論　文

分節木と共有文字列で表現される符号上での効率良い圧縮照合アルゴリズム

喜田　拓也[†a)]

An Efficient Compressed Pattern Matching Algorithm on Codes Represented by Parse Tree and Shared String

Takuya KIDA[†a)]

あらまし　本論文では，VF 符号で圧縮されたテキスト上でのパターン照合アルゴリズムについて論じる．ここで扱う VF 符号とは，テキストを分節木によって可変長の部分文字列に分割し，固定長の符号語を割り当てる形式の符号化手法である．圧縮照合問題の形式的枠組みである Collage system を用いれば，VF 符号上で KMP 型の汎用的なアルゴリズムを組織的に導出でき，$O(m^2+D)$ 時間・領域の前処理の後，$O(n+R)$ 時間で圧縮テキストを走査できる．ここで，m，n，R，D はそれぞれ，パターン長，テキスト長，パターンの出現回数，圧縮辞書のサイズである．しかし，この圧縮辞書のサイズは，各符号語に対する文字列の長さの総和に等しい．そこで，分節木上で共有される文字列の構造を利用し，より効率良く前処理を行うアルゴリズムを提案する．提案アルゴリズムは，大きさ $|T|$ の分節木 T 上の各ラベル文字列が，長さ $|S|$ である文字列 S の一部分として表現されている場合，$O(m^2+|S|+|T|)$ 時間・領域で前処理を行うことができる．ここで，$|S|+|T|$ はもとの D よりも非常に小さいものである．本結果は，Collage system 上で示されており，同種の構造の圧縮法であれば VF 符号に限らず適用できる．

キーワード　VF 符号，圧縮パターンマッチング，Collage system

1. まえがき

圧縮テキスト上でのパターン照合とは，データ圧縮されたテキストに対して，それを明示的に復元することなくパターン照合を行うことである．これは圧縮照合問題（Compressed matching problem）と呼ばれており，90 年代初頭で提案され，以降盛んに研究が行われている．当初この問題は，主に理論的な興味から始まったが，大規模テキストデータベースが個人で比較的簡単に取り扱えるようになった今日では，実用上の重要な課題となってきている．

これまでの研究から，圧縮照合に適したデータ圧縮法を選択すれば，高速に照合が行えることが知られている [1]．そうした圧縮法は，共通して次のような性質を備えている．

- 固定長符号である．特に符号長がバイトの整数倍であることが望ましい．
- 静的でコンパクトな辞書を用いる圧縮法である．

このような視点から既存の圧縮法を再評価すると，よく知られている Huffman 符号や LZ 系圧縮のような可変長符号化では，圧縮データを走査する際にビット切出しや符号語の境界判定などの処理が必要なため，照合に余計な時間がかかってしまう．これに対し VF 符号（Variable-length-to-Fixed-length code）[2] は，各符号語の境界が明らかで，更に 8 ビットの整数倍の符号語長の場合にはバイト単位でデータを処理することができるため，非常に都合がよい．

しかし，上述の性質は圧縮率という観点からは大きな制約となる．事実，過去に提案された VF 符号は他の圧縮法と比べて圧縮率が低く，あまり注目されてこなかった．このような背景の中，近年，圧縮照合の高速化を見据えつつ圧縮率を改善した圧縮法がいくつか提案されている．Klein と Shapira [3]，Kida [4] による接尾辞木 [5] を用いた圧縮法や，Maruyama ら [6]

† 北海道大学大学院情報科学研究科，札幌市
Graduate School of Information Science and Technology, Hokkaido University, N14 W9, Sapporo-shi, 060–0814 Japan
a) E-mail: kida@ist.hokudai.ac.jp

による改良型 BPE 圧縮法は，VF 符号であるにもかかわらず gzip に迫る圧縮率を達成する．

一方，これまでに筆者らは，Collage system と呼ぶ，圧縮されたテキストを表現する形式的枠組みを提案し，各種圧縮法に適用できる統一的な圧縮照合アルゴリズムを得ている [7]．その結果を VF 符号に適用すれば，VF 符号上で KMP 型の圧縮照合を行うアルゴリズムが得られ，$O(m^2 + D)$ 時間・領域の前処理の後，$O(n + R)$ 時間で圧縮テキストを照合することができる．ここで，m，n，R，D はそれぞれ，パターン長，テキスト長，パターンの出現回数，圧縮辞書のサイズである．ただし，ここでいう圧縮辞書のサイズとは，各符号語に対応する文字列すべてを連結した長さに等しく，辞書が大きい場合は前処理部分に問題が生じる．実際，約 4 M バイト程度の英文テキストを [4] で符号化し上述のアルゴリズムで照合すると，テキスト走査部分に比べて前処理に最大で 30 倍以上の時間がかかる場合があることが判明した [8]．したがって，前処理をいかに高速化するかが重要な課題となる．そのために，符号語辞書のサイズに関係する計算量を削減する必要がある．

VF 符号では，符号語と文字列の対応を示す辞書が，分節木と呼ばれる木構造によって表現されている(注1)．分節木の各辺には文字列がラベリングされており，各符号語は分節木の根から葉へ至るパス上の文字列を連結したものになっている．例えば，代表的な VF 符号である Tunstall 符号 [9] では，図 1 のような k 文木を分節木として用いてテキストを分割する．ここで，k は情報源アルファベットのサイズである．このように，辞書に登録されている語の接頭辞は多く共有されている．VF 符号に対する Collage system において，この共有情報を利用して圧縮辞書のサイズを削減することは自然な考えである．

本論文では，より一般的に，分節木上の各ラベルがある文字列 S へのポインタとして表現され，ラベル同士で文字列の一部あるいは全部を共有している場合について考える（図 2）．分節木や共有文字列 S の具体的な構成については議論しないが，例えば，接尾辞木を短く刈り込んだ木構造を分節木として用いる [3] や [4] の VF 符号では，このようなデータ構造が実際に用いられる．この場合，共有先の文字列はラベル長の総和よりも短い(注2)．よって，この文字列長に比例した時間で前処理を行うことができれば，更なる時間短縮につながる．しかし，Collage system の枠組みを単純に

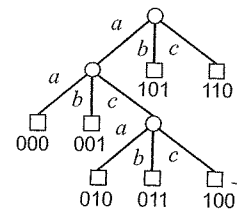


図 1　Tunstall 符号化における分節木の例
Fig. 1　An example of Tunstall tree.

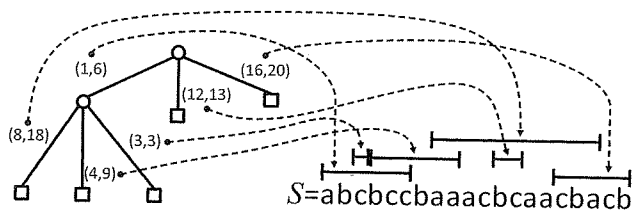


図 2　ラベルが文字列へのポインタで表現されている例
Fig. 2　An example of a parse tree whose labels are represented as pointers.

適用すると，前処理に $O(m^2 + |S| + |\mathcal{T}|^2)$ 時間・領域がかかってしまう．ここで，$|S|$ は共有文字列 S の長さ，$|\mathcal{T}|$ は分節木のノード数である．よって，大きな分節木に対して $O(|\mathcal{T}|^2)$ の計算量は好ましくない．

そこで本論文では，図 2 のような分節木を用いる VF 符号において，前処理を $O(m^2 + |S| + |\mathcal{T}|)$ 時間・領域で行うアルゴリズムを新たに提案する．本結果は，Collage System の枠組みを用いて示しているため，符号語の切出し処理を別にすれば，本質的には符号語長の制約とは無関係である．言い換えると，VF 符号でなくとも，図 2 のような分節木によって表現される符号全体に対して本論文の結果は適用できる．

2. 準　　備

2.1 記法と用語の定義

Σ を有限アルファベットとする．Σ^* は Σ 上の文字列すべてからなる集合である．文字列 $x \in \Sigma^*$ の長さを $|x|$ と書く．長さが 0 の文字列を空語と呼び，ε で表す．したがって，$|\varepsilon| = 0$ である．二つの文字列 x_1 と x_2 を連結した文字列を $x_1 \cdot x_2$ で表す．特に混乱がない場合は，これを $x_1 x_2$ と略記する．

文字列 x，y，z は，$w = xyz$ であるとき，それぞれ

(注1)：VF 符号は単に符号長の制約について言い表したものであるが，分節木による辞書表現は Tunstall 符号に代表される主な VF 符号に共通の仕組みであるため，本論文では，分節木を用いて可変長の部分文字列を固定長の符号語へマッピングする符号化方式を特に VF 符号と呼んでいる．

(注2)：接尾辞木全体では，S は入力テキスト全体に等しいが，文献 [3], [4] の手法では刈り込んだ接尾辞木を用いているので，本質的に S はテキスト全体より短いものにできる．

w の接頭辞，部分文字列，接尾辞と呼ばれる．文字列 w の i 番目の記号を $w[i]$ で表す．また，w の i 番目から始まり j 番目で終わる部分文字列を $w[i:j]$ と書く．ただし，簡便のため，$j<i$ のとき $w[i:j]=\varepsilon$ とする．また，文字列 $w\in\Sigma^*$ のすべての部分文字列からなる集合を $Fac(w)$ と書く．更に，文字列 $u,v\in\Sigma^*$ について，

$lpf_v(u)\equiv u$ の接頭辞でかつ v の部分文字列となる最長の文字列，

$lsf_v(u)\equiv u$ の接尾辞でかつ v の部分文字列となる最長の文字列，

$lps_v(u)\equiv u$ の接頭辞でかつ v の接尾辞となる最長の文字列，

$lsp_v(u)\equiv u$ の接尾辞でかつ v の接頭辞となる最長の文字列，

と定義する．これらから，次の事実が観察できる．

[事実 1] 任意の文字列 $u,v\in\Sigma^*$ について，以下の式が成り立つ．

$$lps_v(u)=lps_v(lpf_v(u)),\quad lsp_v(u)=lsp_v(lsf_v(u)).$$

また，以下の重要な補題が [7] で示されている．

[補題 1]（[7] の補題 6） パターン $P=P[1:m]$ が与えられたとする．このとき，任意の $x,y\in Fac(P)$ に対し，$lpf_P(xy)$ を $O(1)$ 時間で返答する手続きは，$O(m^2)$ 時間・領域を使って構築できる．$lsf_P(xy)$ についても同様である．

[補題 2]（[7] の補題 7） パターン $P=P[1:m]$ が与えられたとする．このとき，任意の $x\in Fac(P)$ に対し，$lps_P(x)$ を $O(1)$ 時間で返答する手続きは，$O(m^2)$ 時間・領域を使って構築できる．$lsp_P(x)$ についても同様である．

整数の集合 A と一個の整数 k に対して，$A\ominus k=\{i-k\mid i\in A\}$ と定義する．文字列 $x,y\in\Sigma^*$ に対して，y 中に現れる x の出現位置の集合を $Occ_x(y)$ と書く．すなわち，$Occ_x(y)=\{i\mid |x|\le i\le |y|, x=y[i-|x|+1:i]\}$ である．また，文字列 $x,u,v\in\Sigma^*$ に対して，uv 中に出現し u と v の境界をまたぐ x の出現位置の集合を $Occ^*_x(u\bullet v)$ と書く．すなわち，$Occ^*_x(u\bullet v)=\{i\mid i\in Occ(x,uv), |u|<i<|u|+|x|\}$ である．

2.2 VF 符号

古典的で重要な VF 符号の一つに Tunstall 符号 [9] がある．Tunstall 符号は，各辺に Σ（$|\Sigma|=k$）の要素がラベル付けされた順序付き k 分木を分節木とし，この木を用いてテキストを二元符号化する．その符号化方法は，以下のとおりである．まず，記憶のない情報源に対し最適な分節木 $\mathcal{T}$（Tunstall 木と呼ばれる）を構築し，そのすべての葉に $\lceil\log N\rceil$ ビットの整数で番号付けを行う．ここで，N は $\mathcal{T}$ の葉の個数である．すなわち，$\mathcal{T}$ の根から葉へ至るパス上の文字を並べた文字列に符号語 1 個が割り当てられる．Tunstall 木の構成手順の詳細は，圧縮照合アルゴリズムとは直接に関係しないので省略する．

テキストは Tunstall 木 $\mathcal{T}$ に登録された文字列で分割され，分割された各々の部分文字列に対応する符号語を割り当てることで符号化する．その手順は以下のとおり．

（1） $\mathcal{T}$ の根を探索のスタート地点とする．

（2） 入力テキストから記号を 1 個読み取り，分節木 $\mathcal{T}$ 上の現節点からその記号でラベル付けされた子へと移る．もし，葉に到達したら，その葉の番号を符号語として出力し，探索の地点を根へ戻す．

（3） ステップ 2 をテキストの終端まで繰り返す．

例えば，図 1 の分節木で，テキスト $T=aaabbacb$ を符号化すると，符号語の系列は 000 001 101 011 となる．Tunstall 木は，テキスト復号時にも必要であるが，各情報源記号の出現確率と符号語長の情報があれば復元できるので，明示的に木を保存しなくともよい．

他の VF 符号においても，分節木が構築された後，それを用いてテキストを分割して符号化する手順はほとんど同じである．分節木上に登録された文字列が等長の符号語に対応付けされ，VF 符号による圧縮テキストがその符号語の列であるという構造に違いはない．

2.3 接尾辞トライ

パターン $P=P[1:m]$ に対する接尾辞トライ（Suffix trie）とは，P の接尾辞のすべてからなる集合を表現するトライのことである．接尾辞トライ（以降 ST_P）のノードは，P の接尾辞を表すか，あるいはその出次数（子供の数）が 2 以上のとき，またそのときに限り明示的（explicit）と呼ばれる．明示的でないノードは暗黙的（implicit）と呼ばれる．ST_P のノード数は $O(m^2)$ 個であり，ST_P は $O(m^2)$ 時間・領域で構築できる．一方，明示的なノードの個数は $O(m)$ 個であることに注意する（[5] を参照）．

ST_P の各ノードは $Fac(P)$ に対応している．以降，特に混乱が生じない限り，文字列 $x\in Fac(P)$ と x を表すノードを同一視する．例えば，「$lpf_P(x)$ を計算する」ということは，「ST_P 上で文字列 $lpf_P(x)$ を表

すノードを求める」ことを意味する．また，各ノード $x \in Fac(P)$ について，その先頭1文字を取り除いた接尾辞 $y = x[2 : |x|]$ を表すノードへのポインタは接尾辞リンクと呼ばれる．接尾辞リンクは，接尾辞トライ構築時に（計算量の増加を伴わずに）すべてのノードに付加することができる．

2.4 Knuth-Morris-Pratt 法

Knuth-Morris-Pratt 法（KMP 法）[5] は，よく知られた線形時間のパターン照合アルゴリズムである．このアルゴリズムの動作は，与えられたパターン P に対する直線的なオートマトン（KMP オートマトン）として模式化できる．P に対する KMP オートマトンは次の二つの関数から構成される．

goto 関数 $g : Q \times \Sigma \to Q \cup \{fail\}$,

failure 関数 $f : Q \setminus \{0\} \to Q$

ここで $Q = \{0, 1, \cdots, |P|\}$ は状態の集合，$fail$ は Q に含まれない特別な記号とする．goto 関数 g は整数 $j \in Q$ と文字 $a \in \Sigma$ を入力とし，$P[j+1] = a$ ならば $j+1$ を返す．そうでないならば $fail$ を返す．ただし，$j = 0$ の場合は，任意の $a \in \Sigma$ のうち $P[1] \neq a$ でないものに対して $g(0, a) = 0$ とする．failure 関数 f は，整数 $j \in Q \setminus \{0\}$ を入力として，$P[1 : k] = P[j-k+1 : j]$ を満たす最大の整数 k を返す．図 3 は，パターン $P = abacb$ に対する KMP オートマトンを示している．このオートマトンは，テキストから1文字ずつ文字を読みながら，現在の状態からその文字に対応する g をたどって状態遷移を繰り返す．g が $fail$ を返した場合には，現在の状態番号を引数に f を呼び出し状態遷移を行い，再度同じ文字で g による遷移ができるかどうかを繰り返す．最終的に一番右端の状態へ到達したら，P が出現したと判断できる．テキスト $abacbbaababacbb$ 上での図 3 の KMP オートマトンの動きは，図 4 のようになる．この例では，番号5の状態へ到達したとき，テキスト中に P が出現したと分かる．

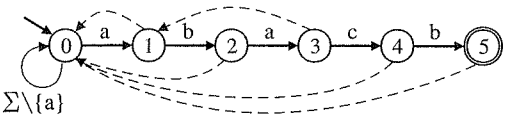
failure 関数を省略するために，状態遷移関数 $\delta : Q \times \Sigma \to Q$ を次のように定義する．

$$\delta(j, a) = \begin{cases} g(j, a) & g(j, a) \neq fail \text{ のとき}, \\ \delta(f(j), a) & \text{それ以外} \end{cases}$$

更に，以下のようにして $Q \times \Sigma^*$ へと拡張する．

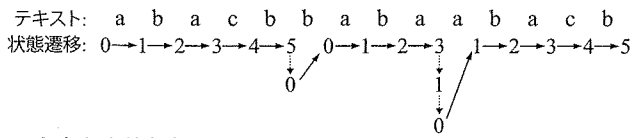
$$\delta^*(j, \varepsilon) = j, \quad \delta^*(j, ua) = \delta(\delta^*(j, u), a).$$

ここで，$j \in Q$，$u \in \Sigma^*$，$a \in \Sigma$ である．関数 δ を特



図中の○は状態を，太い○は最終状態を表す．矢印と点線矢印は，それぞれ goto 関数と failure 関数を表している．

図 3 $P = abacb$ に対する KMP オートマトン
Fig. 3 KMP automaton for $P = abacb$.



矢印と点線矢印は，それぞれ goto 関数，failure 関数による状態遷移を表している．

図 4 KMP オートマトンの動き
Fig. 4 Move of the KMP automaton.

徴づける次の補題は特に重要である．

[補題 3] 任意の $j \in Q$ と文字列 $u \in \Sigma^*$ について，$\delta(j, u) = |lsp_P(P[1 : j] \cdot u)|$ である．

3. Collage System のインスタンスとしての VF 符号上の圧縮照合アルゴリズム

VF 符号化された圧縮テキストは，その符号語を定義する辞書が分節木であり，辞書中の文字列は分節木のラベルを連結したものとして得られる．これは，後述するように，正規的な（regular）Collage system の枠組みに含まれる．よって，VF 符号上での KMP 型照合アルゴリズムは，正規的な Collage System に対する圧縮照合アルゴリズムのインスタンスとして求めることができる．

3.1 Collage system

Collage system [7] とは，以下で定義される組 $\langle \mathcal{D}, \mathcal{S} \rangle$ である．すなわち $\mathcal{D}$ は変数定義の列 $X_1 = expr_1; X_2 = expr_2; \cdots; X_n = expr_n$ で，$expr_k$ は次の形式のいずれかである．

$a \quad (a \in \Sigma \cup \{\varepsilon\})$, 文字代入
$X_i \cdot X_j \quad (i, j < k)$, 連結
${}^{[j]}X_i \quad (i < k, j \text{ は整数})$, 前 i 文字切り落とし
$X_i^{[j]} \quad (i < k, j \text{ は整数})$, 後 i 文字切り落とし
$(X_i)^j \quad (i < k, j \text{ は整数})$, 繰返し

$\mathcal{D}$ 内で定義された変数すべてからなる集合を $F(\mathcal{D})$ と書く．このとき，各変数はそれを評価したときに得られる文字列に対応する．変数 $X \in F(\mathcal{D})$ が表す文字列を $X.u$ と書くことにする．例えば，$X_1 = a; X_2 =$

$b; X_3 = X_1 \cdot X_2; X_4 = (X_3)^3; X_5 = X_4^{[1]}$ のとき，$X_4.u = ababab$，$X_5.u = ababa$ である．ただし以降では，変数 X_i とそれが表す文字列とを同一視し，文字列を表す場合でも特に混乱の生じない限り単純に X と書くことにする．$\mathcal{D}$ における変数代入の個数を $\mathcal{D}$ の大きさとし，$||\mathcal{D}||$ と表す．

一方，$\mathcal{S}$ は $\mathcal{D}$ において定義された変数の列 $\mathcal{S} = X_{i_1}, X_{i_2}, \ldots, X_{i_k}$ である．$\mathcal{S}$ の長さを $\mathcal{S}$ 中の変数の個数 k とし，$|\mathcal{S}|$ と表記する．また，$\mathcal{S}$ 中に現れる変数すべてからなる集合を $F(\mathcal{S})$ と書く．明らかに $F(\mathcal{S}) \subseteq F(\mathcal{D})$ である．Collage system$\langle \mathcal{D}, \mathcal{S} \rangle$ は，$X_{i_1}, \ldots, X_{i_k}$ を連結して得られる文字列を表現する．すなわち $\mathcal{D}, \mathcal{S}$ はそれぞれ，圧縮法における辞書の表現と圧縮テキストに対応している．実際の圧縮法においては，$\mathcal{D}$ や $\mathcal{S}$ は様々な方法で符号化され，圧縮率は $||\mathcal{D}||$ や $|\mathcal{S}|$ よりも，むしろ $\mathcal{D}$，$\mathcal{S}$ の符号化の方法に左右される．

繰返しや切り落としを含まない Collage system を，正規的（regular）と呼ぶ．次に述べるように，VF 符号は正規的な Collage system のクラスに属する．

3.2 VF 符号の Collage system 表現

VF 符号の符号語を Collage system 表現する最も単純な方法は，それぞれが表す文字列を文字の連結に分解して順番に変数割当を行うことである．しかしそれでは，辞書のサイズが各符号語の示す文字列の長さの総和に比例してしまう．VF 符号の符号語が分節木によって文字列と対応付けされることに着目すると，次のように多くの変数を共通にすることができる．

まずは，分節木のある 1 個の辺に注目し，そのラベル文字列を ℓ とする．これを表すのに必要な変数の個数は，文字代入された変数が他のラベルと共有できたとしても，最悪の場合には $|\ell| - 1$ 個必要である．

分節木の各ノードが表す文字列を表現するために，親の文字列を表す変数と自ノードへ至る辺ラベルを表す変数とを連結させて新たな変数を定義する．ただし，分節木の根に対応する変数は ε を表していることとする．このように各ノードが表す文字列に対応する変数定義をすると，各符号語の変数間で多くの接頭辞を共通の変数で表すことができる(注3)．

したがって，ラベル文字列をすべて連結した長さを L，分節木のノード数を N とすると，辞書 $\mathcal{D}$ に登録される変数の個数は $O(L+N)$ 個である．しかし，分節木の各ラベルは長さ 1 以上の文字列であるので，明らかに $N \leq L$ であり，よって辞書のサイズは $O(L)$ である．符号語 c_i に対応する変数を $X_i \in \mathcal{D}$ とすると，VF 符号の系列 $Z = c_{i_1}, \ldots, c_{i_n}$ に対する $\mathcal{S}$ は，$\mathcal{S} = X_{i_1}, \ldots, X_{i_n}$ と書ける．例えば，図 1 の例では，辞書 $\mathcal{D}$ は，

$$X_1 = a; \quad X_4 = X_1 \cdot X_1; \quad X_7 = X_6 \cdot X_1;$$
$$X_2 = b; \quad X_5 = X_1 \cdot X_2; \quad X_8 = X_6 \cdot X_2;$$
$$X_3 = c; \quad X_6 = X_1 \cdot X_3; \quad X_9 = X_6 \cdot X_3;$$

と表現できる．また，符号語の系列 000 001 101 011 に対する $\mathcal{S}$ は，$\mathcal{S} = X_4, X_5, X_2, X_8$ である．

3.3 Collage system に対する照合アルゴリズム

Collage system 上での照合アルゴリズムの基本的な考えは，非圧縮テキスト上での KMP オートマトンの動作を模倣することである．**2.4** で定義した KMP オートマトンの状態遷移関数 δ^* を用いて，関数 $Jump : Q \times F(\mathcal{D}) \to Q$ を次のように定義する．

$$Jump(j, X) = \delta^*(j, X.u).$$

この関数 $Jump$ の意図は，変数 X を受け取り飛び跳ねるようにしてもとの KMP オートマトンの状態遷移を模倣することである．また，任意の $j \in Q$ と $X \in F(\mathcal{D})$ について，集合 $Output(j, X) = Occ_P(P[1:j] \cdot X.u)$ を定義する．

テキスト T を表現する $\langle \mathcal{D}, \mathcal{S} \rangle$ とパターン P が与えられたとき，照合アルゴリズムは，$\mathcal{D}$ から $Jump$ と $Output$ を計算するために必要な情報を前処理で求め，その後に $\mathcal{S}$ の変数列を前から順番に走査しながら照合を行う（図 5）．

Input. Collage system$\langle \mathcal{D}, \mathcal{S} \rangle$ およびパターン $P = P[1:m]$.
Output. T 中に現れる P のすべての出現位置.

```
/* 前処理 */
Jump と Output に必要な情報を計算;
/* 圧縮テキスト走査 */
let S = X_{i_1} X_{i_2} ... X_{i_n}.
ℓ := 0; state := 0;
for k := 1 to n do begin
  for each p ∈ Output(state, X_{i_k}) do
    位置 ℓ + p − m + 1 にパターンが出現;
  state = Jump(state, X_{i_k});
  ℓ := ℓ + |X_{i_k}|
end
```

図 5　Collage system 上の照合アルゴリズム
Fig. 5　A matching algorithm on Collage system.

(注3)：もちろん，各ラベルを定義する際に，その部分文字列を表す変数を共有することも可能であるが，そのためには同じ部分文字列を表現する変数を探索しなければならない．

[7] の結果から，以下の定理が得られている．

[定理 1]（[7] の定理 3 より） Collage system $\langle \mathcal{D}, \mathcal{S} \rangle$ に対するパターン $P = P[1:m]$ の照合問題は，$\langle \mathcal{D}, \mathcal{S} \rangle$ が正規的であるとき，$O(\|\mathcal{D}\| + m^2)$ 領域を使って，$O(\|\mathcal{D}\| + |\mathcal{S}| + m^2 + R)$ 時間で解決できる．ここで，R はパターンの出現回数である．

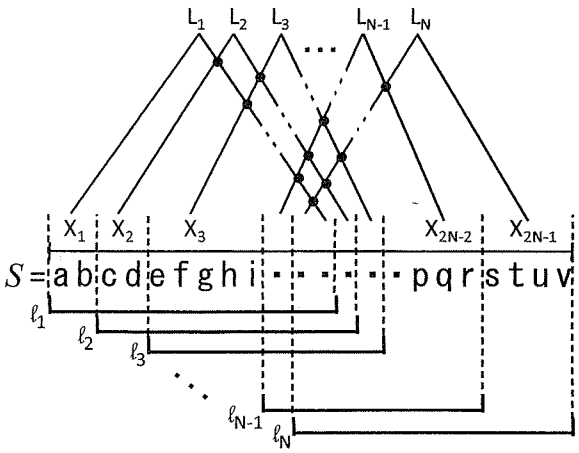
前述のとおり，VF 符号は正規的な Collage system で記述できるので，この定理が当てはまる．

4. ラベル文字列が共有されている場合の前処理アルゴリズム

本章では，図 2 のように，各辺のラベル文字列が共有されている場合を考える．すなわち，ラベル文字列がある文字列 S へのポインタで表現されており，$|S|$ はラベル長の総和 L よりもずっと短いと仮定する．

このとき，この VF 符号を Collage System で表現して前章の議論どおりに照合アルゴリズムを得ると，後述するとおり，大きさ $|\mathcal{T}|$ の分節木に対して前処理に $O(m^2 + |S| + |\mathcal{T}|^2)$ 時間・領域がかかってしまう．この計算量を $O(m^2 + |S| + |\mathcal{T}|)$ に削減するためには，パターン照合に必要な前処理を異なる手段で行わなければならない．

以下に，Collage System を単純に適用した場合の計算量について議論する．根を除いたノードの数が $N = |\mathcal{T}| - 1$ である分節木を考える．すなわち，この分節木の辺の数は N である．i 番目の辺のラベルを ℓ_i とし，これが文字列 S 上へのポインタ (s_i, e_i) で表現されているとする．ここで，$\ell_i[1:|\ell|] = S[s_i : e_i]$ である．更に，すべてのラベルが S 上でだんだんに重なっている場合を考える．すなわち，任意の整数 i, j $(1 \le i < j \le N)$ について，$s_i \le s_j$ かつ $e_i \le e_j$ かつ $s_j < e_i$ である（図 6）．ポインタが指し示す位置で S を分割すると，分割数はたかだか $2N-1$ である．それぞれに変数を割り当て，これを積み上げることで N 個のラベルに対応する変数を表現できる．それぞれの変数は，その区間の文字列長程度の個数の変数によって定義できるので，全部で $O(|S|)$ 個である．ところが，図 6 から明らかなように，可能な限り変数を共有させたとしても，最悪時には，すべてのラベルを表現するために用いる共通部分の変数の総数は，$\sum_{i=1}^{N-1} i = \frac{1}{2}N(N-1)$ となるので，更に $O(N^2)$ 個の変数が必要となる．分節木上の各ノードを表すために，辺ラベル同士をつなげる変数は $O(N)$ 個であるので，結局のところ，辞書に含まれる変数の個数は最悪時に $O(|S| + N^2)$ 個となる．



ℓ_i は，辺のラベルが指し示す文字列の範囲を表す．図中の交点●の位置で新たな変数割当が必要となる．

図 6 ラベル文字列が密に文字列を共有している例
Fig. 6 The worst case any labels overlap each other.

4.1 改善手法

[7] の議論から，関数 *Jump* と集合 *Output* は，それぞれ次のようにして求めることができる．

$$Jump(j, X) = |lsp_P(lsf_P(P[1:j] \cdot X))|,$$

$$Output(j, X) = Occ^{\star}{}_P(P[1:j] \bullet X) \ominus j \cup Occ_P(X).$$

Jump の計算において，lsp_P の部分は [7] の定理 7 から $O(1)$ 時間で答えられる．その内部の lsf_P の部分は，もし $|lsf_P(X)| < |X.u|$ のときは $lsf_P(P[1:j] \cdot X) = lsf_P(X)$ であるし，もし $|lsf_P(X)| = |X.u|$ のときは $X.u \in Fac(P)$ なので [7] の定理 6 より $O(1)$ で答えられる．したがって，各変数 $X \in F(\mathcal{D})$ について，$lsf_P(X)$ を前処理で計算しておけばよい．*Output* の計算については，以下の定理から $lpf_P(X)$ と $Occ_P(X)$ を前処理で求めておけばよいことが分かる．

[定理 2]（[7] 定理 12） Collage system $\langle \mathcal{D}, \mathcal{S} \rangle$ とパターン $P = P[1:m]$ が与えられたとき，任意の整数 $j \in \{0, \ldots, m\}$ と $X \in F(\mathcal{D})$ に対して，$Occ^{\star}{}_P(P[1:j] \bullet X)$ の要素をそのサイズに比例した時間で列挙する手続きは，$lpf_P(X)$ が既に計算されていると仮定すると，$O(m^2)$ 時間・領域で準備できる．

まとめると，圧縮照合アルゴリズムの走査部分で必要となる情報は，任意の $X \in F(\mathcal{S})$ に対して，$lsf_P(X)$, $lpf_P(X)$, $Occ_P(X)$ の三つである．言い換えると，これらの情報が前処理で計算できれば，前章の圧縮照合アルゴリズムを動作させることができる．

まず，$lsf_P(X)$ と $lpf_P(X)$ について議論する．これらは文字列について対称であるので，以下の議論では $lsf_P(X)$ の計算についてのみ議論する（$lpf_P(X)$ は，P と S をそれぞれ反転させた文字列について同様の議論をすれば求められる）．計算量削減のアイデアは，各ラベル文字列 X に対して，$lsf_P(X)$ を $lsf_P(S[1:j])$ と $|X|$ から直接求めることである．

[補題 4] 文字列 S とパターン $P = P[1:m]$ が与えられたとき，任意の整数 $1 \leq j \leq |S|$ について，$lsf_P(S[1:j])$ を $O(1)$ 時間で答える手続きは，$O(|S|+m^2)$ 時間・領域で構築できる．

(証明) 任意の 1 文字 $a \in \Sigma$ について，$a \in Fac(P)$ かどうかは接尾辞トライを使えば分かる．当然，$lsf_P(S[1])$ は $O(1)$ 時間で計算できる．今，$lsf_P(S[1:j-1])$ が計算できているとする．このとき，$lsf_P(S[1:j])$ は，$lsf_P(S[1:j-1])$ と $S[j]$ から次のように計算できる．もし，$S[j] \notin Fac(P)$ とすると，明らかに $lsf_P(S[1:j]) = \varepsilon$ である．逆に $S[j] \in Fac(P)$ とすると，[7] の定理 6 から，$lsf_P(S[1:j]) = lsf_P(lsf_P(S[1:j-1]) \cdot S[j])$ として $O(1)$ 時間で求められる．したがって，任意の j について，$lsf_P(S[1:j])$ をテーブルとして保持すると，$O(|S|)$ 時間・領域でそのテーブルを構築できる．ただし，パターン P について $O(m^2)$ 時間・領域の事前準備が必要となる． □

[補題 5] 文字列 S とパターン $P = P[1:m]$ が与えられたとき，任意の整数 i,j $(1 \leq i,j \leq |S|)$ について，$lsf_P(S[i:j])$ を $O(1)$ 時間で答える手続きは，$O(|S|+m^2)$ 時間・領域で構築できる．

補題 5 を証明する準備として次の補題を証明する．

[補題 6] パターン $P = P[1:m]$ が与えられたとき，任意の $x \in Fac(P)$ と任意の整数 $0 \leq \ell \leq |x|$ について，x の長さ ℓ の接尾辞を $O(1)$ 時間で答える手続きは，$O(m^2)$ 時間・領域で構築できる．

(証明) P の接尾辞トライにおいて，明示的なノードの文字列 y についてのみ，その長さ ℓ の接尾辞を表すテーブル $suf(y,\ell)$ を接尾辞リンクを使って $O(m^2)$ 時間・領域で構築することは容易にできる．暗黙的なノードの文字列 x について，その一番近い明示的な親を $\bar{x}$ とすると，$\bar{x}$ を $O(1)$ で答えるためのテーブルは，接尾辞トライを探索することで $O(m^2)$ 時間・領域で構築できる．これらから，任意の $x \in Fac(P)$ について，その長さ ℓ の接尾辞は $suf(\bar{x},\ell) \cdot w$ として求められる．ここで，$x = \bar{x} \cdot w$ である．これは，文字列 P の部分文字列同士の連結が部分文字列になるかどうか，もし部分文字列になるならば接尾辞トライ上の対応するノードを $O(1)$ で答えよという部分文字列連結問題であり，これは [7] の定理 5 から，$O(m^2)$ 時間・領域の前処理で解決できることが示されている． □

(証明)(補題 5 の証明) 補題 4 より，$lsf_P(S[1:j])$ は $O(|S|+m^2)$ 時間・領域であらかじめ計算できる．任意の整数 i,j $(1 \leq i,j \leq |S|)$ に対して，$|lsf_P(S[1:j])| < |S[i:j]|$ の場合は，明らかに $lsf_P(S[i:j]) = lsf_P(S[1:j])$ であるから，$O(1)$ 時間で返答できる．$|lsf_P(S[1:j])| \geq |S[i:j]|$ の場合，$S[i:j] \in Fac(P)$ となる．また，$S[i:j]$ は $lsf_P(S[1:j])$ の長さ $|S[i:j]|$ の接尾辞でもある．よって，補題 6 を使えば，P に関する $O(m^2)$ 時間・領域の前処理で，$lsf_P(S[1:j])$ の長さ $|S[i:j]|$ の接尾辞として $S[i:j]$ を $O(1)$ 時間で答えることができる． □

以上の補題から，次の定理が導かれる．

[定理 3] 分節木 $\mathcal{T}$ と共有文字列 S による符号の Collage system$\langle \mathcal{D}, \mathcal{S} \rangle$ とパターン $P = P[1:m]$ が与えられたとき，任意の $X \in F(\mathcal{D})$ に対して，$lsf_P(X)$ $(lpf_P(X))$ を $O(1)$ で答える手続きは，$O(m^2+|S|+|\mathcal{T}|)$ 時間・領域で構築できる．

(証明) 補題 5 より，分節木上の任意の辺について，その辺の文字列に対応する変数 X に対する $lsf_P(X)$ $(lpf_P(X))$ を $O(1)$ で答える手続きが $O(|S|+m^2)$ 時間・領域で構築できる．よって，後は分節木の各ノードを $O(|\mathcal{T}|)$ 時間かけて探索しつつ，補題 1 を利用することで，任意の $X \in F(\mathcal{D})$ に対する $lsf_P(X)$ $(lpf_P(X))$ を求めることができる． □

最後に，$Occ_P(X)$ について議論する．$Occ_P(X)$ の要素をその個数分に比例した時間で列挙するだけならば，すべての $X \in F(\mathcal{S})$ に対して，$Occ_P(X)$ を明示的にリストで保持すればよい．しかし，それでは，分節木の上方にある辺ラベルに含まれるパターンの出現がそれ以降の葉の個数分だけ複製されてしまう．一方で，文字列 S 上の出現するパターン P の個数は $O(|S|)$ で抑えられる．各辺のラベルは，S を共有するのと同様，パターンの出現も共有しているので，S 上に出現の情報をもたせることで出現情報の複製を避けることができる．

関数 $o : \{0,\ldots,|S|\} \to \{0,\ldots,|S|\}$ を次のように定義する．

$$o(j) = \begin{cases} 0 & (j \notin Occ_P(S) \text{ の場合}), \\ j-|P|+1 & (j \in Occ_P(S) \text{ の場合}). \end{cases}$$

すなわち，$o(j)$ は，$S[j-|P|+1:j]$ の位置でパターンが出現している場合に，その出現の先頭位置を返す関数である．また，関数 $p:\{0,\ldots,|S|\}\to\{0,\ldots,|S|\}$ を次のように定義する．

$$p(j)=\begin{cases}0 & (Occ_P(S[1:j-1])=\emptyset \text{ の場合}),\\ \max(Occ_P(S[1:j-1])) & (\text{それ以外}).\end{cases}$$

すなわち，$p(j)$ は，S 上の位置 j より手前に出現する最も近いパターン出現位置を返す関数である．これら二つの関数に関して，次の補題が成り立つ．

[補題 7] パターン $P=P[1:m]$ と文字列 S が与えられたとき，関数 $o(j)$ 及び関数 $p(j)$ を $O(1)$ で答える手続きは，$O(|S|+m^2)$ 時間・領域で構築できる．

(証明) 補題 4 より，$lsf_P(S[1:j])$ は $O(|S|+m^2)$ 時間・領域で計算できる．$lsf_P(S[1:j])=P$ となる位置を見つけることは，整数 $1\le j\le|S|$ について $lsf_P(S[1:j])$ を順に問い合わせれば，$O(|S|)$ 時間で行うことができる．またそれを $O(|S|)$ 領域の配列に保持しておけば，$o(j)$ に関する問合せに $O(1)$ 時間で答えることができる．また，同時に，直前の出現位置を記録する配列を構築すれば，$p(j)$ に関する問合せに $O(1)$ 時間で答えることができる． □

[補題 8] パターン $P=P[1:m]$ と文字列 S が与えられたとき，任意の S の部分文字列 $S[i:j]$ に対して，$Occ_P(S[i:j])$ の要素を $|Occ_P(S[i:j])|$ 時間で列挙する手続きは，$O(|S|+m^2)$ 時間・領域の前処理で構築できる．

(証明) 関数 o と関数 p を交互に問い合わせることで，S 上の任意の区間 (i,j) に対して，$O(|Occ_P(S[i:j])|)$ 時間で出現を以下のように列挙できる．

(1) $o(j)<i$ ならば，列挙を終了する．

(2) $o(j)\neq 0$ かつ $i\le o(j)$ ならば，j を出現位置として報告する．

(3) $j\leftarrow p(j)$ とし，ステップ 1 へ戻る．

この手続きは，区間 (i,j) 内に P が出現しない場合には，何も出力せずに 1 回のループで列挙を停止する．P が出現する場合には，それを列挙しつつ，たかだか $|Occ_P(S[i:j])|+1$ 回でループを停止する． □

[定理 4] 分節木 $\mathcal{T}$ と共有文字列 S による符号の Collage system $\langle\mathcal{D},\mathcal{S}\rangle$ とパターン $P=P[1:m]$ が与えられたとき，任意の $X\in F(\mathcal{D})$ に対して，$Occ_P(X)$ をその要素に比例した時間で答える手続きは，$O(m^2+|S|+|\mathcal{T}|)$ 時間・領域で構築できる．

(証明) 補題 8 により，任意の辺について，そのラベル上に含まれるパターンの列挙は解決できる．任意の符号語に対応する変数 X について，$Occ_P(X)$ を列挙するためには，各ノードを境に接続する二つの辺 Y，Z にまたがる出現 ($Occ^{\star}{}_P(Y\bullet Z)$) を列挙する必要があるが，$lsf_P(Y)$ と $lpf_P(Z)$ が計算済みならば，$O(m^2)$ 時間・領域の前処理で，その出現数に比例した時間で列挙できることが [7] の補題 11 で示されている．任意の辺ラベル Y について，$Occ_P(Y)$ を計算したのち，分節木をいったん $O(|\mathcal{T}|)$ 時間かけて探索しつつ P を含まない辺に印を付けておき，符号語に対応する変数 X についての問合せの際に読み飛ばすよう各ノードにポインタを保持しておけば，問合せに $O(|Occ_P(X)|)$ 時間で答えることができる． □

よって，定理 3 と 4 から次の結果が得られる．

[定理 5] パターン $P=P[1:m]$ と VF 符号化テキスト $Z=Z[1:n]$ が与えられたとき，分節木上の各ラベルがある文字列 S へのポインタとして表現されている場合，P と Z に対する圧縮照合問題を，$O(m^2+|S|+|\mathcal{T}|)$ 時間・領域の前処理の後，$O(n+R)$ 時間で解決することができる．ここで，$\mathcal{T}$ は分節木のノード数，R はパターンの出現回数である．

5. むすび

本論文では，分節木 $\mathcal{T}$ 上のラベルが共有文字列 S 上へのポインタとして表されている場合に，その分節木を用いて符号化されたテキストに対する圧縮パターン照合アルゴリズムを Collage System の枠組みを用いて組織的に導出することについて議論し，その際の前処理計算量を $O(m^2+|S|+|\mathcal{T}|^2)$ 時間・領域から $O(m^2+|S|+|\mathcal{T}|)$ 時間・領域に削減できることを示した．ここで，m，$|S|$，$|\mathcal{T}|$ はそれぞれパターン長，共有文字列 S の長さ，分節木 $\mathcal{T}$ の大きさである．

実際，文献 [4] の圧縮法では，分節木の大きさ $|\mathcal{T}|$ は符号長 ℓ に対して $O(2^\ell)$ である．実用的な圧縮率が得られる符号長 ($\ell\ge 16$) に対して $O(|\mathcal{T}|^2)$ は非常に大きく，圧縮パターン照合を高速に行う上での障害であった．したがって，今回の計算量の削減は，圧縮パターン照合の大幅な速度改善につながる．本結果の実際のデータに対する実証実験は今後の課題である．

文献

[1] M. Takeda, Y. Shibata, T. Matsumoto, T. Kida, A. Shinohara, S. Fukamachi, T. Shinohara, and S. Arikawa, "Speeding up string pattern matching by text compression: The dawn of a new era," Trans.

IPSJ, vol.42, no.3, pp.370–384, 2001.
[2] D. Salomon, Data Compression: The Complete Reference, 4th ed., Springer, 2006.
[3] S.T. Klein and D. Shapira, "Improved variable-to-fixed length codes," SPIRE '08: Proc. 15th International Symposium on String Processing and Information Retrieval, pp.39–50, Springer-Verlag, Berlin, Heidelberg, 2009.
[4] T. Kida, "Suffix tree based VF-coding for compressed pattern matching," Proc. Data Compression Conference 2009 (DCC2009), p.449, 2009.
[5] M. Crochemore and W. Rytter, Jewels of Stringology, World Scientific, 2002.
[6] S. Maruyama, Y. Tanaka, H. Sakamoto, and M. Takeda, "Context-sensitive grammar transform: Compression and pattern matching," Proc. 15th International Symposium on String Processing and Information Retrieval (SPIRE 2008), vol.5280 of LNCS, pp.27–38, 2008.
[7] T. Kida, T. Matsumoto, Y. Shibata, M. Takeda, A. Shinohara, and S. Arikawa, "Collage system: A unifying framework for compressed pattern matching," Theor. Comput. Sci., vol.298, no.1, pp.253–272, 2003.
[8] 喜田拓也, "STVF 符号：頻度刈り込み接尾辞木を用いた効率良い VF 符号化," 日本データベース学会論文誌, vol.8, no.1, pp.125–130, 2009.
[9] B.P. Tunstall, Synthesis of noiseless compression codes, PhD thesis, Georgia Inst. Technol., Atlanta, GA, 1967.

（平成 21 年 9 月 11 日受付，22 年 1 月 4 日再受付）

喜田　拓也　（正員）

2001 九州大学大学院システム情報科学研究科博士後期課程了，博士（情報科学）．2001 九州大学附属図書館，研究開発室専任講師．2004 北海道大学大学院情報科学研究科准教授．テキストアルゴリズム及び情報検索技術に関する研究に従事．情報処理学会，日本データベース学会各会員．