_2$=SFVKTA，$S_3$=ALR，$S_4$=MSKPL，$S_5$=FSKFLMAW であることを示している．配列データベース S から頻出パターンを抽出する処理は，k-頻出パターンから$(k+1)$-頻出パターンを作成することによって行われる$(k \geqq 1)$. この処理を k=1 と k=2 の場合について考えてみよう．

表 1．配列データベース

ID	Sequence
1	FKYAKWL
2	SFVKTA
3	ALR
4	MSKPL
5	FSKFLMAW

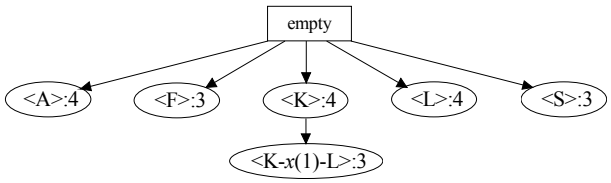

図 1．従来方式で得られたマイニング木

k=1 のとき，配列データベース中に存在するパターンは，<A>:4, <F>:3, <K>:4, <L>:4, <M>:2, <P>:1, <R>:1, <S>:3, <W>:2, <V>:1, <Y>:1 である．これらにより，最小支持数を満たす 1-頻出パターンを選び出すと，その集合は図 1 に表されているように，P_1 ={ <A>:4, <F>:3, <K>:4, <L>:4, <S>:3 }になる．例えば，1-頻出パターン

<A>:4，<F>:3，<K>:4 の射影データベースは，各々，*PDB*(<A>)={(1,5),(2,*end*),(3,2),(5,8)},*PDB*(<F>)={(1,2),(2,3),(5,2),(5,5)},*PDB*(<K>)={(1,3),(1,6),(2,5),(4,4),(5,4)}で表現される．*PDB*(<A>)内の(1,5)は，パターン<A>が 1 番目の配列 S_1 の 4 文字目にあり，S_1 には<A>から始まる 2-パターンの最右端文字は 5 文字目以降に存在する可能性があることを意味する．また，*PDB*(<A>)内の(2,*end*)は<A>が 2 番目の配列 S_2 の最右端文字にあり，S_2 には<A>から始まる 2-パターンの最右端文字は存在しないことを意味する．

k=2 のとき，2-頻出パターンの集合 P_2 は P_1 中の各 1-頻出パターンから構成することができる．例えば，P_1 中の<A>の次にくる 1 文字は，<A>の射影データベース *PDB*(<A>)={(1,5),(2,*end*),(3,2),(5,8)}を用いて選択され得る．それらを集めた集合は{S_1[5],S_3[2],S_5[8]}={<K>,<L>,<W>}になるが，その中で P_1 に含まれていない文字は頻出ではない．従って，2-頻出パターンの最右端文字は<K>, <L>の 2 つの文字に絞られるが，残念なことに<A-K>:1, <A-L>:1 である．また，1～3 文字のワイルドカードを含む文字列<A-*x*(1)>, <A-*x*(2)>, <A-*x*(3)>のどれを考えても，その次にくる 1 文字をさがし 2-頻出パターンを構成することができない．以上から，<A>から始まる 2-頻出パターンは存在しない（図 1）．

次に，P_1 中の<K>から始まる 2-頻出パターン（ワイルドカード数がゼロの場合）について考えてみよう．<A>の場合と同様に，<K>の次にくる 1 文字は<K>の射影データベース *PDB*(<K>)を用いて選択される．それらの中で P_1 に含まれる文字は{S_2[5],S_5[4]}={<T>, <F>}である．従って，<K-T>:1,<K-F>:1 となるので，ワイルドカード数をゼロとする限り，2-頻出パターンは存在しない．次に，1 文字のワイルドカードを含む 2-頻出パターンについて考えてみよう．<K-*x*(1)>の次にくる 1 文字は *PDB*(<K-*x*(1)>)={(1,3+1),(1,6+1),(2, 5+1),(4, 4+1),(5,4+1)}を計算することにより選択される．それらの中で P_1 に含まれる文字は，{S_1[4],S_1[7],S_2[6],S_4[5],S_5[5] }={<A>,<L>,<A>,<L>,<L>}である．これにより，<K-*x*(1)-A>:2, <K-*x*(1)-L>:3 を構成することができる．しかし，2 文字および 3 文字のワイルドカードを含む文字列<K-x(2)>, <K-x(3)>のどちらを考えても，もはや 2-頻出パターンは存在しない．以上により，<K>から始まる 2-頻出パターンは，<K-*x*(1)-L>:3 だけとなる（図 1）．3-頻出パターンを構成するために必要な<K-*x*(1)-L>の射影データベース PDB(<K-*x*(1)-L>)は{(1,*end*),(4,*end*),(5,6)}となるので，この 2-頻出パターンから 3-頻出パターンを構成することができない．

5. 新 *Modified PrefixSpan* 法

新 *Modified PrefixSpan* 法の処理では，前章で説明した入力パラメータの他に，最大誤差数 ε_{max} を必要とする．最大誤差数がゼロのとき，従来の *Modified PrefixSpan* 法と同じである．また，*k*-頻出パターン<pat^k>から可変長ワイルドカード領域が含まれる(*k*+1)-パターンを抽出するときに，同じ最右端文字 α をもつ以下のパターンが数多く見つかる．

<pat^k-*x*(*i*,*i*+ ε_1)-α>:$cnt_{\varepsilon 1}$,<pat^k-*x*(*i*,*i*+ ε_2)-α>:$cnt_{\varepsilon 2}$，…

$\varepsilon_1 \leqq \varepsilon_2$ （$\leqq \varepsilon_{max}$）のとき，明らかに，$cnt_{\varepsilon 1} \leqq cnt_{\varepsilon 2}$ であり，<pat^k-*x*(*i*,*i*+ ε_1)-α>:$cnt_{\varepsilon 1}$ は，<pat^k-*x*(*i*,*i*+ ε_2)- α>:$cnt_{\varepsilon 2}$ に含まれる表現である．従って，このような同じ最右端文字のパターンに対しては，我々は，最も一般的な形式のパターンだけを候補パターンとして選び出している．これにより，冗長性のない(*k*+1)-頻出パターンを抽出している．

以下，表 1 で示される配列データベースの例を用いて，可変長ワイルドカード領域が含まれる頻出パターンを抽出する処理について考えてみよう．ただし，最小支持数 *min_sup*，最大ワイルドカード数 *max_wc*，最大誤差数 ε_{max} を各々3 とする．配列データベース *S* から頻出パターンを抽出する処理は，前章と同じであり，*k*-頻出パターンから(*k*+1)-頻出パターンを作成することによって行う(*k*≧1)．*k*=1 に対応する 1-頻出パターンの構成方法は，図 2 に示されるように，従来の *Modified PrefixSpan* 法と同じである．*k*=2 に対して，従来の方法では，<F>から始まる 2-頻出パターンは抽出できなかったが，新方式ではそれが可能になっている．

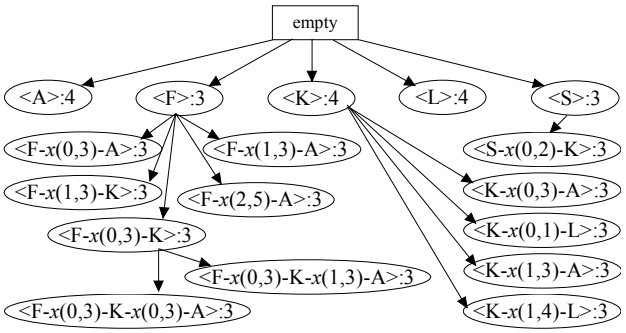

図 2．提案方式で得られたマイニング木

以下では，2-頻出パターン *pat*=<F-*x*(0,3)-K>に着目し，この 2-頻出パターンから 3-頻出パターンを生成する方法について考えてみよう．図 3 に示されるように，ワイルドカード数 *wc* が 0 と 1 に対応する 3-頻出パターンが生成されるが，ワイルドカード数 *wc* が 2 と 3 に対応する 3-頻出パターンは生成されない．

例えば，2-頻出パターンを *pat*=<F-*x*(0,3)-K>とし，その 2-頻出パターンからワイルドカード数 *wc* が 1 の 3-頻出パターンを見つけてみよう．このために，誤差 ε

(a)

ε / ID	0	1	2	3
1	*pat-x*-A *pat-x*-L	*pat-xx*-K	*pat-xxx*-W	*pat-xxxx*-L
2	*pat-x*-A	-	-	-
3	-	-	-	-
4	-	-	-	-
5	*pat-x*-L	*pat-xx*-M	*pat-xxx*-A	*pat-xxxx*-W

(b)

ε / α	0	1	2	3
<A>	<*pat-x*(1,1)-A>:2		<*pat-x*(1,3)-A>:3	
<F>				
<T>				
<K>		<*pat-x*(1,2)-K>:1		
<L>	<*pat-x*(1,1)-L>:2			<*pat-x*(1,4)-L>:2
<M>		<*pat-x*(1,2)-M>:1		
<W>			<*pat-x*(1,3)-W>:1	<*pat-x*(1,4)-W>:2

図 3．<*pat-x*(1,1+ ε)-α>:*cnt* 形式の 3-頻出パタンの計算（0≦ ε ≦ ε_{max}）

を最小値 0 から最大値の 3 まで変化させる．即ち，0 ≦ ε ≦3 とし，3-頻出パターン<*pat-x*(1,1+ ε)-α>:*cnt* の最右端文字 α および支持数 *cnt* を計算する方法について考える．この 3-パターンは，4 つの基本パターン<*pat-x*(1,1)-α>,<*pat-x*(2,2)-α>,<*pat-x*(3,3)-α>,<*pat-x*(4,4)-α>をまとめた表現である．各基本パターンの支持数は，誤差 ε を 0 から 3 まで順に変化させながら，誤差 ε ごとに，同じ α をもつ 3-パターンが異なるタプルに何回出現するかを正確に数え上げることで計算できる．この様子を図 3 の(a)と(b)に示す．

図 3 で，*pat*=<F-*x*(0,3)-K>から始まる 3-パターンの最右端文字 α が<A>の場合について考えてみよう．図 3 の(a)で誤差 ε が 0 のとき，3-パターン<*pat-x*-A>は，配列データベース中に 2 箇所存在するが，それらは，お互いに異なるタプル（1 番目と 2 番目）に含まれているので，図 3 の(b)に示されるように<*pat-x*(1,1)-A>:2 となる．図 3 の(a)で誤差 ε が 1 のとき，最右端文字が<A>であるパターン<*pat-xx*-A>はどのタプルにも存在しないが，誤差 ε が 2 のとき，<*pat-xxx*-A>はそれらのタプルとは異なる 5 番目のタプルに含まれている．従って，図 3 の(b)に示されるように，ここまでの計算では，<*pat-x*(1,1)-A>:2 かつ<*pat-xxx*-A>:1 であるので，<*pat-x*(1,3)-A>の支持数は 3 になる．最後に，誤差 ε が 3 のときは，最右端文字が<A>で終わる 3-パターン<*pat-xxxx*-A>は存在しない．以上から，*pat*=<F-*x*(0,3)-K>から始まる 3-パターンは，<*pat-x*(1,1)-A>:2，<*pat-x*(1,3)-A>:3 の 2 つ存在することになるが，後者は前者を包含するので，前者を除外する．従って，ワイルドカード数を 1 に固定したとき，*pat*=<F-*x*(0,3)-K>から始まる 3-頻出パターンは，<F-*x*(0,3)-K-*x*(1,3)-A>:3 となる．

ワイルドカード数を 1 に保持したままで，他の候補パターンについて考えると，図 3 の(b)に示されるように 6 個の 3-パターンが見つかる．そして冗長なパターンとして，<*pat-x*(1,1)-L>:2 および<*pat-x*(1,3)-W>:1 が除外され，4 個の 3-パターンが残る．しかし，4 個のいずれも最小支持数を満たさないので，他の 3-頻出パターンは存在しない．

以上の処理手順の例を一般化した処理を図 4 に示す．手続き *EnumerateSup* 中の *CandidateCharSet* には，(*k*+1)-パターンの最右端文字 α，誤差 ε，そのときの支持数 *sup_cnt'*から成る 3 項関係(α， ε , *sup_cnt'*）が格納される． 例えば，3-パターンを計算するために，<*pat-x*(1,1+0)>,<*pat-x*(1,1+2)>に追加する最右端文字<A>を見つける過程では，各々(<A>,0,2), (<A>,2,3)が *CandidateCharSet* に格納されている．(<A>,0,2)は，誤差 ε が 0 のとき，<A>が異なるタプルに 2 回現れることを示している． *CandidateCharSet* の全要素については，図 3 の(b)に示すとおりである．*CandidateCharSet* の各要素を求めるためには，図 3 の(a)に示されるようなテーブルを作成する必要がある．これに対応する集合が *RightMostCharTable* である．これには，3 項関係(α ,*i*, ε)が格納される．例えば，誤差 ε が 0 のとき，図 3 の(a)中に 2 件存在するパターン<*pat-x*-A>は，*i*=1 番目と 2 番目の配列に存在するので，*RightMostCharTable* では(<A>,1,0), (<A>,2,0)の 2 つの要素として表現される．誤差 ε が 2 のとき，図 3 の(a)中に 1 件存在するパターン<*pat-xxx*-A>は，5 番目に存

在するので，*RightMostCharTable* では(<A>,5,2)の要素として表現される．即ち，(<A>,5,2)は，誤差 ε が 2 のとき，最右端文字<A>が 5 番目のタプルでの存在を表す．

```
Main( S, min_sup, max_wc, ε max)
{ P1←set of 1-length frequent patterns;make PDB(<pat1>)for each
                                                    <pat1>in P1
  k←1;
  While( Pk ≠ φ )
   {  For each <patk>:cnt ∈Pk ,
      {  For each wc∈{ int| int∈Integer and 0≦int≦max_wc},
         {  RightMostCharTable← φ ; CandidateCharSet← φ ;
            For each (i, j)∈PDB(patk),
            { ε←0;
              While(ε≦ε max)
              { pos←j+wc+ε
                if pos≦length(Sl)then
                {  α ←Sl[pos];
                   call EnumerateSup( α , i, ε ,RightMostCharTable,
                                                    CandidateCharSet);
                }
                ε←ε+1;
              }
            }
            FrequentCharSet←{ t| t∈CandidateCharSet and t satisfies
                                                    with min_sup };
            For each ( β , ε , sup_cnt)∈FrequentCharSet,
            { patk+1←<patk-x(wc,wc+ε)- β >:sup_cnt;
              make PDB(<patk+1>); append patk+1 to Pk+1; }
         }
      }
      k←k+1;
   }
   output P;
}

Subroutine EnumerateSup( α ,i, ε ,RightMostCharTable,CandidateCharSet)
{  if not( α ∈P1) then return;
   if ( α , ε′, sup_cnt′)∈CandidateCharSet then
   {  CandidateCharSet←CandidateCharSet− { ( α , ε′, sup_cnt′) };
      if ( α , i, ε′)∈RightMostCharTable then count←sup_cnt′;
      else count←sup_cnt′+1;
      append ( α , ε , count) to CandidateCharSet;
   }
   else append ( α , ε , 1) to CandidateCharSet;
   append ( α , i, ε ) to RightMostCharTable;
   return;
}
```

図 4．新 *Modified PrefixSpan* 法のアルゴリズム

6. 計算結果と性能評価

ここでは，2 種類のデータセットに対して，従来方式と提案方式の計算結果を比較する．このために利用した計算機環境は，Intel PIV-2.4GHz，メモリー：2GB，SWAP メモリー：2GB，HDD：74.5GB，OS：Linux Red Hat v9.0 である．2 種類のデータセットは，各々，Zinc Finger と Leucine Zipper のモチーフを含む配列データベースである．

6.1. Zinc Finger データセット

PROSITE から Zinc Finger モチーフを含むデータセットを選択するために，アクセッション番号として PS00028 を用いた．このデータセットは 744 件の配列が含まれている．Zinc Finger モチーフは<C-*x*(2,4)-C-*x*(3)-[LIVMFYWC]-*x*(8)-H-*x*(3,5)-H>の形式で知られている．この形式は，72 種類の基本パターンを一つにまとめたものである．しかし，評価のために使われたデータセットには，その内の 34 種類の基本パターンだけが存在する．また，その 34 種類の基本パターンの内，配列データベース内で最も支持率の高いパターンの形式は<C-*x*(2)-C-*x*(3)-F-*x*(8)-H-*x*(3)-H>ある．このパターンは，配列データベース内に 3299 箇所あり，支持率は 82%である．また，最も支持率の低いパターンは<C-*x*(2)-C-*x*(3)-M-*x*(8)-H-*x*(4)-H>である．このパターンの支持率は，0.13%である．

この配列データベースから頻出パターンを抽出するために，入力データとして *max_wc* を 8 にして，従来方式では ε_{max} を 0，提案方式では ε_{max} を 2 として入力し，表 2 でその計算結果が表されいる．提案方式の計算では *min_sup* は 90%から 50%まで下げていって各支持率によって得られた基本パターンの種類数や頻出パターン数や計算時間を表している．また，提案方式では *min_sup* を 90%と 80%にして得られた基本パターンの種類数や頻出パターン数や計算時間を表している．

表 2．Zinc Finger の計算処理

比較項目＼最小支持率		90%	80%	70%	60%	50%
従来方式	正解の数(件)	0	1	1	1	1
	計算時間(sec)	2.23	3.46	6.7	15.25	49.89
	頻出パターン数(件)	81	277	1,108	3,998	19,104
提案方式	正解の数(件)	9	9	-	-	-
	計算時間(sec)	22.9625	128.616	-	-	-
	頻出パターン数(件)	1,042	9,669	-	-	-

従来方式では，最小支持率が 80%のとき，Zinc Finger モチーフの一部分として，82%の支持率をもつ頻出パターン<C-*x*(2)-C-*x*(3)-F-*x*(8)-H-*x*(3)-H>が頻出パターンの集合に含まれていた．それは，1 件の正解に相当する．最小支持率を 50%まで下げたが，他の正解は抽出されなかった．一般に最小支持率を下げると，計算に必要なメモリー領域が増える．我々の計算機環境では，最小支持率を 50%未満にすると，計算の途中でメモリー領域が不足したため，途中で計算を打ち切った．これに対して，提案方式では，最小支持率が 90%のとき，<C-*x*(2,4)-C-*x*(3,5)-F-*x*(8,10)-H-*x*(3,5)-H>:93%ほか 17 件の頻出パターンが正解を含んでいた．それらを基本パターンとして整理すると，9 件の正解（正解総数の 26%）に相当することがわかった．最小支持率を 80%未満に設定すると，提案方式では計算の途中でメモリー領域が不足する．提案方式では最小支持率が 90%のときの計算時間は 23.0 秒であったが，従来方式でこれと同じ程度の計算時間に対応する最小支持率は 60%～50%である．このとき，従来方式では，正解が 1 件だけしか抽出されていないことがわかる．以上から，提案方式は従来の方式に比べて，正解の数が 9 倍多かったと言える．

6.2. Leucine Zipper データセット

ここでは，PROSITE から Leucine Zipper モチーフを含むデータセットを選択するために，アクセッション番号として PS00036 を用いた．Leucine Zipper モチーフの形式は <[KR]-*x*(1,3)-[RKSAQ]-N-*x*(2)-[SA

Q](2)-*x*-[RKTAENQ]-*x*-R-*x*-[RK]>である．この形式は，3780 種類のパターンタイプを一つにまとめたものである．しかし，評価のために使われたデータセットには，その内の 64 種類の基本パターンだけが存在する．配列データベース内で最も多くあるパターンの形式は <R-*x*(1)-RN-*x*(2)-AA-*x*(1)-K-*x*(1)-R-*x*(1)-R>である．このパターンは，配列データベース内に 39 箇所あり，支持率は 32%である．最も支持率が低いパターンは 25 件ありその内の 1 つが<R-*x*(1)-SN-*x*(2)-SA-*x*(1)-R-*x*(1)-R-*x*(1)-R>である．支持率は，0.82%である．

この配列データベースでの抽出のためには入力データとして *max_wc* を 2 にして，従来方式では ε_{max} を 0 で，提案方式では ε_{max} を 2 として入力した．表 3 では，従来方式と提案方式の計算結果が表されている．

表 3．Leucine Zipper の計算処理

比較項目 ＼ 最小支持率		37%	36%	30%	25%	20%	18%	17%
従来方式	正解の数(件)	0	0	1	1	1	1	1
	計算時間(sec)	0.155	0.156	0.172	0.188	0.261	8.344	48.94
	頻出パターン数(件)	970	1009	1644	2392	7207	698812	4138776
提案方式	正解の数(件)	8	8	-	-	-	-	-
	計算時間(sec)	11.7	12.8	-	-	-	-	-
	頻出パターン数(件)	56821	64182	-	-	-	-	-

従来方式では，最小支持率が 30%のとき，Leucine Zipper モチーフの一部分として，32%の支持率をもつ頻出パターン<R-*x*(1)-RN-*x*(2)-AA-*x*(1)-K-*x*(1)-R-*x*(1)-R>が頻出パターンの集合に含まれていた．それは，1 件の正解（正解総数の 1.5%）に相当する．最小支持率を 17%まで下げたが，他の正解は抽出されなかった．最小支持率を 17%未満にすると，計算の途中でメモリー領域が不足のためすべての頻出パターンを抽出できなかった．これに対して，提案方式では，最小支持率が 37% のとき，<K-*x*(2,4)-R-*x*(0,2)-N-*x*(1,3)-A-*x*(0,2)-A-*x*(1,3)-R-*x*(1,3)-R-*x*(1,3)-R>:37%，<R-*x*(1,3)-R-*x*(0,2)-N-*x*(2,3)-A-*x*(0,2)-A-*x*(1,3)-R-*x*(1,3)-R-*x*(1,3)-R>:37%，<R-*x*(1,3)-R-*x*(0,2)-N-*x*(2,3)-A-*x*(0,2)-A-*x*(1,3)-K-*x*(0,1)-R-*x*(0,2)-R>:36%，ほか 7 件の正解を含んでいた．それらをまとめると，8 件の正解（正解総数の 12.5%）に相当することがわかった．提案方式では，最小支持率を 36%未満に設定すると，提案方式では計算の途中でメモリー領域が不足する．提案方式では最小支持率が 37%のときの計算時間は 11.7 秒であったが，従来方式でこれを同じ程度の計算時間に対応する最小支持率は 18%～17%である．このとき，従来方式では，正解が 1 件だけしか抽出されていないことがわかる．以上から，提案方式は従来方式に比べて，正解の数が 8 倍多かったと言える．

以上の評価で得られた結果によって，本論文の提案方式は従来方式に比べて，優れた抽出能力を持っている事を確認した．従来方式では，PROSITE に乗っているモチーフの形式をした頻出パターンを抽出するためには，最小支持率 *min_sup* を低くしないと得られない．だが，我々が提案している方式では，最小支持率 *min_sup* が高くても頻出パターンが得られる．また，従来方式で見落としていた様々なパターンも抽出する事が出来た．

7. まとめ

従来の *Modified PrefixSpan* 法は，最小支持数と最大ワイルドカード数の 2 つの設定パラメータに対して，固定長ワイルドカード領域を含む頻出パターンだけを抽出していたので，可変長ワイルドカード領域を含む頻出パターンの一部分になっている基本パターンを見落としてしまうという問題がある．本論文では，この問題を解決するために，*Modified PrefixSpan* 法に可変長ワイルドカード領域を含む頻出パターンを抽出する機能を持たせる方法を提案した．具体的には，従来方法に最大ワイルドカード数に対する最大誤差数のパラメータを新たに導入した．提案方法の有効性を確かめるために，PROSITE から 2 種類のデータセットを選び，各々についての計算結果を従来方法と比較した．そこでは，提案方法と従来方法の計算終了時間がお互いにほぼ同じなるような最小支持率を各々選択し，各々の方法で抽出された頻出パターン集合を調べた．各々の頻出パターン集合の中を調べた結果，提案方法が従来方法に比べて，正解としての基本パターンが 8～9 倍多く抽出されていることがわかった．提案方法に限らず従来方法では，一般に，最小支持数を下げれば，計算時間や計算に必要なメモリー容量が増える．今回，提案方法で抽出された基本パターンの中には，設定された最小支持数よりも小さな支持数が含まれている．極端な例になるが，Zinc Finger データセットの場合，抽出された可変長ワイルドカード領域を含む頻出パターンの中に，支持数が 1 の基本パターン（<C-*x*(3)-C-*x*(3)-F-*x*(8)-H-*x*(5)-H>）が含まれていた．これを従来方法で最小支持数を 1 と設定して計算をすると，計算の途中で組合わせ爆発が生じる．即ち，*n* 本の配列の平均配列長を *m* とすると，時間計算量は $O(n2^m)$ となってしまうので，従来方法を用いてこの基本パターンを見つけるのは現実的ではない．

今後の課題として，3 章で説明した「曖昧要素」を含む頻出パターンを抽出する方法の研究があげられる．例えば，本論文で扱った Zinc Finger モチーフは，<C-*x*(2,4)-C-*x*(3)-**[LIVMFYWC]**-*x*(8)-H-*x*(3,5)-H> の形式であったが，その中に 1 つの曖昧要素**[LIVMFYWC]**が含まれていた．このモチーフは，<C-*x*(2,4)-C-*x*(3)-**L**-*x*(8)-H-*x*(3,5)-H，<C-*x*(2,4)-C-*x*(3)-**I**-*x*(8)-H-*x*(3,5)-H>，<C-*x*(2,4)-C-*x*(3)-**V**-*x*(8)-H-*x*(3,5)-H>など合計 8 種類の

パターンをまとめたものと解釈できる．

謝辞

本研究の一部は，広島市立大学特定研究費（一般研究費（コード番号：3106）の支援により行われた．

文　献

[1] Needleman, S.B., and Wunsch, C.D.: A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of two Problems, Journal of Molecular Biology, Vo.48, 443-453, 1970.
[2] M. S. Waterman, T. F. Smith, and W. A. Beyer: Some Biological Sequence Metrics, Adv. Math., Vol. 20, pp.367-387, 1976.
[3] Hirosawa M. et al.: Comprehensive Case Study on Iterative Algorithms of Multiple Sequence Alignment, Comput. Applic. Biosci., Vol.11, pp.13-18, 1995.
[4] Timothy L. Bailey and Charles Elkan: Fitting a Mixture Model by Expectation Maximization to Discover Motifs in Biopolymers, Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology, pp.28-36, AAAI Press, Menlo Park, California, 1994.
[5] A. Bairoch, P. Bucher, and K. Hofman: The PROSITE Database: Its Status in 1995, Nucleic Acids Research, Vol.24, pp.189-196, 1996.
[6] PROSITE：http://kr.expasy.org/prosite
[7] E.L.L. Sonnhamer, S. R. Eddy, and R. Durbin: Pfam: A Comprehensive Database of Proteins, Vol.28, Vo.405-420, 1997
[8] Brazma, A., Jonassen, I., Eidhammer, I. AndGilbert, D.: Approaches to the Automatic Discovery of Patterns in Biosequences, Technical Report, Depertment of Informatics, University of Bergen, Norway, 1995.
[9] Inge Jonassen, John F. Collins, and Desmond G. Higgins: Finding Flexible Patterns in Unaligned Protein Sequences, Protein Science, pp.1587-1595, Cambridge University Press, 1995.
[10] Isidore Rigoutsos and Aris Floratos: Combinatorial Pattern Discovery in Biological Sequences: the TEIRESIAS Algorithm, BIOINFORMATICS, Vo.14 No.1, p.55-67, 1998.
[11] Isidore Rigoutsos and Aris Floratos: Motif Discovery without Algnment or Enumeration, Proceedings of Second Annual ACM International Conference on Computational Molecular Biology (RECOMB 98), pp.221-227, March 1998.
[12] Aris Floratos and Isidore Rigoutsos: On the Time Complexity of the TERIESIAS Algorithm, IBM Research Report, RC 21161(94582), April 1998.
[13] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, and Helen Pinto: PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth, Proc. of International Conference on Data Engineering (ICDE 2001), IEEE Computer Society Press, p.215-224, 2001.
[14] Hajime Kitakami, Tomoki Kanbara, Yasuma Mori, Susumu Kuroki, and Yukiko Yamazaki: Modified PrefixSpan Method for Motif Discovery in Sequence Databases, Proceedings of the 7th Pacific Rim International Conference on Artificial Intelligence (PRICAI2002), pp.482-491, Springer-Verlag, August 2002.
[15] D. Gusfield: Algorithms on Strings, Trees, and Sequences, Cambridge University Press, 1997.
[16] Heikki Mannila, Hannu Tivonen, and A. Inkeri VerKamo: Discovery of Frequent Episodes in Event Sequences, Data Mining and Knowledge Discovery, Vo.1, pp.259-289, 1997.
[17] Eleazar Eskin and Pavel A. Prevzner: Finding Composite Regulatory patterns in DNA Sequences, BIOINFORMATICS, Vol.1 No.1, pp.1-9, 2002.
[18] Marie-France Sagot and Alain Viari: A Double Combinatorial Approach to Discovering Patterns in Biological Sequences, Proceedings of the 7th Symposium on Combinatorial Pattern Matching, Springer-Verlag, pp.186-208, 1996.
[19] 五條掘孝 編：情報学，シュプリンガー・フェアラーク・東京，2003年．