2008－AL－117 (12)
2008／3／7

文字列上のビット並列法を利用した木パターン照合アルゴリズム

山本 博章　竹之内大地
信州大学工学部情報工学科

概要. 本論文では木パターン照合問題を考える。木パターン照合問題とは、木パターン P とデータ木 T が与えられたとき、T の部分木で P に一致するすべての部分木を求める問題である。我々はこの問題に対し、文字列上のビット並列法を利用した木パターン照合アルゴリズムを与える。我々は、擬似パターン照合条件という、通常の照合条件を緩めた照合条件を考える。そのとき、無順序木に対して、木パターン P の高さが 1 ワード長以下であれば、我々のアルゴリズムは、$O(nl)$ 時間で擬似パターン照合条件を満足するすべての T のノードを見つけることができる。ここで、n は T のノード数、l は P の葉数である。さらに、P の高さと葉数の積が 1 ワード長以下ならば、アルゴリズムは $O(n)$ 時間で走る。この結果を使うと、順序木に対しては、通常の照合条件にマッチするノードを $O(n + kN(P))$ 時間で見つけることができる。ここで、k は擬似パターン照合するノード数であり、$N(P)$ は、擬似パターン照合するノードが P と一致するかチェックするために訪問する T のノード数であり、$N(P) \leq n$ である。

A Tree Pattern Matching Algorithm using a Bit-Parallel Technique on Strings

Hiroaki Yamamoto　Takenouchi Daichi
Department of Information Engineering, Shinshu University

Abstract In this paper, we deal with a tree pattern matching problem. Let T be a data tree and let P be a tree pattern. We introduce a new concept of a near tree pattern matching rule. Then we present an algorithm which is to find all nodes of T which near match P. We give a bit-parallel algorithm using a bit-parallel technique on strings for unordered trees. Our algorithm runs $O(nl)$ time, where n is the number of nodes of T and l is the number of leaves of P. Furthermore we give an algorithm which runs in $O(n)$ time if the size of P is small. Our algorithm can also be used for efficiently solving an ordered tree matching problem if the number of near matching nodes is small.

1 まえがき

木パターン照合問題または検索問題とは、木のデータから与えられた木パターンに一致する部分を見つける問題である。多くのデータが木構造の形で表現できる中、この問題に対する効率的なアルゴリズムの開発は重要な問題であり多くの研究がなされてきた。最近は、XML に代表されるような半構造データと呼ばれる形式が普及してきており、このようなデータに対する検索技術の一つとして、木パターンの効率的な検索アルゴリズムが求められるようになってきている。実際、XML 検索のために木パターン照合アルゴリズムを開発または応用する研究が活発に行なわれている。

木パターンの照合問題を考える上で、いくつかのバリエーションが存在する。一つは扱う木が順序木か無順序木かである。順序木は、子供に順番が付いた木で、木パターンのマッチングを考える場合、この順番を考慮する。無順序木は子供に順番がないもので、元のデータの中に出現する木パターンについては子供の順番は考えない。さらに、親子関係、先祖・子孫の関係についても木パターンの照合条件においていろいろな定義が存在する。一般に、無順序木に対する木パターン照合問題は難しく、効率なアルゴリズムは得られていない。順序木に対しては、いくつか効率的なアルゴリズムが得られている。[4, 5, 6, 8] 最初に考えられた順序木に対する照合条件は親子関係に加え子供の位置まで一致した条件であり、素朴なアルゴリズムでは $O(mn)$ 時間で木パターン照合問題をできる。ここで、n はデータ木のノード数、m は P のノード数である。これに対しては、Dubiner 他 [5] が $O(n\sqrt{m})$ 時間アルゴリズム与えた。Chauve[4] は、より一般化した照合条件を考え、その下で $O(nl)$ 時間で動作するアルゴリ

ズムを与えた。無順序木に対しても研究されている（例えば、文献 [6, 7, 9] がある）。一般に、無順序木の方が問題として難しくなり、効率的なアルゴリズムの設計が難しい。文献 [7, 9] においても、ある種の制約を設定することによりアルゴリズムを設計している。

本論文では、擬似パターン照合条件と呼ぶ、通常のパターン照合条件を緩和した条件を導入する。そのとき、この条件の下で、T のノードの中で P と一致するすべてのノードを見つけるアルゴリズムを与える。我々のアルゴリズムは、文字列上のビット並列法である Shift-OR 法を利用し、無順序木に対して効率的に擬似パターン照合点を見つけることができる。特に、1 ワード長に収まる小さなパターンに対しては、T のノード数に比例した計算時間で動作する。

順序木に対しては通常の照合条件のもとで照合するノードを検索するアルゴリズムを考える。まず最初に擬似パターン照合点を見つけ、それから通常の照合条件を満足するかチェックすれば通常の照合条件で一致するノードを見つけることができる。そのとき、擬似パターン照合条件に一致するノード数が少なければ、T に比例する時間で計算することができる。

XML では、要素名がノードのラベルとなり、同じラベルは同じ意味を持つノードと考えられるから、木パターンと擬似パターン照合条件で一致するノードは、構造は違うかも知れないが、木パターンと同じ情報を持つと考えることができる。

2 木パターン照合問題

我々は、ラベル付き順序木を考える。順序木は兄弟間に順番が定義された木である。したがって、$d_1, \cdots, d_l$ を兄弟ノードでこの順で順番が定義されているとき、$d_1 < \cdots < d_l$ と記述する。無順序木は、このような順番が存在しない木である。また、木のノードのレベルとは、根をレベル 1 とし、それ以外のノードについては親のレベル $+1$ で定義される。そのとき、木の高さは最も大きなレベルの値として定義される。

さて、このような順序木に対し、本論文では次のような木パターン照合問題を考える。今、T をデータ木、P を木パターンとする。そのとき、次の条件を満足する P のノードから T のノードへの単射 ϕ が存在するとき、P は T に出現するという。特に、P が出現する T の部分で、P の根に対応する T のノードを P の出現点という。

順序木パターン照合条件 C1

- P の任意のノード d に対し、 d のラベルと $\phi(d)$ のラベルは同じである。
- P の任意のノード d_1, d_2 に対し、$d_1 < d_2$ ならば、$\phi(d_1) < \phi(d_2)$ である。
- P の任意のノード d_1, d_2 に対し、d_1 が d_2 の親ならば、$\phi(d_1)$ は $\phi(d_2)$ の親である。

無順序木に対しては以下のように定義される。

無順序木パターン照合条件 C2

- P の任意のノード d に対し、 d のラベルと $\phi(d)$ のラベルは同じである。
- P の任意のノード d_1, d_2 に対し、d_1 と d_2 が兄弟ならば、$\phi(d_1)$ と $\phi(d_2)$ も兄弟である。
- P の任意のノード d_1, d_2 に対し、d_1 が d_2 の親ならば、$\phi(d_1)$ は $\phi(d_2)$ の親である。

さて、我々は次のようなパターン照合条件を緩めた擬似パターン照合条件を導入する。これは、P と T のノード間の 1 対 1 対応の条件を緩和し、パスについて制限をつけたものである。

擬似パターン照合条件 C'

- T の任意のノード d に対し、P が d で擬似パターン照合するとは、P の任意のパスに対し、同じラベル列を持つパスで d から始まるものが存在するときである。

例えば、d が無順序木パターン照合条件で P とマッチすれば、明らかに擬似パターン照合条件を満足する。しかし、逆は成り立つとは限らない。T のノードで擬似パターン照合条件でマッチするノードを、擬似パターン照合点と呼ぶ。

本論文では、P と擬似パターン照合する T 内のすべてのノードを求めるアルゴリズムを与え、そのあと、順序

木に対するパターン照合問題について考える。

3 文字列照合における Shift-OR 法

まず、文字列上のビット並列法である Shift-OR 法について説明する。これはコンピュータのワード単位で処理を行なうシフト演算と論理和を使ったビット並列文字列照合アルゴリズムであり、パターンの長さがワード長以内ならば、検索文字列の長さに比例した時間で動作する。以下に、アルゴリズムの概略を述べる。

今、$P = p_1 \cdots p_m$ を文字列パターン、$T = t_1 \cdots t_n$ をテキストとする。P に含まれる文字 c に対し、ビットマスクを要素とする配列 $B[c]$ を定義する。このとき、配列 $B[c] = b_1 \cdots b_m$ のとき、このビットマスク $b_1 \cdots b_m$ は次を満足するように設定される：任意の j に対し、$b_j = 0$ なる必要十分条件は $t_j = c$ のときである。

さて、検索は照合状態を示すビットベクトル $D = d_1 \cdots d_m$ を使って、t_1 から順番にテキストの文字を読み込みながら、D を以下の式で更新しながら行なう。まず、D の初期値はすべて 1 である。文字 t_j を読んだとき D を以下の式で更新する。

$$D \leftarrow (D \ >> \ 1) \mid B[t_j]$$

ここで、$D \ >> \ 1$ は D を 1 ビット右へシフトする演算を表す。左端には 0 が埋められる。また、演算 $\mid$ はビット毎の論理和を行なう演算である。これを行なったあと、もし $d_m = 0$ ならば、$t_{j-m+1} \cdots t_j$ は P とマッチする文字列となる。

例 1 $P = aabca$、$T = abcaabca$ を考える。そのとき、ビットマスクは $B[a] = 00110$、$B[b] = 11011$、$B[c] = 11101$ となる。$D = 11111$ でスタートする。最初、$T[1] = a$ であるから、D を 1 ビット右へシフトし、$D \mid B[a]$ を計算すると、$D = 01111$ となる。最後の $T[8] = a$ を計算すると、$D = 01110$ となり、右端のビットが 0 なのでここで P が出現したことが分かる。

4 ビット並列型木パターン照合アルゴリズム

さて、本章では、擬似パターンに対し、前章で説明した文字列上の Shift-OR 法利用したビット並列型照合アルゴリズムを示す。文字列上の手法を利用するために木パターンをパスに分解し、データ木上で葉から根に向かってポストオーダーの順で計算していく。さて、P を木パターン、T をデータ木とする。そのとき、アルゴリズムの概略を述べる。以下、n を T のノード数、l を P の葉数とする。

1. 木パターン P を、根から葉へのパス P_i $(1 \leq i \leq l)$ に分割する。これをパスパターンと呼ぶ。
2. 各パスパターンに対し、データ木 T 上で葉から根に向かって Shift-OR 法を適用する。この時、各パスパターンで同期を取りながら照合操作を行なう。
3. T のノードで、すべてのパスパターンの根に対応する部分のビットが一致を示す（すなわち、0 になる）ならば、そのノードがパターン P の擬似パターン照合点である。

上記の流れに沿って、より詳細にアルゴリズムを記述する。

4.1 木パターンの分割とビットマスクの生成

Shift-OR 法を適用するため、木パター P を根から葉へのパス単位で分割し、パスパターンを作成する。各パスパターン P_i に対し、ビットマスクを生成する。今、h を P の高さ、$P_i = q_1 \cdots q_e$ を Q の任意のパスパターンとする。そのとき、P_i に含まれる文字 c に対し、ビットマスクを要素とする配列 $B[P_i, c] = b_1 \cdots b_e \cdots b_h$ を定義する。文字列のときと同様に、このビットマスク $b_1 \cdots b_h$ は次を満足するように設定される：任意の $1 \leq j \leq e$ に対し、$b_j = 0$ なる必要十分条件は $q_j = c$ のときである。任意の $e+1 \leq j \leq h$ に対し、$b_j = 0$ と設定する。

図 3 にビットマスクを計算するアルゴリズムを与える。このアルゴリズムは、l および h を P の葉数および高さ、また、α を P に出現する異なる記号の数としたとき、$O(l(h+\alpha))$ 時間で走る。

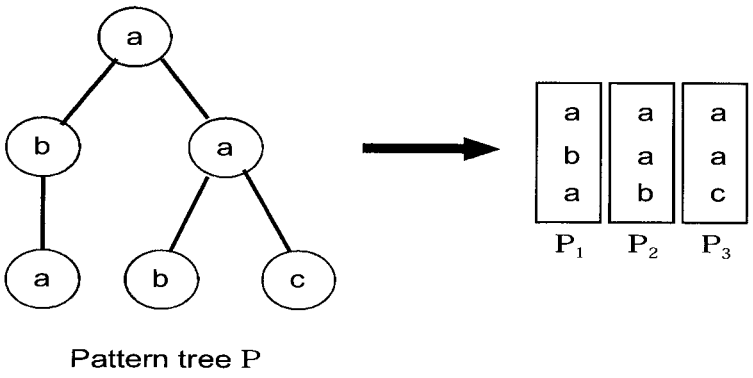

図1　木パターンのパスへの分解

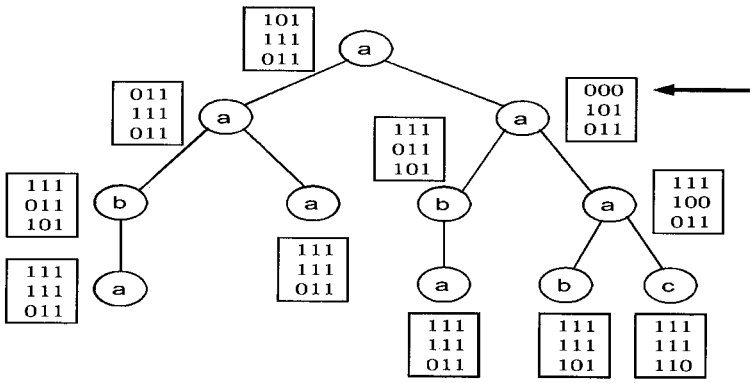

図2　TreeMatch の動作例

例 2 図1に例を与える。左の木パターン P は右に示した3つのパス P_1, P_2, P_3 に分割される。そのとき、ビットマスクは以下のように定義される。

$B[P_1, a] = 010$, $B[P_2, a] = 001$, $B[P_3, a] = 001$
$B[P_1, b] = 101$, $B[P_2, b] = 110$, $B[P_3, b] = 111$
$B[P_1, c] = 111$, $B[P_2, c] = 111$, $B[P_3, a] = 110$

4.2　擬似パターンに対するアルゴリズム

文字列と同様に、パスパターンに対応したビットマスクの配列 $B[P_i, c]$ を使って、T 上で検索を行なう。T の各ノードは、照合状態を示すビット列の配列 $M[P_i]$ が割り当てられる。アルゴリズムはこの $M[P_i]$ を計算していく。ビット演算を簡単に行なうため、ビット列配列 $M[P_i]$ の長さを P の高さ h に統一する。すなわち、$M[P_i] = b_1 \cdots b_h$ とする。そのとき、これは次を満足するように計算される：m をパス P_i の長さ、$M[P_i]$ が T のノード d に割り当てられているとする。そのとき、任意の $1 \leq j \leq m$ に対し、ビット $b_j = 0$ である必要十分条件は d からのパスで p_j からのパスとラベルが同一のものが存在する。

さて、アルゴリズムを図4に与える。任意のパス P_i に対し、$M[P_i] = b_1 \cdots b_e \cdots b_h$ の初期値は、$b_1, \cdots, b_e$ には1をセットし、$b_{e+1}, \cdots, b_h$ には0をセットする。演算 & はビット毎の論理積を表す。

Procedure MakeBitMask(P)

/* 木パターン P からビットマスク $B[P_i, \sigma]$ を作成する*/

Step 1. P の各ノードをポストオーダーの順で巡り、各ノードから葉に向かって出るパスの数を計算する。

Step 2. すべての記号 σ に対し、$B[P_1, \sigma]$ を 1 からなるビット列に初期化する。他のパス P_i $(i > 1)$ に対しては、$B[P_1, \sigma]$ をコピーすることによって初期化する。

Step 3. 深さ優先探索の順で P の各ノード d を巡り、以下を行なう：ノード d のパス数を l_d, レベルを $level_d$ とする.

1. d と同じレベルですでに訪問したノードのパス数の合計を計算する。それを l' とする。
2. d のラベルを σ としたとき、$B[P_{l'+1}, \sigma]$ から $B[P_{l'+l_d}, \sigma]$ の $level$ 番目のビットを 0 にセットする。

図 3　The Procedure *MakeBitMask*

Algorithm TreeMatch(P, T)

/* データ木 T のノードの中で、木パターン P と擬似照合するものをすべて見つける。*/

T のノードをポストオーダーで訪問し、各ノードに対し、次の処理を行なう。今、対象となるノードを d とする。

Step 1. 照合状態の計算

1. ノード d の子を $d_1, \cdots, d_t$ とし、それらの照合状態を $M_1, \cdots, M_t$ とする。また、d のラベルを σ とする。
2. すべての P_i に対し、
   (a) $M[P_i] = M_1[P_i] \& \cdots \& M_t[P_i]$,
   (b) $M[P_i] = (M[P_i] \ << \ 1) \mid B[P_i, \sigma]$

Step 2. マッチングの判定

1. $Temp = M[P_1] \ \& \ \cdots \ \& \ M[P_l]$,
2. $Temp$ の最初のビット (P の根に対応するビット) が 0 ならば、d を P の擬似パターン照合点として出力する。

図 4　The algorithm *TreeMatch*

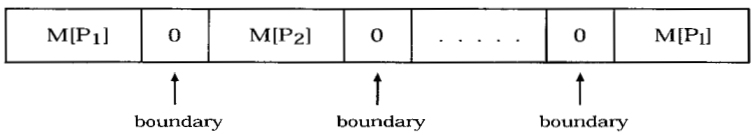

図 5　照合状態の格納

定理 1 アルゴリズム *TreeMatch* は、木パターン P の高さがワード長以下ならば、$O(nl)$ 時間かつ領域ですべての擬似パターン照合点を見つけることができる。

例 3 アルゴリズム *TreeMatch* に対する動作例を図 2 に与える。これは図 1 に与えた木を木パターンとした場合である。矢印の付いたノードで木パターンの根に対応するビットがすべて 0 になっているのが分かる。したがって、このノードは擬似パターン照合点となる。

B[a]	010	0	001	0	001
B[b]	101	0	110	0	111
B[c]	111	0	111	0	110

図6　図1におけるビットマスクの格納

Algorithm TreeMatchFast(P, T)

/* データ木 T のノードの中で、木パターン P と擬似照合するものをすべて見つける。*/

T のノードをポストオーダーで訪問し、各ノードに対し、次の処理を行なう。今、対象となるノードを d とする。

Step 1.　照合状態の計算

1. ノード d の子を $d_1, \cdots, d_t$ とし、それらの照合状態を $M_1, \cdots, M_t$ とする。また、d のラベルを σ とする。
2. $M = M_1 \& \cdots \& M_t$,
3. $M = (M \ << \ 1) \mid B[\sigma]$,
4. $M = M \ \& \ ZMask$

Step 2.　マッチングの判定

1. $Acc = M \mid AccCheck$
2. もし $Acc = AccCheck$ ならば、d を P の擬似パターン照合点として出力する。

図7　The algorithm *TreeMatchFast*

4.3　アルゴリズムの改良

ここでは、より効率的に検索できるようにアルゴリズムを改良する。すなわち、1ワードにすべてのパスのビットマスク詰め込んで計算することにより高速化を図る。これによって、パスごとに行なっていたシフト演算、論理演算がまとめて1回の演算でできるようになる。特に、木パターン P の高さと葉数の積が1ワード長に収まるならば、T のノード当り定数回のビット演算で済み、T のノード数に比例した時間で処理できる。

さて、ビットマスクを要素とする配列 $B[P_i, c]$ と照合状態を示すビット列の配列 $M[P_i]$ を1ワードに格納することを考える。すなわち、前章では各 $B[P_i, c]$ および各 $M[P_i]$ を1ワードに格納したが、ここでは、$B[c] = (B[P_1, c], \cdots, B[P_l, c])$ を1ワード、また、$M = (M[P_1], \cdots, M[P_l])$ を1ワードに格納して計算する。任意の $1 \leq l-1$ に対し、$B[P_j, c]$ と $B[P_{j+1}, c]$ や $M[P_j]$ と $M[P_{j+1}]$ の境界を設定するため1ビット使う。したがって、$B[P_j, c] = b_1 \cdots b_{m+1}$ および $M[P_j] = b_1 \cdots b_{m+1}$ と定義される。例えば、照合状態を示す $M[P_i]$ は図5の様に格納される。また、図6は図1で示したパスに対するマスク $B[a] = (B[P_1, a], B[P_2, a], B[P_3, a])$、$B[b] = (B[P_1, b], B[P_2, b], B[P_3, b])$、$B[c] = (B[P_1, c], B[P_2, c], B[P_3, c])$ を格納した様子を示す。

さらに、境界を示す1ビットが常に0になるよう、$ZMask = 1^{m_1}0 \cdots 1^{m_l}0$ を使う。ここで、m_i はパス P_i の長さを示す。図6に示すように、シフト動作は $M[P_1], \cdots, M[P_l]$ の並で左へ1ビット行なわれるため、境界ビットがないと $M[P_{j+1}]$ の先頭のビットが $M[P_j]$ の最後につくことになり望ましくない。そこで、境界用に1ビット設定する。ただし、この境界ビットは0である必要があるため効率的に計算できるように $ZMask$ を導入した。これとの論理積を取ることによって、常に境界ビットを0に維持することができる。

最後に、擬似パターンの出現点かどうかの判定に、判定マスク $AccCheck = 01^{m_1} \cdots 01^{m_l}$ を使う。照合状態を示すビット列は、この判定マスクと論理和を取ることによって、各パス P_i の根に相当する部分に対し、もしそれが0なら0を返し、その他のビットについてはすべて1にするよう設定されている。したがって、照合状態の根

に対応する部分がすべての0ならば、照合状態と判定マスクの論理和を取ると *AccCheck* と値が一致する。また、一致するのはこの時だけである。したがって、擬似パターン照合条件を満足しているか簡単に判定できる。改良したアルゴリズムを図7に与える。そのとき、次の結果を得る。

定理 2 アルゴリズム *TreeMatchFast* は、$O(n \times \lceil \frac{h \times l}{W} \rceil)$ 時間かつ領域ですべての擬似パターン照合点を見つけることができる。ここで、h と l はそれぞれ木パターン P の高さと葉数、W はコンピュータのワード長を表す。

この結果から直ちに次の系を得る。これは、小さいパターンに対しては高速に動作することを意味している。

系 1 アルゴリズム *TreeMatchFast* は、木パターン P の高さと葉数の積が1ワード長以下ならば、$O(n)$ 時間かつ領域ですべての擬似パターン照合点を見つけることができる。

実際、$h \times l$ がワード長の定数倍ならば、線形時間でアルゴリズムは走る。

4.4 順序木に対するパターン照合アルゴリズム

ここでは、順序木に対し、順序木パターン照合条件 $C1$ の下での木パターン照合問題を考える。このとき、擬似パターン照合点を見つけるアルゴリズムを照合条件 $C1$ をチェックするノードを絞るために利用することができる。そのとき、k を擬似パターン照合点の数とすると次の結果を得る。

定理 3 我々は、$O(n \times \lceil \frac{h \times l}{W} \rceil + kN(P))$ 時間かつ領域ですべての順序木パターン照合点を見つけることができる。ここで、h と l はそれぞれ木パターン P の高さと葉数、W はコンピュータのワード長を表す。

なお、$N(P)$ は T に出現する擬似パターン照合点に対し、照合条件 $C1$ を満足するかチェックするする時間である。これは P のサイズが大きいほど大きくなるが、上限は $N(P) \leq n$ となる。したがって、前と同様に、小さなパターンに対しては、もし k が定数ならば、アルゴリズムは $O(n)$ 時間で走る。

文献 [2, 5] で定義されているような子供の位置まで一致の条件に入れた照合条件に対しては、$N(P)$ の上限は P のノード数となる。したがって、このような条件で我々は $O(n + k|P|)$ 時間で順序木パターン照合問題を解くことができる。ここで、$|P|$ は P のノード数を表す。

参考文献

[1] A.V. Aho, Algorithms for finding patterns in strings, In J.V. Leeuwen, ed. Handbook of theoretical computer science, Elsevier Science Pub., 1990.

[2] A. Apostolico, Z. Galil ed., Pattern Matching Algorithms : Tree Pattern Matching in Chapter 11, Oxford University Press, 1997.

[3] R. Baeza-Yates and G.H. Gonnet, A New Approach to Text Searching, Communications of the ACM, 35, 10, (1992) 74-82.

[4] C. Chauve, Tree pattern matching with a more general notion of occurrence of the pattern, Information Processing Letters, 82, (2001) 197-201.

[5] M. Dubiner, Z. Galil, and E. Magen, Faster Tree Pattern Matching, Journal of the ACM, 41, 2, (1994) 205-213.

[6] P. Kilpelainen and H. Mannila, Orderes and Unordered Tree Inclusion, SIAM J. Comput., 24, 2, (1995) 340-356.

[7] T. Shoudai, T. Uchida, and T. Miyahara, Polynomial Time Algorithms for Finding Unordered Tree Patterns with Interval Variables, Proc. of FCT 2001, LNCS , (2001) 335-346.

[8] H. Tsuji, A. Ishino, and M. Takeda, A Bit-Parallel Tree Matching Algorithm for Patterns with Horizontal VLDC's, Proc. of SPIRE 2005, LNCS 3772, (2005) 388-398.

[9] J.T. Yao, M. Zhang, A Fast Tree Pattern Matching Algorithm for XML Query, Proc. of the WI'04, (2004).