Vol.2014-IOT-27 No.16
Vol.2014-SPT-11 No.16
2014/10/10

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

清水 小太郎[1]　児玉 英一郎[1]　王 家宏[1]　高田 豊雄[1]

概要：近年，Linked Data に関する研究が多数行われている．これらの研究の中でも，Linked Data のリンク切れ修復に関する研究は特に重要であり，その 1 つとして DSNotify というリンク切れ修復フレームワークに関する研究が知られている．この DSNotify では，英数字の特徴ベクトルを利用し，リンク切れ修復を行うという手法をとっており，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問題点が知られている．そこで，本研究では，この問題点を解決するため，Linked Data を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，基本性能評価，精度，スケーラビリティの点で評価を行ったので，この評価結果についても報告を行う．

キーワード：Linked Data, リンク切れ修復

A Framework for Restoring Broken Links of Linked Data

Kotaro Shimizu[1]　Eiichiro Kodama[1]　Jiahong Wang[1]　Toyoo Takata[1]

Abstract: Recently researches on Linked Data are being actively conducted. Among these, the research on restoring broken links of Linked Data is particularly important, and a well-known research result is the DSNotify framework. For DSNotify, broken links are restored by utilizing a feature vector of alphanumeric characters, which has the problem that, the amount of data required to restore broken links would become large enough to degrade system performance. To solve this problem, this paper proposes a new framework for restoring broken Linked Data, which represents Linked Data approximately with the feature vector. Using the proposed framework, broken Linked Data could be restored with largely reduced data size and very high restoring speed. The proposed framework has been evaluated in terms of basic performance, accuracy, and scalability. Evaluation results have been 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行われている．Linked Data とは，外部から参照可能な RDF(Resource Description Framwwork) 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 [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

[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

© 2014 Information Processing Society of Japan

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 が変更される事が考えられる．このような状況では，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と，リンク構造解析によるリンク切れ先推定手法が知られている．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[5], [6] 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 のリンクの修復を行うシステムである．DSNotify に関する研究では，Linked Data の移動により発生するリンク切れを，リンク先の Linked Data の削除と作成の 2 つの事象が連続した状態と定めている．ここでは，ある Linked Data サイトの管理者をユーザと仮定し，DSNotify を用いて，外部の Linked Data サイトとのリンクメンテナンスを行っている場合を考える (図 **1** 参照).

以下，それぞれのモジュールの説明を行う．Monitor は，Linked Data の移動によって起こるリンク切れ検出のために，定期的な Linked Data サイトの監視を行っている．また，この Monitor は，Linked Data の特徴ベクトルを生成し，Indices へ格納する．

Indicies には，Item Index, Removed Item Index, Archived Item Index の 3 つの格納先が存在する．まず最初に，Monitor が生成した特徴ベクトルは，Item Index へ格納される．また，Monitor の監視によって Linked Data の削除が確認された場合には，削除された方の Linked Data の特徴ベクトルを Item Index から，新規の Linked Data の特徴ベクトルと共に，Removed Item Index へ変更する．

Monitor は，監視によって新規に作成された Linked Data を発見した場合には，その特徴ベクトルを利用し，Removed Item Index に格納されている特徴ベクトルと類似度 (*Levenshtein* 距離を利用) の計算を行う．類似度が高い特徴ベクトルは，Archived Item Index へ格納される．その後，ユーザの設定により，Decider や LOD Site Updator を用いて，Linked Data の修正を行うアプリケーションへ通知を発行することが可能である．

また，リンク集合の類似度から，リンクの移動先を推定する，リンク構造解析手法としては，リンク構造解析による Linked Data のリンク切れ先同定手法 [4] 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，事前に保有してある各 Linked Data のリンク先集合と，新規に作成された Linked Data 群のリンク先集合との類似度を，*Jaccard* 係数を用いて計算をする．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．

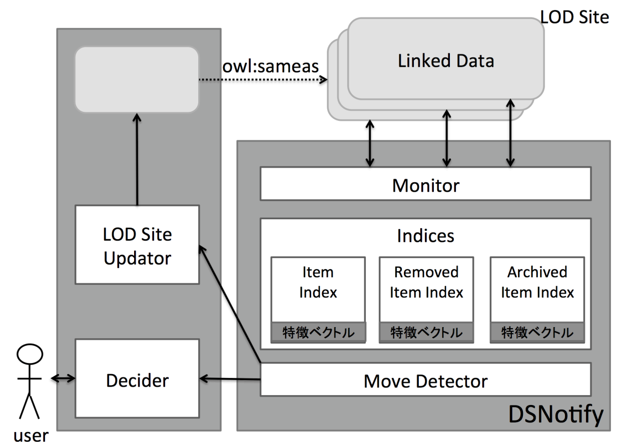

図 **1** DSNotify のモデル

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 のデータサイズは式 (1) にて近似できる．

$$\begin{aligned}&1\,\text{トリプルのObject のデータサイズ} = \\ &\qquad\text{Object の URI 長の平均値}* \\ &\qquad 1\,\text{文字あたりのバイト数}\end{aligned} \tag{1}$$

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 $26 * 2 = 52$ byte，LOD Cloud 全体としては，$52 * 310$ 億 $= 1.61$ TB となり，約 1.61 TB 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．

$$\begin{aligned}&\textit{Levenshtein}\ \text{距離 (リンク切れを起こした URI,} \\ &\text{Object の URI) の計算に要する計算量} * 310\,\text{億}\end{aligned} \tag{2}$$

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

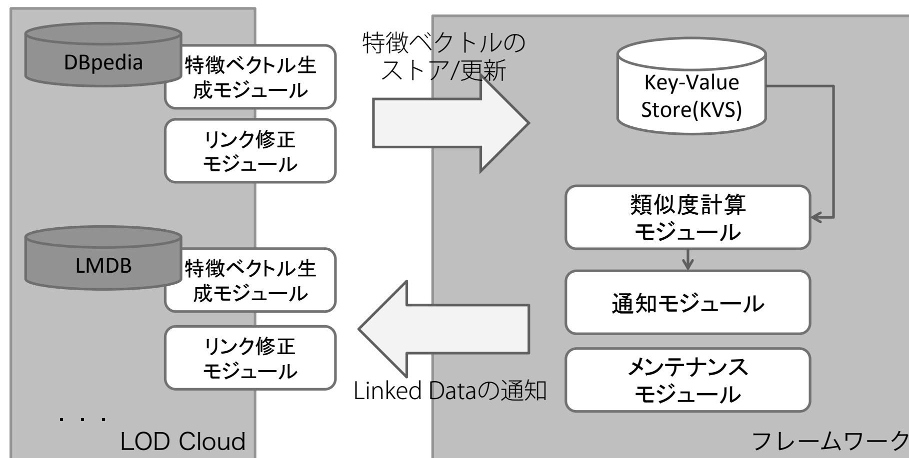

図 2　Linked Data におけるリンク切れ自動修復フレームワークのモデル

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の概算は，DSNotify と同様に 1.61 TB 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．

$$Jaccard \text{ 係数 (リンク切れを起こした URI の集合,} \\ \text{リンク移動候補 Linked Data のリンク集合)} \quad (3) \\ \text{の計算に要する計算量} * 3.10 \text{ 億}$$

しかし，表 **1** に示すように [8]，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 Open Data のデータ推移

年度	Dataset 数	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

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

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 **2** に示す．

LOD Cloud 内の各サービス提供サイトには，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，本提案フレームワーク側は，Key-Value Store(KVS) と類似度計算モジュール，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，それぞれのモジュールの動作について説明する．

- 特徴ベクトル収集時
  特徴ベクトル収集モジュールは，LOD Cloud 内の各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特徴ベクトルの生成を行う．その後，特徴ベクトルと URI をフレームワークへ送信する．特徴ベクトルと URI を受け取ったフレームワークは，それぞれを Key-Value Store へ保存する．
- リンク切れ検出時
  特徴ベクトル生成モジュールは，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 をフレームワーク側へ通知する．類似度計算モジュールは，通知された URI の特徴ベクトルから類似度を計算し，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を通知モジュールにより，RDF ストア側へ通知し，RDF ストア側のリンク修正モジュールにおいて，RDF ストア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元にリンクの修正を行う．
- メンテナンス時
  リンク切れ修復が完了し，必要がなくなった特徴ベクトルや，一定期間が経過し利用されなくなった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．

4.1　特徴ベクトル生成モジュールのアルゴリズム

以下，本提案独自の特徴ベクトル生成モジュールの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，リンク候補先集合を求めるものである．図 **3** に，アルゴリズムの説明で用いる記号を示す．

図 3 の記号のもと，本アルゴリズムは以下のものとなる．

© 2014 Information Processing Society of Japan

B，C：Linked Data
$L(B)$：Linked Data B 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 **3**　アルゴリズムの説明で用いる記号

- 特徴ベクトルの生成
  Linked Data B が与えられたとき，B の特徴ベクトル $char(B)$ を，(1) から (2) の手順により，L 次元ベクトル $(b_1,b_2,\cdots,b_L)$ として求める．但し，$b_i \in \{0,1\}$ とする．また，記号 $\oplus$ は排他的論理和とする．
  (1) $(b_1, b_2,\cdots, b_L)$=(0,0,⋯,0) とする
  (2) 各 $u \in L(B)$ 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，生成された特徴ベクトル $char(B)$ をキー，Linked Data B の URI をバリューとして Key-Value Store にストアする．
- リンク移動先の推定 Linked Data X，Y が与えられたとき，X と Y の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．ハミング距離は，同じ長さのビット列に対し，各成分ごとに，異なる値を持つビットの個数を計算するものである．

$$hamming(X,Y) = support(char(X) \oplus char(Y)) \tag{4}$$

  Linked Data A と Linked Data B の間にリンクが張られており，Linked Data B が Linked Data B' へ移動した場合，B' を含む Key-Value Store からリンク候補先集合 S を以下の手順で求める．
  (1) $0 < \theta < 1$ とする
  (2) $S = \phi$　（空集合）とする
  (3) 各 $B' \in KVS$ に対し $hamming(B, B')$ を計算し，$hamming(B, B')/L \leq \theta$ ならば $S = S \cup \{B'\}$ とし，リンク候補先集合 S を構築する

5. アルゴリズムの動作例

Linked Data A が図 **4** のようにリンクを２つ保持していると仮定する．また $l = 4$，$L = 2^4$ として，$char(B)$ は 16 次元の特徴ベクトルとする．

特徴ベクトル収集時

LOD Cloud 内の各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．特徴ベクトル生成モジュールが，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 にハッシュ関数を適用し，ビット列を取得する．ここでは例として，”$\cdots$0101 0001” が生成されたと仮定する．l は 4 であるため，取得したビット列の下位 4 ビットを取り出す．この取り出したビット列 ”0001” は 10 進数で 1 であるため，$char(A)$ の 1 ビット目を $0-1$ 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，”$\cdots$1101 0110” というビット列を取得したと仮定する．同様に，l は 4 であるため，下位 4 ビットを取り出す．この，ビット列 ”0110” は 10 進数で 6 であるため，$char(A)$ の 6 ビット目を $0-1$ 反転する．以上の手順により，$char(A) = (0, ..., 1, 0, 0, 0, 0, 1)$ となる．最後に，生成した特徴ベクトル $char(A)$ と，Linked Data B の URI をペアとしてフレームワークへ送信する．これを受け取ったフレームワークは，特徴ベクトルをキー，Linked Data の URI をバリューとして Key-Value Store へ保存する．

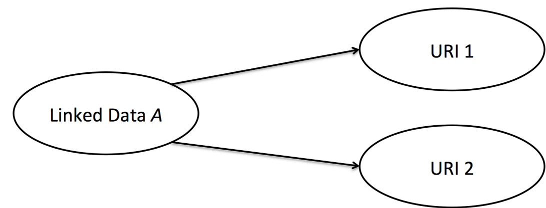

図 **4**　Linked Data A の保持しているリンク

リンク切れ発見時

Linked Data A が Linked Data A' へ移動し，Linked Data A' の特徴ベクトルは計算され，Key-Value Store へ保存されているものとする．特徴ベクトル生成モジュールが，RDF ストアを監視中に，Linked Data A のリンク切れを発見した場合には，Linked Data A の URI をフレームワークへ送信する．これを受け取ったフレームワークの類似度計算モジュールは，Key-Value Store から Linked Data A に対応する特徴ベクトルを取得し，Key-Value Store に保存されている他の Linked Data の特徴ベクトルと比較を行う．各 Linked Data の特徴ベクトルは図 **5** のようになっているものとする．

類似度計算モジュールは，Linked Data A の特徴ベクトルと Key-Value Store に保存されている他の Linked Data の特徴ベクトル間で $hamming$ 距離を計算し，非零成分の個数を計算する．計算結果を図 **6** に示す．

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

図 **5**　保存されている特徴ベクトル

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

図 6　Linked Data$A$ との $hamming$ 距離

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

図 7　Linked Data A との非類似性の計算結果

Linked Data$A$ と Linked DataA' の $hamming$ 距離は 1, Linked Data A と Linked Data$B$ の $hamming$ 距離は 4 となる．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 **7** に計算結果を示す．

図 6 より，Linked Data A と Linked Data A' 間の $hamming$ 距離は 1 であるため，非類似性は 0.0625 となる．ここで，θ を 0.2 とすると 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 内の各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 は Linked Data A' へ移動し，リンク切れ修復が完了した場合，Linked Data A の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は Key-Value Store から削除を行う．

6. 評価

本提案フレームワークの評価を行うため，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．

評価用データとして，DBpedia[11] のスナップショットを利用し，評価を行った．表 **3** に評価に用いたデータの詳細を示す．

DBpedia 3.8 と DBpedia 3.9 を比較したところ，1396 件の Linked Data 中，535 件 (38.32%) 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 **4** に示す．また，評価環境に用いた変数の値としては，$l = 9, L = 512, L\theta = 8$ を用いた．

基本性能評価として，DBpedia 3.8 と DBpedia 3.9 間で本提案手法を用いてリンク切れ監視を行った結果を表 **5** に示す．フレームワーク全体の実行時間は 64.21 秒となり，Key-Value Store のサイズは 12.87 MB となった．Key-Value Store にデータが無い状態のデータサイズは 12.59 MB であるため，実質的に利用しているデータサイズは，0.28 MB 程度である．関連研究の手法をこのデータに適応した場合のデータサイズは 40 MB であるため，関連研究と比較し，約 79%のデータ量の削減となる．また，リンク移動先候補集合の要素数の平均は 29.24 個であった．また，約 88%は集合の要素数が 1 個であり，集合の要素数の最大値は 1115 個であった．この理由としては，特徴ベクトルによるリンク移動先候補集合の作成は有効に働いている一方で，保有している Linked Data が少ない場合，リンク移動先候補集合が増大することが考えられる．

表 **2**　実装環境

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 [10]
Key-Value Store	Kyoto Tycoon 0.9.56 [9]

表 **3**　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	1,396	1,681
トリプル数	174,040	202,553
1Linked Data あたりの平均トリプル数	124.67	120.49
リダイレクト数	535(38.3%)	

表 **5**　DBpedia3.8 - DBpedia3.9 間のリンク切れ監視結果

実行時間	64.21 秒
Key-Value Store サイズ	12.87 MB
リンク移動候補先集合の要素数の平均値	29.24 個

次に，精度評価として，URI の変更が生じていた 535 件の Linked Data を対象に，リンク切れの修復を行った．この結果，生成されたリンク移動先候補先集合には，85% の精度でリンク移動先の URI が含まれていることを確認した．また，リンク候補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 があった．

スケーラビリティの評価として，DBpedia 3.8 から無作為に抽出した 400 件のデータと，DBpedia 3.9 の Linked Data 数を 250 件から 1000 件へ増加させた場合のリンク切れ監視の評価結果を表 **6** に示す．特徴ベクトル生成モジュール実行時間，類似度計算モジュール実行時間は，Linked Data1 件当たりの処理にかかる時間を示している．評価の結果，Linked Data 数が増加した場合，リンク移動候補先集合の要素数の平均値は，殆ど増加しないことが確認され

© 2014 Information Processing Society of Japan

表 4 発生していた URI の変更例 (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	Non-cognitivism
Monty_Python's_Life_of_Brian	Monty_Python's_Life_of_Brian
Two_wrongs_make_a_right_(fallacy)	Two_wrongs_make_a_right

た．これは，生成された特徴ベクトルが Linked Data の内容を正しく近似できており，アルゴリズムによって，リンク移動先を的確に判断出来ているものと考える．

表 6 スケーラビリティ評価

	Linked Data 数			
	250 件	500 件	750 件	1000 件
特徴ベクトル生成モジュール実行時間 (秒)	0.1831	0.3123	0.3033	0.5480
類似度計算モジュール実行時間 (秒)	0.3162	0.6298	0.6178	1.1764
リンク移動候補先集合の要素数の平均値 (個)	11.36	17.93	17.93	17.23

7. まとめと今後の課題

本研究では，今後 Linked Data が普及し，データ量が増加した場合，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，保有するデータ量が関連研究と比較し，79%削減でき，85%の精度であることを確認した．

今後の課題として，保持しているリンクが少ない Linked Data への対応があげられる．保持しているリンクが少ない Linked Data は，カテゴリ等の重複が発生する可能性があり，抽出可能なリンクの特徴が減少する．そのため，他の Linked Data との類似度が高くなってしまう場合があり，本フレームワークでは適切にリンクの修復が行えない．そこで，保持しているリンクが少ない Linked Data には，Linked Data の URI の類似度からリンク移動先を推定する手法との協調動作が考えられる．また，本提案フレームワークの評価には，DBpedia のデータを用いたが，LOD Cloud にある別のサイトでは，保持リンク数が大幅に変わるものもあるため，様々な LOD サイトで大規模な評価を行う必要がある．

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, 5, (3), pp.1–22 (2009).
[2] Tim Berners-Lee: Design Issues: Linked Data, http://www.w3.org/DesignIssues/LinkedData.html, (2006).
[3] RDF 1.1 Concepts and Abstract Syntax http://www.w3.org/TR/rdf11-concepts/
[4] 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,Vol.2011, p.151 (2011).
[5] 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).
[6] 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).
[7] State of the LOD Cloud, http://lod-cloud.net/state/
[8] Christian Bizer: State of the Web of Data, 4th Linked Data on the Web Workshop(LDOW2011). http://events.linkeddata.org/ldow2011/slides/ldow2011-slides-intro.pdf
[9] Kyoto Tycoon http://fallabs.com/kyototycoon/
[10] Orri Erling, Ivan Mikhailov: RDF Support in the Virtuoso DBMS, Studies in Computational Intelligence, Vol.221/2009, pp.7–24 (2009).
[11] Christian Bizer et al.: DBpedia Querying Wikipedia like a Database. Developers track presentation, 16th International World Wide Web Conference(WWW2007).

© 2014 Information Processing Society of Japan