Vol.2017-MPS-112 No.18
2017/2/28

Linked Data におけるリンク切れ自動修復フレームワークの提案と評価

児玉 英一郎[†1]　常木　翔太[†2]　清水　小太郎[†3]　王 家宏[†1]　高田 豊雄[†1]

概要: 近年, Linked Data に関する研究が盛んに行われている．Linked Data とは, 外部から参照可能な RDF(Resource Description Framework)に従い記述されたデータのことであり，2006 年に Tim Berners-Lee によって提唱された概念である．Linked Data を利用したアプリケーションでは，リンクをたどって情報を提供することが多いため，Linked Data におけるリンクの保持は重要なものとなっている．しかし，Web 上でも発生しているリンク切れが Linked Data においても起こり得る．その多くは Linked Data の削除や移動によって発生し，移動によって起こるリンク切れは，リンク切れを検知し，リンクの再構築を行う必要がある．本研究では，このリンク切れの修復問題を解決するため，Linked Data をバイナリ特徴ベクトルで近似し，データサイズの縮小，実行の高速化を狙ったリンク切れ自動修復フレームワークの提案を行う．また，本提案フレームワークに対し，保有するデータ量削減に関する評価，精度に関する評価，スケーラビリティに関する評価を行ったので，その評価結果についても報告する．

キーワード：Linked Data,　リンク切れ修復

A Framework for Restoring Broken Links of Linked Data and Evaluation

Eiichiro Kodama[†1] Shota Tsuneki[†2] Kotaro Shimizu[†3] Jiahong Wang[†1] Toyoo Takata[†1]

Abstract: In recent years, researches on Linked Data are being actively conducted. Linked Data is a concept suggested by Tim Berners-Lee in 2006, and is such data that is represented in accordance with outside-referable RDF (Resource Description Framework). Since many applications using linked data acquire information by tracing the links, it is very important to maintain them functional all the time. As routinely occurs in Web, broken links also occur for the linked data. Broken links are generally caused by deleting or migrating linked data. For broken links caused by the linked data migration, finding and then restoring them are necessary. This paper addresses the subject of automatically restoring broken linked data, and proposes an effective framework for it. The proposed framework represents linked data as the binary feature vector, so that broken linked data could be restored with largely reduced metadata size and very high restoring speed. The proposed framework is evaluated in terms of level of reduced metadata size, accuracy of restored linked data, and scalability of the system that adapted the proposed framework. Evaluation results are reported and discussed.

Keywords: Linked Data, Restore Broken Link

1. はじめに

近年, Linked Data に関する研究が盛んにおこなわれている．Linked Data とは，外部から参照可能な RDF(Resource Discription Framework)に従い記述されたデータのことであり，2006 年に Tim Berners1-Lee によって提唱された概念である[1][2]．その後，海外では盛んに Linked Data に関する研究が進められ，Linked Open Data(LOD)と呼ばれる，すでに公開されている Linked Data を収集，蓄積するプロジェクトが成功を収めている．Linked Data はトリプル[3]と呼ばれるものから構成されており，Subject, Predicate, Object の 3 要素からなる．

Linked Data を利用したアプリケーションでは，リンクをたどって情報を提供することが多いため，Linked Data におけるリンクの保持は重要なものとなっている．しかし，Web でも見られるリンク切れが Linked Data においても起こり得る．その多くは Linked Data の削除や移動によって発生するが，移動によって起こるリンク切れは，リンク切れを検知し，リンクを再構築する必要がある．リンク切れが発生する原因としては，企業の合併や統合による名称の変更，婚姻による苗字の変更，DB のメンテナンスによる表記の統一などの理由により, URI が変更されることが考えられる．そのような状況では，リンク切れが生じ，関連する情報の取得が行えない．その結果，その Linked Data にアクセスするアプリケーションは正常に動作せず，不具合が起こり得る．そこで，このリンク切れを検知し，リンクの再構築を行う必要がある．

2. 関連研究

Linked Data におけるリンク修復の研究としては内容解析によるリンク切れ先推定手法[4][5]と，リンク構造解析によるリンク切れ先推定手法[6]が知られている．

Linked Data の内容から，リンクの移動先を推定する内容解析手法としては，DSnotify[4][5]で用いられている手法が知られている．DSNotify は，Linked Data の移動を検知し，リンク切れが発生した Linked Data のリンクの修復を

†1 岩手県立大学ソフトウェア情報学部
Faculty of Software and Information Science, Iwate Prefrctural University
†2 岩手県立大学大学院ソフトウェア情報学研究科
Graduate School of Software and Information Science,
Iwate Prefectural University
†3 株式会社インターネットイニシアティブ
Internet Initiative Japan Inc.

ⓒ 2017 Information Processing Society of Japan

行うシステムである．DSNotify では Linked Data に対し，英数字からなる特徴ベクトルを生成後，この特徴ベクトル間の類似度を$Levenshtein$距離で算出し，リンクの修復を行うものである．

また，リンク構造を利用して，リンクの移動先を推定する手法としては，リンク構造解析による Linked Data のリンク切れ先同定手法 [6] が知られている．この手法は，事前に RDF ストアのクローリングを行い，リンク構造を予め取得する．Linked Data のリンク切れを検知した場合には，事前に保有している各 Linked Data のリンク先集合と，新規に作成された Linked Data のリンク先集合との類似度を，*Jaccard* 係数を用いて計算する．その中から，最も類似度が高い Linked Data をリンク切れが発生した Linked Data のリンク移動先と判断し，リンク修復を行うものである．

3. 関連研究の問題点

DSnotify の問題点としては，英数字の特徴ベクトルを利用しリンク切れ修復を行うため，リンク切れ修復に必要なデータ量が膨大になってしまうという点があげられる．

例として，2011 年時点の LOD Cloud[7]全体の 310 億トリプルに対してリンク切れ監視を行った場合を考える．

DSNotify が保有しなければならないデータ量を概算すると，以下のようになる．まず，1 トリプルあたりの Object のデータサイズDSは式(1) にて近似できる．

$$\mathrm{DS} = \text{Object の URI 長の平均値} \times 1 \text{ 文字あたりのバイト数} \quad (1)$$

ここで，Object の URI 長の平均値を 26 と仮定すると，1 トリプルあたり$26 \times 2 = 52\mathrm{byte}$，LOD Cloud 全体としては，$52\mathrm{byte} \times 3.1 \times 10^{10} = 1.61TB$となり，約1.61TBのデータ量が必要となる．比較計算量としては，式(2)に示す計算量が必要となる．

$$Levenshtein\text{距離}(u_1, u_2)\text{の計算に要する計算量} \times 3.1 \times 10^{10}, \quad (2)$$

u_1:リンク切れを起こした URI,

u_2:Object の URI

リンク構造解析による Linked Data のリンク切れ同定手法の問題点としては，類似度計算のために，Linked Data のリンク構造全体を保持しなければならないという点があげられる．LOD Cloud 全体にこの手法を適用した場合，保有しなければならないデータ量は，概算で，DSNotify と同様に1.61TBとなる．また，比較計算量としては，式(3)に示す計算量が必要となる．

$$Jaccard\text{係数}(L_1, L_2)\text{の計算に要する計算量} \times 3.1 \times 10^{10}, \quad (3)$$

L_1:リンク切れを起こした URI のリンク集合,

L_2:リンク移動候補 Linked Data のリンク集合

しかし，表 1 に示すように[7]，Linked Data は，2007 年時点でトリプル数が 5 億件であったものが，2011 年には 310 億件と急増しており，今後，Linked Data が普及しデータ量が増加した場合，これらの手法は現在は有効であっても，将来的には有用性が損なわれると考えられる．

表 1　Linked Data のデータ推移

年度	Dataset 数	トリプル数
2007	12	500 百件
2008	45	2000 百件
2009	95	6700 百件
2010	203	26,900 百件
2011	295	31,000 百件
2014	1,014	——

4. Linked Data におけるリンク切れ自動修復フレームワークの提案

4.1 Linked Data におけるリンク切れ自動修復フレームワークのモデル

本研究では，リンク切れ修復に必要なデータ量を削減し，効率的に Linked Data の修復が行える Linked Data 修復フレームワークの提案を行う．本提案のフレームワークのモデルを図 1 に示す．

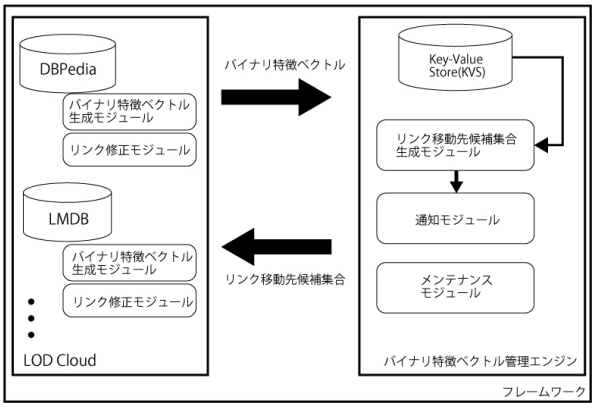

図 1　Linked Data におけるリンク切れ自動修復フレームワークのモデル

LOD Cloud 内のサービス提供サイトには，バイナリ特徴ベクトル生成モジュールとリンク修正モジュールが配置され，バイナリ特徴ベクトル管理エンジン側は，Key-Value Store，リンク移動先候補集合生成モジュール，通知モジュール，メンテナンスモジュールから構成される．以下，本フレームワークの動作について説明する．

- **バイナリ特徴ベクトル生成時**

バイナリ特徴ベクトル生成モジュールは，LOD Cloud 内のサービス提供サイトの RDF ストアから Linked Data を取得し，後述するアルゴリズムによりバイナリ特徴ベクトルの生成を行う．その後，バイナリ特徴ベクトルと URI をバイナリ特徴ベクトル管理エンジンへ送信する．

ⓒ 2017 Information Processing Society of Japan

バイナリ特徴ベクトルと URI を受け取ったバイナリ特徴ベクトル管理エンジンは，それを Key-Value Store へ保存する．

- **リンク切れ検出時**

バイナリ特徴ベクトル生成モジュールは，バイナリ特徴ベクトル生成時にリンク切れを検出した場合，そのリンク切れが発生した URI をバイナリ特徴ベクトル管理エンジン側へ通知する．リンク移動先候補集合生成モジュールは，通知された URI のバイナリ特徴ベクトルを利用し，リンク移動先候補集合生成アルゴリズムに従い，リンク移動先候補集合を作成する．生成されたリンク移動先候補集合は通知モジュールにより，LOD Cloud 側へ通知され，LOD Cloud 側のリンク修正モジュールにおいて，LOD Cloud の管理者の判断により，通知されたリンク移動先候補集合をもとにリンクの修正が行われる．

- **メンテナンス時**

リンク切れ修復が完了し，必要がなくなったバイナリ特徴ベクトルや，一定期間が経過し利用されなくなったバイナリ特徴ベクトルの削除を行う．定期的にメンテナンスをすることにより，類似度計算の際の計算量を削減する．

4.2 バイナリ特徴ベクトル生成アルゴリズムとリンク移動先候補集合生成アルゴリズム

以下，本提案独自のバイナリ特徴ベクトル生成モジュールのアルゴリズムとリンク移動先候補集合生成アルゴリズムの詳細を示す．本アルゴリズムは，バイナリ列の特徴ベクトルを用いることで，計算コストを削減し，リンク移動先候補集合を求めるものである．図 2 に，本アルゴリズムの説明で用いる記号を示す．

X, Y, Y', Z ：Linked Data
$L(X)$ ：Linked Data Xに含まれる URI の集合
h ： 一方向性ハッシュ関数
l ： 正の整数，$L = 2^l$
i ： 正の整数
$LSB(b,i)$ ： バイナリ文字列bの最下位iビット

図 2 アルゴリズムの説明で用いる記号

- **バイナリ特徴ベクトルの生成アルゴリズム**

Linked Data Xが与えられたとき，X のバイナリ特徴ベクトル$char(X)$を，(1)から(2)の手順により，L 次元バイナリベクトル$(b_L, b_{L-1}, \dots, b_2, b_1)$ として求める．但し，$b_i \in \{0,1\}$とする．また，記号⊕は排他的論理和とする．

(1) $(b_L, b_{L-1}, \dots, b_2, b_1) = (0,0,\dots,0,0)$とする
(2) 各 $u \in L(X)$ に対して，$b_{LSB(h(u),l)} \oplus = 1$ によって $b_{LSB(h(u),l)}$を決定する．

その後，生成されたバイナリ特徴ベクトル$char(X)$をキー，Linked Data Xの URI をバリューとして Key-Value Store にストアする．

- **リンク移動先候補集合生成アルゴリズム**

Linked Data X, Yが与えられたとき，XとYのバイナリ特徴ベクトルのハミング距離を式(4)により定義する．但し，xをベクトルとするとき，$support(x)$ はxの中の非零成分の個数とする．

$$hamming(X,Y) = support(char(X) \oplus char(Y)) \quad (4)$$

Linked Data Xと Linked Data Yの間にリンクが張られており，Linked Data Yが Linked Data Y'へ移動した場合，Y'を含む Key-Value Store からリンク移動先候補集合 S を以下の手順で求める．

(1) $0 < \theta < 1$とする
(2) $S = \emptyset$(空集合)とする
(3) Key-Value Store 内に格納されている各 Linked Data Zに対し，$hamming(char(Y), char(Z))$を計算し，$hamming(char(Y), char(Z))/L < \theta$ならば$S = S \cup \{Z\}$とし，リンク移動先候補集合 Sを構築する．

5. アルゴリズムの動作例

本提案アルゴリズムを例を用いて説明する．

Linked Data A が 2 つのリンク URI1 と URI2 を保持しているとする．また$l = 4$, $L = 16$として，$char(A)$は16次元のバイナリ特徴ベクトルとする．

- **バイナリ特徴ベクトル生成時**

LOD Cloud 内のサービス提供サイトの RDF ストアに，Linked Data Aが格納されているとする．バイナリ特徴ベクトル生成モジュールは，RDF ストアから Linked Data Aを取得する．その後，Linked Data Aが保持している URI1にハッシュ関数hを適用し，ビット列を取得する．ここでは例として，$h(\text{URI1}) = \text{"}\dots 0101\ 0001\text{"}$が生成されたと仮定する．$l = 4$であるため，取得したビット列の下位 4 ビットを取り出す．この取り出したビット列$LSB(h(\text{URL 1}), 4) = \text{"0001"}$は 10 進数で1であるため，$char(A)$の$b_1$成分を 0 - 1 反転する．さらに，Linked Data Aが保持している URI2 にハッシュ関数を適応し，ビット列を取り出す．例として，$h(\text{URI 2}) = \text{"}\dots 0110\ 0110\text{"}$というビット列を取得したと仮定する．同様に，$l = 4$であるため，下位4ビットを取り出す．この，ビット列$LSB(h(\text{URL 2}), 4) = \text{"0110"}$は 10 進数で6であるため，$char(A)$の$b_6$成分を 0 - 1 反転する．

以上の手順により，$char(A) = (0, \dots, 0,1,0,0,0,0,1)$となる．最後に，生成したバイナリ特徴ベクトル$char(A)$と，Linked Data Aの URI をペアとしてバイナリ特徴ベクトル管理エンジンへ送信する．これを受け取ったバイナリ特徴ベクトル管理エンジンは，バイナリ特徴ベクトルをキー，Linked Data の URI をバリューとして Key-Value

ⓒ 2017 Information Processing Society of Japan

Store へ保存する．

- **リンク切れ検出時**

Linked Data Aが Linked Data A'へ移動し Linked Data A'のバイナリ特徴ベクトルが計算され，Key-Value Store へ保存されているものとする．バイナリ特徴ベクトル生成モジュールが，RDF ストア監視中に，Linked Data Aのリンク切れを発見した場合には，Linked Data Aの URI をバイナリ特徴ベクトル管理エンジンへ送信する．これを受け取ったバイナリ特徴ベクトル管理エンジンのリンク移動先候補集合生成モジュールは，Key-Value Store に保存されている他の Linked Data のバイナリ特徴ベクトルとの比較を行う．ここでは，その他の Linked Data はA', B, C, Dだけであったとし，それぞれのバイナリ特徴ベクトルは図 3 のようになっていたとする．

$$char(A) = (0,\dots,0,0,1,0,0,0,0,1)$$
$$char(A') = (0,\dots,0,1,1,0,0,0,0,1)$$
$$char(B) = (0,\dots,0,0,1,0,1,1,1,0)$$
$$char(C) = (0,\dots,0,1,0,1,0,1,1,0)$$
$$char(D) = (0,\dots,0,0,1,1,1,0,0,1)$$

図 3　保存されているバイナリ特徴ベクトル

リンク移動先候補集合生成モジュールは，Linked Data Aのバイナリ特徴ベクトルと Key-Value Store のバイナリ特徴ベクトル間で$hamming$距離を計算し，非零成分の個数を計算する．図 3 の場合の計算結果を図 4 に示す．

$$hamming(char(A), char(A')) = 1$$
$$hamming(char(A), char(B)) = 4$$
$$hamming(char(A), char(C)) = 6$$
$$hamming(char(A), char(D)) = 2$$

図 4　Linked Data Aとの$hamming$距離

計算した$hamming$距離をLで除算し，Linked Data Aとの非類似性を計算する．図 5 に計算結果を示す．

$$hamming(char(A), char(A')) / L = 0.0625$$
$$hamming(char(A), char(B)) / L = 0.25$$
$$hamming(char(A), char(C)) / L = 0.375$$
$$hamming(char(A), char(D)) / L = 0.125$$

図 5　Linked Data Aとの非類似性計算結果

図 5 より Linked Data Aと Linked Data A'への$hamming$距離は1であるため，非類似性は0.0625 となる．ここで閾値θは0.2であったと仮定すると Linked Data A', Dが閾値以下となるため，リンク移動先候補集合Sは，$S = \{A', D\}$となる．通知モジュールは，このリンク移動先候補集合Sを LOD Cloud 内の問い合わせのあったサービス提供サイトへ送信し，サービス提供サイトの管理者の判断により，リンクの修正を行う．

- **メンテナンス時**

Linked Data Aが Linked Data A'へ移動し，すべてのリンク切れ修復が完了した場合，Linked Data Aのバイナリ特徴ベクトルは今後利用されなくなるため，メンテナンスモジュールによって Key-Value Store から削除を行う．

6. 評価

6.1 評価環境

本提案フレームワークの評価を行うために，構築実験として本提案フレームワークに従ったプロトタイプの実装を行った．本実装環境を表 2 に示す．

表 2　実験環境

OS	OS X 10.9.4
プロセッサ	Intel Core i5
メモリ	8GB
使用言語	Java 1.8.0_05
RDF ストア	Virtuoso 7.1.0
Key-Value Store	Kyoto Tycoon 0.9.56

本構築実験で作成したプロトタイプを利用して，保有するデータ量の削減に関する評価，精度に関する評価，スケーラビリティに関する評価を行った．

評価用データは，DBpedia[8]のスナップショットを利用した．利用したデータセットを表 3，その内訳を表 4 に示す．

表 3　利用した DBpedia のデータセット

利用したデータセット	データセットの説明
External_links	記事についての外部 Web ページへのリンク
Homepages	人物や団体などのホームページへのリンク
Images	Wikipedia の記事に記載されているメイン画像と対応するサムネイルへのリンク
Interlanguage_links	Wikipedia の異言語間のリンクから抽出された関連するリソースに対するリンク
Labels	カテゴリについてのラベル
Page_links	Wikipedia の記事間の内部リンクをもとに生成された，DBpedia 記事間の内部リンク
Persondata	Wikipedia から抽出された，生まれた場所や日付などに関する情報
Redirects	Wikipedia での記事間のリダイレクト
Short_abstracts	Wikipedia の記事の短い要約

ⓒ 2017 Information Processing Society of Japan

表 4　評価用データ

	DBpedia 3.8	DBpedia 3.9
作成日	2012 年 6 月	2013 年 4 月
Linked Data 数	3,769,926	4,004,478
トリプル数	177,867,270	198,414,516
1Linked Data あたりの平均トリプル数	47.18	49.54
リダイレクト数	2,386,041(59.58%)	

DBpedia 3.8 と DBpedia 3.9 を比較したところ，2,386,041 件(59.58%)の Linked Data でリダイレクトが発生し，URI の変更が生じていた．実際に発生していた Linked Data の URI の移動例を表 5 に示す．

表 5　発生していた URI の変更例(一部抜粋)

移動前	移動後
Johann_Sebastian_Bach/Biography	Johann_Sebastian_Bach
Keplers_laws	Kepler's_laws_of_planetary_motion
Kevin_ONeill	Kevin_O'Neill
King's_Royal_Rifle_Corps	King's_Royal_Rifle_Corps
Law_of_gravitation	Newton's_law_of_universal_gravitation
Laws_of_Motion	Newton's_laws_of_motion
Lords_Supper	Lord's_Supper
MetaEthics/NonCognitivism	NonCognitivism
Monty_python's_Life_of_Brian	Monty_python's_Life_of_Brian
Two_wrongs_make_a_right_(fallacy)	Two_wrongs_make_a_right

また，評価環境に用いたパラメータの値としては，$l = 9$，$L = 512$，$\theta = 0.25$を用いた．

6.2 保有するデータ量削減に関する評価

保有するデータ量削減に関する評価として，DBpedia 3.9 の 60,000 件の Linked Data を利用し，保有するデータ量の算出を行い，DSNotify，リンク構造解析手法との比較を実施した．評価結果を表 6 に示す．

表 6　データ量削減に関する評価結果

	データ量
本提案フレームワーク	7.45MB
DSNotify	599.39MB
リンク構造解析手法	574.39MB

評価の結果，本提案フレームワークで保有するデータ量は 7.45MB，DSNotify のデータ量は 599.39MB，リンク構造解析手法のデータ量は 574.39MB となり，大幅なデータ量の削減を確認した．これは，本フレームワークの，バイナリ特徴ベクトルによる Linked Data の近似が有効に作用しているものと考えられる．

6.3 精度に関する評価

精度に関する評価として，DBpedia 3.8 と DBpedia 3.9 を利用し，リンクの移動が発生している 400 件のデータを対象にリンク切れ修復精度を算出し，DSNotify，リンク構造解析手法との比較を実施した．正解とするデータは，DBpedia 3.8 と DBpedia 3.9 間でリダイレクトが発生しているデータとした．本評価結果を表 7 に示す．

表 7　精度に関する評価結果

	精度
本提案フレームワーク	89.75%
DSNotify	67.75%
リンク構造解析手法	91.50%

本提案フレームワークの精度が 89.75%，DSNotify の精度は 67.75%，リンク構造解析手法の精度は 91.50%という結果となり，DSNotify と比較し，22%の精度の向上，リンク構造解析手法とは同程度の精度であることを確認した．DSNotify よりも精度が向上した要因としては，本提案フレームワークで用いた手法が，実際の Linked Data の移動に対してうまく対応できているためであると考えられる．また，本提案手法でリンク移動先集合が適切に生成されなかったものとしては，少数のリンクしか保持していない Linked Data や，内容の大幅な変更があった Linked Data などがあった．

6.4 スケーラビリティに関する評価

スケーラビリティに関する評価として，DBpedia 3.8 から無作為に抽出した 400 件の Linked Data を対象にリンク切れ監視を行う状況を想定し，DBpedia 3.9 の Linked Data 5,000 件を移動先候補としたときの実行時間を表 2 の環境を用いて計測後，DSNotify，リンク構造解析手法との比較を実施した．表 8 にバイナリ特徴ベクトルなどの生成時間を示す．

表 8　生成時間

	生成時間
本提案フレームワークのバイナリ特徴ベクトル生成時間	約 3.55 秒
DSNotify の特徴ベクトル生成時間	約 0.38 秒
リンク構造解析手法のリンク集合生成時間	約 0.16 秒

バイナリ特徴ベクトル生成時間は，本提案フレームワークが約 3.55 秒(内ハッシュ計算時間が約 1.92 秒)，DSNotify が約 0.38 秒，リンク構造解析手法が約 0.16 秒となり，本提案フレームワークが遅いという結果となった．これは，単純に，文字列をバイナリ特徴ベクトルとする DSNotify やリンク集合を取得するリンク構造解析手法に比べ，本提案フレームワークでは RDF の各トリプルのハッシュ値からバイナリ特徴ベクトルを生成するため，その計算のコストが大きいためと考えられる．

DBpedia の場合，約 450 万件の Linked Data が存在しているため，本提案フレームワークによるバイナリ特徴ベクトルの生成には約 1 時間程度必要となる，このため，例えば，本フレームワークは週 1 回程度の運用が考えられる．

ⓒ 2017 Information Processing Society of Japan

次に，類似度計算時間の算出実験結果を表 9 に示す．

表 9　類似度計算時間の算出実験結果

	類似度計算時間
本提案フレームワーク	約 14.86 分
DSNotify	約 2.25 ヶ月
リンク構造解析手法	約 14.51 時間

表 9 から，本提案フレームワークが約 14.86 分，DSNotify が約 2.25 ヶ月，リンク構造解析手法が約 14.51 時間となり，関連研究と比較し，本提案フレームワークが大幅に高速であるという結果となった．本結果から，Linked Data が今後普及し，データ数が増加していった場合でも，本提案フレームワークは十分に対応可能であると考えられる．この結果の要因としては，バイナリ特徴ベクトルによる比較が，DSNotify の文字列距離による比較よりも有効に作用したものと考えられる．

7. 考察

本提案手法の適用範囲を知るための調査を行った．調査方法，調査結果を以下に示す．

7.1 調査方法

本調査方法として，Internet Archive の Wayback Machine[9]を利用し，過去に保存された Linked Data の URI に対し，現在リンク切れが発生しているかの調査を行った．調査したデータセットは表 10 に示す 7 カテゴリ，37 データセットである．

表 10　調査したデータセットの概要

カテゴリ	概要
Government	政府が発行した統計などに関する Linked Data
Media	映画，音楽，テレビに関する Linked Data
Linguistics	オントロジーなど言語学に関する Linked Data
Publications	図書館情報や科学出版物に関する Linked Data
Life Sciences	生物学や薬学に関する Linked Data
Cross-Domain	ナレッジベースなどの Linked Data
Geographic	地理情報に関する Linked Data

7.2 調査結果

図 6 に，リンク切れの発生率を表したグラフを示す．

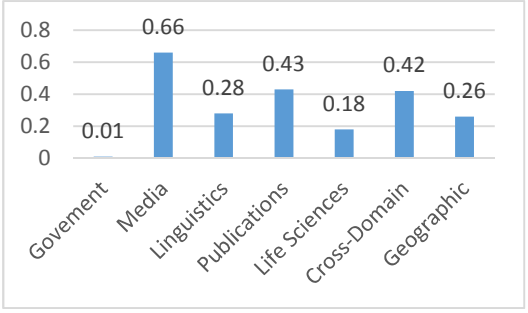

図 6 リンク切れ発生率

Media カテゴリでは，約 6 割ほどリンク切れが発生している．このため，本提案フレームワークが有効に作用するものと考えられ，リンク切れ監視間隔を週 1 回や，2 週に 1 回など短くするという運用方法が考えられる．これに対し，Government カテゴリでは，殆どリンク切れが発生していないことから，3 ヶ月に 1 度や，半年に 1 度など，リンク切れ監視間隔を短くするといった運用方法が考えられる．

8. おわりに

本研究では，今後 Linked Data の普及により，Linked Data の量が増加した場合，関連研究の手法では有用性が損なわれることを問題点とし，この解決を図った．また，この問題点を解決するため，リンク切れ修復に必要な保有するデータ量を削減し，効率的に Linked Data の修復が可能な Linked Data 修復フレームワークの提案と評価を行った．評価の結果，関連研究と比較し，保有するデータ量の削減，精度の向上，実行時間の削減を確認した．

参考文献

[1]Christian Bizer, Tom Heath, Tim Berners-Lee:Linked Data - The Story So Far, International Journal on Semantic Web and Information Systems, Vol.5(3), pp.1--22 (2009).
[2]Tim Berners-Lee: Design Issues: Linked Data http://www.w3.org/DesignIssues/LinkedData.html
[3]RDF 1.1 Concepts and Abstract Syntax http://www.w3.org/TR/rdf11-concepts/
[4]Bernhard Haslhofer, Niko Popitsch: DSNotify - Detecting and Fixing Broken Links in Linked Data Sets, Proc. of the 8th International Workshop on Web Semantics (WebS'09), co-located with DEXA 2009, pp.89--93 (2009).
[5]Niko Popitsch, Bernhard Haslhofer: DSNotify: Handling Broken Links in the Web of Data, Proc. of the 19th International Conference on World Wide Web, pp.761--770 (2010).
[6]三上考明，児玉英一郎，王家宏，高田豊雄:リンク構造解析による Linked Data のリンク切れ先同定手法の提案，電気関係学会東北支部連合大会講演論文集, p.151 (2011).
[7]State of the LOD Cloud http://lod-cloud.net/state/
[8]Christian Bizer et al.: DBpedia Querying Wikipedia like a Database, Developers track presentation at the 16th International World Wide Web Conference (2007).
[9]Internet Archive: Wayback Machine http://archive.org/web/

ⓒ 2017 Information Processing Society of Japan