T2R2 東京工業大学リサーチリポジトリ
Tokyo Tech Research Repository

論文 / 著書情報
Article / Book Information

論題	Office XML文書の部分木に着目した類似スタイルのファイル検索
Title	Style Search for Office XML Documents using XML Sub-tree Similarity
著者	上垣外英剛, 渡辺陽介, 横田治夫
Author	Hidetaka KAMIGAITO, Yousuke WATANABE, Haruo YOKOTA
掲載誌/書名	第4回データ工学と情報マネジメントに関するフォーラム(DEIM2012), , ,
Journal/Book name	DEIM2012, , ,
発行日 / Issue date	2012, 3

Powered by T2R2 (Tokyo Institute Research Repository)

Office XML 文書の部分木に着目した類似スタイルのファイル検索

上垣外英剛† 渡辺 陽介†† 横田 治夫†††

† 東京工業大学 工学部 情報工学科
†† 東京工業大学 学術国際情報センター
††† 東京工業大学 大学院情報理工学研究科 計算工学専攻
〒 152-8552 東京都目黒区大岡山 2-12-1
E-mail: †kamigaito.h.aa@m.titech.ac.jp, ††watanabe@de.cs.titech.ac.jp, †††yokota@cs.titech.ac.jp

あらまし 近年，Office 文書に代表されるような，フォント，配置，サイズ等の詳細なスタイル情報を含んだ複数の XML ファイルから構成されるアーカイブ形式の文書が増加している．しかし，既存のテキストベースの検索エンジンは，本文の文字列やタグ情報などを検索対象とするのみで，スタイル情報を活用してはいない．スタイル情報まで踏み込んだ検索ができれば，本文の内容に依らずに体裁の類似した文書を探し出すなどの要求が実現できると考えられる．そこで本稿では，Office 文書内のスタイル情報の XML ファイルに着目した類似検索手法として SOS を提案する．SOS の内部処理における XML ファイル同士の類似度計算の精度向上のため，既存の XML ファイルの類似度計算アルゴリズム LAX を改良した手法 LAX+を提案する．提案手法に対し，実際に docx，xlsx，pptx のファイルを用いて，スタイルの類似した文書を検索した際の適合率，再現率を評価する．

キーワード Office Open XML, XML Similarity, 類似スタイル検索

Style Search for Office XML Documents using XML Sub-tree Similarity

Hidetaka KAMIGAITO†, Yousuke WATANABE††, and Haruo YOKOTA†††

† Department of Computer Science, Tokyo Institute of Technology
†† Global Scientific Information and Computing Center, Tokyo Institute of Technology
††† Department of Computer Science, Graduate School of Information Science and Engineering
Tokyo Institute of Technology
2-12-1 Ookayama, Meguro-ku, Tokyo, 152-8550 Japan
E-mail: †kamigaito.h.aa@m.titech.ac.jp, ††watanabe@de.cs.titech.ac.jp, †††yokota@cs.titech.ac.jp

Abstract Recently, the number of office documents represented by XML archive format is increasing. XML files in office documents contain information about page structures and styles such as font, size and arrangement. But, existing text-based search engines do not focus on these style information. If we utilize them, we can achieve similarity search for office documents based on structures and styles. We propose SOS, a similarity search method based on structures and styles of office documents. To compute similarity between office documents, we have to compute similarity of XML files in the office documents. We also propose an XML similarity algorithm LAX+ which is an extension of existing XML leaf node clustering algorithm LAX. In our experiments, we use docx, xlsx and pptx files and evaluate SOS and LAX+ by precision and recall.

Key words Office Open XML, XML Similarity, Style Search

1. はじめに

近年，Microsoft Office に代表されるような XML 記述を用いたアーカイブ形式の文書が増加している．これらの文書は内容と共に，フォント，テキストサイズ，配置，文字色，リストのマーカー等を始めとした多くの構造情報・スタイル情報を XML ファイルに記録している．こうしたファイル数の増加に伴い，Office 文書の検索要求は高度化・多様化してきている．

しかしながら，既存のテキストベースの検索エンジンは，本文の文字列やタグ情報などを検索対象とするのみで，スタイル情報を活用してはいない．スタイル情報まで踏み込んだ検索ができれば，本文の内容に依らずに体裁の類似した文書を探し出

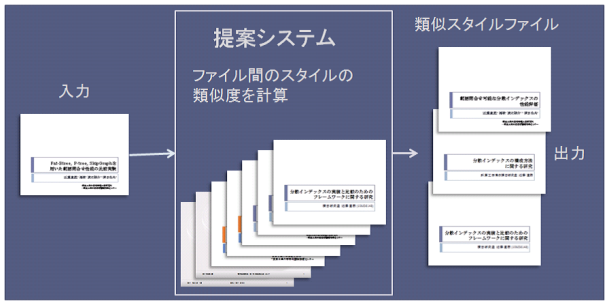

図 1　類似スタイル Office 文書の検索システム

したり、テンプレートによる文書の管理を行うなどの要求が実現できると考えられる．

本研究では図 1 に示すような，ある Office 文書ファイルを与えるとそれに類似したスタイルを持つ Office 文書をランキングして返す検索システムの構築を目指す．そのための Office Open XML 文書 (以下 OOXML 文書) を対象にしたスタイル類似度検索手法として，Structure-based Office Document Search (以下 SOS) を提案する．SOS では，まず OOXML 文書内に含まれる複数 XML ファイル同士の類似度を計算し，そしてそれらを集約して OOXML 文書間の類似度を求めている．よって，XML ファイル間の類似度を精度よくかつ効率的に計算するアルゴリズムが特に重要な役割を担う．本研究では XML ファイル間の類似度計算アルゴリズムとして，我々の研究グループが以前に提案したアルゴリズム LAX [11] を改良した LAX+を提案している．本稿では提案手法 SOS, LAX+に対し，実際に docx，xlsx，pptx のファイルを用いて，スタイルの類似した文書を検索した際の適合率，再現率を評価する．

本稿の構成は以下の通りである．まず，2. 節で関連研究について述べ，3. 節で提案手法を理解するために必要な前提知識である OOXML の概要を説明する．次に 4. 節において SOS を提案し，5. 節にて XML ファイル間の類似度計算アルゴリズムとして，既存手法の LAX と今回改良を加えた LAX+を説明する．6. 節で提案手法 LAX+および SOS の評価を行い，最後に 7. 節でまとめと今後の課題を述べる．

2. 関 連 研 究

Office 文書の本文の内容検索を実現した検索エンジンであればすでに多く存在する．ローカル環境を対象としたものは Windows デスクトップサーチ [2], Mac OS X の Spotlight [3] が，Web 上では Google filetype 検索 [4] 等がある．基本的にこれらの検索エンジンでは，本文の文字列やタグ情報などを検索対象とするのみで，スタイル情報を活用してはいない．Office 文書のスタイル情報は XML 構造を用いた記述がなされているため，文書の内容に重点を置いた文字列による検索では，直接スタイルを検索することは難しい．

XML ファイルの類似度計算には多くの方法が存在する [5]．そのなかでも Tree Edit Distance(TED) [6] が最も有名である．TED は一般的な文字列の比較に用いられているレーベンシュタイン距離 [7] を，木構造にも使えるように拡張したアルゴリズムで，精度が高いという特徴を持つ．しかし，TED は現在最も計算量が少ない方法 [8] でも $O(n^3)$ であり，データ量が多い XML ファイルや，多くの XML ファイルを比較する用途には向いていない．一方で計算量が少ないアルゴリズムとしては tag similarity [9]，entropy を用いた手法 [10] 等が存在する．いずれも計算量 $O(n)$ と高速ではあるが，精度が低いという問題点がある．精度が高く計算量も多くはないアルゴリズムとしては LAX [11] が存在する．LAX は部分木のリーフノードを比較することで XML 文書の類似度を計算するアルゴリズムで，計算量は $O\ (n^2)$ である．本研究では，OOXML 文書内の XML ファイルを対象に，LAX を改良した手法を提案する．

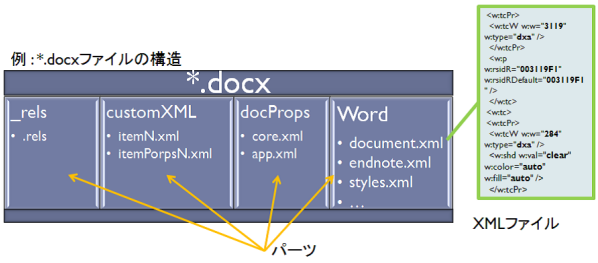

図 2　OOXML の構造

3. Office Open XML

Office Open XML ファイルフォーマット (以下 OOXML) は，Zip アーカイブとして保存されているため，解凍することで構造を展開できる．OOXML の構造を.docx を例として図 2 に示した．OOXML の構造はいくつかの Markup Langauge(以下 ML) によって定義されている [1]．主にパーツと，パーツ間の関係を記述したファイルで構成されていて，各パーツは 1 つ以上の XML ファイルによって構成されている．この節ではスタイル情報を読み取る際に重要なパーツファイルについて説明する．

- *Shared ML*: WordProcessing ML，Spreadsheet ML，Presentation ML において共通して使用されるパーツを含んでいる．これらのパーツはメタデータかユーザー定義に関係する情報を含んでおり，スタイル情報を検索する際に重要である．
- *WordProcessing ML*: 主に.docx 拡張子の文書に使用されている．WordProcessing ML では document.xml が多くのパーツを参照しており，中心的な役割を果たしている．また単一の document.xml でメインとなる内容を記録をしている．重要な役割が document.xml に集約されているため WordProcessing ML は簡単な構造をしている．注釈に関するパーツが多いのも特徴である．
- *Spreadsheet ML*: 主に拡張子.xlsx のファイルに使用されている．Spreadsheet ML を構成するパーツファイルの数は他の ML と比較して非常に多い．これはワークブック，ワークシート，セルのそれぞれの区分けで文書を管理しているからである．実用上 xlsx は主にワークシートとそれらを取りまとめるワークブックによって構成されているが，Spreadsheet ML の

構造もそれを踏まえた形となっている．複数のセルに関係するパーツは worksheetN.xml ($N = 1, 2, \ldots$)，複数のワークシートに関係するパーツは workbook.xml からそれぞれの rels ファイルを通じて参照される．ただし変更履歴はワークシートに関するものであっても workbook.xml から間接的に参照される構造になっている．

- *Presentation ML*: 主に拡張子.pptx のファイルに使用されている．ほとんどのパーツが presentation.xml と slideN.xml ($N = 1, 2, \ldots$) から参照されており，これら 2 つのパーツが Presentation ML の中心的な役割を果たしている．また，その他のパーツの中でも slideMasterN.xml と slideLayoutN.xml はそれぞれ直接，もしくは間接的に presentaion.xml か slideN.xml から参照されており，他のパーツと比べると重要性が高い．

OOXML はスタイル情報が 1 つ以上の XML によって構成される複数のパーツによって記述されているため，単一の XML ファイルの比較では十分ではない．そのため全体的なスタイルの検索に必要な類似度計算を行うには，スタイル関連の情報を多く含む複数の XML ファイル間での比較を考慮する必要がある．次節では複数の XML ファイル間での類似度計算を行い，Office 文書に対するスタイル類似度検索を行う手法について説明する．

4. Structure-based Office document Search

本節では提案手法 Structure-based Office document Search(以下 SOS) について説明する．

以下では，類似検索におけるクエリに相当する OOXML 文書を o_q，検索対象となる OOXML 文書集合を $O = \{o_i | 1 \leqq i \leqq I\}$ と表記する．前述の通り，各 OOXML 文書 o_i の内部は複数のパーツの集合 $P_i = \{p_{ij} | 1 \leqq j \leqq J_i\}$ によって構成されており，各パーツ p_{ij} も複数の XML ファイルの集合 $F_{ij} = \{f_{ijk} | 1 \leqq k \leqq K_{ij}\}$ によって構成されている．そのため，2 つの OOXML 文書 o_q, o_i 間の類似度 $S_{oo}(o_q, o_i)$ を求めるには，下の階層から段階的に集約していく必要がある．まずパーツを構成している XML ファイルの類似度 S_{xml} を計算し，その後パーツごとの類似度 S_{parts} を集計し，最後に OOXML 文書の類似度 S_{oo} が決定される．

SOS における OOXML 文書同士の類似度の大まかな計算手順は以下のようになっている．

（1） **XML ファイル同士の類似度計算**:
o_q と o_i 中の j 番目のパーツ p_{qj}, p_{ij} に含まれる XML ファイルのうち，ファイル名が対応するもののペア (f_{qjk}, f_{ijk}) の類似度 $S_{xml}(f_{qjk}, f_{ijk})$ を計算する．この値は，5. 節で述べる LAX(従来手法) または LAX+(提案手法) によって求められる．

（2） **パーツ同士の類似度計算**:
OOXML 文書に含まれる同一パーツ間の類似度を，そのパーツに属する XML ファイルの類似度を集約して計算する．OOXML 文書 o_q, o_i の中の j 番目のパーツ p_{qj}, p_{ij} 間の類似度 $S_{parts}(p_{qj}, p_{ij})$ は，各パーツが含んでいる XML ファイル集合 F_{qj}, F_{ij} から以下の式で求められる．

$$s_{q \to i} = \sum_{f_{qjk} \in F_{qj}, f_{ijk} \in F_{ij}} S_{xml}(f_{qjk}, f_{ijk}) \quad (1)$$

$$s_{i \to q} = \sum_{f_{qjk} \in F_{qj}, f_{ijk} \in F_{ij}} S_{xml}(f_{ijk}, f_{qjk}) \quad (2)$$

$$s = \begin{cases} s_{q \to i} & (|F_{qj}| > |F_{ij}|) \\ s_{i \to q} & (otherwise) \end{cases} \quad (3)$$

$$S_{part}(p_{qj}, p_{ij}) = \frac{s}{max(|F_{qj}|, |F_{ij}|)} \quad (4)$$

片方の OOXML 文書にしか j 番目のパーツが存在しない場合は $S_{parts}(p_{qj}, p_{ij}) = 0$ とする．$s_{q \to i}$ と $s_{i \to q}$ を分けて求める理由は，LAX が $S_{xml}(f_x, f_y) \neq S_{xml}(f_y, f_x)$ となる非対称な性質の類似度尺度であるためである．本研究で改良を行った LAX+は対称な類似度尺度であるため，LAX+を用いて XML ファイルの類似度を求めている場合は，2 回同じ計算を行う必要はない．

（3） **OOXML 文書同士の類似度計算**:
OOXML 文書同士の類似度を，それらに含まれるパーツの類似度から計算する．OOXML 文書 o_q, o_i の類似度 $S_{oo}(o_q, o_i)$ は，各文書に含まれるパーツの集合 P_q, P_i から以下の式で計算される．

$$S_{oo}(o_q, o_i) = \sum_{p_{qj} \in P_q, p_{ij} \in P_i} \left(\frac{w_j}{\sum_{l=1}^{|P_q|} w_l} \times S_{part}(p_{qj}, p_{ij}) \right) \quad (5)$$

ただし，w_j は j 番目のパーツに対する重みづけ変数を表わしている．

（4） **OOXML 文書類似度に基づくランキング**:
文書集合 $O = \{o_i | 1 \leqq i \leqq I\}$ の要素のうち，類似度 $S_{oo}(o_q, o_i)$ の値が閾値を超えた o_i のみ，値の高い順にランキングする．なお，閾値は 0 から 100 の範囲で指定可能である．

5. XML ファイル間の類似度計算

この節では本研究で提案する XML ファイルの類似度計算アルゴリズム LAX+について説明する．OOXML 文書は XML ファイルのリーフノードに主なスタイル情報が記録されているため，OOXML 文書の検索を実現するためにはリーフノードに着目した XML 類似度計算手法を用いることが望ましいと考えられる．また SOS では複数の XML ファイル間で比較を行うため，計算量が少ない手法でなければならない．そこで本研究では，計算量が $O(n^2)$ と高速で精度も高くリーフノードに着目したアルゴリズムである LAX [11] に着目して，LAX に改良を加えた LAX+を SOS における XML 文書の類似度計算関数とする．

まず 5.1 節で LAX の概要について述べ，5.2 節で LAX を OOXML 文書に適用する際の問題点について述べる．その後，5.3 節で LAX+について述べる．

5.1 既存手法: LAX

LAX は与えられた 2 つの XML ファイル f_{base}, f_{target} に対して類似度 $S_{xml}(f_{base}, f_{target})$ を求めるアルゴリズムである．非対称な性質を持つ類似度尺度であるため，ここでは第 1 引

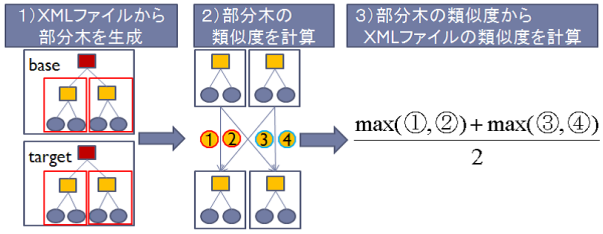

図 3　LAX

数のファイルを base, 第 2 引数のファイルを target と区別する．LAX の処理手順を図 3 に示し，以降では各手順の詳細を述べる．

（1）　部分木の生成:

LAX では XML ファイルの木構造を部分木に切り分ける点をカッティングポイントと呼んでいる．XML ファイル内のあるノード x が，どれだけカッティングポイントに相応しいかを表す値 w_x は以下の式で表される．

$$w_x = |children(x)| \times d_x^i \tag{6}$$

ただし，$|children(x)|$ はノード x の子ノードの数，d_x はノード x からリーフノードまでの最大距離，i は重み付け変数を表わしている．base，target 両 XML ファイルそれぞれに対して，全ノード中最も大きい w_x を持つノード x を見つけ，x の子ノードをルートノードとした部分木に元の XML ファイルを分割する．以降では，分割によって生成された部分木の集合を $T = \{t_u | 1 \leqq u \leqq n\}$ と表記する．

（2）　部分木同士の類似度計算:

base の部分木集合と target の部分木集合のそれぞれの要素同士で，部分木としての類似度を求める．LAX では，部分木 t_u, t_v の類似度を計算する関数 $S_{sub}(t_u, t_v)$ は以下の式で定義される．

$$S_{sub}(t_u, t_v) = \frac{|Leaf_Pairs(t_u, t_v)|}{|Leaf(t_u)|} \times 100 \tag{7}$$

ただし，$|Leaf(t_u)|$ は部分木 t_u に属するリーフノード数，$|Leaf_Pairs(t_u, t_v)|$ は部分木 t_u，t_v に属するリーフノードの中で，PCDATA が同じペアを作った場合のペア数とする．

（3）　XML ファイルの類似度計算:

部分木同士の類似度から XML 文書全体の類似度を求める．LAX における XML ファイル f_{base}，f_{target} の類似度を求める関数 $S_{xml}(f_{base}, f_{target})$ は，各ファイルの部分木集合 T_{base}, T_{target} から以下の式で定義される．

$$S_{xml}(f_{base}, f_{target}) = \frac{\sum_{t_u \in T_{base}} (max_{t_v \in T_{target}}(S_{sub}(t_u, t_v)))}{|T_{base}|} \tag{8}$$

5.2　OOXML に LAX を用いる際の問題点

ここでは OOXML に LAX を使用した場合に生じる問題点について述べる．各問題の解決については 5.3 節で改良手法を提案する事で行う．

（1）　**リーフノード数の差による不公平**:

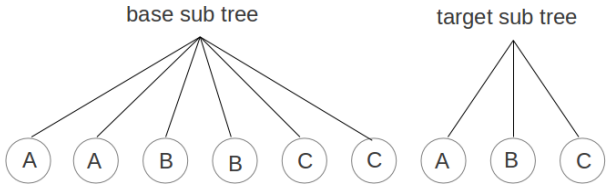

図 4　リーフノード数の差

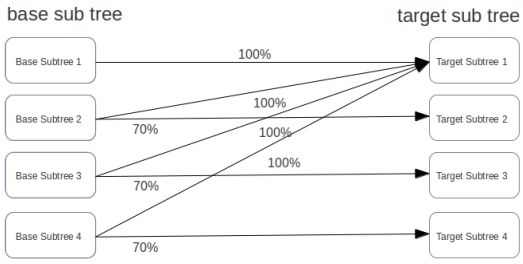

図 5　局所一致による影響

XML ファイルではテキスト量の増大に伴い，一つの部分木のリーフノード数が増大する傾向がある．LAX では部分木の比較に式 7 を使っているために，base 側の部分木のリーフノードが多い場合に部分木の類似度が低くなってしまう．例として図 4 の場合，リーフノードの傾向は base，target で類似性が見られるが，base のリーフノードが 6 つ，target が 3 つなので，類似度が 50%を上回ることはない．

（2）　**局所一致による影響力の強さ**:

LAX では式 8 において max を求めているため，局所的な一致が，XML ファイル全体の類似度に強く反映されてしまうことがある．図 5 では base の全ての部分木が Target_Subtree_1 との比較で 100%の類似度であるため，類似度計算によって求められた base と target の類似度は 100%となる．しかしこの値は base の各部分木と Target_Subtree_1 だけの類似度が強く反映された結果で，極めて局所的であり，XML ファイル全体の類似度とは言い難い．

（3）　**部分木のサイズの差による不公平**:

LAX では式 8 で部分木の類似度を求める際に，リーフノード数の差を考慮していない．そのため，リーフノードが少ない部分木では類似度として大きな値が出やすい一方で，リーフノードが多い部分木では大きな類似度は出にくくなる．結果として LAX では小さな部分木が多い XML ファイルほど高い値が出やすくなる傾向がある．

5.3　提案手法: LAX+

5.2 節の問題を解決するため，LAX に対して改良を行った LAX+を提案する．改良の詳細は 5.3.1 節，5.3.2 節，5.3.3 節にて詳しく説明する．

5.3.1　双方向での base と target の比較

LAX に使われている部分木の類似度 S_{sub} は非対称性があり，計算においては base となる XML ファイルから target となる XML ファイルを比較した値しか考慮されない．そのため 5.2 節（2）のような局所的な一致が起こった際にそれに対処

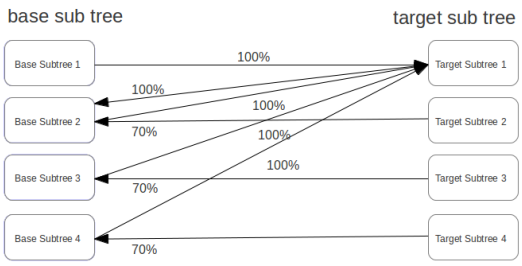

図 6　局所的な一致の改善

することができない．そこで target から base を見た値も同時に算出するように変更を行う．変更前の LAX でも元々base, target の全ての部分木のリーフノードの組み合わせを計算しているため，この方法を取ることによる計算量の増加は僅かなものとなる．

この方法を取れば図 6 の様に base，target どちらかが局所的な一致によって類似度を計算している際に，より値が小さい結果を用いるようにすることで局所的な一致を防ぐことができる．ただし，base，target 双方が局所的な一致によって類似度を計算している場合はこの方法でも局所的な一致は改善されない．

5.3.2　部分木の類似度計算

5.1 節で示したように，LAX では部分木の比較において完全一致したリーフノードのペア数を用いているために，部分木のサイズの差による影響を受けて 5.2 節の（1）の様な問題が起こってしまう．そこで部分木同士の類似度 $S_{sub}(t_u, t_v)$ の計算式を新たに $S^{+}_{sub}(t_u, t_v)$ として以下の様に変更する．

$$S^{+}_{sub}(t_u,\, t_v) = |Matched_Leaf(t_u, t_v)| \qquad (9)$$

ただし，$|Matched_Leaf(t_u, t_v)|$ は部分木 t_u，t_v に含まれるリーフノードのうち，相手側の 1 つ以上のリーフノードと PCDATA が一致するものの数とする．

LAX では部分木の比較ごとに正規化した値を部分木の類似度として算出しているため，部分木のサイズが考慮されず，5.2 節の（3）の問題が起きていたが，この方法では部分木比較の段階では正規化を行わないので，この問題も避けることができる．

5.3.3　XML 文書間の類似度計算

5.3.2 節の変更によって，部分木比較の段階での正規化は行われないため，XML ファイル間の類似度計算を行う際に正規化する．5.3.1 節の変更も考慮すると，それらの計算を，base から見た target の類似度，target から見た base の類似度，の 2 種類に対して行う必要がある．LAX では式 8 を使って正規化された部分木比較の結果の平均を求めていたが，改良手法では以下の式を用いる．

$$\begin{aligned} & S^{+}_{xml}(f_{base},\, f_{target}) \\ = & \; min(\\ & \frac{\sum_{t_u \in T_{base}} max_{t_v \in T_{target}}(S^{+}_{sub}(t_u,\, t_v))}{|Leaf(f_{base})|}, \\ & \frac{\sum_{t_v \in T_{target}} max_{t_u \in T_{base}}(S^{+}_{sub}(t_u,\, t_v))}{|Leaf(f_{target})|} \\ &) \end{aligned} \qquad (10)$$

ただし，T_{base}，T_{target} は base と target の XML ファイルの部分木の集合，$|Leaf(f)|$ は XML ファイル全体のリーフノードの数とする．

LAX+によって求められる XML ファイル同士の類似度は，5.3.2 節，5.3.3 節の改良によって以下のような性質を持つ．

$$S^{+}_{xml}(f_{base},\, f_{taret}) = S^{+}_{xml}(f_{target},\, f_{base}) \qquad (11)$$

この性質により，4. の（2）においては，半分の計算量で結果を求めることができる．ただし計算量のオーダー自体は，LAX の $O(n^2)$ から変化していないため，リーフノード数が増大した場合は大幅に計算時間が増えることに変わりはない．

6.　評 価 実 験

本節では，実験によって LAX+と LAX の比較，および SOS と既存の全文検索エンジンとの比較を行う．

6.1　LAX と LAX+の比較実験

ここでは，単一の XML ファイルに対する LAX と LAX+の precision，recall を実験によって求めて比較し，5.3 節にて行った LAX の改良が精度向上に貢献したかを評価し，考察を行う．

6.1.1　実験データ・実験手法

実験データは Web 上で収集した 287 個の pptx ファイルを用いた．それらを人手で 1 枚目のスライドのスタイルを基準にし，53 個のグループに分類して正解情報を作成した．

実験は LAX，LAX+共に以下の手順で行う．

（1）　pptx 文書群から検索クエリ o_q を 1 つ選択し，それ以外の文書集合を検索対象とする．

（2）　検索クエリと検索対象の各文書に対して，pptx 内の 1 枚目のスライドの書式情報を記録した XML ファイル slide1.xml のみを使って XML ファイル同士の類似度計算を行う．

（3）　類似度に基づいて上位 k 件までランキングし，実際の正解と照らし合わせて precision, recall を求める．

（4）　すべての文書が検索クエリとなるように手順 1, 2, 3 を繰り返し，precision, recall の平均値を求める．

6.1.2　実 験 結 果

LAX，LAX+の precision-recall グラフを図 7 に示す．横軸は recall で，縦軸はその recall 値における precision の平均値を表している．図 7 から分かるように，全体的に LAX+が LAX よりも高い precision を示した．

11 点補完平均適合率は LAX が 0.374，LAX+が 0.583 となった．最も precision, recall が接近した際のそれぞれの値は LAX は precision が 0.438，recall が 0.447，LAX+は precision が 0.562，recall が 0.555 となった．

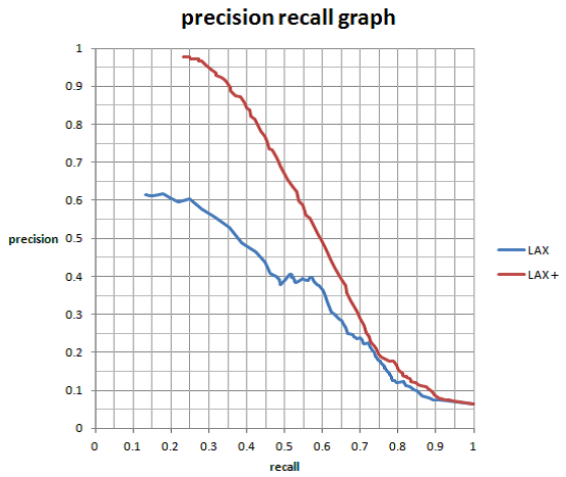

図 7　LAX(既存手法) と LAX+(提案手法) の precision-recall グラフ

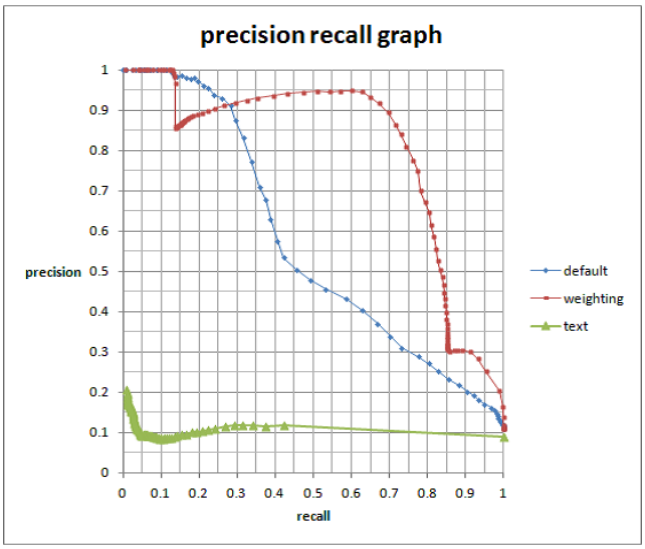

図 8　docx での precision recall グラフ

以上の結果，いずれも LAX+の方が精度が高いことから，5.3 節において加えた変更が XML ファイル同士の類似度計算の精度を改善したことが確認された．

6.2　SOS の評価実験

ここでは，SOS を用いて実際にスタイル検索が可能であるかを OOXML 文書の種類ごとに調査する．また，テキスト検索と比較することで，OOXML 文書に対するスタイル検索では XML ファイル同士の類似度の活用が有効であることを示す．さらに各 OOXML 文書の ML において重要なパーツを調べ，パーツごとの重み付けを行った場合での結果も比較する．

6.2.1　比較対象: テキスト検索

この実験で行うテキスト検索の方法について説明する．

まず，テキスト検索エンジンにはオープンソースの全文検索エンジン Apache Solr [13] を用いる．Apache Solr は文書を登録し検索クエリを POST するだけで簡単に全文検索ができる．文書の登録時には tokenizer によって単語単位に切り分けられるが，今回は tokenizer として text cjk を用いた．キーワード集合から登録された文書のスコアを求める方法は，tf-idf [14], [15] をベースとしたものとなっている．

今回の実験では，検索対象となる OOXML 文書に含まれる全 XML ファイルから開始タグ・終了タグを除去し，1 つのテキストに統合したものを Apache Solr に登録している．文書同士の類似度の計算は，検索クエリとなる文書 o_q 中の全 XML ファイルから，タグを単語間の区切りとみなしてキーワードの集合を抽出し，スコア付きのキーワード検索結果を取得する．

6.2.2　実験データ・実験手法

本実験に用いた docx, xlsx, pptx ファイルについて述べる．

- docx: Wordprocessing ML 用の実験データは，Web 上で 250 個の拡張子.docx のファイルを収集した．正解情報は，文書全体のスタイル情報を基準にし，人手により 22 個のグループに分類した．
- xlsx: Spreadsheet ML 用の実験データは Web 上で 108 個の拡張子.xlsx のファイルを収集した．正解情報は，1 枚目のシートのスタイル情報を基準にし，人手により 31 個のグループに分類した．実験に際して，1 枚目のシートの類似度に基づいた検索を行うため，対象から sheet1.xml 以外の sheetN.xml ($N = 2, 3, \ldots$) を除外した．
- pptx: Presentation ML 用の実験データは 6.1 節で使用されたものと同じ実験データを使用した．実験に際して，1 枚目のスタイルの類似度に基づいた検索を行うため，対象から slide1.xml 以外の slideN.xml，slideMasterN.xml，slide- LayoutN.xml，notesSlideN.xml，notesMasterN.xml を除外した ($N = 2, 3, \ldots$).

上記のデータを用いて，実験は SOS，テキスト検索共に以下の手順で行った．

（1） OOXML 文書群から検索クエリとなる文書 o_q を選択し，それ以外の全ての文書を検索対象として類似度計算を行う．

（2） 類似度によるランキングの上位 k 件までの結果から，実際の正解と照らし合わせて precision，recall を求める．

（3） すべての文書が検索クエリとなるように手順 1, 2 を繰り返し，precision, recall の平均値を求める．

なお，SOS に関しては，OOXML 文書全体の類似度を求める際に，パーツごとの類似度に重みを入れない場合と入れた場合での実験も行った．重みのつけ方に関しては，同一グループとの類似度を異なるグループとの類似度で割った値をそれぞれのパーツの重みとした。重みづけの最適化については今後の課題である．

6.2.3　実 験 結 果

docx, xlsx, pptx ごとの実験結果をそれぞれ図 8，図 9，図 10 に示す．図中の default は SOS を OOXML 文書内の各パーツに重み付けをしない状態で用いた際の結果，weighting は SOS を重み付け有りで用いた結果，text は 6.2.1 節のテキスト検索を用いた結果をそれぞれ表わしている．表 1 は 11 点補完平均適合率，表 2 は precision と recall が最も接近した際の値である．

まず，default(パーツ重み無し SOS) とテキスト検索との比

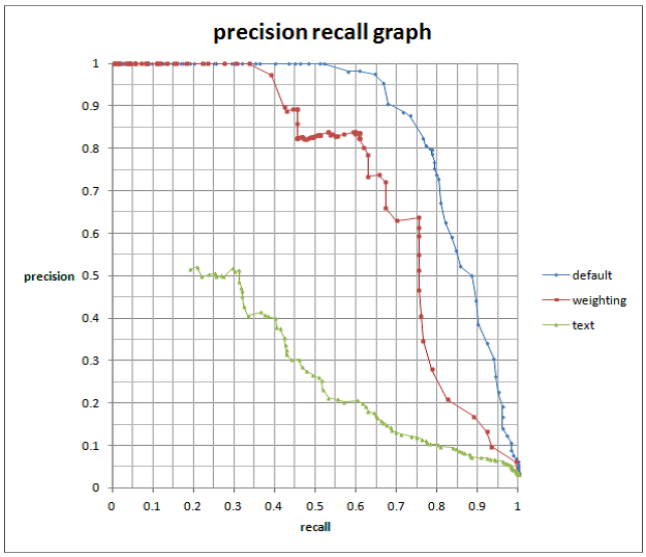

図 9　xlsx での precision-recall グラフ

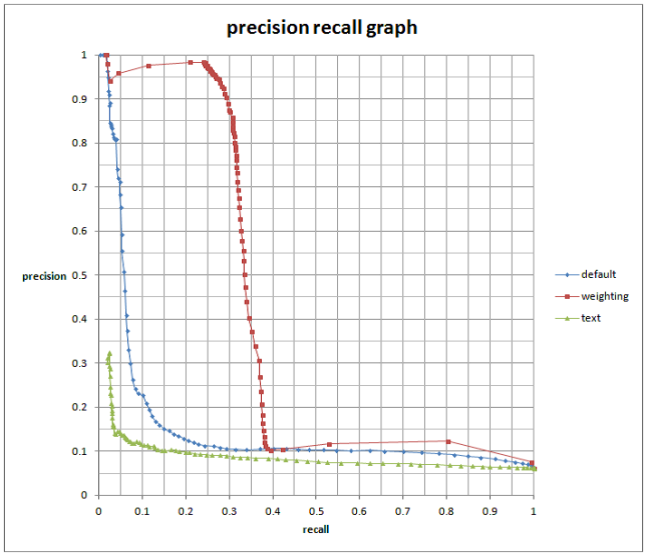

図 10　pptx での precision-recall グラフ

表 1　11 点補完平均適合率

	SOS (default)	SOS (weighting)	text
docx	0.560	0.790	0.111
xlsx	0.822	0.689	0.298
pptx	0.191	0.417	0.102

表 2　precision-recall 最接近時の値

	SOS (default)		SOS (weighting)		text	
	precision	recall	precision	recall	precision	recall
docx	0.479	0.492	0.776	0.762	0.088	0.087
xlsx	0.788	0.788	0.660	0.674	0.400	0.402
pptx	0.151	0.150	0.371	0.352	0.114	0.112

較であるが，この実験に関してはすべての文書形式で SOS がよい結果となった．xlsx に対してはテキスト検索も高い precision を示したが，これは xlsx に用いられる SpreadSheet ML が他の文書形式に比べ構造化されており，さらにキーワードを抽出するパーツを持っているからであると考えられる．

SOS の default の結果を文書形式別に見ると，図 8，図 9 で docx, xlsx に対しては高い適合率を示している．その一方，図 10 で pptx に対しては適合率が低かった．図 10 の実験は 6.1 の実験と同じデータを用いており，しかも 6.1 の方が今回の実験よりも高い値を示していることから，単体の XML ファイル同士の類似度は精度がよいが，それらを集約してパーツや OOXML 文書全体の類似度を求める処理に工夫が必要であることを示している．

SOS に重み付けを行った際の結果 (weighting) については文書形式ごとに異なる傾向を示した．各 precision-recall グラフが不自然な形をしているが，これは主に，同じ文書形式でも特定のパーツを持っている OOXML 文書と持っていない文書が存在すること，部分的な文書のグループの傾向が全体に強く反映されてしまったことの 2 つの理由による．文書形式ごとの傾向であるが，まず docx に関しては重み付け前と比較して，適合率が大幅に上昇している．xlsx については，今回試した重み付けでは適切ではなかったため，適合率が下がってしまった．pptx については重み付けの結果 precision が大幅に上昇している．しかし，この値は 6.1 の実験結果と比較するとまだ低く，スライドごとにスタイルが変えられる pptx については，パーツ単位の重みづけではなく，個別のスライドのスタイル単位での重みづけ等の更なる改良が必要であることが分かった．

今回の実験では，パーツごとの重みを適切に設定できれば，類似検索の精度を改善できることが示された．ただし，実験データごとに最適な重みは変化すると考えられるため，重みづけの最適化をどのように行うかについては今後の課題である．

7.　まとめと今後の課題

本研究では，XML ファイル同士の類似度を計算するアルゴリズム LAX+と，それを利用して OOXML 文書の類似スタイル検索を行う手法 SOS を提案した．実験により，LAX を改良した手法である LAX+が，OOXML 文書を構成するパーツに含まれている XML ファイルの類似度計算に対して有効であることを示した．また，SOS を用いて，複数の XML ファイルの比較が必要な OOXML 文書の類似スタイル検索が行えることも確認した．OOXML 文書のパーツに対する重み付けを工夫することで，検索精度をより高めることができると分かった．

今後の課題として，まずパーツの重みの設定方法が必要だが，現在の実験データのみでは各文書形式のパーツの傾向を読み取るには文書数が少ない．よって実験データを増やし，各パーツのスタイル類似検索における重要性を分析していく必要がある．別の課題として，SOS，LAX+の処理を効率化するためにインデックスを活用することが挙げられる．また，OOXML 文書と並んでよく使用される Open Document 文書に，SOS, LAX+を適用することについても検討していきたい．

謝辞　本研究の一部は日本学術振興会科学研究費補助金基盤研究 A（#22240005）の助成により行われた．

文　　献

[1]　ECMA-376 3rd edition,

http://www.ecma-international.org/publications/standards/Ecma-376.htm
[2] Windows デスクトップサーチ, http://www.microsoft.com/japan/windows/desktopsearch/default.mspx
[3] Mac OS X Spotlight, http://support.apple.com/kb/HT2531
[4] Google インデックスに登録できるファイル形式, http://support.google.com/webmasters/bin/answer.py?hl=ja&answer=35287
[5] J.Tekli, R.Chbeir, K.Yetongnon, “An overview on XML similarity: Background, current trends and future directions”, LE2I Laboratory UMR-CNRS, University of Bourgogne, 21078 Dijon Cedex, France Computer Science Review (2009).
[6] K.C.TAI. “The Tree-toTree Correction Problem”, Journal of the Association for Computing Machinery Vol 26, No 3, July 1979.
[7] V.I.Levenshtein, “BINARY CODES CAPABLE OF CORRECTING DELETIONS, INSERTIONS,AND REVERSALS”, Presented by Academician P.S.Novikov January 4, 1965) Translated from Doklady Akademii Nauk SSSR, Vol. 163,No.4,pp.845-848,August,1965. Original article submitted January 2,1965.
[8] E.D.Demaine, S.Mozes, B.Rossman, and O.Weimann, “An Optimal Decomposition Algorithm for Tree Edit Distance”, ACM Trans. Algorithms, 6(1) : Article 2, 2009.
[9] D.Buttler, “A Short Survey of Doucment Structure Similarity Algorithms”, In Proceedings of the 5th International Conference on internal Computing,USA, pp.3-9, 2004.
[10] S.Helmer, “Measuring the Structural Similarity of Semistructured Documents Using Entropy”, In Proceedings of the VLDB ’07Conference, pp.0122-1032, 2007.
[11] W.Liang and H.Yokota, “LAX: An Efficient Approximate XML Join Based on Clustered Leaf Nodes for XML Data Integration”, In Proceedings of BNCOD ’05, Springer LNCS 3567, pp82-97, 2005
[12] W.Viyanon, S.K.Madria, S.S.Bhowmick, “XML data integration based on content and structure similarity using keys”, Proceedings of the OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008. Part I on On the Move to Meaningful Internet Systems:(pp. 484-493)
[13] Apache Solar, http://lucene.apache.org/solr/
[14] G.Salton and M.J.Mcgill, “Introduction to Modern Information Retrieval”, McGraw- Hill, Tokio, 1983.
[15] G. Salton and C. Buckley “Term-Weighting Approaches in automatic Text Retrieval”, Information Processing and Management:an International Journal, 24(5), pp.513-523, 1988.