Vol.2013-DPS-156 No.2
Vol.2013-GN-89 No.2
Vol.2013-EIP-61 No.2
2013/9/11

Linked Dataにおけるリンク修復の際の効率的なブロッキング手法の提案

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 に関する研究が盛んに行われている．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．Linked Data で発生したリンク切れは，その発生を検知し，リンクを再構築する必要がある．しかし，Linked Open Data のスケールは大きく，全データを対象に探索すると，その探査には膨大な時間がかかる．そこで，本稿では，Linked Data におけるリンク再構築時に，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，探索空間の縮小を目的を図る効率のよいブロッキング手法を提案する．

キーワード：Linked Data，Semantic Web

A Blocking Approach for Efficiently Restoring Broken Links of Linked Data

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

Abstract: Recently research on Linked Data is being actively conducted. Because for applications that apply the Linked Data, it is often the case that information is provided with using the links, it is important to carefully maintain links of Linked Data. As often encountered in the Web, broken links are common occurrences in the Linked Data. For the broken links in Linked Data, we have to detect their occurrences, and reconstruct them. The problem is that, however, the volume of Linked Data is often extremely high, and too much time is needed if we traverse all the data. Aiming at narrowing the search space and improving the computation cost, in this paper, an effective blocking-based link reconstruction approach is proposed, which is characterized by applying the feature vector of binary data in reconstructing links of Linked Data.

Keywords: Linked Data, Semantic Web

1. はじめに

近年，Semantic Web の新たなデータ共有の方法として，Linked Data[1], [2] 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，2006 年に Tim Berners-Lee によって提唱された概念である [3]．その後，海外では活発に研究が進められ，LOD（Linked Open Data）という，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 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である．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．仮に，移動によって起こるリンク切れが発生した状態のままであった場合，その Linked Data にアクセスするアプリケーションが正常に

[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.2013-DPS-156 No.2
Vol.2013-GN-89 No.2
Vol.2013-EIP-61 No.2
2013/9/11

図 1 トリプルの例

動作できないなどの不具合が起こり得る．リンクの再構築のためには，移動先を発見しなければならないが，LOD 内のリソースはスケールが大きいため，全データを対象にリンク切れ先を探索すると，その探索には膨大な計算コストが必要となる．そこで，効率よく計算を行うことが重要であり，そのための技術として探索空間を絞り込むためのブロッキングと呼ばれる手法が知られている．

このような状況の中，本研究では，Linked Data におけるリンク再構築時に有用な効率のよいブロッキング手法を提案する．

2. Linked Data

Linked Data で利用するメタデータは，RDF を用いて記述され，主語 (Subject)，述語 (Property)，目的語 (Value) の 3 つの要素（トリプル）でリソースの関係情報を表す．トリプルの集合は RDF のグラフと呼ばれ，「主語・目的語間の関係のステートメント」を表す．主語と述語は URI (Uniform Resource Identifer) で表現される．URI は，一定の書式によってリソースを表す識別子のことであり，リソースの場所と名前によって表現されているため，対象となるリソースを一意に特定することができる．また，メタデータは，ソフトウェアによる処理を可能とするために，機械処理可能な形式で書かれている．この形式にはいくつかの種類が存在しており，XML 形式で記述される RDF/XML[4] や，N-Triples[5] が知られている．

図 1 に RDF の有向グラフの例を示す．この例では，Germany が主語，Capital が述語，Berlin が目的語となる．この例を日本語で表現した場合には，「Germany の Capital は Berlin である」という意味となる．

また，トリプルの構成要素である，Property も W3C などの標準化団体が Vocabulary として定義している．代表的なものとして，Dublin Core[6] と，FOAF [7] が挙げられる．Dublin Core は，Web ページや文章の作者，タイトル，作成日といった書誌的な情報をメタデータとして記述するための Vocabulary である．現在では，メタデータ記述のための Vocabulary として，2003 年に国際標準となっている．FOAF は，名前，関心領域，ホームページなどといった，人を描写するための様々な Vocabulary を定義している．これらは，Semantic Web の様々なプロジェクトで用いられるだけではなく，個々人の情報を記述する共通語彙して，RSS などほかの RDF/XML で利用することも可能となっている．

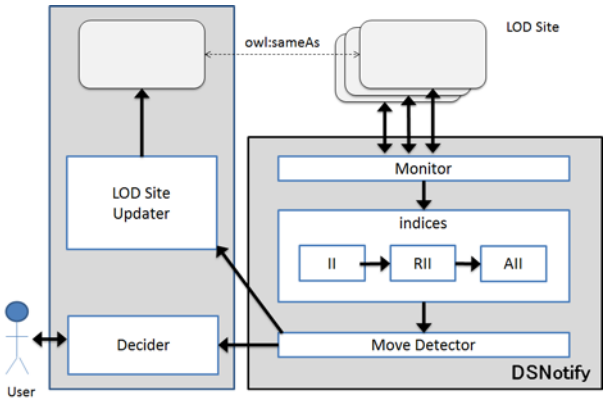

図 2 DSNotify のワークフロー

3. 関連研究

Linked Data におけるリンク修復の研究としては，リンク切れ先の探索，修復を行う DSNotify[8], [9] が知られている．DSNotify において，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 つの事象が連続して起こっている状態と定めている．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 2**）．

DSNotify において，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 つの事象が連続して起こっている状態と定めている．DSNotify では，この Linked Data の移動によっておこるリンク切れの検出のために，図中の Monitor により定期的な Linked Data サイトのクローリングを行っている．Monitor では，Linked Data の特徴ベクトルを保存し，管理用メタデータを付与する．生成された特徴ベクトルは，Indices へ格納される．その格納された Linked Data の状態によって，Item Index（図中 II），Removed Item Index（図中 RII），Archived Item Index（図中 AII）の 3 つの格納先が存在する．Monitor のクローリングにより発見された Linked Data の特徴ベクトルと，管理用メタデータは Item Index に保存される．その後，クローリングにより Linked Data の変更が発見された場合には，特徴ベクトルの更新を行う．この際，現在の日時から，ある一定期間までに変更された Linked Data を Item Index へ格納している．また，Linked Data の削除が確認されれば場合には，Removed Item Index へと特徴ベクトルの格納先が変更される．クローリング中に新規作成 Linked Data が発見された場合には，それが過去に削除された Linked Data の移動先ではないかを，Removed Item Index に格納されている特徴ベクトルと Levenshtein 距離等を用いて，特徴ベクトル同士の類似度を算出し，判断を行う．この時，比較の際の計算コストの削減のために，比較対象を最近追加されたものに

Vol.2013-DPS-156 No.2
Vol.2013-GN-89 No.2
Vol.2013-EIP-61 No.2
2013/9/11

表 1　DBpedia における移動，削除，追加の件数

SS3.2	SS3.3	移動	削除	追加
726,592	881,696	11,164	30,791	174,728

限定するというブロッキング手法を用いている．もし，削除された Linked Data と，新規作成 Linked Data の特徴の類似度が高い場合には，Move Detector がそれを Linked Data の移動と判断する．Move Detector によって，Linked Data の移動が発生している，ということが確認された場合には，その特徴ベクトルは Archived Item Index へと格納される．その後，ユーザの設定次第で，Linked Data の修正を行うアプリケーションなどに通知を発行することが可能である．

4. 関連研究の問題点

関連研究の問題点としては，比較対象を，最近追加されたものに限定するというブロッキング手法に問題があるといえる．DBpedia は過去のある時点のデータを Snapshot として公開 [10] しており，2008 年 10 月に作成された Snapshot3.2 と 2009 年 5 月に作成された Snapshot3.3 の比較に基づいて算出された移動，削除，追加の件数を**表 1** に示す．

この Snapshot を例にした場合，Snapshot3.2 の約 72 万件と Snapshot3.3 の約 88 万件の間では，約 17 万件の追加が起こっており，DSNotify のブロッキング手法を用いると，追加された約 17 万件を対象に探索することとなる．これは，Snapshot3.3 の全データ 88 万件を探索対象とする場合に比べると約 81%のコスト削減が見込まれる．しかし，**表 2** に示すように [11]，Linked Data は，2007 年には全データが 5 億であったのが，2011 年には 310 億件と急増しており，このようなブロッキング手法では，今後データの総数がより増加した場合，探索対象の最近追加されたものも比例して増加することが予想され，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 2　Linked Open Data におけるデータの推移

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

5. Linked Data におけるリンク修復の際の効率的なブロッキング手法の提案

本研究では，Linked Data の急激な増加にも対応可能な，効率的なブロッキング手法を提案する．これは，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，探索空間を縮小することを目的としている．以下に，本提案手法を示す．以下において，B，C を Linked Data とし，$L(B)$ を Linked Data B に含まれる URI の集合とする．また，h を一方向性ハッシュ関数，l を正の整数とし，$L = 2^l$ とする．そして，i を正の整数，$LSB(b,i)$ をバイナリ文字列 b の最下位 i ビットとする．

- 準備

  Linked Data B が与えられたとき，B の特徴ベクトル $char(B)$ を，以下の 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,$\cdots$,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 $char(B)$ をバリュー，Linked Data B の URI をキーとしてキーバリュー型データベース DB にストアする．

- 探索空間の決定

  Linked Data X，Y が与えられたとき，X と Y の非類似性（discrepancy）を次式により定義する．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．

$$discrepancy(X,Y) = support(char(X) \oplus char(Y)) \quad (1)$$

  Linked Data A と Linked Data B の間にリンクが張られてあり，Linked Data B が Linked Data B' へ移動したとき，B' を含む DB から探索空間 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 $B' \in DB$ に対し $discrepancy(B,B')$ を計算し，$discrepancy(B,B')/L \leq \theta$ ならば $S = S \cup B'$ とし，探索空間を構築する

6. 例を用いた説明

Linked Data B が**図 3** のようにリンクを 2 つ保持していると仮定する．また，$l = 4$，$L = 2^4 = 16$ として，$char(B)$ は 16 次元の特徴ベクトルとする．

特徴ベクトルの生成と保存

Linked Data B が保持している URI1 をハッシュ関

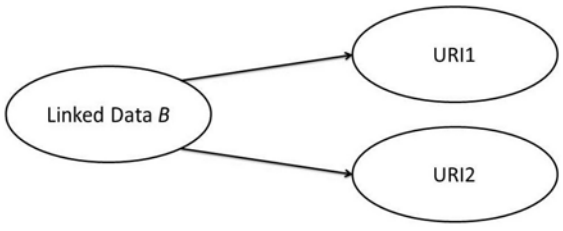

図 3　Linked Data B の保持しているリンク

Vol.2013-DPS-156 No.2
Vol.2013-GN-89 No.2
Vol.2013-EIP-61 No.2
2013/9/11

数にかけ，ビット列を取り出す．ここでは例として，”...0010 0110” というビット列が取り出されたと仮定する．l は 4 であるため，取り出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 ”0110” は 10 進数で 6 であるため，$char(B)$ の 6 ビット目を $0-1$ 反転する．さらに，Linked Data B が保持している URI2 をハッシュ関数にかけ，ビット列を取り出す．例として，”...1101 0010” というビット列を取り出したと仮定する．同様に，l は 4 であるため，下位 4 ビットを取り出す．取り出したビット列 ”0010” は 10 進数で 2 であるため，$char(B)$ の 2 ビット目を $0-1$ 反転する．最後に，生成した特徴ベクトル $char(B)$ をキー，Linked Data B の URI をバリューとしてデータベースにストアする．

探索空間の決定

Linked Data B の探索空間 S を Linked Data C_1，C_2，C_3 から求める．各 Linked Data の特徴ベクトルは**図 4** の様になっているものとする．

$$char(B) = (0, ..., 0, 1, 0, 1, 0, 1, 0)$$
$$char(C_1) = (0, ..., 1, 1, 0, 1, 0, 1, 1)$$
$$char(C_2) = (0, ..., 0, 1, 0, 1, 0, 1, 0)$$
$$char(C_3) = (0, ..., 1, 0, 1, 0, 1, 0, 1)$$

図 4 特徴ベクトルの例

Linked Data B と Linked Data C_1，C_2，C_3 に対し，特徴ベクトルの排他的論理和をとり，非零成分の個数を計算する．計算結果を**図 5** に示す．Linked Data B の特徴ベクトルと，Linked Data C_1 の特徴ベクトルの排他的論理和の非零成分の個数は 2 となる．同様に，Linked Data C_2，C_3 についても非零成分の個数を計算する．

$$supp(char(B) \oplus char(C_1)) = 2$$
$$supp(char(B) \oplus char(C_2)) = 0$$
$$supp(char(B) \oplus char(C_3)) = 7$$

図 5 Ljnked Data B との非零成分の個数の計算結果

計算した非零成分の個数を L で除算し，Linked Data B との非類似性を計算する．**図 6** に計算結果を示す．Linked Data B と Linked Data C_1 の非零成分の個数は 2 であるため，L で除算した値は 0.125 となり，非類似性は 0.125 となる．同様に，Linked Data C_2，C_3 について非類似性を計算する．ここで，θ を 0.2 とすると，Linked Data C_1，C_2 が閾値以下であるため，探索空間 $S = \{C_1, C_2\}$ となる．

$$discrepancy(B, C_1)/L = 0.125$$
$$discrepancy(B, C_2)/L = 0$$
$$discrepancy(B, C_3)/L = 0.4375$$

図 6 Linked Data B との非類似性の計算結果

表 3 実装環境

OS	Windows 7 Ultimate 64bit
CPU	Intel Core 2 Duo E8400 3.00GHz
メモリ	6GB RAM
使用言語	Java ”1.6.0_33”
RFD ストア	Virtuoso 6.4 [12]

7. 評価

本提案手法のアルゴリズムの評価を行うために，本アルゴリズムの実装を行った．実装環境を**表 3** に示す．

基本性能評価として，DBpedia の Snapshot 3.6 から 400 件の Linked Data を抽出し，Snapshot 3.7 から抽出した Linked Data 数を 2 万件から 10 万件に変化させ，本提案手法を適用した場合の実行時間を**表 4** に示す．表 4 において，特徴ベクトル生成時間，探索空間生成時間，合計実行時間は，Linked Data 1 件当たりの実行時間を示している．

表 4 Linked Data 数を変化させた実行時間

Linked Data 数	20,000	40,000	60,000	80,000	10,000
特徴ベクトル生成時間	0.648	0.779	1.114	1.982	2.632
探索空間生成時間	0.117	0.251	0.324	0.515	0.669
合計実行時間	0.765	1.03	1.438	2.497	3.301

同様の評価を，DSNotify に適用した場合の実行時間を**表 5** に示す．

表 5 DSNotify で Linked Data 数を変化させた実行時間

Linked Data 数	20,000	40,000	60,000	80,000	10,000
特徴ベクトル生成時間	0.252	0.386	0.506	0.765	1.482
リンク候補先決定時間	0.937	2.263	2.815	4.287	4.629
合計実行時間	1.189	2.649	3.321	5.052	6.111

評価結果を比較した場合，本提案手法では，特徴ベクトル生成時間が多く要しているものの，探索空間生成時間は，DSNotify のリンク候補先決定の処理に比べ，短くなっており，合計実行時間としては減少している．これは，本提

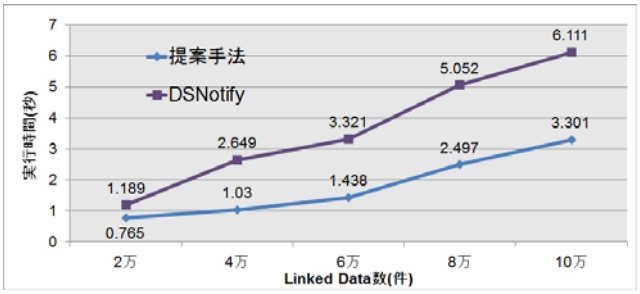

図 7 合計実行時間の推移

表 7 仮想的な Linked Data の移動例

	移動前	移動後
1	Brian_Ashton_%28rugby_player%29	Brian_Ashton_%28rugby_union%29
2	Thomas_Malthus	Thomas_Robert_Malthus
3	Wilhelm_Conrad_R%C3%B6ntgen	Wilhelm_R%C3%B6ntgen
4	Manuel_del_P%C3%B3pulo_Vicente_Garc%C3%ADa	Manuel_Garc%C3%ADa_%28tenor%29
5	Eli%C3%A1n_Gonz%C3%A1lez_affair	Elian_Gonzalez_affair
6	Suad_Husni	Soad_Hosny
7	Haewon%2C_Princess_of_Korea	Lee_Haewon
8	Richard_Redman_%28Bishop%29	Richard_Redman_%28bishop%29
9	Dorothy_Crowfoot_Hodgkin	Dorothy_Hodgkin
10	Juan_D%C3%ADaz_%28Mexican_American_boxer%29	Juan_D%C3%ADaz_%28boxer%29
11	Sophie_Buxhoeveden	Baroness_Sophie_Buxhoeveden
12	Morton_Barnett_Cohen	Morton_Cohen_%28politician%29
13	Rudy_Fernandez_%28actor%29	Rudy_%22Daboy%22_Fernandez
14	Vernon_Smith_%28football%29	Vernon_Smith_%28American_football%29
15	Jennifer_Jones_%28actor%29	Jennifer_Jones_%28actress%29

案手法の探索空間の生成の際には，ビット同士の排他的論理和をとっているのに対し，DSNotify では，文字の挿入，削除，置換をする Levenshtein 距離を用いて計算しているため実行時間の差が生じたものと考えられる．また，本提案手法では，特徴ベクトルを生成しデータベースに保存後は，Linked Data の変更がない限り，特徴ベクトルの更新の必要はないが，DSNotify においては，一定期間ごとに最近追加されたものを更新しなければならず，実際の利用を想定した場合の実行時間は，本提案手法の方がより少ないと予想される．

前述の合計実行時間をグラフに表したものを**図 7** に示す．このグラフが直線的に増えていくと仮定した場合には，Linked Data300 万件を対象にリンク修復を行った場合，Linked Data 1 件当たり 170 秒の実行時間の差が生じると予想される．

次に，本提案手法で生成された探索空間のサイズを**表 6** に示す．Linked Data 数が増加しても，1〜100 件の探索空間平均サイズは比例して増加しない．これは，本提案手法のブロッキングが，有効に作用しているものと考えられる．すなわち，DSNotify の探索空間は，Linked Data 数と共に比例して大きくなるが，本提案手法の探索空間のサイズは，Linked Data 数と比例しないという結果が得られた．

表 6 生成された探索空間のサイズ

Linked Data 数	20,000	40,000	60,000	80,000	10,000
1〜10 件	321	315	313	312	311
11〜20 件	5	10	6	2	2
21〜30 件	1	3	3	5	2
31〜40 件	0	1	6	3	4
41〜50 件	0	0	1	5	2
51〜60 件	0	0	0	1	3
61〜70 件	0	0	0	1	4
71〜80 件	3	0	0	0	0
81〜90 件	1	0	0	0	0
91〜100 件	3	0	0	0	0
101 件 〜	6	12	12	12	12
合計	340	341	341	341	340
1〜100 件の探索空間平均サイズ	26	13	19	25	31

次に，l や θ の値を変化させた場合の評価として，DBpedia の Snapshot 3.2 と 3.3 に対し，追加のデータを付与し，仮想的にリソースの削除，追加，変更を発生させたデータセット [13] を用いて，l と $L\theta$ の値を変化させた評価を行った．Snapshot 3.2 から無作為に 400 件の Linked Data を抽出し，Snapshot 3.3 から抽出した 2 万件の Linked Data に対し，172 件の仮想的なデータを生じさせたものを用いた．Snapshot 3.2 の平均トリプル数は 5.19 個であり，Snapshot 3.3 の平均トリプル数は 5.15 個であった．仮想的なデータの移動例を**表 7** に示す．

特徴ベクトルの生成時間と，探索空間の生成時間の評価結果を**表 8** に示す．この結果から，特徴ベクトルの次元が増加した場合，特徴ベクトルの生成時間はあまり増加しないが，探索空間生成時間が増えることが確認できた．次に，仮想的に移動させた 173 件の Linked Data に対し，探索空間を生成し，探索空間の中に正解が含まれていた割合を**表 9** に示す．

生成された探索空間には，72% 〜 89% の精度で正解が含まれているという評価結果が得られた．DSNotify に対して同様のデータセットで評価を行ったところ，精度は 70% であることから，本提案手法は，DSNotify と同程度の精度であることが確認できた．探索空間が適切に生成されなかったものとしては，少数のリンク先しか保持していない

表 8 特徴ベクトル生成時間と探索空間生成時間

l の値	特徴ベクトル生成時間 (秒)	探索空間生成時間 (秒)
$l = 8$	0.36	0.013
$l = 10$	0.364	0.032
$l = 12$	0.37	0.107

表 9　探索空間に正解が含まれていた割合

l の値	$L\theta = 1$	$L\theta = 2$	$L\theta = 3$
$l = 8$	0.728	0.809	0.89
$l = 10$	0.728	0.809	0.89
$l = 12$	0.728	0.809	0.89

Linked Data の移動や，大幅なリンクの変更があった場合などが挙げられる．また，本提案手法は，個々の LOD サイトで分散して実行することが可能であるため，スケーラビリティが高いと考えられる．

8. まとめと今後の課題

本研究では，今後 Linked Data の総数が増えた場合，関連研究である DSNotify のブロッキング手法では，有用性が損なわれることを問題点としてとらえ，この解決を図った．また，この問題点の解決のために，Linked Data の急激な増加にも対応可能な新たなブロッキング手法を提案した．また，その手法と関連研究の比較評価を行った．その結果，関連研究と比較して，40% ～ 50% の時間で実行でき，同程度の精度であることを確認した．

今後の課題として，保持リンクが少ない Linked Data への適切な探索空間の生成が挙げられる．保持リンクが少ない Linked Data では，抽出できるリンクの特徴が少ないため，本提案手法では適切に探索空間が生成できない．これには，Linked Data が保持しているリンクが少ない場合には，Levenshtein 距離により，探索空間を生成する手法が考えられる．この場合，データベースには特徴ベクトルと，Levenshtein 距離の計算に利用するリンクの URI を保存しなければならず，データベースに保存する情報が重複してしまう問題がある．また，Linked Data サイトによって Linked Data のリンクの保持数は大幅に変わるため，どの程度でリンクの保持が少ないと判断するかが課題となる．

参考文献

[1] SweoIG/TaskForces/CommunityProjects/LinkingOpenData http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData
[2] Linked Data - Connect Distributed Data across the Web http://linkeddata.org/
[3] Tim Berners-Lee: Design Issues: Linked Data, http://www.w3.org/DesignIssues/LinkedData.html, (2006).
[4] RDF/XML Syntax Specification (Revised) http://www.w3.org/TR/REC-rdf-syntax/
[5] N-Triples http://www.w3.org/2001/sw/RDFCore/ntriples/
[6] DCMI Metadata Terms http://dublincore.org/documents/dcmi-terms/
[7] FOAF Vocabulary Specification 0.98 http://xmlns.com/foaf/spec/
[8] 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).
[9] 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).
[10] DBpedia 3.7 Downloads http://wiki.dbpedia.org/Downloads37
[11] 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
[12] Orri Erling, Ivan Mikhailov: RDF Support in the Virtuoso DBMS, Studies in Computational Intelligence, Vol.221/2009, pp.7–24 (2009).
[13] The ISLab Instance Matching Benchmark http://islab.dico.unimi.it/iimb/