ウェブサーバへの最短訪問間隔を保証する 時間計算量が O(1)のウェブクローリングスケジューラ

森本 浩介(1)　上田 高徳(1),(2)　打田 研二(1)　山名 早人(3),(4)

(1) 早稲田大学大学院 基幹理工学研究科 〒169-8555 東京都新宿区大久保 3-4-1
(2) 早稲田大学 メディアネットワークセンター 〒169-8050 東京都新宿区戸塚町 1-104
(3) 早稲田大学 理工学術院 〒169-8555 東京都新宿区大久保 3-4-1
(4) 国立情報学研究所 〒101-8430 東京都千代田区一ツ橋 2-1-2

E-mail: {morimoto, ueda, k_uchida}@yama.info.waseda.ac.jp, yamana@acm.org

あらまし ウェブクローリングを行う際に, ウェブサーバに短い間隔で連続アクセスするとウェブサーバに負荷を掛けてしまうため，ウェブクローラは同一サーバに対するクロール間隔を十分に空けなければならない．しかし，ウェブに存在する大量のサーバごとにクロール間隔をスケジューリングしながら高速なクローリングを実現する方法は計算量の観点から自明ではない．本論文では，クローリング待ちの URL 数が閾値を超えた場合に URL を破棄することにより，時間計算量が O(1)で，空間計算量の上限が URL 数に依存しないスケジューリングアルゴリズムを提案する．提案アルゴリズムはウェブサーバに対するアクセスの最短間隔を任意に設定でき，空間計算量とクローリング網羅率のトレードオフを調整できる．提案手法は URL を破棄するため，クローリングできないページが発生する可能性があるが，PageRank の高いページを取りこぼさない特色がある．提案手法を評価した結果，既存手法に対して空間計算量が 10 分の 1 のときでも，クローリングできたページの PageRank 総和は PageRank 全体の 80%になり，PageRank の高いページを優先的に収集可能なことを確認できた.

キーワード ウェブクローラ，スケジューリング

An O(1) Time Complexity Web Crawling Scheduler with Guarantee of Minimum Interval of Accesses to a Web Server

Kosuke MORIMOTO(1)　Takanori UEDA(1),(2)　Kenji UCHIDA(1)　and　Hayato YAMANA(3),(4)

(1) Graduate School of Fundamental Science and Engineering, Waseda University, 3-4-1 Okubo, Shinjuku-ku, Tokyo, 169-8555 Japan
(2) Media Network Center, Waseda University, 1-104 Totsuka-cho, Shinjuku-ku, Tokyo, 169-8050 Japan
(3) Science and Engineering, Waseda University, 3-4-1 Okubo, Shinjuku-ku, Tokyo, 169-8555 Japan
(4) National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, 101-8430 Japan

E-mail: {morimoto, ueda, k_uchida}@yama.info.waseda.ac.jp, yamana@acm.org

1. はじめに

昨今, ウェブ上のデータを用いた様々な研究 [1][2][3] や，過去のウェブページの閲覧を可能にするサービス [4]，ウェブ上のデータによりキャンペーン効果の測定や市場調査を行うサービス [5] が行われており，ウェブ上のデータに対する需要は高まる一方である．このような研究やサービスを行うにあたり，ウェブを巡回してウェブ上に存在するデータを収集するウェブクローラは必要不可欠な存在である．nutch [6] や YaCy [7] のようなオープンソースのウェブクローラ・検索エンジンの開発が盛んであり，ウェブクローリングそのものを提供するサービス [8] もある．旧来は商用検索エンジンの構築に用いられるソフトウェアであったウェブクローラは，今後，研究とビジネスの両分野において広く用いられるようになると考えられる．

ウェブクローラは，(1) ウェブページをダウンロードする，(2) ダウンロードしたウェブページからリンク先 URL を抽出する，(3) リンク先 URL のうち，未収集の URL を新たにダウンロードする，という 3 つの動作を繰り返すことでウェブ上のデータの網羅的な収集を行う．ウェブは広大なため，通信帯域を有効利用して可能な限り高速に広範囲のウェブページを収集する必要がある．しかし同時に，各ウェブサーバへの負荷を最小限にする必要もある．ウェブクローラが同一ウェブサーバに連続アクセスするとサーバに大きな負荷を掛け，最悪の場合，ウェブサーバが停止してしまう可能性がある．従ってウェブクローラは，同一ウェブサーバへの連続アクセスを避けながら，高速にクローリングするという目的を達成しなければならない．

上記の目的を達成するためには，クローリング中に発見した URL を管理して同一サーバへのアクセス間隔を効率よく制御，すなわちスケジューリングする必要がある．しかし，空間計算量と時間計算量の観点から効率の良いスケジューリングアルゴリズムは自明ではない．これまでの既

存研究では，優先順位付きキューを用いた手法 [9][10][11][12] が中心であり，優先順位付きキューはその時間計算量と空間計算量がキューに挿入された URL 数に依存して増加する問題がある．Google によると，インターネット上に存在する URL 数は 2008 年 7 月の時点で 1 兆を突破している [13]．従って，URL 数に依存して時間計算量と空間計算量が増加するアルゴリズムを使うことは望ましくない．

2008 年に Texas A&M 大学の Lee らはウェブクローラの IRLbot [14] を提案した．IRLbot で用いられるスケジューリングアルゴリズムは，URL に対するハッシュ計算とリンク数のカウント，key-value DB の検索，キューに対する操作で実現しており，時間計算量は少ないものの，空間計算量は URL 数に依存する．

そこで本論文では，Lee らの手法 [14] を修正し，クローリング待ちの URL 数が閾値を超えた時に URL を破棄することで，以下の特徴を実現したスケジューリングアルゴリズムを提案する．

- 同一ウェブサーバへのアクセス間隔の最小値を厳密に保証
- 時間計算量が O(1)
- 空間計算量の上限がユーザ指定のパラメータのみに依存し，クローリング対象の URL 数に依存しない
- 空間計算量とクローリング網羅率のトレードオフをユーザが与えるパラメータで制御可能
- PageRank[15] が高いウェブページを優先的に収集

提案手法は空間計算量の上限を 2 つのパラメータで制御可能であり，ウェブクローラの実行環境に応じて使用する物理メモリ使用量の調整ができる．提案手法は URL を破棄することで空間計算量の上限を保証するため，物理メモリ使用量を削減するとクローリングできないページが発生する．すなわち，空間計算量とクローリング網羅率はトレードオフの関係になる．ただし本手法は，被リンク数が多いページであれば，クローリングできる可能性が高いという特色をもつ．被リンクが多いページは PageRank が高いと考えられ，提案アルゴリズムは PageRank の値を計算せずとも，URL の破棄操作のみで PageRank の高い URL を優先的に収集可能なアルゴリズムといえる．

本論文は以下の構成をとる．まず，2 節で関連研究について説明し，3 節で提案手法の O(1)スケジューラを述べる．4 節で実験結果について述べ，5 節でまとめる．

2. 関連研究

本節では，既存研究で論じられたウェブクローラのスケジューリングアルゴリズムについてまとめる．まず，1994 年に提案された最初期のウェブクローラ [16][17][18] について説明する．[16][17][18] では，ソフトウェアとしてのデザイン方法，ウェブページを収集した後のインデックスの作成方法，データベースとの連携といったウェブクローラの基本的な構造が述べられているが，スケジューリングについて議論していない．また，この頃のウェブクローラの収集ページ数は数万から十数万であり，今日対象となる数百億から一兆ページには到底及ばないため，利用されていたスケジューリングアルゴリズムが現在のウェブに適するかは明らかではない．

前述したクローラの登場後，様々なウェブクローラが提案され，同時にスケジューリング手法も提案された．スケジューリング手法は，(1) 再収集に関する手法 [19][20], (2) 再収集に特化しないスケジューリング手法に分けられ，さらに再収集に特化しない手法は (2-1) ウェブ全体の収集が困難なためウェブページに優先順位を付ける手法 [21], (2-2) 高速化を目指す手法 [6][9][10][11][12][14][22] に分けられる．

高速なクローリングを目標としたウェブクローラは[14]以外は Peer to Peer や MapReduce などの分散処理によって実現している．これらのウェブクローラの収集規模は数千万から数億ページである．Heydon らの手法[9][10], Edwards らの手法[11]，Shkapenyuk らの手法[12]，は優先順位付きキューを用いてスケジューリングの機能を実現している．優先順位付きキューの時間計算量は，N を優先順位付きキューに格納されている要素数とすると $O(\log N)$ であるため，格納された URL 数が増えるにつれて時間計算量が増加してしまう問題がある．また，物理メモリが不足するほど膨大な URL を扱う場合に二次記憶装置を利用して優先順位付きキューを構築すると，実質的な計算時間は大きく悪化してしまう．

分散クローリングが一般的な中，IRLbot [14] は計算機 1 台で高速なクローリングを実現することを目指している．IRLbot は，STAR（Spam Tracking and Avoidance through Reputation）と BEAST（Budget Enforcement with Anti-Spam Tactics）という機構を用いてスケジューリングを行っている．STAR は被リンク数に基づいてクローリングの優先度を制御する機構であり，クローリング中に発見した PLD（pay-level domain：レジストラに費用を支払って取得するドメイン）単位で被リンク数をカウントし，被リンク数に応じて，同一 PLD x に対する単位時間あたりのダウンロード数 Budget B_x を決定する．スパムサイトに対するクローリングの抑制や人気のあるサイトに対する優先的なクローリングを実現するため，B_x の決定方法を複数提案している．一方 BEAST は，キューを用いて URL を保持し，ダウンローダへ URL の送出を行う機構であり，内部にクローリング待ちの URL を保持する複数のキューを持つ．1 つのキューは単位時間ごとに同時にクローリングできる URL 集合を保持する．BEAST は，STAR が決定した Budget に従い，同じ PLD x に属する URL を 1 つのキューにB_x個まで追加

Algorithm 1: Crawl

Global Variables
$Q = (q_0, q_1, \cdots, q_{\mathrm{B}-1} \mid q_i \; is \; a \; FIFO \; Queue)$

Input: $\mathcal{S}$: seed URLs for the crawling
1. $\mathcal{U} \leftarrow \mathcal{S}$
2. **loop**
3. **foreach** $u \in \mathcal{U}$
4. Download data d from u;
5. **if** u is a HTML URL **then**
6. $\mathcal{C} \leftarrow$ URLs linked from d;
7. Enqueue($\mathcal{C}$); // stores non-visited // URLs from $\mathcal{C}$
8. // Here, put d into storages or // something you want to do.
9. **end**
10. $\mathcal{U} \leftarrow$ Next(); // receives a new crawling URL set // from Q
11. **continue;**

する. キューが不足して追加できなかった URL は別途保持し，ダウンロード用のキューがすべて空になったあと読み出して，Budget に従いながら再びキューに追加する．

3. 提案手法：O(1) クローリングスケジューラ

本節では提案手法を，ウェブクローリングの最も基本的な動作であるデータのダウンロード手順を示しながら説明する. Algorithm 1 にウェブクローラの主ルーチンを示した. クローラは与えられた初期ページ URL 集合 $\mathcal{S}$，すなわちシードセットからクローリングを開始し，ダウンロードした HTML ページからリンク先 URL セット $\mathcal{C}$ を抽出する. この $\mathcal{C}$ はクローリング対象の URL 候補セットとなり，すでにクロール済みの URL を $\mathcal{C}$ から除去したものが，クローリング対象の URL セットになる．Algorithm 1 では，クローリング中に発見したリンク先 URL セットを Enqueue 関数により保存および重複除去し，ダウンロードが終了したら新たなクローリング URL セットを Next 関数により取得している. この 2 つの関数 Next と Enqueue が本アルゴリズムの主要関数である．

Algorithm 1 においては省略しているが，実際のクローリング時にはサーバのレスポンスタイムを償却するために，$\mathcal{U}$ 中の URL をマルチスレッドで並列にダウンロードすることになる．よって，同一のウェブサーバに連続でアクセスしないためには, $\mathcal{U}$ に同一のウェブサーバに属する URL が存在しないように $\mathcal{U}$ を構成する必要がある．Web 上の URL は膨大なため，時間計算量および空間計算量を抑えて，いかに効率よくこのような $\mathcal{U}$ を構成するかが課題となる. 提案スケジューリングアルゴリズムは，時間計算量がデータ量に依存する処理やデータ構造を利用せず，IP アドレスの剰余計算と配列へのアクセス，および FIFO キューに対する操作で処理を実現している．

3.1. データ構造とパラメータ

本節では，提案するアルゴリズムで用いるデータ構造とパラメータについて述べる. 本手法ではデータ構造として，

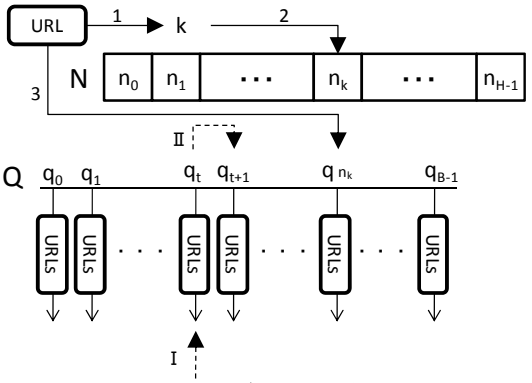

図 1. 提案手法の流れ

長さ $\mathrm{H} = 2k \; (k \geq 1)$ の整数型配列 $N = (n_0, n_1, \cdots, n_{\mathrm{H}-1})$ と，B 個の FIFO キュー $Q = (q_0, q_1, \cdots, q_{\mathrm{B}-1})$ の 2 つを用いる．配列 N は，$n_{2j} = 0, \; n_{2j+1} = 1 \; (0 \leq j \leq \mathrm{H}/2 - 1)$ と初期化する．ここで，配列長 H とキューの個数 B はユーザが設定するパラメータとなる．またユーザは，同一 IP アドレスに対する最小のクローリング間隔 T を設定する. 言い換えると，本アルゴリズムは同一 IP アドレスに対するアクセス間隔が必ず $\mathrm{T} + \varepsilon \; (\varepsilon \geq 0)$ となることを保証する．

3.2. O(1) スケジューリングアルゴリズム

ここで，我々の提案手法である O(1) クローリングスケジューラのアルゴリズムについて述べる．提案アルゴリズムは 2 つの関数，Enqueue と Next で構成される．Enqueue 関数の疑似コードを Algorithm 2, Next 関数の疑似コードを Algorithm 3 に示した．また，提案アルゴリズムの流れを図 1 に示した. アラビア数字を併記した実線の矢印が Enqueue 関数に相当し，ローマ数字を併記した破線の矢印が Next 関数に相当する．Enqueue 関数は URL を発見するたびに実行され，(1) URL を$0 \leq k \leq \mathrm{H} - 1$ の整数に変換し，(2) n_k を取得したあと，(3) キューに URL を挿入する．Next 関数はダウンロード中の URL 集合が空になると実行される．Next 関数は呼ばれた回数をステップ t として記憶し，アクセス間隔を制御するため，(I) 最後に Next が呼ばれてから時間 T が経過するまで待ち，t をインクリメントしたあと，(II) ステップ t において，$q_p \; (p \equiv t \; (\mathrm{mod} \; \mathrm{B}))$ をダウンローダに出力する．ここで，q_p 中には，同一のホスト IP アドレスを持つ URL が含まれないことが保証される. 以下，Enqueue 関数と Next 関数を詳しく説明する．

Enqueue 関数では，URL のホスト IP アドレス a を整数に変換し，剰余 $r \equiv a \; (\mathrm{mod} \; \mathrm{H})$ を求め n_r の値を取得する（Algorithm 2 の 4 行目）．ここで，$t < n_r < t + \mathrm{B}$ の場合，

$$Q' = \left\{ q_p \;\middle|\; \begin{matrix} p \equiv n_r \; (\mathrm{mod} \; 2), \\ p \equiv i \; (\mathrm{mod} \; \mathrm{B}), \quad t < i < n_r \end{matrix} \right\}$$

のキューそれぞれには, 同じ剰余値 r を持つクローリング待機中の URL がすでに 1 つずつ存在することを意味する．したがって，$q_p \; (p \equiv n_r \; (\mathrm{mod} \; \mathrm{B}))$ が URL の追加対象のキューとなる．

Algorithm 2: Enqueue

Global Variables
$Q = (q_0, q_1, \cdots, q_{\mathrm{B}-1} \mid q_i \ is\ a\ FIFO\ Queue)$
$N = (n_0, n_1, \cdots, n_{\mathrm{H}-1} \mid n_{2j} = 0,\ n_{2j+1} = 1)$
B: the number of FIFO queues **initialized by the user**
H: the length of the array **initialized by the user**
t : current step **initialized by** 0

Input: $\mathcal{C}$: HTML URLs required to be enqueued
1 **foreach** $c \in \mathcal{C}$
2 **if** c was already visited **then**
3 **continue**;
4 $r \leftarrow$ The IP address of the server maintaining c (mod H);
5 **if** $n_r \geq t + B$ **then**
6 **continue**;
7 **if** $n_r \leq t$ **then**
8 $n_r \leftarrow t + 1 + \{(t \ \mathbf{and}\ 1)\ \mathbf{xor}\ (n_r \ \mathbf{and}\ 1)\}$;
9 $i \leftarrow n_r \pmod{\mathrm{B}}$
10 $q_i \leftarrow q_i \cup c$;
11 $n_r \leftarrow n_r + 2$;
12 **end**

Algorithm 3: Next

Global Variables
$Q = (q_0, q_1, \cdots, q_{\mathrm{B}-1} \mid q_i \ is\ a\ FIFO\ Queue)$
B: the number of FIFO queues **initialized by the user**
T: crawling interval **initialized by the user**
τ: last Next function call time
initialized by the crawling start time
t : current step **initialized by** 0

Output: q_p: URLs allowed to be downloaded in step t

1 **if** $\tau + \mathrm{T} < current\ time$ **then**
2 **wait** until $\tau + \mathrm{T} \geq$ current time become **true**;
3 $t \leftarrow t + 1$;
4 $p \leftarrow t \pmod{\mathrm{B}}$;
5 $\tau \leftarrow$ current time;
6 **return** q_p;

ただし，$n_r \geq t + \mathrm{B}$ の場合，同一の剰余値を持つクローリング待ちの URL が B 個を超え，キューが不足していることになる．本アルゴリズムでは，キューが不足した場合，単に URL を破棄する（Algorithm 2 の 6 行目）．URL を破棄するため，本アルゴリズムでは収集できない URL が発生する可能性がある．B を大きくすると URL の破棄が減るため，クローリングの網羅率を向上できるが，空間計算量は大きくなる．反対に B を小さくすると破棄される URL が増え，空間計算量は小さくなる．つまり，空間計算量とクローリング網羅率はトレードオフの関係になる．ここで，被リンク数が多い，すなわち PageRank が高い URL はクローリング中に出現する回数が多いと考えられ，破棄したあとに再度出現することが期待できる．逆に，破棄されたあと再出現しなかった URL は被リンク数が少なく PageRank も低いと考えられる．このことから本手法は，URL を破棄する操作のみで PageRank が高い URL を優先的に収集することが期待できる．B と PageRank の関係は実験の節において示す．

また，$n_r \leq t$ の場合，剰余値 r を持つクローリング待ちの URL は存在しないことになる．この場合，q_p $\left(p \equiv n_r \pmod{\mathrm{B}}\right)$ に追加すると，クローリングされるまでの間隔が空きすぎる．そこで，アクセス間隔を守りながらクローリングのスループットを可能な限り速めるために，$n_r \leq t$ の場合には n_r を

$$n_r = \begin{cases} t+2\ (r \text{ is even}), & t+1\ (r \text{ is odd}) \quad t \text{ is even} \\ t+1\ (r \text{ is even}), & t+2\ (r \text{ is odd}) \quad t \text{ is odd.} \end{cases}$$

と更新したうえでキューに挿入し，直後のステップでダウンロードを行うようにする．偶奇分けしているのは，同一の剰余値 r を持つ URL がキューにおいて 1 つ飛びに挿入されるようにするためである．これにより，アクセス間隔の保証ができる．たとえば，q_t のダウンロードが完了した直後に，q_{t+1} のダウンロードが開始されたことを考える．ここで，q_t のうち最後にダウンードされた URL と q_{t+1} で最初にダウンロードされた URL のホスト IP アドレスが等しい場合，アクセス間隔を守っていない．したがって，剰余値 r が等しい URL は 1 つ飛びに挿入する必要がある．Algorithm 2 ではビット演算でこの更新を行い，計算速度の向上を図っている（Algorithm 2 の 8 行目）．

キューに URL を追加したあと，n_r には 2 を加える．この 2 を加える処理も，1 つ飛びに挿入するためである．なお，Enqueue 関数では剰余値 r ごとに，次に追加するべきキューのインデックスを配列 N に保持している．このため，異なるウェブサーバの URL でも剰余値 r が等しければ，n_r の更新で衝突が発生する．その結果，アクセス間隔の正誤差 ε が発生する可能性がある．ただし，配列長 H を大きくすることで，正誤差を小さくすることができる．

以上までで，クローリング URL セットの構成は完了しており，Next 関数では，ステップ t をインクリメントし，クローリング URL セットを返せばよい．ただし，アクセス間隔を保証するために，最後に Next が呼ばれてから時間 T が経過していなければ，経過するまで待機する．その後 t をインクリメントしたあと，グローバル変数 τ に現在時刻を記録し，q_t を返す．

以上をまとめると，提案アルゴリズムは，H を増やすことでアクセス間隔の正誤差 ε を減らすことができ，B を増やすことでクローリングの網羅率を上げることができる．ただし，H か B を大きくすると空間計算量も大きくなるため，ユーザは必要な性能に応じて H と B を設定することになる．また，提案手法は URL を破棄するため，収集できない URL が発生する可能性がある．しかし，被リンク数が多い URL は破棄したあとに再度出現することが期待できる．被リンク数が多いページは PageRank も高いという特徴から，提案アルゴリズムは PageRank が高いページを失わずに収集することができると期待できる．本性質は実験の節において確認する．

表 1. IRLbot と提案手法の比較

	IRLbot	提案手法
リンク数のカウント	する	しない
同一 IP からの同時クローリング	する	しない
URL の破棄	しない	する
二次記憶装置の使用	可能性有	しない

3.3. 時間計算量と空間計算量

本節では提案手法の時間計算量と空間計算量を見積もる．本スケジューリングアルゴリズムは剰余計算と配列へのアクセス，および FIFO キューに対する操作で実現している．一般的に剰余計算と配列へのインデックスによるアクセス，FIFO キューの先頭に対する追加・削除はすべて $O(1)$ である．従って，本アルゴリズムは $O(1)$ の時間計算量で実行可能である．

次に本アルゴリズムの空間計算量を見積もる．挿入対象の q のインデックスを保持する配列 N の空間計算量は $O(H)$ である．また，同一の剰余値 r をもつ URL は 1 つおきに挿入されるため，ある時刻に FIFO キューに挿入されている同一の剰余値を持つ URL の数の上限は $B / 2$ である．よって，剰余値の異なり数が H であることに注意すれば，空間計算量の上限は $O(BH)$ となる．以上より，提案手法の時間計算量は $O(1)$ であり，空間計算量の上限は $O(BH)$ である．B と H がユーザ指定の定数であることを考えれば，空間計算量の上限も $O(1)$ として扱える．

3.4. 既存手法との比較

1 節で述べたように，本手法は IRLbot の STAR と BEAST をもとにしたアルゴリズムである．本節では提案手法の各プロセスと既存手法 IRLbot の比較を行う．表 1 に IRLbot と提案手法の比較を示した．

IRLbot の STAR は，PLD 単位でリンク数をカウントし，被リンク数に応じて PLD に対する同時クローリング数 Budget を決定している．BEAST は，STAR によって決定された Budget に応じて URL をキューに振り分ける．キューが不足してキューへの挿入できなかった URL は別途保持し，キューにある URL のクローリングが終了したのち，再度 Budget に基づいてキューを再構築する．

対して提案手法では，リンク数のカウントは行わず，IP アドレスをもとに整数に変換し，挿入すべき FIFO キューを決定する．また，FIFO キューへの挿入が認められない URL は破棄する．URL を破棄することにより，収集できない URL が発生する点がデメリットである．しかし，被リンク数が多いウェブページ，すなわち PageRank が高い URL は，URL を破棄したあとに再出現すると期待できる．次の 4 節において，URL を破棄する影響を確認した実験について述べる．

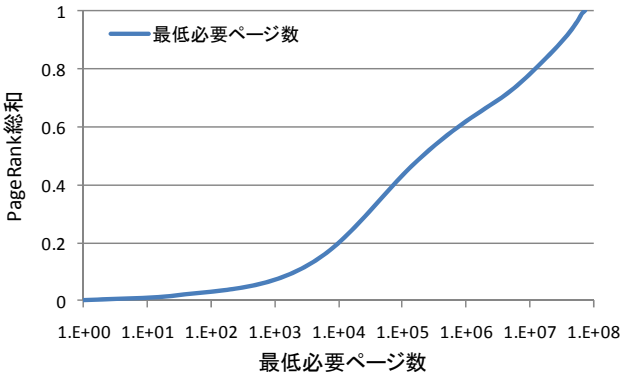

図 2. PageRank 総和と最低必要ページ数

4. 評価実験

本節では，提案手法を用いて行った実験の方法，使用したデータセットについて述べ，実験結果の考察を行う．実験はクローリングシミュレーションで行う．現実のウェブ空間と同等の環境で実験を行うため，Laboratory for Web Algorithm [23] で公開されているウェブグラフデータを用いてシミュレーションを行った．

4.1. 実験方法

本論文では以下の 3 つの実験を行った．

実験1. パラメータ B の変化によるクローリングページ数と PageRank 総和の変化の評価
実験2. パラメータ H の変化によるクローリングページ数と PageRank 総和の変化の評価
実験3. クローリングの進捗にともなう PageRank 総和の増加の評価

実験 1 ではパラメータ H を固定し，パラメータ B を変化させた場合のクローリングページ数と PageRank 総和の変化を確認した．実験 2 では実験 1 とは逆にパラメータ B を固定し，パラメータ H を変化させた場合のクローリングページ数と PageRank 総和の変化を確認した．実験 1 と実験 2 はパラメータ B とパラメータ H それぞれの，クローリング網羅率と PageRank 総和に対する影響を確認するために行った．

実験 3 は，すべてを網羅的に収集する従来手法と URL を破棄する提案手法において，クローリングの進捗につれて PageRank の総和がどのように増加するかを確認するために行った．提案手法がもつ，PageRank の高いウェブページを優先的に収集するという性質の検証が目的である．

4.2. データセット

本実験では，Laboratory for Web Algorithm [23] で公開されているウェブグラフデータのうち uk-2006-05 を用いてクローリングシミュレーションを行った．まず，クローリングを開始するシードを選択するために，uk-2006-05 のグラフデータから強連結成分を大きい順に 1024 個抽出した．そして，その強連結成分それぞれから 1 つずつシードページ

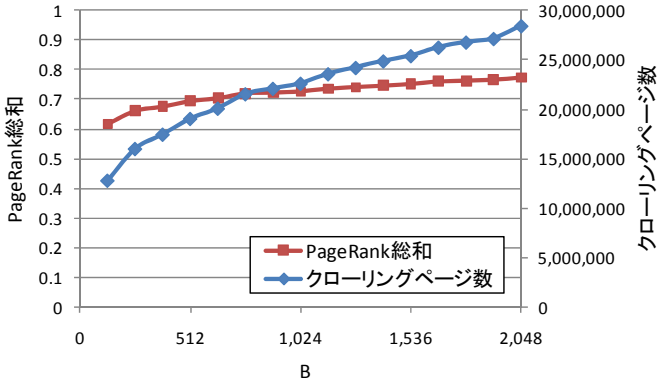

図 3. パラメータ B を変化させたときの
クローリングシミュレーション結果

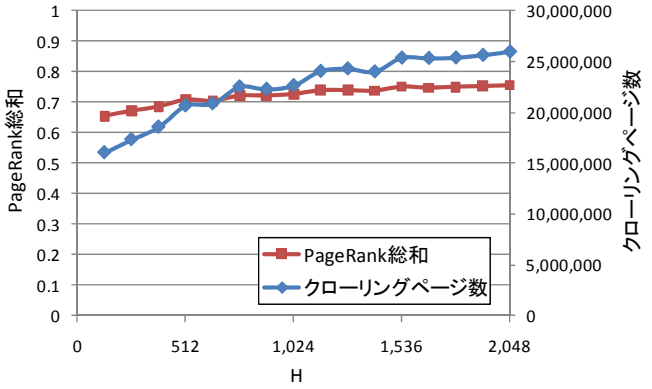

図 4. パラメータ H を変化させたときの
クローリングシミュレーション結果

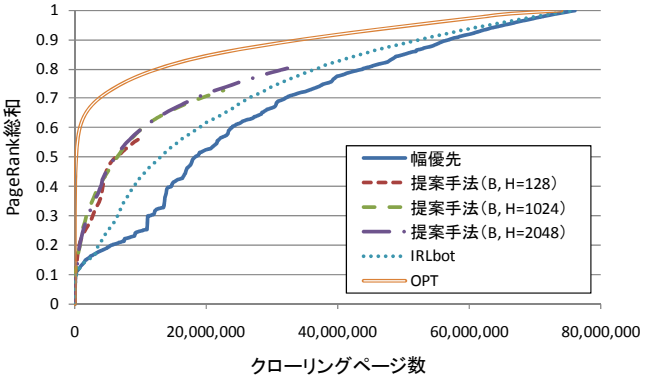

図 5. クローリングの進捗による PageRank 総和の増加

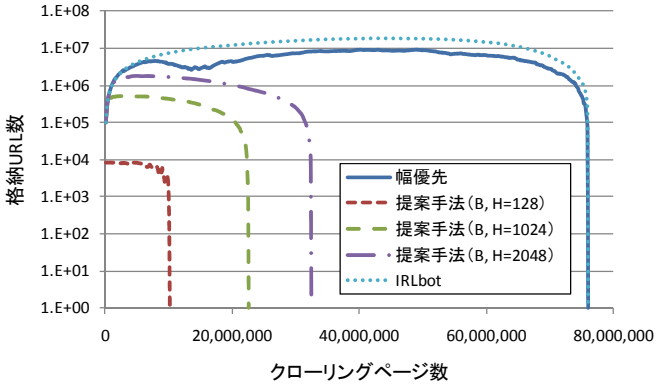

図 6. クローリングの進捗と格納 URL 数の変化

を選択し，そのシードから訪問できるページおよびリンク集合を実験に用いるグラフとした．抽出したグラフは 76,097,063 個のページからなっている．このグラフにおいて PageRank を計算した．図 2 に，グラフデータから計算した PageRank の総和と総和を達成するために最低限必要なページ数を掲載した．最低必要ページ数は，PageRank の大きい順にページを選択することで算出できる．図 2 から，全体の 0.01%のページに PageRank の 20%が集中していることがわかる．

4.3. 実験結果・考察

以下，行った 3 つの実験についての結果を掲載し，考察を行う．

実験1. パラメータ B の変化によるクローリングページ数と PageRank 総和の変化の評価

図 3 は，パラメータ B を変化させたときのクローリングページ数と PageRank 総和の変化を示すグラフである．なお，H は 1024 で固定している．図 3 から，提案手法は B を大きくするとクローリングページ数と PageRank 総和が増加することが確認できる．しかし，クローリングページ数の変化と比較して，PageRank 総和はクローリングページ数が倍になっても，およそ 0.1 程度しか増加していない．このことから提案アルゴリズムが PageRank の大きなページを優先的に収集しているといえる．

実験2. パラメータ H の変化によるクローリングページ数と PageRank 総和の変化の評価

図 4 は，パラメータ H を変化させたときのクローリングページ数と PageRank 総和の変化を示すグラフである．なお，B は 1024 で固定している．実験 1 と同様に，パラメータ H も全体としては値を大きくすることでクローリングページ数と PageRank 総和が増加することが確認できた．なお，図 4 において H を増加したときにクローリングページ数が減少した部分があるが，これは H を増加させたことで，剰余値の衝突が変化し，ページを多数持つホスト同士が同一の剰余値を持ってしまい，H を増加させたにも関わらず URL が多く破棄された結果と考えられる．

実験3. クローリングの進捗にともなう PageRank 総和の増加の評価

次に，クローリングの進捗にともなって，PageRank の総和がどのように増加するかを既存手法と比較した．もっとも少ないページ数で最多の PageRank を達成する，理論上の最適クローリング手順を OPT とする．ここで，OPT は事前に PageRank を計算し，リンク構造を無視して PageRank

が大きい順にページを収集した場合である．リンク構造を無視，すなわち全ての URL が既知の必要があるため，OPT のようなクローリングは不可能である．図 5 は，提案手法と OPT，幅優先型のクローリング，IRLbot で用いられているアルゴリズムの 4 種類における，クローリングページ数と PageRank 総和の関係である．提案手法では B と H を 3 種類に変化させている．図 6 はスケジューラに格納された URL 数の変化のグラフであり，空間計算量を示している．

図 5 からわかるように，提案手法は OPT には及ばないものの，幅優先型のクローリングと IRLbot によるクローリングと比較して，PageRank が高い URL を優先的に収集できていることがわかる．これは，PageRank が高い URL は破棄されたとしても再出現するためクローリングでき，逆に，PageRank が低い URL は破棄されるとクローリングできないためと考えられる．すなわち，PageRank が高いページが優先的にクローリングされていることになる．また，図 6 からわかるように，空間計算量に相当する格納 URL 数については，PageRank の 80%を収集する $B, H = 2048$ の時には IRLbot の方式で網羅的にクローリングした場合と比較して 1/10 程度で済んだ．

5. まとめ

本論文では，同一のウェブサーバへのアクセス間隔の最小値を保証する，時間計算量が $O(1)$ で，空間計算量の上限がユーザが指定したパラメータのみに依存するスケジューリングアルゴリズムを提案した．提案アルゴリズムはユーザが指定するパラメータで，ウェブサーバに対するアクセスの最短間隔を任意に設定でき，空間計算量とクローリング網羅率のトレードオフを調整できる．提案手法は URL を破棄するため，クローリングできないページが発生する可能性があるが，被リンク数が多い，すなわち PageRank が高い URL は破棄したあとに再度出現することが期待できる．すなわち提案手法には，URL を破棄しても PageRank の高いウェブページを取りこぼさない特色がある．クローリングシミュレーションの結果，URL を破棄することで空間計算量を少なく抑えられるだけでなく，PageRank が大きいページを優先的に収集できることがわかった．今後の課題として，実ウェブ空間での実験があげられる．

謝辞

本研究の一部は，JST 情報基盤戦略活用プログラム「多メディア Web 解析基盤の構築及び社会分析ソフトウェアの開発」，科学研究費補助金「挑戦的萌芽研究(21650010)」，科学研究費補助金（18049068）の助成による．

参 考 文 献

[1] J. Hays and A. A. Efros, “Scene Completion Using Millions of Photographs,” In Proc. of SIGGRAPH 2007, USA, Aug. 2007.

[2] Y. Morimoto, Y. Taguchi, and T. Naemura, “Automatic colorization of grayscale images using multiple images on the web,” In Proc. of SIGGRAPH 2009, USA, Aug. 2009.

[3] 黒木さやか，山名早人，立石健二，細見格，“アンカーテキストとリンク構造を用いた同義語抽出手法”, DEIM 2010, 兵庫, Feb. 2010.

[4] Web Archive: http://www.archive.org/ (2011.1.7 参照).

[5] データセクション株式会社: http://www.datasection.co.jp/ (2011.1.7 参照).

[6] nutch: http://nutch.apache.org/ (2011.1.7 参照).

[7] YaCy - The Peer to Peer Search Engine: Home: http://yacy.net/ (2011.1.7 参照).

[8] 80legs: http://www.80legs.com/ (2011.1.7 参照).

[9] A. Heydon and M. Najork, “Mercator: A scalable, extensible Web crawler,” In Proc. of the 8th WWW, pp. 219-229, Canada, May 1999.

[10] M. Najork and A. Heydon, “High-Performance Web Crawling,” SRC Research Report, COMPAQ System Research Center, Sep. 2001.

[11] J. Edwards, K. McCurley and J. Tomlin, “An Adaptive Model for Optimizing Performance of an Incremental Web Crawler,” In Proc. of the 10th WWW, China, pp.106-113, May 2001.

[12] V. Shkapenyuk and T. Suel, “Design and Implem-entation of a High-Performance Distributed Web Crawler,” In Proc. of the 18th ICDE, USA, Feb. 2002.

[13] Official Google Blog: http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html (2011.1.7 参照).

[14] H. Lee, D. Leonard, X. Wang and D. Loguinov, “IRLbot: Scaling to 6 billion pages and beyond,” ACM Trans. on the Web, vol.3, Jun. 2009.

[15] Sergey Brin and Lawrence Page, “The Anatomy of a Large-Scale Hypertextual Web Search Engine,” In Proc. of the 7th WWW, Australia, pp. 107-117, Apr. 1998.

[16] D. Eichmann, “The RBSE Spider – Balancing Effective Search Against Web Load,” In Proc. of the 1st WWW, pp. 113-120, Switzerland, May 1994.

[17] O. A. McBryan, “GENVL and WWWW: Tools for Taming the Web,” In Proc. of the 1st WWW, pp.79-90, Switzerland, May 1994.

[18] B. Pinlerton, “Finding What People Want: Experi-ences with the WebCrawler,” In Proc. of the 2nd WWW, USA, Oct. 1994.

[19] J. Cho and H. Garcia-Molina, “Effective Page Refresh Policies For Web Crawlers,” ACM Trans. on Database Systems, Vol. 28, No. 4, pp. 390-426, Dec. 2003.

[20] 田村孝之，喜連川優, “大規模 Web アーカイブ更新のための階層的スケジューリング手法,” 情報処理学会論文誌 データベース, Vol. 2, No. 3, pp. 67-75, Sep. 2009.

[21] A. Arasu, J. Cho, H. Gracia-Molina, A. Paepcke and S. Raghvan, “Searching the Web,” ACM Trans. on Internet Technology, Vol. 1, No. 1, pp. 2-43, Aug. 2001.

[22] A. Singh, M. Srivatsa, L. Liu and T. Miller, “Apoidea: A Decentralized Peer-to-Peer Architecture for Crawling the World Wide Web,” In Proc. of SIGIR Workshop on Distributed IR, Canada, pp. 126-142, Aug. 2003.

[23] Laboratory for Web Algorithm: http://law.dsi.unimi.it/. (2011.2.5 参照).