エンティティリンキング機能を有する 知識ベースと外部情報源の統合利用手法

大森　雄基† 　北川　博之†† 　天笠　俊之†††

† 筑波大学大学院 システム情報工学研究群　〒 305–8577 茨城県つくば市天王台 1 丁目 1–1
†† 筑波大学 国際統合睡眠医科学研究機構　〒 305–8577 茨城県つくば市天王台 1 丁目 1–1
††† 筑波大学 計算科学研究センター　〒 305–8577 茨城県つくば市天王台 1 丁目 1–1
E-mail: †omori.yuki.sm@alumni.tsukuba.ac.jp, ††kitagawa@cs.tsukuba.ac.jp, †††amagasa@cs.tsukuba.ac.jp

あらまし　知識ベースとは，主語・述語・目的語の組の集合である RDF 形式で人間の持つ知識を集積したものであり，様々な知識処理における重要な情報源として活用が進められつつある．しかし，気象情報や運行情報などの即時性が重要な情報や，詳細で専門的な知識など，知識ベース以外の情報源も多く，知識ベースと外部情報源の統合利用により多様な応用に対応することが期待される．一方で，外部情報源の入出力形式は多様であり，その出力結果に知識ベースのエンティティを選出する問題も存在するため，知識ベースと外部情報源の統合利用は一般に極めて煩雑となる．本研究では，SPARQL の Magic Properties を利用することで外部情報源を知識ベースと一体として活用できる統合利用環境のアーキテクチャを示す．またその際に重要となる，外部情報源が出力する情報に適切な知識ベースのエンティティを選出するエンティティリンキングについても議論する．

キーワード　知識グラフ, SPARQL, データ統合技術, エンティティリンキング, 問合せ処理

1 はじめに

近年，人類の保有する知識の増大ともに，これらの知識を構造化して保持する RDF 形式の知識ベースと，これに対する問合せ言語である SPARQL の利用環境が構築され，様々な知識処理における重要な情報源として活用が進められつつある．しかし，知識の全てが知識ベースに集積されているわけではない．例えば，気象情報や交通機関の運行情報といった即時性が重要な情報は静的な集積という形式はそぐわない．また，専門的な知識などを提供する知識ベース以外の情報源も多い．このため，これらの情報を有効活用するためには，知識ベースと非知識ベースの外部情報源の連携利用が重要となっている．

しかしながら，外部情報源は，問合せとその返り値に独自の形式を持つことが多く，この形式に対応して情報を解釈する必要がある．このため，知識ベースと外部情報源の連携には，外部情報源の多様な形式に対応しつつ，知識ベースと外部情報源の両方の結果を突き合わせるなど，極めて手間のかかる作業が必要となる．また，外部情報源からの得られるオブジェクトが，知識ベース中のどのエンティティにあたるかは，そのままでは不明であり，これを人手で判別するのも極めて大変な作業となる．例えば，外部情報源として地図情報を利用して特定の駅から近い教会を検索し，知識ベースの情報を利用して教派を絞り込みをする場合，地図上のオブジェクトと知識ベース上のエンティティの対応を一つ一つ検討し，集約する必要がある．このような問題に対応するには，知識ベースと外部情報源に対して，統合的に問合せが出来ることが重要となる．

これを受けて本研究では，図 1 のように，外部情報源にアクセスし，外部情報源をあたかも知識ベースと一体であるかのように扱える統合利用環境のアーキテクチャを提案する．

本研究のアーキテクチャは以下の 2 点である．

(1) Magic Properties を利用して外部情報源にアクセスし，取得した値を問合せ結果に組み込む．Magic Properties とは，知識ベース検索に対する問合せ言語 SPARQL において，様々な応用に対応した知識利用のために設定可能な独自の述語 [1] である．これは，もともとは SPARQL 問合せ中に応用に応じた計算手続きを組み込むための機能 [2] であるが，本研究では外部情報源へのアクセスに転用する．

(2) 外部情報源から得られるオブジェクトに対して，知識ベース中の適切なエンティティを選択する，エンティティリンキングの機能を有する．

なお，これ以降，Magic Properties を利用して外部情報源にアクセスする関数を呼び出す述語を，*External Source Predicate (ESP)* と呼ぶ．また，既存の知識ベース中の述語を *Knowledge Base Predicate (KBP)* と呼ぶ．

本提案手法により，例えば外部情報源である地図情報源を用いて，「つくば市の設立日と隣接する都市」を探す処理は，ESP `<udp:m:AdminDivNext>` を用いて以下のような SPARQL 問合せで記述できる．

```
SELECT DISTINCT * WHERE {
?x rdfs:label "Tsukuba"@en .
?x rdf:type dbo:City .
?x dbp:establishedDate ?date .
?x <udp:m:AdminDivNext> ?y.}
```

このとき エンティティ $?x$ から $?x \xrightarrow{\texttt{udp:m:AdminDivNext}} ?y$

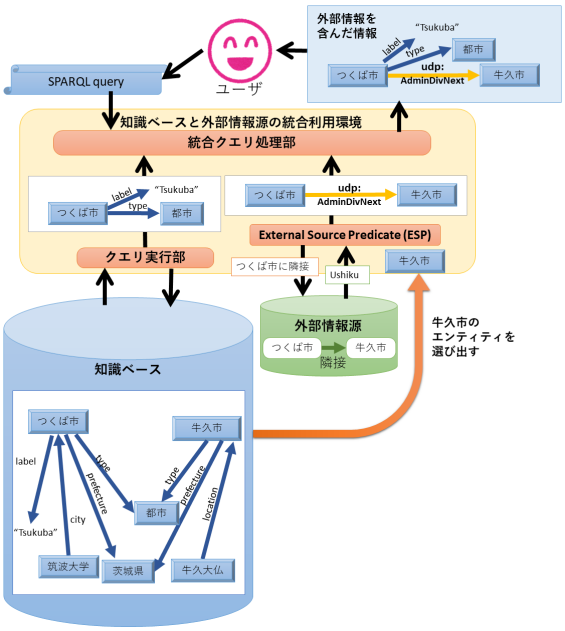

図 1　本研究が提案するアーキテクチャ

を満たすエンティティ $?y$ を探す処理は，ESP である `udp:m:AdminDivNext` が外部情報源の地図情報源に問合せて地図上でエンティティ $?x$ が対応するポリゴンが隣接するポリゴンをもつオブジェクトを検索し，そのオブジェクトを知識ベース中のエンティティ $?y$ にエンティティリンキングすることで実現する．

本研究の貢献は以下の通りである．

1．Magic Properties を利用した ESP が外部情報源にアクセスする関数を呼び出すことによって，エンティティ間の述語の拡張を行うアーキテクチャの提案と実装．

2．外部情報源から得られるオブジェクトに対応する知識ベース中のエンティティを探さなければならないという問題を認識し，外部情報源から得られるオブジェクトに対する複数の，エンティティリンキング手法の設計．

3．プロトタイプをシステムと複数の問合せを用いた作成し，複数のエンティティリンキング手法の精度評価

本論文の以降の部分は以下の構成である．第 2 章では関連研究を説明する．第 3 章では，この研究に関わる前提知識について述べる．第 4 章では，提案手法のアーキテクチャについて説明する．第 5 章では，本研究のエンティティリンキングの要件と，複数のエンティティリンキング手法について述べる．第 6 章では，実験について述べる．第 7 章では，本研究のまとめを述べる．

2　関 連 研 究

2.1　フェデレーション

RDF 形式の知識ベースは，複数のデータセット間で共通の語彙を使ったり，相互にリンクしているために，複数のエンドポイントに統合的に問合せをする需要がある．問合せ側がエンドポイントを指定する SERVICE 句を用いる手法 [3] や，SPARQL を用いて複数の知識ベースを統合的に検索するフェデレーション [4] という技術がある．

フェデレーションは複数のエンドポイントに問合せを行う．統計情報を使うもの [5] [6] [7]，RDF のデータセットに関するスキーマ定義である VoID [8] を使うもの [9]，ASK クエリを使うもの [9] [10]，複数の方式を組み合わせるもの [11] などがある．

本研究では知識ベースだけではなく，SPARQL で問合せできない外部情報源も利用するため，これに合わせたアーキテクチャを設計する．

2.2　メディエータ

意味的な情報を利用して異種情報源を統合するため，外部情報源に知識ベースに利用されるオントロジーを与える方式がある [12]．各外部情報源を取り込むために，様々な変換規則の置き方が提案されている [12]．

本研究は，たとえ同名であっても異なるエンティティを識別する問題を取り扱う必要があるため，オントロジーに対応する変換規則を定めただけでは問題を解決できない．このため，エンティティリンキングを応用して外部情報源のオブジェクトと知識ベース上のエンティティの対応関係をとる．

2.3　エンティティリンキング

エンティティリンキングとは，自然言語中の現実の物事を指し示す語句に対して，知識ベース中の同一の物事を指すエンティティを割り当てる行為である [13]．文章中からエンティティを割り当てるべき語句 (mention) を特定する Mention Detection と，mention に該当する可能性のあるエンティティを選出する Candidate Selection，そこからエンティティを絞り込む Linking Decision の段階に分けられる [14]．

Linking Decision に候補エンティティ同士の関連性を用いる方法としては，TAGME [15] がある．これは，Wikipedia のアンカーリンクの行先の重複度を利用してエンティティ間の関連の強さを計算する．また，DoSeR [16] はエンティティ間の関係をページランクで計算する．また，エンティティに付属する文書を利用した手法も存在する [17] [18]．また，エンティティ同士およびエンティティと語句の参照関係からエンティティと単語の埋め込みを学習する手法も存在する [19]．一方，近年はニューラルネットワークを用いた手法も盛んであり [14]，Yamada ら [20] は，BERT を用いて単語とエンティティを同時に入力とすることによるリンキングを行っている．

本研究では自然言語ではなく，外部情報源から得られるオブジェクトを知識ベースに統合するという問題を扱うため，これに合わせてエンティティリンキング法を設計する．

3　前 提 知 識

3.1　RDF

RDF とは，主語・述語・目的語のタプルとして，情報を記述したもの [21] [22] で，意味的な情報を記述することにも用いられる．主語には，現実の概念に対応する概念を IRI[1] として

1：URI を Unicode 文字で拡張したもの．

示したエンティティを持つ．目的語には，エンティティの IRI もしくは 数値や文字列などのリテラルを持つ．また，述語は，主語と目的語の関係を示す IRI を持つ．

例えば，Dbpedia 上で『筑波大学はつくば市にある。』を示す RDF 記述は以下のようになる．

dbr:University_of_Tsukuba $\xrightarrow{\text{dbo:city}}$ dbr:Tsukuba

3.2 SPARQL

RDF 形式の知識ベースの検索には，問合せ言語 SPARQL が通常用いられる．SPARQL 問合せは，RDF 記述に対応した知識グラフに対するパタンマッチングを行い，目的とする情報を取得する [23]．問合せの結果は変数に対応するタプルの集合であり，各変数を列名とする表として表記できる．

例として，以下の問合せを示す．

```
SELECT DISTINCT * WHERE
{?x rdfs:label “Jeffrey Ullman"@en . }
```

WHERE 以下は Where 句と呼ばれ，RDF に対するマッチングのパターンを示すグラフパターンを定めている．このパターンの最も基本的な部品は，上記に述べた RDF に対応する主語・述語・目的語である．主語は IRI もしくは空ノード，変数が入る．また目的語は IRI・空ノード・リテラル，そして変数をとる．また，述語は IRI と変数が入る．主語・述語・目的語の組を *Triple Pattern* と呼び，RDF のトリプルの不明部分を変数に変えたものに相当する．*Triple Pattern* の集合を，*Basic Graph Pattern (BGP)* と呼び，WHERE 句の基本的な中身となる．以上の問合せを実行すると，Jeffrey Ullman を指し示すエンティティが変数 *?x* の列に得られる．

3.3 Magic Properties

SPARQL 処理系には独自のユーザ定義述語を設定する機能がある [1] [24]．この機能は，知識ベースに対する様々な応用に知識利用する際，SPARQL 問合せ中に応用に応じた計算手続きを組み込むためのものである [2]．これを利用するには，少なくともユーザ定義の述語に対して，主語が決まった場合の目的語の算出方法，またはその逆が必要である．例えば図 2 は，既存のトリプルから生年月日を取得し，年齢を計算するユーザ定義述語を作る例である．udp:Age は年齢を指す Magic Property である．これがクエリ中に述語として記述された場合，年齢を計算するユーザ定義関数を呼び出す．ユーザ定義関数は，udp:Age の主語に当たる dbr:Jeffly_Ulkman を受け取り，

dbr:Jeffly_Ulkman $\xrightarrow{\text{dbo:birthDate}}$ 1945-11-23

というトリプルを取得することで，生年月日を取得し，現在の日付から 77 歳と計算できる．これを目的語の値として返す．

本研究では，この機能を用いて，外部情報源にアクセスして情報を取得し，知識ベースから得られる情報と統合する．

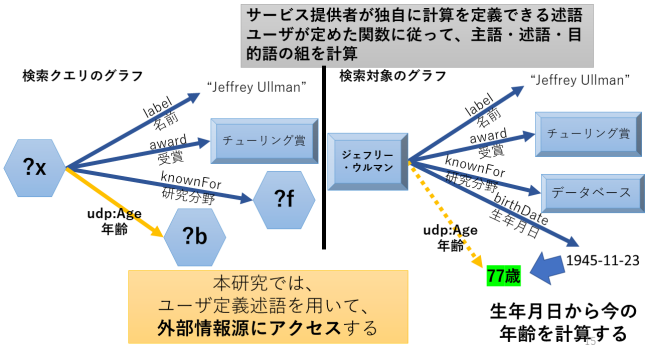

図 2　生年月日から年齢を計算する Magic Property の例

4 提案アーキテクチャ

4.1 拡張知識ベースの定義

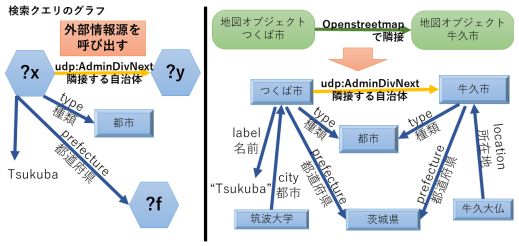

図 3　ESP を利用したクエリ例をグラフとして示したもの．左側が問合せのグラフ．右側が対応する知識ベース中の部分グラフと，ESP による拡張を表す．

本研究では，知識ベース中のエンティティ間の関係を，外部情報源から取得した関係を示す述語 *External Source Predicate (ESP)* で拡張する．

図 3 は，ESP が知識グラフを拡張することを示した図である．知識ベース中には，dbr:Tsukuba や dbr:Ushiku など様々なエンティティやリテラルが，知識ベースの述語である *Knowledge Base Predicate (KBP)* で接続されている．このとき，知識グラフを拡張するための外部情報源の例として，例えば，Openstreetmap (OSM) を利用することを考える．OSM は，オープンライセンスの地図アプリケーションであり，地図としての閲覧が可能であるのみならず，地図上の地物が xml 形式でまとめられている．OSM の地物には，その座標やポリゴン・地物間の関係を示すデータと，属性を定めるための Tag が存在している．このため，OSM 上でつくば市と牛久市が隣接しているという情報も取得が可能である．これを利用して，OSM 上で自治体のポリゴンが隣接していることを示す KBP である udp:AdminDivNext で dbr:Tsukuba と dbr:Ushiku を接続する．以上のようにして，外部情報源を用いて知識ベースを拡張することが可能である．

4.2 全体のアーキテクチャ

提案手法のアーキテクチャの概要は図 1 に示した通りである．ユーザは，通常の SPARQL 問合せに本アーキテクチャが提供する ESP を加えることで，通常の SPARQL で知識ベースを検索したときと同様に，前節で示した拡張知識ベースがあたかも存在するかのように問合せることが出来る．このため，知識ベースと外部情報源を統合的に検索可能である．

第 1 章で示した問合せ例では，<udp:m:AdminDivNext> が知識ベース中には存在しない述語である ESP となっている．すなわち，知識ベース中にはこの述語に対応するトリプルは実在せず，主語に対応するエンティティから目的語に対応するエンティティを求めるのは外部情報源を呼び出すことによって行われる．このとき，知識ベース DBpedia と OSM を用いて『つくば市の設立日と隣接する都市を探したい』という問合せを考える．このとき，つくば市の設立日は KBP である dbp:establishedDate を用いて知識ベースから取得できる．一方で，OSM 上で外接する地物が隣接する自治体であることは ESP である udp:m:AdminDivNext を用いて取得する．したがって，この問合せは以下のように既述可能である．

```
SELECT DISTINCT * WHERE {
?x rdfs:label "Tsukuba"@en .
?x rdf:type dbo:City .
?x dbp:establishedDate ?date .
?x <udp:m:AdminDivNext> ?y.}
```

ESP は，主語の「つくば市」エンティティから，OSM へ問合せる関数を呼び出し，返ってきた自治体名を知識ベース DBpedia 上のエンティティと照らし合わせて該当するエンティティを返さなければならない．このとき，ESP によって，DBpedia 上には存在しないトリプル

$$\texttt{dbr:Tsukuba} \xrightarrow{\texttt{udp:m:AdminDivNext}} \texttt{dbr:Ushiku,}_{\texttt{I}}\texttt{baraki}$$

が与えられる．

なお，本論文の以下では，議論の簡単化のため，この例のように ESP の目的語はエンティティであるものと仮定するが，それ以外の場合への拡張は容易である．

このアーキテクチャを実現するため，4 つの課題が存在する．

a） ESP の登録方法

ESP を登録するにあたり，外部情報源の多様性に対応しつつ，ユーザの負担を最小化しなければならない．

b） 拡張した SPARQL 問合せの実行

ESP によって拡張した知識ベースに対しての SPARQL 問合せのアルゴリズムを開発する必要がある．当然，問合せ中には ESP が複数含まれ得るため，これを想定したアルゴリズムを設計しなければならない．

c） 外部情報源のオブジェクトとエンティティの対応

ESP が得た外部情報源に対する問合せ結果のオブジェクトと，知識ベース中のエンティティの対応は必ずしも自明ではない．例えば，外部情報源が”Fuchū”という文字列を返した場合，東京都の府中市を指していたとしても，同名の地名は広島のなど全国にあるため，同名のエンティティは多数存在する．ここから，dbr:Fuchū,_Tokyo を選ばなければならない．また，略称・別名など，エンティティの名前と外部情報源の示した文字列が必ず一致するとは限らない．このため，ESP が得た外部情報源から得られるオブジェクトに対応する知識ベース中のエンティティを識別するのは容易ではない．

4.3 ESP の登録方法

アーキテクチャにおいて，(1) 主語エンティティから外部情報源へアクセスに用いるキーを生成，(2) そのキーを用いて外部情報源にアクセス，(3) 問合せ結果として得られたオブジェクトに対応する知識ベース中のエンティティを探し出して目的語のエンティティとする必要がある．図 4 は (1),(2),(3) の概要図である．各 ESP における (1)(2) の処理は以下に示すように，ユーザが定義，登録するものとする．よって，ESP の登録時には以下の事が必要となる．

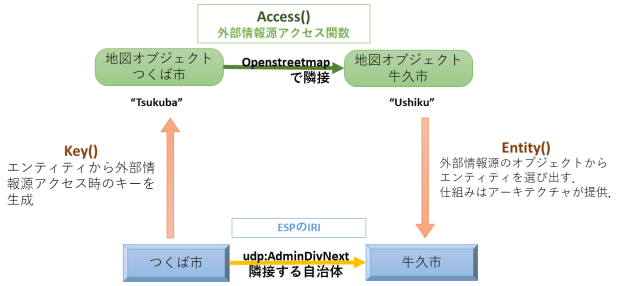

図 4 ESP 処理の概要図．外部情報源へアクセスする関数 $Access()$ と，ESP の IRI の登録だけは必須である．また，主語エンティティから外部情報源アクセス時の生成する関数 $Key()$ も登録する必要があるが，特定の述語の値を利用する場合はその述語を指定するだけでよい．

a） ESP の IRI

ESP の IRI を登録する．例えば ESP の IRI として udp:m:AdminDivNext を登録し，問合せ中にこの ESP がある場合，以下に示す問合せ処理では，以下の c) に示す外部情報源アクセス関数を呼び出す．

b） エンティティ x から外部情報源アクセス時のキーを生成する関数 $Key()$

上記の (1) の処理を行う関数（プログラムコード）を登録する．エンテイティの名称や座標等を取得し，外部情報源検索のキーとすることが考えられる．上記の例では，主語エンティティ x =dbr:Tsukuba から，rdfs:label を取得することで $Key(x) = \{$”Tsukuba”$\}$ を得る．このように，単純に主語エンティティに対して特定の述語の目的語を $Key(x)$ の値とする場合には，述語を指定するだけでよい．

c） 外部情報源アクセス関数 $Access()$

b) で定義した $Key(x)$ により得られるキー key_x を用いて外部情報源をアクセスする関数（プログラムコード）を登録する．すなわち，$Access(Key(x))$ は，主語エンティティ x から ESP の目的語を外部情報源アクセスにより取得する操作に対応し，$R = Access(Key(x))$ とすると，R は ESP の目的語のエンティティに対応する外部情報源が出力するオブジェクトの集合である．上記の例では，$key_x =$ ”Tsukuba” に対して，OSM に問合せて隣接する自治体名の集合である $R =$ { ”Tsuchiura”, ”Ushiku”... } を返す関数である．

以上 3 項目を登録すれば，本アーキテクチャは問合せにおいて ESP の利用が可能となる．$R = Access(Key(x))$ のオブジェクトと知識ベース上のエンティティとのマッチング $Entity()$ については，アーキテクチャが提供する．この詳細は 4.5 節と第 5 章で述べる．

4.4 拡張した SPARQL 問合せの実行

本研究における拡張 SPARQL 問合せは，従来の SPARQL 問合せに ESP を加えたものである．SPARQL の Where 句を表す BPG においても ESP を加えたものを取り扱う．BPG は Triple Patterns の集合である．Triple Pattern は 主語, 述語, 目的語から構成される．BPG はラベル付き有向グラフである．BPG は変数，エンティティ，リテラルを頂点とする．また，辺は KBP で構成される．

ここで，問合せから ESP を除外した時に，残るそれぞれの連結部分グラフを *Segment* と呼ぶ．図 5 は，ESP を含む SPARQL 問合せが，ESP を除外することで *Segment* に分割されることを示した図である．

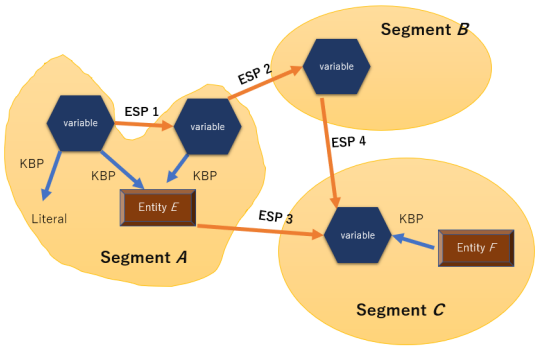

図 5　問合せ ESP を除いた時に非連結になる部分を Segment とする．

Segment は，変数，KBP を辺とする．また，*Segment* 同士は ESP で接続されている．つまり，各 ESP は主語側と 目的語側にそれぞれ *Segment* を持つ．

図 5 に *Segment* の例を示した．*Segment A* は，2 つの変数，1 つのエンティティを頂点 とし，3 つの KBP を辺として持つ．ESP 1 は 主語側も目的語側も *Segment A* 内の頂点であり，*Segment A* 自身を接続している．*Segment B* は変数を 1 つだけを頂点として持つ．また，ESP 2 は 主語側に *Segment A* の変数，目的語側 に *B* の変数を持つため，主語側を *Segment A* ，目的語側を *B* として接続していることになる．同様に，ESP 3 は 主語側を *Segment A* ，目的語側を *C* として両 *Segment* を接続している．ESP 4 は　主語側を *Segment B*，目的語側を *C* として接続している．

Segment 内部は通常の SPARQL クエリと同等の BPG であるため，直接知識ベースに対応する SPAQL 問合せを発行してその結果を得ることが可能である．ただし，各 *Segment* に対する問合せ結果を表す表は，変数のみでなくエンティティを表す頂点に対応する列も持つものとし，列名と中身の値はそのエンティティの IRI とする．例えば，*Segment A* が，変数 $?x$, $?y$, エンティティ E を持つとき，*Segment A* の問合せ結果は $?x$, $?y$, E を列名とする表として表記する．

ESP の主語エンティティを x とすると，ユーザが登録した $Key()$, $Access()$ を用いて，外部情報源のアクセス結果，$Access(Key(x))$ が得られる．また，この要素オブジェクトを $r \in R$ に対して，4.5 に示すエンティティリンキングにより，ESP の目的語エンティティを $y = Entity(r)$ を得る．

これらの処理を含む問合せアルゴリズムを設計する上では以下の点に留意しなければならない．

- ESP の $Key()$, $Access()$ を実行する際には，必ず主語のエンティティ x が特定されている必要がある．
- ESP の $Access()$ や $Entity()$ を実行する際には，外部情報源に対するアクセスとエンティティリンキングが行われるため，待ち時間や少なくない計算時間が発生する．このため，ESP の関数の実行回数はできるだけ少なくなることが望ましい．
- ESP の $Key()$, $Access()$ ならびに $Entity()$ を実行後には，ESP に関わるトリプルが判明しているために，KBP と同等の扱いをしてもよい．

以上を留意した問合せ処理の手順は以下による．

`Step 1.` Triple Pattern を含む *Segment* について全て知識ベースに問合せを行い，その結果を中間結果表の集合とする．

`Step 2.` 各 ESP の主語側の中間結果表に基づき，ESP の主語にあたる異なるエンティティの個数を数え，その数がもっとも少ない ESP を選択する．最小の ESP が複数ある場合は，ランダムに 1 個選択する．

`Step 3.` 主語側の中間結果表から主語エンティティの集合を抽出し，各エンティティを主語として順次 ESP を評価する．すなわち，$Key()$, $Access()$, $Entity()$ を実行する．結果を中間結果表へ自然結合する．

`Step 4.` 全ての ESP の処理が終わったかを確認し，未処理の ESP が存在する場合は 2 へ戻る．

`Step 5.` Step 4 までで得られた中間結果表から問合せの最終結果を出力する．

ここで，Step 3 についてより詳しく説明する．Step3 では，外部情報源にアクセスを行うことで，ESP に対応する主語・目的語の組の表を得る．この ESP のアクセス結果は繋がりのある中間結果表と join することが可能となる．この点を考慮に入れた Step 3 の詳細を以下に示す．また処理の概要を図**??**に示す．

`Step 3-1.` 中間結果表から主語エンティティ集合を取得する．ESP の主語が変数の場合は，エンティティ集合を取得する．また，主語がエンティティの場合は，そのエンティティを取得する．

`Step 3-2.` 主語エンティティ x を外部情報源に対するキー $Key(x)$ に変換する．

`Step 3-3.` 外部情報源にアクセスし，目的語に対応するオブジェクトの集合 $R = Access(Key(x))$ を取得する．

`Step 3-4.` 各外部情報源オブジェクト $r \in R$ について，適切なエンティティ $y = Entity(r)$ を割り当てる．ただし，適切なエンテイティが発見できない場合は，存在しないと判定する．この Step の詳細は次の節に述べる．

`Step 3-5.` 主語側および目的語側の中間結果表を自然結合して，中間結果表の集合を更新する．

4.5 外部情報源のオブジェクトとエンティティの対応

$R = Access(Key(x))$ は，対応する外部情報源オブジェクトの集合であるため，それぞれの $r \in R$ に対して知識ベース

中のエンティティを探し出して，ESP の目的語エンティティ $y = Entity(r)$ として返す必要がある．また，対応するエンティティが知識ベース中に無い場合，存在しないと判定する必要がある．

これは以下の手順による．

`Step 3-4-1`． 外部情報源オブジェクト $r \in R$ に対する候補エンティティ集合 $C' = CS(m)$ を選出する．この Step を Candidate Selection と呼ぶ．

`Step 3-4-2`． もし，ESP の目的語側がエンティティであるか，Triple Pattern に含まれる変数の場合，目的語側の中間結果表から ESP の目的語となり得るエンティティの集合（Allow set と呼ぶ） A を取得し，$C = C' \cap A$ とする．また，目的語側が Triple Pattern に含まれない変数の場合は，$C = C'$ とする．

`Step 3-4-3`． 主語エンティティ x と候補エンティティ $c \in C$ との関連から，各外部情報源オブジェクトに対する割当エンティティ $y = LD(C, x)$ を出力する．あるいは，割当るべきエンティティは存在しないと判定する．この Step を Linking Decision と呼ぶ．

本研究では 2 種類の Candidate Selection $CS()$，2 種類の Linking Decision $LD()$ を用いる．この詳細は第 5 章 に示す．

5 エンティティリンキング

外部情報源から取得する $r \in R$ は外部情報源中のオブジェクトであるため，これが知識ベース上のどのエンティティにあたるかを判定する必要がある．本研究では，外部情報源から得られる外部情報源オブジェクト r は文字列であると仮定する．

第 4.5 節で説明した通り，外部情報源の文字列とエンティティの対応関係をとるアルゴリズムは 3Step からなる．このうち，Step 3-4-1 の Candidate Selection (CS) と，Step 3-4-3 の Linking Decision (LD) について，それぞれ 2 つの手法を提案する．

5.1 Candidate Selection (CS)

知識ベース中の全てのエンティティに対して割当エンティティに該当するかを確認することは，計算コスト上望ましくない．そこで，外部情報源から取得した文字列 $r \in R$ から文字列を使ったマッチングを行い，候補エンティティの集合 $C' = CS(r)$ を得る．以下，2 つのアルゴリズムを用意した．

a） DictCS

事前に用意した語句と候補エンティティとの対応辞書を用いる．この辞書の構築には，Wikipedia から辞書を構築する手法を含む研究 [13] [25] [26] を参考に，英語版 Wikipedia から収集した記事名とアンカーリンク情報を利用する．アンカーリンクは，Wikipedia の文書中の語句から Wikipedia ページへ飛ぶリンクであり，語句と Wikipedia ページの対応関係が得られる．知識ベース DBpedia のエンティティと Wikipedia ページは一対一対応となっており，Wikipedia ページの文字列変換で DBpedia エンティティの IRI を得られる．辞書検索時に先頭一致，もしくは辞書側の先頭の 1,2 単語を除いた先頭一致を行う．

例えば，Wikipedia 中の語句 ”University of Tsukuba” から，`https://en.wikipedia.org/wiki/University_of_Tsukuba` へ飛ぶアンカーリンクが存在する．このとき，対応する DBpedia の IRI は `dbr:University_of_Tsukuba` となる．このため，辞書には

”University of Tsukuba” → `dbr:University_of_Tsukuba`

と登録する．当然，表などで，”Tsukuba”と略されることもあるため，

”Tsukuba” → `dbr:University_of_Tsukuba`

も登録される．また，つくば市を示す `dbr:Tsukuba` も，”Tsukuba”のアンカーリンクで参照されるため，

”Tsukuba” → `dbr:Tsukuba` も登録される．

もし，外部情報源が r = ”University of Tsukuba”という文字列であったとき，DictCS は，辞書を”University of Tsukuba”，”of Tsukuba”，”Tsukuba”で検索する．このため，`dbr:University_of_Tsukuba`，`dbr:Tsukuba` が候補エンティティ集合として返される．

外部情報源から取得した文字列 r が，

b） LabelCS

`rdfs:label` 中の単語列に対して，単語単位で部分一致するエンティティを候補とする．

例えば，外部情報源が r = ”Tsukuba”という文字列であったとき，

`dbr:University_of_Tsukuba` $\xrightarrow{\texttt{rdfs:label}}$ `University of Tsukuba`

`dbr:Tsukuba` $\xrightarrow{\texttt{rdfs:label}}$ ”Tsukuba”

が合致する．しかし，

`dbr:Tsukubamirai` $\xrightarrow{\texttt{rdfs:label}}$ ”Tsukubamirai”

は単語としては一致しないために，候補としない．

5.2 Linking Decision (LD)

候補エンティティ集合 C から，割当エンティティ $y = LD(C, x)$ を返す．あるいは該当無しと判定する．

候補エンティティ集合 C のうち，外部情報源の返した値 r に対応するエンティティ $y \in C$ は，ESP と共に新たなトリプルを構成する．このため，ESP の主語エンティティ x とは，何らかの関係があると期待される．このため，LD では主語エンティティ x と 候補エンティティ $c \in C$ の関係を知識ベース上から取得し，もっとも関係が強いエンティティを返す．この考えのもと，以下の 2 つのアルゴリズムを示す．

a） GraphVoteLD

この方法は，TAGME [15] を参考にした方法である．2 つのエンティティ t, c の関連度 $rel(t, c)$ を定義する．まず，t,c に相当する Wikipedia ページをそれぞれ w_t, w_c とする．次に，w_t, w_c からアンカーリンクをたどって到達できるページの集合を W_t, W_c とする．このとき，$rel(t, c) = |W_t \cap W_c|$ とする．つまり，w_t, w_c から共通にたどれるページの数として $rel(t, c)$ を定義する．

また，主語エンティティ x から知識ベース上で任意のをたどって得られる目的語エンティティの集合に，x 自身を加えたもの

を文脈エンティティ集合 $ctx(x)$ とする．

全ての候補エンティティ c に対して，x の文脈エンティティ集合 $t \in ctx(x)$ との関連度 $rel(t,c)$ を計算し，合計が最大であるものを y とする．すなわち，$y = \arg\max_{c} \sum_{t \in ctx(x)} rel(t,c)$ とする．ただし，この合計が閾値以下のとき該当する y は無しとする．

b） AbstractCosSimLD

主語エンティティ x の `dbo:abstract` の値，および $c \in C$ の `dbo:abstract` の値の単語集合の Cosine 類似度を計算し，最大であるものを y とする．

6 実　　験

6.1 実験環境

本研究のアーキテクチャは，外部情報源から取得したオブジェクト r に対し，エンティティリンキングを用いて該当する知識ベース中のエンティティ y を選ぶという構造上，問合せに対する実行結果は曖昧性を持つこととなる．このため，エンティティリンキングを組み込んだアルゴリズムが，問合せに対してどの程度正確な問合せ実行を行うのかを検証する．また，併せて実際に 4.4 に示したアルゴリズムで問合せを実行できるかも検証するため，アーキテクチャのプロトタイプを作成し，ESP を登録したうえで，問合せを実行する実験を行った．

実験に使用する知識ベースは英語版 DBpedia の 2020 年 6 月のスナップショットである．これは英語版 Wikipedia を RDF 形式に変換したものである [27]．また，外部情報源には地図サービス OpenStreetMap (OSM) を用いる．

ESP は，OSM の検索サービスである Nominatim と Overpass を，OSM node の英語名を基に検索し，エンティティリンキングを行って目的語のエンティティを返す．例えば，`udp:m:AdminDivNext` の主語エンティティ $x =$ `dbr:Tsukuba` から，`rdfs:label` をたどって文字列 "Tsukuba" $= Key(x)$ を得る．"Tsukuba"をキーとして Nominatim と Overpass に問合せると，つくば市と境界線を共有する OSM オブジェクトの情報を含んだ xml が返ってくるため，ここから隣接自治体名の文字列の集合を抜き出して外部情報源の返す値 $R = Access(Key(x))$ とする．この値の一つである "Ushiku" $= r \in R$ に対して，エンティティリンキングを行うことで，`dbr:Ushiku,_Ibaraki` $= y = Entity(r)$ を得る．これにより，

`dbr:Tsukuba` $\xrightarrow{\texttt{udp:m:AdminDivNext}}$ `Ushiku,_Ibaraki`

というトリプルを構築し，知識ベースを拡張する．

本研究では以下 4 つの ESP を用意した．

a） `<udp:m:AdminDivNext>`

主語のエンティティに隣接する行政区画のエンティティを目的語とする．隣接自治体の情報は DBpedia では多くないため，この ESP の導入によって知識ベースを拡張する．

b） `<udp:m:Dist1km>`, `<udp:m:Dist5km>`, `<udp:m:Dist10km>`

座標付きエンティティに対し，その座標から 1km, 5km 10km 以内にある OSM node を検索し，その英語名を基に目的語のエンティティを返す．一般的に地理的なエンティティであっても，DBpedia 中の全てのエンティティに座標情報がついているわけではないので，距離を扱う際には地図サービスに問合せることが望ましい．この ESP によって，知識ベースを拡張する．

この環境のもと，異なる性質の BPG をもつ 8 つの拡張 SPARQL 問合せを実行し，その結果の妥当性を検証した．8 つの問合せの正解は著者が作成した．提案手法の問合せ結果と正解データの表を，行の集合として扱い，Recall, Precision, F1 を算出した．

6.2 ベースライン手法

提案手法と同等の機能を有する既存の手法は存在しないが，エンティティリンキングの性能比較のため，外部情報源の返した文字列 r を英語版 wikipedia で検索した場合の結果得られるエンテイティを ESP の目的語とする手法（enWikiSearch と呼ぶ）をベースライン手法とした．この実装には，Python の"wikipedia"パッケージを利用した．

6.3 実験結果

表 1　各問合せと，各 CS, LD 及びベースライン (Wikipedia) の Recall (R), Precision (P), F1．"-"はエンティティを選出しなかったことを表す．平均の計算からは除外する．

CS	DictCS						LabelCS								
LD	Graph VoteLD			Abstract CosSimLD			Graph VoteLD			Abstract CosSimLD			enWikiSearch		
	R	P	F1	R	P	F1	R	P	F1	R	P	F1	R	P	F1
Q1	**1.0**	**1.0**	**1.0**	0.89	1.0	0.94	**1.0**	**1.0**	**1.0**	0.89	1.0	0.94	0.33	0.33	0.33
Q2	**1.0**	**1.0**	**1.0**	0.89	1.0	0.94	**1.0**	**1.0**	**1.0**	0.89	1.0	0.94	0.33	1.0	0.5
Q3	0.67	0.29	0.4	0.67	0.13	0.22	**0.67**	**0.33**	**0.44**	0.67	0.2	0.31	-	-	-
Q4	**0.75**	**1.0**	**0.86**	0.75	0.75	0.75	**0.75**	**1.0**	**0.86**	**0.75**	**1.0**	**0.86**	-	-	-
Q5	**0.86**	**0.86**	**0.86**	0.86	0.4	0.55	0.43	0.75	0.55	0.43	0.38	0.4	-	-	-
Q6	1.0	0.67	0.8	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	0.5	1.0	0.67
Q7	0.93	0.46	0.62	0.64	0.53	0.58	**0.93**	**0.72**	**0.81**	0.64	0.64	0.64	-	-	-
Q8	**0.93**	**0.65**	**0.76**	0.64	0.5	0.56	0.93	0.62	0.74	0.64	0.53	0.58	-	-	-
Avl.	0.89	0.74	0.79	0.79	0.66	0.69	0.84	**0.80**	**0.80**	0.74	0.72	0.71	0.39	0.78	0.50

実験結果を Table 1 に示す．実験結果の評価は，正解の各問合せの結果と，統合環境の問合せ結果を行の集合と見なし，Recall, Precision, F1 を算出したものである．なお，Recall, Precision, F1 が空欄の部分は，統合環境が，全ての $r \in R$ について，エンティティが無いと判定したことを示す．

提案手法は，全ての問合せについて，ベースライン手法を上回った．また，Query1, 2 では LD に GraphVoteLD を用いた場合が全て正解となった．また，Query5, 8 以外は，LabelCS, GraphVoteLD とした場合の性能が最も良く，Query8 についても僅差であった．しかし，Query5 では半分以上の行を見つけることが出来なかった．

6.4 考　　察

前述の通り，全ての CS と LD の組み合わせでベースライン手法を上回っている．ベースライン手法の enWikiSearch 方式は英語版 Wikipedia の検索ウィンドウで検索したものを自動化した場合に相当するが，結果は入力キーワードのみに依存するため，キーワードが多義的であった場合に正しいエンティティを選出できない．一方で，エンティティリンキングを組み込んだ提案手法は，グラフ関係や Abstract 文書との比較から，ある

程度正しいエンティティを選ぶことが出来る．ただし，Query3では，Precisionが極めて低い．Query3は,「つくば市の代表点（市役所）から5kmの駅」を問合せるものである．これに対し，Openstreetmapは361件のnodeの名前を返したが， このうち駅は4件である．このため，偶然無関係な建物と名前が一致した駅が紛れてしまったことにより，不正解の行が数多く発生した．このことから，外部情報源が最終的な問合せ結果に対応しないオブジェクトを大量に返した場合は，性能が低下すると考えられる．

表2　CSの性能比較．全ての候補エンティティを返し，LDを行わない場合の結果.

CS	DictCS			LabelCS		
	Recall	Precision	F1	Recall	Precision	F1
Q1	1.00	0.16	0.28	1.00	0.13	0.22
Q2	1.00	1.00	1.00	1.00	1.00	1.00
Q3	1.00	0.04	0.08	0.67	0.04	0.08
Q4	1.00	0.80	0.89	0.75	1.00	0.86
Q5	0.86	0.17	0.29	0.43	0.13	0.20
Q6	1.00	0.40	0.57	1.00	1.00	1.00
Q7	0.93	0.23	0.37	0.93	0.72	0.81
Q8	0.93	0.22	0.35	0.93	0.62	0.74

Candidate Selection (CS) を軸に見ていくと，多くの組み合わせでLabelCS方式が上回っている．また，Precisionだけを見ても上回る場合が多い．DictCS方式は辞書の作成時にWikipedia中のアンカーリンク元の文字列を利用しているため，表記ゆれや略称等に対応しやすい．一方で，LabelCS方式は，`rdfs:label`の文字列を用いた辞書であるため，略称等は含まない場合が多い．このため，一般的に言えば，LabelCS方式よりもDictCSの方が多くのエンティティを選出できると考えられ，Recallも向上できると期待できる．表2は，CSのみを行い，LDを行わずに全ての候補エンティティを結果として出力した場合のRecall, Precision, F1を示したものである．ただし，目的語側 の中間結果表の制約は受けるものとする．表2より，DictCSが発見した正解のエンティティはLabelCSに比べ多いものの，半数以上の問合せで同数であった．一方で，DictCSはLabelCSに比べ間違ったエンティティを含んでしまう傾向がある．また，Query5ではLabelCSは正解のエンティティを半分以上発見できなかった一方，DictCSは多くの正解のエンティティを発見した．本実験では，外部情報源にOSM nodeの英語名を利用しているという性質上，略語の利用は少ないと考えられる．略語や表記ゆれの探索による候補エンティティの増加は，有効なエンティティの選出に寄与せず，表2のようにQuery5以外ではむしろ不要なエンティティを増加させた．このため，DictCSは性能を下げた可能性がある．

Linking Decision (LD) を軸に見ていくと，概ねGraphVoteLD方式の性能が高い．今回扱ったエンティティは，都市名・駅名が多い．このため，知識ベース上のトリプルはある程度の数があるものの，概要文書は貧弱である場合が少なくない．例えば，GraphVoteLDでリンキングに成功した例の一つでは，つくばみらい市 (`dbr:Tsukubamirai,_Ibaraki`) を主語とするトリプルは236個存在する．しかし，知識ベース中の`dbo:abstract`中にある概要文書は，下に示した通り，100語以下であり，人口・人口密度・面積といった記述で埋まっている．このような傾向から，AbstractCosSimLDが有効に働かなかった場面があったと考えられる．逆に，トリプル数が少ないが概要文書はある程度書かれている事が期待されるエンティティを対象とする場合には，概要文書を利用する方式が有効になる可能性がある．

dbr:Tsukubamirai,_Ibaraki の概要文書

Tsukubamirai (つくばみらい市, Tsukubamirai-shi) is a city located in Ibaraki Prefecture, Japan. As of 1 October 2020, the city had an estimated population of 51,035 in 20,030 households and a population density of 645 persons per km². The percentage of the population aged over 65 was 26.3%. The total area of the city is 79.16 square kilometres (30.56 sq mi).

7 まとめ

RDF形式の知識ベースは，静的に意味的情報を保持しており，SPARQLで問合せできる．一方，非知識ベースの外部情報源は，問合せとその返り値に独自の形式を持つことが多い．このため本研究では，知識ベースと非知識ベース外部情報源の統合利用アーキテクチャを提案した．このアーキテクチャは，SPARQLにおけるMagic Propertiesを利用し，外部情報源にアクセスする関数を呼び出すことで，外部情報源の情報を取り入れ，エンティティ間の関係を拡張可能である．また，外部情報源から取得したオブジェクトと知識ベース上のエンティティの対応を求めることが容易でないという問題に対して，エンティティリンキングを自動的に行う機能を提供する．また，提案アーキテクチャのプロトタイプを実装し，実際に拡張SPARQL問合せを実行することで，アーキテクチャの有効性を確認した．また複数のエンティティリンキングを実行し，エンティティリンキングの必要性と性質の違いを明らかにした．一方，外部情報源が返した値の大部分が最終的な問合せ結果に適合しない場合は，性能が下がるという問題を発見した．

今後の課題は，より多様な外部情報源とESPを用いた検証，より高度なエンティティリンキング方式の設計，ユーザ定義のエンティティリンキング方式の組み込み等である．また，知識ベースに存在しないエンティティを取り込む知識グラフの拡張も重要な課題である．

謝　辞

本研究の一部は，JSPS科研費 (JP19H04114, JP22K19802, JP22H03694)，JST CREST (JP-MJCR22M2)，NEDO (JPNP20006), AMED (JP21zf0127005) による．

文　献

[1] 3.3 Magic Properties, SPIN - Modeling Vocabulary, W3C Member Submission 22 February 2011. https://www.w3.org/Submission/spin-modeling/spin-magic.
[2] Feature:JavaScriptFunctions. W3C. https://www.w3.org/2009/sparql/wiki

/Feature:JavaScriptFunctions.
[3] Carlos Buil-Aranda et al. Semantics and Optimization of the SPARQL 1.1 Federation Extension. In *The Semanic Web: Research and Applications*, 2011.
[4] Muhammad Saleem et al. A fine-grained evaluation of SPARQL endpoint federation systems. *Semantic Web*, 2016.
[5] Bastian Quilitz and Ulf Leser. Querying Distributed RDF Data Sources with SPARQL. In *The Semantic Web: Research and Applications*, 2008.
[6] Steven Lynden et al. ADERIS: An Adaptive Query Processor for Joining Federated SPARQL Endpoints. In *On the Move to Meaningful Internet Systems: OTM 2011*, 2011.
[7] Muhammad Saleem et al. CostFed: Cost-Based Query Optimization for SPARQL Endpoint Federation. *Procedia Computer Science*, 2018.
[8] Describing linked datasets with the void vocabulary, w3c interest group note 03 march 2011. https://www.w3.org/tr/2011/note-void-20110303/.
[9] Olaf Görlitz and Steffen Staab. SPLENDID: SPARQL Endpoint Federation Exploiting VOID Descriptions. In *COLD'11*, 2011.
[10] Andreas Schwarte et al. FedX: Optimization Techniques for Federated Query Processing on Linked Data. In *The Semantic Web – ISWC 2011*, 2011.
[11] Muhammad Saleem et al. HiBISCuS: Hypergraph-Based Source Selection for SPARQL Endpoint Federation. In *The Semantic Web: Trends and Challenges*, 2014.
[12] Fajar Ekaputra et al. Ontology-based data integration in multi-disciplinary engineering environments: A review. *Open Journal of Information Systems*, 2017.
[13] Wei Shen et al. Entity Linking with a Knowledge Base: Issues, Techniques, and Solutions. *IEEE TKDE*, 2015.
[14] Özge Sevgili et al. Neural entity linking: A survey of models based on deep learning. *Semantic Web*, No. Preprint, 2022.
[15] Paolo Ferragina and Ugo Scaiella. TAGME: On-the-Fly Annotation of Short Text Fragments (by Wikipedia Entities). In *CIKM '10*, CIKM '10, 2010.
[16] Stefan Zwicklbauer et al. DoSeR - A Knowledge-Base-Agnostic Framework for Entity Disambiguation Using Semantic Embeddings. In *The Semantic Web. Latest Advances and New Domains*, 2016.
[17] Ledell Wu et al. Scalable Zero-shot Entity Linking with Dense Entity Retrieval. In *EMNLP 2020*, November 2020.
[18] Lajanugen Logeswaran et al. Zero-Shot Entity Linking by Reading Entity Descriptions. In *ACL 2019*, July 2019.
[19] Ikuya Yamada et al. Joint Learning of the Embedding of Words and Entities for Named Entity Disambiguation. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, August 2016.
[20] Ikuya Yamada et al. Global Entity Disambiguation with BERT. In *NAACL 2022*, July 2022.
[21] RDF 1.1 Concepts and Abstract Syntax, W3C Recommendation 25 February 2014. https://www.w3.org/TR/rdf11-concepts/.
[22] Farzaneh Mahdisoltani et al. YAGO3: A Knowledge Base from Multilingual Wikipedias. In *Conference on Innovative Data Systems Research*, 2015.
[23] SPARQL 1.1 Query Language, W3C Recommendation 21 March 2013. https://www.w3.org/TR/2013/REC-sparql11-query-20130321/.
[24] Apache Jena - ARQ - Writing Property Functions. https://jena.apache.org/documentation/query/writing_propfuncs.html.
[25] Silviu Cucerzan. Large-scale named entity disambiguation based on Wikipedia data. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pp. 708–716, Prague, Czech Republic, June 2007. Association for Computational Linguistics.
[26] Razvan Bunescu and Marius Paşca. Using encyclopedic knowledge for named entity disambiguation. In *11th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 9–16, Trento, Italy, April 2006. Association for Computational Linguistics.
[27] Jens Lehmann et al. DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web*, 2015.