XML 索引の更新コスト削減のための部分文書の統計量に基づくフィルタの評価とその最適化

欅　惇志† 宮崎　純† 波多野賢治†† 山本豪志朗† 加藤　博一†

† 奈良先端科学技術大学院大学情報科学研究科　〒 630–0192 奈良県生駒市高山町 8916–5
†† 同志社大学文化情報学部　〒 610–0394 京都府京田辺市多々羅都谷 1–3
E-mail: †{atsushi-ke,miyazaki,goshiro,kato}@is.naist.jp, ††khatano@mail.doshisha.ac.jp

あらまし　我々は過去の研究において，XML 情報検索における動的な索引更新のための，索引スキーマと更新コスト削減を目的とした二種類のフィルタの提案を行った．本稿ではそれらフィルタの評価と，その結果を踏まえた高精度化を目指す．Web 上では常に新たなデータが作成され続けているため，Web 検索システムではデータの更新に応じて索引を更新し続けなければ，その時々によって変化するユーザの情報要求に対応することができない．その際，XML 情報検索は文書のうちの一部分，即ち部分文書を単位とした検索を行う特性上，検索対象の文書数と比較して検索対象のデータ数が大幅に増大する．それに伴い索引の更新コストも増大するため，その抑制のための取り組みを行うことで高速な索引更新の実現を目指す．評価実験の結果，初期データと追加データの割合によっては適切に索引語の重み計算ができないということが判明したものの，一方のフィルタのみを利用した場合には検索精度を維持しつつ索引構築時間を 37%，索引サイズを 18% 削減させることに成功した．

キーワード　XML 部分文書検索，動的な索引更新，更新コスト削減のためのフィルタ

An Evaluation and a Refinement of Filters Based on Statistics of XML Elements for Reducing Update Cost of Indices

Atsushi KEYAKI†, Jun MIYAZAKI†, Kenji HATANO††,

Goshiro YAMAMOTO†, and Hirokazu KATO†

† Graduate School of Information Science, Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara, 630-0192, Japan
†† Faculty of Culture and Information Science, Doshisha University
1-3 Tatara-Miyakodani, Kyotanabe, Kyoto 610-0394, Japan
E-mail: †{atsushi-ke,miyazaki,goshiro,kato}@is.naist.jp, ††khatano@mail.doshisha.ac.jp

1. はじめに

現在，様々なデータが構造化文書の形式によって記述・作成されている．中でも特に Extensible Markup Language (XML) の利用が顕著であり，XHTML によって記述された Web 文書や，電子商取引サイトにおける商品データ，オフィスアプリケーションのファイルフォーマットなど，多くの場面において用いられている．これら膨大に蓄積された XML データに対して効率的に情報アクセスを行うためには，情報検索技術を有効に活用する必要があるために，XML データに対する情報検索技術である XML 情報検索システムの開発が求められる．

XML 情報検索ではこれまで，検索精度の向上を目指した研究 [5], [12], [15] と検索速度の向上を目指した研究 [8], [9] [10] が取り組まれてきた．情報検索システムの目的はユーザの要求に合致した情報を提示することであるために高精度検索を目指す必要があることは言うに及ばない．また，XML 情報検索ではその性質上，一つの XML 文書から複数の検索対象 (部分文書) が取り出され (注1)，検索に要する時間は文書検索と比較して長

(注1)：XML 情報検索用のテストコレクションである INEX 2008 Wikipedia collection [3] では，一つの XML 文書に約平均 67 個の部分文書が含まれる．部分文書については 2.2 で詳述する．

時間に及ぶため，高速検索を実現することも重要な課題である．

これら高精度検索と高速検索の両立を目指した研究 [9], [10] も存在するが，実運用を想定した XML 情報検索システムを目指す上ではこれらの目標だけでは十分ではない．なぜならば，既存研究ではデータの更新は考慮されていないためである．現実世界では常に新しいデータが生み出され続け，特に Wikipedia(注2) のように複数人での編集を前提としたコンテンツではデータの書き換えが頻繁に発生する．これらのデータに対しても即座に検索対象として利用することができなければ，その時々の需要に合わせた情報検索を行うことができない．そこで我々は過去に，高精度 · 高速検索を維持しつつ，新たなデータの追加によって起こる索引の更新を高速に行う手法の提案を行った [16].

更新を考慮した XML 情報検索を実現するためには，以下の二つの手順，即ち，(1) 新規データ中に含まれる索引語の重みを算出する，(2) 算出された索引語の重みを索引に登録する，を踏む必要がある．(1) に関して，索引語の重みを算出するためには文書や文書集合全体から算出される統計量を用いるのが一般的である [5]．それに対して従来の研究では，高速な検索を実現するため検索結果を提示する際に不要な統計量はデータベースへ格納しておらず，新たに追加された XML 文書中の索引語の重みを即座に算出することができない．従って，データの更新に対応するためには，予め索引語の重み計算に必要な統計量をデータベースへ登録し，高速な検索を維持しつつ即座に新規索引語の重み計算を可能とする索引スキーマを用いる必要がある．

それに加えて，XML 情報検索では元の XML 文書数と比較して遥かに多い個数の部分文書が取り出されるために，出現する全ての索引語の重みを算出するためには膨大なコストを費やす必要がある．そこで，データ更新におけるコストを削減するために Element filter を設け，検索結果として提示される可能性の低い部分文書に含まれる索引語の重みの計算は行わない．これにより，部分文書のスコア計算に必要な計算回数が削減され，索引再構築時間の短縮に繋がると考える．

同様に，(2) において全ての索引語の重みを索引へ登録した場合には，索引更新に膨大な時間を費やす必要がある．そのため，全ての索引語を登録するのではなく，更新による効果が見込まれる索引語のみを更新すべく Term Filter の定義を行った．従って，索引再構築時間の短縮と作成される索引のサイズの削減効果が見込まれる．

本稿では，これらのフィルタの効果を評価し，フィルタを適用することで発生する処理コストが，適用することで生まれるコスト削減効果に見合うのかどうかを確認する．

2. XML 情報文書検索に関する基本事項

本節では，文書検索に対する XML 情報検索の位置づけについて述べ，更に部分文書の概要について説明する．

(注2)：http://en.wikipedia.org/wiki/Main_Page

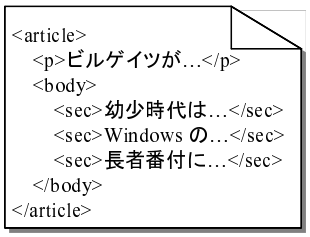

図 1　XML 文書

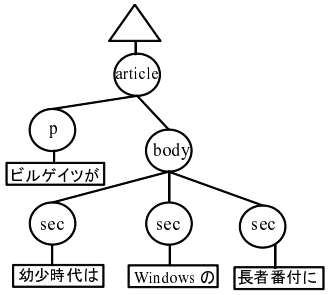

図 2　XML　木

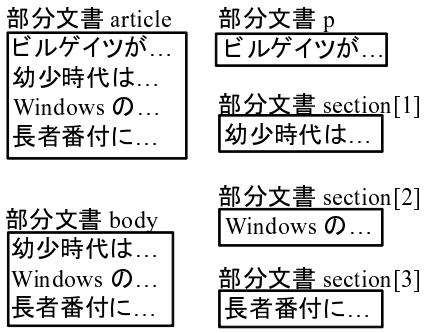

図 3　部 分 文 書

2.1 文書検索と XML 情報検索の比較

多くの Web 検索システムは，検索結果としてクエリに適合する文書のリストを提示する際に，スニペット [6] と呼ばれる 150 文字前後の要約文も併せて提示する場合が多い．スニペットはクエリキーワードとその周辺のテキストを抽出する技術であり，検索結果中からいずれの文書が閲覧するのに適切であるのかをシステム利用者が判断するための要約文である．しかしながら，スニペットは文章の文脈を考慮しないために理解不能なスニペットが生成される可能性がある．実際に，インターネット利用者に対して行った Web 検索の信頼性に関する調査 [14] では，多くの検索システム利用者がスニペットを利用していることと，その反面，必ずしも満足な結果が得られているわけではないということが報告されている．このことから，検索システム利用者はスニペットのみから情報要求を満たすことはできず，結局のところ文書検索システム利用者は文書を閲覧し，必要な箇所を自ら発見しなければならない．

これに対して，XML 情報検索における最大の関心は，クエリに適合する箇所，即ち部分文書を抽出し，それらをクエリに対する適合度の大きさで順位付けを行い提示することである．多くの Web 検索システムがクエリに適合する文書のリストを提示するのに対して，XML 検索システムはクエリに適合する部分そのもののリストを提示することができる．これにより，ユーザは文書中から情報要求を満たす部分を自ら発見する必要がなくなるために，情報検索を行う際のユーザの負担を大きく軽減することが可能である．

2.2 部 分 文 書

部分文書の概要について説明するために，図 1〜図 3 を用いて具体例を示す．

まず，図 1 は XML 文書の例であり，図 2 は XML 文書を木構造で表現した図である．構造化文書は一般的に木構造で表現することができ，文書構造の視認性の向上を目的として度々木構造で表現される．本稿においても同様に，XML 文書を木構造と見立てて議論を進めることとする．このとき，XML 文書のそれぞれの開始タグと終了タグが XML 木の各要素ノードのノード名に対応しており，タグの入れ子は要素ノードの親子関係によって表現されている．図 3 の各部分文書は，図 2 の XML 木の各要素ノード以下に含まれるテキストノードと対応する．つまり，文書全体を表す `article` ノードと対応する部分文書は子孫に存在するテキストノード全てを結合した文字列であり，`body` ノードと対応する部分文書はその子ノードである三つの `section` ノードに含まれるそれぞれのテキストノードを結合した文字列である．

なお，このとき，仮に情報要求を満たす内容が「幼少時代は…」と「Windows の…」,「長者番付に…」であった場合は，XML 情報検索においてはユーザに対して `body` ノード以下の部分を提示することが適切である．

3. 関 連 研 究

ここでは，XML 情報検索における二つの目標である高精度検索と高速検索に関して，それぞれ 3.1，3.2 にて関連研究の説明を行う．

3.1 高精度な XML 情報検索に関する関連研究

XML 情報検索に用いられる検索技術は，文書単位を検索粒度としたテキスト文書に対する情報検索技術を XML 情報検索用に拡張して用いられることが多い．例えば，ベクトル空間モデルである TF–IDF を XML の経路式を考慮し拡張させた TF–IPF [1] や，古典的確率モデルである Okapi's BM25 を構造化文書検索用に拡張させた BM25F [7]，XML 情報検索 (部分文書検索) 用に拡張させた BM25E [5] などである．

上記のような，クエリに含まれる各索引語の重みを利用して検索結果を発見するアプローチ以外として，クエリによって指定される構造情報を用いたスコアリング手法 [12] や，索引語の重みだけでなく各部分文書の持つ情報を考慮して検索結果として有用な検索結果である部分文書を特定するというアプローチ [15] も存在する．また，文献 [13] では予め検索対象から解として不適切な部分文書を省くことを目指している．解として不適切な部分文書とは，如何なるクエリに対しても解とはならない部分文書のことであると定義しており，これらの部分文書を検索対象から除外したとしても検索精度が悪化することはなく，更には検索対象を削減することでクエリ処理時間の短縮にも繋がると報告されている．

3.2 高速な XML 情報検索に関する関連研究

高速な XML 情報検索を実現すべく，1) クエリ処理時に読み込むデータ量を最小限に留めるためにデータベースへ格納される登録データの圧縮や削減，2) 高速なクエリ処理のための Top-k アルゴリズムの考案，などが取り組まれている．中でも，高速な XML 情報検索を目指す研究では，文献 [5], [7] をはじめとする索引語の重み付け手法を利用することで，高精度かつ高速な XML 情報検索の実現を目指しているものも存在する [9], [10]．索引語の重みは様々な統計量を用いた複雑な数式によって算出されるが，これはユーザからクエリが投げかけられる以前の事前処理段階で算出しておくことで，検索速度を低下させることなく高精度検索を実現することが可能である．

TopX [9] では，効率的な検索の実現を目的として，二種類の索引スキーマ，即ち，各部分文書に対する得点計算用の索引と構造の制約確認用の索引の提案，そして高速なクエリ処理を行うために構造化文書検索に特化した Top-k アルゴリズムの提案を行なっている．得点計算用索引ではタグと索引語のペアに対してリストが作成されており，リスト中の各エントリはノード識別子 (文書番号 + ノード番号) と，対応する索引語の重みやクエリ処理時に必要となるデータが格納されている．構造検索用索引は，各部分文書がクエリによって指定される構造を満たすのかどうかを確認する用途で用いられ，タグごとにリストが定義される．リスト中の各エントリは，タグを満たす部分文書のノード識別子が列挙されている．また，効率的にクエリ処理を行うために，Top-k アルゴリズムのうちの一つである Threshold Algorithm [2] (TA) の拡張手法を提案している．TA が文書検索に利用される場合，リスト中のデータは索引語の重みの降順にソートされる．TopX でも索引語の降順にソートを行うものの，同一の文書のデータを纏めて読み込んでおくことで，構造制約の確認を容易に行うことを可能としている．

これまで従来研究において提案された索引スキーマは何れも，静的なデータセットに対して高速にクエリ処理を行うことに焦点を当てたスキーマであるために，必ずしもデータ更新を行う際に適切であるとはいえない．データの更新が発生した場合には，新たに追加された索引語の重みを算出する必要が生まれる．このとき，文献 [4], [5] などの代表的なスコアリング手法で用いられる統計量は，1) 計算対象データから算出できる統計量 (局所的重み)，2) 文書集合全体から算出される統計量 (大域的重み)，3) パラメータや定数，の三種類に分類される．その際，1), 3) は予め算出する必要はないが，2) に関しては予め算出して格納しておかなければその都度文書集合全体を走査することとなり，高速な索引更新，ついては高速な情報検索は不可能である．従って，高精度検索を維持しつつ，高速なクエリ処理とデータ更新を目的とした索引スキーマを用いる必要がある．

4. 索引スキーマと二種類のフィルタ

2. と 3. を踏まえて，データの更新を考慮しつつ，高精度・高速な XML 情報検索を実現する上での課題と，その解決のために我々が文献 [16] にて行った取り組みについて述べる．我々のシステムでは，従来手法の検索システムで考慮している索引構築，クエリ処理に加えて，データの追加に対する処理の定義を行った．システムのフレームワークを図 4 に示す．

(1) 初期索引構築　まず，その時点までに蓄積された XML 文書全てを解析して初期索引の構築を行う．高速にクエリ処理を行うことだけを目的とするのではないために，システムの索引スキーマでは従来手法では考慮されていなかった，索引語の重み計算に必要な統計量の格納も行う．

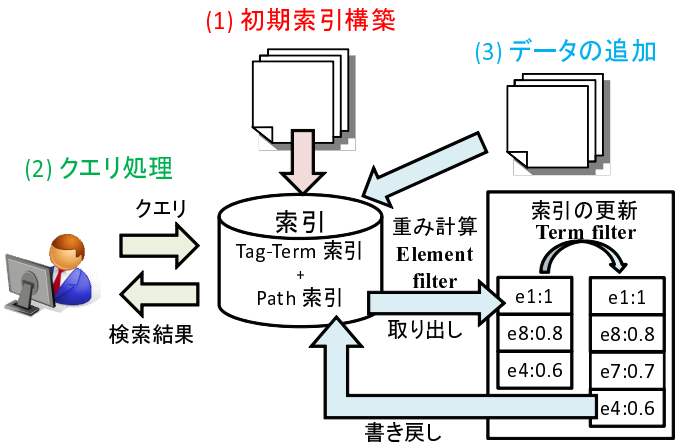

図 4　システムのフレームワーク

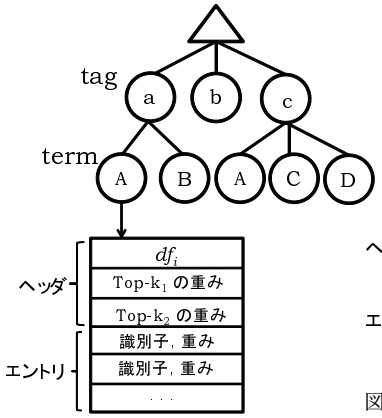

図 5　提案する Tag–Term 索引

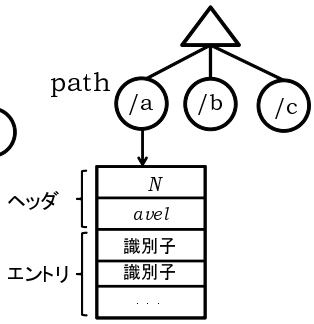

図 6　提案する Path 索引

(2) クエリ処理　続いて，クエリ処理のパートではユーザのクエリを用いて索引に問い合わせて，検索結果を取り出してユーザへ提示する．

(3) データの追加　最後に，データの追加パートでは，新たに追加されたデータに対しても検索対象として扱うことができるように索引語の重み計算と索引への登録を行う．

以降，上記の処理を行う上で必要な，索引スキーマの定義とデータ追記時の処理について述べる．その際，部分文書単位の検索の特性である，元データ量と比較した場合に起こるデータ量の増大に対応するため，更新に関わるコストを削減処理についても述べる．

4.1　索引スキーマの定義

ここでは，高速なデータ更新と高速なクエリ処理を両立させるための索引について説明する．我々のシステムでは TopX と同様に部分文書の得点計算用索引 (Tag–Term 索引) と，構造制約の確認用索引 (Path 索引) の二種類を構築する．索引に格納される各エントリのデータとしては，TopX と同様に Tag–Term 索引では識別子 (文書番号 + ノード番号) と索引語の重み及びクエリ処理時に必要となるデータであり，Path 索引では識別子である．索引語の重み算出には，高精度情報検索に適切であると言われるスコアリング手法である BM25E [5] を利用する．その際，ある部分文書 i における索引語 j の重み $w_{i,j}$ は以下の数式で算出される．

$$w_{i,j} = \frac{(k_1+1)tf_{i,j}}{k_1((1-b)+b\frac{el_i}{avel_i})+tf_{i,j}} \log \frac{N_i - df_{i,j} + 0.5}{df_{i,j} + 0.5} \quad (1)$$

ただし，$tf_{i,j}$ は部分文書 i に含まれる索引語 j の索引語頻度，$df_{i,j}$ は部分文書 i と同じ path 式で表される部分文書群中の索引語 j の部分文書頻度，N_i は同じ path 式で表される部分文書数，el_i は部分文書 i の部分文書長 (部分文書中に含まれる索引語数)，$avel_i$ は部分文書 i と同じ path 式を持つ部分文書群の平均部分文書長，k_1, b はパラメータであり，過去に執り行った実験の結果を踏まえて，それぞれ経験則的に 2.5, 0.85 を設定する．

ここで，新たなデータが追加された際の索引語の重み算出について議論を行う．既存の索引中の索引語の重みと同様に，追加された索引語の重みの計算する際には BM25E による重み計算を行うが，$tf_{i,j}$ や el_i などは新たなデータから算出される局所的重みであり，k_1, b といった各種パラメータや定数は固定の値を用いるために，それぞれ参照することは容易である．その一方で，$df_{i,j}$, N_i, $avel_i$ といった統計量は文書集合全体から得られる大域的重みであるために，予めデータベースへ格納しておかなければ即座に索引語の重みを算出することはできない(注3)．故に，これらの統計量を Tag–Term 索引と Path 索引に含まれるリストのヘッダとして登録する．これらを踏まえて，Tag–Term 索引と Path 索引の概略図をそれぞれ図 5，図 6 に示す．

4.2　データの追加と，更新コストの削減

新しいデータが追加された際の手順について述べる．まずは追加されたデータに含まれるそれぞれの索引語に対して重みの算出を行い，その後該当リストへの登録を行う．索引語の重み計算は 4.1 で定義した索引に含まれる大域的重みを利用することで索引語の重みを計算可能であるが，部分文書単位での情報検索では，元の XML 文書数と比較して遥かに多い個数の部分文書が取り出されるために，存在する全ての部分文書中の全ての索引語の重みを算出するためには膨大なコストを費やす必要がある．そこで，データ更新におけるコストを最小限に留めるために Element filter を設け，検索結果として提示される可能性の低い部分文書に含まれる索引語の重みの計算は行わない．

また，このとき，全ての索引語の重みを索引へ登録すると，同様に索引更新に膨大な時間を費やす必要があることが予想される．そのため，全ての索引語を登録するのではなく，更新による効果が見込まれる索引語のみを更新すべく Term filter を用いる．これにより，更新コストを最低限に抑えつつ，高精度・高速な XML 情報検索の実現を目指す．

4.2.1　Element filter

検索結果として提示することが不適切な部分文書や提示される可能性が低い部分文書に対して索引語の重み算出を行ったとしても，その効果は薄い．従って，データが追加された時点で

(注3)：本稿では索引語の重み付け手法として BM25E を利用するが，任意のスコアリング手法を利用することが可能である．ただし，その際は文書集合全体から得られる統計量を予め索引へ登録を行う必要がある．

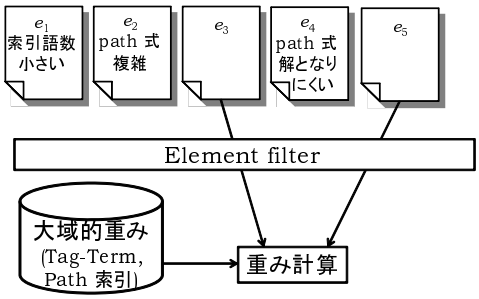

図 7　Element filter の動作

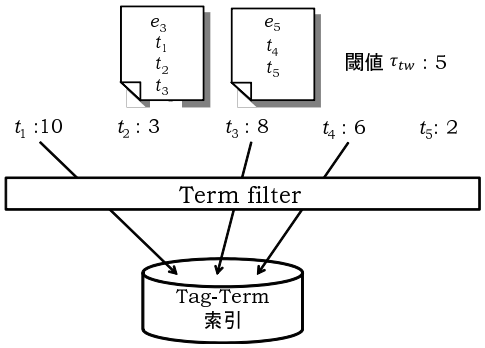

図 8　Term filter 動作

それらの部分文書を特定し，索引語の重み計算を行わないことで更新における計算コストを削減するというのが Element filter のアイデアである．解にならない部分文書の条件としては，文献 [13] や [15] において，報告されている

（1） 索引語数の小さな部分文書

（2） 極めて深い path 式を持つ部分文書

を考慮する．(1), (2) は過去の研究結果 [15] から部分文書長が 15 以下の部分文書は解として不適切である可能性が高いという傾向が判明しているため，5. の評価実験では Element filter におけるフィルタのうちの一つとして $\tau_{el} = 15$ を設定する．また，極めて深い path 式の深さの設定 τ_{depth} については 5.2 にて調査を行う．

Element filter の動作を図 7 に示す．データの追加に伴って五つの部分文書 e_1, e_2, e_3, e_4, e_5 が取り出されたとする．これらの部分文書に対して Element filter を適用した場合には，部分文書 e_1, e_2, e_4 はそれぞれ，索引語数が小さい，部分文書の持つ path 式が複雑，といった理由のためにその後の処理の対象から除外される．従って，部分文書 e_3, e_5 に含まれる索引語のみ重みの計算が行われる．

4.2.2　Term filter

Threshold Algorithm によるクエリ処理では，索引語の重みの降順にソートされた複数リストを読み込むことで高速なクエリ処理を行う．このとき，索引語の重みの小さなデータを登録しても読み取られる可能性が低く，登録しても効果が薄いことが見込まれるため，それらのデータの登録を行わないというのが Term filter のアイデアである．その際の分類は，Top-k 番目のデータの索引語の重みを閾値 τ_{tw} として利用する．これは，予め図 5 のヘッダへ Top-k 番目のデータの重みの値を登録しておくことで，リストの更新が発生する度にリストを走査して閾値を探し出す手間を回避する．

以上より，Term filter の具体的例を図 8 に示す．閾値 τ_{tw} の値は Tag-Term ごとに設定されるが，ここでは説明のの簡潔化のために全ての τ_{tw} が 5 であったとする．このとき，索引語 t_1 $(10 > \tau_{tw})$, t_3 $(8 > \tau_{tw})$, t_4 $(6 > \tau_{tw})$ の重みは閾値よりも大きな値であるために索引中のリストへ登録される．その一方で，t_2 $(3 > \tau_{tw})$ と t_5 $(2 > \tau_{tw})$ は閾値よりも小さな重みを持つために登録対象から外れる．

5. 評価実験

本節では，提案手法の有用性の確認のために行った評価実験について述べる．

5.1 実験準備

本稿で述べた索引スキーマを用いることで適切に動的な索引構築を行うことが可能であるかどうか，また，二種類のフィルタを適用することによって更新コストを削減することが可能であるかどうか，そして各フィルタを適用することによって検索精度を落とさずにコストを削減可能であるのかどうかの検証を行うため，索引構築時間と索引サイズ，クエリ処理時間と検索精度を計測する．

従来研究で行われている，全ての文書集合から索引を構築するアプローチを From scratch (*scratch*) アプローチとし，文書集合全体のうちのある一定の割合の文書集合から初期索引を構築して，その後に残りのデータを追加データとして用いて索引の動的な更新を行うアプローチを提案手法とする．それぞれのフィルタの性能を調査するために，フィルタを適用しないアプローチを No filter アプローチ (*no–filter*)，Element filter のみを用いたアプローチを Element filter アプローチ (*elem*)，Term filter のみを用いたアプローチを Term filter アプローチ (*term*)，両方のフィルタを適用したアプローチを Elem Term アプローチ (*elem–term*) と呼称する．なお，追加されたデータ中の索引語の中には，大域的重みが算出できない場合(注4) が起こる．このような場合には，擬似的な重みとして初期索引に含まれる全ての大域的重みの平均値を利用することとする．今回の実験においては，全体の 80% のデータから初期索引を作成して，残りの 20% のデータを追加データとして用いた．なお，動的な索引更新を行う上では大域的重みを予め索引に格納している必要があるため，それらのデータの索引更新時間や索引サイズは提案手法による実験結果に計上している．

評価実験には INEX 2008 Wikipedia collection [3] を使用した．このテストコレクションは三つの要素から構成されており，一つ目が約 67 万個の XML 文書から構成される INEX document colleciton，二つ目がクエリ集合である INEX topics,

(注4)：例えば BM25E では，新たに追加された部分文書の持つ path 式がこれまでに出現した部分文書の持つ path 式には存在しない場合に，大域的重みを算出することができない

表 1　各部分文書の持つ path 式の深さ

深さ	1	2	3	4	5	6	7	8	9	10	15
全体	.015	.040	.19	.44	.69	.88	.96	.99	.99	.99	1.00
上位	.24	.34	.69	.89	.97	.99	1.00	1.00	1.00	1.00	1.00

そして三つ目がクエリに対する解答が定義されている INEX relevant assessments である．なお，クエリ集合である INEX topics に含まれるクエリは二種類存在し，一方はキーワードのみを指定する content only (CO) クエリであり，もう一方はキーワードと構造を指定する content and structure (CAS) クエリである．次節以降の評価実験では構造とキーワードの両方を指定するクエリのみで構成されるクエリ計 17 個を用いた．また，今回用いたテストコレクションの評価ツールでは，各クエリ対して，最大で 1,500 件のデータを提示することが可能である．これを踏まえて，Term filter で用いる索引語の重みは Top-1500 における索引語の重みを τ_{tw} として扱う．

実験環境は，CPU: Intel Xeon X7560 (2.3GHz)，メモリ: 1TB (DDR3)，OS: Oracle Enterprise Linux release 5.5，ディスク容量: 4.5TB (7200 rpm) である．また，索引構築においては Generalized Search Tree (注5) (GiST) を利用し，GNU G++ での実装を行った．

以降，Element filter の閾値設定のための予備実験の後に，各アプローチごとに索引の構築時間と構築される索引のサイズの計測を行う．また，フィルタを適用し，動的に更新して構築された索引に対してクエリ処理を行った場合に，検索速度や検索精度に対してどのような影響を及ぼすのか調査を行う．

5.2　Element filter の閾値設定のための予備実験

4.2.1 にて説明した Element filter では索引語が小さすぎる部分文書や，深すぎる path 式を持つ部分文書を索引語の計算対象から省くための処理を行う．ここでは，深すぎる path 式の深さの閾値を設定するために行った予備実験について説明する．

各部分文書の持つ path 式の深さを計測し，各 path 式の深さ以下の部分文書が部分文書全体のうちのどの程度の割合に該当するのか累積割合を集計した結果を表 1 に纏めた．更に，高精度検索を目指すために過去に行った手法 [15] を用いて得た検索結果のうちの上位の結果に含まれる部分文書の持つ path 式の深さにおいても同一の計測を行った．

表 1 より，検索結果上位の部分文書の持つ path 式の深さと部分文書全体の持つ path 式の深さでは傾向の違いが見て取れる．このような傾向の違いを利用して，検索結果上位の部分文書を最大限拾いつつ，それ以外の不要な部分文書を排除するための設定として，閾値 $\tau_{depth} = 6$ とする．これにより，深さが 6 以上の path 式を持つ部分文書は更新対象から除外される．

なお，索引語長に関する閾値は 4.2.1 で述べた通り $\tau_{el} = 15$ を設定し，部分文書長 15 以下の部分文書は計算対象から除くこととする．

(注5)：http://gist.cs.berkeley.edu/

表 2　索引構築時間 (hour)

	scratch	*no–filter*	*elem*	*term*	*elem–term*
real	7.43(1.00)	8.17(1.10)	4.67(.629)	7.17(.966)	3.94(.530)
user	.611(1.00)	1.22(1.99)	.921(1.51)	.993(1.63)	.734(1.20)
sys	.546(1.00)	.656(1.20)	.434(.796)	.533(.978)	.316(.579)

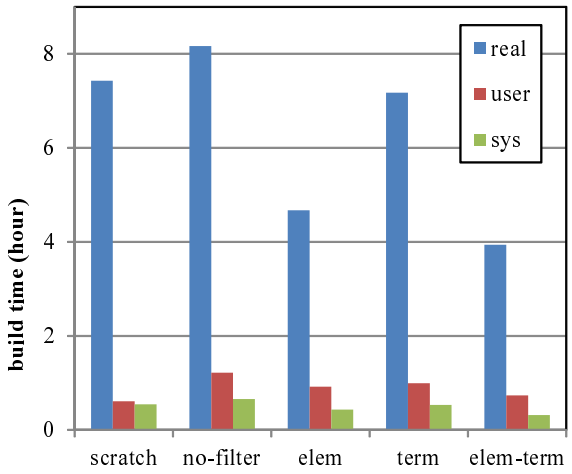

図 9　索引構築時間 (hour)

表 3　索引のディスク容量 (GB)

	scratch	*no–filter*	*elem*	*term*	*elem–term*
索引サイズ	14.6(1.00)	17.0(1.17)	11.9(.819)	9.01(.619)	5.45(.380)

5.3　索引の構築に関する実験

先ほど述べた索引構築アプローチ，即ち，*scratch*，*no–filter*，*elem*，*term*，*elem–term* に対して，それぞれ索引を構築する際にどの程度の時間を費やす必要があるのか調査を行った．動的更新に要した時間を明確に表すために，初期索引の構築に要した時間を差し引いた構築時間を表す．表 2，図 9 は各アプローチによって索引を構築した際の計測時間に加えて，*scratch* に要する時間を 1.00 とした場合の各アプローチに要する時間との比率を括弧付きの数字で示す．

まず，全てのアプローチにおいてユーザ時間とシステム時間の和と比較して実行時間が長くなっていることより，I/O コストがボトルネックになっているということが分かる．このことからも，更新対象のデータを選別して索引の更新を削減することは妥当であると考えられる．また，*scratch* と比較して *no–filter* で索引構築時間が大きくなっていることより，動的に索引語の重みを算出することで処理時間が増加したことを示す．それに対して *elem* では 37%，*term* では 3%，*elem–term* では 47% の構築時間の短縮が実現した．これにより，フィルタを適用することで得られる効果が，フィルタを適用することで発生するコストよりも大きいということが判明した．

次に，構築された索引のサイズを計測した結果を表 3 と図 10 に示す．索引構築時間と同様，フィルタ適用の効果を確認するために追加データ分の索引サイズを記載することとする．*no–filter* が *scratch* よりも大きくなっているのは大域的重みの

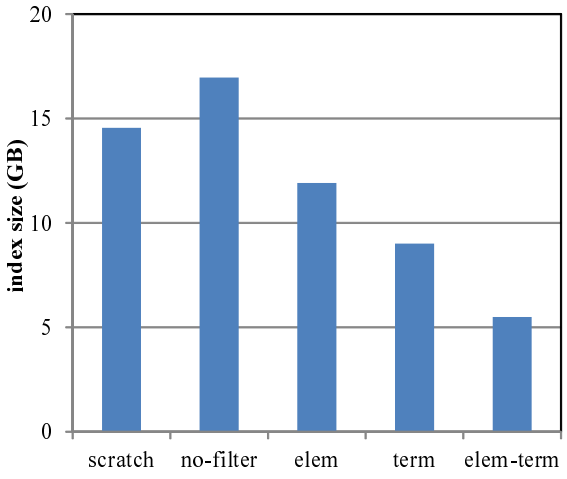

図 10　索引のディスク容量 (GB)

増加分である．*elem* では 18%，*term* では 38%，*elem–term* では 62% のディスクサイズの削減が実現した．索引構築時間に続き索引サイズにおいても，フィルタを適用することで得られる効果が，フィルタを適用することで発生するコストよりも大きいということが判明した．

以上の索引構築における構築時間と索引サイズの減少より，提案フィルタを適用することで更新コストを削減することが可能であるということが明らかになった．その際，Element filter は索引更新時間，Term filter は索引サイズの削減により効果を示した．

5.4　クエリ処理に関する実験

続いて，クエリ処理に要する時間と，その際の検索精度を計測した．INEX project(注6) では再現率が 1% における検索精度である iP[.01] を公式尺度とし，システムの検索精度評価に用いている．また，INEX project で利用されるその他の評価尺度としては Mean Average interpolated Precision (MAiP) が存在する．MAiP は複数の再現率点における精度の平均から求められ，INEX project においては 101 個の再現率から計算される．本稿においては上記の二種類の評価尺度を用いて検索精度の計測を行った．それぞれの結果を表 4，図 11，図 12 に示す．

クエリ処理時間については，全てのアプローチにおいて大きな差は生じなかったが，索引サイズが小さなアプローチではその分クエリ処理時間が短くなる傾向が見られた．

その一方で，*scratch* と比べて *no–filter* の iP[.01] における検索精度が低下するということが明らかになった．その原因として，提案手法では大域的重みが適切に算出できていないということが挙げられる．*scratch* では文書集合全体から大域的重みを算出しているのに対して，*no–filter* では全体の 80% のデータから大域的重みを算出している．それによって検索精度への悪影響が生じたということである．これに伴って，*elem*, *term*, *elem–term* においても検索精度が低下した．しかしながら，*elem* の検索精度は *no–filter* から低下しなかったことから，

(注6)：https://inex.mmci.uni-saarland.de/

表 4　クエリ処理時間と検索精度

	scratch	*no–filter*	*elem*	*term*	*elem–term*
real (s)	10.0	10.0	9.94	8.22	8.18
user (s)	9.60	9.65	9.51	7.89	7.85
sys (s)	.426	.405	.431	.328	.332
iP[.01]	.564	.525	.531	.482	.482
MAiP	.156	.164	.165	.147	.147

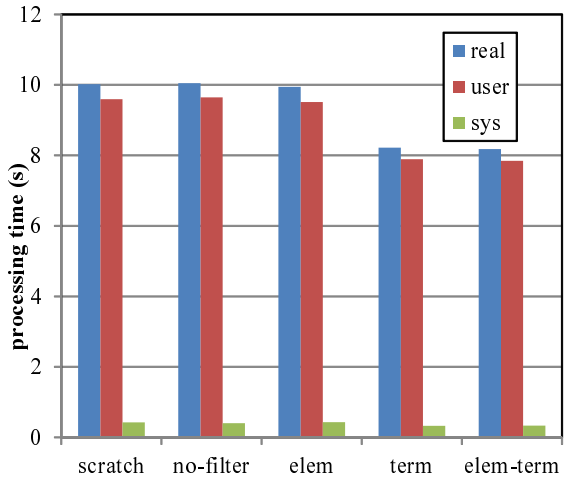

図 11　クエリ処理時間

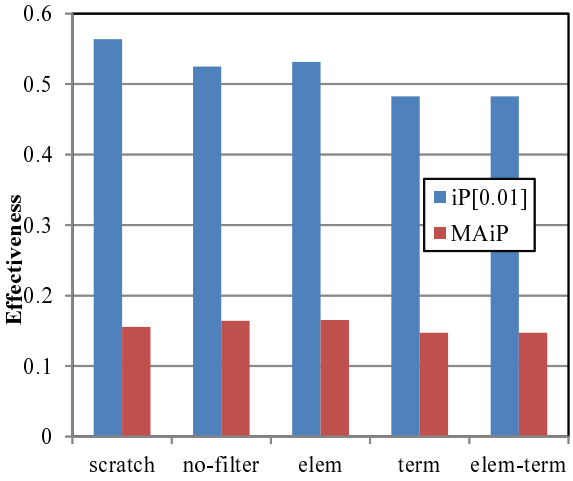

図 12　検索精度の比較

Element filter そのものが検索精度へ悪影響を及ぼすことはないという結果が得られた．つまり，Element filter で計算対象外と判定された部分文書中には解となる部分文書がなかったということである．

また，その一方で，*term*, *elem–term* では *no–filter* と比較して iP[.01] の検索精度が低下するという結果を示した．その原因として，Term filter によって小さな索引語の重みが除外されたために一部の部分文書に対して正確な得点を算出できず，そのことが検索結果のランキングに影響を及ぼしたと考えられる．従って，重みが高い索引語のみを登録することは必ずしも適切とは言い切れないといえる．検索精度を低下させないための対策としては，閾値として利用する k 番目の値を大きくするなど，Term filter の閾値の設定方法を再考する必要がある．

表 5　初期索引用データの割合とその検索精度への影響

	scratch	*no–filter*			
初期索引割合 (%)	100	50	80	90	95
iP[.01]	.564	.486	.525	.527	.564

5.5　大域的重みに関する追加実験

5.4 による実験の結果，初期索引に用いるデータ量と動的更新に用いるデータ量の比率が 8:2 の場合に適切に大域的重みを算出することができていないということが判明した．従って，初期索引の割合と追加データの割合を段階的に変化させ，その検索精度への影響を計測した．表 5 の通り，初期索引に用いるデータ量の変動に伴って検索精度に変化が見られ，95% のデータから大域的重みを算出した場合には検索精度の低下が見られなくなった．一定の割合のデータが追加される度に，大域的重み及び索引中の索引語重みを再計算することで検索精度を維持することが可能であると考えられるが，頻繁に重みの再計算を行うことは実用性に問題が生じる．従って，適切に大域的重みを算出する手法を考案する必要がある．

6. おわりに

本稿では，データの追加 · 編集に対応し，更新を考慮した XML 情報検索システムのフレームワークに関する議論を行った．高精度 · 高速検索を維持しつつ高速な索引更新を満たすために過去に提案した索引スキーマの定義と，索引更新コストを最低限に留めるための二種類のフィルタを適用した際の効果を確認した．

評価実験の結果，初期索引の割合が十分に確保されていない場合には適切に大域的重みを算出することができず，その結果索引語の重み付けに悪影響を及ぼすということが判明した．その一方で，Element filter を適用した場合には検索精度を維持しつつ，索引構築時間を 37%，索引サイズを 18% 削減させることに成功した．また，Term filter を適用した場合に一部の部分文書に対しては正確な得点を算出できずに検索精度が低下するという結果が得られた．検索精度が低下したものの，索引更新時間や索引サイズの削減は重要な要素であるため，今後は検索精度を低下させずにコスト削減可能なフィルタを設定することを目指す．

また，今回の提案ではデータ編集の際の手順として，該当データの全削除と新規データの追加を前提としているが，データの編集により適切な更新方法を考慮する必要がある．これは，データの編集が行われた影響範囲のデータのみを更新することで，全てのデータの更新を行う必要がなく，より高速な索引構築が可能であると考えられるためである．また，本稿では考慮していないキーワードのみを指定した問合せへの対応なども必要であると考えられる．キーワード検索に対しては複数の Tag–Term リストを参照しなければならないために，各リストの配置や構造を工夫することで効率的なクエリ処理を行うことが可能であると考えられる．

謝辞　本研究の一部は，科研費補助金基盤研究 (A)(課題番号:22240005) ならびに基盤研究 (C)(課題番号:23500121)，日本学術振興会科学研究費補助金 若手研究 (B) (課題番号: 22700248) の支援による．ここに記して謝意を表す．

文　献

[1] Torsten Grabs and Hans-Jörg Schek. PowerDB-XML: A Platform for Data-Centric and Document-Centric XML Processing. In *Proceedings of the First International XML Database Symposium*, volume 2824 of *Lecture Notes on Computer Science*, pages 100–117. Springer Berlin, September 2003.

[2] Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. A Survey of Top-k Query Processing Techniques in Relational Database Systems,. *ACM Computing Surveys (CSUR)*, 40:1–58, 2008.

[3] Jaap Kamps, Shlomo Geva, Andrew Trotman, Alan Woodley, and Marijn Koolen. Overview of the INEX 2008 Ad Hoc Track. In *Advances in Focused Retrieval*, volume 5631 of *Lecture Notes on Computer Science*, pages 1–28. Springer Berlin, September 2008.

[4] Fang Liu, Clement Yu, Weiyi Meng, and Abdur Chowdhury. Effective Keyword Search in Relational Databases. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, pages 563–574, 2006.

[5] Wei Liu, Stephen Robertson, and Andrew Macfarlane. Field-Weighted XML Retrieval Based on BM25. In *Advances in XML Information Retrieval and Evaluation*, volume 3977 of *Lecture Notes on Computer Science*, pages 161–171. Springer Berlin, June 2006.

[6] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schutze. *Introduction to Information Retrieval*, pages 157–159. Cambridge University Press, July 2008.

[7] Stephen Robertson, Hugo Zaragoza, and Michael Taylor. Simple BM25 Extension to Multiple Weighted Fields. In *Proceedings of the 13 ACM International Conference on Information and Knowledge Management*, pages 42–49, November 2004.

[8] Toshiyuki Shimizu and Masatoshi Yoshikawa. Full-Text and Structural Indexing of XML Documents on B+ Tree. *IEICE Transactions on Information and Systems*, E89-D(1):237–247, 2006.

[9] Martin Theobald, Holger Bast, Debapriyo Majumdar, Ralf Schenkel, and Gerhard Weikum. TopX: Efficient and Versatile Top-k Query Processing for Semistructured Data. *The VLDB Journal*, 17(1):81–115, 2008.

[10] Andrew Trotman, Xiang-Fei Jia, and Shlomo Geva. Fast and Effective Focused Retrieval. In *INEX'09 Proceedings of the Focused retrieval and evaluation*, pages 229–241, December 2009.

[11] 波多野 賢治, シーハム アメルヤヒア, and ディベッシュ スリバスタバ. XML 情報検索における構造問合せを利用した部分文書スコアリング. In **電子情報通信学会技術研究報告**, number 254, pages 13–18, October 2007. DE2007-117.

[12] 波多野 賢治, 絹谷 弘子, 吉川 正俊, and 植村 俊亮. XML 文書検索システムにおける文書内容の統計量を利用した検索対象部分文書の決定. **電子情報通信学会論文誌**, J89-D(3):422–431, March 2006.

[13] 中村 聡史. 情報検索に対する信頼性調査および結果. **人工知能学会誌**, 23(6):767–774, November 2008.

[14] 欅 惇志, 波多野 賢治, and 宮崎 純. 有益な検索結果提示のための部分文書再構成手法の提案. **情報処理学会論文誌：データベース**, 4(1):1–13, March 2011.

[15] 欅 惇志, 宮崎 純, and 波多野 賢治. XML 情報検索のための動的な索引管理手法の一提案. In **情報処理学会研究報告**, volume 2011-DBS-153, pages 1–8, November 2010.