Vol.2017-IFAT-125 No.9
Vol.2017-DC-104 No.9
2017/3/10

ウェブブラウザにおける既読コンテンツの検出・表示手法の検討

早乙女 高大[1,a)]　相田 仁[1,b)]

概要：検索エンジンは WWW において情報を検索するのに不可欠な技術である．しかし，検索エンジンを実装する際の課題として重複した内容のページや検索エンジンスパム等が指摘されている．重複した内容のページが検索結果に現れると，既に読んだものとは異なる内容を期待しているにもかかわらず同一内容のページを何度も見ることとなり，利便性が損なわれる．本研究では，ユーザーがあるページを読んだ後にその内容を記録し，別のページにアクセスした際に記録された内容と類似していた場合はその部分に書式変更等の処理を施して表示するという手法，その手法のウェブブラウザ用拡張機能としての実装，およびその実装を行うにあたっての類似度判定の方法について検討している．

キーワード：類似度，ウェブブラウザ，拡張機能

A Study of Methods for Detecting and Showing Already-read Contents in Web Browsers

Sotome Takahiro[1,a)]　Aida Hitoshi[1,b)]

Abstract: The search engines are indispensable technology for searching for information in the WWW. However, they have problems such as pages with duplicate contents and spamdexing. Duplicate contents on a search engine result page (SERP) degrade the usability, as the users expecting something different from what they have already read end up seeing pages with the same contents. In this research, we propose the methods which record what the users have read, detect the already-read contents, and annotate them by modifying text styles. We also implement the method as an extension for web browsers, and discuss the methods for calculating the similarity for the implementation.

Keywords: Similarity, web browsers, extensions (add-ons)

1. はじめに

インターネット等の情報技術は 21 世紀に入って急速に普及した．その中でも検索エンジンはインターネットにおいて情報を検索する際には不可欠な技術となっている．しかしながら，検索エンジンを実装しようとする際の課題として，重複した内容のページや検索エンジンスパム等が指摘されている [1]．多くの場合，検索エンジンによる検索結果を利用する際には上位に表示されたウェブサイトから順次閲覧することになり，その最中ユーザーは既に読んだものとは異なる内容を期待している．そのような状況において重複した内容のページが検索結果に現れると，ユーザーは同一内容のページを何度も見ることとなり，利便性が損なわれる．

一例として，あるエラーメッセージが表示され，解決策を求めているという状況を考える．図 1 は Google において

firefox で接続がリセットされる

というキーワードで検索を行った結果の上位 10 件（左半分が 1 位から 5 位，右半分が 6 位から 10 位）を示してい

[1] 東京大学大学院工学系研究科
School of Engineering, The University of Tokyo
[a)] sotome@aida.t.u-tokyo.ac.jp
[b)] aida@ee.t.u-tokyo.ac.jp

るが，そのうち 3 番目，および 10 番目の項目は，6 番目の項目 “OKWave” が他に転載されたものである．

2. 背景と関連研究

2.1 重複した内容

重複するコンテンツの例として，次のものが挙げられる．

引用・転載 ユーザーに対するサービスの一環として，他のウェブサイトの内容を転載する場合がある．

複数サイト間の情報共有 ユーザーに対する情報の露出を増やすために，複数のサイトが同一の内容を掲載する場合がある（コンテンツシンジケーション）．

まとめ（キュレーション） 複数の情報源を 1 ページにまとめる，あるいは一つの情報源の内容を取捨選択するなどして読みやすくするサイトやサービスは多数存在する．

また，重複はしていないが，似通ったコンテンツが多くなる例としては，同一の主題や同一の製品について説明している複数のページがある．例えば，ある事件が発生した際にその事件について報じる複数のニュースサイト，ある製品に対する複数のレビュー記事，ある医薬品についてその作用・用法を説明する複数のサイトなどの状況が考えられる．

2.2 ブラウザ拡張機能

特定のウェブサイト A が別のウェブサイト B の内容を転載していることが既知であるならば，転載先のウェブサイトを検索時に除外して検索したり，検索結果において非表示にしたりする方法が有効であり，実際にそのような機能を実現するブラウザ拡張機能が提供されている．

Personal Blocklist[2] は，Google の検索結果からドメイン名を指定して特定のサイトを除外する機能を持つ Chrome 用拡張機能である．Google の検索結果から特定のサイトを除外する機能は，かつて Google 側のサービスとして “Blocked Sites” という名前で提供されており，ブロックされた回数の多いサイトの検索結果における順位を下げるなどサービスの質の改善のために利活用することが意図されていたが，このサービスは現在廃止されており，その代わりとして [2] が提供されている．

2.3 ユーザースクリプト

Greasemonkey[3]，Tampermonkey[4] は現在表示中のウェブページに任意の JavaScript コード（ユーザースクリプト）を挿入することでウェブページの外観や挙動を変更する機能を持つ拡張機能である．これらの拡張機能を用い，検索結果において特定のウェブサイトを除外する機能を持つユーザースクリプトは多数開発・公開されている．その例として Google Hit Hider by Domain[5] や Google Domain Blocker[6] 等が挙げられる．

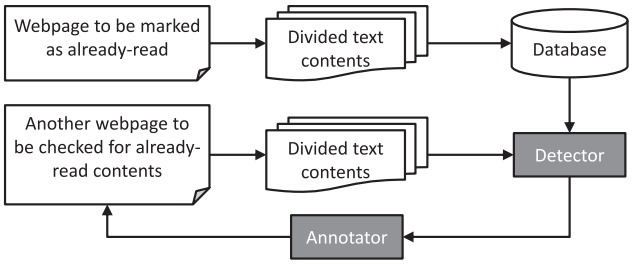

図 2 提案手法の概略
Fig. 2 A summary of the proposed method.

しかし，他サイトの転載であることが実際にアクセスするまで分からないウェブサイトの場合は，これらの拡張機能やユーザースクリプトでは対応できない．

3. 提案手法

これらの重複コンテンツの問題に対し，本研究では類似度算出による既読コンテンツの検出・表示手法を提案する．本手法の概略を図 2 に示す．本提案手法では，最初に，ユーザーが現在読んでいるページを読み終わった際に，そのページの内容を適当な単位に分割し，ページの URL や閲覧日時などの付加情報とともに記録する．その後にユーザーが別のページを読もうとした際に，同様に現在のページの内容を分割した後，現在の内容と記録されていた内容（既読コンテンツ）との間で類似度を算出する．最終的に，現在表示中のページの内容のうち，記録された内容との類似度がある閾値を超えたものに対し，一律に同一の書式を適用するか，または類似度に応じて適用する書式を変化させる．

3.1 既読コンテンツの表示手法

既読コンテンツとの類似度が高い部分に対して行う書式設定方法には，図 3 のようなものが考えられる．

背景色変更（網かけ） 背景色を文字色に近づけることにより，既読コンテンツを目立たなくさせる．方法 1 では類似度がある閾値以上の部分に対して同じ背景色を適用する．方法 2 では類似度に応じた濃度で網かけを行う．

文字色変更 背景色変更とは逆に，文字色を背景色に近づけることにより，既読コンテンツを目立たなくさせる．方法 3 では類似度がある閾値以上の部分に対して同じ文字色を適用する．方法 4 では類似度に応じて文字色を変更する．

文字サイズ変更 既読コンテンツを縮小表示することにより目立たなくさせる．方法 5 では類似度がある閾値以上の部分に対して，文字を一律に縮小する．方法 6 では類似度に応じて文字サイズを変更する．

図 1 重複コンテンツの例
Fig. 1 An example for duplicate contents.

1. ほぼ一致する文。かなり類似した文。
   あまり類似していない文。ほとんど類似していない文。
2. ほぼ一致する文。かなり類似した文。
   あまり類似していない文。ほとんど類似していない文。
3. ほぼ一致する文。かなり類似した文。
   あまり類似していない文。ほとんど類似していない文。
4. ほぼ一致する文。かなり類似した文。
   あまり類似していない文。ほとんど類似していない文。
5. ほぼ一致する文。かなり類似した文。
   あまり類似していない文。ほとんど類似していない文。
6. ほぼ一致する文。かなり類似した文。
   あまり類似していない文。ほとんど類似していない文。

図 3 既読コンテンツの表示手法
Fig. 3 Methods for showing already-read contents.

3.2 基礎技術

ここでは，提案手法を実現するために必要な類似度判定の尺度，および類似度判定を効率よく行うライブラリについて説明する．

3.2.1 N-gram

文章や単語の意味を比較しようとする際には「文脈ベクトル」が用いられる．文脈ベクトルとは，文章を特定の規則に基づいて分割したとき，単語毎の出現率により与えられるベクトルである．

主要な文脈ベクトルの生成方法に N-gram がある．N-gram とは，ある文字列に含まれる，最大 N 文字までの部分文字列のである．$1 \leq N \leq 3$ の範囲が一般的に用いられ，それぞれ unigram, bigram, trigram と呼ばれる．例えば，文字列 ‘既読コンテンツ’ に対する trigram には次の 9 要素 {‘既’, ‘既読’, ‘既読コ’, ‘読コン’, ‘コンテ’, ‘ンテン’, ‘テンツ’, ‘ンツ’, ‘ツ’} があり，その出現回数は全て 1 である．

また，N-gram に対する類似度関数の一つにコサイン類似度がある．数値ベクトルに対するコサイン類似度 (Cosine Similarity) は，文字列 x，y の文脈ベクトルをそれぞれ $\boldsymbol{x}$, $\boldsymbol{y}$ とするとき，次式により与えられる．[7]

$$\cos(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{|\boldsymbol{x}||\boldsymbol{y}|}$$

互いに類似している文字列であるほどコサイン類似度は高くなり，完全に一致している場合には 1 となる．

3.2.2 レーベンシュタイン距離

3.2.1 節にて述べた「文脈ベクトル」とそれに対する「類似度関数」によらない類似度判定手法として「レーベンシュタイン距離 (Levenshtein Distance)」がある．レーベンシュタイン距離は編集距離の一種であり，文字の挿入，置換，削除等の操作によりある文字列 x を 別の文字列 y に変更しようとした際に必要な操作の最小回数として与えられる．[8]

同じ類似度の文章であっても，一度に比較する文章量によりレーベンシュタイン距離は変化するため，本研究においては，算出したレーベンシュタイン距離を対象となる 2 つの文章のうち長い方の文字数で割るという正規化を行っている．

3.2.3 SimString

SimString[9] は，類似文字列検索（文字列集合の中から，ある文字列と一定以上の類似度を持つものを探し出す操作）を高速に行うためのライブラリである．

SimString が対応している類似度関数は，コサイン係数，ジャッカード係数，ダイス係数，およびオーバーラップ係

数である．また，文字列の類似度を計算するための特徴量として，3.2.1 節にて述べた文字単位の N-gram に対応している．

SimString は C++用ライブラリであるが，SWIG を経由することで Python/Ruby から直接呼び出すことが可能である．また，それ以外の言語であってもコマンドラインインタフェースを用いて SimString を利用可能である．

ただし，SimString では類似文字列検索においては類似度が閾値以上であるか否かのみに着目することにより高速化を図っているため，複数の文字列について与えられた類似度閾値を満足するか否か求めることは可能であるが，類似度の最大値を直接求めることは仕様上不可能である．

4. 実装

これらの基礎技術による既読コンテンツ検出，およびおよび類似度に基づいた書式設定を行うシステムを下記の二通りの方式として実装した．

4.1 スタンドアローンモデル

提案手法における全ての処理をブラウザ拡張機能内で完結させる方式である．この方式は Mozilla Add-on SDK による Mozilla Firefox 用の拡張機能 (Extension) として実装されており，主に「コンテンツ記録部」「検出・表示部」の 2 つに分けられる．

4.1.1 コンテンツ記録部

コンテンツ記録部は，現在のページを既に読んだものとして，Mozilla Add-on SDK において提供されている高レベル API である simple-storage を用い，当該ページの全ブロックレベル要素が持つテキストノードを要素単位で記録する．

例えば，次の HTML に対しては，「既読コンテンツ」「未読コンテンツ」の 2 文字列が記録される．

```
<p><em>既読</em>コンテンツ</p>
<p>
<a href="foo.html">未読</a>コンテンツ
<img src="bar.png"></img>
</p>
```

また，よりきめ細かい検出を行うために，各要素の持つテキストノードについて，それらを句読点や空白で分割したものを同時に記録する．例えば，

　この文章は，まだ読まれていない．

という文字列は，元の文字列と共に「この文章は」「まだ読まれていない」という 2 文字列に分割されて記録される．

要素単位，句読点単位のいずれの場合も，記録の前に「全ての全角英数を半角に，連続する空白文字や改行を単一の半角スペースに置換する」という正規化を行う．

4.1.2 検出・表示部

検出・表示部は，現在のページの文字列データをコンテンツ記録の際と同様に取得し，記録されたコンテンツとの間の類似度を算出する．

類似度を算出した後，現在のコンテンツに対する「既読として記録したコンテンツ」の類似度に関して，図 3 のように，その最大値に応じて書式を設定するか，または類似度がある閾値を超えた場合に一律に書式を設定するなどの方法でユーザーに通知する．

要素単位の書式設定は，該当する要素に CSS によるスタイルを設定することにより行う．句読点単位の書式設定の場合は，そのままでは該当する部分の書式を直接設定することが不可能であるため，その文字列が含まれる要素から該当する文字列を検索し，`<span></span>`タグで囲むという処理を行った後，追加された `span` 要素に対して前述の書式設定を行う．

4.2 クライアントサーバモデル

一方，スタンドアローンモデルの一連の処理のうち，図 2 において “Database” および “Detector” で示される記録・検出処理の部分をブラウザの外部で動作するサーバプログラム上で行うことも可能である．この場合は，ブラウザ拡張機能はユーザインタフェースの提供，現在のページ内容の取得，および書式設定のみを行い，これらの部分における処理の流れはスタンドアローンモデルと同様である．

この方法では前者と比較して，ブラウザ拡張機能の仕様に束縛されない実装が可能となり，より自由度が高まる．

この実装では，HTTP サーバプログラムとして nginx を，クライアントからの記録・検出要求を CGI(Common Gateway Interface) を介して処理するためのプログラミング言語として Perl を，記録されたサイトやその閲覧日時を保持するためのデータベース管理システムとして MySQL を，類似度算出のためのプログラムとして SimString をそれぞれ用いた．

4.2.1 記録時の処理の流れ

最初に記録時の処理の流れについて説明する．現在のページを既に読んだものとして記録する際の，クライアントおよびサーバ側の処理の流れを図 4 に示す．

最初に，拡張機能のみによる実装と同様の処理により，現在表示されているページの文字列データ，およびその文字列が含まれる HTML 要素の一意な CSS セレクタを取得し，JavaScript オブジェクトの形で保持する．

続いて，このオブジェクトにユーザ認証のための資格情報を付加した上で，JavaScript における標準 API である `JSON.stringify()` を用いて JSON 形式の文字列に変換し，XMLHttpRequest を用いて記録・検出用サーバ上の `record.cgi` へと送信する (1).

要求を受け取ったサーバプログラムは `record.cgi` にその入力を渡す．`record.cgi` は，最初に各要素の文字列データについて形態素解析エンジン MeCab[10] を用いた形態素

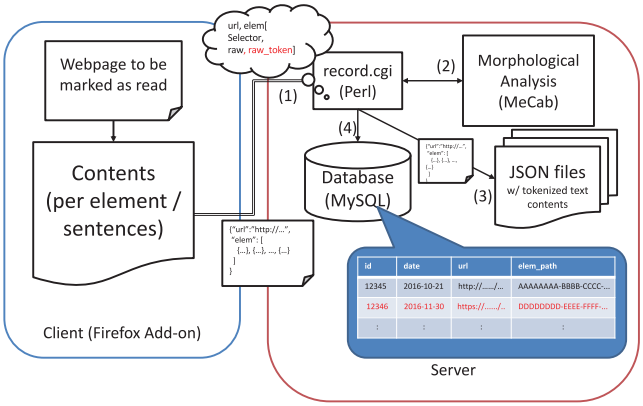

図 4 クライアントサーバモデルにおける記録時の処理
Fig. 4 A workflow of recording in client-server model.

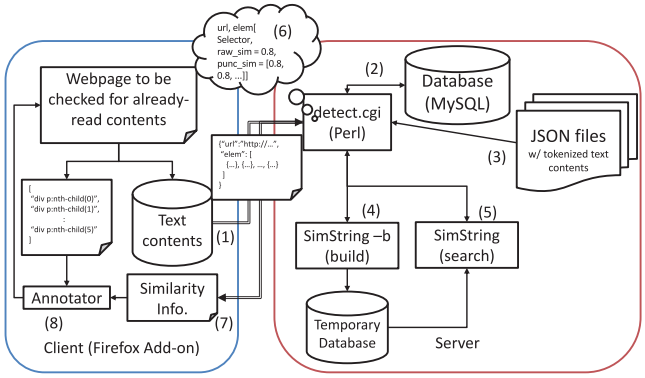

図 5 クライアントサーバモデルにおける検出時の処理
Fig. 5 A workflow of detection in client-server model.

解析を行い，その結果を現在のページ情報に付加する (2)．これは将来的に形態素解析に基づく類似度判定を行うためのものである．

続いて，当該ページ情報を，UUID（汎用一意識別子）により重複しないファイル名とした JSON 形式ファイルとして保存し (3)，そのファイル名と現在時刻，現在のページの URL を既読コンテンツテーブルに追記する (4)．データベースの肥大化を避けるため，追記を行う際には事前に既読コンテンツテーブルから `url` フィールドが現在の URL と完全に一致するレコードを全て削除する．

4.2.2 検出時の処理の流れ

続いて，検出時の処理の流れについて説明する．現在のページから既読コンテンツを検出する際の，クライアントおよびサーバ側の処理の流れを図 4 に示す．

現在のページの文字列データおよび一意な CSS セレクタの取得については記録時と同様である．

取得した文字列データを含む JavaScript オブジェクトに類似度関数や検出単位等を含むパラメータ，および資格情報を付加した上で JSON 形式の文字列に変換し，XMLHttpRequest を用いて記録・検出用サーバ上の `detect.cgi` へと送信する (1)．

要求を受け取った `detect.cgi` は，最初に受け取った JSON ファイルから上記パラメータを取得する．次に，既読コンテンツテーブルから既読コンテンツの実体が格納されている JSON 形式ファイルのリストを取得する (2)．ここでは，パラメータに基づいて，例えば「直近 1 ヶ月」といった期間指定検索を行うことも可能である．

続いて，データベースから得られた既読コンテンツファイルに順次アクセスし，要素単位，または句読点単位の文字列データを読み込む (3)．

続いて，読み込んだデータについて，SimString による類似文字列検索用一時データベースを作成する (4)．この一時データベースは全ての検出処理が完了するまで保持される．

引き続き，SimString により，各文字列に対して一定以上の類似度を持つ文字列が存在するかどうか，順次検索する (5)．

SimString による検索の終了後，その結果に基づき現在のページ情報に類似度の情報を付加し (6)，JSON 形式へと変換した上でクライアントへと返す (7)．

類似度検索の結果をサーバから受けとった Firefox アドオンは，その結果に含まれる要素単位，あるいは句読点単位の類似度に基づき，書式設定を行う (8)．

4.2.3 連続書式変化の場合の SimString の実行方法

類似度に応じた書式設定を行う場合，スタンドアローンモデルでは類似度の最大値を直接求めることが可能である．しかし，クライアントサーバモデルの場合は 3.2.3 節でも述べた SimString の仕様により，類似度の最大値を直接求めることは不可能である．

したがって，類似度に応じた書式設定を行う場合は，類似度を段階的に変化させて SimString を繰り返し実行することが必要となる．この場合の SimString の実行方法として，次のものが考えられる．

線形探索 (Linear search) 類似度の閾値を 0 から 1 の間で線形的に変化

二分探索 (Bisection search) ある閾値における結果に応じて次回の検索時の閾値・検索範囲を狭める

傾斜配分探索 (Inclined search) 類似度の高い既読コンテンツを探索する処理により多くの計算機資源を付与 (基本的な処理の流れは「線形探索」と同様)

5. 性能評価

5.1 スタンドアローンモデルにおける検出性能評価

あるウェブサイトが別のウェブサイトの内容を転載している最初の例として，Wikipedia 日本語版と，ポータルサイトのサービスの一環としてその内容を転載している「goo Wikipedia 記事検索」が挙げられる．これらの 2 サイトについて，転載元 (Wikipedia) のある記事を既読として記録し，転載先 (goo Wikipedia) において既読コンテンツの検

	Stand-alone		Client-server	
Style	BG	None	BG	None
Threshold 80%	172.243	174.934	1.749	1.273
Grad. (Direct)	242.574	179.130	N/A	N/A
Grad. (Linear)	N/A	N/A	53.606	3.965
Grad. (Bisection)	N/A	N/A	92.700	42.099
Grad. (Incl.)	N/A	N/A	51.097	1.515

表 1 実装による既読コンテンツ検出・表示の所要時間の変化
Table 1 Elapsed time for detecting and annotating already-read contents by implementation models.

出を行い，レーベンシュタイン距離が 10 以下，またはコサイン類似度が 0.8 以上の部分を表示した．その結果を表 6 に示す．

表 6 のうち左側が転載元，右側が転載先である．要素単位であればいずれの方式であっても，転載されたコンテンツは既読として検出されている．一方，句読点で分割した場合，既読として検出されたコンテンツは 2 割程度であった．

(以下，‘LD’ はレーベンシュタイン距離を，‘CS’ は trigram による文脈ベクトルに対するコサイン類似度を表すものとする．)

5.2 実装間の性能比較

第 4 章において示した「スタンドアローンモデル」と「クライアントサーバモデル」という実装の差異により所要時間にどのような影響が出るかを調べるために，以下の条件で既読コンテンツの検出・表示を行い，検出および書式設定に必要な所要時間を計測した．

固定条件 書式設定: 背景色，類似度関数: コサイン，検出単位: 文単位

変動条件 検出方法: スタンドアローン/クライアントサーバモデルの各方法

スタンドアローンモデルおよびクライアントサーバモデルにおける所要時間を表 1 に，クライアントサーバモデルにおいて背景色変更を行った際の検出方法による所要時間の変化を図 7 に示す．図 7 のうち右側は，左側の「コサイン類似度 0.8 を閾値とした場合」の部分を拡大したものである．また，図 7 のうち “F.C.” は文字色，“B.C.” は背景色，“Size” は文字サイズを変化させた場合の所要時間を表す．

この結果から，既読コンテンツの検出部分を外部のサーバに置くことにより，同じ検出処理であってもより高速に実行できることが分かった．特に，コサイン類似度 0.8 を閾値とした一律書式設定においては，スタンドアローンモデルでは検出および書式設定に 3 分程度かかっており実用的ではなかったものが，クライアントサーバモデルを用いることにより 2 秒程度に短縮され，実用に足る処理速度となった．しかし，類似度に応じた書式設定の所要時間に着

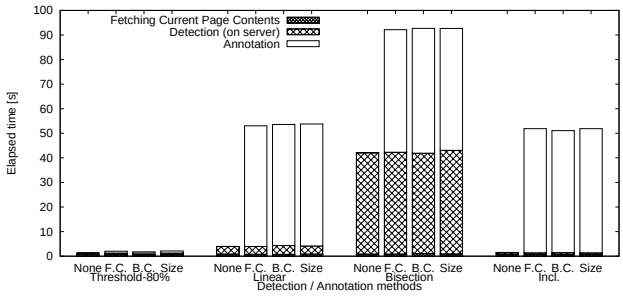

図 7 検出方法による既読コンテンツ検出・表示の所要時間の変化
Fig. 7 Elapsed time for detecting and annotating already-read contents by detection methods.

	Threshold 80%	Gradation
Per Element	0.023	0.070
Per Punc.	0.523	48.676

表 2 既読コンテンツの書式設定所要時間 [s]
Table 2 Elapsed time for annotation of already-read contents.

目すると，双方の実装間で変化は見られず，50 秒から 60 秒程度となっている．また，クライアントサーバモデルにおける類似度に応じた書式設定の所要時間は，どのような書式設定を行うかにかかわらず 49〜50 秒であった．

5.3 書式設定処理時間

次に，検出単位による書式設定処理時間の変化を調べるために，次の条件で書式設定に必要な所要時間を計測した．

固定条件 書式設定: 背景色，非コンテンツ部: 無視する，類似度関数: コサイン

変動条件 検出単位: 要素単位/句読点単位，検出方法: クライアントサーバモデルの閾値 80%/線形探索

所要時間の計測方法は，Firefox における JavaScript API である `console.time()`，および `console.timeEnd()` の組を書式設定処理の前後に挿入することで行った．実験の結果を表 2 に示す．

その結果から，いずれの書式設定方法を用いた場合でも呼び出される処理や関数はほぼ同じである（唯一異なる部分は類似度の値）にもかかわらず，句読点単位の処理であり，かつ連続的書式設定の場合の所要時間が極端に長いことがわかった．

5.4 既読コンテンツの量による検出時間

このアドオンは使用を続けると蓄積された既読コンテンツの量が多くなり，その結果検出に必要な所要時間が延びるものと考えられる．

蓄積された既読コンテンツの量が処理時間に与える影響を調べるために，任意の大きさの既読コンテンツデータベースを作成し，それに対して検出を行いサーバ上の所要時間を計測する実験を行った．

このデータベース中のテーブルでは，実際には同一の既

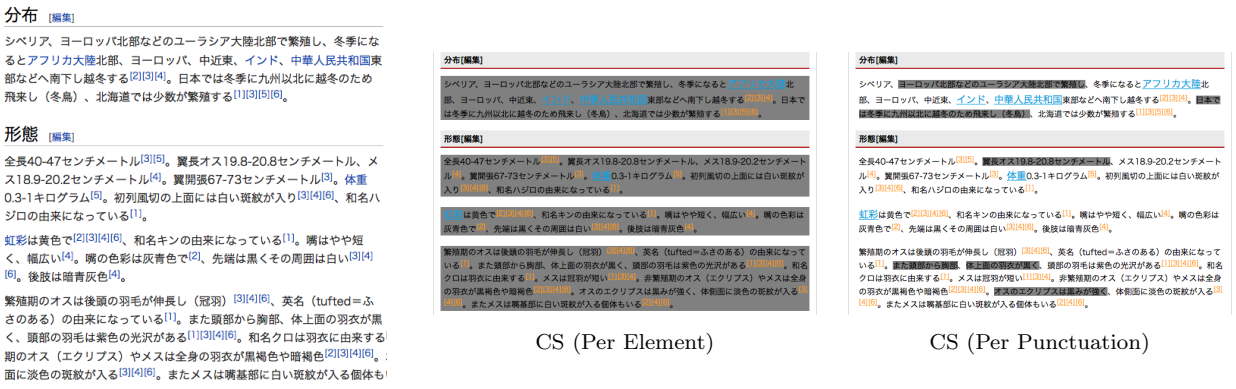

図 6 転載サイトに対する既読コンテンツ検出の結果

Fig. 6 Results for detecting already-read contents on a reproduced website.

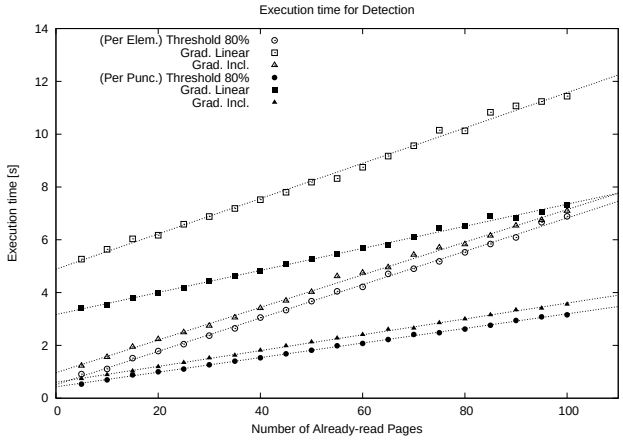

図 8 既読コンテンツの量による検出所要時間の変化 [s]

Fig. 8 Elapsed time for detection by the amount of already-read contents.

読コンテンツファイルが複数回参照されるが，見かけ上は複数の既読コンテンツが記録されているように見える．この既読コンテンツファイルには 62 個の HTML 要素データが含まれており，句読点単位で分割された文字列データは合計で 1247 個存在する．

所要時間の計測は，検出サーバ上で Wireshark を稼動させた上で次の条件で既読コンテンツの検出を行い，Wireshark に表示されるタイムスタンプを見ることにより行った．所要時間は検出サーバにリクエストの最初のパケットが到達してから，それに対する応答 (ACK) を送信し終わるまでの時間差とする．

固定条件 書式設定: 無し（検出のみ），非コンテンツ部: 無視する，類似度関数: コサイン，時間によるフィルタリング: 無し

変動条件 検出単位: 要素単位/句読点単位，検出方法: 閾値 80%/線形探索/二分探索/傾斜配分探索，既読コンテンツ数: $5, 10, 15, \ldots, 100$

実験の結果を図 8 に示す．図 8 の点線部分は各系列の線形回帰による近似曲線である．

いずれの条件においても，所要時間は既読コンテンツの量により線形的に変化した．また，いずれの条件においても決定係数 $R^2 > 0.995$ であった．

1 回の既読コンテンツファイル処理あたりの所要時間に着目すると，「二分探索」以外における所要時間は 10ms オーダーであり，「二分探索」では 100ms オーダーであった．

6. 考察

これらの性能評価の結果から，HTML 要素単位で検出を行う場合は，算出された類似度に応じて該当する要素の書式を変更することで 3.1 節で述べた表示方法は容易に実装でき，かつ高速な書式設定処理が可能であることが分かった．

しかし，それよりも小さい単位で検出・表示を行おうとした場合，該当する文字列を，それを含む要素の中から指定しなければならない．そのため，4.1.2 節で述べたタグ挿入処理が行われる．その際，当該 HTML ソース中に何らかの要素の開始タグが存在するが，それに対する終了タグが当該 HTML ソースの外部にある場合，span タグを挿入することで，HTML 要素の入れ子構造が崩れてしまうことになる．

実際に，5.1 節において行った実験では，要素単位の検出では転載先のサイトにおいて転載元コンテンツを検出・表示することに成功したが，句読点単位では検出は行なわれたものの書式設定に失敗している領域が発生していた．

また，この書式設定用タグ挿入処理により所要時間も増大した．スタンドアローンモデルによる 5.1 節での実験では，特に句読点単位の検出において，書式が設定されるまでに最大で 5.9 秒の待ち時間が発生した．しかし，要素単位の検出では，表示方法にかかわらず 2 秒以内に処理が終了した．特に，句読点単位の書式設定を行った際の検出・書式設定を合わせた所要時間は，平均で要素単位の 3.56 倍であった．また，句読点単位の検出であっても，検出のみで書式設定を行わない場合の処理時間は 3 秒以内に収

まった．

これに加え，図 7，および表 2 が示すように，句読点単位の書式設定であっても類似度に応じて書式を段階的に変化させるかどうかにより書式設定の所要時間が大幅に増大することがわかった．該当する処理を記述するソースコードがほとんど同一であるにもかかわらずこのような差異が生じる原因は不明である．

7. おわりに

本研究では，検索エンジンを利用するにあたってしばしば問題となる重複コンテンツに対し，それらのコンテンツを読み飛ばすことを補助する既読コンテンツの検出・表示システムの提案，およびウェブブラウザ用拡張機能による本システムの構築を目指した．また，既読コンテンツの蓄積・検出部分を外部のサーバに移動した場合の性能の変化についての評価を行った．

最初にウェブブラウザ用拡張機能として実装したシステムに対し評価実験を行い，レーベンシュタイン距離およびコサイン類似度による類似度算出が重複コンテンツの検出に対して有効であることが示された．しかし，所要時間の面では，ページ内容や検出単位等の条件により，検出および書式設定の所要時間が数十秒から数分に達することがあるため，実用的なシステムとはいえない．

そこで，拡張機能とサーバプログラムを組み合わせた実装を行い，従来の実装との性能比較を行ったところ，句読点単位での検出時間が従来の実装と比較して 0.73%から 23.5%程度に短縮された．また，100 ページ分程度の既読コンテンツであれば，検出方法を工夫することにより 10 秒以下で検出が完了し，実用に足ることが示された．

しかし，類似度に応じて連続的に書式を設定する処理を句読点単位で行った場合，所要時間がそれ以外のものと比較して極端に大きくなるため，そのような条件下においては未だ実用的ではない．

今後の課題として，最初に多言語対応が挙げられる．特に，本研究では日本語のページのみを想定していたが，英語のように分かち書きを行う言語の場合は，文字単位ではなく単語間 N-gram により特徴量をとる必要がある．

また，形態素解析により分割された単語の類似度を利用することで，検出精度が高まる可能性がある．しかし，形態素解析されたデータ間の類似度を算出するために単語の重要度を得る方法として知られている tf-idf 法は非常に高コストであるため，一般的なパーソナルコンピュータ上で実行することは現実的ではない．したがって，既に算出済の IDF データを利用することが必要となる．

表示手法に関しては，ある表示手法がどれほど見やすかったか・効果的だったか等の評価は主観的なものであるため，幅広い層の被験者を対象とした実験が必要である．

本研究における実装上の課題としては，クライアントサーバモデルによる実装において既読コンテンツ蓄積・検出用サーバを外部に配置した場合（クラウドサービスとしての運用），既読コンテンツをその都度サーバへアップロードすることによる帯域幅の圧迫の可能性，および「ユーザーの読んだコンテンツ」を外部に送信することによるプライバシーの侵害への懸念が挙げられる．

プライバシー侵害を避けるため，TLS(SSL) を用いてクライアント―サーバ間の通信が第三者に読み取られないようにする，データベースの内容をユーザー毎に暗号化し，サーバ管理者であったとしてもその内容を読み取れないようにするなど，適切なプライバシー保護手段を講じる必要がある．

参考文献

[1] Henzinger, M.R., Motwani, R., and Silverstein, C.: *Challenges in web search engines*, SIGIR Forum, vol.36, no.2, pp.1122, Sept. 2002. 入手先 ⟨http://doi.acm.org/10.1145/792550.792553⟩
[2] Google: Personal Blocklist, 入手先 ⟨https://chrome.google.com/webstore/detail/nolijncfnkgaikbjbdaogikpmpbdcdef⟩
[3] Lieuallen, A., Boodman, A. and Sundstrm, J.: Greasemonkey, 入手先 ⟨http://www.greasespot.net/⟩
[4] Biniok, J.: Tampermonkey, 入手先 ⟨https://tampermonkey.net/⟩
[5] Scher, J.: Google Hit hider by Domain, 入手先 ⟨https://greasyfork.org/scripts/1682⟩
[6] Jobson, P.: Google Domain Blocker, 入手先 ⟨https://github.com/pjobson/pjUserscripts⟩
[7] Salton, G., Wong, A. and Yang, C.: *A vector space model for automatic indexing*, Communications of the ACM, vol.18, no.11, pp.613620, 1975
[8] Sankoff, D. and Kruskal, J.B.: *Time warps, string edits, and macromolecules: the theory and practice of sequence comparison*, Addison Wesley, Reading, Massachusetts (1983).
[9] 岡崎直観，辻井潤一：高速な類似文字列検索アルゴリズム，情報処理学会創立 50 周年記念全国大会 (2010)，入手先 ⟨http://www.chokkan.org/software/simstring/⟩
[10] Kudo, T.: *Mecab: Yet another part-of-speech and morphological analyzer*, 入手先 ⟨http://mecab.sourceforge.net/⟩