圧縮された半構造化文書からの頻出木パターン発見

Finding frequent tree patterns from compressed semi-structured documents

村上征嗣 Seiji Murakami　土井晃一郎 Koichiro Doi　山本章博 Akihiro Yamamoto

京都大学 情報学研究科
Graduate School of Informatics, Kyoto University

In this paper we present a new method for finding frequent patterns from semi-structured data, where a frequent tree pattern means a subgraph which frequently occurs in a given semi-structured data. We make use of a tree grammar based data compression method called TGCA for semi-structured data. We made experiments of our proposed method with some semi-structured data and show efficiency of our method.

1. はじめに

本論文では，半構造化文書をはじめとする木構造データに対する木文法圧縮を用いて木構造データからの頻出木パターンの発見をより効率化する手法を提案する．頻出木パターンの発見は大規模な木構造データから有用な情報を効率的に抽出することを目的としており，木構造データとしても表現できる HTML/XML 文書の普及とともに既に多くのアルゴリズムの研究が行われている．また，データ圧縮も大規模データに対して需要の高い技術であり，両者を組み合わせることは大規模な木構造データの情報活用のために意義のあることと考えられる．

多くの既存の発見アルゴリズムでは頻出木パターン候補の生成に対する工夫が行われてきた．本研究の提案手法はそれらの手法とは異なる新たなアプローチによる効率化の手法であり，既存の発見アルゴリズムの多くと競合することなく組み合わせることも可能である．さらに，そのような組み合わせによって既存のアルゴリズムの長所を生かしつつ相乗的な計算効率の改善も期待できる．

本研究では根付き順序木と根付き無順序木に対して手法を提案しているが，本稿ではページ制限の都合上，根付き無順序木に対する木文法による圧縮アルゴリズムと，それを利用した頻出木パターンアルゴリズムの高速化の結果を載せる．根付き順序木に対する成果に関しては [Murakami 08] を参照されたし．

2. 準備

根付き無順序木 (rooted unordered tree)U は $U = (V, \mathcal{L}, v_0, E, label)$ により定義される．$V, \mathcal{L}$ は互いに異なる要素からなる有限集合，$v_0 \in V$，$E \subseteq V^2$ であり，$label$ は $V \to \mathcal{L}$ の関数である．V の各要素は T の節点または**ノード** (node) といい，特に $v_0 \in V$ を T の**根** (root) という．E の各要素は有向枝 (directed edge) あるいは単に**枝** (edge) という．$label$ はラベル関数 (labeling function) といい，各ノードを $\mathcal{L}$ の要素と対応付けている．$v \in V, l \in \mathcal{L}$ について $label(v) = l$ であるとき，l をノード v の**ラベル** (label) という．根 v_0 以外のノード v にはそれぞれ $(v', v) \in E$ であるノード v' が必ずただ 1 つ存在し，v' を v の**親** (parent)，v を v' の**子** (child) と呼び，$v' = \pi(v)$ で表記する．

根付き順序木とは根付き無順序木にノードの兄弟間の順序関係をつけたものである．以下，根付き無順序木，根付き順序木を単に順序木，無順序木と呼ぶ．

データ木 D，**パターン木** T を任意の無順序木であるとする．

定義 1 パターン木 T のノードの全集合 $\{t_1, ..., t_n\}$ に対し，無順序データ木 D のノードのある部分集合 $\{d_{k_1}, ..., d_{k_n}\}$ が存在して，

1. すべての $1 \le i \le n$ に対し，$label(t_i) = label(d_{k_i})$.
2. すべての $1 \le i \le n$，$1 \le j \le n$ に対し，$t_i = \pi(t_j)$ ならば $d_{k_i} = \pi(d_{k_j})$.

をすべて満たすとき，T は D に**出現している**という．また，d_{k_i} を t_i の (もしくは，$\{d_{k_1}, ..., d_{k_n}\}$ を T の) D における**出現**という．

T の根を $Root(T)$ で表すとする．$Root(T)$ の D における出現を特に**ルート出現** (root occurrence) と呼び，$Occ_D(Root(T))$ で表すとする．また，その数を $|Occ_D(Root(T))|$ により表すとする．そして，T の D における**出現回数**を $|Occ_D(Root(T))|$ と定め，さらに T の D における**出現頻度** (occurrence frequency) を $|Occ_D(Root(T))|/|D|$ と定め，$freq_D(T)$ と表記する．また，$0 < \sigma \le 1$ なる**最小サポート** (minimum support) と呼ばれる値 σ に対して $freq_D(T) \ge \sigma$ であるとき，T は D において σ **頻出** (σ-frequent) であるという．

無順序木における頻出木パターン発見問題

> **無順序データ木 D と最小サポート σ の入力に対し，σ 頻出であるような正規パターン木 T をすべて発見せよ．**

なお，無順序木で効率的にパターン発見を行うには，同じ頻出木パターン候補を生成しないようにするために正規形という概念が重要になるが，我々はその点とは直接関わらないパターン木のデータ木での出現チェックに対する高速化を行っているので省略する．正規形については s[Asai 03] 等を参照されたし．

3. 提案手法

本手法は木文法圧縮が施された木構造データに対して直接的に頻出木パターンの発見を行うことを可能にし，さらに無圧

連絡先: 土井晃一郎，京都大学 情報学研究科 知能情報学専攻，〒 606-8501 京都府京都市左京区吉田本町，Tel: 075-753-5996，Fax: 075-753-5628，E-mail: doi@i.kyoto-u.ac.jp

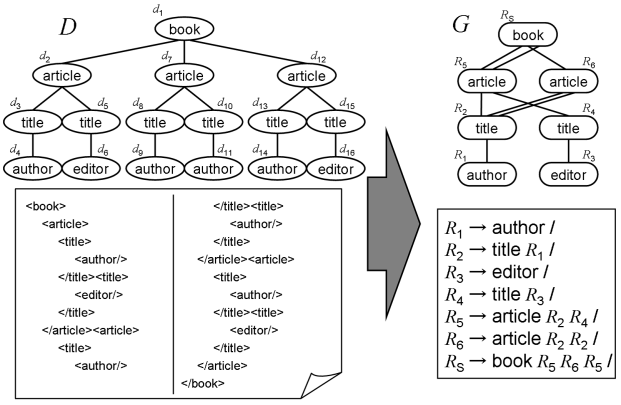

図 1: TGCA による圧縮の例

縮の木構造データを入力とする従来法からの効率化も可能としている．

3.1 木構造データの木文法圧縮

木構造データに対する木文法圧縮のアルゴリズムとして**TGCA**(Tree Grammar Compression Algorithm)[小沼 06]を提案している．TGCA は半構造化文書に対して木文法による圧縮を行うアルゴリズムであるが，本研究ではこれを圧縮アルゴリズムとして用いる．なお，本稿では TGCA による圧縮の性質のうち提案手法の説明に必要な部分にだけ触れるものとする．

3.1.1 木文法圧縮アルゴリズム TGCA による圧縮

木文法 (tree grammar) とは木を生成する文法のことであり，ある木文法がある特定の木を生成するとき，その木文法はその特定の木の表現の 1 つであると捉えることができる．TGCA は，半構造化文書が表す木構造データに対してそのような木文法による表現を生成する．

TGCA が生成する木文法 G は**規則** (rule) の集合からなる．G の要素であるそれぞれの規則は，$R \to L\ \Psi$ / という形をしている．R は非終端記号，$L \in \mathcal{L}$ は元の木構造データのいずれかのノードの開始タグに相当する終端記号，Ψ は R を除く非終端記号の長さ 0 以上の有限列，'/' は L の終了タグに相当する終端記号である．

TGCA は，半構造化文書のタグ列を先頭から順番に読み取り，スタックと規則リストを用いた動作によって木文法 G を生成する．G の各規則はそれぞれが木構造データの中の部分木のいずれかに対応しているが，複数回繰り返し出現する同じ部分木は 1 つの規則によって置き換えられるので，それにより圧縮が実現される．また，それぞれの規則はすべて左辺に異なる非終端記号を持っている．つまり，TGCA による圧縮はそれぞれの部分木に名前を与えることと等価である．それぞれの規則は左辺の非終端記号により一意に特定できるので，以降は左辺の非終端記号を規則およびそれに対応する部分木の識別子として用いる．例えば，非終端記号 R_i を左辺に持つ規則を規則 R_i と呼ぶ．また，規則 R_i の右辺の L を $label_G(R_i)$ で，Ψ を Ψ_i で表す．

図 1 に TGCA による圧縮の例を示す．TGCA は，図 1 の左側のタグ列を受け取ると，右側の文法を $R_1, ..., R_S$ の順で生成する．非終端記号 R_j が規則 R_i の右辺に現れているとき，規則 R_j は規則 R_i の**子規則**であるという．規則 R_i の子規則の非終端記号の列 Ψ_i の長さを $width(R_i)$ で表し，Ψ_i に

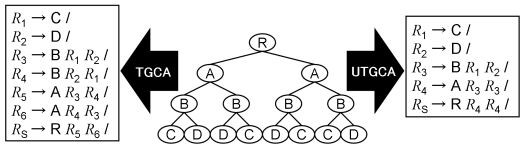

図 2: TGCA と UTGCA との違いの例

含まれる k 番目の非終端記号が R_j であるとき $R_j = R_i.k$ と表す．例えば，図 1 では $width(R_S) = 3$，$R_S.2 = R_6$ である．また，1 つの非終端記号が他の規則の Ψ に複数回繰り返し出現する場合もある．R_j が R_i の右辺 Ψ_i に n 回出現しているとき，$|(R_i, R_j)| = n$ と書く．$|(R_i, R_j)| = 0$ ならば，R_j は R_i の子規則ではないことを意味する．例えば，図 1 では $|(R_6, R_2)| = 2$ となる．さらに，G の大きさ $|G|$ を G の規則の数と定め，D から G への TGCA の**圧縮率**を $|G|/|D|$ と定める．図 1 では，圧縮率は 7/16 である．そして，最後に生成された規則 R_S を**開始規則**と呼ぶ．圧縮前の左側のタグ列は，開始規則 R_S に対して右辺の非終端記号がすべて置き換えられるまで他の規則を適用することで得られる．この操作が，**展開** (expansion) にあたる．ここで，規則 R_i を適用するとは，R_i を子規則とする規則に対してその右辺に出現する非終端記号 R_i を規則 R_i の右辺に書き換えることである．

3.1.2 無順序木の圧縮

与えられたデータ木が無順序データ木である場合は兄弟である各ノードの順番を任意に入れ替えても差し支えがない．そこで，その非終端記号に対応する規則が生成された順を非終端記号間の全順序関係として用いて各規則の右辺の非終端記号の列を整列させる [小沼 06] にはない動作を TGCA に新たに追加する手法を提案する．以降はこの手法を **UTGCA** と呼ぶことにする．例えば，図 1 における圧縮後の規則の生成順は $R_1, ..., R_6, R_S$ であるので，UTGCA では R_S の右辺は list R_5 R_5 R_6 / のように並べ替えられる．こうして整列を行うと，例えば図 2 のように兄弟順が異なるだけで無順序木としては互いに同値である部分木があった場合はそれらが並び替えによってすべて同じ規則に置き換えられることになるので，圧縮率を向上させることができる．なお，現実に利用されている XML データの多くに共通する傾向として開始規則の右辺は他の規則に比べて圧倒的に長くなることが多く，また開始規則の右辺を整列させても圧縮率の向上にはまったく貢献しないので，本研究で実装した UTGCA は開始規則の右辺に対しては整列の操作を省略している．

TGCA の計算量は，与えられたデータ木の大きさ $|D|$ に対し $O(|D|)$ である．しかし，UTGCA では兄弟順の並べ替えのためのコストが加わり，計算量は $O(|D| \log b)$ となる．ここで，b は D における最大分岐数である．すなわち，UTGCA では計算量は大きくなる．

また，UTGCA により圧縮された木構造データに無順序頻出木パターン発見アルゴリズムを適用することを考えると，データ木の圧縮アルゴリズム UTGCA においては規則の生成順，頻出木パターン発見アルゴリズムで生成されるパターン木においては辞書順により兄弟間の順序関係が統一されているという違いが生じている．

頻出木パターンの発見において，UTGCA の兄弟順を辞書順とすることで圧縮後のデータ木を正規形にしたとしても，正規形のデータ木における正規パターン木の出現は図 3 のように必ずしも互いに前順に従うわけではない．つまり，無順序デー

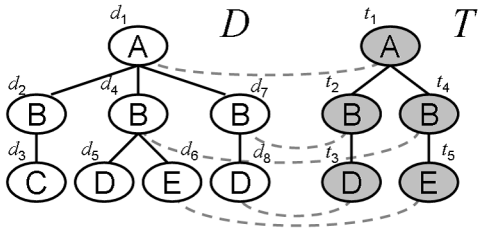

図 3: パターン木とデータ木がともに正規形の場合の例

タ木を正規形にしたとしても，パターン木が最右拡張された際の新しい最右葉の出現を最右枝上のノードの出現より弟に限定することはできず，結果的には発見の効率に大きな改善が望めるわけではない．なお，既存手法の [Asai 03][Nijssen 03] もデータ木の兄弟順により計算効率の変化が生じるような振舞いをしないアルゴリズムである．

また，辞書順による比較は生成順による比較と比べると手間がかかるため，UTGCA の兄弟間の順序関係には規則の生成順を採用している．

3.2 TG-DAG とその適用

TGCA により圧縮された木構造データは，図 1 の右上のように DAG 構造により表現できる．この TGCA の DAG の構造を $\mathcal{D}$ とおき，木構造データの TGCA 圧縮 G により表される構造 $\mathcal{D}$ を木構造に対して **TG-DAG** と名付ける．根付き木では根以外のすべてのノードはただ 1 つの親を持つが，TG-DAG は直観的には根付き木に対して根以外のすべてのノードが 1 つ以上の親を持つ，という定義の変更が施されたグラフである．TGCA 圧縮により元のタグの列は木文法に圧縮されるが，このときにそのタグ列の表す木構造は TG-DAG に圧縮されるといえる．

3.2.1 TG-DAG の頻出木パターン発見への適用

G の各規則 R_i を展開した部分木が元のデータ木において部分木として何回出現しているかは，規則をたどっていくことにより展開せずに計算することができる．これを利用して，頻出木パターン発見の計算を行う．

頻出木パターンの発見においてデータ木は新たな頻出木パターン候補を生成した後の出現チェックのときに使われる．このとき，代表的な頻出木パターンアルゴリズムである FREQT[Asai 02] では，データ木はノードの走査のために (1) そのノードの親をたどる，(2) そのノードの兄弟順で一番最初の子をたどる，(3) そのノードの 1 つ下の弟をたどる，の 3 つの操作が行われる．よって，頻出パターンの発見を $\mathcal{D}$ から行うには $\mathcal{D}$ に対してもこの操作が可能である必要がある．$\mathcal{D}$ では (2) は R_i の最初の子は $R_i.1$ によりたどることができるが，(1) と (3) の操作は木構造と同じようにはいかない．例えば，図 1 の R_2 は R_5 と R_6 から枝が張られており，木構造と違って親を一意に定めることができない．また弟をたどる操作についても，R_j が R_i と $R_{i'}$ の両方の子規則である場合は Ψ_i と $\Psi_{i'}$ の内容が異なれば R_j の親が R_i なのか $R_{i'}$ なのかによって R_j の兄弟も異なってしまう．すなわち，(3) を可能にするためにはその規則の親を指定する必要がある．

TGCA 圧縮においては，圧縮前の木で複数回出現する同じ部分木が 1 つの規則にまとめられている．そのため，前述のような拡張を加えて TG-DAG を用いて出現のチェックを行うと，圧縮前の木では複数回出現する同じ部分木に対して繰り返し行わなければならなかった計算が 1 つの規則に対して一括して行うことが可能になり，出現リストの保持する出現の数を圧縮して減らすことができる．列挙された頻出パターン候補の出現リストの長さの合計は頻出木パターン発見アルゴリズムの計算効率に大きく影響するが，TG-DAG を用いることでその値も軽減されてより効率のよい頻出木パターンの発見が期待できる．

表 1: TGCA および UTGCA の比較

データ名	TGCA	圧縮時間 規則数	UTGCA	圧縮時間 規則数
dblp.xml		12.205 s		20.441 s
(3,332,130 nodes)		4,700 rules		2,484 rules
nasa.xml		1.836 s		3.213 s
(476,646 nodes)		8,738 rules		6,014 rules
swissprot.xml		12.481 s		21.173 s
(2,977,031 nodes)		58,610 rules		58,002 rules
treebank.xml		14.805 s		24.914 s
(2,437,666 nodes)		471,312 rules		466,236 rules
wsu.xml		0.196 s		0.385 s
(74,557 nodes)		20 rules		20 rules

4. 実験結果

本章では，圧縮された木構造データから頻出木パターンの発見を行う提案手法と無圧縮の木構造データから発見を行う従来手法の比較のための実験を行う．また，それに先立ち 4.1 節では本研究で無順序木に対して導入した UTGCA の性能を TGCA と比較する実験も行う．

なお，実験に用いた木構造データはすべて Washington University XML Data Repository [*1] よりダウンロードした．また，実験の環境はすべて CPU:Xeon5150*2，主記憶のサイズは 2GB である．

4.1 無順序木の圧縮

まず，本研究で TGCA を拡張して新たに作成した UTGCA の性能を TGCA と比較するための実験を行った．この実験の結果を示したのが表 1 である．

まず，表 1 を見るとすべての実験結果において UTGCA の圧縮時間は TGCA に比べておよそ 2 倍弱かかっている．一方で，圧縮後の規則数は dblp.xml では UTGCA が TGCA による圧縮の場合と比べて半分近くまで減らしているものの，treebank.xml と swissprot.xml では規則数を TGCA より減らすことはあまりできず，UTGCA の効果はほとんど見られなかった．さらに wsu.xml においては，TGCA による圧縮率が非常に良いということもあるが圧縮後の規則数はまったく変わっていない．以上より，UTGCA は TGCA と比べると圧縮時間はおよそ 2 倍弱かかるが，圧縮率の向上が見られるかどうかは XML データの “個性” に依存する，といえる．

4.2 頻出木パターン発見

無圧縮と TG-DAG，UTG-DAG の三者で比較を行う．TG-DAG に対する頻出木パターン発見アルゴリズムと UTG-DAG に対するそれはどちらも同じアルゴリズムであり，得られる頻出パターンも同一となる．頻出木パターン発見アルゴリズムは [Murakami 08] に正規形であるかのチェックを加えることにより実装を行った．詳しい実装については省略する．

*1 http://www.cs.washington.edu/research/xmldatasets

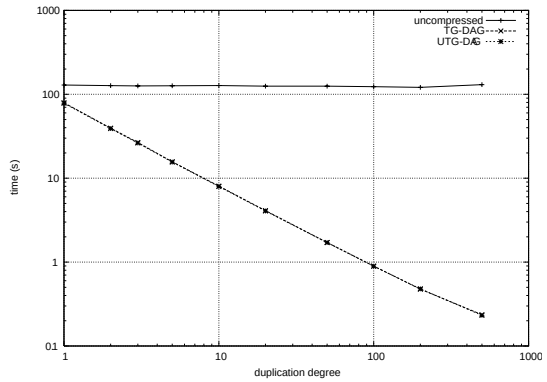

図 4: 様々な duplication degree における比較

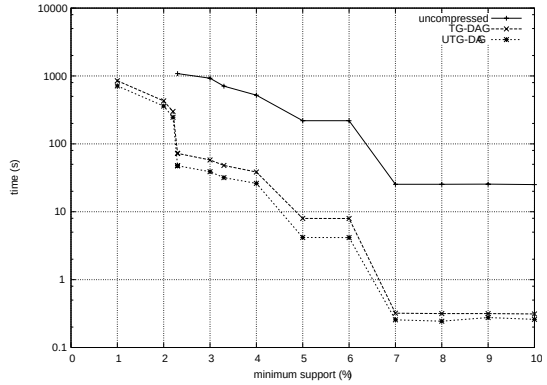

図 5: 様々な最小サポートにおける比較

最初に，圧縮率と両アルゴリズムの計算時間に関する実験を行った．用いた XML データは treebank.xml である．この実験では，圧縮率を人為的に調整するために，duplication degree という値を導入する．本実験で用いる treebank.xml では根が 56384 個の子ノード $v_1, ..., v_{56384}$ を持つが，duplication degree が n のとき，$0 \leq m \leq 56384/n$ に対して $subtree(v_{nm+1}), ..., subtree(v_{nm+n})$ をすべて $subtree(v_{nm+1})$ に置き換えるという操作を行う．よって，duplication degree が高ければ高いほど圧縮率は良くなる．そして，3 章で述べた通り，圧縮率が良くなればそれに応じて頻出木パターン発見の計算時間も短くなるはずである．その結果を示したのが図 4 である．ここでも uncompressed がほぼ一定のままであるのに対して TG-DAG と UTG-DAG は duplication degree の上昇に反比例して計算時間が短くなっているのが分かる．この実験で用いた treebank.xml は UTGCA で圧縮しても TGCA で圧縮した場合と圧縮率はほとんど変わらないために，計算時間もほぼ同じ値となり図 4 では線が重なってしまっているが，厳密には計算時間を測定したすべての duplication degree において僅かながら UTG-DAG の方が計算時間は短かった．

次に，最小サポートの値と両アルゴリズムの計算時間に関する実験を行った．この実験で用いた dblp.xml は，UTGCA で圧縮した場合と TGCA で圧縮した場合とでは UTGCA の方が倍近く圧縮率が良いという差が生じる．この実験の結果を示したのが図 5 である．最小サポートの値が大きくなるにつれて uncompressed と TG-DAG/UTG-DAG の差は多少は大きくなり，TG-DAG/UTG-DAG の方が有利になっているといえる．また，TG-DAG と UTG-DAG の計算時間を比べてみると，$\sigma = 6\%$ までは圧縮率で優れる UTGCA の方が最小サポート値の増加とともに徐々にアドバンテージを広げているのに対し，$\sigma = 7\%$ からは逆に差が縮まっている．これは，$\sigma = 7\%$ 以上における頻出パターンはすべてサイズが 1 の木パターンであったということと，TGCA と UTGCA では大きさ 2 以下の部分木とそれに対応する規則だけに限った圧縮率ではまったく差が生じないということから，このように最小サポート値の増加とともに逆に差が縮まったと考えられる．

5. まとめと今後の課題

本論文では，木文法により圧縮された木構造データの構造を TG-DAG として定義し，それを用いて木構造データからの頻出木パターンの発見をより効率的に行うための手法を提案した．

今後の課題としては，半構造化文書における属性やテキストへの対応が挙げられる．別の課題としては，木構造データの圧縮を用いて計算の効率化を実現するという本研究の手法を頻出木パターンの発見以外の行為にも適用の余地がないかを検討していきたいと考えている．

FREQT[Asai 02] においてデータ木の親をたどる必要があるのは出現リストに，最右葉のみ保持しているからである．TG-DAG においては根からのパスとして保存しなくてはいけないので，根からパターン候補生成を行っていけば，[Murakami 08] における出現リストの処理のオーバーヘッドは回避できることが期待される．

謝辞

本研究の一部は科研費 (19300046, 20700135) の助成を受けたものである．

参考文献

[Asai 02] Asai, T., Abe, K., Kawasoe, S., Arimura, H., Sakamoto, H., and Arikawa, S.: Efficient Substructure Discovery from Large Semi-structured Data, in *Proc. of the 2nd SIAM International Conference on Data Mining (SDM'02)*, pp. 158–174, SIAM (2002)

[Asai 03] Asai, T., Arimura, H., Uno, T., and Nakano, S.: Discovering Frequent Substructures in Large Unordered Trees, in *Proc. of the 6th International Conference on Discovery Science (DS'03)*, pp. 47–61 (2003)

[Murakami 08] Murakami, S., Doi, K., and Yamamoto, A.: Finding Frequent Patterns from Compressed Tree-structured Data, in *Proc. of the 11st International Conference on Discovery Science(DS'08)*, pp. 284–295, Lecture Notes in Artificial Intelligence 5255 (2008)

[Nijssen 03] Nijssen, S. and Kok, J. N.: Efficient Discovery of Frequent Unordered Trees, in *the 1st International Workshop on Mining Graphs, Trees and Sequences (MGTS-2003)*, pp. 55–64 (2003)

[小沼 06] 小沼 潤, 土井 晃一郎, 山本 章博：木文法を用いた半構造化文書の圧縮と反単一化, 信学技報 AI2006-9, pp. 45–50, (社) 電子情報通信学会 (2006)