1G4-OS-19a-5in

クエリ変換手法に基づくLOD検索の高速化のための機構を備えたSPARQLエンドポイントの試作

A Preliminary Prototype Implementation of SPARQL Endpoint with Transformation-Based Query Optimization

山形　祐史 [*1]　　福田　直樹 [*2]
Yamagata Yuji　　Fukuta Naoki

[*1][*2]静岡大学大学院情報学研究科
Graduate School of Informatics, Shizuoka University

On the retrieval of Linked Open Data using SPARQL, it is not an easy task to construct a proper query considering costs of the query execution. A wrongly formed query causes enormous consumption of endpoints' computing resources. Preventing such an execution of time-consuming queries, transforming such a query into a low-computation-cost query that approximates to the original query may reduce loads of endpoints. In this paper, we present a preliminary idea and its concept on building endpoints having a mechanism to automatically avoid unwanted amount of inference computation by predicting its computational costs and allowing it to transform such a query into speed optimized query. Our preliminary experiment shows a potential benefit on speed optimizations of query executions by applying query rewriting approach. We also present a preliminary prototype system that distinguishes whether a query execution is time-consuming or not by using machine learning techniques at the endpoint's side.

1. はじめに

LOD 検索はエンドポイントに対して RDF[**] 検索のクエリ標準言語である SPARQL[*†] に基づくクエリを発行して行う．オントロジーに基づく推論をエンドポイントで利用する場合，計算の複雑性の高さなどの課題があり，エンドポイントにおける推論器による推論機能の提供は重要な課題の 1 つとなっている．

LOD 検索において，問い合わせを受け取る側は，検索結果を得るまでにかかる時間をどの程度まで許容可能か，推論の厳密さをどの程度まで妥協することができるかなどの，問い合わせを送る側の意図を把握することが難しい．

この課題に対処するために，LOD 検索クエリ作成者が，検索意図に応じた適切なクエリを作成する場面を支援するというアプローチも考えられる [山形 14]．しかし，クエリ作成者が，エンドポイントにおけるクエリの実行にどの程度の時間を要するかを常に予想しながら，適切なクエリを作成することは，容易でない．そうした洗練されたクエリを作成できない検索者から送られたクエリの実行のためには，エンドポイント側での多くの計算資源を消費してしまう．クエリの実行に伴うオントロジーに基づく推論時間を推定して，推論に関して極端に処理負荷の高いクエリの実行を抑制したり，あるいはより低い負荷で実行可能な近似したクエリの実行に置き換えることで，そうしたエンドポイント側における推論の負荷を低減できる可能性がある．

本研究では，LOD 検索クエリの問い合わせ文の実行における低負荷化と最適化をエンドポイント側で自動的に行うための手法を検討し，その機構を試作する．

2. 研究の背景

2.1 推論器の改良とその課題

OWL Full を除く OWL のサブ言語は，それぞれが異なる記述論理に相当する言語として設計されている．記述力と計算の複雑性の異なるサブ言語を用意することで，必要な記述力，あるいは許容可能な計算論的複雑性の面から，用途に応じたサブ言語の選択を可能にしている．

また，推論の所要時間を短縮するための試みとして，推論アルゴリズムの改良や，推論器の改良などが行われてきた．

タブローを改良した Hyper tableaux アルゴリズム [Baumgartner 96] を利用した推論器 HermiT は，それまでオントロジーに基づく推論タスクの 1 つである classification が不可能であった GALEN オントロジーの classification を可能にした [Motik 09].

Romero らは，classification を行うための推論器として MORe を提案している [Romero 12]．MORe は適用できる記述論理の範囲が広い推論器と，適用できる記述論理の範囲は狭いが高速な推論器を組み合わせた推論器である．各推論器はブラックボックス化されており，各推論器における推論アルゴリズムは考慮する必要がない．MORe には HermiT と ELK[Kazakov 11] を組み合わせた実行モード，JFact と ELK を組み合わせた実行モードがある．また，実験的なモードとしてさらに別の実行モードが用意されている [*‡]．ELK は OWL 2 EL に対応する推論器であり，ELK を用いることで HermiT や JFact 単体を用いるよりも高速に classification を行うことを可能にしている．Romero らはいくつかのオントロジーに対する HermiT と MORe の classification に要する時間を比較しており，比較結果によると，Gene Ontology などの OWL 2 EL の表現範囲をこえる公理が存在しないオントロジーの classification に要する時間は HermiT 単体と比較して 69.0%から 96.6%短縮，OWL 2 EL の表現範囲内に収まる公理の数が全体の 94.9%から 98.1%のオントロジーでは 65.8%から 74.6%短縮され，OWL 2 EL の表現範囲に収まる公理が全体の 45.2%である Biomodels オントロジーでは 22.4%短縮されている．

Kang らは，個々のオントロジーにおける推論にかかる計算量が十分に特徴づけされていないという問題に対して，機械学習技術を用いて取り組むための体系的な研究を行っている [Kang 12].

連絡先: 山形　祐史，静岡大学情報学部，〒 432-8011 静岡県浜松市中区城北 3-5-1，cs10101@s.inf.shizuoka.ac.jp

[**] http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/
[*†] http://www.w3.org/TR/rdf-sparql-query/
[*‡] http://code.google.com/p/more-reasoner/wiki/MOReCli

彼らはオントロジーの classification にかかる時間の予測に利用可能ないくつかの尺度を示し，推論器の推論性能を予測する様々な分類器を作成，評価している．評価結果によると，予測精度が 80%を超える高精度な分類器を得ている．

彼らはまた，これまでアルゴリズムと推論器の高度な最適化が行われてきたが，オントロジーの classification は未だに困難な問題であり，定量的な分析が可能になること，そして文法的特徴を利用して推論性能を予測することが可能になることが望ましいとしている．

Gonçalves らの研究 [Gonçalves 12] では，いくつかのオントロジーをランダムに複数等分し，オントロジー部分集合とその和集合に対して classification する実験を行っている．実験結果から，classification にかかる時間が全体集合に近づくにつれて，線形的に増加するとは限らないことを示している．同研究において Gonçalves らは，推論にかかる時間を増加させるオントロジー部分集合であるホットスポットを特定するアルゴリズムを提供しているが，推論にかかる時間を増加させるオントロジー部分集合単体での機能性と，オントロジー全体集合における影響との関係は明らかになっていないとしている．

2.2 クライアントサイドでの推論付き SPARQL クエリ支援

RDF を検索するためのクエリ言語標準である SPARQL を用いて LOD を検索する際に，推論速度を考慮した効果的な SPARQL クエリを行うためのクエリ構成支援システムの試作について述べる．

ホットスポット [Gonçalves 12] と呼ばれる推論速度を制約している箇所を特定するために，LOD エンドポイントに事前にリクエストを送信し，ホットスポットの特定とそれに伴うクエリの書き換えを支援する機能の実現を考える．本機能は，リクエストを送信する機能，リクエストを受信する機能，ホットスポットを特定する機能，ホットスポットを送信（返信）する機能で構成される．

ホットスポット特定を，SPARQL クエリを用いて実現することを考える．ホットスポット特定の際には，特定に極端な時間がかかるなどのエンドポイントへの負荷を考慮する必要がある．例えば規模の小さいオントロジーであれば，オントロジーをダウンロードして，ダウンロードしたオントロジーに対してホットスポットの特定を行う，という方法が考えられる．これらの課題をふまえて，我々は，クライアントサイドにおける推論付き SPARQL クエリの構成支援システムを試作した [山形 14].

このシステムでは，クエリ構成者が使用するエディタ上で，そのクエリの実行時間特性などの予測などを，機械学習の仕組みを用いて実現している．

クライアントサイドでの，推論時間を考慮した SPARQL クエリ構成支援による，時間のかかるクエリ実行の回避には限界がある．例えば，エンドポイントへの負荷を考慮せず，時間のかかるクエリを送る検索者が存在する可能性がある．推論時間を考慮したクエリを構成することを拒絶する検索者によるクエリが集中した場合，エンドポイントへの負荷が大きく，他の軽いクエリを処理することもできなくなる可能性がある．

3. 本研究のアプローチ

3.1 概要とシステムの構成

本研究では，推論に時間のかかるクエリの実行を，エンドポイント構築者側で回避することを支援するためのシステムを試作する．支援のために，受け取ったクエリが，実行に時間がかかるクエリであるかどうかを判定できる仕組みを試作する．

エンドポイントレベルで時間のかかるクエリ実行を回避する仕組みを提供することで，これらの問題に対処する仕組みについて考える．検索者によって送られたクエリを受け取った際に，受け取ったクエリの実行に時間がかかるかどうかを判定する．時間のかかるクエリでないと判定された場合は，そのままクエリを実行する．時間のかかるクエリであると判定された場合には，クエリの実行を拒絶すること，または時間のかからないクエリに書き換えた後に実行し，実行結果とともにクエリを書き換えたことの通知を行うことが考えられる．

これらのことの実現に向けて，時間のかかるクエリ，かからないクエリの判定機構，クエリ実行の拒絶を検索者に伝える機構などを備えた仲介システムを試作する．仲介システム（フロントエンド EP と呼ぶ）は，検索者とエンドポイント（バックエンド EP と呼ぶ）の間に位置する．フロントエンド EP を介したクエリ実行の概観を，図 2 に示す．

クエリ実行時間を推測するための方法として，機械学習器を用いて判定を行う方法と，過去に行われたクエリの中から類似するクエリを探し出し，その実行時間を調べる方法が考えられる．本論文では，機械学習器を用いた実行時間のかかるクエリ判定について検討し，判定機構を試作する．

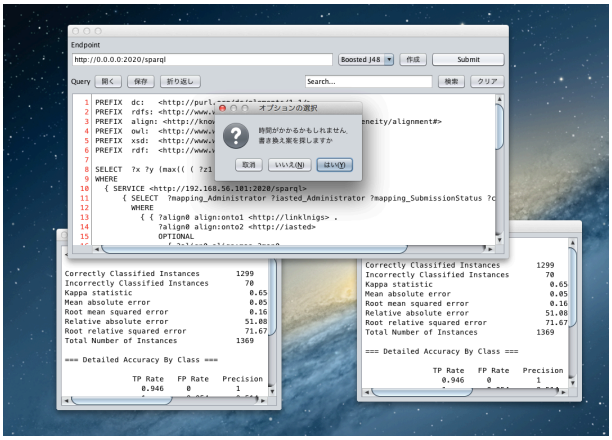

図 1: クエリ構成支援システムの動作例

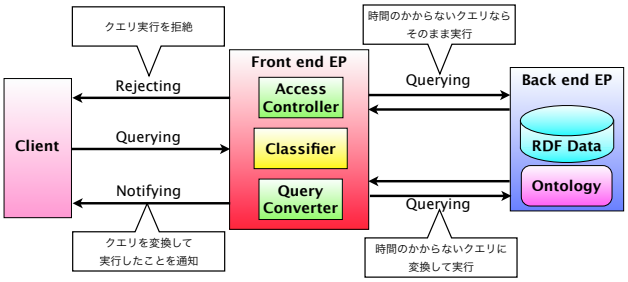

図 2: フロントエンド EP を介したクエリ実行の概観

3.2 過去クエリの蓄積とその参照支援

クエリ構成者が推論の影響を考慮してクエリを書き換えていくために，過去に作成された参考となるクエリを提示することによる支援について考える．同じ URI が含まれるクエリを起点として，クエリ中で参照しようとしているクラスと等価なクラスを参照していたり，クラスがどのようなプロパティを参照するときにどのようなクエリを構成するかを，クエリ構成者が比較検討できるようにしたい．SERVICE 句を用いたクエリを構成する際には，SERVICE 句を用いた過去のクエリから，どのエンドポイントにアクセスする場合に，どのようにアクセス先のオントロジーにあったクエリを記述するかを参照できるようにしたい．これらの目的のために，フロントエンド EP には，実行されたクエリを履歴として蓄え，そのクエリに対してアクセスする手段を提供する．本機構の詳細は，誌面の都合により割愛する．

3.3 予備実験

以前に我々は，ある基準に従って生成したクエリの実行時間を観測することで，クエリ実行時間をクエリ書き換えによってどの程度減少させる可能性があるか調べた．

Linklings オントロジー [**] の記述を一部削除し，クラスのインスタンスを付与したデータを用意した．このデータセットをもとに，Joseki[*†](v3.4.4) を用いて，推論器 Pellet(v2.3.0)[Sirin 07] が動作する SPARQL エンドポイントを作成した．作成したエンドポイントに対して，オントロジーのクラスのインスタンスを得るクエリをクラスごとに発行し，実行時間を計測した．各クエリの 100 回実行時，200 回実行時の平均実行時間を図 3 に示す．実験環境は MacBook Pro, 2.6 GHz Intel Core 2 Duo, 6 GB 667 MHz DDR2 SDRAM, OS X 10.8.5 である．

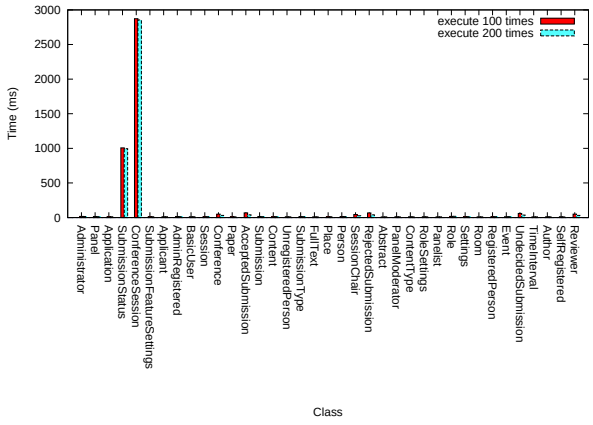

図 3: クエリの平均実行時間

エンドポイント側でクエリの答えをキャッシュしている可能性を考え，各クエリを 200 回実行したときの総実行時間から，各クエリを 100 回実行したときの総実行時間を 0.99 倍し，各クエリを 100 回実行したときの総実行時間から減じることで，初回実行時間を求める．

図 4 が，この方法で求めた初回実行時間の推定値を示したものである．初回実行時に，いくつかのクラスのインスタンスを得るのに時間を要していることがわかる．特に，図 3 では SubmissionStatus クラスと ConferenceSession クラスのインスタンスを得るクエリで顕著であったのが，図 4 ではさらに多くのクラスでクエリ実行に時間がかかる場面が出てきている．

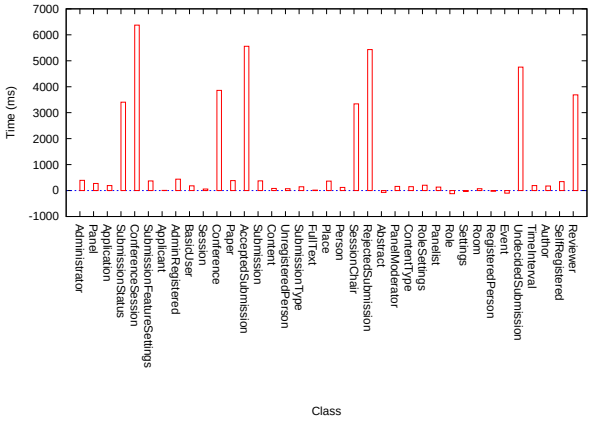

図 4: クエリの初回実行時間

ConferenceSession の親クラスである Event クラスのインスタンスを得るには約 200 分の 1 の時間で済んでいる．このような特性を，クエリの構成に効果的に活かす方法の検討を，現在進めている．

4. 実行時間のかかるクエリの判定

フロントエンド EP では，SPARQL クエリの実行に時間がかかるかどうかを判定するための機械学習に基づく機構を実装している．

機械学習に使用する属性を，クエリの内容から推論を考慮して取得する機構として，次の機能を実装している．

1 つは，クエリ実行時間に関わる属性として，クエリに存在する各クラス記述の出現回数を取得する機構である．もう 1 つは，最初に出現するクラス記述を抽出する機構である．これらの機構は Jena フレームワーク [*‡] を用いて実装した．アルゴリズムの概要を Algorithm 1 に示す．

List_Q は，クエリ中に存在する URI を格納するためのリストである．List_C は，オントロジーの名前付けされたクラスの URI を格納するためのリストである．Count[] は，各クラス記述の出現回数をカウントするための整数型配列である．配列のサイズは，List_C のサイズで与えられる．FirstAppearedClass は，最初に出現するクラス記述を格納するための文字列型変数である．メソッド getURIs() は，クエリ中に存在する URI をリスト形式で取得するメソッドである．メソッド listNamedClasses() は，オントロジー中の名前付けされたクラスの URI をリスト形式で取得するメソッドで，このメソッドは，実際には Jena フレームワークで提供されている OntClass インタフェースで定義されている，listNamedClasses() メソッドの返すイテレータを，リスト化している．メソッド index() は，リスト中の要素のインデックスを返すメソッドである．メソッド setValue() は，抽出した属性値を，機械学習器が参照可能とするためのメソッドである．クエリ中に存在する URI のリスト（List_Q）と，クラスのリスト（List_C）をつきあわせる．URI のリスト中の要素 u が，クラスのリスト中に存在するなら，そのクラスの出現回数（Count[List_C.index(u)]）をカウントアップする．u が入力されたクエリ中で最初に出現したクラス記述であれば，あ

[**] http://www.linklings.com/
[*†] http://www.joseki.org/
[*‡] https://jena.apache.org/

わせて取得し，FirstAppearedClass に格納する．

Algorithm 1 入力クエリの属性値抽出

Input: Q : クエリ, O : オントロジー
Output: S : 属性値群

```
List_Q ⇐ Q.getURIs();
List_C ⇐ O.listNamedClasses();
Count[List_C.size] ⇐ 0;
FirstAppearedClass ⇐ ∅;
for each URI u in List_Q do
  if u is in List_C then
    Count[List_C.index(u)] ⇐ Count[List_C.index(u)] + 1;
    if u is a class appeared for the first time then
      FirstAppearedClass ⇐ u
    end if
  end if
end for
S ⇐ setValue(FirstAppearedClass, Count);
return S;
```

入力されたクエリを事例として時間のかかるクエリであるかを判定する仕組みの実現には，Weka[Hall 09] を用いており，フロントエンド EP 内に実装している．

入力されたクエリを，本機構を用いて時間のかかるクエリであるかどうか判定し，事前に設定されたしきい値より時間のかかるクエリであれば実行せずクエリ構成者に通知するかクエリの書き換えを行い，時間のかからないクエリであればそのまま実行する．一連の流れを図 5 に示す．

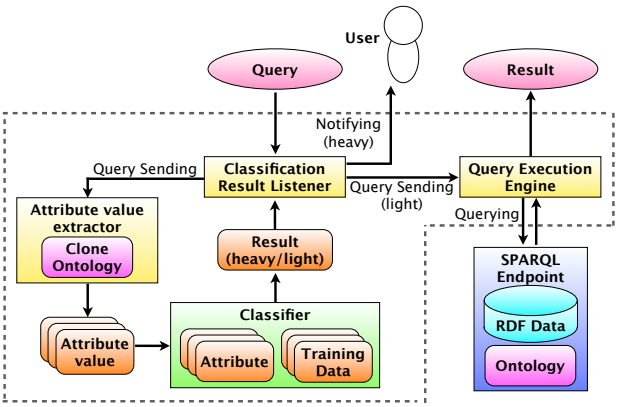

図 5: 時間のかかるクエリ判定の流れ

様々な条件に対して有効な判定を行えるようにするには，選択可能な属性として，適切なものを用意することが必要となる．適切な属性選択し，精度と適用範囲のバランスを取るための機構の実現は，今後の課題となる．

5. おわりに

本論文では，クエリ変換手法に基づく LOD 検索の高速化のための機構をエンドポイント側に持たせるための機構について，その基本的なアイデアを述べた．予備実験として，クエリ変換手法に基づくクエリ実行の高速化の可能性を示した．機械学習器を用いて，クエリ実行に時間がかかるかどうかをエンドポイント側で判定する機構の実現と，その試作について述べた．

今後の課題としては，エンドポイントに対して行われたクエリとその実行時間を蓄積し，利用する場合における，プライバシへの配慮方法の検討と，その際の動作の効率性の改善がある．クエリは機械学習に利用し，学習した結果だけをフロントエンド EP 内でのクエリ実行時間推定だけに用いることは可能である．一方で，クエリ構成者が用いる支援機構と連携して，より高速で効果的なクエリを可能とするために，これらの過去クエリのログを有効に活用するためには，クエリの内容に残るプライバシ面での課題への対応の他に，大量に蓄積されたクエリからの高速な類似クエリの検索手法の実現が重要となる．これらの課題の検討は今後の課題である．

SPARQL1.1 で提供される，複数のエンドポイントに対する横断的検索への対応，マッピングの信頼度を用いた横断的検索への対応 [Fujino 13] も，今後の課題である．

参考文献

[Baumgartner 96] Baumgartner, P., Furbach, U., and Niemelä, I.: Hyper Tableaux, in *Logics in Artificial Intelligence*, pp. 1–17, Springer (1996)

[Fujino 13] Fujino, T. and Fukuta, N.: Utilizing Weighted Ontology Mappings on Federated SPARQL Querying, in *The 3rd Joint International Semantic Technology Conference (JIST2013)* (2013)

[Gonçalves 12] Gonçalves, R. S., Parsia, B., and Sattler, U.: Performance Heterogeneity and Approximate Reasoning in Description Logic Ontologies, in *The Semantic Web–ISWC 2012*, pp. 82–98, Springer (2012)

[Hall 09] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H.: The WEKA data mining software: an update, *ACM SIGKDD explorations newsletter*, Vol. 11, No. 1, pp. 10–18 (2009)

[Kang 12] Kang, Y.-B., Li, Y.-F., and Krishnaswamy, S.: Predicting Reasoning Performance Using Ontology Metrics, in *The Semantic Web–ISWC 2012*, pp. 198–214, Springer (2012)

[Kazakov 11] Kazakov, Y., Krötzsch, M., and Simančík, F.: Concurrent Classification of $\mathcal{EL}$ Ontologies, in *The Semantic Web–ISWC 2011*, pp. 305–320, Springer (2011)

[Motik 09] Motik, B., Shearer, R., and Horrocks, I.: Hypertableau Reasoning for Description Logics, *Journal of Artificial Intelligence Research*, Vol. 36, pp. 165–228 (2009)

[Romero 12] Romero, A. A., Grau, B. C., and Horrocks, I.: MORe: Modular Combination of OWL Reasoners for Ontology Classification, in *The Semantic Web–ISWC 2012*, pp. 1–16, Springer (2012)

[Sirin 07] Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., and Katz, Y.: Pellet: A Practical OWL-DL Reasoner, *Web Semantics: science, services and agents on the World Wide Web*, Vol. 5, No. 2, pp. 51–53 (2007)

[山形 14] 山形 祐史, 福田 直樹：推論速度を考慮した効果的な SPARQL クエリ構成支援システムの試作, 情報処理学会第 76 回全国大会 (2014)