Vol.2017-DPS-172 No.15
Vol.2017-SPT-26 No.15
Vol.2017-EIP-78 No.15
2017/11/30

Linked Data におけるリンク切れ修復の際の計算時間削減手法に関する提案

常木　翔太[†1]　児玉 英一郎[†1]　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 2006 年に Tim Berners-Lee によって提唱された概念であり，外部から参照可能な RDF(Resource Description Framework)に従い記述されたデータ(RDF データ)を主に取り扱う．RDF データ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，RDF データ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 RDF データにおいても起こり得る．その多くは RDF トリプルにおいて Subject や Object に指定された URI 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，このリンク切れ修復の際の計算時間削減手法に関する提案を行う．また，本提案に対し，基本性能評価，修復の際の計算時間を構成する特徴ベクトルなどの生成時間，移動先候補集合生成時間に関する評価を行ったので，その評価結果について報告する．

キーワード：Linked Data,　リンク切れ修復

A Method to Reduce Processing Time of Restoring Broken Links of Linked Data

Shota Tsuneki[†1] Eiichiro Kodama[†1] 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 mainly deal with data that is represented in accordance with outside-referable RDF (Resource Description Framework) (RDF Data). Since many applications using RDF data acquire information by tracing the links, it is important to maintain link data functional all the time. As routinely occurs in Web, broken links also occur for the RDF data. Broken links are generally caused by deleting or migrating of URIs specified as Subjects or Objects in RDF triples. For broken links caused by the migration, finding and then restoring them are necessary. This paper proposes a method to reduce processing time in restoring broken links, studies its performance, and reports and discusses experiment results, giving processing time in generating feature vectors and destination candidate sets, which constitute the processing time at the time of restoration.

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行われている．Linked Data とは，2006 年に Tim Berners-Lee によって提唱された概念であり[1][2], 外部から参照可能な RDF(Resource Description Framework)に従い記述されたデータ(RDF データ)を主として取り扱う．また，RDF データは Subject, Predicate, Object の 3 要素からなる RDF トリプル[3]の集合となっている．Linked Data に関する研究は海外でも盛んに行われており，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．

RDF データ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, RDF データ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 RDF データにおいても起こり得る．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，データベース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. このような場合に，リンク切れが生じ，関連する情報の取得が行えなくなってしまう．

例えば, 銀行が公開している API を利用して銀行口座の利用履歴を自動的に取得するアプリケーションがあり，銀行の API の URI は銀行側が管理していて，アプリケーションは RDF データをたどって API にアクセスしているものとする．

このとき, 銀行の名称変更で銀行の URI が変わってしまうと, アプリケーションは名称変更があった銀行の API にアクセスできなくなり利用履歴を取得できなくなってしまう．このように，リンク切れが発生した RDF データにアクセスするアプリケーションは正常に動作せず，不具合が生じる．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．

このような状況のもと，本研究では，Linked Data におけるリンク切れ修復の際に精度を落とすことなく計算量を削減する手法の提案を行う．また本提案手法に関する基本性能評価，計算時間削減に関する評価の結果についても報告を行う．

†1　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

Vol.2017-DPS-172 No.15
Vol.2017-SPT-26 No.15
Vol.2017-EIP-78 No.15
2017/11/30

2. 関連研究

Linked Data におけるリンク切れ修復の研究としては，DSNotify [4][5]や Linked Data におけるリンク切れ修復フレームワークに関する研究[6]が知られている．

DSNotify は，各 URI に対し，英数字からなる特徴ベクトルを生成し保存する．その後，定期的に RDF ストアの変更を監視する．監視時にリンク切れの URI を発見した場合，リンク切れを起こした URI の特徴ベクトルと新規に追加された各 URI の特徴ベクトル間の類似度を$Levenshtein$距離で算出し，閾値以上であれば，リンクの移動が発生したものとみなし，リンクの修復を行う．この DSNotify においては，英数字からなる特徴ベクトルを保持するためデータ量が膨大になってしまうという問題がある．また，実行時間が長い点も問題である．この問題点を解決するため清水らの手法[2]が提案されている．

清水らの手法では，各 URI に関係するデータからバイナリ特徴ベクトルを生成し，リンク切れが起きた URI のバイナリ特徴ベクトルと，Key-Value ストアに保存されているその他の URI のバイナリ特徴ベクトルとの類似性を計算後，リンク移動先候補集合を生成，通知することでリンク切れ修復を行う研究となっている．URI の特徴ベクトルをバイナリベクトルとして持つことにより，データ量の削減及び実行時間の削減を実現している．

3. 関連研究の問題点

関連研究[2]では，要素数が少ない RDF データに対しバイナリ特徴ベクトルを生成した際に，抽出可能な特徴が少なくなるという点が問題視されている．このとき，他の RDF データとの類似性が高くなってしまう傾向があり適切にリンクの修復を行えない．

また，各バイナリ特徴ベクトル間の類似性計算時間が，Key-Value ストアのサイズに比例するという点が問題視されている．例えば，保存されているバイナリ特徴ベクトルの件数が 5,000 件の場合，URI1 件あたりの類似性計算時間は約 2 秒かかる．しかし，保存されているバイナリ特徴ベクトルの件数が 2014 年時点における DBpedia 内の項目数に該当する URI の数である 450 万件だった場合，約 33 分もかかってしまう．

LOD Cloud 内の RDF トリプルの数は，表 1 に示すように[7]， 2007 年時点で 5 億トリプルであったものが，2011 年には 310 億トリプルと急増しており，今後，Linked Data が普及し，RDF トリプル数が増加した場合，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．

本研究では，これら 2 つの問題点のうち，計算量の削減に関する問題を解決する手法を提案する．

表 1　LOD Cloud 内の RDF トリプル数の推移

年度	Dataset 数	RDF トリプル数
2007	12	500 百万件
2008	45	2,000 百万件
2009	95	6,700 百万件
2010	203	26,900 百万件
2011	295	31,000 百万件
2014	1,014	——

4. リンク切れ修復の際の計算時間削減手法の提案

本研究では，Linked Data におけるリンク切れ修復の際に精度を落とすことなく計算量を削減する手法を提案する．

4.1 本提案手法が想定している動作環境の例

本提案で想定している動作環境のモデルの例を図 1 に示す．

図 1 の LOD Cloud 内の各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，URI，バイナリ特徴ベクトル，特徴ベクトルのノルムを管理するデータベース(DB)，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ている．本提案手法は，図 1 内のバイナリ特徴ベクトル生成モジュールとリンク移動先候補集合生成モジュール間で動作することを想定している．

4.2 本提案手法

以下，本提案手法の詳細を示す．表 2 に本提案手法の説明に用いる記号を示す．

表 2 本提案手法の説明に用いる記号

$\boldsymbol{U}$	URI の集合
$\boldsymbol{R(u)}$	$u \in U$を Subject にもつ RDF トリプルの集合
$\boldsymbol{O_{R(u)}}$	$R(u)$内の各トリプルの Object である URI の集合
$\boldsymbol{h}$	一方向性ハッシュ関数
$\boldsymbol{l}$	正の整数，$L = 2^l$
$\boldsymbol{i}$	0 以上の整数
$\boldsymbol{LSB(b,i)}$	バイナリ文字列bの最下位iビット

● バイナリ特徴ベクトルなどの生成と保存

$u \in U$に対して$R(u)$が与えられたとき，(1)から(4)の手順により，$R(u)$ のバイナリ特徴ベクトル$char(R(u))$をL次元バイナリベクトル$(b_0, b_1, \dots, b_{L-2}, b_{L-1})$，$b_i \in \{0,1\}$として求め，ノルム$\|char(R(u))\|$を計算後，保存する．ここで，記号⊕は排他的論理和とする

(1) $char(R(u)) = (b_0, b_1, \dots, b_{L-2}, b_{L-1}) = (0,0,\dots,0,0)$ とする

Vol.2017-DPS-172 No.15
Vol.2017-SPT-26 No.15
Vol.2017-EIP-78 No.15
2017/11/30

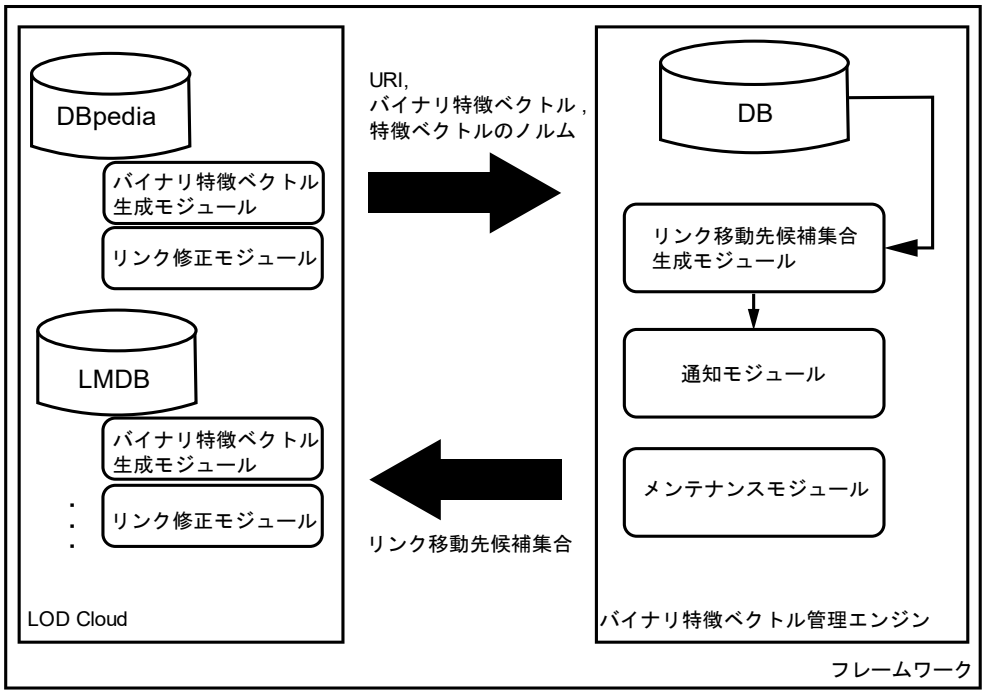

図 1 動作環境のモデルの例

(2) $o \in O_{R(u)}$ に対して，$b_{LSB(h(o),l)} \oplus = 1$ によって $b_{LSB(h(o),l)}$を決定する．

(3) (2)で生成されたバイナリ特徴ベクトル$char(R(u))$のノルム$\|char(R(u))\|$を計算する．

(4) uと$char(R(u))$，$\|char(R(u))\|$を DB へ保存する．

● **リンク移動先候補集合の生成**

$u_1, u_2 \in U$に対して$R(u_1), R(u_2)$が与えられたとき，$R(u_1), R(u_2)$のバイナリ特徴ベクトルのハミング距離を式(1)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．

$$hamming(R(u_1), R(u_2)) = support\left(char(R(u_1)) \oplus char(R(u_2))\right) \quad (1)$$

u_1とu_2の間にリンクが張られており，u_2がu_2'へ変更された場合，u_2'を含む DB 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) $\|char(R(u_2))\|$を求め，εをパラメータとし，DB 内に格納されている$char(R(u_2))$以外の式(2)を満たす$char(R(u))$を取得する．

$$\|char(R(u_2))\| - \varepsilon \leq \|char(R(u))\| \leq \|char(R(u_2))\| + \varepsilon \quad (2)$$

(4) 取得した各$char(R(u))$に対し，$hamming\left(char(R(u_2)), char(R(u))\right)$を計算し，$hamming\left(char(R(u_2)), char(R(u))\right) / L < \theta$ ならば $S = S \cup \{u\}$とし，リンク移動先候補集合 Sを構築する．

5. 提案手法の動作例

本提案手法を例を用いて説明する．

$u_a \in U$に対して，$O_{R(u_a)} = \{o_1, o_2\}$とする．また$l = 4$ ，$L = 16$として，$char(R(u))$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などの生成と保存**

LOD Cloud 内のサービス提供サイトの RDF ストアに$R(u_a)$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$R(u_a)$を取得する．その後，$O_{R(u_a)} \ni o_1$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，$h(o_1) =$ "…0101 0001"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$LSB(h(o_1), 4) =$ "0001" は 10 進数で1であるため，$char(R(u_a))$のb_1成分を 0 - 1 反転する．さらに，$O_{R(u_a)} \ni o_2$ にハッシュ関数hを適応し，ビット列を取り出す．例として，$h(o_2) =$ "…0110 0110"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$LSB(h(o_2), 4) =$ "0110"は 10 進数で6であるため，$char(R(u_a))$のb_6成分を 0 - 1 反転する．以上の手順により，$char(R(u_a)) = (0,1,0,0,0,0,1,0,\ldots,0)$となる．

次に，$char(R(u_a))$の 1 の数をカウントする．$char(R(u_a)) = (0,1,0,0,0,0,1,0,\ldots,0)$であるため 1 の数は 2 つとなり，$\|char(R(u_a))\| = 2$とする．最後に，$u_a$，生成したバイナリ特徴ベクトル $char(R(u_a))$，ノルム $\|char(R(u_a))\|$を DB へ保存する．

●リンク移動先候補集合の生成

u_aが$u_{a'}$へ移動し$u_{a'}$のバイナリ特徴ベクトルが計算され，DB へ格納されているものとする．u_aのリンク切れを発見した場合には，DB へu_aを利用してクエリを実行し，u_aのバイナリ特徴ベクトル$char(R(u_a))$とノルム $\|char(R(u_a))\|$を取得する．次に取得した $\|char(R(u_a))\|$とパラメータεをもとに式(2)を用いて比較するバイナリ特徴ベクトルの集合を取得する．ここでは，DB に保存してある URI はu_a以外に $u_{a'}$, u_b, u_c, u_d, u_eだけであったとし，DB の内容は表 3 のようになっていたとする．

表 3　DB に保存されているデータ

URI	バイナリ特徴ベクトル	ノルム
u_a	$char(R(u_a)) = (0,1,0,0,0,0,1,0,\dots,0)$	2
$u_{a'}$	$char(R(u_{a'})) = (0,1,1,0,0,0,1,0,\dots,0)$	3
u_b	$char(R(u_b)) = (1,0,0,1,0,1,1,0,\dots,0)$	4
u_c	$char(R(u_c)) = (1,0,1,1,0,1,0,0,\dots,0)$	4
u_d	$char(R(u_d)) = (0,1,1,1,0,0,1,0,\dots,0)$	4
u_e	$char(R(u_e)) = (1,1,0,1,1,1,0,0,\dots,0)$	5

このときパラメータεが 2 であったとすると，式(2)を満たすノルムをもつu_a以外の URI は$u_{a'}$, u_b, u_c, u_dであるため，これらのバイナリ特徴ベクトル$char(R(u_{a'}))$, $char(R(u_b))$, $char(R(u_c))$, $char(R(u_d))$を取得する．

次に，$char(R(u_a))$と取得してきた各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．計算結果を図 2 に示す．

$$hamming\left(\mathrm{char}(R(u_a)), char(R(u_{a'}))\right) = 1$$
$$hamming\left(\mathrm{char}(R(u_a)), char(R(u_b))\right) = 4$$
$$hamming\left(\mathrm{char}(R(u_a)), char(R(u_c))\right) = 6$$
$$hamming\left(\mathrm{char}(R(u_a)), char(R(u_d))\right) = 2$$

図 2 $hamming$距離の計算結果

計算したhamming 距離をLで除算し，URI Aとの非類似性を計算する．図 3 に計算結果を示す．

$$hamming\left(\mathrm{char}(R(u_a)), char(R(u_{a'}))\right) / L = 0.0625$$
$$hamming\left(char(R(u_a)), char(R(u_b))\right) / L = 0.25$$
$$hamming\left(\mathrm{char}(R(u_a)), char(R(u_c))\right) / L = 0.375$$
$$hamming\left(\mathrm{char}(R(u_a)), char(R(u_d))\right) / L = 0.125$$

図 3 非類似度の計算結果

図 3 より$\mathrm{char}(R(u_a))$と$char(R(u_{a'}))$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$u_{a'}$, u_dが閾値以下となるため，リンク移動先候補集合Sは，$S = \{u_{a'}, u_d\}$となる．

6. 評価

6.1 評価環境

本提案手法の評価を行うために，構築実験として本手法に従ったプロトタイプの実装を行った．本実装環境を表 4 に示す．

表 4　実装環境

OS	Windows 10 Education
プロセッサ	Intel Core i7 – 7500U
メモリ	16GB
使用言語	Python 3.6.2
RDF ストア	Virtuoso 7.2.4.2
DB	SQLite3
Key-Value Store	Redis 3.2

本構築実験で作成したプロトタイプを利用し，基本性能評価として，保有するデータ量に関する評価を行った．また，本手法の有用性を示すために，修復の際の計算時間を構成する特徴ベクトルなどの生成時間，移動先候補集合生成時間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である DBpedia 3.8 と DBpedia 3.9 を利用した．評価用データの概要を表 5, DBpedia を構成するデータセットを表 6 に示す．

表 5　データセットの概要

	DBpedia 3.8	**DBpedia 3.9**
作成日	2012 年 6 月	2013 年 4 月
項目数	3,769,926	4,004,478
トリプル数	177,867,270	198,414,516
1 項目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%)	

表 5 内の項目数は Wikipedia に掲載されている各記事の数に相当し，DBpedia では，記事の項目名を利用して RDF データに用いる URI を作成している．

DBpedia 3.8 と DBpedia 3.9 を比較したところ，2,386,041 件(59.58%)の URI 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 URI の移動例を表 7 に示す．また，評価環境に用いたパラメータの値としては，$l = 9$，$L = 512$，$\theta = 0.25$，$\varepsilon = 5$とした．

6.2 基本性能評価

本提案手法の基本性能評価として，保持するデータ量に関する評価を行った．

6.2.1 保持するデータ量に関する評価

DBpedia3.9 から無作為に抽出した URI 50,000 件の特徴ベクトルなどを DB へ格納し，DB のメモリ使用量とダンプデータのデータ量の算出を行った．その後，清水らの手法との比較を行った．本評価結果を表 8 と表 9 に示す．

表 6　DBpedia を構成するデータセット

DBpedia を構成するデータセット	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

表 7 実際に発生していた URI の移動例

移動前	移動後
http://dbpedia.org/resource/San_Vittorino_(L'Aquila)	http://dbpedia.org/resource/San_Vittorino
http://dbpedia.org/resource/Schiffler's_theorem	http://dbpedia.org/resource/Schiffler_point
http://dbpedia.org/resource/Tempest_(DC_Comics)	http://dbpedia.org/resource/Tempest_(comics)
http://dbpedia.org/resource/WNJR_(AM)	http://dbpedia.org/resource/WNSW
http://dbpedia.org/resource/Web-footed_Coquí	http://dbpedia.org/resource/Web-footed_coquí
http://dbpedia.org/resource/Pseudo-ainhum	http://dbpedia.org/resource/Ainhum
http://dbpedia.org/resource/Bobby_Thomas_(jazz_drummer)	http://dbpedia.org/resource/Bobby_Thomas
http://dbpedia.org/resource/Constance_Burge	http://dbpedia.org/resource/Constance_M._Burge
http://dbpedia.org/resource/Kadal	http://dbpedia.org/resource/Kadal_(film)
http://dbpedia.org/resource/Photoglo	http://dbpedia.org/resource/Jim_Photoglo
http://dbpedia.org/resource/Carrier_Strike_Group_6	http://dbpedia.org/resource/Carrier_Strike_Group_Six
http://dbpedia.org/resource/Nyxem_Worm	http://dbpedia.org/resource/Blackworm
http://dbpedia.org/resource/Katharine_Blake	http://dbpedia.org/resource/Catherine_Blake_(disambiguation)
http://dbpedia.org/resource/Charles_Henderson_(politician)	http://dbpedia.org/resource/Charles_Henderson
http://dbpedia.org/resource/Howie_Centre,_Nova_Scotia	http://dbpedia.org/resource/Howie_Centre

表 8 メモリ使用量の評価結果

	データ量
本提案手法	12.0MB
清水らの手法	9.9MB

表 9 ダンプした際のファイルサイズの評価結果

	ファイルサイズ
本提案手法	7.02MB
清水らの手法	5.82MB

評価の結果，本提案手法のメモリ使用量は 12.0MB，ダンプした際のファイルサイズは 7.02MB，清水らの手法のメモリ使用量は 9.9MB，ファイルサイズは 5.82MB となり，本提案手法のほうが約 2MB 多くメモリを使用することが分かった．これは本提案手法では，URI とバイナリ特徴ベクトルに加え，特徴ベクトルのノルムを管理しているためである．

6.3 特徴ベクトルなどの生成時間に関する評価

DBpedia3.9 から無作為に抽出した URI 50,000 件の特徴ベクトルなどの生成時間の計測を行った．そして，清水らの手法との比較を行った．本評価結果を表 10 に示す．

表 10 特徴ベクトルの生成時間の評価結果

	生成時間
本提案手法	258 秒
清水らの手法	246 秒

評価の結果，本提案手法の特徴ベクトルなどの生成時間は 258 秒，清水らの手法の特徴ベクトルの生成時間は 246 秒となった．本提案手法では，特徴ベクトルに加え，ノルムの生成も行っているが，ノルムの生成時間は，12 秒程度

Vol.2017-DPS-172 No.15
Vol.2017-SPT-26 No.15
Vol.2017-EIP-78 No.15
2017/11/30

に抑えられていることがわかった．

6.4 移動先候補集合生成時間に関する評価

DBpedia 3.8 と DBpedia 3.9 を利用し，リンクの移動が発生している 300 件のデータを対象にリンク移動先候補集合の生成を行った，DB にはリンクの移動が発生していた URI の移動後の URI を含む，5,000 件の URI が登録されているものとし，その実行時間を表 4 の環境を用いて計測後，清水らの手法との比較を行った．本評価結果を表 11 に示す．

表 11 リンク移動先候補集合生成時間の評価結果

	生成時間
本提案手法	55.7 秒
清水らの手法	164.7 秒

表 11 に示すように，本提案手法のリンク移動先候補集合生成時間は，清水らの手法に比べ，109 秒削減できている．

7. 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，リンク切れ修復の際に精度を落とすことなく計算時間を削減する手法の提案と評価を行った．評価の結果，関連研究と比較し，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]児玉英一郎，常木翔太，清水小太郎，王家宏，高田豊雄 ： Linked Data におけるリンク切れ修復フレームワークの提案と評価，情報処理学会研究報告数理モデル化と問題解決（MPS），Vol.2017-MPS-112，pp.1--6 (2017)..
[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).