社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2008－DBS－146 (22)
2008／9／22

Webコンテンツ一貫性管理のための制約発見支援

高橋 公海† 森嶋 厚行† 松本亜季子†† 杉本 重雄† 北川 博之†††

† 筑波大学大学院図書館情報メディア研究科 〒305–8550 茨城県つくば市春日 1–2
†† 筑波大学図書館情報専門学群 〒305–8550 茨城県つくば市春日 1–2
††† 筑波大学大学院システム情報工学研究科 〒305–8550 茨城県つくば市天王台 1–1–1
E-mail: †{mtaka,mori,sugimoto}@slis.tsukuba.ac.jp, ††s0512261@ipe.tsukuba.ac.jp,
†††kitagawa@cs.tsukuba.ac.jp

あらまし 近年，Webサイトを通じた情報発信が広く普及し，管理しなくてはならないコンテンツの量も増加している．それに伴い，コンテンツの一貫性の維持が困難となっている．我々はこれまで，コンテンツ制約を入力とすることによって，DB等をバックエンドに持たない非定型Webコンテンツの一貫性管理の支援を行うシステムの研究開発を行ってきた．本稿では，既存のWebコンテンツからコンテンツ制約を発見するための手法について提案する．
キーワード Webサイト管理, コンテンツ一貫性

Support of Constraint Discovery for the Integrity Management of Web Contents

Masami TAKAHASHI†, Atsuyuki MORISHIMA†, Akiko MATSUMOTO††, Shigeo SUGIMOTO†, and Hiroyuki KITAGAWA†††

† Grad. Sch. of Lib., Info. and Media Studies, U. of Tsukuba. Tsukuba, Ibaraki, 305–8550 Japan
†† Sch. of Lib., and Info. Sci., U. of Tsukuba. 1–2 Kasuga, Tsukuba, Ibaraki, 305–8550 Japan
††† Grad. Sch. of Sys. and Info. Eng., U. of Tsukuba. 1–1–1 Tennohdai, Tsukuba, Ibaraki, 305–8550 Japan
E-mail: †{mtaka,mori,sugimoto}@slis.tsukuba.ac.jp, ††s0512261@ipe.tsukuba.ac.jp,
†††kitagawa@cs.tsukuba.ac.jp

Abstract Today, publishing information from Web sites are common, but as the size of the Web contents to be managed increases, it is becoming difficult to maintain their content integrity. So far, we have been developping a system to support the content integrity of Web sites without backend DBs, which tries to satisfy given constraints on the contents. This paper proposes a method to automatically generate such constraints from existing Web content.
Key words Web-site Management, Content Integrity

1. はじめに

近年，Webサイトを通じた情報発信が広く普及し，コンテンツの量も増加している．Webの特徴の一つは分散管理であるが，一方で，その特徴がコンテンツ一貫性の維持を困難とする一因となっている．例えば，大学の研究室のWebサイトでは，各構成員が自分のホームページ上で研究論文リストを公開することが多いが，これらの論文リストの間には矛盾が多く見られる等といった問題がある．一般に，コンテンツの一貫性を管理するためには，バックエンドにDBシステムを配置し，DBに格納されているデータからWebページを作成するアプローチがとられる．しかし，筑波大学のWebサイトを対象とした我々の予備調査[2]では，バックエンドDB等をもたずに手作業で管理されているWebサイトも，数多く存在する．また，ある学部のWebサイトとその学部に属する学科のWebサイトが同じ内容を含むにも関わらず，管理者が別であるために統一的に管理されていないということもよくあることである[1]．このような場合，例えばある学科HPの入試説明会情報に変更があった場合，学群のHPでも同様の変更を行わなければならないが，確実な変更を保証することは多大な努力を要するものである．

この問題に対し，我々は，DB等をバックエンドに持たないWebコンテンツの一貫性管理の支援を目的としたシステムの研究開発を行ってきた[2], [4]〜[6]．このシステムは，Webコンテンツ間に成立すべき制約 (例えば，研究室Webサイトにおける学生の論文リストは，研究室の論文リストのサブセットである，等) を与えることにより，既存のWebコンテンツに対して

後付けでコンテンツの一貫性管理が行えるようにするものである．本システムを利用すれば，既存の Web サイトを，DB をバックエンドにした Web サイトに再構築しなくとも，このような一貫性管理が可能になる．

しかし，既存の Web コンテンツに対する制約は人手で与えなくてはならないため，膨大な Web コンテンツを対象とした場合には既存 Web コンテンツからの制約発見支援が必須である．**本論文で扱う問題と関連研究．**本論文では，Web コンテンツ間の制約を，Web ページの HTML や XML 要素間 (以下 Web ページ要素) の**包含従属性** (inclusion dependency) [3] に限定し，システムを用いてその発見を支援するための手法を提案する．例えば，「学生の論文リストを表すコンテンツ X が研究室の論文リスト Y のサブセットでなければならない」という制約を $X \subseteqq_{IND} Y$ と記述する．この制約の発見を支援するために，既存の Web コンテンツにおいて $X \subseteqq Y$ の包含関係が存在するか否かを決定するのが本論文で扱う問題である．包含関係の存在は包含従属性を示すための根拠となる．

情報統合の分野において，包含関係の発見に関する研究は行われてきた. [7] では，リレーション (群) のインスタンスが与えられたとき，これらのリレーションの属性間に包含関係が成立するかの判定を行う効率よいアルゴリズムを提案している．そこでは，与えられたリレーションインスタンスに対して，属性 X の値の集合とと属性 Y の値の集合に包含関係が存在するとき，$X \subseteqq Y$ を出力する (図 1(上))．

それに対し，我々の問題では次の 2 点が異なる．

（1） Web ページのコンテンツには間違いも多く含まれるため，厳密な包含関係の判定ではうまくいかない．すなわち，一文字異なるだけで包含関係の可能性を除去してしまうと，包含従属性の発見率が低下する．

（2） Web ページ要素はリレーション属性と異なり階層構造を成している (図 1(下))．したがって，4 章で詳述するように，出力される包含関係に重要なものとそうでないものが含まれる．例えば，$X \subseteqq Y$ が得られたとき，階層構造での Y の上位要素 Z(図 1(下)) に関して $X \subseteqq Z$ が得られることは自明であり，制約としての重要性は低いと考えられる．

そこで，本研究では，(1) に対しては**許容率**という概念を導入し，どれだけの許容率でどの包含関係が存在するかをもれなく計算することを提案する．また，(2) に対しては，スコア付けルールによって，より重要と考えられる包含関係に上位スコアをつけるような手法を提案する．

これらを実現するために，本手法の出力は単なる $X \subseteqq Y$ ではなく，組 $(X \subseteqq Y, p, s)$ の集合とする．ここで X, Y は Web ページ要素，p は許容率 $(0 \leqq p \leqq 1)$，s は重要度スコア $(0 \leqq s \leqq 0)$ である．ここで，X に含まれるデータのうち割合 p を除去すれば $X \subseteqq Y$ が成立するとき，$X \subseteqq Y$ は**許容率** p で**成立**する，と呼ぶ．例えば，図 2 の Web ページ要素の組を考える．ここで，要素 A の a,b,c,d は要素 B に含まれているため，要素 A が a,b,c,d のみであれば $A \subseteqq B$ と判定できる．しかし，要素 A に x が追加された場合には，要素 B に x が含まれていないため，厳密な判定では包含関係は存在しない．しかし，1/5 個のデータを除去すれば $A \subseteqq B$ が成立するため，これは許容率 0.2 で成立する．後述する方法で重要度スコアが 3 と与えられたと仮定すると，本手法では $(A \subseteqq B, 0.2, 3)$ を出力する．

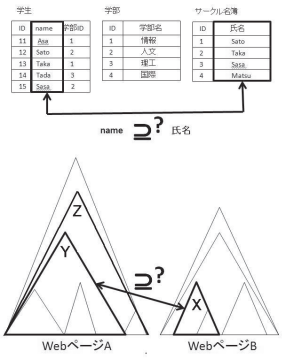

図 1　関連研究で扱う問題 (上)，我々の扱う問題 (下)

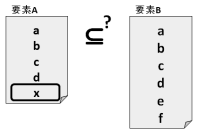

図 2　許容率導入例

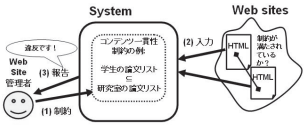

図 3　コンテンツ一貫性制約を用いた Web サイト管理

本論文の構成は次の通りである．2 章で，我々が開発中の Web サイトコンテンツ一貫性管理システムについて説明する．3 章で，許容率を導入した包含関係の発見アルゴリズムを提案する．まず，最も単純なアルゴリズムを説明し，次に I/O コストを削減するために複数の Web ページ要素を並列に処理する single pass アルゴリズムを提案する．4 章で，ルールを用いた重要度スコアの計算方法を提案する．5 章で実験結果を説明する．6 章はまとめと今後の課題である．

2. コンテンツ一貫性制約を用いた Web サイト管理手法

図 3 は我々が提案している明示的な一貫性制約を用いた Web コンテンツ管理システム [4] [5] の概要である．以下にその利用手順を述べる．(1) まず，利用者がコンテンツ一貫性制約を登録する．(2) 制約が登録されると，システムは定期的もしくは Web サイトの更新が行われた際などに Web サイトのチェックを行い，先に発見しておいた制約と照らし合わせて，制約が破られていないかどうか調べる．(3) その際，もし制約違反を発見したら，Web サイト管理者に報告もしくは自動修正を行う．

本システムを利用するためには，Web コンテンツに関する制

約をあらかじめ与える必要がある．しかし，Web コンテンツを構成するページ数が増えると，手作業で一つずつ発見することは現実的ではない．そこで，その発見支援のための仕組みが重要となる．本論文は，その制約の一つである包含従属性の発見支援に関するものである．

3. Web コンテンツ間の包含従属性発見支援

提案手法では，Web コンテンツに関する包含従属性の発見を支援するために，既存の Web サイトに含まれる全ての Web ページに含まれる全ての要素の組合せに対して，どれだけの許容率で包含関係が成立するか計算する．本章では重要度スコアの計算については扱わない．これについては次の章で説明する．

本手法では，Web ページのコンテンツが，Web ページ要素の階層構造になっていると仮定する (実際の Web ページそのものでなく，ラッピング結果でもよい)．また，Web ページ要素 x, y が表す集合 X,Y に次の関係が成立すると仮定する．

Web ページ要素が表す集合が満たすべき性質: x が y の下位要素であれば，$X \subseteqq Y$ である．

この性質を満たすような集合としては，各 Web ページ要素に含まれる文字列を N-gram 等で単純に分割したものや，形態素解析で分割したものが考えられる．提案手法はその集合の作成方法とは独立しているため，この性質を満たしていればいずれも適用可能である．

以降で提案するアルゴリズムは厳密な包含関係を発見するための [7] でのアルゴリズムを拡張し，許容率を導入したものである．いずれも，Web ページ要素 dep と ref に対応する集合 $depValues$ と $refValues$ に対して，$depValues \subseteqq refValues$ かどれだけの許容率で成立するかを判定する．最初に説明する Brute Force アルゴリズムは，一組の $(depValues, refValues)$ が与えられたときに，この判定を行うアルゴリズムである．このアルゴリズムの問題は，Web 要素の個数が n であるとき，${}_nC_2$ 回の実行 (とディスクスキャン) が必要であることである．二番目に説明する single pass アルゴリズムは，一度に複数の $depValues_i \in D$ と $refValues_j \in R$ が与えられた時，これらの全ての組合せ $depValues_i \in refValues_j$ がどれだけの許容率で成立するか，いずれの Web 要素も一度のディスクスキャンで計算するものである．

3.1 Brute Force アルゴリズム

アルゴリズムを図 4 に示す．これは，二つの集合 $depValues$ と $refValues$ が与えられたとき，包含関係 $depValues \subseteqq refValues$ が成立するかどうか判定する．

具体的には，まず図 4 の 1 行目で，$depValues$ 中の値のうち $refValues$ に含まれないものがあるたびに 1 ずつ減算するカウンタ C の初期値を計算する．$C = 0$ になるということは，$depValues$ のいずれの値も $refValues$ に含まれていなかったということである．それ以降では，$depValues$ に含まれる各値が $refValues$ に含まれているかどうかを順に判定していき，含まれていない値を見つけるたびに C を 1 つ減じる．そして，C が 0 になれば，許容率 $p = 1$ で $depValues \subseteqq refValues$ が成立するため，1 を出力する (10 行目)．

```
Input: refValues, depValues
Output: 許容率 for depValues ⊆ refValues
 1  C = |depValues|; // 含まれない値があるたびにカウントダウン
 2  while depValues has next value do
 3      currentDep := depValues.next();
 4      if refValues is empty then return 1-(C-a/depValues.size);
 5      while true do
 6          currentRef := refValues.next() ;
 7          if currentDep = currentRef then break;
 8          else if currentDep < currentRef then
 9              C := C - 1;
10              if C == 0 then return 1;
11              if depValues has next then
12                  currentDep := depValues.next();
13                  currentRef := refValues.prev();
14              else break;
15          else if refValues has no next value then
                return 1-(C-a/depValues.size);
16  return 1-(C/depValues.size);
```

図 4　Brute Force アルゴリズム

テストが終わった段階 (16 行目) で $C > 0$ の場合には，許容率 $p = 1 - (C/depValues.size)$ を返す．$depValue$ が最後の値までテストされていないにも関わらず，$refValue$ の次の値が無くなってしまった場合 (4 行目，15 行目) には，許容率 $p = 1 - (C - a/depValues.size)$ を返す．ここで，a は $depValue$ の残りの要素数である．

もし $depValue$ 中の現在の値が $refValue$ 中の現在の値と等しい (7 行目) ならば，同じものが含まれているため，$depValues$ と $refValues$ を次の値に進める．もし $depValue$ 中の現在の値が $refValue$ 中の現在の値よりも小さい (8 行目) ならば，その値は $refValue$ 中に含まれないと判定できる．このようにして，$depValues$ の値が $refValues$ に含まれるかの判定を行う．

3.2 Single-Pass アルゴリズム

Single-Pass アルゴリズムは，Brute Force アルゴリズムと同じ作業を行うが，一つの組ごとではなく，複数の組合せの処理を同時に行う．具体的には，$D = \{depValues_1, \ldots, depValues_m\}$, $R = \{refValues_1, \ldots, refValues_n\}$ の時，包含関係 $depValues_i \in refValues_j$ がどれだけの許容率で成立するかの判定を並行して行う．

図 5 は本アルゴリムの考え方を図示したものである．具体的には，各 $depValues_i$ と $refValues_j$ を処理するためのオブジェクトを生成し，これらが値をやり取りしながらカーソルを進めていく．このとき，あらかじめ $depValues_i$ と $refValues_j$ 中の値はソートしておく．このアルゴリズムでは，集合中の値は 1 度だけ読まれ，包含関係を満たす可能性のある全ての候補のペアは並行してテストされるため，I/O の量を最小化することができる．各集合はオブジェクトによって管理される．

基本的なアイデアは次の通りである．すなわち，各 dep オブジェクトは，現在指しているカーソルの値が関係する全ての ref オブジェクトの値との比較が終わった時に初めて，自身のカーソルを次に進める．同様に，各 ref オブジェクトは，現在指しているカーソルの値が関係する全ての dep オブジェクトの値との比較が終わったときに初めて，自身のカーソルを次に進める．

実際の動作は，各 dep オブジェクトが主体となって行う．すなわち，dep オブジェクトのカーソルを移動する決定は dep オ

ブジェクト自身が行う．ref オブジェクトのカーソルを移動する決定は，後述する monitor オブジェクトが dep オブジェクトからの ref の値の要求状況を見て判断する．

図 6 は，各 dep オブジェクトが実行するアルゴリズムの一部である．その dep オブジェクトのカーソルが現在指す値と，ref オブジェクトの現在の値を入力として比較し，dep のカーソルを動かすかどうかを判定する部分である．また，条件がそろえば，包含関係と許容率の組を出力する．

dep オブジェクトが ref オブジェクトの値を要求する (カーソルを動かして欲しい) ときには，currentWaiting および nextWaiting というキューにその ref オブジェクトの参照を代入することによって行う．

Monitor オブジェクトは，そのキューの内容をみて，ref オブジェクトのカーソルを進める．図 7 にそのコードを示す．具体的には，ある ref オブジェクトと関係する全ての dep がその ref オブジェクトに要求を出したとき，その ref オブジェクトのカーソルを進め，次の値をそれらの dep オブジェクトに渡す．

各 dep オブジェクトは ref の値が渡されたら図 8 のアルゴリズムを起動する．このアルゴリズムは現在のキューの状態に応じて適切に図 6 のプログラムを呼び出すものである．

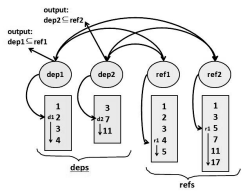

図 5　Single-Pass アルゴリズムの概要

```
Input: referencedObject, referencedValue,dependentValue
Output: satisfied IND or null,
 1  if dependentValue = referencedValue then
 2      if ∋ next dependent value then
 3          if ∋ referencedObject.wantNextValue(this) then
 4              nextWaiting :=
                nextWaiting ∪ referencedObject;
 5          else
 6              return 1-(C-a/depValues.size);
 7          else
 8              C := 1-(C-a/depValues.size);
 9              return this ⊆ referencedObject, C;
10  else if dependentValue > referencedValue then
11      if referencedObject.wantNextValue(this) then
12          currentWaiting :=
                currentWaiting ∪ referencedObject;
13      else
14              return 1-(C-a/depValues.size);
15      else
16          C = C -1;
17          if(C < 0){
18              C := 1; return C;
19          }else{
20              if ∋ next dependent value then
21                      next :=
                next ∪ referencedObject;
22              else
23                  C := 1-(C-a/depValues.size);
24                  return this ⊆ referencedObject, C;
25              }
26  return null;
```

図 6　dep オブジェクトでの値の比較とカーソル移動

```
 1  //モニタが値を渡す Ref を入れておくキュー
 2  Queue monitor_queue;
 3
 4  while(true){
 5      //終了判定
 6      for each dep in D{
 7          if(dep.refobjectlist.isEmpty) D.dep.remove;
 8      }
 9      for each ref in R{
10          if(ref.depobjectlist.isEmpty) R.ref.remove;
11      }
12      if(D.isEmpty || R.isEmpty)     break;
13
14      //キューに追加
15      for each ref in R{
16          boolean flag = true;
17          for each dep in ref.depobjectlist{
18              if(!contains(dep.currentWaiting, ref)
                 && !contains(dep.nextWaiting, ref)){
19                  flag = false;
20                  break;
21              }
22          }
23          if(flag) monitor_queue.add(ref);
24      }
25      if(monitor_queue.isEmpty)     break;
26
27  //dep オブジェクト起動
28  while(r=monitor_queue.next() != null){
29      //ref のカーソルを進める
30      for each ref in R {
31          if(ref.cursor.hasNext)     ref.cursor++;
32      }
33      for each dep in r.depobjectlist{
34          dep.Algorithm3(r, r.value);
35      }
36  monitor_queue.clear;
37  }
```

図 7　Monitor オブジェクトが実行するアルゴリズム

```
Input: referencedObject, referencedValue
 1  // compare with next dependent value
 2  if referencedObject∈ nextWaiting then
 3      nextWaiting := nextWaiting\ referencedObject;
 4      next := next ∪ {(referencedObject, referencedValue)}
 5      return;
 6
 7  // compare with current dependent value
 8  currentWaiting := currentWaiting\ referencedObject;
 9  processComparison(referencedObject, referencedValue);
10
11  // Do we need current value any longer?
12  if currentWaiting = φ and
                (next != φ or nextWaiting != φ) then
13      dependentValue := next dependent value;
14      // update waiting lists
15      currentWaiting := nextWaiting;
16      nextWaiting := φ
17      // test corresponding inclusion dependencies
18      do{
19          next_old = next;
20          next = φ;
21          for each otherReferencedObject in next_old{
22              processComparison(otherReferencedObject,
                value of otherReferencedObject);
23          }
24          boolean cond = (currentWaiting = φ) and next != φ then
25          if(cond){
26              dependentValue := next dependent value;
27              // update waiting lists
28              currentWaiting := nextWaiting;
29              nextWaiting := φ
30          }
31      }while(currentWaiting = φ and next != φ);
```

図 8　dep オブジェクトのメインアルゴリズム

4. スコアリング

これまで，我々は Web ページ要素間の包含関係 $X \subseteq Y$ に関

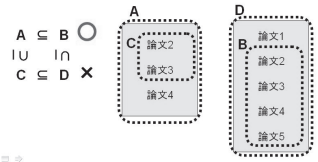

図 9　スコアリングルール 1

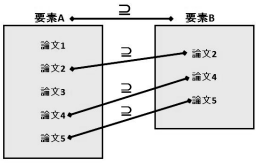

図 10　フィルタリングルール 2

する出力 $(X \subseteqq Y, p, s)$ のうち，p を計算する方法を説明してきた．ここでは，重要度スコア s の計算方法について議論する．

重要度スコアを計算する動機は次の通りである．すなわち，ページに含まれる全ての要素の組み合わせの包含関係を計算する場合，自明なルールとそうでないルールが混在し，価値の高い包含関係が埋もれてしまうことである．これは，Web ページ要素間に階層関係が存在することに主に起因する (図 1)．

そこで，我々は出力される包含関係に対して重要度スコア $(0 \leqq s \leqq 1)$ を計算するための 3 つのルールを定義し，これを用いて自明でない包含関係の発見を支援することを提案する．

以下の各ルールの説明においては，x，y をそれぞれ Web ページ要素とし，X，Y をそれぞれの要素が含む単語の集合とする．デフォルトの重要度スコアとして，あらかじめ全ての包含関係に 1 が与えられているとするすなわち，全ての包含関係に対して $(X \subseteqq Y, p, 1)$ がデフォルトの状態であるとする．そして，ルールを適用することによって重要度のスコアを変更する．

4.1　スコアリングルール 1

これは，他のルールから演繹できる自明な関係の重要度を下げるルールである (図 9)．

ルール 1：$r_1 : A \subseteqq B$，$r_2 : C \subseteqq D$ が許容率 $p >> 0$ で成立すると仮定する．ルール 1 を適用する前の r_2 の重要度スコアを $score_{old}(r_2)$ とする．もし，C が A の下位要素であり，かつ，D が B の上位要素である時，ある $c < 1$ を用いて $score(r_2) = score_{old}(r_2) \times c$ とする．

4.2　スコアリングルール 2

これは，Web ページ要素の集合間の包含関係と，それらの部分集合間の包含関係が与えられたとき，後者の関係が前者から容易に推測できる場合に，後者の重要性を下げるルールである．

例えば，図 10 のような場合，論文リストの各論文同士の関係の重要度が下がることになる．具体的には，図中で要素 A と要素 B を ul タグの要素，論文 1〜5 を li タグの要素とする．また，要素 A と要素 B の関係以外に，論文 2・4・5 同士の関係が発見されているとする．このとき，論文 2・4・5 同士の関係の重要度が下げられる．

ルール 2：$r_0 : A \supseteqq B$，$r_1 : C_1 \supseteqq D_1$，$r_2 : C_2 \supseteqq D_2 ... r_n : C_n \supseteqq D_n$ が許容率 $p >> 0$ で成立するとする．ルール 2 を適用する前の r_i の重要度スコアを $score_{old}(r_i)$ とする．$|B|$ を B が表す Web ページ要素のサイズとする．また，$0 < \alpha \leqq 1$ とする．この時，以下の条件が全て成立する場合に，ある $c < 1$ を用いて $score(r_i) = score_{old}(r_i) \times c$ (ただし，$i \neq 0$) とする．

(1)　$\forall_{1 \leqq i \leqq n}(A \supseteqq C_i)$
(2)　$\forall_{1 \leqq i \leqq n}(B \supseteqq D_i)$
(3)　$\dfrac{|\bigcup_{1 \leqq i \leqq n} D_i|}{|B|} > \alpha$

ここで，α をカバー率と呼ぶ．カバー率とは，要素 B の部分要素のうち，どれぐらいの割合が要素 A の部分要素との間に包含関係があるかを表したものである．$\alpha = 1$ の時には，全ての B の子要素に対して，A の子要素との間に包含関係が存在することは自明であるため，これらの包含関係の重要性は低くなる．本ルールでは Web ページコンテンツの不完全さを考慮し，α が 1 よりやや小さい場合にも適用可能な余地を残している．

4.3　スコアリングルール 3

これは，許容率をわずかにあげるだけで自明でない包含関係が得られるとき，前者の重要度を下げるためのルールである．

図 12 に例を示す．ここでは，C と B では「札幌」と「北海道」のように一部の語が異なっており，かつ「北海道」が C の外に存在するため，低い許容率では $B \subseteqq A$ しか発見できないが，少し許容率を上げるとより自明でない $B \subseteqq C$ が発見できる．つまり，許容率が低い時には body のような全体との関係になるが，少し許容率を上げるだけで，その度合い以上に，粒度のかなり小さな要素同士の組み合わせが発見できる場合に，body のような全体との関係を除去しようというものである．

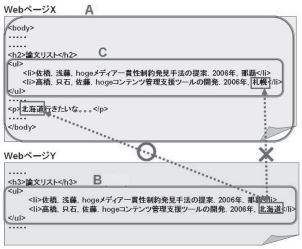

図 11　フィルタリングルール 3

ルール 3：$r_1 : A \supseteqq B$ が許容率 p_1 で成立し，$r_2 : C \supseteqq B$ が許容率 p_2 で成立するとする．また，$p_1 < p_2$ とする．ルール 3 を適用する前の r_1 の重要度スコアを $score_{old}(r_1)$ とする．この時，次の条件が全て成立する場合に，ある $c < 1$ を用いて $score(r_1) = score_{old}(r_1) \times c$ とする．

(1)　$A \supseteqq C$
(2)　$\frac{|C|}{|A|} \times \frac{p_2}{p_1} < \beta$

ここで，$\beta \leqq 1$ である．$\beta = 1$ の時，許容率の変化と A から C へのサイズの変化が等くなる．β の値が小さいほど，劇的に C のサイズが小さくなることが求められる．

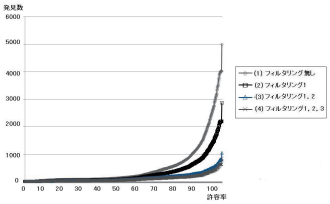

図 12　包含関係の発見数

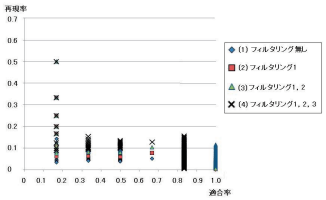

図 13　再現率と適合率

5. 実　　験

本節では，提案手法の性質を調べるために行った予備実験について説明する．本実験では，Web コンテンツに含まれる各 Web ページ要素 x の内容を文字 3-gram で単純に分割したものを対応する集合 X とした．

5.1 実　験　1

実験 1 では，ある Web ページの組を対象に本アルゴリズムを適用した．対象としたページは，ある研究室の教員の Web ページと，同じ研究室に所属するある学生の Web ページである．正解となる包含関係の集合を用意し、アルゴリズムの出力から重要度 1 の包含関係だけを選択し，適合率と再現率を計算した．実験は下記の 4 つの場合について行い，結果を比較した．今回の実験では，α を 0.8，β を 0.1 とした．

（1） スコアリング無し (全ての重要度スコアが 1)
（2） スコアリングルール 1 のみ適用
（3） スコアリングルール 1，2 を適用
（4） スコアリングルール 1，2，3 を適用

図 12 に，許容率と成立した包含関係の関係を示す．スコアリングによって有用度の低い包含関係は 1 より低いスコアをつけられるため，重要と考えられる包含関係の数を減らすことが出来ることが分かる．図 13 は，適合率と再現率の関係を示したものである，スコアリングルールを適用するに従って，再現率と適合率が共に上がる傾向にあることが分かる．

5.2 実　験　2

実験 2 では，筑波大学情報学群の Web サイト，および情報学群に属する各学類 (情報科学類，知識情報図書館学類，情報メディア創成学類) の Web サイトを対象とし，関連のある部分が発見できることの確認を行った．

これらは同じような内容を含んでいるにも関わらず，管理者が違うために統一的に管理されていないサイトである．

適用の結果，情報学群への問合せ先として各学類の事務の住所・電話番号・メールアドレス等が一覧で示されているページがあり，各学類の Web サイトに含まれる事務の住所・電話番号・メールアドレスとの包含関係等を発見することが出来た．図 14 に発見できた包含関係の例を示す．これは学類長らの受験生に向けたメッセージが書かれたページであり，図 14 の A,B にはほぼ同じ内容が含まれており，包含関係があった．この包含関係は許容率 43%で成立した．

図 14　筑波大学情報学群のサイトに適用して発見された包含関係の例

6. まとめと今後の課題

本稿では，既存の Web コンテンツを入力として，コンテンツ間の包含関係を計算するためのアルゴリズムの提案を行った．本アルゴリズムでは，Web コンテンツには誤りが多いこと，および Web コンテンツが階層構造を持つことに着目し，下記の工夫を行った．(1) 許容率の概念を導入し，この値と共に包含関係と共に出力する．(2) 自明でない包含関係を優先するために重要度スコアを導入し，それを計算するためのルールを作成した．予備実験の結果，これらの手法が有効に働く可能性が高いことが分かった．今後の課題としては，重要度スコアや許容率の閾値を指定してより効率よく発見を行うための仕組みの開発などがあげられる．

7. 謝　　辞

ゼミなどでコメントいただきました筑波大学大学院図書館情報メディア研究科の阪口哲男准教授，永森光晴講師に感謝いたします．本研究の一部は科学研究費補助金特定領域研究 (#19024006)，科学研究費補助金基盤研究 (B)(#19300081)，科学研究費補助金若手研究 (B)(#20800076) による．

文　　献

[1] 筑波大学情報学群，”http://inf.tsukuba.ac.jp/”，(参照 2008-08-22)
[2] 澤菜津美，森嶋厚行，飯田敏成，杉本重雄，北川博之．コンテンツ一貫性制約を用いた Web サイト管理手法の提案．DEWS2007, 7 pages，2007 年 3 月．
[3] Serge Abiteboul, Richard Hull, Victor Vianu: Foundations of Databases. Addison-Wesley 1995.
[4] Natsumi Sawa，Atsuyuki Morishima，Shigeo Sugimoto，Hiroyuki Kitagawa．Wraplet: Wrapping Your Web Contents with a Lightweight Language．Proc. IEEE SITIS’ 2007.
[5] 澤菜津美，森嶋厚行，杉本重雄，北川博之．情報統合利用を目的とした HTML ページのラッピング支援．DEWS2008, 2008 年 3 月．
[6] 高橋公海，澤菜津美，森嶋厚行，杉本重雄，北川博之．Web コンテンツ一貫性管理支援ツールの開発．第 70 回情報処理学会全国大会講演論文集 (第 5 分冊)，pp. 189-190, 2008 年 3 月．
[7] J. Bauckmann，U. Leser，F. Naumann．Efficiently Computing Inclusion Dependencies for Schema Discovery．InterDB’06 (ICDE Workshop)，2006.