ブルームフィルタを用いた重複ウェブページの削除についての実現

The implementation of deleting duplicates web pages based on Bloom filter

12N8100023F　情報工学専攻　李　超　LI Chao

1　はじめに

現在、検索エンジンは主にインターネットに存在する情報を得る手段として使われている。単にキーワードを入力して、検索ボタンを押せば、ほしい情報が得られる。しかし、検索エンジンは便利な一方で、重複ウェブページが存在し、大量の重複情報が検索されるため、検索エンジンの性能に悪影響を与えている。それだけではなく、大量なウェブ保存空間を占めて、ユーザーの利便性と検索エンジンの適合性 (*Precision*) にも影響を与える。

重複ウェブページは重複コンテンツとも呼ばれる。重複コンテンツとは、ドメイン内または複数ドメインにまたがって存在する、他のコンテンツと完全に同じであるか非常によく似たコンテンツのまとまりを指す。

本研究では、あるナレッジミュニティウェブサイトに対して、クローラと重複検証システムを実装する。クローラは自動的にこのウェブサイト上のすべてのページをアクセスし、保存する。そして、保存したウェブページに対して、重複検証システムで重複性を検証する。そして、検証した結果より、再現率と適合率に基づいて検証システムを評価する。更に、類似度の閾値の設定について検討する。

1) クローラ クローラは、幅優先探索によりウェブサイトのすべてのウェブページを訪問し、ページの内容を抽出し、データベースに保存することである。探索する時、保存したウェブページを一回だけアクセスするために、ブルームフィルタを用いて重複アドレスを検証する。

2) 重複検証システム *CDC*(*Content−Defined Chunking*) により保存したウェブページを分割し、分割した文字列をブルームフィルタで重複性を検証する。

2　ブルームフィルタ

ブルームフィルタ（*Bloom Filter*）は、1970 年に *Burton H.Bloom* が考案した空間効率の良い確率的データ構造であり、要素が集合のメンバーであるかどうかのテストに使われる (図 1)。ブルームフィルタの長所は空間効率と計算時間が良いことであり、偽陰性がないことである。逆に、保存した要素の数が増加すると共に、偽陽性率が高くなる。そして、保存した要素の削除については困難である。

最初に、全て 0 に設定された M ビットのビット配列を用意する。同時に k 個のハッシュ関数が定義されており、それぞれがキー値を M 個の配列位置のいずれかにマッピングする。ブルームフィルタに要素を追加するには、それを k 個のハッシュ関数に入力して k 個の配列インデックスを得る。そして、それらの位置のビットを全て 1 にする [**?**]。

まだ、要素がその集合に含まれるかどうかを調べるには、それを k 個のハッシュ関数に入力して k 個の配列インデックスを得る。それらの位置のビット群のひとつでも 0 だった場合、その要素はその集合には含まれていない。逆に示された全ビットが 1 なら、その要素はその集合に含まれているか、それらのビットは他の要素を追加したときに偶然全部 1 になった（偽陽性）と考えられる。

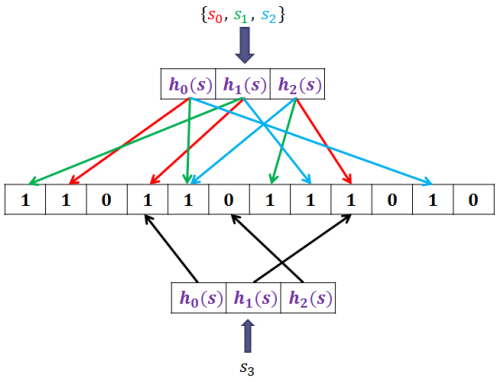

図 1: ブルームフィルタより、要素の保存と検証

本研究でブルームフィルタの実装については、ブルームフィルタのバリアントの一つ *PBF*(*Partial Bloom Filter*) を選んだ (図**??**)。標準的なブルームフィルタと違って、一つのサイズは M のビット配列ではなく、k 個のハッシュ関数に対して、k 個サイズはスライスと呼ばれた m のビット配列を用いた。PBF のメモリ空間のサイズは $k * m = M$ である。

PBFのサイズ $M = 24\ bit$
ハッシュ関数の個数 $k = 3$
スライスのサイズ $m = \frac{M}{k} = 8\ bit$

$h_1(S_1) = 5$
$h_2(S_1) = 2$
$h_3(S_1) = 1$

$h_1(S_2) = 3$
$h_2(S_2) = 6$
$h_3(S_2) = 8$

$h_1(S_3) = 4$
$h_2(S_3) = 2$
$h_3(S_3) = 8$

$h_1(s)$	$h_2(s)$	$h_3(S)$
0	0	1
0	1	0
1	0	0
1	0	0
1	0	0
0	1	0
0	0	0
0	0	1

図 2: Partial Bloom Filter

ブルームフィルタは以下の 2 つの数式で、偽陽率をコントロールできる [**?**]。n はブルームフィルタに保存している要素の個数であり、M はブルームフィルタのメモリ空間のサイズである。P はブルームフィルタの偽陽率である。

$$n \approx -M \frac{(\ln 2)^2}{|\ln P|} \tag{1}$$

$$K = \log_2 \frac{1}{P} \tag{2}$$

表 1: 32 Kilobytes のメモリ空間より、複数 PBF の偽陽率の変化 [?]

P	0.1%	0.01%	0.001%	0.0001%
k	10	14	17	20
m	26214	18724	15420	13107
n	18232	13674	10939	9116

PBF を実装する時、最初に偽陽率は応用しているシステムより設定できる。例えば、表 (??) から、最大の偽陽率として 0.1% がほしければ、ハッシュ関数は 10 個を用意すれば良い。なぜなら、$k = \log_2 \frac{1}{0.001} \approx 9.96$ であるからである。もし、フィルタのサイズが $32KB$ の時には、一つのスライスのサイズは 26214 ビットであり、最大 18232 個要素が保存できる。

3 Web クローラとウェブページ内容の抽出

3.1 Web クローラ

本システムの Web クローラは 4 つのシステムモジュールで構成されている (図 3, 4)。

1) ページダウンロードモジュール

ページダウンロードモジュールは、URL に対して存在するページをダウンロードするモジュールである。クローラの性能と直接関連しているので、クローラを設計する時、このモジュールのスレッド、排他制御などの処理を慎重に考えなければならない。

2) ページ分析モジュール

ページ分析モジュールは、ウェブページを分析し、ページの中に存在しているリンクを抽出するモジュールである。ページ分析モジュールはクローラに対して最も重要な部分であり、検索エンジンの再現率に影響する。

3) リンクフィルタモジュール

リンクフィルタモジュールは、ページ分析モジュールでウェブページを分析する時、抽出したリンクについて重複性を検証するモジュールである。

4) URL 配列モジュール

リンクの保存と管理のモジュールである。配列のデータ構造を持ち、ページダウンロードモジュールにリンク提供する。まだ、リンクを保存する前にブルームフィルタを用いて、リンクの重複性検証を行う。

3.2 ウェブページ内容の抽出

ページダウンロードモジュールはページの中にあるすべての内容を保存したが、HTML は自分自身では説明できないので、ダウンロードしたページの中身はかなり無用なものがある。例えば、広告、ウェブページサイトのナビゲーションなど無用である。このような文字列は検証システムに入ると、再現率に影響が大きいだろう。できれば、検証システムに入れる前に、ページの本文だけ残ってほしい。

ここで、Java の HTML ドキュメント解析 API「HTMLParser」[?] を用いて、ウェブページを解析し、中身を抽出する。「HTMLParser」とは、線形まだはネスト化のいずれかで HTML を解析するために使用する Java ライブラリである。解析する時、HTML のタグを抽出し、木構造を作る。ほしいタグをフィルタすることである。

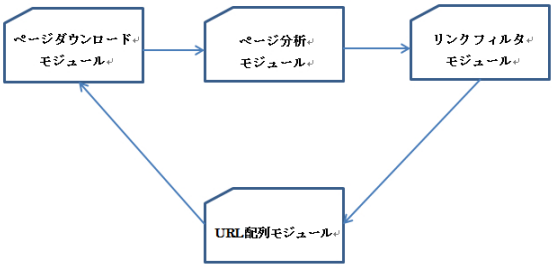

図 3: クローラのシステムモジュール

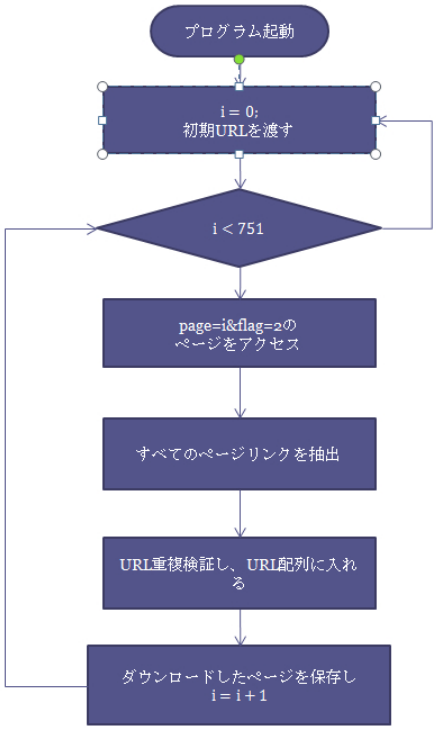

図 4: 実装したクローラの流れ

抽出したページ内容では、以下の様に、上のナビゲーションバーと右側の広告バーを抜いて、テキストだけを残した (図 5, 6)。

図 5: ウェブページ

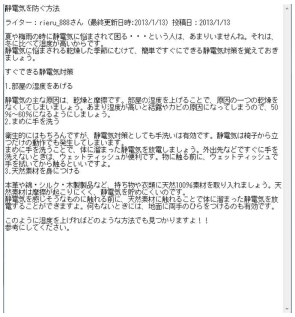

図 6: 抽出したページ内容

3.3 Web クローラと内容抽出の連動

ウェブページ探索を効率化するために、Web クローラと内容抽出モジュールはマルチスレッド化する。2 つのスレッドを同時に動かして、ウェブページを探索する。まだ、URL 配列が空きの時、スレッドは 1 秒または 2 秒待つ (図 7)。

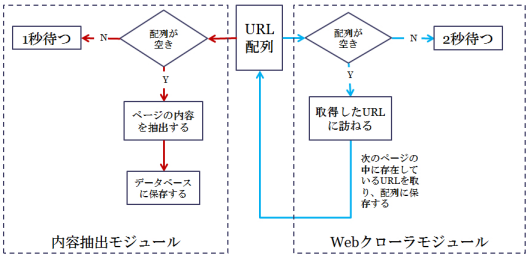

図 7: Web クローラと内容抽出の連動

4 重複ページ削除の実現

4.1 重複ページ削除の提案手法

ブルームフィルタを用いて、文章の類似度検証モジュールは三つのステップで行う

まず、$CDC(Content-definedChunking)$ により、文章を分割する。つまり、文章の特徴を抽出する。

そして、抽出した特徴は集合の要素として、ブルームフィルタを生成する。

最後に、閾値を設定し、他の文章との類似度を検証する。

4.2 Content-Defined Chunking(CDC) の概略

文章の内容より、Content-Defined Chunking(CDC) で分割するのは、固定分割のブレークポイントが設定し値で決めることと違って、文章の内容より計算し、ある条件を満たすブレークポイントを決める。

ここで、ハッシュ関数の 2 つの特徴を利用する。一つは、ハッシュコードが同じ時でも、内容は必ずしも同じではない。ハッシュコードが違うと、内容は必ず違う。もう一つは、分割した文字列のサイズが大きくならないようにするためには、ハッシュ関数での衝突はできれば、その頻度が多ければ多いほうど良い。このようなハッシュ関数を使って、生成したハッシュコードが設定した値と同じの時、ブレークポイントを決める。

4.3 *Rabin Fingerprint* アルゴリズム

Rabin Fingerprint アルゴリズムは M.O.Rabin が 1981 年提案した。CDC のブレークポイントを決めるハッシュ関数としてよく使われっている。

具体的には、以下の様に計算する。

最初に、文字列 A を定義する。

$$A = a_m a_{m-1} ... a_1 \tag{3}$$

$k-bit$ の *Rabin Fingerprint* は次の順に処理する。

1. 文字列 A を多項式 $A(t)$ で表現する。

$$A(t) = a_m t^{m-1} + a_{m-1} t^{m-2} + ... + a_2 t + a_1 \tag{4}$$

2. 既約多項式 $P(t)$ を選ぶ。

$$P(t) = p_k t^k + p_{k-1} t^{k-1} + ... + p_0 \tag{5}$$

3. *Rabin Fingerprint* を計算する。

$$f(A) = A(t) \bmod P(t) \tag{6}$$

例えば、文字列 A はビットに列で、$A = 101010$ であり、4-bit の *Rabin Fingerprint* を求めるために、既約多項式 $p = 11111$ を選んだ結果は以下のとおりである:

$$\begin{array}{r} 101010 \\ 11111 \quad \oplus \\ \hline 10100 \\ 11111 \quad \oplus \\ \hline 1011 \end{array} \quad \rightarrow \quad f(A)$$

Rabin Fingerprint は内容より生成したものである。一つの *Rabin Fingerprint* の $f(A)$ は多項式を既約多項式 $P(t)$ で割った時の余り多項式であり、すべての結果から同じ条件を満たすデータを多く得られるため、計算は一回しか行わない。

4.4 可変長分割

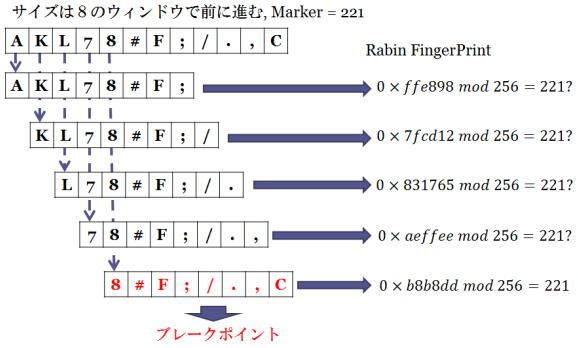

図 8: *Rabin Fingerprint* を用いて、CDC の実現

図**??**のような、固定しているウィンドウより、一回 1 文字ずつ前に進み、8 位の文字の *RabinFingerprint* を計算する。計算した *RabinFingerprint* と *marker* 値をマッチングして、分割する [**?**]。

4.5 ブルームフィルタの生成

分割した文字列をハッシュコードに生成し、ブルームフィルタに格納する。一つの文字列は文章の一つの要素として、要素は 1 個ずつ k 個ハッシュ関数でハッシュコードを生成し、各ハッシュ関数に対するスライスにハッシュコードとマッピングする位置に 1 を設定する。

具体的な実現については、0.1% の偽陽率について、ハッシュ関数の個数 k は $\log_2 \frac{1}{0.001} = 9.96$. 約 10 になる。そして、抽出したウェブページの本文が最大に $16KB$ と考え、予想している文字列は $256byte$ であり、つまり格納する要素数 n は約 64 個である。数式 (1) より、ブルームフィルタのサイズ M は $2119.205byte$ である。

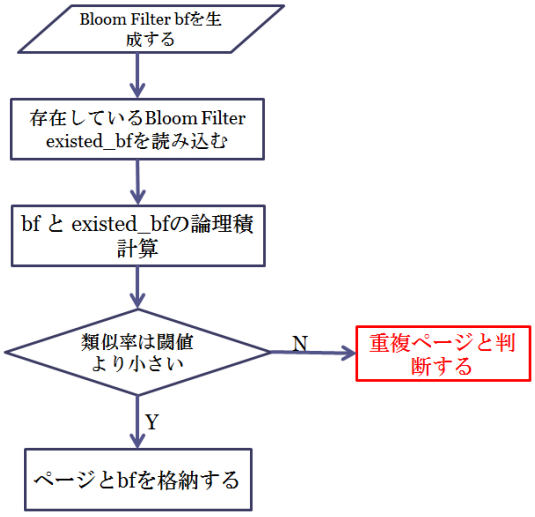

図 9: 重複ページ削除の流れ

4.6 重複ページ削除の実験結果

適合率 (*Precision*) とは、検索結果の中にどの程度正解が含まれるかを示す。本システムより、w_0 個重複ウェブページを検索した結果として、その中に、w 個が正解となると、本システムの適合率は $Precision = \frac{w_0}{w}$ である。

再現率 (Recall) とは、正解のうち、どの程度が検索にヒットするかを示す。本システムより、検索した結果の中に正しい結果にとって、重複ウェブページの全体に占める割合を示す。例えば、w_0 個重複ウェブページを正しくて検索し、実際に、存在している重複ウェブページは w_n であると。本システムの再現率は $Recall = \frac{w_0}{w_n}$ である。

表 2: 閾値により、適合率と再現率の変化

閾値	検索した件数	誤検数	適合率 (%)	再現率 (%)
1	18	0	100%	15%
0.9	42	2	95.24%	33.33%
0.85	77	6	92.2%	59.17%
0.8	139	24	82.73%	95.83%
0.75	173	56	67.63%	97.5%
0.7	247	131	46.96%	96.67%

一般に、再現率の高いシステムは適合率が低く、その逆に、適合率が高いシステムは再現率が低い傾向にある。評価指標が 2 つあると、どちらのシステムが優れているか比較が難しいので、再現率と適合率の調和平均を取った値を F 値（$F-measure$）という指標で性能を表す。

F 値 =（2 ×適合率×再現率）/（適合率＋再現率）

実験データより、F 値が最も高いのは閾値は 0.8 の時、$F = \frac{2*0.8273*0.9583}{0.8273+0.9583} = 0.89$ である (図 8,10, 表 2)。

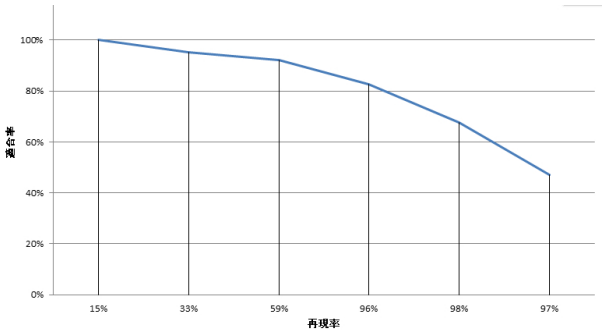

図 10: 適合率と再現率関係

5 結論

本研究では, ブルームフィルタを用いて、*CDC* によりウェブサイト上に存在している重複ウェブページの削除について提案した。提案システムでは, 自動的ウェブサイト上のページを探索し、探索したウェブページ中の URL を収集する。ブルームフィルタにより、収集した URL の重複性を検証し配列に保存する。ウェブページを探索する同時に、URL 配列から URL を取り、ページ内容を *HTMLParser* ツールにより、必要な内容だけを抽出しデータベースに格納する。まだ、データベースに保存しているウェブページの相互間の類似率を求め、削除することができる。

実験の結果より、F 値（$F-measure$）を標準として、類似度の閾値が 0.8 にセットすることを薦める。

本研究では、重複ウェブページの削除について提案したが、具体的応用に対してまだ改善することがある。他のアルゴリズムの性能、効率データに対するコンテストが足りないが、本研究の結果は今後に更に複雑な研究やシステムに参考されることができるだろう。

謝辞

本研究を通じ, 懇切丁寧な御指導, 御鞭撻, 及び多くの御支援を賜りました, 中央大学理工学部情報工学科浅野孝夫教授に深く感謝致します.

参考文献

[1] M. O. Rabin. “Fingerprinting By Random Polynomials.” Technical Report TR-15-81, Harvard University, 1981.

[2] Almeida, Paulo; Baquero, Carlos; Preguica, Nuno; Hutchison, David (2007), “Scalable Bloom Filters”, Information Processing Letters 101 (6): 255261.

[3] Bloom, Burton H. (1970), “Space/Time Trade-offs in Hash Coding with Allowable Errors”, Communications of the ACM 13 (7): 422426.

[4] 非営利団体 OSDN（Open Source Development Network）-HTMLParser, http://htmlparser.sourceforge.net/ (最終アクセス 2014 年 2 月 13 日)