木分割アルゴリズムを用いたXSLT実行手法

筑波大学
図書館情報メディア研究科
2017年3月
ZHU YATING

目次

第 1 章

はじめに

XML[7](Extensible Markup Language) は Web のデータ形式における事実上の標準として広く用いられている．ここで，ある程度まとまった量の XML データを管理・蓄積する場合を考える．このような場合，DTD(Document Type Definition) 等のスキーマ言語を用いて格納すべきデータの構造を前もって定義しておき，それに対して妥当なデータを作成・利用するのが一般的である．データを蓄積・保存する場合，XML は非常に有用な言語であるが，ユーザに提示する際には HTML(HyperText Markup Language) など他の形式に変換した方が都合がよいことが多い．このような変換を行うための変換用言語として，XSLT[8](XML Stylesheet Language Transformations) がよく使われている．

ここで，XML データに対して XSLT を用いて変換処理を行うことを考える．近年，サイズの大きな XML データが急速に増加しており，そのようなデータに対する変換処理の効率が悪化するという問題が生じている．これまで，サイズの大きい XML データの処理に関しては，XPath (XML Path Language) の実行手法 [4] については研究されているが，XSLT の実行手法に関する研究は行われていない．近年，XML データのサイズが増加する一方で，計算機プロセッサの性能は大幅に向上している．このため，この状況に応じた効率の良い XSLT 変換手法の重要性が増大している．Java などのプログラミング言語自体は複数スレッドの生成・実行に対応しているが，XSLT 処理系で複数スレッドの実行を考慮して処理効率の向上を図ったものは著者の知る限り存在しない．

そこで本論文では，並列化によって XSLT 変換をより高速に行うための手法を提案する．より具体的には，DTD を用いて XML データを分割すべき箇所を求め，そこで変換処理を切り分けることで，XML データの変換処理を並列化し，処理効率の向上を図る手法を提案する．XML データと比較すると，DTD は十分に小さく，短時間で分割位置を決定できるという利点がある．

提案アルゴリズムを Java を用いて実装し，評価実験を行った．その結果，本アルゴリズムを用いた場合，並列化しない処理手法を用いた場合と比較して，処理効率の向上が認められる結果が得られた．

関連研究

本論文に最も関連が強いのは，Rajesh らの研究 [4] である．この研究は，データ分割，クエリ分割，ハイブリッド分割という XML データや XPath 式を分割する 3 つの分割実行戦略に基づいた，並列化アルゴリズムを提案している．しかし，これらのアルゴリズムでは，データの分割位置や XPath 式の分割位

置を決定するために大量の統計情報を収集する必要がある．したがって，分割位置決定処理の負荷が大きいという問題がある．また，これらのデータが更新された場合，大量の統計情報を再計算する必要がある．その他，文献 [3] では，木の各ノードに重みをつけ，木の分割数が最小となるように分割をする木分割線形時間アルゴリズムを提案している．文献 [2] では，分散 XML に対する XSLT 実行手法を提案している．これは，分割されている XML データを並列に変換するものである．

本論文の構成は以下の通りである．2 章では，XML，DTD，XSLT，木変換機に関する定義を行う．3 章では，本研究で提案する木分割アルゴリズムについて説明する．4 章では，評価実験について述べる．5 章では，まとめと今後の課題を述べる．

第 2 章

諸定義

本章では，XML，DTD 及び XSLT に関する諸定義を行う．

2.1 XML について

2.1.1 XML の概要

XML とは，文書やデータの意味や構造を記述するためのマークアップ言語の一つであり，近年，様々なデータの記述フォーマットとして急速に普及している．マークアップ言語とは「タグ」という特別な文字列によって文書の一部を囲う，木構造をした言語であり，文書の意味や構造を記述することができる．また，XML は HTML や TEX とは異なり，ユーザが作成する独自のタグによって，データの意味や構造を表すことができる．このため，XML 自身には特定のデータ構造（タグやその階層関係）が規定されておらず，ユーザによって様々な構造の XML データが生成される．しかし，大量の XML データを管理するために，XML データの構造を統一することが必要な場合がある．そこで，スキーマ言語を用いて XML データの構造を定義し，その定義に従って XML データを生成することにより，XML データの構造を統一する．このスキーマ言語の定義に従って生成された XML データを妥当な XML データと呼ぶ．本論文では妥当な XML データを対象とする．XML データの例を図 2.1 に示す．

```
<item>
 <books>
  <book>
   <titile>吾輩は猫である</title>
   <author>夏目漱石</author>
  </book>
 </books>
</item>
```

図 2.1　XML データの例

図 2.1 に示した XML データは，ルートの子要素が item，item の子要素が books，books の子要素が book，book の子要素が title，author である XML データを表している．

2.1.2　木と生け垣の定義

XML に関係の深い，木 (tree) 及び生け垣 (hedge) について定義する [6].

木とは，任意の 2 つのノード間で，パスが一つのみであり，ループを持たないグラフのことである．木において，あるノードを選んで，それを一番「上」にあると考えると，そのノードを基準として 2 つノードに上下の関係を考えることができる．このとき，その一番上のノードを根（root）という．二つノードを連結するのは枝（edge）と呼ぶ．1 つ枝の両端点と考えると，根に近いノードは親ノード（parent node），根から遠いノードは子ノード（child node）という．根は親ノードを持たないノードであり，子ノードを持たないノードを葉ノード（leaf node）と呼び．1 つの親ノードは多数の子ノードが持てるが，1 つの子ノードは 1 つの親ノードしか持たない．同じ親ノードを持つ多数ノード互いに兄弟ノード（sibling node）という．あるノードと考えると，根からそのノードまでのパス上で，そのノードを除いてすべてのノードを祖先ノード（ancestor node）と呼び，そのノードの子ノード，さらに子ノードの子ノード，葉ノードまでのすべてのノードはそのノードの子孫ノード（descendant node）と呼び，そのノードを根とすると，そのノードを含めてすべての子孫ノードから構成する木を部分木と呼ぶ．また，各ノードに重みが付いている木を，「重み付き木」と呼ぶ.

木の簡単な例を図 2.2 に示す．ノード a は根ノードであり，ノード b，d，f，g，h は葉ノードである．ノード e を基準とすると，ノード c は親ノード，ノード f は兄弟ノード，ノード g と h は子ノードである．ノード e の祖先ノードは a，c であり，子孫ノードは g，h である．また，ノードに付いている数字は重みを表し，破線で囲まれた部分は一つの部分木を表す.

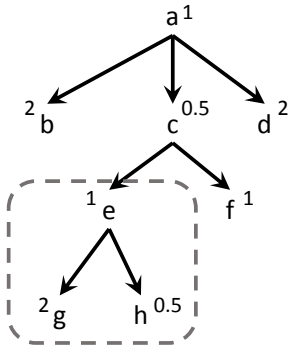

図 2.2　木の例

ノード名（ラベル）の有限集合を Σ とする．Σ で構成された木 (Σ-tree) を $\mathcal{T}_\Sigma$ と表す．ノード $\sigma \in \Sigma$ が子として木 $w \in \mathcal{T}_\Sigma$ を持つとき，$\sigma(w)$ と表すことができ，$\sigma(w)$ も $\mathcal{T}_\Sigma$ に属する木である．上記のように $\mathcal{T}_\Sigma$ は Σ 上の文字列及び括弧を表す記号'(' と')' で表現される．本論文では単なる木と表記した場合は全て，ノード名の有限集合 Σ で構成された木 (Σ-tree) のことを指す．生け垣は木の有限の並び (sequence) である．生け垣の集合 $\mathcal{H}_\Sigma$ は $\mathcal{T}_\Sigma^*$ として定義される. またノード v のノード名を label(v) と表し，ノード

v の重みを weight(v) と表す．

生け垣 $h \in \mathcal{H}_\Sigma$ に対して，h を構成するノード集合 $Dom(h)$ は以下のように定義される:

- $h = \epsilon$ のとき $Dom(h) = \emptyset$ ；
- $h = t_1 \cdots t_n$ であり, $t_i \in \mathcal{T}_\Sigma$ のとき $Dom(h) = \bigcup_{i=1}^{n} \{iu \mid u \in Dom(t_i)\}$ ；
- $h = \sigma(w)$ のとき $Dom(h) = \{\epsilon\} \cup Dom(w)$ ．

以上の定義により，ルートノードを持つ XML データは木で，ルートノードを持たない XML（部分）データは生け垣で表すことができる．また本節以降，$t, t_1, t_2, \cdots$ は木を表し，$h, h_1, h_2, \cdots$ は生け垣を表す．また $h = t_1 \cdots t_n$ と表記されていた場合，全ての t_i は木である．

2.2　DTD の概要

DTD は，XML データの文書構造を定義するスキーマ言語である．DTD を用いてデータ構造を定義することによって，XML データの利用における処理の正確性や安全性を向上させることが可能となる．DTD を用いて定義できるのは，XML データ内に記述できる要素やその出現順序，出現回数，要素が持つ属性，属性の型などである．ただし，本論文では要素のみを考慮し，属性は扱わないものとする．また，XML データの文書構造を定義できるスキーマ言語として，DTD のほか RELAX NG や XML Schema などがあるが，本論文は DTD のみを対象とする．DTD の例を図 2.3 に示す．

```
<!ELEMENT item (books)>
<!ELEMENT books (book+)>
<!ELEMENT book (title, subtitle?, author*)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT subtitle (#PCDATA)>
<!ELEMENT author (#PCDATA)>
```

図 2.3　DTD の例

図 2.3 に示した DTD は，前述の XML データの例に基づいた DTD である．この DTD では 6 つの要素について定義している．1 行目では，item 要素の子要素として books 要素が出現することを定義している．同様に，2 行目では books 要素の子要素として book 要素が出現することを定義し，3 行目では book 要素の子要素として，title，subtitle，author が出現することを定義している．?，+，*は要素の出現回数を定義する演算子であり，?は 0 または 1 回，+ は 1 回以上，*は 0 回以上出現することを示す．4 行目以降で使われている # PCDATA は任意の文字列を表している．

図 2.1 に示した XML データは図 2.3 の DTD に定義に従っているため，図 2.3 の DTD に対して妥当な XML データであるといえる．また，図 2.4 に図 2.3 の DTD に対して妥当でない XML データの例を示す．

```
<item>
 <books>
  <book>
   <titile>吾輩は猫である</title>
   <isbn>9784041001011</isbn>
  </book>
 </books>
</item>
```

図 2.4　妥当性が満たされない XML データの例

図 2.4 に示した XML データが妥当でない理由は，図 2.3 の DTD では定義されていない isbn 要素が出現しているためである．なお，author 要素については，図 2.3 の DTD では 0 個以上出現すると定義されているため，author 要素が出現しなくても図 2.3 の DTD の定義に従っていることになる．

2.3　XSLT について

2.3.1　XSLT の概要

XSLT[8] は W3C により標準化された，XML 文書を他の文書に変換 (transform) する変換用言語である．ここでの変換とは，XML 文書の構造，マークアップ，内容を変更して他の形態の文書にすることである．XSLT 処理系は入力として 2 つのもの，変換処理の規則を記述した XSLT スタイルシートと入力木と呼ばれる変換元となる文書を必要とする．そして，XSLT 処理系は入力木に XSLT スタイルシートに従って変換を行って文書である出力木を出力する．XSLT スタイルシートはテンプレートと呼ばれる単位で変換の規則を記述していく．

テンプレートに最低限必要な要素は，そのテンプレートがどのノードに適用するかを決定する match 属性と，変換の際の動作モードを指定する mode 属性，そして変換の規則や出力木に出力される要素や文字データが記述される処理内容である．match 属性には XPath 式のサブセットである XSLT パターンが指定できる．mode 属性については省略が可能であるが，省略された場合には mode 属性がないモードとして処理が行われていく．処理内容には XSLT が標準で用意している関数や命令が使用できるほか，ユーザ自身が定義した関数も使用できる．

2.3.2　下降型木変換機の定義

XSLT の基盤をなす下降型木変換機について定義する [6]．以下，下降型木変換機を単に木変換機と呼ぶ．簡単のため，ノード名及び変数は全てアルファベット 1 字で表記する．また $\mathcal{H}_\Sigma(Q)(\mathcal{T}_\Sigma(Q))$ で表される生け垣 (木) は Q に含まれる要素でラベルされた葉ノードを持つことが可能である．

木変換機は 4 次組 (Q, Σ, q_0, R) で表させる．ここで，

- Q は木変換機の状態を示す変数の有限集合，
- Σ は入力及び出力で使用するノード名，
- $q_0 \in Q$ は木変換機の初期状態，
- R は $(q, a) \to c(q_1, q_2, \cdots, q_n)$ の形をした遷移規則の有限集合．ただし，$q_1, q_2, \cdots, q_n \in Q$，$a, c \in \Sigma$

である．

$Tr = (Q, \Sigma, q_0, R)$ で定義される木変換機による，状態 q での木 t に対する変換は $Tr^q(t)$ として表記され，以下のように再帰的に定義される．

- $t = \epsilon$ のとき，$Tr^q(t) := \epsilon$ である．
- $t = a(t_1 \cdots t_n)$ かつ，$a(q, a) \to h \in R$ が存在するとき，$Tr^q(t)$ は h 内の $p \in Q$ でラベルされた全てのノード u を生け垣 $Tr^p(t_1) \cdots Tr^p(t_n)$ に置き換えにることによって得られる．そのような Q に含まれるアルファベットでラベルされたノード u は葉ノードにのみ現れる．それゆえ h は下方向にのみ拡張されていく．
- $(q, a) \to h \in R$ となる規則が存在しないとき，$Tr^q(t) := \epsilon$ となる．

最後に，Tr による t の変換を単に $Tr(t)$ と記述した場合，それは初期状態による変換 $Tr^{q_0}(t)$ を意味する．

例として，木変換機 T_r による木（図 2.5）の変換を考える．

木変換機 Tr

$Tr = (Q, \Sigma, p, R)$，$Q = \{p, q\}$ であり，R が以下の 4 つの規則から成る [6]

$$(p, a) \to d(e) \qquad (p, b) \to c(pq)$$
$$(q, a) \to cq \qquad (q, b) \to d(q)$$

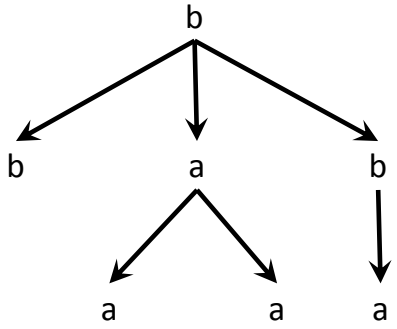

図 2.5　入力木 T

$T_r(T)$ は木 T'(図 2.6) を出力する．定義された木変換機による変換には重要な 2 つの特徴がある．それは複製と消去である．$(p, a) \to c(pq)$ による変換によって，入力木のカレントノードの子ノードは 2 回

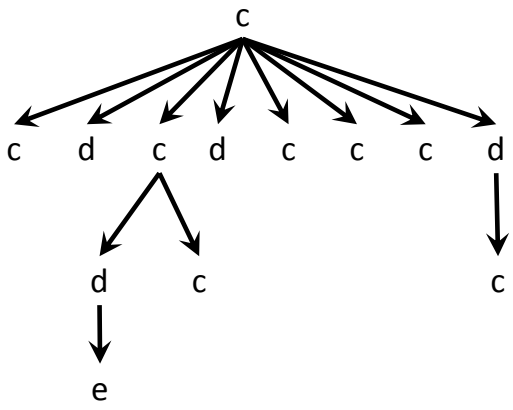

図 2.6　出力木 T'

複製されている．一つの複製では状態が p として変換処理が進んでいき，もう一つの複製では状態が q として変換処理が進んでいく．そして要素 c はそれら二つの複製の親ノードとなっている．したがって，入力木のカレントノードの子ノードに対応する出力木のノードが複数存在することになり複製と呼ばれる．一方 $(q, a) \to cq$ も同様に 2 回子ノードの複製が行われているが，この場合は一つの複製は単なる要素 c に変換され，もう一つの複製では状態が q として変換処理が進んでいくが，この複製の親ノードは与えられていない．それゆえ入力木でのカレントノードに対応する階層のノードは存在しなくなるため，これは消去と呼ばれる．例として，$Tr^q(a(aa))$ は ccc と変換され，変換後には入力木で存在した階層関係は存在しなくなっている．

2.3.3　制限された XSLT

XSLT を木変換機の機能に制限した場合，XSLT テンプレートは以下の様に制限される．

- match 属性に指定できる XSLT バターンは単一のノード名に限定される
- 処理内容に記述できる関数及び命令は apply-templates 命令に限定される
- 処理内容に出力される要素の記述は認めるが文字データの記述は認めない

2.3.2 節で例として挙げた木変換機 T_r を XSLT スタイルシートを表すと図 2.7 の様になる．

Tr の各規則が 1 つのテンプレートに対応している．例えば，変換規則 $(q, a) \to cq$ の左辺の q は 14∼17 行目のテンプレートでの mode 属性に，a は match 属性にあたる．また変換規則右辺での $c \in \Sigma$ は要素 c にあたり，$q \in Q$ は 16 行目の ‘<xsl:apply-templates mode=”q”/>’ というノードに対応する．Tr での初期状態 $p \in Q$ に相当するものとして，ルートノードに appley-templates 命令を適用するテンプレートが 4∼6 行目に存在している．

```
1.   <?xml version="1.0"?>
2.   <xsl:stylesheet
     xmlns="http://www.w3.org/1999/XSL/Transform"
     version="1.0">
3.
4.   <xsl:template match="/ ">
5.       <xsl:apply-templates mode ="p" />
6.   </xsl:template>
7.
8.   <xsl:template match="a" mode ="p">
9.       <d>
10.        <e/>
11.      </d>
12.  </xsl:template>
13.
14.  <xsl:template match="a" mode ="q">
15.      <c/>
16.        <xsl:apply-templates mode ="q" />
17.  </xsl:template>
18.
19.  <xsl:template match="b" mode ="p">
20.      <c>
21.        <xsl:apply-templates mode ="p">
22.        <xsl:apply-templates mode ="p">
23.      </c>
24.  </xsl:template>
25.
26.  <xsl:template match="b" mode ="q">
27.      <d>
28.        <xsl:apply-templates mode ="q" />
29.      </d>
30.  </xsl:template>
31.  </xsl:stylesheet>
```

図 2.7　木変換機 Tr の XSLT スタイルシートによる表現

第 3 章

木分割アルゴリズムを用いた XSLT 実行手法

本章では，木分割アルゴリズムを用いた XSLT 実行手法について述べる．

3.1 提案手法の概要

本手法は，主に DTD に関する処理と変換処理の 2 つの処理に分けられる．3.1.1 節と 3.1.2 節で詳しく説明する．

3.1.1 DTD に関する処理

DTD に関する処理は，DTD を木に変換する処理と DTD の木分割処理の 2 つの処理に分けられる．まず，DTD を木に変換する処理を説明する．DTD では異なる内容モデルが同じラベルを含む場合があるため，グラフで表現した場合に，ループが存在することがある．たとえば，図 3.1 に示す DTD において，ノード item は子孫ノード book を持ち，ノード book は子ノード item を持っている．これをグラフで表現すると，図 3.2 に示すように，ノード item とノード book の間にループがある．しかし，DTD の木分割処理を行うためには，DTD をループを含まない木に変換することが必要である．そこで，後述の DTD を木に変換するアルゴリズムにおいては，もしあるノードの祖先ノード集合の中にそのノードのラベルと同じのノードがあれば，そのノードを削除する．DTD の木分割処理では，DTD を木に変換するアルゴリズムで作成された木に対して，各ノードに重みを付与する．*や + が付されたノードは XML データにおいて出現回数が多い可能性があるため，そのようなノードの重みは大きくする．逆に，?や | が付されたノードは XML データにおいて出現回数が少ない可能性があるため，そのようなノードの重みは小さくする．木を分割する際に，数値 k を設定し，分割されたそれぞれの部分木の重みが k 以下という条件の下で，分割数が最小となるように分割する．

DTD を木に変換するアルゴリズムを示す．まず，記号の意味を説明する．Σ をラベル集合とする．Σ 上の正規表現を次のように定義する．

- ϵ と $a \in \Sigma$ は正規表現である．

```
<!ELEMENT item (books,music)>
<!ELEMENT books (book)>
<!ELEMENT book (item,isbn)>
<!ELEMENT music (title, author)>
<!ELEMENT isbn (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
```

図 3.1　ループを含む DTD

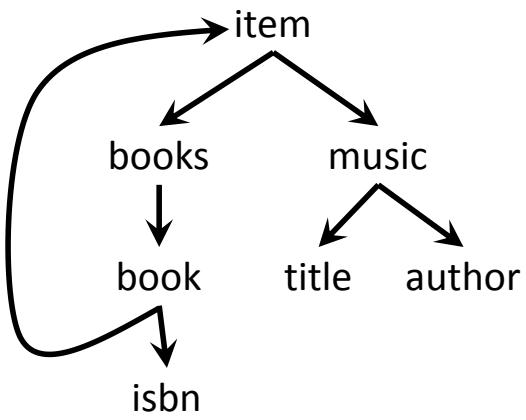

図 3.2　ループを含む DTD のグラフ表現

- $r_1, ..., r_n$ を正規表現とする $(n \geqq 1)$．このとき，，$r_1, ..., r_n$ および $r_1|...|r_n$ は正規表現である．
- r を正規表現とする．このとき，r^*，$r?$，および，$r+$ は正規表現である．

DTD を組 $D = (d, sl)$ と定義する．ここで，d は Σ から Σ 上の正規表現集合への写像，$sl \in \Sigma$ は文書要素である．ラベル $a \in \Sigma$ に対して，$d(a)$ を a の内容モデルという．たとえば，図 3.1 の DTD は $D = (d, item)$ と表すことができる．DTD の木構造を $T_D = (V_D, E_D)$ と表す．ここで，V_D はノードの集合，E_D は辺の集合である．ノード v の祖先ノードの集合を $A(v)$ と表す．

DTD を木に変換するアルゴリズムを以下に示す.

入力：$DTD\ D = (d, sl)$

出力：DTD の構造を表す木 T_D

1. **begin**
2. $V_D = \emptyset$; $E_D = \emptyset$
3. j=1
4. $l(v_1) = sl$; $V_D \leftarrow \{v_1\}$
5. $V \leftarrow \{v_1\}$
6. **while** $V \neq \emptyset$ **do**
7. **begin process node**
8. choose a node v_i from V
9. delete v_i from V
10. let $\{l_{j+1}, l_{j+2}, ..., l_{j+n}\}$ be the set of elements appearing in $d(l(v_i))$
11. let v_k be a new node with $l(v_k) = l_k$ $(j+1 \leqq k \leqq j+n)$
12. let $A(v_k)$ be a set of ancestor nodes of v_k, i.e., $A(v_k) = A(v_i) \cup \{v_i\}$ $(j+1 \leqq k \leqq j+n)$
13. $V_D \leftarrow V_D \cup \{v_{j+1},\ ..., v_{j+n}\}$
14. $E_D \leftarrow E_D \cup \{v_i \rightarrow v_{j+1}, ..., v_i \rightarrow v_{j+n}\}$
15. $V \leftarrow V \cup \{v_{j+1}, ..., v_{j+n}\}$
16. $j = j + n$
17. **if** $l(v_k) = l(v_m)$ for some $v_m \in A(v_k)$ **then**
18. delete v_k from V
19. **end if**
20. **end process node**
21. **return** $T_D = (V_D,\ E_D)$
22. **end algorithm**

次に，DTD の木分割処理のアルゴリズムを示す (文献 [3] の木分割アルゴリズムを用いる)．k は正整数を表す．ただし，k の値は最小のノードの重みより大きい値とする．分割した辺の集合を C，ノード v の子ノードの集合を $S(v)$，重みを $w(v)$ と表す．ノード v を部分木の根とした場合に，この部分木のすべてのノードの重みの和を $W^*(v)$ と表す.

DTD の木分割処理のアルゴリズムを以下に示す.

入力：DTD の構造を表す木 T_D，正整数 k

出力：辺集合 C

1. **begin**
2. $C = \emptyset$
3. assign $W^*(v) = w(v)$ to all leaf nodes in T_D;
4. **for** i=maximum-level-in-T **downto** 1 **do**

5. **begin process i_th level**;
6. **while** there is an unprocessed node v in level i **do**
7. **begin process node** v:
8. remove heaviest sons of v one by one from $S(v)$ until $W^*(v) = \Sigma_{u \in S(v)}\ W^*(u) + w(v) \leqq k$;
9. For every such son w removed, add the edge $v \rightarrow w$ to C
10. **end process node**
11. **end process level**
12. **end algorithm**

3.1.2　変換処理

変換処理では，上記アルゴリズムで得られた辺集合 C を用いる．XML データの根から変換処理を開始する．変換過程において，C に属する辺に遭遇する度に新たなスレッドを生成し，その辺の始点を根とする部分木を変換する．なお，各スレッドが XML データの必要な部分にアクセスする際，XML データは分割せずに，各スレッドが共通の XML データにアクセスして変換処理を行う．ここで，もし各スレッドで処理する XML データのサイズの差が大きければ，1 つのスレッドの処理時間が長くなってしまい，全体の処理時間も長くなってしまう．DTD の木を分割した後，各部分の重みの和の値が近いため，各部分に対応する XML データのサイズもほぼ同じであり，各スレッドの処理時間もほぼ同じであると期待される．このようにして，処理効率の向上を図っている．

変換処理のアルゴリズムに出現する記号の意味を説明する．XML データを T_x，そのノードの集合を V_x，辺の集合を E_x と表す．根が v の部分木を $T(v)$ と表す．辺集合 C は DTD の木分割処理のアルゴリズムの出力の辺集合 C である．

XML データ変換処理のアルゴリズムを以下に示す．

入力：XML データ $T_x = (V_x,\ E_x)$，木変換機 T_r，辺集合 C

出力：変換した木 $T' = (V',\ E')$

1. **begin**
2. $T' = \emptyset$
3. n=0
4. **for** i=1 to maximum-level-in-T **do**
5. **begin process i_th level**:
6. **while** there is an unprocessed node $v \in V_x$ in level i **do**
7. **begin process node** v:
8. let v_p be the parent node of v
9. **if** $v_p \rightarrow v \in C$ **then**
10. create a new thread and $T'(v) \leftarrow T_r(T(v))$ by the thread
11. **else**
12. let r be a rule in T_r applicable to v

13. apply r to v and add the obtained nodes to V' and the abtained edges to E'
14. **end if**
15. **end process node**
16. **end process level**
17. merge the trees $T'(v)$ obtained by threads into T'
18. **return** T'
19. **end algorithm**

3.2　提案手法の具体例

3.2.1　DTD に関する処理の具体例

例として，図 3.3 に示す $DTD\ D$ を考える．$A(g) = \{a, f\}$ であり，ノード g の子ノード a と f が $A(g)$ に属しているため，DTD を木に変換するとき，ループを生成しないために，g の子ノード a，f を削除する．同様に，ノード i の内容モデルの中に i を含むため，DTD を木に変換した後，ノード i の子ノードとしてノード i を含まないようにする．DTD を木に変換するアルゴリズムを用いると，図 3.3 の DTD は図 3.4 の木に変換することができる．ノードの重みについて，*や + が付されたノードの重みを 2，?や | が付されたノードの重みを 0.5 に設定している．

```
<!ELEMENT a(b,f)>
<!ELEMENT b(c+ ,(d|h) )>
<!ELEMENT c(e)>
<!ELEMENT d(e?)>
<!ELEMENT h(e)*>
<!ELEMENT f(g*,e)>
<!ELEMENT g(f|a)*>
<!ELEMENT e(i)>
<!ELEMENT i(#PCDATA|i|j|k)*>
<!ELEMENT j(#PCDATA)>
<!ELEMENT k(#PCDATA)>
```

図 3.3　DTD D

次に，DTD の木分割処理のアルゴリズムを用いて木を分割する．ここで，数値 k を 12 とし，変換した後の木（図 3.4）を入力とした場合，図 3.5 に示す 3 つの部分木が出力される．部分木 (1)，(2)，(3) それぞれの重みの和は 7.0，11.5，8.0 である．各部分木の重みの合計は k より小さい．

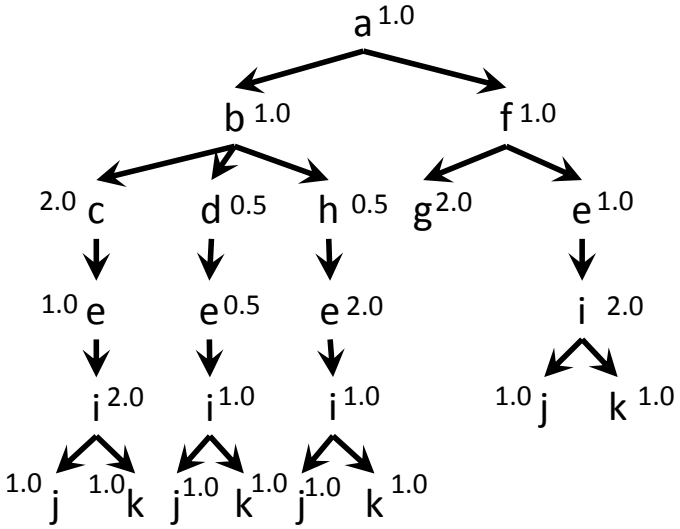

図 3.4　DTD から変換した木

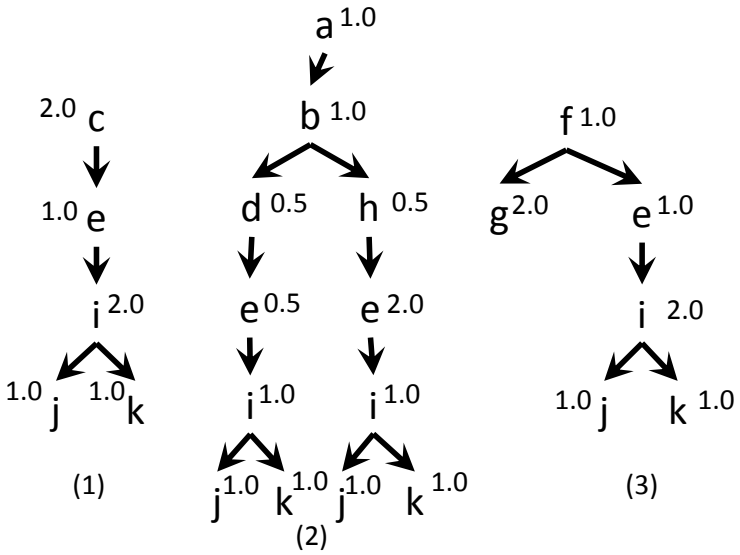

図 3.5　分割した DTD の木

3.2.2　変換処理の具体例

例として，図 3.6 の変換規則を考える．また，すべての要素の変換した後の mode 属性は p とする．初期状態は p とする．*DTD D*(図 3.3) に妥当な XML データを図 3.7 に示す．その XML データを木で表したものを図 3.8 に示す．この XML データを図 3.6 の交換規則で HTML データに変換する変換結果である．HTML データはウェブページで表現できるため，人間に対して，よりやすい表現形式といえる．

図 3.10 に，変換処理のイメージ図を示す．XML データを 5 個スレッドで変換していることが分かる．

```
(p,a)→html(p)
(p,b)→body(p)
(p,c)→h1(p)
(p,d)→h2(p)
(p,e)→span(p)
(p,f)→head(p)
(p,g)→title(p)
(p,h)→h3(p)
(p,i)→a(p)
(p,j)→b(p)
(p,k)→img(p)
```

図 3.6　木変換機の変換規則

```
1.  <?xml version="1.0" standalone="yes"?>
2.
3.  <a>
4.    <b>
5.      <c>
6.        <e>
7.          <i>
8.            <k></k>
9.            <k></k>
10.           <j></j>
11.         </i>
12.       </e>
13.     </c>
14.     <c>
15.       <e>
16.         <i>
17.           <k></k>
18.           <j></j>
19.         </i>
20.       </e>
21.     </c>
22.     <c>
23.       <e>
24.         <i>
25.           <k></k>
26.         </i>
27.       </e>
28.     </c>
29.     <h>
30.       <e>
31.         <i>
32.           <j></j>
33.           <j></j>
34.         </i>
35.       </e>
36.       <e>
37.         <i>
38.           <j></j>
39.         </i>
40.       </e>
41.     </h>
42.   </b>
43.   <f>
44.     <g></g>
45.     <g></g>
46.     <e>
47.       <i>
48.         <k></k>
49.         <j></j>
50.         <j></j>
51.         <j></j>
52.       </i>
53.     </e>
54.   </f>
55. </a>
```

図 3.7　XML データ

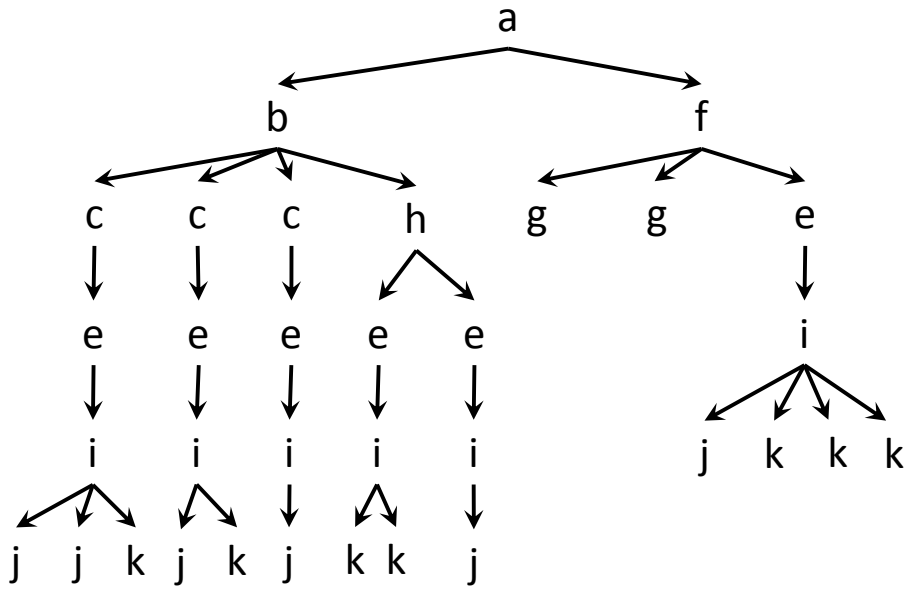

図 3.8　XML データの木

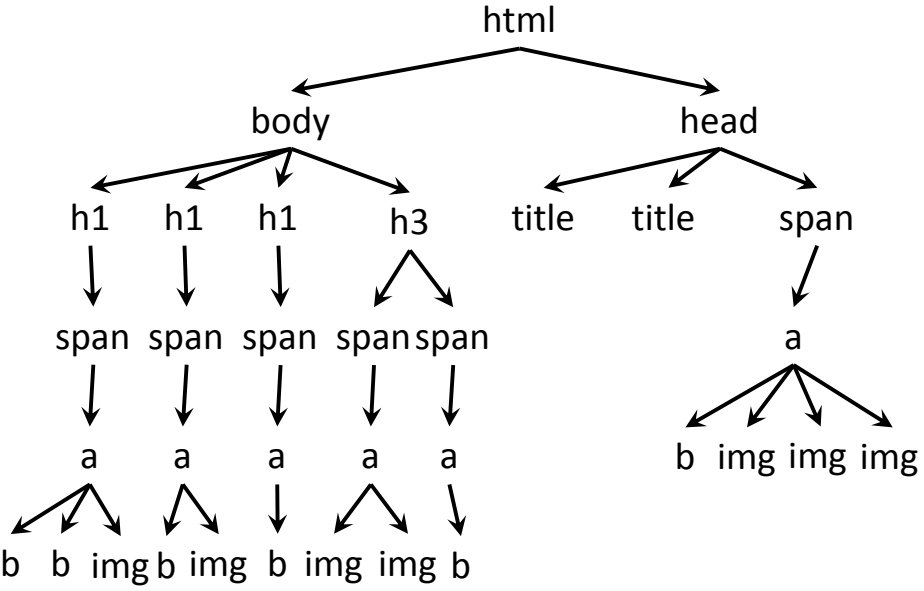

図 3.9　変換結果

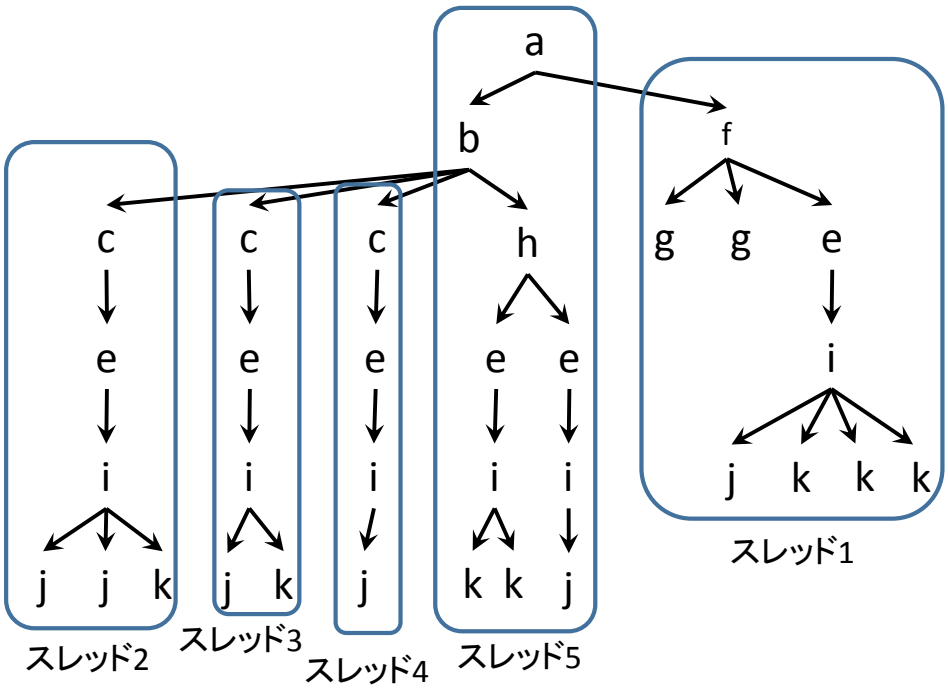

図 3.10　変換処理のイメージ図

第 4 章

評価実験

本章では，提案手法に対する評価実験について述べる．

4.1　評価実験の概要

第 3 章の木分割アルゴリズムを用いた XSLT 実行手法を Java を用いて実装した．DTD は XMark[5] と呼ばれるベンチマークソフトのものを用い (付録)，XML データも XMark を用いて作成した．このデータに適用する木変換機を作成した上で，提案手法での XSLT 実行時間と並列処理を行わない場合の XSLT 実行時間を比較し，提案手法の実行戦略が妥当かどうか，及び，木変換機の変換規則の数の違いによる実行時間の変化を評価した．

XMark は異なるサイズの XML データを生成できる．実験用 XML データとして，10MB，20MB，30MB，40MB，50MB，60MB の 6 つを用意した．また，木変換機は変換規則数が 10，30，50，71 のものを著者が作成した．

4.2　評価実験の結果

実験環境は以下の通りである．

CPU：Intel(R) Core(TM) i3-4030U CPU @ 1.90GHz
メモリ：4G
OS：Windows 7 Professional(64bit)
使用言語：Java 1.7.0_79

表 4.1〜 表 4.4 に 4 つの木変換機に対する評価実験の結果を示す．それらをグラフ化したものを図 4.1〜 図 4.4 に示す．DTD 分割数が 0 の行は，並列処理を行わない単一スレッドによる実行時間を表す．なお，実行時間の単位はすべて分である．

表 4.1　変換規則数が 10 個の場合の実行時間

	XML サイズ					
DTD 分割数	10M	20M	30M	40M	50M	60M
0	0.0075	0.0191	0.0434	0.070	0.116	0.171
4	0.0046	0.0114	0.0161	0.034	0.066	0.121
9	0.0026	0.0052	0.0081	0.011	0.026	0.054

表 4.2　変換規則数が 30 個の場合の実行時間

	XML サイズ					
DTD 分割数	10M	20M	30M	40M	50M	60M
0	0.370	1.511	3.517	6.303	9.868	14.291
4	0.046	0.134	0.367	0.642	1.143	1.361
9	0.013	0.064	0.135	0.183	0.262	0.771

表 4.3　変換規則数が 50 個の場合の実行時間

	XML サイズ					
DTD 分割数	10M	20M	30M	40M	50M	60M
0	0.780	3.203	7.146	12.981	20.606	35.902
4	0.194	0.410	0.730	3.594	8.099	18.274
9	0.031	0.090	0.312	1.470	1.929	14.943

表 4.4　変換規則数が 71 個の場合の実行時間

	XML サイズ					
DTD 分割数	10M	20M	30M	40M	50M	60M
0	1.204	4.844	10.756	15.237	34.055	52.098
4	0.353	0.721	1.191	6.410	13.429	39.762
9	0.126	0.338	0.377	1.515	4.281	21.476

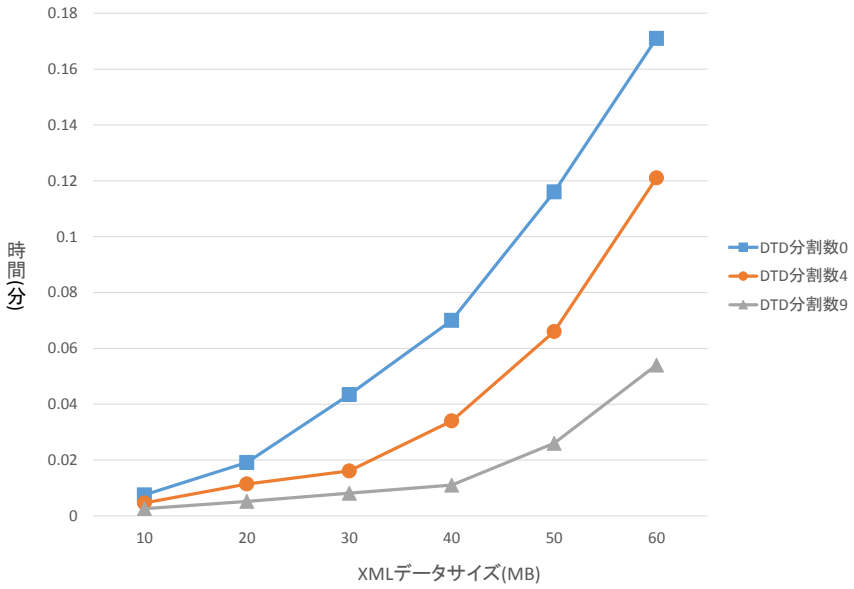

図 4.1　変換規則が 10 個の実行時間

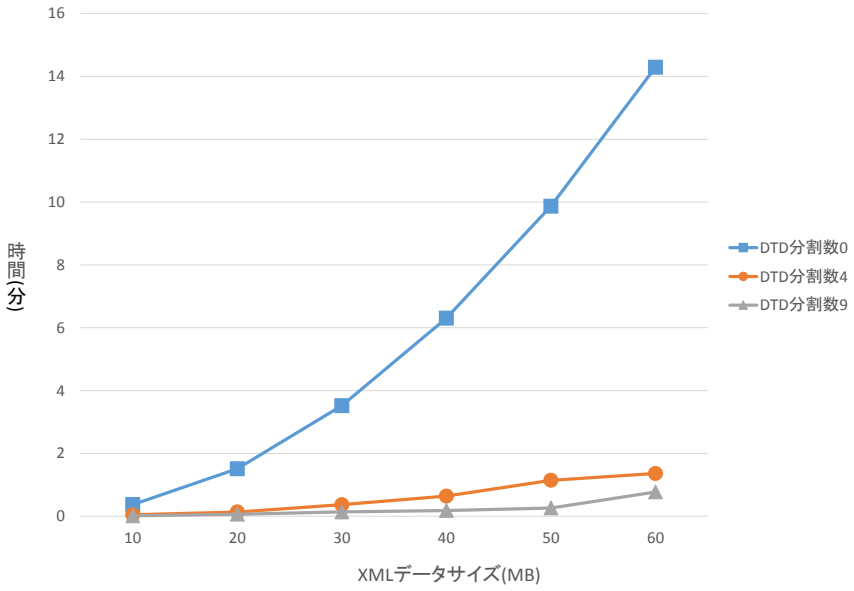

図 4.2　変換規則が 30 個の実行時間

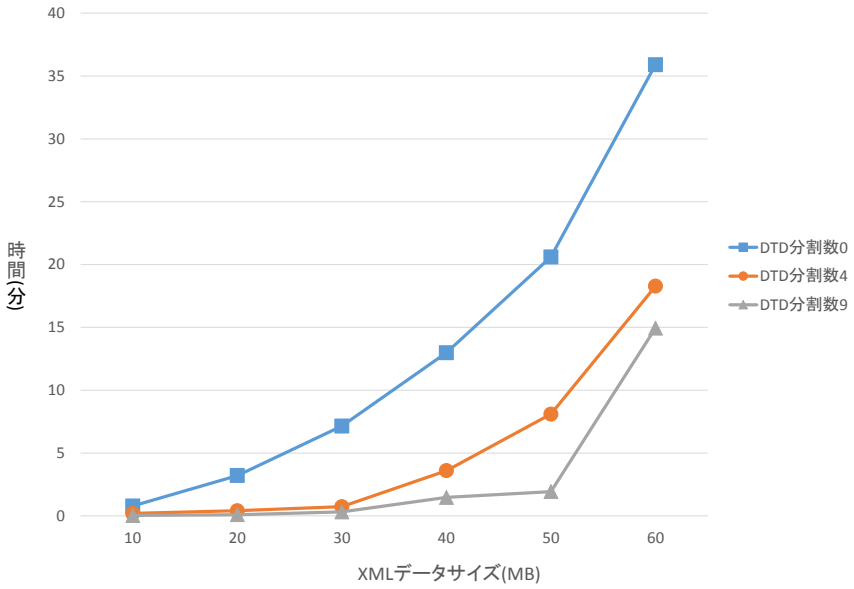

図 4.3　変換規則が 50 個の実行時間

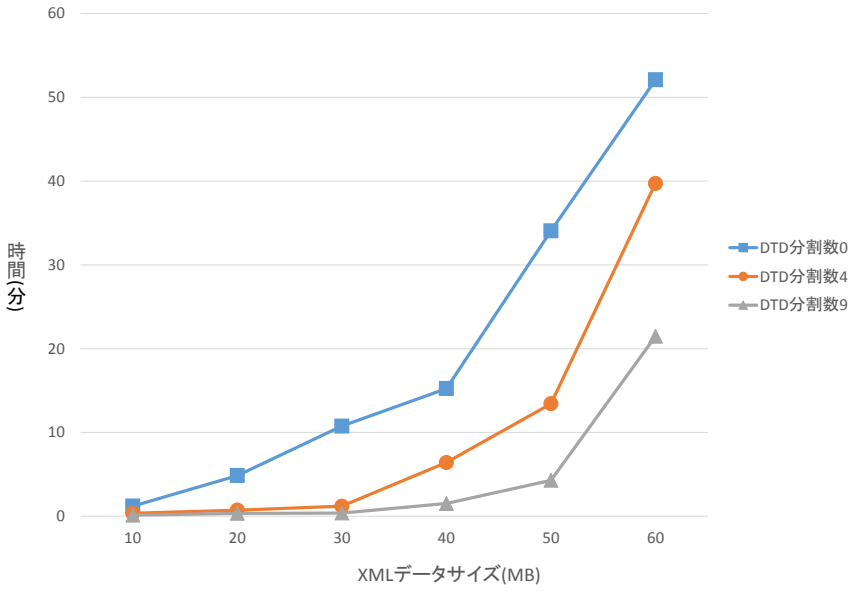

図 4.4　変換規則が 71 個の実行時間

表 4.5〜 表 4.7 に DTD 分割数が 0，DTD 分割数が 4 の実行時間と DTD 分割数が 9 の実行時間に示す．図 4.5〜 図 4.7 にそれらをグラフ化したものを示す．

表 4.5　DTD 分割数が 0 の場合の実行時間

	XML サイズ					
変換規則個数	10M	20M	30M	40M	50M	60M
10	0.0075	0.0191	0.0434	0.070	0.116	0.171
30	0.370	1.511	3.517	6.303	9.869	14.291
50	0.780	3.203	7.146	12.981	20.606	35.902
71	1.204	4.844	10.756	15.237	34.055	52.098

表 4.6　DTD 分割数が 4 の場合の実行時間

	XML サイズ					
変換規則個数	10M	20M	30M	40M	50M	60M
10	0.0046	0.0114	0.0161	0.034	0.066	0.121
30	0.046	0.134	0.367	0.642	1.143	1.361
50	0.194	0.410	0.730	3.594	8.099	18.274
71	0.353	0.721	1.191	6.410	13.429	39.762

表 4.7　DTD 分割数が 9 の場合の実行時間

	XML サイズ					
変換規則個数	10M	20M	30M	40M	50M	60M
10	0.0026	0.0052	0.0081	0.011	0.026	0.054
30	0.013	0.064	0.135	0.183	0.262	0.771
50	0.031	0.090	0.312	1.47	1.929	14.943
71	0.126	0.338	0.377	1.515	4.281	21.476

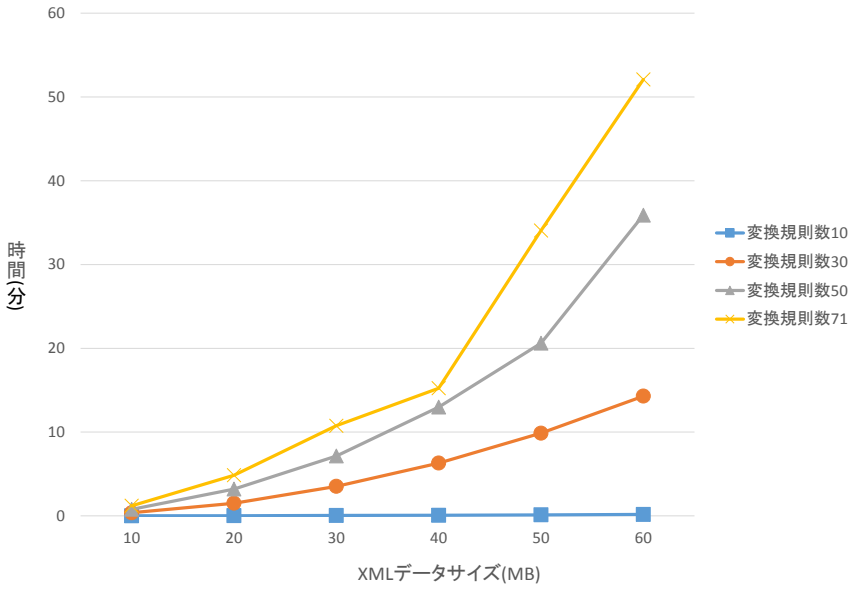

図 4.5　DTD 分割数が 0 の場合の実行時間

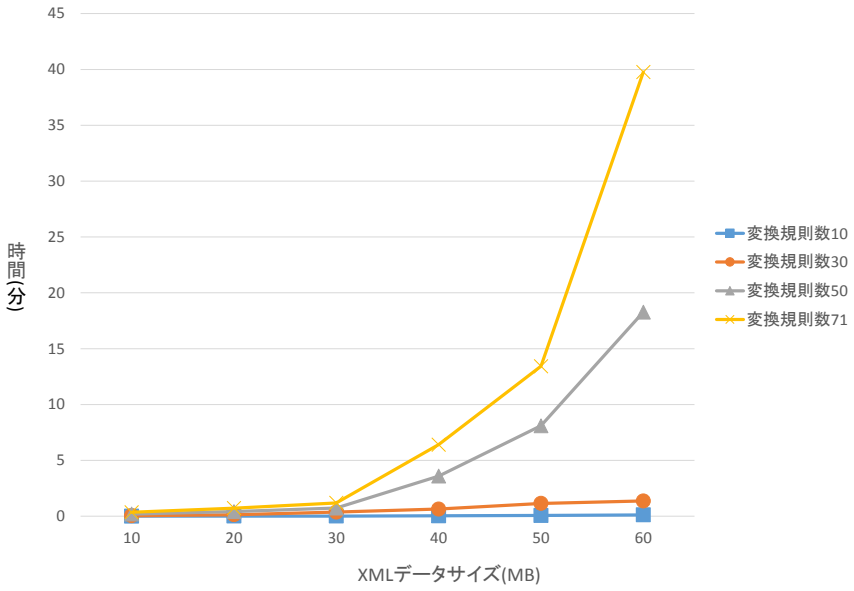

図 4.6　DTD 分割数が 4 の場合の実行時間

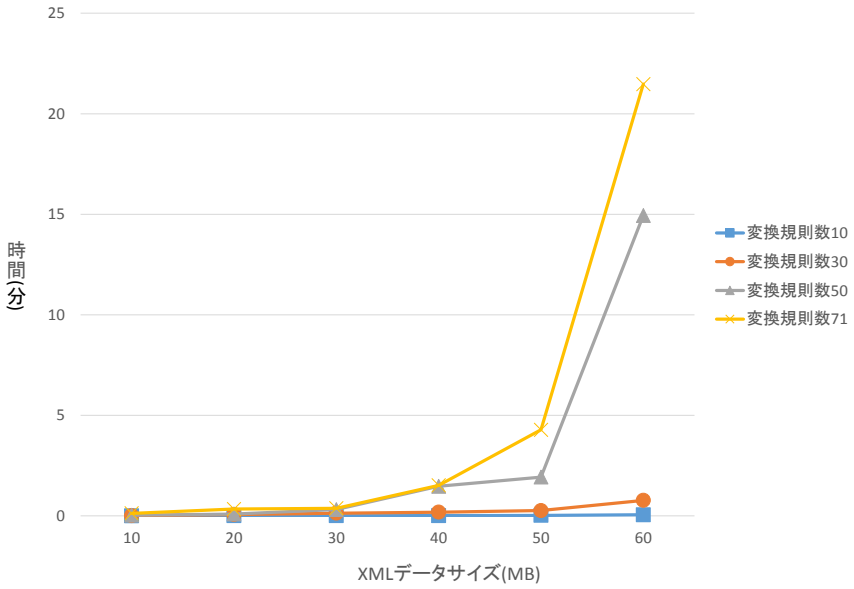

図 4.7　DTD 分割数が 9 の場合の実行時間

4.3　評価実験に関する考察

10MB の XML データに対する変換処理では，XML データのサイズが小さいため，ほとんどの変換処理において並列化の効果が見られなかった．20MB 以上の XML データについては，提案手法と並列処理を行わない場合での実行時間に差が生じており，提案手法の効果が確認できる．また，図 4.1～ 図 4.4 の「DTD 分割数 4」と「DTD 分割数 9」2 つ折れ線を見ると，DTD 分割数がより多ければ，すなわちスレッドがより多ければ，実行時間がより小さいことが分かる．変換規則が 30 個については，スレッドあり場合の実行時間を大きく短縮させることができたのに対して，他の異なる変換規則個数ではそのような著しい時間の差を見ることができなかった．このことから，全体的には並列化の効果も認められるが，変換規則個数によって並列化の効果に差がいじることが分かる．

図 4.5～ 図 4.7 から見ると，並列化の有無にかかわらず，変換規則の数が多いほど，実行時間が長くなる．また，DTD 分割数が大きくなり，XML データのサイズが増加するに従って，変換規則数による実行時間の時間差が大きくなることが分かる．原因として，スレッドごとに変換する際に，変換に必要な情報を得ることと各スレッドを一つの木にマージすることが多い時間がかかることが考えられる．

第 5 章

むすび

本研究では，木分割アルゴリズムを用い，XSLT のサブセットである木変換機の変換処理を並列実行する手法を提案した．提案手法により処理時間を短縮することができたが，変換規則個数によって並列化の効果に差が生じた．本研究は下降型木変換機に限定したため，今後の課題として，より能力の高い木変換機への対応などが挙げられる．また，今回評価実験に使用した XML データは XMark で生成し，変換規則も自分で書いたものであるため，実際に使用された XML データや変換規則に即したものであるとは限らない．より多く，大きいな XML データを使用した評価実験や，DTD 以外のスキーマ言語への対応なども今後の課題となる．

謝辞

本研究を進めるに当たり，適切なご指導・御助言をしていただきました，指導教員の鈴木伸崇准教授，副指導教員阪口哲男准教授に心から感謝致します．

また，様々な御助言をしていただきました鈴木伸崇研究室の皆さま，本当にありがとうございました．特に日頃のゼミ等での多くの適切な御助言などの御協力をしてくださった院生の呉さん，王さん，関根さん，長尾さん，ゼミ生の皆さん本当にありがとうございました．

参考文献

[1] F. Zavoral and J. Dvorakovam , “Perfomance of XSLT Processors on Large Data Sets”, Applications of Digital Information and Web Technologies, 2009. ICADIWT ’09. Second Inter-national Conference, no.10905932, pp.110-115, London, UK, Oct. 2009.

[2] H. Mizumoto and N. Suzuki, “An XSLT Transformation Method for Distributed XML”, Proceedings of the 4th International Conference on innovative Computing Communication Technology (INTECH 2014), 10p., 2014.

[3] Kundu, Sukhamay, and Jayadev Misra. “A linear tree partitioning algorithm”.SIAM Journal on Computing 6.1 (1977): 151-154.

[4] R. Bordawekar, L.Lim, A. Kementsietsidis and B.W. Kok “Statistics-based Parallelization of XPath Queries in Shared Memory Systems”, The University of Hawaii, http://www2.hawaii.edu/ lipyeow/pub/edbt10-xpathmulticore.pdf, January 2015.

[5] Schmidt A, Waas F, Kersten M, Carey M.J, Manolescu I, and Busse R, “XMark: A benchmark for XML data managemet” VLDB ’02 Proceedings of the 28th international conferenceon Very Large Data Bases. Hong Kong, China, 2002–08–20/08-23. VLDB, 2002, p.974–985.

[6] Wim Martens,Frank Neven, “Typechecking Top-Down Uniform Unranked Tree Transducers”. Lecture Notes in Computer Science, 2002, Volume 2572/2002, p.64-78.

[7] World Wide Web Consortium (W3C). “Extensible Markup Language (XML) 1.0 (Fifth Edition)”, (online), available from 〈 https://www.w3.org/TR/REC-xml/〉, (accessed 2016-05-28).

[8] World Wide Web Consortium (W3C). “XSL Transformations (XSLT) Version 2.0”. (online), available from 〈 http://www.w3.org/TR/xslt20/〉, (accessed 2016-05-30).

[9] Y. Wu and N. Suzuki, “An Algorithm for Detecting XSLT Rules Affected by Schema Updates”, 情報処理学会研究報告, 2015-DBS-161(6), 6p., 2015.

付録

評価実験用 DTD

```
1. <! −− DTD for auction database −−>
2.
3. <! −− $ Id: auction.dtd,v 1.15 2001/01/29 21:42:35 albrecht Exp $ −−>
4.
5. <!ELEMENT site (regions, categories, catgraph, people, open_auctions, closed_auctions)>
6. <!ELEMENT categories (category+)>
7. <!ELEMENT category (name, description)>
8. <!ATTLIST category id ID #REQUIRED>
9.
10. <!ELEMENT name (#PCDATA)>
11. <!ELEMENT description (text | parlist)>
12. <!ELEMENT text (#PCDATA | bold | keyword | emph)*>
13. <!ELEMENT bold (#PCDATA)>
14. <!ELEMENT keyword (#PCDATA)>
15. <!ELEMENT emph (#PCDATA)>
16. <!ELEMENT parlist (listitem)*>
17. <!ELEMENT listitem (text | parlist)*>
18. <!ELEMENT catgraph (edge*)>
19. <!ELEMENT edge EMPTY>
20. <!ATTLIST edge from IDREF #REQUIRED to IDREF #REQUIRED>
21.
22. <!ELEMENT regions (africa, asia, australia, europe, namerica, samerica)>
23. <!ELEMENT africa (item*)>
24. <!ELEMENT asia (item*)>
25. <!ELEMENT australia (item*)>
26. <!ELEMENT namerica (item*)>
27. <!ELEMENT samerica (item*)>
```

28. <!ELEMENT europe (item*)>
29. <!ELEMENT item (location, quantity, name, payment, description, shipping, incategory+, mailbox)>
30. <!ATTLIST item id ID #REQUIRED featured CDATA #IMPLIED>
31.
32. <!ELEMENT location (#PCDATA)>
33. <!ELEMENT quantity (#PCDATA)>
34. <!ELEMENT payment (#PCDATA)>
35. <!ELEMENT shipping (#PCDATA)>
36. <!ELEMENT reserve (#PCDATA)>
37. <!ELEMENT incategory EMPTY>
38. <!ATTLIST incategory category IDREF #REQUIRED>
39.
40. <!ELEMENT mailbox (mail*)>
41. <!ELEMENT mail (from, to, date, text)>
42. <!ELEMENT from (#PCDATA)>
43. <!ELEMENT to (#PCDATA)>
44. <!ELEMENT date (#PCDATA)>
45. <!ELEMENT itemref EMPTY>
46. <!ATTLIST itemref item IDREF #REQUIRED>
47.
48. <!ELEMENT personref EMPTY>
49. <!ATTLIST personref person IDREF #REQUIRED>
50.
51. <!ELEMENT people (person*)>
52. <!ELEMENT person (name, emailaddress, phone?, address?, homepage?, creditcard?, profile?, watches?)>
53. <!ATTLIST person id ID #REQUIRED>
54.
55. <!ELEMENT emailaddress (#PCDATA)>
56. <!ELEMENT phone (#PCDATA)>
57. <!ELEMENT address (street, city, country, province?, zipcode)>
58. <!ELEMENT street (#PCDATA)>
59. <!ELEMENT city (#PCDATA)>
60. <!ELEMENT province (#PCDATA)>
61. <!ELEMENT zipcode (#PCDATA)>
62. <!ELEMENT country (#PCDATA)>
63. <!ELEMENT homepage (#PCDATA)>

64. <!ELEMENT creditcard (#PCDATA)>
65. <!ELEMENT profile (interest*, education?, gender?, business, age?)>
66. <!ATTLIST profile income CDATA #IMPLIED>
67.
68. <!ELEMENT interest EMPTY>
69. <!ATTLIST interest category IDREF #REQUIRED>
70.
71. <!ELEMENT education (#PCDATA)>
72. <!ELEMENT income (#PCDATA)>
73. <!ELEMENT gender (#PCDATA)>
74. <!ELEMENT business (#PCDATA)>
75. <!ELEMENT age (#PCDATA)>
76. <!ELEMENT watches (watch*)>
77. <!ELEMENT watch EMPTY>
78. <!ATTLIST watch open_auction IDREF #REQUIRED>
79.
80. <!ELEMENT open_auctions (open_auction*)>
81. <!ELEMENT open_auction (initial, reserve?, bidder*, current, privacy?, itemref, seller, annotation, quantity, type, interval)>
82. <!ATTLIST open_auction id ID #REQUIRED>
83.
84. <!ELEMENT privacy (#PCDATA)>
85. <!ELEMENT initial (#PCDATA)>
86. <!ELEMENT bidder (date, time, personref, increase)>
87. <!ELEMENT seller EMPTY>
88. <!ATTLIST seller person IDREF #REQUIRED>
89.
90. <!ELEMENT current (#PCDATA)>
91. <!ELEMENT increase (#PCDATA)>
92. <!ELEMENT type (#PCDATA)>
93. <!ELEMENT interval (start, end)>
94. <!ELEMENT start (#PCDATA)>
95. <!ELEMENT end (#PCDATA)>
96. <!ELEMENT time (#PCDATA)>
97. <!ELEMENT status (#PCDATA)>
98. <!ELEMENT amount (#PCDATA)>
99. <!ELEMENT closed_auctions (closed_auction*)>
100. <!ELEMENT closed_auction (seller, buyer, itemref, price, date, quantity, type, annotation?)>

```
101. <!ELEMENT buyer EMPTY>
102. <!ATTLIST buyer person IDREF #REQUIRED>
103.
104. <!ELEMENT price (#PCDATA)>
105. <!ELEMENT annotation (author, description?, happiness)>
106. <!ELEMENT author EMPTY>
107. <!ATTLIST author person IDREF #REQUIRED>
108.
109. <!ELEMENT happiness (#PCDATA)>
```