Webページ移動先発見のための効率的なクローリング手法

澤 菜津美[†] 森嶋厚行[†] 飯田敏成[†,☆]
杉本重雄[†] 北川博之[††]

近年，World Wide Web は社会における重要なメディアの 1 つとして大きな役割を果たしている．Web の特徴の 1 つに分散管理があるが，これはしばしば内容の一貫性が維持されない原因となっている．我々は Web コンテンツの一貫性維持支援の 1 つとして，Web ページの移動によってリンク切れが引き起こされたときに，移動先を発見する問題に取り組んでいる．これまでの実験の結果，多くのページが，同一 Web サイト内で移動していることが分かった．したがって，Web サイト中の Web ページをクローリングすることが，移動先発見の手法として有効である．しかし，大規模な Web サイト全体を網羅的にクローリングすることはコスト的に問題がある．本稿では，クローリングにおけるページ訪問の順序を工夫することにより，より少ないページ数で移動先ページを発見するための手法を提案する．実験の結果，提案手法が有効であることが分かった．

An Efficient Crawling Method for Finding Moved Web Pages

Natsumi Sawa,[†] Atsuyuki Morishima,[†] Toshinari Iida,[†,☆]
Shigeo Sugimoto[†] and Hiroyuki Kitagawa[††]

The World Wide Web has become an indispensable medium in our society. The integrity of its contents, however, is not always maintained because of its distributed architecture. As an effort to support the maintenance of the integrity, we have been tackling the problem of finding moved Web pages when the movement causes broken Web links. Our previous experiments on the problem showed that many moved Web pages can be found at the Web sites that the Web pages were originally located at. Therefore, crawling through the Web site is an effective way to find moved Web pages. But naive crawling methods would take huge costs when the size of the Web site is large. This paper proposes a novel crawling algorithm that visits Web pages in an efficient order so that it requires a fewer number of page accesses to find the destinations of moved Web pages. Our experimental results showed that our proposed algorithm is effective.

1. はじめに

近年，World Wide Web（以下 Web）は社会における重要なメディアの 1 つとして大きな役割を果たしている．Web の特徴の 1 つに分散管理が行われていることがあげられる．この特徴は，Web を便利なツールとする一方で，Web コンテンツの一貫性の維持を困難としている要因でもある．ここでいう Web コンテンツの一貫性の維持とは，分散管理されているコンテンツ間の関係を，当初意図されていた状態のままに維持することである．たとえば，(1) あるページからリンクされている先のページの内容が当初意図していたとおりである，(2) あるページのコンテンツと別に管理されているページのコンテンツが一致している，等がある．本稿では Web リンクの一貫性の維持の問題のうち，特にリンク切れへの対処の問題に焦点を当てる．Web のリンク切れは，以前から重要な問題として認識されてきた．たとえば，1998 年のジョージア工科大学によるアンケート[1] では，利用者から見た Web の問題点として，リンク切れが第 3 位にあげられている．また，2003 年の Science の記事[2] でも，Web のリンク切れが依然重要な問題であることが指摘されている．近年，Web-DB 連携システムなどによって動的に生成された Web コンテンツが増えてきているが，そのような状況においても，リンク切れの問題は解決できていない．なぜなら，外部サイトへのリンク集のメンテナンスなどは Web-DB 連携システムでは解決

† 筑波大学大学院図書館情報メディア研究科
Graduate School of Library, Information and Media Studies, University of Tsukuba
†† 筑波大学大学院システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba
☆ 現在，株式会社日立製作所
Presently with Hitachi, Ltd.

できないうえ，そもそもコストのかかる Web-DB 連携システム構築を行わずに，人手による HTML ページ管理を依然として行っているサイトも数多く存在するからである．

この問題を解決するため，我々はこれまで，ページの移動によって生じたリンク切れを対象として，Web ページの移動先を自動発見し，ページ移動によるリンク切れの修正を支援するシステム（WISH システムと呼ぶ）を開発し[3)~6)]，実験や公開☆を行ってきた．移動先の探索は，移動前のページのコンテンツや，移動前の場所に関する情報などを用いて行われる．

WISH システムは，Google などの Web 検索エンジンによるページ検索とは異なる方法でページの移動先発見を行う．一般に，Web 検索エンジンは，あらかじめ構築したインデックスを利用して必要なページの検索を行う．しかし，ページの移動先を発見するためには，移動先ページがインデックスされている必要があるため，このアプローチは次の点で不十分である．(1) 移動後でなければ移動先ページにインデックスを貼れないため，移動先ページがインデックスされるまでに時間を要することがある．(2) マイナーなページなど，ページによってはインデックスされないことがある．

我々が着目した点は，検索エンジンを利用しなくても，人間はページの移動先を迅速に発見できることが多いことである．人間がページの移動先を発見できる理由は，ページの移動先探索においては「移動先がありそうな場所」に偏りが存在するからである．これらの場所を優先して探索することにより，人間は比較的短時間でページの移動先を発見することができる．この性質を利用して，WISH システムでは，リンク切れ発見後にクローラを利用して「移動先がありそうな場所」を探索するアプローチが用いられる．そのために，我々はいくつかのヒューリスティクスを開発し，WISH システムに組み込んでいる．これまでの実験[5),7)]から，WISH システムは，Google などの Web 検索エンジンを利用した方式と比較して，ページの移動先の発見率が大幅に高いことが分かった．

また，これまでの実験の結果から，Web ページの移動先の多くが同一サイト内であることが判明した[5)]．同一サイト内のページをクローリングすることにより，これらのページは発見できるが，サイトの規模が大規模になると，単純なクローリングでは，クローリングのコストが膨大なものになってしまう．さらに，リンク集などの場合，リンク先のサイトの管理者は WISH の利用者と異なることが一般的であるため，アクセスは最小限に行い，負担をかけないようにする必要がある．

本稿では，同一サイト内のクローリングにおけるページ訪問の順序を工夫して，より少ないページ数で移動先ページを発見するための手法を提案する．ポイントは，(1) サイトのルートから移動前のページの場所へのリンク経路に着目して訪問先の優先順位を決定すること，および，(2) アンカ文字列に着目して，ありそうな場所を優先的に探索する，の 2 点である．実験により，本手法は，他の手法と比較して少ないページ訪問数で移動先のページに到着することが分かった．

本稿は次のように構成されている．まず，2 章で WISH システムの概要を説明する．3 章で提案手法を説明する．4 章では，実験により提案手法の評価を行う．5 章で関連研究を説明する．6 章はまとめと今後の課題である．

2. WISH システム概要

2.1 アーキテクチャ

図 1 は WISH システムのアーキテクチャである．処理の流れは次のとおりである．あらかじめ，ユーザがシステムに監視対象とするリンクを登録する（単純化のため，ここではただ 1 つのリンク u とする）．監視モジュール（LI-Manager）は，登録された u のページコンテンツのキャッシュ c を保存し，監視を開始する．リンク切れが発見されると，ページ探索モジュール（PageChaser）が，移動先の探索を始める．探索の結果は「ページの移動先らしさ」を表すスコアでランキングされた URL のリスト $\bar{U}$ であり，LI-Manager に返される．最後に LI-Manager は，指定されたポリシに基づき，結果を利用する．たとえば，スコアの閾値によりリンクを自動更新したり，移動先の候補をメールで利用者に連絡したりする．

2.2 移動先の発見

移動先の発見は，次の 2 段階で行われる．

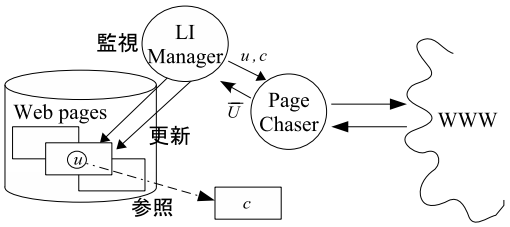

図 1 WISH システム
Fig. 1 The WISH system.

☆ http://wish.slis.tsukuba.ac.jp/LIM-RO.html

ステップ 1：移動先の候補収集 クローラなどを用いて u の移動先の候補となる URL の集合 U を収集する．

ステップ 2：ランキング U 中の各 URL に「移動先らしさ」を表すスコアをつけ，ランキングを行った結果 $\bar{U}$ を計算する．

ステップ 2 におけるスコアの計算は，キャッシュ c の情報だけでなく，移動前のページの位置情報や，他のページからのリンク情報などを利用して行われる．この詳細については文献 5)，7) にある．

本稿で議論するのは，ステップ 1 において，移動元と同一サイト内から移動先候補を収集することである．具体的には，同一サイトのページを順次クローリングし収集する過程において，どのような順序で訪問していけば，少ないページアクセス数でもページの移動先の正解が U に含まれるか，を議論する．この効率が高いほど，ページの移動先を発見するためのコストが少なくてすむことになる．候補収集手法の効率を評価するためには，あらかじめページが移動したこと，およびその移動先が分かっている Web ページを対象に，何回のページアクセスでその移動先にたどりつけるかを見ればよい（4 章の実験ではこれを行う）．このステップ 1 の問題はステップ 2 のランキングの問題とは完全に独立していることに注意していただきたい．本稿ではステップ 2 については議論せず，ステップ 1 の問題に焦点を当てる．

3. 提案するクローリング手法

本章では，まず本手法で利用する「リンク経路」について説明する．次に，ページの移動前にリンク経路と関連するアンカ文字列を求める方法を説明し，最後に，ページの移動後にこれらの情報を利用してクローリングのためのページ訪問順序を決定する方法を説明する．

3.1 リンク経路

定義. 各 Web ページの URL を u_i，Web リンクを有向辺 (u_i, u_j) とし，Web を有向グラフでモデル化する．このとき，u_0 から u_m への**リンク経路**（link path）とは $u_0 \to u_1 \to \cdots \to u_m$（ただし $m \geq 0$）の形をした Web リンクの有限列のうち，URL 群 $u_0, u_1, \ldots, u_m$ がすべて異なるものである．

Web サイトのルートページの URL を u_0 とし，監視対象ページの URL を u_{end} とした場合のリンク経路の例を**図 2** に示す．一般には，このようなリンク経路は複数存在する．本手法では，このような，ルートページ u_0 から監視対象ページ u_{end}（移動前）までの 2 点間のリンク経路と，このリンク経路上の各リンクに含まれるアンカ文字列の情報を利用して，ページの訪問順序を決定する．本手法のポイントは，ページの移動先はルートからページ移動前の位置へのリンク経路の周辺に存在する可能性が高いというヒューリスティクスを利用し，その周辺を優先してクローリングを行うことである．

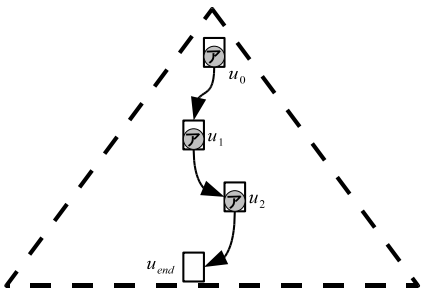

図 2 リンク経路の例
Fig. 2 Example of a link path.

具体的には，本手法は下記の 2 つのフェーズから構成される．

[移動前フェーズ] これは移動前のページ u_{end} を指すリンクが監視対象として WISH システムに登録された時点で実行される．具体的には，与えられた u_{end} が所属する Web サイトを探索して，リンク経路 $u_0 \ldots u_{end}$ を（一般には複数）発見する．詳細は 3.2 節で説明する．

[移動後フェーズ] これは u_{end} へのリンク切れが発見された時点で実行される．リンク経路の周辺を，アンカ文字列を手がかりに順に探索する．詳細は 3.3 節で説明する．

以下では，それぞれのアルゴリズムの詳細を説明する．なお，誤解が生じない範囲で，URL u に存在する Web ページのことをその URL u で表すことがある．

3.2 移動前フェーズ：リンク経路の発見

移動前フェーズでは，監視対象リンクが指す URL u_{end} を手がかりとして，ルートページ u_0 から u_{end} までのリンク経路を発見する．Web リンクはリンク元からリンク先への 1 方向にしかたどれないため，最も自明な方法は，ルートの URL u_0 から開始して u_{end} にたどり着くまでリンクを網羅的にたどることである．しかし，この方法は大規模サイトなどでは現実的ではない．そこで我々は，u_{end} から開始して u_0 に向かい逆向きにリンク経路を探索する手法を用いる．

もう 1 つのポイントは，リンク経路が一般には複数あることである．リンク経路の発見のためにページアクセス数が増えてしまっては本末転倒であるため，できるだけ移動前フェーズにおいてもページアクセスの

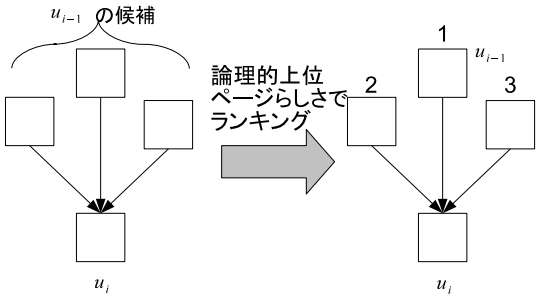

図 3 u_i に対する論理的上位ページらしさによる順位付け
Fig. 3 Page ranking by their likelihood of being logically superordinate to u_i.

回数を減らしたい．したがって，すべてのリンク経路を数え上げるのではなく，「もっともらしい」リンク経路を優先して探索し，移動前フェーズにおけるページアクセス数が限定された場合でも，効率良いページ移動先探索に役に立つリンク経路を見つけられるようにする．4 章の実験では，下記で説明する提案アルゴリズムが，限定されたページアクセスでも効果的なリンク経路を発見することが示される．

リンク経路発見手法の概要． 本手法のアイデアは次のとおりである．すなわち，u_i を与えられたときに u_{i-1} を求めることを 1 ステップとして，これを再帰的に繰り返すことによりリンク経路を探索する．この探索は，次の 3 段階から構成される（**図 3**）．(1) u_i へのリンクを持つ同一サイト内のページを収集し，それらを u_{i-1} の候補とする．(2) u_{i-1} の候補を，u_i の「論理的上位ページらしさ」によって順位付けする．(3) u_{i-1} の候補のうち，「論理的上位ページらしさ」の高いものから深さ優先に探索を行う．すなわち，最初は最も順位の高いものを u_{i-1} として再帰的に以上のプロセスを適用し，その後 2 番目に順位が高いものを u_{i-1} として同じプロセスを適用するということを繰り返す．

Web サイトの論理構造と論理的上位ページ． 上記の (2) において，u_{i-1} の候補に順位付けを行う理由は，効率的なページ移動先探索に役に立ちそうなリンク経路を優先して発見したいからである．

そこで，何らかの基準でリンクを選択する必要がある．その基準のことを本手法では，「論理的上位ページらしさ」と呼んでいる．

あるページが別のページの論理的上位ページであるとは，それらのページが表しているそれぞれの概念が，実世界で論理的上位下位の関係にあるようなものである．たとえば，大学と学部，学部と学科，学科と研究室，研究室とその構成メンバを表すような各ページでは，それぞれ前者が後者の論理的上位ページである．また，これらのページ間の論理的上下関係によって表現される構造を Web サイトの論理構造と呼ぶ．我々は，このような論理構造に対応したリンク経路の周辺を探索していくことにより，ページの移動先発見を効率的に行うことができると考えている．

本手法では，論理的上位ページの厳密な定義は行わず，表現されている論理的構造の上下関係が，Web サイトの URL，Web ページのファイル名，リンク関係などにある程度反映されていることに着目した．提案アルゴリズムに組み込まれているヒューリスティクスは次のとおりである．

(1) URL u_{i-1} の長さ（ディレクトリ列の長さ）が u_i より短いとき，ページ u_{i-1} が u_i の論理的上位ページである可能性が高い．u_{i-1} が u_i の部分列であるとき，その可能性はさらに高くなる．

(2) Web サイトの主要な論理構造（すなわち論理的上下関係の構造）を表現する Web ページはファイル名が index.html（もしくは類するもの）である可能性が高い．

(3) ページ u_{i-1} が u_i の論理的上位ページであるとき，ページ u_i から u_{i-1} に向かって「戻る」などのリンクが存在する（すなわち，相互リンクがある）可能性が高い．

本手法では，これらのヒューリスティクスを利用して，「論理的上位ページらしさ」の順位付けを行う．

アルゴリズムの詳細． **図 4** がリンク経路探索のアルゴリズムである．関数 PathFind は初期入力として，監視対象リンクが指す URL である u_{end} と，空のアンカ文字列リストを受け取る．出力するのは，リンク経路の集合 *ResultPathSet* および各リンク経路上のアンカ文字列のリストの集合 *AnchorListSet* である．

PathFind が呼び出されると，まず URL のリスト P を $[u_{end}]$ に初期化する（8 行目）．P はリンク経路探索の中間結果（一般には，$[u_i, u_{i+1}, \ldots, u_{end}]$ が格納される）を保持するためのデータ構造である．

次に，URL u_{end} を構成する文字列からディレクトリ部分を削除し，u_0 を求める（9 行目）．u_{end} をディレクトリ単位で区切った複数の URL（ただし，ルートページの URL は除く）を，URL 集合 *candidates* に追加する（11 行目）．*candidates* は，リンク経路上に現れうる URL 候補の集合である．たとえば，http://a/b/c/dog.html という URL の場合，http://a/b/，http://a/b/c/が *candidates* に加えられる．最後に，PathFind から呼び出される関数 PathGrow によって，リンク経路を構築していく．

```
Input: リンク経路のリスト P
Output: リンク経路の集合 ResultPathSet
        およびアンカ文字列リストの集合 AnchorListSet
 1    Global Set candidates={}; // c_i の候補の集合
 2    Global Set searched_set={};
                          // リンク抽出が終わった URL の集合
 3    Global Set ResultPathSet={}; // 完成したリンク経路の集合
 4    Global Set AnchorListSet={}; // アンカ文字列リストの集合
 5    Global int max=N; // PathGrow を呼び出す最大回数
 6    Global URL u_0; // ルートページの URL
 7    void PathFind(u_end){
 8      List P = [u_end]; // リンク経路 P を [u_end] に初期化
 9      u_0 = get_root(u_end); // u_end から u_0 を抽出する
10      // u_end をディレクトリでカットして追加
11      candidates.addAll(cut_dir(u_end));
12      // 関数 PathGrow を呼び出す
13      PathGrow(P,[]);
14    }

15    void PathGrow(P,anchors) { // P と anchors を求める
16      max = max − 1;
17      if(max > 0) {
18
19        URL u_i = P.getFirst(); // P から先頭を取り出す
20        if (!searched_set.contains(u_i)) {
21          List U = containedURLs(u_i); // URL の抽出
22          candidates.add(U);
23          searched_set.add(u_i);
24        }
25        // candidates から u_i へリンクする c_j を抽出する
26        Set upper_set={}; // 抽出した c_j の集合
27        for each c_j in candidates {
28          // c_j が u_i にリンクしているかどうかを調べる
29          if (c_j → u_i) {
30            if (c_j = u_0) { // c_j が u_0 ならばリンク経路完成
31              P.addFirst(c_j);
32              anchors.addFirst(anchor(c_j → u_i));
33              ResultPathSet.add(P);
34              AnchorListSet.add(anchors);
35            } else {
36              // u_{i−1} 候補として upper_set に追加
37              upper_set.add((c_j,anchor(c_j → u_i),null));
38            }
39          }
40        }
41        for each (c_k, a_k, null) in upper_set {
42          // 論理的上位ページらしさを判定
43          score_k = Check(c_k,u_i);
44          upper_set.update(c_k, a_k, score_k); // スコアを格納
45        }
46        List sortedList = sort(upper_set);
47        // 再帰呼び出し
48        for each (c_k, a_k, score_k) in sortedList {
49          P.addFirst(c_k);
50          anchors.addFirst(a_k);
51          PathGrow(P, anchors);
52          P.removeFirst();
53          anchors.removeFirst();
54        }
55      }
56      return;
57    }
```

図 4 移動前フェーズ：リンク経路発見アルゴリズム
Fig. 4 Pre-movement phase: the algorithm for finding link paths.

関数 PathGrow は次のように動作する．すなわち，PathFind から最初に呼ばれたときは，リンク経路の中間結果は $P = [u_{end}]$ であり，アンカ文字列のリストは $anchors$=[ ] である．そして，PathGrow の再帰呼び出しによって，あるリンク経路 $P = [u_0, u_1, \ldots, u_{end-1}, u_{end}]$ を構築する．このリンク経路 P が 1 つ完成するたびに，リンク経路の集合である $ResultPathSet$ に保存され，次の P の探索へと移る．同時に，完成した $anchors$ は，アンカ文字列リストの集合 $AnchorListSet$ に保存される．このように，複数のリンク経路が（存在すれば）発見されることになる．

定数 max は，関数 PathGrow の呼び出し回数を制限するために利用される．すなわち，max 回の PathGrow 関数を実行した時点でアルゴリムは終了する．

関数 PathGrow の詳細は次のとおりである．

19–24 行目 P の先頭から URL u_i を取り出し，まだ訪問していないページならば，関数 $containedURLs$ により u_i のコンテンツから URL を抽出する．抽出した URL は集合 $candidates$ に入れる．この $candidates$ には，これまでの探索で収集した URL がすべて格納されている．

25–40 行目 各 URL $c_j \in candidates$ に対し，c_j から u_i へリンクしているかどうかを調べる．true ならば，さらに，$c_j = u_0$ かどうか調べる．$c_j = u_0$ ならばリンク経路完成となり，P は $ResultPathSet$ へ格納され，$anchors$ は $AnchorListSet$ へ格納される．$c_j \neq u_0$ ならば，c_j が集合 $upper\_set$ に格納される．この $upper\_set$ は，図 3 で説明した u_{i-1} の候補集合である．

41–45 行目 $upper\_set$ に含まれる各 c_k に対し，“論理的上位ページらしさ” を判定し，スコア $score_k$ を求め，$upper\_set$ に格納する．論理的上位らしさを表すスコアは，先に説明したヒューリスティクスを用いて計算する．ここでは，URL を構成するディレクトリ列の差の数に加え，ファイル名が index.html であれば 2 点追加し，さらに相互リンクが存在すれば 1 点追加することとする．

46 行目 関数 $sort$ は $upper\_set$ に含まれる c_k を $score_k$ 順でランキングし，リスト $sortedList$ に格納する．

47–54 行目 $sortedList$ の先頭から 1 つ c_k を取り出し，これを P の先頭に追加する．また，対応するアンカ文字列を $anchors$ に追加する．そして，再帰的に PathGrow を呼び出す．

56 行目 定数 max 回の PathGrow 呼び出しが行われた時点で，探索を終了する．

3.3 移動後フェーズ：リンク経路とアンカ文字列を利用したクローリング

u_{end} へのリンク切れが発見されると，移動前フェーズで発見されたルートから u_{end} への各リンク経路

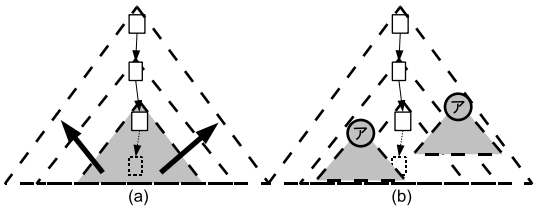

図 5 リンク経路を利用したクローリングアルゴリズム
Fig. 5 Crawling algorithm using link paths.

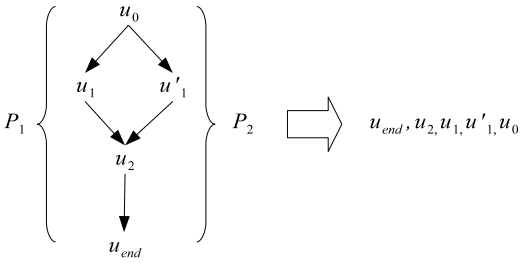

図 6 URL 群のトポロジカルソート
Fig. 6 Topological sort of a url collection.

$P_i \in ResultPathSet$ を利用して，同一サイト内で移動先ページ v_{end} を探索する．

クローリング手法の概要． まずは簡単化のために，$ResultPathSet$ に含まれるリンク経路が 1 つの場合について説明する．基本は，リンク経路 $P_i = [u_0, \ldots, u_{end}]$ 上のページ u_{end-1} を起点としてリンクをたどり探索を行うことである（**図 5** (a)）．この探索においては，P_i 上のアンカ文字列のリスト $anchors_i$ の情報を利用する．この探索が終了したら，u_{end-2}，u_{end-3} と順にルートに向かって起点を変更していく．1 度訪問したノードは探索しない．この，P_i 上の各 u_j を起点とした探索を以下では traverse(u_j,$anchors_i$) と表記する．

ここで，traverse(u_j,$anchors_i$) について説明する．これは，原則は幅優先探索であるが，移動前フェーズにおいて保存されていたアンカ文字列リスト $anchors_i$ に含まれるアンカに出会った場合のみ，局所的に深さ優先探索に切り換える（図 5 (b)）．これは，取得していたリンク経路上のアンカの文字列と一致した場合には，その先に移動先がありそうだというヒューリスティクスによる．この際，$anchors_i$ に含まれるすべてのアンカとの一致を調べるのではなく，起点の変更に応じて最初は u_{end} にいたるリンクのアンカとのみ一致を調べ，次は u_{end} と u_{end-1} にいたるリンクのアンカ，という風に順に候補を増やしていく．

次に，$ResultPathSet$ に含まれるリンク経路が 2 つ以上の場合について説明する．この場合，リンクを順序関係と考えて，すべてのリンク経路に現れる URL 群およびアンカ文字列群のトポロジカルソートを行い，その順に探索を行う（**図 6**）．これは，上で説明したリンク経路が 1 つの場合の一般化になっていることに注意していただきたい．このような順に traverse を呼び出す理由は，可能な限り局所的に探索を行いたいからである．もし，先にリンク経路 P_1 上のページを起点とした探索をすべて行うと，P_2 を利用する前に u_0 を起点とした Web サイト全体を探索することになってしまう．リンク経路が 2 つ以上の場合に利用する $anchors$☆は，各 u_j を含むリンク経路上のアンカ文字列をすべて含むものである．$anchors$ 中のどのアンカ文字列との一致を調べるかについては，リンク経路が 1 つの場合と同様である．

アルゴリズムの詳細．図 7 がリンク経路を利用したクローリングアルゴリズムである．全体を制御するための関数 crawl と，上記で説明した部分探索を実現する関数 traverse で構成されている．

関数 crawl は，入力として，リンク経路の集合 $ResultPathSet$ と，アンカ文字列リストの集合 $AnchorListSet$ を受け取る．出力は，監視対象ページの移動先候補 URL のリスト $candidates$ である．

探索最大ページ数 $page$ を設定し，移動先候補の集合 $candidates$ に格納される移動先 URL の総数が $page$ に達するまで探索する．関数 crawl の詳細は次のとおりである．

4–5 行目 $ResultPathSet$ に含まれるすべてのリンク経路のノードをリンク構造によってトポロジカルソートしたものを，リスト $OrderedURLs$ に格納する．

6–8 行目 リスト $OrderedURLs$ から URL u_j を先頭から順に取り出し，traverse(u_j, $anchors$) を実行する．

9 行目 最後に $candidates$ を返す．

次に，関数 traverse について説明する．関数 traverse の動作は，幅優先探索のためのキューと，深さ優先探索のためのスタックを組み合わせることにより実現する．具体的にはスタックを 1 つ用意し，URL を格納するキューをスタックに出し入れすることで，幅優先探索と深さ優先探索の切替えを行う．

切替えの例を**図 8** に示す．各ノードは Web ページを表す．網がかかっているアンカは，$anchors$ に含まれるアンカ文字列とマッチするものである．また，現在の訪問 Web ページを u とする．このとき，下記の

☆ 特定のパスのアンカ集合ではないので，添え字の i は用いない．

Input:
リンク経路の集合 Set $ResultPathSet$={P_1,P_2,$\cdots$,P_n},
アンカ文字列リストの集合 Set AnchorListSet =
$[anchors_1, anchors_2, \ldots]$
Output: 移動先候補の URL リスト $candidates$

```
1    Global List candidates=[];// 移動先候補
2    Global int page=N; // 探索最大ページ数

3    List crawl(ResultPathSet,AnchorListSet){
4      List of URLs OrderedURLs =
5                          TopologicalSort(ResultPathSet);
6      for each u_j in OrderedURLs {
7        traverse(u_j, anchors);
8      }
9      return candidates;
10   }

11   void traverse(u,anchors) {
12     Queue q_0 = []; // キュー
13     Stack st = []; // スタック
14     q_0.enqueue((u,match)); // キューに追加
15     st.push(q_0); // スタックに追加
16
17     while(!st.empty()) {
18       // 最大探索ページ数を超えたらブレイク
19       if(candidates.size() ≥ page) break;
20
21       Queue q_1 = st.pop();// キューを取り出す
22       (u, flag) = q_1.dequeue(); // url を 1 つ取り出す
23
24       if(!candidates.contains(u)) {
25         candidates.add(u); // 移動先候補リストに追加
26         List A = u.getAnchors(); // u に含まれるアンカを収集
27         List M = anchor_match(A, anchors);
                    // アンカ文字列が一致するものを取り出す
28         List UM = A - M;
                  // アンカ文字列が不一致なものを取り出す
29
30         Queue q_2=[];
31         Queue q_3=[];
32         if(flag = match) {
33           if(!q_1.empty()) st.push(q_1);
34         }else { // flag = unmatch
35           if(!q_1.empty()) q_2 = q_1;
36         }
37         q_2.enqueue((UM,unmatch));
38         q_3.enqueue((M,match));
39
40         // スタックに戻す
41         if(!q_2.empty()) st.push(q_2);
42         if(!q_3.empty()) st.push(q_3);
43       }else {
44         if(!q_1.empty()) st.push(q_1);
45       }
46     }
47   }
```

図 7 移動後フェーズ：クローリングアルゴリズム
Fig. 7 Post-movement phase: the crawling algorithm.

2 つのパターンが考えられる．

まず，図 8 (a) のように，u_a から u へのアンカが，$anchors$ 中のアンカ文字列にマッチしていない場合である．この場合，原則として幅優先探索なので次の訪問は u_c となるが，u が $anchors$ 中のアンカ文字列にマッチするアンカを持つ場合は，そちらが優先される．よって，この例では次の訪問順序は u_d，u_c，u_e の順になる．この場合を一般化すれば，スタックには次のようなキューを順にプッシュすることになる．(1) 現在の u を取り出した元のキュー（u の兄弟ページの URL 群がすでに入っている）に，u からのリンクされたページの URL のうち，そこへのアンカが $anchors$ 中の文字列にマッチしないもの（図 7 中では 28 行目で代入される変数 UM に格納）を追加したもの．図 7 では 35，37，41 行目に対応する．(2) u からリンクされたページの URL 群のうち，そこへのリンクのアンカが $anchors$ 中のアンカ文字列にマッチするもの（変数 M に格納）から構成されるキュー．図 7 では 38，42 行目に対応する．

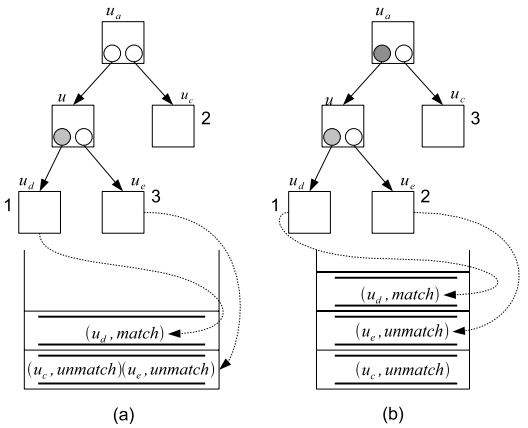

図 8 幅優先探索と深さ優先探索の切替え
Fig. 8 Switching of the breadth-first and depth-first searches.

次に，図 8 (b) のように，u_a から u へのアンカが，$anchors$ 中のアンカ文字列にマッチしている場合がある．この場合，u 以下のページを優先して訪問する．特に u の子のうち，アンカ文字列にマッチするリンク先のページを先に探索する．よって，この例では次の訪問順序は u_d，u_e，u_c の順になる．この場合を一般化すれば，スタックには次のようなキューを順にプッシュすることになる．(1) 現在の u を取り出した元のキュー（u の兄弟ページの URL 群がすでに入っている）．図 7 では 33 行目に対応する．(2) u からリンクされたページの URL のうち，そこへのリンクのアンカがアンカ文字列にマッチしない URL 群（変数 UM）から構成されるキュー，図 7 では 37，41 行目に対応する．(3) u からリンクされたページの URL のうち，そこへのリンクのアンカがアンカ文字列にマッチするもの（図 7 変数 M）から構成されるキュー．図 7 では 38，42 行目に対応する．

図 8 に示すように，キューにページの URL を格納するときは，そのページへのリンクのアンカ文字列が $anchors$ に含まれてたかどうかを表すフラグ（match もしくは unmatch）を格納する．したがって，キューから URL を取り出しながら，上記の場合分けを繰り

返し行うことができる．最初の訪問ページをキューに格納するときは，兄弟ページを考慮しないためフラグはどちらでもよい（動作が同じである）が，本コードでは match をセットしている（14 行目）．

探索の際，同じ優先順位となる URL が複数ある場合は，URL に含まれるディレクトリの数が大きい方を優先する．これは，探索が遠回りになることを避けるためである．

関数 traverse は，指定された *page* 回のページアクセスが生じると処理を終了する（2 行目，19 行目）．

4. 実　　験

3 章で説明したアルゴリズムを実装し，提案手法の有効性を検証した．まず，実験 1 として，提案手法を含む 4 つのクローリング手法について，移動ページの捕捉率の比較を行った．その結果，提案手法が，少ないページのアクセス数で移動先ページの捕捉率を高くできることが分かった．ここでいうページのアクセス数とは，実行された http リクエスト数のことであり，404 などのエラーを返したリクエスト数も含んでいる．エラーの割合については実験結果のところで説明する．次に，実験 2 として，提案手法の移動前フェーズにおいてパラメータ max を変更し，その影響を調べた．このパラメータは，リンク経路発見の際に何回 PathGrow の再帰呼び出しを行うかを指定するものである．その結果，max の値は，移動前ページの URL のディレクトリ長と同じ値で十分な性能を示すことが分かった．これは，次の 2 つの意味を持つ．(1) 移動先ページを発見するために有効なリンク経路が優先して発見されていること，および，(2) 移動前フェーズでのページアクセスコストが大幅に大きなものにならないこと，である．

実験方法の詳細は次のとおりである．本実験では，同一サイト内でのページ移動によりリンク切れが起こったことがすでに分かっているページを利用した．具体的には，1000 以上のページを持つ Web サイトにおいてページ移動によるリンク切れが発生しており，かつ，過去の実験で同一サイト内に移動先ページが存在することが既知であるページセットを利用した．これらは，移動した事実は WISH システムが発見したが，「本当の移動先がどこにあるか」は人手により調査したものである．また，このセットの作成は，提案アルゴリズムの開発者とは別人が行った．したがって，実験セットとしてこのデータを利用することは適切であると考えられる．総数は 36 ページである．

提案手法の本来の利用においては，移動前フェーズのリンク経路探索は，文字どおりページが移動する前に行われる．しかし，本実験ではすでにリンク切れが発生した後に行うため，リンク切れ発生前の過去のサイト構造を入手し，リンク経路の発見を行わなければならない．したがって，本実験では移動前フェーズでのリンク経路発見のために，インターネットアーカイブ[8] を使用した．その後，発見されたリンク経路を用いて，現在の Web サイト構造での移動先探索を行った．

4.1 実　験　1

本実験では次の 4 手法について，捕捉率の比較を行った．

手法 (a) 提案手法：3 章で説明した手法を用いてクローリングを行う．ただし，移動前フェーズのパラメータ max の値は，探索する移動前ページの URL に含まれるディレクトリ長と同一とする．

手法 (b) 簡略化手法：これは，提案手法を簡略化し，移動前フェーズの処理を除いたものである．具体的には，リンク経路とアンカ文字列の発見を行わず，移動後フェーズを実行するときには次のように設定する．(1) リンク経路として，サイトルートから移動前ページ URL u に向かってディレクトリが 1 つずつ増えるような単純な経路を 1 つ仮定する．具体的には，u = http://site/a/b/c/のとき，http://site/→ http://site/a/→ http://site/a/b/→ http://site/a/b/c/をリンク経路とする．(2) アンカ集合は空集合である．つまり，移動後フェーズでは幅優先探索のみが行われ，深さ優先には切り替わらないことになる．

手法 (c) 幅優先探索：Web サイトのルートページから探索を開始し，幅優先のクローリングを行う．

手法 (d) 深さ優先探索：Web サイトのルートページから探索を開始し，深さ優先のクローリングを行う．

4.2 結　　果

結果を**図 9** に示す．x 軸はアクセスページ数を，y 軸は移動先ページの捕捉率を表す．実験対象とした総移動ページ数を N（= 36），x ページ以下で発見できた移動先ページ数を $n(x)$ としたとき，x ページアクセス時の移動先ページの捕捉率 $f(x)$ は以下の式で計算する．

$$f(x) = \frac{n(x) \times 100}{N}$$

図 9 の (a') は，手法 (a) の移動後フェーズでアクセスしたページ数のみを考慮したときの捕捉率の状況である．また，(a) は，移動前フェーズでアクセスしたページ数も x に組み入れたときの値である．

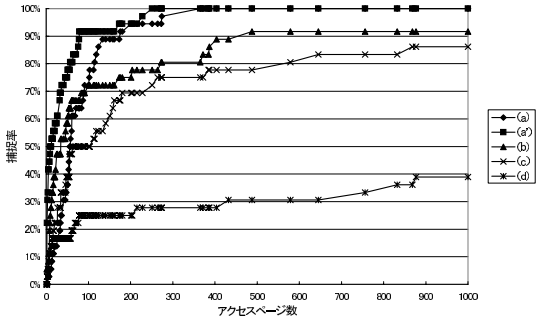

図 9 捕捉率の比較
Fig. 9 Comparison of capture rates.

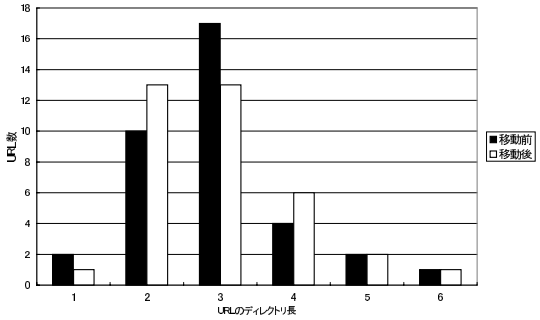

図 10 各 URL のディレクトリ長の分布
Fig. 10 Distribution of lengths of url directories.

図 9 から次のことが分かる．(1) 単純な深さ優先探索（手法 (d)）より，単純な幅優先探索（手法 (c)）の方が捕捉率が高い．この理由を次に説明する．**図 10** は，実験セットに含まれる移動元ページと移動先ページの各 URL のディレクトリ長の分布を調べたものである．たとえば，URL が http://site/a/b/c/の場合，ディレクトリ長は 4 となる．図 10 から分かるように，全体的にディレクトリ長が短い．したがって，実験セットにおいて，移動先ページが比較的ルートページに近い場所に存在していたといえる．その結果，幅優先探索による捕捉率が比較的高かったと考えられる．(2) 簡略化方式（手法 (b)）は手法 (c)，(d) より捕捉率が高い．これは，移動先ページはリンク経路の周辺にあるという我々の仮説が正しいことを示している．(3) 提案方式（手法 (a)）は，移動前フェーズのための初期コストがかかるものの，簡略化方式（手法 (b)）よりも捕捉率が高い．これは，より正確なリンク経路とアンカ文字列を移動前フェーズで発見する価値があることを示している．

提案手法は，前処理の結果としてパスとアンカ文字列のデータを記録しておく必要がある．**図 11** は，実験 1 で提案手法を実行する際に，監視対象 1 ページを処理するために記録する必要があったデータのサイズについてまとめたものである．図中の URL 数とは，記録したパスに含まれる URL の数である．これらの値の間には，総データ量 = URL 数 × URL の平均長 + アンカ数 × アンカ文字列の平均長，という関係がある．提案手法以外では，保存しておくべき情報は監視対象となる URL の情報だけなので，URL の平均長である 37 バイトが必要となるデータ量である．

URL 数	4.6
URL 平均長（バイト）	37
アンカ数	6
アンカ文字列平均長（バイト）	10.5
総データ量（バイト）	233.2

図 11 保存データ量（監視対象 1 ページあたり平均）
Fig. 11 Sizes of stored data.

いずれの手法に関しても，ページアクセス数（すなわち総 http リクエスト数）のうち，5%程度の 404 エラーを返した．

4.3 実 験 2

本実験では，移動前フェーズにおける関数 PathGrow の呼び出し回数の最大値 max の影響を評価した．具体的には，max を各移動前ページの URL のディレクトリ長 +0，+5，+10（それぞれを d + 0，d + 5，d + 10 と表記）として，それぞれ移動前フェーズと移動後フェーズを実行し，探索結果を比較した．

4.4 結 果

移動前フェーズで，max を d + 0，d + 5，d + 10 と変更してリンク経路の探索を行ったところ，発見されたリンク経路の数は**図 12** のような結果になった．図 12 の横軸は，対象となった 36 ページに番号を振り並べたものである．これらの各ページについて，max = d + 0，d + 5，d + 10 とそれぞれで発見されたリンク経路のパス数を棒グラフとして表している．各ページによって発見されたリンク経路数は異なるが，max の値が増えると，発見されるリンク経路の数も増えることが分かる．

次に，各 max の値で発見されたリンク経路を使用して，移動後フェーズのクローリングを実行した．その結果が**図 13** である．max = d + 0，d + 5，d + 10 において，捕捉率の差はほとんどない．これは，移動前フェーズにおけるリンク経路の発見において，有効なリンク経路を優先して発見できていることを示している．すなわち，リンク経路発見アルゴリズムのポイントである論理的上位ページらしさの計算がうまく働いており，重要なリンク経路が発見できれば，必ずしもすべてのリンク経路を発見する必要はないことが分かる．

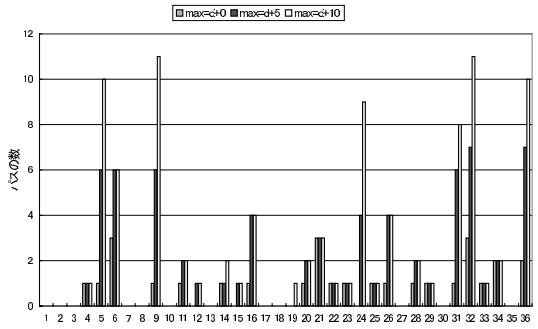

図 12 発見リンク経路の数
Fig. 12 Numbers of discovered link paths.

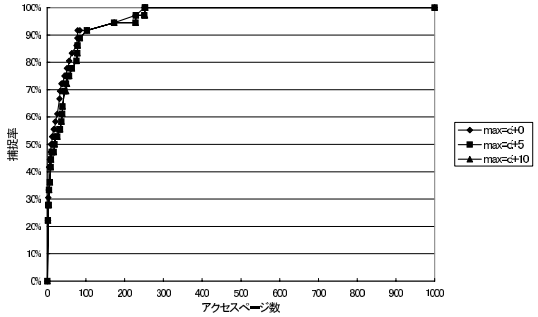

図 13 各 max の値による移動先捕捉率（移動後フェーズのみ）
Fig. 13 Capture rates for different max values (only the post-movement phase is considered).

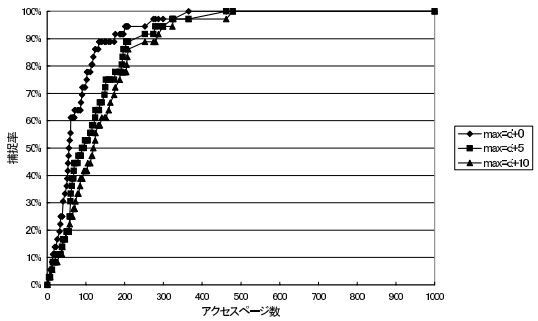

図 14 各 max の値による移動先捕捉率（移動前フェーズを含む）
Fig. 14 Capture rates for different max values (the pre-movement phase is also considered).

図 14 は，移動後フェーズのアクセスページ数に移動前フェーズでのアクセスページ数を加えて捕捉率を計算したものである．移動後フェーズでの捕捉率に違いがないため，max の値が大きいほうが，移動前フェーズでのアクセスページが増加する分不利になる．

以上のことから，max は $\mathrm{d}+0$，すなわち移動前ページ URL のディレクトリ長の値で，十分有効なリンク経路を発見できることが分かった．

5. 関 連 研 究

Web リンク切れの対応策・システムに関する研究．リンク切れの問題が深刻な問題として認識され[9]，その修復のためのアプローチについても研究が進められてきているが[10]，リンク切れ修復のためのソフトウェアによる支援技術は未成熟である．現在，すでに実用レベルのものとしては，リンク切れを発見し報告するためのソフトウェア[11] がある．このようなソフトウェアが数多く存在することから，リンク切れ問題の重要性が認識されていることは間違いない．また，管理しているリンク集にリンク切れが生じると，リンク先の Web サイトの管理者に電子メールを送信するような Web サイトも存在する[12]．以上のように，リンク切れを発見し報告するソフトウェアは数多く存在するが，その後の処理は利用者に任されており，リンク切れの自動修正もしくは修正支援システムはまだ実用には至っていない．

研究レベルでは，Web のリンク切れの問題に対処するため，これまで様々な手法が提案されてきた．Peridot は IBM によって開発されたソフトウエアツールであり，彼らが特許出願した手法[13),14)] を利用している．Peridot は WISH システムと同様に，リンク切れした Web リンクに対して移動先を発見しようと試みる．具体的には，Peridot は各 Web ページのコンテンツから求められる**フィンガープリント**（*fingerprints*）と呼ばれる情報を用いて，ページの移動先らしさを計算する．したがって，この手法では，移動先候補となる Web ページのフィンガープリントがあらかじめデータベースに格納されていなければならない．Peridot のほかにも，Phelps と Wilensky は**レキシカルシグネチャ**（*lexical signatures*）を利用して移動ページを発見することを提案している[15]．これは，各 Web ページごとに用意された小さなキーワード集合であって，インデックスサーバに投入したときにその Web ページが高い確率で上位に来るよう注意深く選ばれたものである．

以上の Peridot とレキシカルシグネチャによるアプローチでは，ページの移動先を発見するために，移動先 Web ページの情報が，あらかじめ何らかのデータベースに格納されている必要がある．本稿で提案しているクローリング手法は，これらのデータベースを効率的に構築するために利用可能である．したがって，本クローリング手法は，我々の WISH システムだけに適用可能な特殊な手法でなく，他の研究に対しても組み合わせて利用可能な汎用性の高い手法であるとい

える．

リンク切れに対処する他のアプローチ． Ashman は，WWW およびハイパーテキストシステムにおいてリンクの一貫性を管理するための戦略についてサーベイを行っている[16]．そこでの分類によると，WISH システムのアプローチは *corrective* アプローチに分類される．これは，リンク切れが起こった後に，代わりとなるリンクを探すアプローチのことである．corrective アプローチのほかにも，必要なときにだけ動的にリンクを実体化する *adaptive* アプローチ[17] や，更新を許可しないなどによりリンク切れ自体を起こらないようにする *preventive* アプローチ[18] が提案されてきた．これらのアプローチはそれぞれ異なる特性を持つため，与えられた状況によって適切にアプローチを使い分けることが望ましい．たとえば，あるリンクの目的が，内容が固定されたコンテンツへのアクセスである場合，preventive アプローチは有効であると考えられる．一方，大学研究室の一覧リンク集やブックマークなどのように，リンク先が内容が固定されたコンテンツでなく，更新されうるサイトなどへのアクセスを目的としている場合には，corrective アプローチが有効であるといえる．本クローリング手法は，特に corrective アプローチの実現に有効な手段を提供するものである．また，WWW の文脈では，キャッシュやインターネットアーカイブの情報を，preventive アプローチの実現や，corrective アプローチで移動先が見つからない場合のリンク先として利用することができる．

Web を対象としたクローリング手法の研究． Web を対象としたクローリング手法の研究は数多くある．たとえば，インデックスサーバのためのクローリングとして，PageRank の高いページから優先して収集する手法[19] や，更新された Web ページを優先的に収集する手法[20] などがある．また，特定の用途に特化したクローリングとして，地域情報収集のためのクローリング手法[21] などがある．これらのクローリング手法では基本的に，Web の物理的なグラフ構造に主に着目して，ページ訪問の順序を決定している．たとえば，Web ページを表すノード間の辺の数などに着目して，ページの訪問順序を決定する．それに対して，本手法では，Web サイトの論理的な構造の抽出を試み，その構造との関連で順序を決定する．その理由は，Web ページの「移動していそうな場所」は，その論理構造に依存すると考えられるからである．本稿の実験結果は，この仮説が間違っていないことを示している．

6. まとめと今後の課題

本稿では，同一サイト内で移動した Web ページを発見するための効率的なクローリング手法を提案した．本手法では，クローリングの手がかりとして，リンク経路とアンカ文字列の情報を用いる．実験を行った結果，本手法は他の手法よりも少ないページアクセス量で移動先ページを捕捉することを確認できた．したがって，提案手法は移動先の発見に有効であるといえる．今後の課題としては，探索順序に関してさらなる工夫を行い，より効率の良いアルゴリズムを開発することや，本手法を拡張して，同一サイト以外へのリンクにも対応したアルゴリズムを開発することなどがある．

謝辞 ゼミなどでご議論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（#18049005，#19024006）による．

参 考 文 献

1) GVU's WWW User Surveys. http://www.gvu.gatech.edu/user_surveys/
2) Dellavalle, R.P., Hester, E.J., Heilig, L.F., Drake, A.L., Kuntzman, J.W., Graber, M. and Schilling, L.M.: INFORMATION SCIENCE: Going, Going, Gone: Lost Internet References, *Science*, Vol.302, No.5646, pp.787–788 (2003).
3) 中溝昌佳，森嶋厚行，有山智洋，杉本重雄，北川博之：WWW コンテンツ一貫性維持のためのリンク更新機構の提案，日本データベース学会 Letters，Vol.2, No.2, pp.65–68 (2003).
4) 中溝昌佳，森嶋厚行，杉本重雄，北川博之：WWW における信頼度の高いリンクの発見，情報処理学会研究報告，Vol.2004, No.72, pp.397–402 (2004).
5) 中溝昌佳，森嶋厚行，杉本重雄，北川博之：WWW リンク一貫性維持支援システムにおけるリンク切れ自動修復，日本データベース学会 Letters，Vol.3, No.3, pp.5–8 (2004).
6) 中溝昌佳，飯田敏成，森嶋厚行，杉本重雄，北川博之：Web リンク切れの自動修正における信頼度の高いリンク情報の利用，電子情報通信学会第 16 回データ工学ワークショップ（DEWS2005）講演論文集，No.4B-i4 (2005).
7) Morishima, A., Nakamizo, A., Iida, T., Sugimoto, S. and Kitagawa, H.: Automatic Correction of Broken Web links: Ideas, Experiments, and Lessons Learned. (submitted)
8) Internet Archive. http://www.archive.org/

9) Ashman, H. and Davis, H.: Panel missing the 404: link integrity on the world wide web, *Computer Networks*, Vol.30, No.1–7, pp.761–762 (1998).
10) Davis, H.C.: Hypertext link integrity, *ACM Comput. Surv.*, Vol.31, 4es, No.28 (1999).
11) Xenu's Link Sleuth. http://www.cs.washington.edu/lab/sw/LinkSleuth.html
12) Huxley, L., Place, E., Boyd, D. and Cross, P.: Planet sosig — a spring-clean for sosig: A systematic approach to collection management (2002). http://www.ariadne.ac.uk/issue33/planet-sosig/
13) Beynon, M. and Flegg, A.: Hypertext request integrity and user experience, US Patent Application Publication, US 2004/0267726 A1 (2004).
14) Beynon, M. and Flegg, A.: Guaranteeing hypertext link integrity, US Patent Application Publication, US 2005/0021997 A1 (2005).
15) Phelps, T. and Wilensky, R.: Robust hyperlinks: Cheap, everywhere, now, *DDEP/PODDP 2000*, pp.28–43 (2000).
16) Ashman, H.: Electronic document addressing: Dealing with change, *ACM Comput. Surv.*, Vol.32, No.3, pp.201–212 (2000).
17) Tanaka, K., Nishikawa, N., Hirayama, S. and Nanba, K.: Query pairs as hypertext links, ICDE 1991, pp.456–463 (1991).
18) Ingham, D.B., Caughey, S.J. and Little, M.C.: Fixing the "broken-link" problem: The w3objects approach, *Computer Networks*, Vol.28, No.7–11, pp.1255–1268 (1996).
19) Cho, J., Garcia-Molina, H. and Page, L.: Efficient Crawling Through URL Ordering, *Computer Networks*, Vol.30, No.1–7, pp.161–172 (1998).
20) 熊谷英樹，山名早人：リンク構造を利用したWebページの更新判別手法，電子情報通信学会第15回データ工学ワークショップ（DEWS2004）講演論文集，No.5-B-02 (2004).
21) 張　建偉，石川佳治，黒川沙弓，北川博之：地域ウェブ情報源の収集のためのクローリング手法の提案，電子情報通信学会第16回データ工学ワークショップ（DEWS2005）講演論文集，No.4B-i5 (2005).

(平成18年12月21日受付)
(平成19年 1 月26日採録)

（担当編集委員　安形 輝）

澤　菜津美（学生会員）

2006年筑波大学図書館情報専門学群卒業．現在，同大学大学院図書館情報メディア研究科博士前期課程に在籍．WWW応用に興味を持つ．日本データベース学会学生会員．

森嶋　厚行（正会員）

1993年筑波大学第三学群情報学類卒業．1998年同大学大学院工学研究科修了．博士（工学）．1998〜2001年日本学術振興会特別研究員．1999〜2000年AT&T Labs-Research客員研究員．2001年芝浦工業大学工学部情報工学科講師．現在，筑波大学大学院図書館情報メディア研究科/知的コミュニティ基盤研究センター准教授．情報統合技術，XML/WWWデータベース，メタデータデータベース等に興味を持つ．ACM，IEEE-CS，電子情報通信学会，日本データベース学会各会員．

飯田　敏成

2005年筑波大学図書館情報専門学群卒業．2007年同大学大学院図書館情報メディア研究科博士前期課程修了．現在，株式会社日立製作所勤務．

杉本　重雄（正会員）

1977年京都大学工学部情報工学科卒業．1982年同大学大学院工学研究科博士後期課程情報工学専攻単位取得退学．京都大学工学博士．1982年京都大学工学部助手．1983年図書館情報大学図書館情報学部助手，助教授，教授を経て2002年より筑波大学大学院図書館情報メディア研究科/知的コミュニティ基盤研究センター教授，現在に至る．ディジタルライブラリ，メタデータに関心を持つ．Dublin Core Metadata Initiativeの評議委員会委員．ACM，IEEE-CS，電子情報通信学会，日本ソフトウェア科学会，日本データベース学会ほか会員．

北川 博之（フェロー）

1978 年東京大学理学部物理学科卒業．1980 年同大学大学院理学系研究科修士課程修了．日本電気（株）勤務の後，1988 年筑波大学電子・情報工学系講師．同助教授を経て，現在，筑波大学大学院システム情報工学研究科教授，ならびに計算科学研究センター教授．理学博士（東京大学）．異種情報源統合，XML とデータベース，WWW の高度利用，データマイニング等の研究に従事．著書『データベースシステム』（昭晃堂），『The Unnormalized Relational Data Model』（共著，Springer-Verlag）等．日本データベース学会副会長，電子情報通信学会フェロー，ACM，IEEE-CS，日本ソフトウェア科学会各会員．