D-001

構造情報の再利用による XML データに対する類似検索の高速化
Reusing Structural Information for Faster Similarity Search over XML Data

小柳涼介 † 天笠俊之 ‡ 北川博之 ‡
Ryosuke Koyanagi Toshiyuki Amagasa Hiroyuki Kitagawa

1. まえがき

Extensible Markup Language (XML) § は現在，様々な用途に広く使われている．このため，内容が類似している情報の散在がしばしば見かけられる．このような類似した複数の情報を見つけ出すことができれば，情報の相互補完，重複部分の除去，コピーの検出等，様々な用途に活用できる．特に，与えられた XML データ（クエリ XML）に対して，データベース中の XML データ（対象 XML）において類似したすべての部分木を列挙する問題を類似部分木探索問題と呼び，これまでにいくつかの手法が提案されている [1, 2, 3]．

[1, 2] では，XML を木構造とみなし木編集距離を用いることで類似度を測っている．[1] は類似し得る部分木ノード数の上界を定めておき，候補となりうる部分木に対してのみ木編集距離計算を行っている．[2] では，クエリと共通のテキストノードを持たない部分木に対して，テキストを除いて等価な部分木を同一の木とみなし計算回数を削減している．

我々は [3] において，XML におけるテキスト情報の重要性について触れ，木編集距離に加えテキストの類似度も考慮することで XML に対する柔軟な類似検出手法を提案した．さらに，テキストの類似度による枝狩りを行うことで木編集距離計算回数を大幅に削減した．

これらの手法はどれも木編集距離計算にかかる計算時間がボトルネックとなっており，いかに木編集距離計算回数を減らすか，という点で効率化を図っている．

実世界に存在する XML データのうちの多くは DTD や XML Schema によって予め構造が定義されており，その定義に従って XML が作成，出力されている．定義に従って出力されている限り，要素名や各要素が子要素として取りうる要素は限られているため，巨大な XML データには似た構造を持つデータが頻出しやすいと考えられる．本手法ではこの点に着目し，予め対象 XML に含まれるすべての部分木の構造情報を列挙し同じ構造を持つ部分木を一元化しておくことで，木編集距離の計算結果の再利用による高速化を図る．

2. 提案手法

2.1. 問題定義

本手法では，XML データ D に含まれる任意の部分木 D_i とクエリ Q 間の構造類似度，テキスト類似度を以下のように定める．

$$Sim_s(D_i, Q) = 1 - \frac{TED(D_i, Q)}{|D_i| + |Q|} \tag{1}$$

$$Sim_t(D_i, Q) = Jaccard(W_{D_i}, W_Q) \tag{2}$$

ここで，D_i は D における後行順番号 i 番目のノードを根とする部分木を表し，Q はクエリ XML を表す．さらに，$|D_i|, |Q|$ はそれぞれの木に含まれるノード数を，W_{D_i}, W_Q はそれぞれの木が持つテキストノードに含まれる単語の集合を表す．

これら二つの類似度をパラメータ α で重み付けをし統合した値を XML データ D_i, Q 間の類似度とする．XML データ D における類似部分木探索問題とは，与えられた Q, θ, α に対して，以下の条件を満たす D 中の全ての部分木 D_i を列挙する問題である．

$$\alpha Sim_s(D_i, Q) + (1 - \alpha) Sim_t(D_i, Q) \geq \theta \tag{3}$$

2.2. 類似度

本手法では，テキスト類似度として二つの部分木が持つテキストノードに含まれる単語集合間の Jaccard 係数を使用する．Jaccard 係数の計算には b -bit Minwise Hashing [4] を用いる．対象 XML の全ての部分木に対して予め b-bit Minhash Sketch を求めておくことで，クエリ処理時に高速に Jaccard 係数を求めることができる．

また，式 3 を変形させることで

$$Sim_t(D_i, Q) < \frac{\theta - \alpha}{1 - \alpha} \tag{4}$$

が成り立つとき，D_i と Q は類似し得ないことがわかる．上式の右辺の値をテキスト類似度の下限値とする．

また，構造類似度として二つの部分木の木編集距離 [5] を使用する．木編集距離の計算には [6] の手法を使用する．

2.3. 構造番号

予め対象 XML の持つ構造情報を調べておくことで，クエリ処理時の計算を効率化する．

ある部分木 D_i, D_j についてテキストノードを無視すると木が等しくなる時，その部分木同士を「構造が等しい部分木」とする．すべての部分木に対し，構造が等しい部分木は同じ構造番号を，構造が異なる部分木は違う構造番号を持つように構造番号を割り当てる．また，木 D_i の構造番号を $St(i)$ とし，構造番号が t であるような任意の木を $\hat{D}_t$ とする．

2.4. 構造類似度の上限と下限

ある部分木 D_i について全てのテキストノードをクエリ中のどのノードともマッチしないように置き換えた木を D_i^ϵ とする．この時，空文字列に置き換えたすべてのノードがクエリのどのノードともマッチしないと考えた時，以下の式が成り立つ．

$$TED(D_i^\epsilon, Q) \geq TED(D_i, Q) \tag{5}$$

次に，ある部分木 D_i についてテキストノードをクエリ中の任意のノードと一致するように置き換えた木を D_i^* とする．この時，以下の式が成り立つ．

$$TED(D_i, Q) \geq TED(D_i^*, Q) \tag{6}$$

さらに，$TED(D_i^*, Q), TED(D_i^\epsilon, Q)$ は，構造が等しい部分木ならばそれぞれは同じ値となる．

これらのことから以下の定理を示すことができる．

定理 1.

$$Sim_s(\hat{D}_t^\epsilon, Q) \leq Sim_s(\hat{D}_t, Q) \leq Sim_s(\hat{D}_t^*, Q) \tag{7}$$

2.5. アルゴリズム

提案手法の具体的アルゴリズムを以下に示す．

まず，予め対象 XML に対する前処理として全てのノードの，後置順番号，子孫ノードの最小後置順番号，構造番号，ノードの値，b -bit Minhash Sketch を後置順でファイルに格納しておく．

クエリが与えられたら，クエリ中の各部分木の b -bit Minhash Sketch を計算した後，対象 XML をファイルから後置順に読出し，各ノードに対し以下の処理を行う．

1. D_i のノード数が上限値以上ならスキップする．
2. $Sim_t(D_i, Q)$ を計算し，テキスト類似度の下限値未満ならスキップする．
3. D_i がクエリとの共通テキストノードを持っているなら $1 - TED(D_i^\epsilon, Q)$ を計算し，構造類似度とする．

†筑波大学システム情報工学科コンピュータサイエンス専攻
‡筑波大学システム情報系情報工学域
§http://www.w3.org/XML/

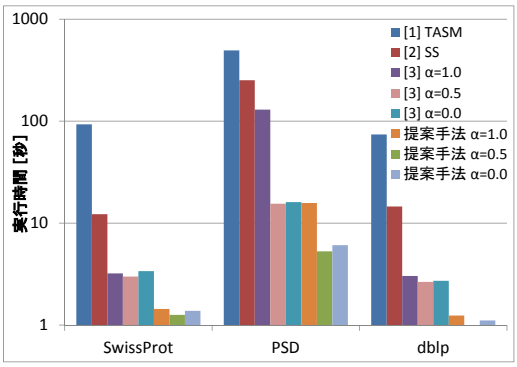

図 1: 既存手法との比較

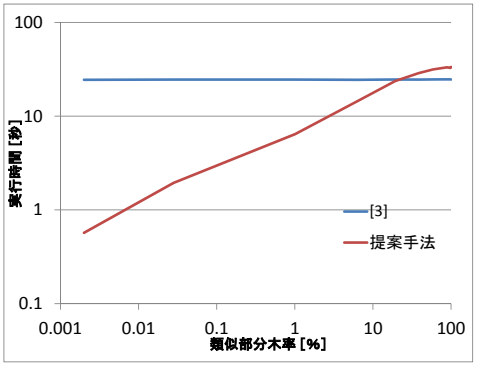

図 2: 類似部分木率による実行時間の変化

4. D_i がクエリとの共通テキストノードを持っていないなら $1-TED(D_i^*,Q)$ を計算し，構造類似度の上界とし，閾値未満であればスキップする．
5. $1-TED(D_i,Q)$ を計算し，構造類似度とする．
6. 総合類似度を求め，閾値以上ならば D_i を結果として出力する．

ここで，求めた $TED(D_i^\epsilon,Q), TED(D_i^*,Q)$ は構造番号と紐つけて記憶しておき，同じ構造番号を持つ部分木が読み込まれた時に再利用する．

3. 評価実験

実際に XML ファイルを使用し，計算時間を計測した．対象 XML として実データである dblp, SwissProt, PSD (Protein Sequence Database) と，類似部分木率を調節した人工データを使用した．実データは全て XML Data Repository ¶ にて配布されているデータを使用した．

なお，本実験は全て 64bit Windows 7, Intel Core i7-2600 3.40GHz, 8GB RAM で構成された PC で実行されている．実験プログラムは C++で記述され，x86_64-w64-mingw32 でコンパイルを行った．既存手法である Structure Search は著者により提供された Java によるプログラムを JRE 64bit 上で実行し実験を行った．

3.1. 既存手法との比較

dblp, SwissProt, PSD を対象 XML とした時の，既存手法との実行時間の比較を図 1 に示す．なお，本実験では $\theta=0.8, |Q|=64$ とし，SS, TASM では類似度上位 5 件の検出を行った．

木編集距離のみを考慮した $\alpha=1.0$ を TASM, SS, [3] と比較すると，どのケースでも大きく実行時間が改善されていることが分かる．TASM とは最大 64 倍，SS とは最大 16 倍，[3] とは最大 8 倍の速度の改善が見られた．

また，テキスト類似度が考慮されている $\alpha=0.5, 0.0$ では，[3] と比べ b-bit Minwise Hashing を採用したことによる高速化が確認できた．しかし，本手法は b-bit Minwise Hashing による Jaccard 係数の推定を行っているため，愚直に Jaccard 係数の計算を行っている [3] に対し若干の誤差が生じる．

3.2. 類似部分木率による実行時間の比較

次に，以下の手順により生成した対象 XML を使用し，類似部分木率を変化させた時の実行時間の変化を調べた．

1. dblp から 64 ノードの部分木をクエリ XML として抽出する．
2. 抽出した部分木に対しランダムにノイズを混入させた部分木を 50000 個含むような XML を生成し，対象 XML とする．
3. ノイズを混入させる確率により類似部分木率を調節する．

[3] と提案手法の実行時間の比較を図 2 に示す．なお，本実験ではどちらの手法も $\alpha=1.0, \theta=0.8$ としている．

提案手法では構造毎に上界を求めて枝刈りを行っているが，類似部分木率が大きくなると枝刈り回数が少なくなり，効率が落ちていることが分かる．

また，20% を境にして提案手法は [3] よりも効率が悪くなっている．これは，上界を求める為の $TED(D_i^*,Q)$ の計算がボトルネックとなっているためであると考えられる．しかし，実データでの実験結果から，実際の XML データではクエリと類似している部分木が大部分を占めるということは稀であると考えられる．

4. まとめ

本稿では，類似部分木探索問題をテキストと構造を考慮して解いた [3] に対し，XML データが持つ構造の特徴を利用し高速化する手法を提案した．部分木の構造毎に構造類似度の上界を求めることで，枝刈りを行い木編集距離計算回数の削減を実現した．XML データによっては構造類似度の上界の計算がオーバーヘッドとなる可能性があるが，実データを対象とした実験では既存手法に比べて大幅な実行時間の改善が確認できた．

今後の課題として，本手法では構造を考える際に無視するノード（本手法ではテキストノード）の選択方法の考察や，構造の出現頻度を考慮した上界計算，b-bit Minwise Hashing を使用したことによる精度劣化の検証などが挙げられる．

参考文献

[1] Augsten, N., Barbosa, D., Bohlen, M. M. and Palpanas, T.: Efficient Top-k Approximate Subtree Matching in Small Memory, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 23, No. 8, pp. 1123–1137 (2011).

[2] Cohen, S.: Indexing for Subtree Similarity-search Using Edit Distance, in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pp. 49–60, New York, NY, USA (2013), ACM.

[3] 小柳涼介, 天笠俊之, 北川博之：テキストおよび構造の類似度に基づいた XML データに対する効率的な類似検索, in *DEIM Forum 2014 D7-5* (2014).

[4] Li, P. and Konig, A. C.: b-Bit minwise hashing., in Rappa, M., Jones, P., Freire, J. and Chakrabarti, S. eds., *WWW*, pp. 671–680, ACM (2010).

[5] Bille, P.: A Survey on Tree Edit Distance and Related Problems, *Theor. Comput. Sci.*, Vol. 337, No. 1-3, pp. 217–239 (2005).

[6] Zhang, K. and Shasha, D.: Simple Fast Algorithms for the Editing Distance Between Trees and Related Problems, *SIAM J. Comput.*, Vol. 18, No. 6, pp. 1245–1262 (1989).

¶ http://www.cs.washington.edu/research/xmldatasets/www/repository.html