情報処理学会論文誌　データベース　Vol.8　No.2　43–56 (June 2015)

共同執筆コンテンツにおける単語の起源追跡

中村 晃[1,†1,a)]　鈴木 優[2,b)]　石川 佳治[1,c)]

受付日 2014年12月21日, 採録日 2015年2月5日

概要：現在，Web 上には Wikipedia を代表としたコラボレーションプラットフォームが多数存在する．本論文では，版管理された共同執筆型のコンテンツに対して，記述の起源を追跡する手法を提案する．不特定多数の編集者がコンテンツの編集に関与する Wiki システムや，ソフトウェアの共同開発を前提としたコード管理システムにおいて，記述の正確な起源の特定は重要である．実際に，Wikipedia における編集者や記事の質推定などのように，記述の起源を利用した研究や応用アプリケーションがすでに存在する．しかし，記述に対する正確な起源の付与は，記述の復元を考慮する必要があるため容易ではない．なぜならば，記述が復元された場合，削除以前の記述の起源を参照する必要があるからである．既存手法では，小さな粒度の記述の復元を検出することは困難であった．そこで，本研究では削除された記述の位置を保持したまま管理することによって，小さな粒度での復元を厳密に検出し，記述の起源を正確に推定する．評価実験では Wikipedia 日本語版において，人手により特定した 186 件の単語の起源とシステムの推定した起源との照合を行った．その結果，提案手法の正解率は 86.0%となり，既存手法と比較して 24.7 ポイントの精度向上を確認することができた．

キーワード：Wikipedia，起源，オーサーシップ，バージョン管理，共同執筆

Provenance Tracking of Terms in Collaborative Authoring Systems

Akira Nakamura[1,†1,a)]　Yu Suzuki[2,b)]　Yoshiharu Ishikawa[1,c)]

Received: December 21, 2014, Accepted: February 5, 2015

Abstract: Numerous collaboration platforms on the Web are used in order to share and edit documents or source code. We propose a method of provenance tracking for collaborative authoring systems having revisioned contents such as Wiki systems or code management systems. Accurate provenance of each text is important and have potential applications. Actually, studies and applications utilizing provenance already exist, such as a study of measuring quality in Wikipedia. However, attributing accurate provenance to texts is difficult because of restoration. Provenance of restored text should refer to provenance of the text before deletion. Restoration detection of small granularity like a term level is difficult for existing techniques. Our proposed method manages provenance with keeping positions of deleted terms to detect small granularity restoration strictly, and to track provenance exactly. In evaluation experiment, we used 186 article-term sets chosen at random from Japanese Wikipedia as a dataset. We compared provenance determined by systems and true provenance manually labeled by observers. As a result, accuracy of our proposed method is 86.0% on this dataset, and outperforms accuracy of the state-of-the-art algorithm with increases of 24.7 points.

Keywords: Wikipedia, provenance, authorship, version control, collaborative authoring

[1] 名古屋大学大学院情報科学研究科
Graduate School of Information Science, Nagoya University, Nagoya, Aichi 464–8603, Japan
[2] 奈良先端科学技術大学院大学情報科学研究科
Graduate School of Information Science, Nara Institute of Science and Technology, Ikoma, Nara 630–0192, Japan
[†1] 現在，SCSK 株式会社
Presently with SCSK Corporation
[a)] nakamura@db.ss.is.nagoya-u.ac.jp
[b)] ysuzuki@is.naist.jp
[c)] ishikawa@is.nagoya-u.ac.jp

1. はじめに

現在, Web 上では多数のコラボレーションプラットフォームが利用されている．Wiki システムやコード管理システム，グループウェアなどの協業ツールがその筆頭である．そのようなコラボレーションプラットフォームにおいては，複数のユーザが共同で同じコンテンツを作成する．た

とえば，Wiki システムでは複数の編集者が共同で同一のドキュメントを執筆する．コード管理システムにおいては，複数の開発者が同一のソースコードを編集する．

これらのコンテンツは，しばしば版ごとに管理される．つまり，コンテンツが編集されるたびに，その時点での版の状態が記録される．たとえば，Wikipedia[*1]においては，すべての記事に対して，記事を立ち上げた版から最終版までの各版の全記述とその編集者，および編集した日時が記録されている．

版ごとに管理されたコンテンツにおいては，過去の版を容易に参照できる．コンテンツにどのような変更が加えられたのかを知りたい場合，ユーザは過去の版を参照すればよい．しかし，すべての版が記録されていても，最新の版における各記述が，いつ誰によって書かれたのかを容易に知ることはできない．記述の起源を特定するためには，過去の版を順に遡り，その記述が最初に追加された版を探さなければならない．膨大な版数を持つコンテンツにおいては，このような特定を人手で行うことは困難である．したがって，自動的に記述の起源を追跡するシステムが必要となる．そこで本論文では，版管理された共同執筆コンテンツにおいて，記述の起源を追跡する手法を提案する．

不特定多数のユーザがコンテンツの形成に関与する場合，記述の起源には多くの価値があると考えられる．誰もが編集に携わることのできる共同執筆コンテンツにおいては，悪意を持った編集者や質の低い編集者の存在が問題となる [1] が，そのような編集者が存在する限り，記述の信憑性に確証を持つことはできない．しかし，記述に起源を付与できれば，権威ある編集者による記述かどうかを信憑性の基準とすることができると考えられる．また，記述が削除されず残存している期間やその残存率を利用して，記述の質を推定するといった試みも存在する [2], [3], [4], [5], [6]．

上述した記述の起源を利用した研究の多くは，単純に差分を用いて起源の特定を行っている．ある編集が行われたとき，編集前の版と編集後の版の差分をとることにより，編集によって追加された記述と削除された記述，変更されず残存した記述を識別することができる．追加された記述はその編集を起源とし，変更されなかった記述は編集前の起源を引き継げばよい．これを最初の版から最終版までの各版のペアに対して順に適用することにより，各記述に起源を付与する．

ところが，この手法では，復元された記述に対して正しい起源を付与できない．復元とは，追加された記述が後の版で削除され，その後に再び同一の記述が追加されることである．復元の詳細な定義は 3.1 節において述べる．復元された記述の起源は，削除前のその記述の起源とすべきである．しかし，単純な差分を用いた手法では，記述が復元された編集を起源とする誤りが生じる．したがって，復元を考慮しない手法を用いた場合，正確な記述の起源を特定できない．

復元を反映させるためには，過去の版における記述を保持しておき，同一の記述が追加された場合に復元を検出して起源を紐付ける方法が考えられる．しかし，この復元検出方法は，記述に一意性があることを前提にしている．コンテンツ上の複数箇所に出現するような一意性のない記述に対してこの手法を適用した場合，実際には復元ではない記述の追加に対しても，復元を誤検出する可能性が生じる．誤検出の例として，コンテンツのある版において「リンゴの苗木が寄贈された」という記述が削除された状況を想定する．その後の版において，別の箇所に「リンゴの漢名は苹果である」という記述が追加されたとき，上述の復元検出方法を単語単位で適用すると，「リンゴの」という記述が復元であると判定される．しかし，「リンゴの」は復元された記述であると考えるよりも，新たに追加された記述であると考えるほうが自然である．これは，「リンゴの」という記述に一意性がないために生じる誤りである．

記述の一意性は，その記述の粒度の大きさに比例すると考えられる．記述の粒度とは，段落単位，文単位，単語単位のように，区切ることのできる文字列のまとまりのことである．段落程度の粒度の大きさを持つ記述であれば一意性があり，この復元検出方法を問題なく適用できると考えられる．一方で，単語単位のように段落単位や文単位と比べて粒度の小さい記述には一意性がなく，この復元検出方法を適用した場合には復元の誤りが生じる．そのため，既存研究 [7] においては，復元の検出対象を文単位以上の粒度を持つ記述に限定している．既存の手法では，単語単位で正確な復元を検出することは困難である．小さな粒度での記述の復元が，起源追跡の難しさの大きな原因となっている．

本研究では，より正確な起源を特定するために，単語単位で正確な復元検出を行うことを目指す．我々は，記述の復元を検出する際に，記述の出現する文脈が削除前と追加後で同一ならば，粒度が小さい記述でも復元の判定を行うべきであると考えた．しかし，曖昧な情報である文脈を，機械的にかつ効率的な方法でとらえることは難しい．そこで，本研究では同じ文脈に出現する単語かどうかを判定するために，単語の位置に着目する．すなわち，単語が削除される前と同じ位置に追加されているかどうかを復元の判定の基準とする．ただし，削除された単語はその後の版に存在しないため，削除された単語の位置を基準として利用することができない．提案手法は，削除された単語を削除前の位置に保持したまま版データを管理することによって，削除された単語の位置および起源を把握し，単語単位での復元を検出可能にする．これにより，小さな粒度での記述の復元をより厳密に検出し，記述の起源を単語単位で，正

[*1] http://ja.wikipedia.org/

確に追跡できる．

2. 関連研究

2.1 記述の起源を求めるための要素技術

まず，起源特定のために用いられる要素技術について述べる．版管理されたコンテンツにおける記述の起源を追跡するためには，最初の版から最新の版まで順に，2 版間の単語の対応付けを行う必要がある．これは最長共通部分列問題 [8] におけるアルゴリズムを適用することができる．最長共通部分列問題は，2 つの配列において共通の最長部分列（LCS）を発見するという問題であり，diff などのファイル比較に利用される．起源に関する既存研究では，diff を用いて検出した LCS に基づき，2 版間の記述の対応付けおよび起源の付与を行っている．最長共通部分列問題では，各配列中で共通の要素を初めから終わりまで順に対応付けるため，対応付けが交差することはない．しかし本研究では，2 言語間の文対に対する単語対応付け問題 [9] のように，交差した対応付けを許したアルゴリズムを適用する．詳細は 3.3.2 項において述べるが，この方法を適用することにより，章の再編成などにおいて記述の移動が行われた場合にも，起源の欠損なく対応付けることが可能になる．

Wiki システムでは，記述の状態を過去のある版の状態に戻すことを**差し戻し**という．また，削除された記述の復元や，追加された記述の削除によって対象の編集におけるすべての行動を打ち消すことを**取り消し**という．差し戻しや取り消しの検出によって，復元された記述の起源を削除前の記述の起源に結びつけることができる．したがって，差し戻しや取り消しの検出は，より正確な起源の抽出にとって有用である．差し戻しの基本的な検出方法は Kittur ら [10] や Halfaker ら [11] による手法である．この手法では，編集後の版における全記述が，それよりも前の版における全記述と一致するかどうかを，差し戻しの判定基準としている．一致した場合，その版間に存在する編集が取り消されたと検出される．

編集者が，1 つ前の版で行われた編集を取り消した場合，編集後の版は 2 つ前の版と完全に一致する．したがって上述の手法により，差し戻しおよび取り消しを検出可能である．ところが，2 つ以上前の版で行われた編集を取り消した場合，編集後の版は過去の版とは完全に一致しない可能性がある．すなわち，取り消しを行ったとしても，取り消した編集より後に別の箇所が改訂されていた場合，上述の手法では取り消しを検出することができない．Flöck ら [12] は，この問題を解決すべく，新たな取り消し検出手法 DIFF を提案している．DIFF は，版間における記述全体の一致ではなく，編集で追加された単語集合および削除された単語集合の，編集間における包含関係に着目した手法である．DIFF の取り消し検出基準について簡潔に述べる．s 番目の編集において追加した単語集合を W_a^s，削除した単語集合を W_r^s と表す．それより過去の t 番目の編集において，追加した単語集合を W_a^t，削除した単語集合を W_r^t と表す．このとき，$W_a^t \subseteq W_r^s \wedge W_r^t \subseteq W_a^s$ となるとき，s 番目の編集は t 番目の編集を取り消したと検出する．

DIFF は単語の種類と数だけを考慮する．したがって，微量の編集が行われたときに，その後の編集で誤った取り消し検出が行われる場合がある．例として，単語「の」が 1 語だけ削除される編集が行われた状況を考える．その後の編集において，単語「の」の追加を含む編集が行われたときに，DIFF は取り消しを検出する．しかし，「の」は一般に出現頻度の高い単語であり，誤った取り消し検出が行われる可能性は十分にある．本研究では，復元の検出時に単語の位置の情報を用いることによって，このような誤検出を防いでいる．また，DIFF は取り消しを検出する手法であって，記述の復元を検出する手法ではない．取り消しの検出によって抽出される復元は，実際に存在する復元全体のうちの一部にすぎない．過去のある版において削除された記述の一部分だけを復元した場合には，取り消しに該当しないために，上述の手法では復元を検出できない．我々の手法ではそのような復元も検出が可能である．網羅性の高い復元検出を行うことによって，より正確な起源追跡を行っている．

2.2 記述の起源を求める研究

次に，本研究と同様に起源追跡そのものを目的とした研究を紹介する．de Alfaro ら [13] や，Flöck ら [7], [14] は版管理されたコンテンツの記述に対して，復元を考慮しつつ単語単位で起源を付与する手法を提案している．de Alfaro ら [13] の手法では，版の履歴はトライ木によって保持される．編集において追加された単語を周辺の単語とともに単語シーケンス化して，版の履歴から探索することにより復元を検出する．Flöck ら [7], [14] の提案する WikiWho は，コンテンツの全版を，版，段落，文，単語の粒度に分割し，それらをノードとする k 部グラフを用いた手法である．コンテンツの編集によって新たな版が作られたとき，WikiWho はまず版のノードを作成する．そして，版における記述を，段落に分割する．このとき，過去の版において同一の記述内容を持つ段落ノードがあった場合，現在の版からその段落にエッジをはる．段落が過去の版に存在しない場合は，新たにその段落のノードを作成し，現在の版からエッジをはる．エッジには，上位のノードにおける下位のノードの順番がラベル付けされる．つまり，版における段落の順序に従って，エッジにラベル付けが行われる．新たに作成した段落ノードは，文に分割され，同様の処理が行われる．新たに作成した文ノードは，単語に分割されるが，文ノードから単語ノードへのエッジは，2 版以上遡ることはない．したがって，WikiWho では，単語単位の復元を検出できない場合がある．単語の復元によって，単

語の所属する文全体が過去の版に存在する文と一致すれば復元を検出できるが，一致しなければ単語が復元されていても WikiWho では検出できない．単語単位での復元の検出漏れにより，起源付与の精度が低下すると考えられる．

我々の知る限り，WikiWho は単語単位の起源付与問題において最高の精度を持つシステムである．したがって，本研究では起源付与の精度において WikiWho の精度を上回ることを目指す．

2.3　記述の起源を利用した研究

共同執筆コンテンツの 1 つである Wikipedia では，不特定多数のユーザが編集に参加できるという特徴から，質が低い情報が含まれるという欠点が存在する [1]．そこで，Wikipedia の記事や編集者，記述に対して質の高さを測定する研究が行われている．質の測定には，記述の残存率や残存の期間が利用される場合がある [2], [3], [4], [5], [6]．記述の残存に関する情報を抽出するためには，記述の起源を特定する必要がある．また，質の測定のために，編集者間における記述の残存関係や削除関係を利用する試みが行われている [15]．編集者間の関係抽出にも同様に，記述の起源が必要となる．

質の測定以外にも，起源の特定を前提とした編集者間の関係を用いた研究として，編集者ネットワークの可視化に関する研究 [16] や，バイアスごとに編集者のクラスタリングを行った研究 [17] があげられる．

上述したこれらの研究は，記述の起源を正確に取得できるという仮定に基づいている．しかし，これらの研究の多くは，起源の特定に diff を用いた単純な差分抽出手法を適用しており，記述の復元は考慮していない．または，前述の差し戻し検出手法に従って復元の一部を検出し，起源を対応付けている．このような単純な手法では，復元の検出が不十分なため，正確な起源を抽出することができない．Adler ら [4], [5], [6] は，追加された単語シーケンスの最長のマッチを過去の版から探索することによって復元を考慮した起源の追跡を行っている．しかし，復元の判定基準に単語の位置を考慮していないために，1 章で述べたように単語単位の復元を正確に検出することは困難である．

本研究では厳密な復元の検出を行うため，これらの起源追跡手法よりも高精度に起源を特定できる．したがって，上述の起源情報を利用した研究に対して性能の向上に貢献できると考えられる．

3.　提案手法

本章では，版管理された文書形式のコンテンツにおける，記述の起源を単語単位で追跡する手法について述べる．

3.1　定義

文書の最終版が形成されるまでのすべての版を含むデータを，**編集履歴**と呼ぶ．一般に，編集履歴には各版におけるコンテンツの記述や編集者の情報，版が生成された日時などが含まれる．本章では簡単化のため，コンテンツにおける全版の記述だけを編集履歴として取り扱うものとする．

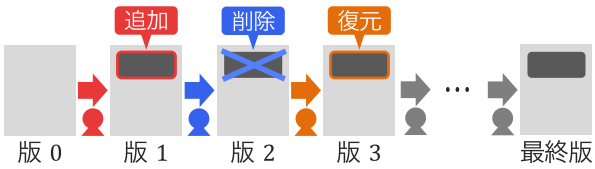

図 1　記述の復元
Fig. 1　Restoration of text.

コンテンツの編集履歴を $C = \{c_0, c_1, \ldots, c_{n-1}\}$ と表す．$c_i \in C$ は i 番目の版におけるコンテンツの全記述を意味する．本手法は，この編集履歴 C を入力として，記述の起源を単語単位で追跡する．そして，最終版の各単語に対して 1 つずつ起源を割り当てた結果を出力する．ここでいう起源とは，記述が最初に文書に追加された版のことである．ただし，記述の復元が行われたとき，その記述の起源は削除される前の記述の起源に紐付けられる．以下に復元の例を示す．

図 1 のように，版 1 で追加された記述が版 2 において削除され，再び版 3 において同一の記述が追加されたとき，これを復元と見なす．すなわち，記述の起源は版 3 ではなく，最初に追加された版 1 となる．本研究ではこの例のように，復元を考慮して起源を紐付けることによって，より人間の直感に近い起源の特定を目指す．

提案手法はコンテンツの編集履歴 C とともに，以下に示す 4 つの閾値 k，l，r，h を入力とする．

- k：2 版間での対応付けを許す最小の連続共通単語数．
- l：シーケンスが一致する復元と見なす最小の連続共通単語数．
- r：位置が一致する復元と見なす最小の連続共通単語数．
- h：復元の検証対象とする版数．削除されてから過去 h 件以内の復元を検証する．

それぞれの閾値に関連した処理の詳細は 3.3 節において述べる．

3.2　基本的な方針

1 章で述べたように，復元を考慮しない場合，記述の起源は，編集履歴中の最初の版から最終版までの各版のペアに対して順に差分をとり，単語の対応付けを行うことによって求めることができる．以下に単語の対応付けの例を示す．

図 2 は，コンテンツの編集履歴における版 0 と版 1 に焦点をあてて，2 版間の差分と単語の対応関係を表した図である．この図では，編集者 A が初めにコンテンツを立ち上げ，版 0 を生成している．版 0 は単語 a–f からなる文書で

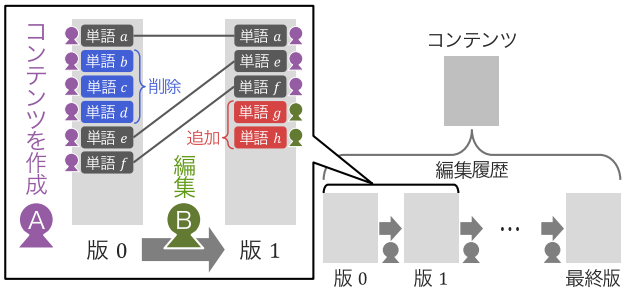

図 2　単語の対応付けと差分の抽出

Fig. 2　Mapping and detection of removed terms and added terms.

ある．したがって，これら a–f の単語の起源は当然，すべて版 0 である．次に，編集者 B が版 0 の文書を改変し，版 1 を形成している．B が編集した後の版 1 の記述は，単語 a，e，f，g，h からなる．これら 2 版の差分をとり，単語の対応付けを行うと，B の編集によって単語 b–d が削除され，単語 g，h が新たに追加されていることが分かる．したがって，単語 g，h の起源は版 1 となる．そして，残りの単語 a，e，f は変更されず，版 0 から引き継がれているため，版 0 の対応する単語の起源を紐付ければよい．すなわち，単語 a，e，f の起源は版 0 である．この 2 版間の差分抽出を，最終版まで順に実行し，単語とその起源を管理する．つまり，版 0 と版 1 の次は版 1 と版 2，その次は版 2 と版 3 というように，版間の差分抽出と単語の対応付けを順に行いつつ，各版における単語と単語に紐付いた起源を管理していく．これにより，最終版の各単語に対して起源を特定することができる．

上記の手法によって求められる起源は，復元を考慮していない．そのため，単語に対して誤った起源を付与する可能性がある．そこで我々は，復元を検出し，正確に単語の起源を追跡するために，上記の手法をもとに，単語の復元に対応した新しい手法を提案する．

復元を考慮して記述の起源を求めるためには，追加された記述に対して復元の可能性を検証すればよい．すなわち，追加された記述が過去に存在した記述であるかどうかを調べればよい．もし過去の版に存在した記述であれば復元と見なし，記述の起源を復元もとの記述の起源に紐付ける．しかし，1 章で述べたように，単語単位のような粒度の小さな記述に対してこの方法を用いた場合，復元を誤検出してしまうおそれがある．これは，粒度の小さな記述には一意性がなく，同様の記述が文書中の複数の箇所に出現する可能性があるためである．つまり，記述が過去と同様の文脈において使用されているかどうかを考慮して復元かどうかの判定を行う必要がある．ところが，文脈を機械的に判断して復元の判定を行うことは困難である．特に，Wikipedia などの膨大な文書数，版数，記述量を持つデータに対して，文脈を考慮した効率的な復元の検出を行うことは難しい．そこで提案手法では，粒度の小さい記述に対して，その位置をもとに復元の検出を行い，起源の追跡を行う．つまり，削除された単語がその後再び同じ位置に追加された場合，それを復元と判定する．位置を考慮することにより，復元の誤検出を防ぐことができ，復元の検出精度を向上させることができると考えられる．

本手法では，復元の検出のために，削除された単語の位置を保持しておく必要がある．したがって，各版の単語と起源の管理を行う際に，削除された単語に関してもその位置を保持したまま残しておく．これにより，単語単位の復元判定にその位置を利用することができる．削除された単語と起源の管理方式に関しては，3.3.1 項で詳細に述べる．

提案手法の基本的な方針は上述のとおりである．次節では，提案手法の詳細について説明する．

3.3　共同執筆コンテンツにおける単語の起源追跡手法

前節で述べたように，版管理された文書形式のコンテンツにおける単語に対して起源を特定するという問題は，連続した 2 版間における単語の対応付けに落とし込むことができる．2 版間の単語の対応付けを最初の版から順に行い，復元を考慮しつつ起源を紐付けることにより，最終版における各単語の起源を特定することが可能となる．

したがって，以下では復元を考慮した 2 版間の単語の対応付けと，単語および起源の管理方法に関して説明する．2 版に着目したとき，本手法の手順は以下のとおりである．

(1) 2 版間における単語の対応関係をとる．
(2) 後の版において対応漏れした単語に対して，復元の可能性を検証する．
(3) 後の版における単語に起源を紐付け，起源管理データを更新する．

これらの手順を，入力として与えられたコンテンツの全版 $c_0, c_1, \ldots, c_{n-1}$ に対し順に実行する．そして，最後に最終版の時点における起源管理データを出力する．各手順の詳細な説明は，それぞれ 3.3.2 項，3.3.3 項，3.3.4 項において述べる．まずは 3.3.1 項で単語とその起源の管理方式について説明する．

3.3.1　単語と起源の管理方式

提案手法では，記述の起源を単語ごとに付与する．したがって，形態素解析器を用いて各版の記述 $c_i \in C$ を単語に分割する必要がある．ここでは c_i を単語に分割した結果の単語配列を $T_i = \{t_0, t_1, \ldots, t_{m-1}\}$ と表す．$t_j \in T_i$ は i 番目の版における記述 c_i のうち，j 番目に出現する単語を意味する．

本手法では，復元の判定において「単語の追加された位置が，削除前と同じかどうか」を 1 つの基準として用いる．したがって，現在の版には存在しない，削除された単語の位置およびその起源を何らかの方法で保持しておく必要がある．削除された単語の情報を管理するために，我々は **OFF 単語**の概念を導入する．**図 3** は OFF 単語を用い

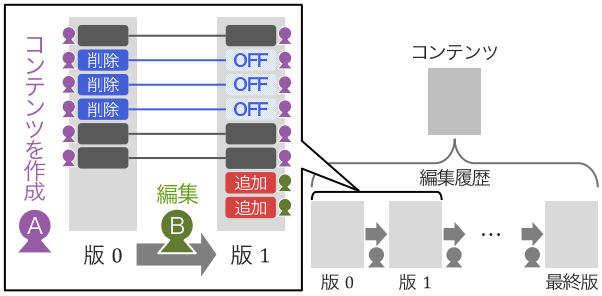

図 3　OFF 単語を用いた削除単語の位置と起源管理

Fig. 3　Management of positions and provenance of removed terms using off-terms.

表 1　図 3 の版 1 時点での D の例

Table 1　An example of D in revision 1 on Fig. 3.

	d_0	d_1	d_2	d_3	d_4	d_5	d_6	d_7
term	a	b	c	d	e	f	g	h
provenance	0	0	0	0	0	0	1	1
state	t	f	f	f	t	t	t	t
removed	0	1	1	1	0	0	0	0

た削除単語の位置と起源管理の例を表した図である．この図では，図 2 と同様に，編集履歴における版 0 と版 1 に焦点をあてている．編集内容も同一である．しかし，図 2 とは異なり，版 0 において削除された単語が版 1 では OFF 単語となり，そのままの位置に保持されている．このように，削除単語を OFF 状態にして保持することによって，後の版においても削除された単語の元の位置を把握することができる．なお本論文では，実際の記述上には出現しない OFF 単語と対比し，記述上に存在する単語のことを **ON 単語**と表記する．

次に，単語およびその起源を管理する方式について説明する．ある版の時点における各単語の起源を含む情報を管理するデータを $D = \{d_0, d_1, \ldots, d_{z-1}\}$ と表す．$d \in D$ はそれぞれ，以下のような 4 つの属性を持つ．

- *term*：単語の文字列を表す．
- *provenance*：単語の起源となる版番号を示す．
- *state*：*true* または *false* の値をとり，それぞれ ON 単語または OFF 単語であることを示す．*true* であれば T_i 中に出現する単語である．
- *removed*：OFF 単語ならば，最後に削除された版番号を表す．ON 単語ならば 0 の値をとる．

削除された単語はすべて OFF 単語として保持しておく．したがって，版数が多くなればなるほど $|D|$ は大きくなり，処理コストが増大する．そこで提案手法では，削除されてから版数 h が経過した単語については，そのデータを破棄する．そのために，単語ごとに最後に削除された版番号 *removed* を保持しておく．

表 1 に図 3 の版 1 時点での D の例を示す．*state* は *true* ならば t，*false* ならば f と略表記している．

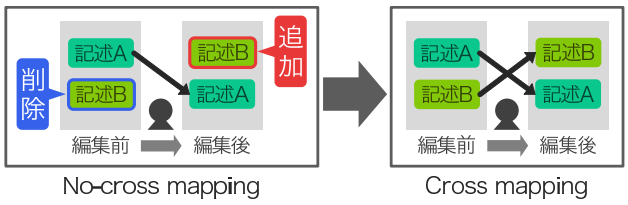

図 4　交差移動を許さない対応付けと許す対応付け

Fig. 4　No-cross mapping and cross mapping.

3.3.2　シーケンスを重視した単語の対応付け

前述のように，版管理されたコンテンツにおける単語の起源追跡は，最初の版から最後の版まで各編集前後の 2 版間における単語の対応付けを行えばよい．したがって，ここでは 2 版間における単語の対応付け方法について説明する．

コンテンツの編集履歴における，ある編集前後の版に着目する．編集前の版を l 個の単語群 $T_p = \{t_0, t_1, \ldots, t_{l-1}\}$ と表し，編集後の版を m 個の単語群 $T_n = \{t_0, t_1, \ldots, t_{m-1}\}$ と表すとき，T_p と T_n の単語の対応付けを行う．すなわち，$f : T_p \to T_n$ となる部分写像 f 求める．f は単写である．本論文では，f およびその逆部分写像 f^{-1} における対応関係を M と表記する．

通常，文書間の単語の対応付けを行う場合には，最長共通部分列問題 [8] を適用する．提案手法では最長共通部分列問題のアルゴリズムを用いず，以下に定義する**シーケンスを重視した単語の対応付け手法**を適用する．この手法は，2 文書間に共通する連続単語を，連続性の高い単語群から優先して対応付けするという手法である．最長共通部分列問題では，各文書中の単語を初めから終わりまで順に対応付けするが，本手法は対応付けの際に単語群の順序を考慮せず，完全に一致した単語群のうち，サイズの大きいものから順に対応付ける．**図 4** のように単語群の対応付けが前後で交差する場合に，最長共通部分列問題を適用すると，図中左側の対応付け結果のように対応漏れが発生するのに対して．シーケンスを重視した単語の対応付け手法では，図中右側の対応付け結果のように漏れなく対応付けることができる．すなわち，コンテンツ形成の過程において，章の再編成などにより記述が交差して移動した際に，起源の欠損を防ぐことができる．

シーケンスを重視した単語の対応付け手法のアルゴリズムを**図 5** に示す．T_p と T_n，および，対応付けを許す最小連続単語数の閾値 k が与えられたとき，T_p と T_n 間での単語の対応付け結果 M を返す．本手法の手順は以下のとおりである．

(1) **索引の構築**：T_n に対して k 単語索引を作成する．
(2) **共通シーケンスの洗い出し**：索引を活用し，T_p と T_n の間で共通する連続単語を洗い出す．
(3) **共通シーケンスの割当て**：シーケンスの大きさ順に単語を割り当てる．

```
Input: terms T_p = {t_0, t_1, ..., t_{l-1}} in a previous revision,
    terms T_n = {t_0, t_1, ..., t_{m-1}} in a next revision, a threshold
    k
Output: matches M
 1: /* 索引の構築 */
 2: create an empty k-terms index I
 3: for i ← 0 to l − k do
 4:     key ← k-terms sequence t_i, t_{i+1}, ..., t_{i+k-1} ∈ T_n
 5:     add ⟨key, i⟩ to I
 6: end for
 7: /* 共通シーケンスの洗い出し */
 8: create an empty sequences list S
 9: create an empty previous positions set V_p
10: for i ← 0 to m − k do
11:     key ← k-terms sequence t_i, t_{i+1}, ..., t_{i+k-1} ∈ T_p
12:     V ← I.getValues(key) // T_n における key の位置を取得
13:     for j ∈ V do
14:         if V_p contains j − 1 then
15:             update size of a corresponded sequence seq ∈ S
16:         else
17:             create a new sequence seq
18:             seq.size ← k
19:             seq.start_prev ← i
20:             seq.start_next ← j
21:             add seq to S
22:         end if
23:     end for
24:     V_p ← V
25: end for
26: /* 共通シーケンスの割り当て */
27: sort S in descending order of size
28: create an empty matches M
29: for i ← 0 to |S| − 1 do
30:     if seq_i ∈ S is overlapped with assigned sequences then
31:         cut seq_i to avoid overlaps
32:         if k ≤ seq_i.size then
33:             insert seq_i at a proper position of S
34:         end if
35:     else
36:         for j ← 0 to seq_i.size − 1 do
37:             add ⟨seq_i.start_prev + j, seq_i.start_next + j⟩ to M
38:         end for
39:     end if
40: end for
41: return M
```

図 5　交差移動を許す単語の対応付けアルゴリズム
Fig. 5　Cross mapping algorithm.

各ステップについて解説する．

はじめに，T_n 中の単語を順にたどり，n-gram のように T_n における k 連続単語の位置を示す索引を構築する．これは，図 5 の 2–6 行目に該当する．あらかじめ T_n 中における k 単語シーケンスの索引を構築することにより，次のステップである共通シーケンスの洗い出しの効率化を図る．

次に，T_p と T_n の間で共通する連続単語を洗い出す．これは，図 5 の 8–25 行目に該当する．T_i の先頭から順に，手順 (1) と同様に k 単語ずつに区切り，索引を活用して T_n における位置 V を取得する．そして，各位置 $j \in V$ に対して，前の単語と連続ならばシーケンスの連続数 $size$ を更新し，不連続ならば新たに $size = k$ のシーケンスを作成しシーケンスリスト S に追加する．

最後に，貪欲法 [18] でシーケンスの割当てを行い，T_p と T_n 間で単語を対応付ける．これは，図 5 の 27–40 行目に該当する．まず，S 中のシーケンスは，$size$ の降順にソートしておく．そして，S から最大のシーケンスを取り出し，シーケンスに含まれる単語の割当てを行う．シーケンスの取り出しおよび割当てを，S が空になるか割り当てられる単語が存在しなくなるまで繰り返す．このとき，すでに割り当てたシーケンスと新規割当てのシーケンスが重複した場合，重複を避けて新規シーケンスを縮小する．その後，$k \leq size$ であれば S の適切な位置に追加し，そうでなければ棄却する．

上記の手法を用いることにより，T_p と T_n 間における単語の対応付け結果 M を得ることができる．T_p における対応漏れの単語は編集によって削除された単語であり，T_n における対応漏れの単語は編集によって追加された単語である．しかし，単語の起源を正確に紐付けるためには，追加された単語が復元であるかどうかを検証する必要がある．

3.3.3　OFF 単語を利用した復元の検証

復元の検証方法について解説する．提案手法では，復元を 2 種類の状況にわけてとらえ，それぞれ異なる方法で検証している．まず 1 つ目は，削除された記述が一意性のあるまとまった単位で再び追加された場合である．この場合は，削除された連続単語群を保持しておくことにより，容易に復元の検出が可能である．しかし前述のとおり，この方法では単語単位のように一意性のない小さな粒度の記述に対して復元を検出できない．そこで，2 つ目は削除された記述が削除前の位置と同じ位置に追加されたときに，これを復元として検出する．位置の判定を行うために，3.3.1 項で説明した OFF 単語の概念を導入する．つまり，削除された単語は OFF 単語としてそのままの位置に保持しておき，後の版において単語が追加されたときに，その位置にある OFF 単語と照合し，同じ単語の場合に復元と判定する．

復元の検証アルゴリズムを**図 6** に示す．編集前の版における起源管理データ D_p，編集後の版における単語群 T_n，T_p と T_n 間での対応付け結果 M，および閾値 l，r を与えたとき，編集前の時点における起源管理データ D_p と T_n 間での対応付け結果 M' を返す．手法の流れは以下のとおりである．

(1) **シーケンスが一致する復元の検証**
  (a) **前処理**：D_p 中の ON 単語に対し，M を利用して T_n 中の単語と対応付ける．未対応の場合は OFF にする．
  (b) **追加されたシーケンスの取得**：M を利用し，編集後の版で未対応の単語シーケンスを取得する．
  (c) **索引の構築**：D_p 中の OFF 単語に対して l 単語索

Input: provenance data D_p in a previous revision, terms $T_n = \{t_0, t_1, \ldots, t_{m-1}\}$ in a next revision, matches M, two thresholds l and r $(r < l)$
Output: matches $M^{'}$ (between D_p to T_n)

```
 1: create an empty matches M'
 2: /* シーケンスが一致する復元の検証 */
 3: j ← 0
 4: for i ← 0 to |D_p| − 1 do
 5:    if d_i.state then
 6:       if M.containsKey(j) then
 7:          add ⟨i, M.getValue(j)⟩ to M'
 8:       else
 9:          d_i.state ← false
10:       end if
11:       j ← j + 1
12:    end if
13: end for
14: S_a ← getAddedSequences(|T_n|, M, l)
15: create an empty l-terms index I
16: for i ← 0 to |D_p| − l do
17:    if ¬d_i.state ∧ ¬d_{i+1}.state ∧ ... ∧ ¬d_{i+l−1}.state then
18:       key ← l-terms sequence
          d_i.term, d_{i+1}.term, ..., d_{i+l−1}.term'
19:       add ⟨key, i⟩ to I
20:    end if
21: end for
22: S ← findSequences(S_a, I, l)
23: M' ← assignSequences(S, M', l, D_p) // assigned d.state ← true
24: /* 位置が一致する復元の検証 */
25: S'_a ← getAddedSequences(|T_n|, M', r)
26: for s'_a ∈ S'_a do
27:    y ← (start position of s'_a in T_n) − 1
28:    if y < 0 then
29:       x ← 0
30:    else
31:       x ← 1 + M'.getKey(y)
32:    end if
33:    create an empty off-terms sequence s_f
34:    while x < |D_p| ∧ ¬d_x.state do
35:       add d_x.term to s_f
36:       x ← x + 1
37:    end while
38:    if r ≤ |s_f| then
39:       M' ← noCrossMapping(s_f, s'_a, r, M')
40:    end if
41: end for
42: return M'
```

図 6　復元検証アルゴリズム
Fig. 6　Restoration detection algorithm.

引を作成する．

(d) **共通シーケンスの洗い出し**：索引を活用し，共通する連続単語を洗い出す．

(e) **共通シーケンスの割当て**：シーケンスの大きさ順に単語を割り当てる．割り当てられた単語は ON にしておく．

(2) **位置が一致する復元の検証**

(a) **追加されたシーケンスの取得**：$M^{'}$ を利用し，編集後の版で未対応の単語シーケンス $S^{'}_a$ を取得する．

(b) **対応位置の取得および対応付け**：$s^{'}_a \in S^{'}_a$ の D_p における位置を取得し，その位置の OFF 単語シーケンスと対応付けを行う．

まず，図 6 の 3–23 行目に該当する，シーケンスが一致する復元の検証について解説する．ここでの問題は，D_p における連続 OFF 単語および，T_n における未対応の連続単語の間で共通するシーケンスの検出と，シーケンスの対応付けである．前処理として D_p 中の未対応の ON 単語を OFF に変更する理由は，3.3.2 項の Mapping 処理において閾値 k 未満の連続数だったシーケンスに，周囲の OFF 単語が加わることにより対応付けられる，もしくは，後に行う位置が一致する復元の検証において対応付けられる可能性があるためである．また，手順 (1) の (e) において対応付けられた D_p 中の単語は，*state* を OFF から ON にしておく．これは，後述する OFF 単語の挿入において，復元済みの OFF 単語を重複して挿入しないための処理である．

シーケンスが一致する復元の検証では，復元と認める最小シーケンスサイズの閾値 l を設けることにより，復元検出時の粒度の問題を回避している．そのため，D_p における OFF 単語シーケンスや T_n における未対応のシーケンスを探索する際には，l 以上の連続単語数を持つものに限定できる．

次に，図 6 の 25–41 行目に該当する，位置が一致する復元の検証について解説する．本手法は，OFF 単語を利用したヒューリスティックかつシンプルな手法である．T_n における未対応の単語シーケンス $s^{'}_a \in S^{'}_a$ に対して，はじめに $s^{'}_a$ の直前の単語に対応する D_p における単語の直後の位置 x を取得する．すなわち，シーケンス直前の単語を基点にして位置を参照する．そして，D_p に $d_x, d_{x+1}, \ldots$ からなる OFF 単語のシーケンス s_f が存在するとき，$s^{'}_a$ と s_f 間で単語の対応付けを行う．このときの対応付けは図 5 とは異なり，交差移動を許さない対応付けを行う．最小シーケンズサイズの閾値 r は 1 や 2 のように小さな値を想定しており，交差移動を許したときにシーケンスの一意性を欠く可能性が比較的高いためである．

上記の手法によって，厳密な復元の検出に基づいた D_p と T_n 間での対応付け結果 $M^{'}$ を取得できる．

3.3.4　起源管理データの更新

最後に，これまでに得られた対応付けを利用し，編集後の版における単語に起源を紐付けて起源管理データの更新を行う．起源管理データの更新アルゴリズムを**図 7** に示す．編集前の時点における起源管理データ D_p，編集後の版における単語群 T_n，D_p と T_n 間での対応付け結果 $M^{'}$，編集後の版番号 v，および閾値 h を与えたとき，編集後の起源管理データ D_n を返す．ここでは，以下の手順に従って新しい起源管理データ D_n を生成する．

Input: provenance data D_p at a previous revision, a terms set T_n in a next revision, matches $M^{'}$, a revision-number v, threshold h
Output: provenance data D_n of the next revision
1: create an empty list D_n
2: /* 起源の紐付けと付与 */
3: **for** $i \leftarrow 0$ **to** $|T_n| - 1$ **do**
4: create an empty object d
5: $d.term \leftarrow t_i \in T_n$
6: $d.state \leftarrow true$
7: $d.removed \leftarrow 0$
8: **if** $M^{'}.containsValue(i)$ **then**
9: $j \leftarrow M^{'}.getKey(i)$
10: $d.provenance \leftarrow d_j.provenance \in D_p$
11: $d_j.state \leftarrow true$
12: **else**
13: $d.provenance \leftarrow v$
14: **end if**
15: add d to D_n
16: **end for**
17: /* OFF 単語の挿入 */
18: $S_f \leftarrow getOffSequences(D_p)$
19: **for** $s_f \in S_f$ **do**
20: $x \leftarrow s_f.start$
21: **if** $0 < x$ **then**
22: $y \leftarrow M^{'}.getValue(x-1) + 1$
23: **else**
24: $y \leftarrow 0$
25: **end if**
26: **for** d **in** $d_x, d_{x+1}, \ldots, d_{x+s_f.size-1} \in D_p$ **do**
27: **if** $d.removed = 0$ **then**
28: $d.removed \leftarrow v$
29: **end if**
30: **if** $v - d.removed < h$ **then**
31: insert d at the position y in D_n
32: $y \leftarrow y + 1$
33: **end if**
34: **end for**
35: **end for**
36: **return** D_n

図 7　起源更新アルゴリズム
Fig. 7　Provenance update algorithm.

(1) **起源の紐付けと付与**：T_n から D_n を生成し，$M^{'}$ を用いて起源を紐付ける．対応付けが存在しない場合は，新たな起源を付与する．
(2) **OFF 単語の挿入**：D_p における OFF 単語のうち，削除されてから h 版未満の単語を D_n の適切な位置に挿入する．

図 7 の 3–16 行目に該当する起源の紐付けと付与のステップにおいては，D_p から起源の引き継ぎを行うが，このとき引き継がれた単語 $d \in D_p$ に対しては，$d.state \leftarrow true$ としておく．すなわち，復元された OFF 単語を ON 単語に変更する．これは，復元された OFF 単語を次のステップで D_n に挿入させないためである．

図 7 の 18–35 行目に該当する OFF 単語の挿入では，位置が一致する復元の検出時と同様にシーケンス直前の単語を基準にして D_n における適切な位置を参照し，挿入する．このとき，閾値である過去 h 版の条件を満たさない OFF 単語は不要になるため破棄する．また，現在の版において削除された単語であれば，$removed$ の値を現在の版番号に変更した後に挿入を行う．最後に，新しく生成された D_n を返して起源管理データの更新を終了する．

このように，全版を順に処理しながら単語の起源を追跡することによって，最終版における各単語の起源を明らかにする．提案手法に関する説明は以上である．次に，評価実験について述べる．

4. 評価実験

本研究における起源追跡手法の有効性を確認するために，評価実験を行う．本章では，評価実験の手順，結果，考察についてまとめる．

4.1 実験の概要とデータセット

本研究の目的は，版管理された文書形式の共同執筆コンテンツにおいて，記述のより正確な起源を単語単位で特定することである．したがって，実験では提案手法を実装したシステムが，文書中の単語の起源を正確に求められているかどうかを検証する．

評価実験には，2014 年 6 月 24 日時点での Wikipedia 日本語版のデータ[*2]を利用する．Wikipedia では不特定多数の編集者が共同で記事の執筆に携わっている．Wikipedia における記事の記述はその版ごとに，記事の全記述および編集された日時や編集者が記録され，編集履歴データとして定期的に Web 上に公開されている．Wikipedia は，不特定多数のユーザが共同執筆を行うコンテンツである点と，編集履歴が版管理されたデータである点から，本研究の評価実験対象として適切であると考えられる．

評価に用いる起源の正解データは人手によって作成し，本システムの推定した起源の正解率を算出する．また，提案手法の有用性を確認するために，2 章において紹介した，既存手法である WikiWho に関しても同様の実験を行い，起源の正解率を比較する．

4.2 実験の手順

実験の手順は以下のとおりである．
(1) Wikipedia の編集履歴データから実験対象をランダムに選択する．
(2) 人手によって実験対象に対して起源の正解ラベルを与える．
(3) 提案手法と WikiWho それぞれによって推定した起源と正解データを照合し，正解率を算出する．

手順 (1)–(3) のそれぞれについて，4.2.1 項，4.2.2 項，

[*2] http://dumps.wikimedia.org/jawiki/20140624/

4.2.3 項で説明する．

4.2.1　実験対象の選択

評価実験にあたって，実験対象を選択する必要がある．Wikipedia では，コンテンツとして記事が存在する．したがって，まず記事をランダムに選択する．ただし，実験においては，このときの条件として 30 版以上の編集履歴を持つ記事に限定した．これは，本手法が版数の多く複雑な編集履歴を持つコンテンツに対して，正確な起源を推定できるかどうかを確かめるためである．また，実験対象に対して正解ラベルを人手で付与するため，ラベル作成者の負担を減らす目的で，選択する記事を文量が過度に多すぎない記事に限定した．具体的には，最終版の単語数が 10,000 語以内の記事である．

記事選択の後，最終版から実験対象とする単語をランダムに 1 つ選択する．このとき，人手による評価を実現するためには，評価に多大な労力や時間を要するような実験対象の選択を回避する必要がある．対象単語と同種の単語が編集された版数や各版における出現頻度が多いと，起源特定のために確認しなければならない版数や箇所が膨大な数になり，評価実験の遂行が困難になってしまうからである．したがって，ランダムに選択した単語が以下の条件を満たすまで，単語の選択を繰り返す．

- 編集履歴において対象単語と同種の単語が追加または復元された版数が 2 件以上，10 版以下である．
- 最終版における対象単語と同種の単語の出現頻度が 10 回以下である．

上記のように対象記事の選択と対象単語の選択を繰り返し，実験対象となるデータセットを作成する．

4.2.2　正解データの作成

4.2.1 項において選択した実験対象に対して，人手により起源の正解データを作成する．既存研究 [7] では，評価者に対して対象記事の最終版および対象単語と，システムが単語の起源と判定した編集を表示し，起源の正誤を人手によって判定するという方法が用いられている．しかし，我々はその方法では正確な評価を行うには不十分であると考える．なぜならば，システムによって単語の起源と推定された版は実際には復元された版で，真の起源はそれ以前の版であるケースが想定されうるからである．このとき，評価者は記事の最終版およびシステムにより起源と推定された版だけを見て，単語の復元が存在するかどうかを判定することはできない．復元を考慮して正確に実験の評価を行うためには，評価者に対して編集履歴の全版を掲示し，その中から対象単語の起源となる版を選択してもらう必要がある．ところが，対象記事の全版から起源を選択するという方法は，評価者に大きな負担がかかるため現実的でない．そこで本実験では，対象単語と同種の単語が編集された版をすべて掲示し，その中から対象単語の起源を評価者に選択してもらうという方法を用いる．

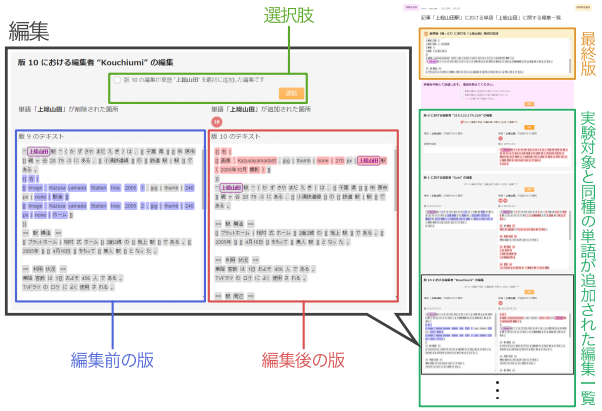

図 8　評価実験の画面

Fig. 8　A screenshot of a web page for evaluation experiment.

我々は評価実験のために，Web ページを作成した．評価者はこの Web ページを利用して実験対象単語の起源となる版を選択する．

Web ページ上での評価の流れは以下のとおりである．まず，評価者は実験用のアカウントを作成し，自分専用のページから評価対象を 1 つ選択する．このとき，評価対象は 4 つ表示され，評価者はその中から任意の対象を選択することができる．

評価対象の選択後，**図 8** のようなページへ遷移する．ここでは，選択した評価対象記事の編集履歴のうち，対象単語と同種の単語が追加もしくは削除された編集を古い順にすべて表示している．また，最終版における対象単語周辺の記述を，ページのどの位置からでも確認できるようになっている．評価者は最終版の記述の状態を確認しながら，対象となる単語が最初に記事に追加された編集を探し，その版を単語の起源であるとして選択する．

各編集は，編集前後の記事の全記述と差分，単語の対応付けによって表される．枠で囲われた各編集の左半分は編集前の記事の記述，右半分は編集後の記事の記述である．それぞれの記述は単語ごとに区切られており，編集によって削除された単語は青色で，追加された単語は赤色で表されている．対象単語と同種の単語は強調して表示される．それらの単語が編集されていた場合，評価者はその単語の位置を容易に参照できるようになっている．評価者は編集を古い順にたどり，対象単語の起源を発見した時点で評価結果を送信できるようになっている．もし編集一覧から起源を発見できなかった場合や，ページ上に表示されていない版が起源であると判断した場合は，そのように報告して評価を終了させることができる．結果の送信を行うと，ページは次の評価対象を選択する画面に遷移する．このように，評価者は評価対象の選択と起源の選択を繰り返すことによって正解データを生成する．

評価者が誤った起源を付与する可能性を考慮し，1 件の評価対象に対して 2 人以上かつ，過半数の評価者が同じ版

表 2　実験対象に関する統計データ
Table 2　Statistical data of experiment targets.

1 記事あたりの平均版数	184
最終版における平均の単語数	3,789
最終版における平均の文字数	8,025

を起源と判定したときに，正解としてその起源をラベル付けする．正解を付与された評価対象はそれ以降，評価者の評価対象を選択する際の一覧には表示されない．また，自分が一度選択した評価対象は一覧には表示されない．

本実験では，10 人のボランティアに評価者として参加してもらった．その結果，正解の起源が付与された 186 件のデータセットを生成することができた．正解データを作成した対象記事の版数および最終版における総単語数，総文字数の平均値を**表 2** に示す．

4.2.3　正解率の算出

人手によって特定した正解の起源と，提案手法および既存手法である WikiWho により推定した起源を照合し，186 件のデータセットに対する手法それぞれの正解率 $Accuracy = \frac{a}{186}$ を算出する．a はデータセット中の，人手による起源と手法による起源が一致した数である．提案手法は，プログラミング言語 Java を用いて実装した[*3]．提案手法の起源推定手順は，3 章において述べたとおりである．

WikiWho は 2 章において解説したように，編集前後で記述の対応付けをする際に，段落，文，単語の順に粒度の大きい記述から照合していく．本実験では段落の区切りを 2 連続以上の改行 “\n\n”，文の区切りを改行 “\n” もしくは句点 “. ” とする．WikiWho のソースコードは Web 上に公開[*4]されており，本実験ではこれを使用する．WikiWho で差分をとるために利用する diff には，python の difflib[*5] を用いる．

記述の単語群への分割には，提案手法，WikiWho ともに形態素解析器 lucene-gosen[*6] を利用する．なお，形態素解析器には，ユーザ辞書として Wikipedia の記事名一覧を追加しておく．

4.3　実験結果と考察

評価実験の結果および考察について述べる．実験に用いた計算環境は**表 3** に示すとおりである．作成した正解データおよび提案手法の起源解析結果については Web 上に公開[*7]している．

4.3.1　正解率と実行時間

正解率に関しては**表 4** のとおり，提案手法の正解率が 86.0%であり，WikiWho の正解率を 24.7 ポイント上回る結果であった．なお，表は提案手法実行時の閾値 k を 3，l を 10，r を 1，h を 20 としたときの結果である．提案手法の実行速度に関しては，実験対象のデータに対して 1 記事あたり 5.482 秒であった．ただし，実行時間には各版の記述を形態素解析器によって単語に分割する手順を含む．また，版ごとに単語への分割を，版のペアごとに 3.3.2 項で述べた単語の対応付けの部分を，並列化して実行している．十分な計算環境下であれば Wikipedia の全記事に対する解析や，コンテンツが編集されるたびごとのリアルタイム解析も実現可能な速度である．

[*3] https://github.com/akilynx/ToP4CAS
[*4] http://people.aifb.kit.edu/ffl/wikiwho/
[*5] https://docs.python.org/2.7/library/difflib.html
[*6] https://code.google.com/p/lucene-gosen/
[*7] http://wikidiff.db.ss.is.nagoya-u.ac.jp/list

表 3　計算環境
Table 3　Computing environment.

OS	Ubuntu 14.04.1 LTS
CPU	Intel(R) Xeon(R) CPU E5620 @ 2.40GHz
CPU コア数	16
メモリ	32 GB

表 4　正解率と平均実行時間
Table 4　Accuracy and run time.

	WikiWho	提案手法
正解数/全体数	114/186	160/186
正解率	0.613	0.860
1 記事あたりの平均実行時間［秒］	6.612	5.482

提案手法の起源推定精度における優位性について考察する．提案手法は，OFF 単語の概念を導入し，単語の位置をもとに小さな粒度の復元を厳密に検出する．単語単位での厳密な復元の検出が，より正確な起源の紐付けを可能にし，起源推定の精度向上につながったと考えられる．また，シーケンスを重視して 2 版間の単語を対応付けることによって，章の再編成時などに生じる記述の交差移動を許容する対応付けが行われ，交差移動にともなう起源の欠損を抑制する効果をもたらしていると推測できる．

4.3.2　閾値と正解率および実行時間の関係

提案手法においては，復元検出時の記述の粒度や，遡る版数の範囲は，閾値によって決定される．すなわち，閾値を変動させることによって，閾値を用いた各手順が起源推定の精度に与える影響を計ることができる．それぞれの閾値を変動させたときの正解率および実行時間の変化を**図 9** に示す．提案手法によって付与された起源の正解率を棒グラフで，1 記事あたりの平均実行時間を折れ線グラフで示す．

まず，正解率に関して述べる．正解率は，k が 3，l が 9–11，r が 1，h が 20 または 30 のとき，最も高い値を示す．図中では，h が 0 のとき，最も低い正解率の 20.4%となる．閾値 h は，単語が削除されてから復元の検証対象として OFF 単語化し，起源管理データ中に残しておく版数を表す．つまり，h が 0 のとき，復元の検証がいっさい行われない．h が 20 のときと比較して，0 のときは正解率が

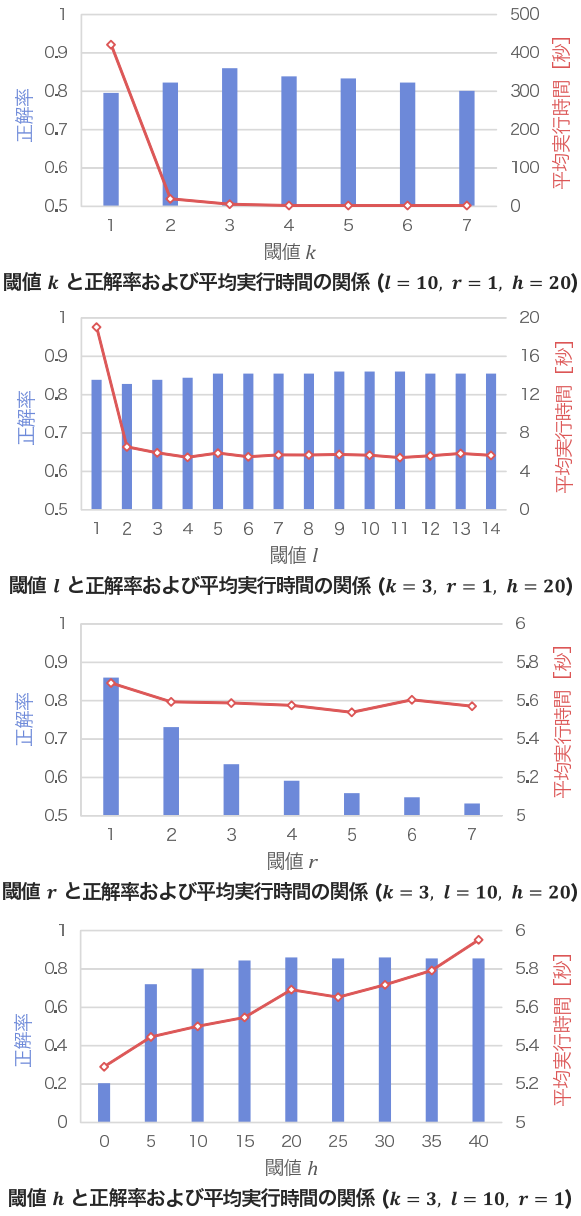

図 9　閾値と正解率および平均実行時間の関係

Fig. 9　Influence of thresholds on accuracy and running velocity.

65.6 ポイントも低下していることから，起源の正確な特定のためには復元の検証が重要であることが分かる．

正解率に関して最も注目すべきは，閾値 r を 1 から増加させたときに，正解率が反比例して低下するという点である．閾値 r は OFF 単語の位置を利用して復元を検出する際に，復元と認める最小連続単語数である．r を増加させたときに正解率が低下するということは，位置が一致する復元の検出時に，粒度の小さな記述の復元を検出できずに起源推定の精度が低下しているという状況を示している．すなわち裏を返せば，単語単位のように粒度の小さい記述の復元を，OFF 単語の位置を利用して検出することによって，起源の推定精度が向上するということである．この事実は我々の仮定を実証し，提案手法の有効性を裏付ける結果である．単語単位での復元は，既存研究の WikiWho では検出できないため，本手法との精度の差を生じた要因であると考えられる．

次に，実行時間について述べる．提案手法において時間的コストを要する点は，形態素解析を除けば主に 2 カ所である．

1 つはシーケンスを重視した単語の対応付けである．本手法では，k-連続単語の索引を事前に構築することにより，シーケンスの洗い出しにかかる時間を短縮している．編集前の版における総単語数を m，編集後の版における総単語数を n，索引参照先数の平均を c とするとき，索引の構築からシーケンスの洗い出しまでにかかる時間的コストは $O(cm+n)$ である．k が十分に大きいとき，c は 1 に近づく．しかし，k が小さいときは c が大きくなり，図のように実行速度が大幅に低下する．

時間的コストを要するもう 1 つの箇所は，復元の検証である．閾値 l を利用したシーケンス一致判定を行う復元の検証では，上述のシーケンスを重視した単語の対応付けと同様に，索引の構築が実行時間短縮の鍵となっている．したがって，l が 1 のように小さな値をとる場合には実行速度が低下する．しかし，逆に l が大きすぎる場合は，同様に実行速度が低下する．過度に大きい値の l によってシーケンス一致判定による復元の検証が機能しなくなると，その後に行う位置の一致判定による復元の検証に負担がかかる．つまり，シーケンスの一致判定によって対応付けられなかった復元を，システムは位置の一致判定によって検出する．通常ならば前者の処理の方が速いため，前者の復元検出に漏れがあると後者の処理に時間がかかり，正解率としてはさほど変わらないが，システム全体としての速度が低下する．たとえば，l を 100,000 としたとき，正解率は 83.3%，1 記事あたりの平均実行時間は 12.90 秒である．l が 10 のときと比較すると，正解率は 2.7 ポイントだけ低下するが，実行時間は 2.35 倍となる．

表 5　誤りの原因とその数

Table 5　Causes of errors.

誤りの原因	対象数
復元検出における誤り	7
解釈の違いによる誤り	7
閾値による誤り	5
単語分割における誤り	4
対応付けにおける誤り	3

4.3.3　誤りの原因についての考察

実験結果をもとに，提案手法の誤りとなった原因を調査した．誤りの原因は大きく 5 種類にわけることができる．不正解となった 26 件の起源を原因によって分類した結果を**表 5** に示す．最も多かった誤りの原因は，復元検出時の誤りと解釈の違いによる誤りで，どちらも 7 件が該当する．復元検出における誤りは，主に基準位置のずれによっ

て位置の一致する復元を検出できなかったことに起因する．OFF 単語を用いた方法では，削除された単語の位置を完全に保管しておくことはできない．特に，OFF 単語シーケンスの直前の単語を基準としているために，基準の位置に別の単語が挿入された場合や，基準単語が移動させられた場合に，OFF 単語の位置がずれてしまう可能性がある．OFF 単語の位置がずれれば，復元を正しく検出できなくなり誤りを生じる．

システムと評価者の起源に対する解釈の違いによって生じた誤りについて述べる．この誤りは主に，実験対象に起源の正解ラベルを付与する際に，評価者が 1 対多対応に単語を紐付けたことによる．Wikipedia では，編集前の版における記述の一部が，編集後の版で 2 カ所に複製されることがある．このとき，評価者は起源が分岐していると解釈し，両方に編集前の記述の起源を紐付ける．ところが提案手法は，起源の分岐を想定していないため，片方に前の版と同一の起源を紐付け，もう片方には新しい起源を付与する．この解釈の違いが誤りの原因となった．本課題は，システムに起源の分岐時の処理を組み込むことにより，解決することができる．

次に多かった誤りの原因は，閾値である．ただし，閾値の問題は単純に解決できない．図 9 を参照すれば分かるように，閾値の変動が新たな誤りを生じる．すべての対象に対応可能な最適の閾値を決定することは困難である．この問題の 1 つの解決案として，動的に閾値を設定するという方法があげられる．提案手法の閾値 l に着目したとき，l は一意性を持つ最小のシーケンスサイズであればよい．したがって，単語の共起確率などを用いてシーケンスごと動的に閾値を算出すれば，より高精度に復元を検出できると考えられる．

そのほかには，形態素解析における記述の過度な分割や，周辺の記述の違いによって同一の記述でも異なる分割結果が得られるなどといった問題が確認された．また，単語の対応付けにおいて，文脈的には対応付けすべき単語でも，周辺の単語が大きく入れ替わっているために，システムにとっては対応付けが困難な状況が存在した．

5. おわりに

本論文では，文書形式の版管理された共同執筆コンテンツにおける，記述の起源を単語単位で追跡する手法を提案した．記述の起源を正確に追跡するためには記述の復元を厳密に検出する必要がある．提案手法では，OFF 単語を用いた起源管理の方式を導入することにより，単語の位置に基づいた復元の判定を行っている．これにより，単語単位のような小さな粒度の記述に対して復元を検出できるようになり，起源特定の精度向上につながる．

評価実験は，Wikipedia のデータを対象に行った．ランダムに選んだ 186 件の対象に人手で起源の正解ラベルを付与し，提案手法を実装したシステムの起源推定結果と照合した．提案手法の正解率は 86.0%であり，既存手法に対する優位性を確認した．

図 10　応用アプリケーションの例（記事「菓子」）

Fig. 10　An example of applications using our proposed method (article: Confectionery).

本手法を導入することにより，共同執筆コンテンツの編集や閲覧を支援できる．例として，Wikipedia の編集履歴データに対して本手法を適用して実装した応用アプリケーション例[*8]を**図 10** に示す．図は記事「菓子」の 2014 年 6 月 24 日の時点での記述である．提案手法を拡張した本システム[*9]は単語ごとに，起源である追加された版と同時に，削除された版や，復元された版およびそれらの編集者を特定することができる．図ではカーソルを合わせた単語「強調」に関して，起源などの情報が黒い吹き出しで表示され，同時に「強調」と同じ起源を持つ単語が赤枠で示されている．これらの情報を参照することにより，編集者は編集の効率化を図ることができる．また，コンテンツの閲覧者は起源を記述の信憑性判定に利用することができる．さらに本システムを利用すれば，記述の起源だけでなく，編集者間の削除や復元，残存の関係を抽出することができ，編集者間の関係を利用した応用アプリケーションや研究にとって有用である．

今後の課題として，復元検出精度の向上と，動的な閾値の決定があげられる．復元検出では，前述のとおり，OFF 単語の基準点がずれることによって，誤りが生じることが分かっている．基準点のずれを解決する案として，OFF 単語シーケンスの直前と直後の両側を基準として，2 つの基準点内の範囲に単語が追加されたときに，復元を検証する方法があげられる．また，動的な閾値は，単語の共起確率を用いて決定できると考えられる．

謝辞　本研究の一部は，JSPS 科研費（25280039, 26540043, 23700113）の助成を受けたものです．ここに記して謝意を表します．

*8　http://wikidiff.db.ss.is.nagoya-u.ac.jp/
*9　https://github.com/akilynx/ToP4CAS

参考文献

[1] 鈴木　優：Wikipedia における情報の質，情報処理学会論文誌：データベース，Vol.6, No.4, pp.46–58 (2013).

[2] Kramer, M., Gregorowicz, A. and Iyer, B.: Wiki Trust Metrics Based on Phrasal Analysis, *Proc. 4th International Symposium on Wikis, WikiSym '08*, pp.24:1–24:10, ACM (2008).

[3] Hu, M., Lim, E.-P., Sun, A., Lauw, H.W. and Vuong, B.-Q.: Measuring Article Quality in Wikipedia: Models and Evaluation, *CIKM '07: Proc. ACM International Conference on Information and Knowledge Management*, pp.243–252 (2007).

[4] Adler, B.T. and de Alfaro, L.: *WWW '07: Proc. 16th International Conference on World Wide Web*, p.261 (2007).

[5] Adler, B.T., Chatterjee, K., de Alfaro, L., Faella, M., Pye, I. and Raman, V.: Assigning Trust to Wikipedia Content, *WikiSym '08: Proc. International Symposium on Wikis* (2008).

[6] Adler, B.T., Pye, I. and Raman, V.: Measuting Author Contributions to the Wikipedia, *WikiSym '08: Proc. International Symposium on Wikis* (2008).

[7] Flöck, F. and Acosta, M.: WikiWho: Precise and Efficient Attribution of Authorship of Revisioned Content, *Proc. 23rd International Conference on World Wide Web*, ACM (2014).

[8] Bergroth, L., Hakonen, H. and Raita, T.: A Survey of Longest Common Subsequence Algorithms, *Proc. 7th International Symposium on String Processing Information Retrieval* (*SPIRE'00*), pp.39–48, IEEE Computer Society (2000).

[9] Brown, P.F., Pietra, V.J.D., Pietra, S.A.D. and Mercer, R.L.: The Mathematics of Statistical Machine Translation: Parameter Estimation, *Comput. Linguist.*, Vol.19, No.2, pp.263–311 (1993).

[10] Kittur, A., Suh, B., Pendleton, B.A. and Chi, E.H.: He Says, She Says: Conflict and Coordination in Wikipedia, *Proc. SIGCHI Conference on Human Factors in Computing Systems, CHI '07*, pp.453–462, ACM (2007).

[11] Halfaker, A., Kittur, A., Kraut, R. and Riedl, J.: A Jury of Your Peers: Quality, Experience and Ownership in Wikipedia, *Proc. 5th International Symposium on Wikis and Open Collaboration, WikiSym '09*, pp.15:1–15:10, ACM (2009).

[12] Flöck, F., Vrandečić, D. and Simperl, E.: Revisiting Reverts: Accurate Revert Detection in Wikipedia, *Proc. 23rd ACM Conference on Hypertext and Social Media, HT '12*, pp.3–12, ACM (2012).

[13] de Alfaro, L. and Shavlovsky, M.: Attributing Authorship of Revisioned Content, *Proc. 22nd International Conference on World Wide Web, WWW '13*, pp.343–354, International World Wide Web Conferences Steering Committee (2013).

[14] Flöck, F. and Rodchenko, A.: Whose article is it anyway? Detecting authorship distribution in Wikipedia articles over time with WIKIGINI, *Wikipedia Academy* (2012).

[15] Suzuki, Y.: Effects of Implicit Positive Ratings for Quality Assessment of Wikipedia Articles, *Journal of Information Processing*, Vol.21, No.2 (2013).

[16] Brandes, U., Kenis, P., Lerner, J. and van Raaij, D.: Network Analysis of Collaboration Structure in Wikipedia, *WWW '09: Proc. 18th International Conference on World Wide Web*, pp.731–740 (2009).

[17] Nakamura, A., Suzuki, Y. and Ishikawa, Y.: Clustering Editors of Wikipedia by Editor's Biases, *Web Intelligence'13*, pp.351–358 (2013).

[18] Cormen, T.H., Leiserson, C.E., Rivest, R.L. and Stein, C.: *Introduction to Algorithms*, MIT press (1990).

中村 晃

2013 名古屋大学工学部電気電子・情報工学科卒業，2015 名古屋大学大学院情報科学研究科博士前期課程修了．修士（工学）．ソーシャルメディア解析の研究に従事．日本データベース学会会員．

鈴木 優　（正会員）

1999 年神戸大学工学部情報知能工学科卒業，2001 年奈良先端科学技術大学院大学博士前期課程修了．2004 年同博士課程修了．2004 年立命館大学講師．2009 年京都大学特定研究員．2010 年名古屋大学研究員，2011 年同大学特任助教．2014 年奈良先端科学技術大学院大学情報科学研究科特任准教授．博士（工学）（奈良先端科学技術大学院大学）．ソーシャルメディアの解析，情報検索の研究に従事．日本データベース学会，電子情報通信学会，人工知能学会，IEEE-CS，ACM 各会員．

石川 佳治　（正会員）

1989 年筑波大学第三学群情報学類卒業．1994 年筑波大学大学院博士課程工学研究科単位取得退学．同年奈良先端科学技術大学院大学助手．1999 年筑波大学電子・情報工学系講師．2004 年同助教授．2006 年名古屋大学情報連携基盤センター教授．2013 年同大学大学院情報科学研究科教授．博士（工学）（筑波大学）．データベース，データ工学に興味を持つ．日本データベース学会，電子情報通信学会，人工知能学会，ACM，IEEE CS 各会員．

（担当編集委員　乾 孝司）