

高速な Property Path クエリ検索を可能にする SPARQL 処理系

石川 康貴^{1,a)} 田浦 健次朗^{1,b)}

概要：セマンティックウェブやソーシャルネットワークなど数多くの分野でグラフ構造を持ったデータが見られ、全般的なデータ量の増大に伴って、これを効率的に処理するシステムの必要性が高まっている。こういったデータに対する処理系として従来用いられてきた関係データベース（RDB）等は、複数ホップの頂点を辿る検索には適していないため、現在までにグラフデータに適した新しいシステムが考案されている。その一つに RDF という型式のグラフデータを処理する SPARQL 処理系があり、これは RDF に対するクエリ言語である SPARQL に対応したグラフデータベースである。また、最新の SPARQL の仕様である SPARQL1.1 には、クエリを正規表現で記述できる Property Path クエリというものが含まれる。これはクエリの表現力を大きく上げることが出来ると考えられ、これに対応した SPARQL 処理系もいくつか提案されている。しかし、これらの現行の処理系は Property Path クエリ検索に対して効率的な並列化が成されているとは言えず、速度が十分とは言えない。そこで本稿では、Property Path クエリ検索に対応した新たな SPARQL 処理系を提案する。提案する SPARQL 処理系では Property Path クエリ検索を高速化するために効率的な並列化を行っており、それらの詳細な手法について説明している。

キーワード：グラフ処理系, RDF, SPARQL, Property Path

A SPARQL Processing System Supporting Efficient Property Path Search

ISHIKAWA YASUTAKA^{1,a)} KENJIRO TAURA^{1,b)}

Abstract: In many area, for example, semantic web, social network, we can see the data including graph structure, and more efficient system for graph data is needed with increasing of the amount of data. Up to date, several new graph specific systems have been proposed because existing system like Relational DataBase (RDB) is not suitable for graph traversal. SPARQL processing system is one of these graph specific systems which managing RDF graph data and SPARQL query, one of graph query languages. Besides, the latest SPARQL specification, SPARQL1.1, has Property Path query which enable us to write query with using regular expression and enhance the power of expression. However, many SPARQL processing system can't manage Property Path query, and systems following SPARQL1.1 don't provide enough searching power. Therefore, in this article, I propose new SPARQL processing system, which can manage Property Path query and provide enough searching power by using several techniques and efficient parallelization.

Keywords: Graph Processing System, RDF, SPARQL, Property Path

1. はじめに

近年, IoT (Internet of Things) などの言葉で表されるように, センシング, ネットワークの技術が発達し, 今まで素

¹ 東京大学大学院情報理工学系研究科
University of Tokyo
^{a)} ishikawa@eidoss.ic.i.u-tokyo.ac.jp
^{b)} tau@eidoss.ic.i.u-tokyo.ac.jp

通りされていた情報がデータとして収集されるようになってきている。米国の IDC による調査 [1] によれば、2013 年に地球上で生成されたデータ量は 4.4ZB であり、2020 年には 10 倍の 44ZB にまで増えると予測されている。このようにデータの増加速度は年々上昇していくと考えられ、こういった現状はしばしばビッグデータという言葉を用いて表されている。

その中でも、Facebook や Twitter などの SNS が持つ「グラフ」という構造を持つデータが注目を集めている。グラフは vertex とそれらを繋ぐ edge という二種類の要素で基本的に構成されている。例として、facebook, twitter のような SNS が挙げられ [2]、それらでの交友関係やフォロー関係はグラフとして表される。そういった構造のデータに対しては例えば「友達の友達」や、「ある人から“知っている”という関係で辿った時にどの人まで辿り着けるか」を検索するといった、従来の関係データベース (RDB) での処理には適していないような、特有なクエリの需要が存在する。しかし、それらは従来のリレーショナルモデルに基づく RDB のようなデータベースには、比較的重い処理とされる join 演算を繰り返し行わなければならないこともあって不向きであり、これらを処理するための新しい仕組みが必要となっている。

こういった処理を実現するために現在登場している仕組みとして、まず一つ目にグラフデータベース (GraphDB) がある。GraphDB にはデータをグラフで処理しやすい形で保持していること、グラフ処理専用のクエリを備えていることといった特徴があり、グラフ特化型のデータベースである。もう一つにグラフ処理系 (graph processing system) というものがあり、これは静的なデータ解析を得意としている。グラフデータベース (graph database) には代表的なものとして、neo4j[3]、HypergraphDB[4] といったものが挙げられ、またグラフ処理系では代表的なものとして、Pregel[5]、Trinity[6]、といったものがある。

前者に挙げた GraphDB で使われるクエリ言語の型式として、[7] で分類されている中で最も基本的なものの一つに、Regular Path Query (RPQ) がある。これはクエリを用いて辿る edge のラベルを指定し、その両端の vertex を返す、といったものであり、辿るパスの指定には * や ? のような正規表現を用いることが出来るため、グラフデータ特有の検索をユーザーが分かりやすい形で行うのに大変有用であると言える。

また、グラフデータを処理するための枠組みは様々なに研究、開発されており、クエリやデータ形式は様々なものが乱立している。そんな中で、これらを標準化するために一つの型式として登場したのが RDF [8] と SPARQL [9][10] である。RDF ではグラフデータを (subject, object, predicate) という 3 つの要素から成る triple という単位に分解し、グラフを表しやすいように定式化したデータ型式である。ま

た、SPARQL は RDF に対するクエリを定義し、RDB における SQL に替わるようなものを提供するための言語である。現在 SPARQL のバージョンとして、1.0 と 1.1 が存在している。

SPARQL 1.0 のクエリをサポートしたようなシステムで、高速化のための並列分散処理に工夫が成されているものはいくつか研究・提案がされている [11][12]。しかし、SPARQL 1.0 における仕様では、グラフデータに対するクエリ言語として、基本的なものしか規定されておらず、これらのシステムは RPQ が実現するような表現力は持っていない。また、SPARQL 1.0 の表現力を強化したものとして、W3C [13] によって SPARQL 1.1 [10] が策定され、新たに Property Path という RPQ と同等の正規表現を用いたクエリを記述することが可能になったが、[14] で示されるように SPARQL 1.1 が要求しているセマンティクスには一部問題があり、これに従ったシステムは小規模のグラフに対しても処理を現実的な時間で終わらせることが出来ない場合があるといった問題を抱えている。

そこで本稿では、ユーザーにとって有益である、Property Path クエリをサポートし、なおかつ並列化等の工夫により、十分な速度を持った SPARQL 処理系を提案する。また、提案システムで用いるいくつかの手法についてそれぞれ述べていく。

本稿の構成としては、まず 2 章ではこれらの背景知識について説明し、3 章では関連研究として既存の SPARQL 処理系等を説明し、4 章では拡張した SPARQL 文法や提案する処理系についての説明、6 章でまとめを行う。

2. 背景

2.1 グラフデータ

「点」と「線」の主に二つの要素から成り立っており、それらが繋がっているようなデータ構造を「グラフ」と呼ぶ。点を vertex、線を edge と呼び、それらには付加的な属性として label がついていることもある。また、各 vertex、または edge に属性として、連想配列が付属しているようなグラフを property graph [15] などと呼ぶ。

グラフの edge には有向のものとそうでないものがあり、どの vertex から出てどの vertex に入るのかが示されているものを有向グラフ、そうでないものを無向グラフという。

グラフ構造の例として、Facebook, twitter などの SNS でのソーシャルネットワーク、コンピュータのネットワークトポロジー、タンパク質の分子構造といったものが挙げられる。

2.2 Regular Path Query

グラフデータに対しては、それに対して検索を行うためのクエリ言語として、様々なものが用意されているが、その一つに Regular Path Query (RPQ) [7] がある。RPQ

は property graph に対しての使用が想定されており、グラフの経路をラベルで指定することによって、指定されたパターンと合致する経路（パス）の両端を検索結果として返すものである。

図1はRPQのクエリとそれに合致するパスの例である。クエリはラベルとして、 a または b が続いているようなパスの両端の vertex を表している。

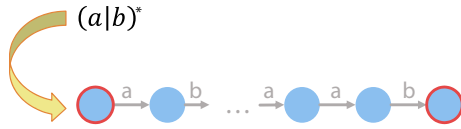


図1 Regular Path Query の例

2.3 RDF

RDF[8]はグラフデータを表す型式の一種であり、図2のように表される。RDFを構成する要素は (subject, object,

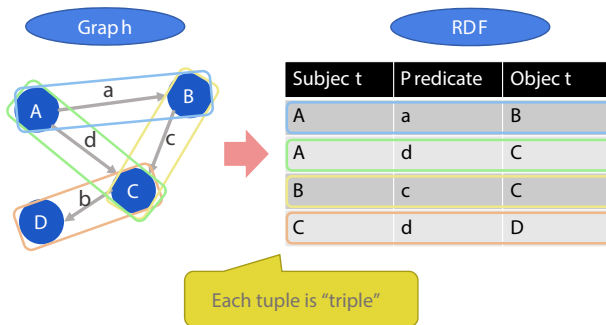


図2 RDF データの概念図

predicate) であり、これらを3つまとめた単位を triple と呼ぶ。subject, object は vertex を表し, predicate は subject から object に向かう edge のラベルを表している。RDF ではこれらの triple の集合としてグラフを表している。この抽象構文を実際に表す方法は様々あり、その代表的なものの一つに xml を用いた RDF/XML[16] がある。図3はその例である。

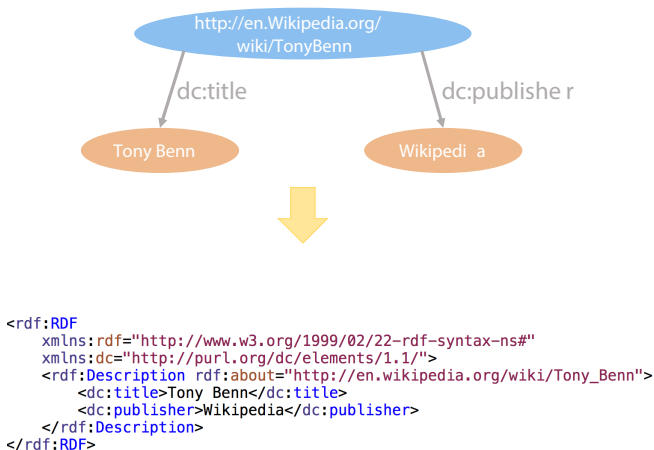


図3 Notation3 の例

RDF 型式で表されるグラフデータに関連する動きとして、DBpedia[17] などに代表されるセマンティック web が挙げられる。これは web 上のデータを RDF を用いて構造化し、意味情報を持たないワールド・ワイド・ウェブ (WWW) の HTML のデータから意味のあるデータを構築することによって、WWW の利便性を高める、というものである。DBpedia はその一つであり、ウィキペディアから情報を抜粋し、その情報を RDF 型式で構造化した形で web 上で利用可能にするプロジェクトである。

2.4 SPARQL

SPARQL はグラフデータに対する一種のクエリ言語であり、関係データベースにおける SQL のように、RDF 型式で表されたグラフデータに対する検索クエリを標準化しようという文脈で作られたものである。

SPARQL にも様々な表現形式が存在し、そのうちの 하나가、図4のようになる。

```

PREFIX abc:<http://example.com/exampleOntology#>
SELECT ?capital ?country
WHERE {
  ?x abc:cityname ?capital;
    abc:isCapitalOf ?y.
  ?y abc:countryname ?country;
    abc:isInContinent abc:Africa.
}

```

図4 SPARQL クエリの例

図4の例は、アフリカにある全ての国の首都の名前を返す、というような SPARQL クエリを表している。

2.5 節で後述するが、SPARQL の仕様には、2008 年に正式な W3C 勧告となった 1.0 と、2013 年に同じく W3C 勧告になった 1.1 が存在する。

2.5 Property Path

SPARQL1.0 では、対応しているクエリは基本的なものに限られていて、RPQ のような正規表現等を使った表現には対応していなかった。しかし、SPARQL1.1 からは Property Path と呼ばれるクエリに対応しており、パスの指定に RPQ と同等の正規表現を用いることが出来るようになった。図5は SPARQL1.1 から書けるようになった Property Path を利用したクエリである。この図を例に挙げて説明すると、このクエリはある vertex と、そこから like という関係で辿り続けた時の先の vertex の組を返すようなクエリである。

3. 関連研究

3.1 SPARQL 処理系

Hammoud ら [12] の分類によれば、SPARQL 処理系は大きく分けて4系統に分けることが出来る。

(1) 単一マシンでクエリを処理する

表 1 既存システムの比較

SPARQL Processing System	Parallelism	Managing Property Path
RDF-3X[18]	parallelized as category1	No
TripleBit[19]	parallelized as category1	No
Hexastore[20]	parallelized as category1	No
Sesame[21]	not sufficient	Yes
Jena[22]	not sufficient	Yes
Trinity.RDF[11]	distributed as category2	No
DREAM[12]	distributed as category3	No

```
SELECT ?A ?B
WHERE{
  ?A (rdf:like) * ?B
}
```

図 5 Property Path Query の例

- (2) 複数マシンにデータを分割し、それぞれが同じクエリを処理する
- (3) 複数マシンにデータを分割し、更にそれぞれで元クエリを分割したサブクエリを実行する
- (4) 複数マシンがそれぞれデータ全体を持ち、分割されたサブクエリを実行する

例えば、1 の例には、RDF-3X[18]、TripleBit[19]、Hexastore[20] といったものがある。これらは単一のマシンで RDF データを処理するためのシステムであるが、SPARQL1.1 でサポートされているような Property Path クエリには対応しておらず、表現力が十分なシステムであるとは言えない。また、Property Path に対応したようなシステムとして、Sesame[21]、Jena[22] などの直近のものが挙げられるが、これらは Property Path クエリ処理に対する並列化等が不十分である。また 3.2 章で後述する SPARQL1.1 のセマンティクスに則った処理を行うため、[14] で述べられているように、少ない数の vertex に対しても現実的な時間で処理を終えられないことがある。

また、分散処理するシステムの例として、Trinity.RDF[11]、DREAM[12]、Huang らの手法 [23] が挙げられる。例えば DREAM は上記の分類の 4 に該当し、Master-Slave モデルを採用している。Master はユーザーから受け取ったクエリをコストモデルに従ってサブクエリに分割し、各々がグラフデータ全体を持ったような Slave に分配し、処理を行う。このシステムでは、各 Slave マシンでの処理は 1 に該当するような処理を行うため、外部システムを利用することが出来、高い拡張性を持っていると言える。しかし、Master がクエリを分割するコストモデルは Property Path には対応していないため、これらのクエリを処理することは出来ない。また、Huang らが提案したシステムが保持するデータ構造では、探索がグラフ上で 2 ホップ以内なら同一のマシン内で探索を行えるという特徴を持っているが、システムが Property Path への対応がないのに加えて、仮

に Property Path のようにホップ数が大きい探索を実行することを考えると、高確率で 2 ホップを超えてしまい性能が大きく低下してしまうことが考えられる。

3.2 SPARQL1.1 の問題点

2008 年に W3C 勧告になった SPARQL1.0[9] では、先ほどの例で挙げたような簡単なクエリしか実現することが出来ず、グラフデータに対するクエリの表現力の観点で問題があった。

これを解決するため策定され、2013 年に W3C 勧告になったのが SPARQL1.1[10] であり、Property Path 等のクエリが追加され、表現力の面で大幅に改善された。

しかし、[14] で示されているように、この文法が要求するセマンティクスには問題があり、少ない数の vertex で構成されたグラフに対しても探索が終わらなかったり、検索結果のデータ量がヨタバイト単位に到達してしまう、といった問題がある。[14] では、SPARQL1.1 に対応したいくつかの SPARQL 処理系に対して実験を行っているが、13 個の vertex から成る完全グラフ（全ての vertex が edge1 ホップで繋がっているグラフ）に対して Property Path で表されるようなクエリを用いて検索を行った場合、どの処理系も現実的な時間で処理を終わらせることが出来ないことが示されている。

4. 提案システム

本稿では前述の SPARQL1.1 のセマンティクスではなく、独自のセマンティクスを採用した上で、Property Path クエリを高速に実行出来るシステムを提案する。提案しているシステムでは、Property Path を含んだクエリを効率的に探索するため、いくつかの手法を用いている。以下ではそれらを説明する。

4.1 実行するセマンティクス

前述の 3.2 節で述べたように、SPARQL1.1 で規定されているパスは図 5 のようなクエリに対して、条件を満たす両端の vertex に対しての全ての経路を取得するようになっているが、これでは現実的な時間での実行は非常に難しい。

本稿で提案しているシステムでは、こういったクエリに

対して考える全ての経路を探索するのではなく、経路に依らず両端の vertex の組のみを探索するようにしている。図 5 を例に挙げると、例えば (A,B) の組として、(Bob, Alice) の組が見つかったとすると、その後の探索で Bob と Alice を繋ぐ全ての経路を探索しようとせずに、その時点で探索を終えるようにしている。これにより、Property Path クエリの探索を現実的な時間内に終えることが可能になる。提案するシステムでは Property Path クエリとして、predicate のパスの指定に以下の正規表現をサポートする。

- elt^* : elt を 0 回以上繰り返すパス
- elt^+ : elt を 1 回以上繰り返すパス
- $elt?$: elt が 0 または 1 回あるパス
- $elt1/elt2$: $elt1$ に続いて $elt2$ が来るようなパス
- $\sim elt$: elt を逆に辿るようなパス
- $elt1|elt2$: $elt1$ または $elt2$ を通るようなパス
- $elt\{n,m\}$: elt が n から m 回続くようなパス

4.2 効率的な RPQ 探索の並列化

グラフデータベースでのクエリの一つに Regular Path Query (以下 RPQ) というものがある。これは、正規表現によってグラフ上でのパスを記述し、その両端の vertex を取得する、というものである。このクエリは SPARQL における Property Path クエリのサブセットのように捉えることができる。

そこで本システムでは当研究室の岩成ら [24] が提案している高速な RPQ 検索を並列化して応用することを提案している。以下にそのアルゴリズムの概略を Algorithm.1 で示す。

Algorithm 1 Parallelized property path query

Input: graph G , property path query Q , Memotable $M(= \phi)$, queue $q(= \phi)$

Output: a set of node pairs N

```

1: divide edges of  $G$  by label
2: convert  $Q$  into NFA  $A_L$ 
3: for a node  $n_0$  in  $G$  do
4:   add the pair  $\{n_0, \text{start state } S_0 \text{ in } A_L\}$  to  $q$  and  $M$ 
5:   while  $q$  is not empty(in parallel) do
6:     pop pair  $\{n, S\}$  from  $q$ 
7:     if  $S$  is final state  $F$  then
8:       add the pair  $\{n, n_0\}$  to  $N$ 
9:       continue
10:    for label  $l$  outgoing from  $S$  do
11:      for node  $n_{next}$  linked with  $n$  by label  $l$  in  $G$  do
12:        if pair  $\{n_{next}, S_{next}\}$  is not in  $M$  then
13:          add  $\{n_{next}, S_{next}\}$  to  $q, M$ 
14:      clear  $M$ 

```

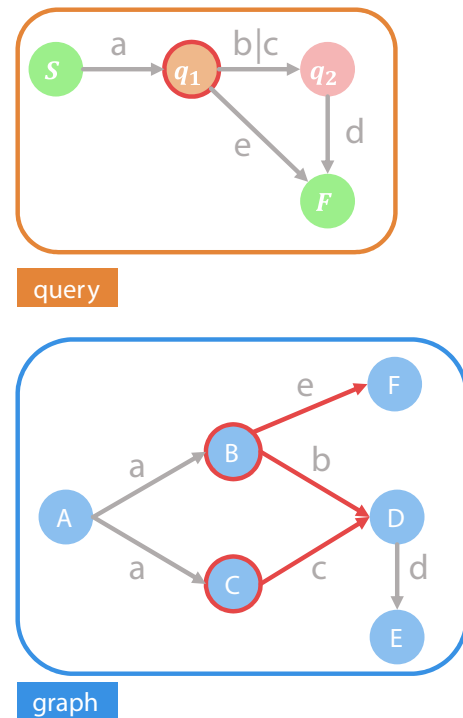


図 6 並列化された Property Path Query 検索の例

この手法では、正規表現が NFA (非決定性有限オートマトン) に変換できることを利用し、まず与えられた SPARQL クエリを NFA に変換する。そして、グラフ上で探索が行われた vertex と、そこに辿り着いた時の NFA 上の状態の組を記録しながら探索を進めていく。

このように NFA の状態とグラフ上の vertex の組を記録することで、グラフ上でエッジを辿って合流するような vertex に対して、その vertex に同じ状態で辿り着くような検索を行っていた場合に枝刈りが可能となる。また、並列に実行した各タスクのワーカーが同一のメモテーブルを参照・更新することで、他のワーカーの探索結果を枝刈りに利用することが可能になる。

図 6 はその例であり、B,C から同じ状態の q_1 で D に遷移するため、その先の探索が枝刈りされる。

4.3 Property Path クエリの分割によるメモ化

一般的なグラフにおける vertex の重要度を表す指標に centrality (中心度) というものがある。centrality はソーシャルグラフの中での重要人物や、ネットワークアーキテクチャでの重要なインフラ部分を抽出するといったものに使われる。centrality にはいくつか種類があり、例として下記のようなものが存在する。

- betweenness centrality
- degree centrality
- closeness centrality

例えば、betweenness centrality はある 2 点とそれを結ぶ最短経路が与えられた時に指定した vertex がその経路上に存

在する確率を表し, degree centrality は指定した vertex が繋がっている edge の数を表す. centrality が相対的に高い vertex を central vertex と呼ぶ.

また, 一般に RDF データがよく使われる web-graph は, スケールフリー性と呼ばれる性質を持っている. この性質は, 多くの vertex の中で一部の vertex だけが数多くのエッジを持ち, 多くの他の vertex と繋がっているが, その他大勢の vertex は少ないエッジしか持っていない, というようなものであり, centrality が高いものの数は限られることになる. web のリンクなどを例に挙げると, 一部の人気の web ページは多くのページからリンクを張られ閲覧されているが, その他大勢の web ページ張られているリンクの数はとても少ない.

そこで, 本システムでは, この性質を利用し, Property Path クエリを高速化する方法を提案する. スケールフリー性を持ったグラフにおいては, ある 2 点間の最短経路を取ってきた際に, 頻繁に central vertex がパス上に出現すると予想されるため, central vertex で Property Path クエリを分割し, そこから始まる探索結果を再利用する.

```
SELECT ?A ?B
WHERE{
  ?A (rdf:like) * ?B
}
```

↓

```
SELECT ?A ?B
WHERE{
  ?A (rdf:like) * Central vertex
  Central vertex (rdf:like) * ?B
  ?A (rdf:like) * ?B
}
```

図 7 Property Path クエリの分割

central vertex で分割したクエリは図 7 のようになる. この時, 図 8 のようなグラフでの探索を考え, 橙色の vertex が central vertex であるとする. ある vertex からスタートした探索が橙色の central vertex を初めて通った時, 図 8 のようにそこから始めた時の探索結果をメモとして保存する. 同一の Property Path クエリがまた別の vertex から探索を開始して, central vertex に辿り着いたとすると, メモを利用することでそこから先の探索を省略することが可能になる.

この手法の利点として, 探索の中間結果のごく一部を保存しておくだけで, 多くの検索での枝刈りが可能になる, という点が挙げられる. これによって大きなグラフに対してもメモリを効率よく利用して探索を高速化することが出来る.

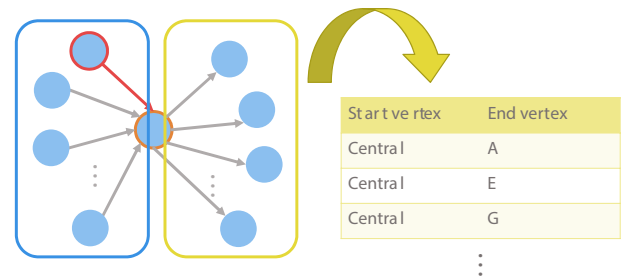


図 8 central vertex によるグラフの分割

4.4 Property Path を考慮したクエリ順序入れ替え

この節では, SPARQL クエリの並列実行の仕方, あるいは実行するクエリの順序について述べる. 例として, 図 9 のようなクエリのそれぞれの行をサブクエリとして全て記述通りに直列に実行することを考えると,

```
SELECT ?A ?B
WHERE{
  ?A (rdf:like) * ?B
  ?A rdf:hobby "Soccer"
  ?B rdf:hobby "Running"
}
```

図 9 複数の Triple を含んだ SPARQL クエリ

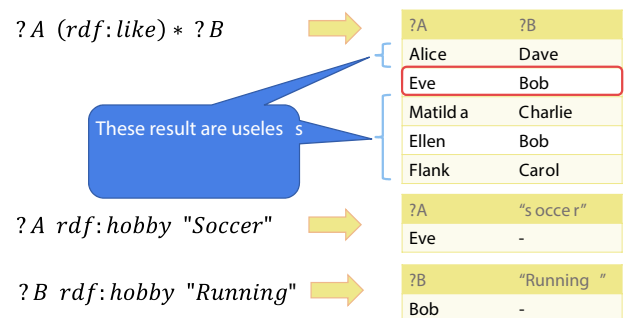


図 10 並べ替えを行わない場合の Triple 毎の検索

図 10 に示されるような結果になり, 一番最初に実行された Property Path クエリの中間結果のほとんどが無駄になってしまっている. これらのサブクエリを並列実行するとしても一番上の Property Path の探索に最も時間がかかるため, この計算の時間に全体の実行時間が律速されてしまう. 更に, 最終的な結果を表示する前段階で, それぞれのクエリの中間結果を join する必要があり, これらの処理のオーバーヘッドが生じてしまう. これらの問題を解決するために, [12] などでは分散環境での実行においても, コストモデルを立て, ある程度直列に実行するようにしている. これにより, 前のクエリの検索結果を利用することで次の探索の際に大きく枝刈りが可能になるし, また, 中間結果のサイズが抑えられ, 分散環境のそれぞれのマシンに置かれた中間結果の通信のコストを削減出来る.

本システムでは、同一ノード内において Property Path を含んだようなクエリの並べ替えによるクエリ実行の高速化を提案する。提案する並べ替えのセマンティクスは次の通りである。

- (1) 辿る edge が多いものはより後に実行される。また、* を含んだクエリは一番後に実行される
- (2) サブクエリの vertex が変数ではなくリテラルであった場合、そのサブクエリは先に実行される
- (3) サブクエリに含まれたラベルの、グラフにおける出現回数が多い場合、そのサブクエリはより後に実行される

これらを図9のクエリに適用すると、図11のようになる。この順序でクエリを直列に実行することによって、?A、?B に対する検索の候補を最初から絞ることが出来、Property Path クエリ探索を実行する際の探索の開始点を大きく絞って計算を大幅に枝刈りすることが可能になる。

```

SELECT ?A ?B
WHERE{
  ?A (rdf:like) * ?B
  ?A rdf:hobby "Soccer"
  ?B rdf:hobby "Running"
}

```

↓

```

SELECT ?A ?B
WHERE{
  ?B rdf:hobby "Running"
  ?A rdf:hobby "Soccer"
  ?A (rdf:like) * ?B
}

```

図 11 Property Path を含むクエリの並べ替え

5. 実験

現在、実装途中であるため、今回は予備実験として、4.4 節で述べた、クエリの実行順序並べ替えによる効果の測定を行った。なお、クエリの並べ替えは手動で行っており、ここでは並列化は行っていない。

実験には、データセット生成のために LUBM[25] を使用した。これによって学術情報に関するオントロジーを表す人工データを生成することが出来、データサイズは任意に指定出来る。実験環境としては、Intel (R) Xeon (R) CPU E5-2699 v3 (2.30GHz)、メモリは 770GB のマシンを用い、C++を用いて実装を行った。

測定には3つの、全て Property Path を含んだようなクエリを使用し、それぞれ Q1, Q2, Q3 とする。これらは図

12, 13, 14 のようになっている。

```

SELECT ?nameA ?nameB
WHERE{
  ?A (takesCourse / ^takesCourse)+ ?B.
  ?A undergraduateDegreeFrom "schoolA".
  ?B undergraduateDegreeFrom "schoolB".
  ?A name ?nameA.
  ?B name ?nameB.
}

```

図 12 Q1

```

SELECT ?addressA ?addressB
WHERE{
  ?A (^publicationAuthor / publicationAuthor)+ ?B.
  ?A emailAddress ?addressA.
  ?B emailAddress ?addressB
}

```

図 13 Q2

```

SELECT ?nameA ?nameB
WHERE{
  ?A ((masterDegreeFrom / doctoralDegreeFrom)
    (^masterDegreeFrom / ^doctoralDegreeFrom))+ ?B.
  ?A researchInterest "Research".
  ?B researchInterest "Research".
  ?A name ?nameA.
  ?B name ?nameB.
}

```

図 14 Q3

Q1 を例にとると、これは取っている授業を辿っていった両端の学生の出身校が指定の場所であった場合、二人の名前を返すようなクエリである。

この Q1 を並べ替えると図 15 のようになる。

```

SELECT ?nameA ?nameB
WHERE{
  ?A undergraduateDegreeFrom "schoolA".
  ?A (takesCourse / ^takesCourse)+ ?B.
  ?B undergraduateDegreeFrom "schoolB".
  ?A name ?nameA.
  ?B name ?nameB.
}

```

図 15 実行順序変更後の Q1

Q1～3 に対して手動で並び替えを行い、それぞれをデータセットに対して実行して比較したのが、次の図 16 である。なお縦軸は対数スケールとなっている。

示されているように、いずれも並び替え後の実行時間は短くなっている。Q2 の場合は性能の向上幅はさほど大きくないが、Q1, Q3 の実行の際には、検索速度が大幅に

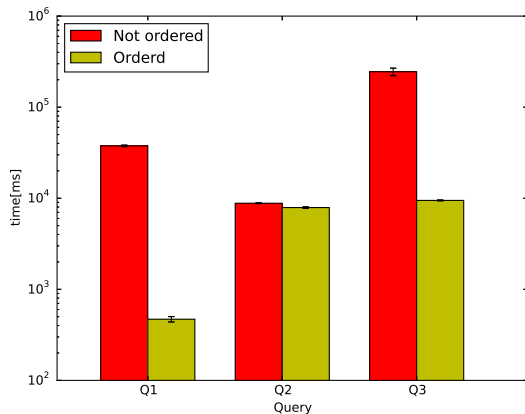


図 16 実行順序並べ替えの比較

向上している。これは、Q1, Q3 では SPARQL クエリに、Vertex のラベルを指定している行があり、その部分を先に実行することで、Property Path クエリの実行の際の探索空間が大幅に縮小したためと考えられる。また、Q2 では Vertex のラベルを指定している行はなく、実行順序を変更しても Property Path 検索の際の探索空間の大きさがさほど変わらなかったと考えられる。これらの結果から、Property Path クエリのように重い処理を含んだ場合、クエリの実行順序の並び替えが大変有用になってくると言える。

6. まとめ

本稿では、近年急激に総量が増加しているデータの中でも、グラフという構造を持ったものに注目し、それを処理するシステムの重要性について触れた。また、グラフデータに対する、クエリ言語の中でグラフ特有のクエリを書くことが出来る Regular Path Query について説明した。

グラフデータの規格の一つとして RDF, それに対するクエリ言語の SPARQL について説明し、それらを用いて検索する既存の SPARQL 処理系について述べ、分類について説明した。そして、それらのシステムの大半では、RPQ と同様に正規表現のクエリが書ける Property Path を扱えないこと、また扱えるシステムも並列化等が十分でなく、高速化の余地があることを示した。

それらを踏まえて本稿では、Property Path をサポートした上でなおかつそれを高速処理するシステムを提案した。このシステムで使う手法は大きく分けて、

- 効率的な RPQ 探索を並列化して利用
- Property Path を考慮したサブクエリ並べ替え
- web-scale グラフの特徴を利用したクエリ分割

といったものになる。

現在、これらの手法を実装したシステムの実装を進めており、今後更なる評価を行う予定である。

参考文献

- [1] : The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things, <http://www.emc.com/leadership/digital-universe/2014iview/index.htm> (2014).
- [2] Ugander, J., Karrer, B., Backstrom, L. and Marlow, C.: The Anatomy of the Facebook Social Graph, p. 17 (online), available from <http://arxiv.org/abs/1111.4503> (2011).
- [3] : neo4j - The World's Leading Graph Database, <http://www.neo4j.org/> (2012).
- [4] Iordanov, B.: HyperGraphDB: A Generalized Graph Database, *Web-Age Information Management*, Vol. 6185 (online), available from <http://link.springer.com/chapter/10.1007/978-3-642-16720-1.3> <http://www.springerlink.com/index/10.1007/978-3-642-16720-1> (2010).
- [5] Malewicz, G., Austern, M. H., Bik, A. J. C., Dehnert, J. C., Horn, I., Leiser, N. and Czajkowski, G.: Pregel, *Proceedings of the 2010 international conference on Management of data - SIGMOD '10*, New York, New York, USA, ACM Press, p. 135 (online), DOI: 10.1145/1807167.1807184 (2010).
- [6] Shao, B., Wang, H. and Li, Y.: Trinity, *Proceedings of the 2013 international conference on Management of data - SIGMOD '13*, New York, New York, USA, ACM Press, p. 505 (online), DOI: 10.1145/2463676.2467799 (2013).
- [7] Barceló Baeza, P.: Querying graph databases, *Proceedings of the 32nd symposium on Principles of database systems - PODS '13*, p. 175 (online), DOI: 10.1145/2463664.2465216 (2013).
- [8] : RDF 1.1 Concepts and Abstract Syntax, <http://www.w3.org/TR/rdf11-concepts/> (2014).
- [9] : SPARQL Query Language for RDF, <http://www.w3.org/TR/rdf-sparql-query/> (2008).
- [10] : SPARQL 1.1 Query Language, <http://www.w3.org/TR/sparql11-query/> (2013).
- [11] Zeng, K., Yang, J., Wang, H., Shao, B. and Wang, Z.: A distributed graph engine for web scale RDF data, *Proceedings of the VLDB Endowment*, Vol. 6, No. 4, pp. 265–276 (online), DOI: 10.14778/2535570.2488333 (2013).
- [12] Hammoud, M., Rabbou, D. A. and Nouri, R.: DREAM : Distributed RDF Engine with Adaptive Query Planner and Minimal Communication, *Proceedings of the VLDB Endowment*, Vol. 8, No. 6, pp. 654–665 (2015).
- [13] : World Wide Web Consortium (W3C), [WorldWideWebConsortium\(W3C\)](http://www.w3.org/).
- [14] Arenas, M., Conca, S. and Pérez, J.: Counting beyond a Yottabyte, or how SPARQL 1.1 property paths will prevent adoption of the standard, *Proceedings of the 21st international conference on World Wide Web - WWW '12*, p. 629 (online), DOI: 10.1145/2187836.2187922 (2012).
- [15] Rodriguez, M. a. and Neubauer, P.: Constructions from Dots and Lines, Vol. X, No. X, pp. 35–41 (online), DOI: 10.1002/bult.2010.1720360610 (2010).
- [16] : RDF 1.1 XML Syntax, <http://www.w3.org/TR/rdf-syntax-grammar/> (2014).
- [17] Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R. and Ives, Z.: DBpedia: A nucleus for a Web of open data, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 4825

- LNCS, pp. 722–735 (online), DOI: 10.1007/978-3-540-76298-0_52 (2007).
- [18] Neumann, T. and Weikum, G.: The RDF-3X engine for scalable management of RDF data, *VLDB Journal*, Vol. 19, No. 1, pp. 91–113 (online), DOI: 10.1007/s00778-009-0165-y (2010).
- [19] Yuan, P., Liu, P., Wu, B., Jin, H., Zhang, W. and Liu, L.: TripleBit: A Fast and Compact System for Large Scale RDF Data, *Proc. VLDB Endow.*, Vol. 6, No. 7, pp. 517–528 (online), DOI: 10.14778/2536349.2536352 (2013).
- [20] Weiss, C. U. O. Z., Weiss, C., Karras, P. N. U. o. S., Bernstein, A. U. o. Z., Karras, P. and Bernstein, A.: Hexastore: sextuple indexing for semantic web data management, *Proceedings of the VLDB Endowment archive*, Vol. 1, No. 1, pp. 1008–1019 (online), DOI: 10.1145/1453856.1453965 (2008).
- [21] Broekstra, J., Kampman, A. and Harmelen, F. V.: Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema, *International Semantic Web Conference ISWC*, Vol. 1, pp. 54–68 (online), DOI: 10.1007/3-540-48005-6_7 (2002).
- [22] Wilkinson, K., Sayers, C., Kuno, H. and Reynolds, D.: Efficient RDF storage and retrieval in Jena2, *Proceedings 1th International Workshop on Semantic Web and Databases*, pp. 35–43 (online), DOI: citeulike-article-id:926609 (2003).
- [23] Huang, J., Abadi, D. J. and Ren, K.: Scalable SPARQL Querying of Large RDF Graphs, *Proceedings of the VLDB Endowment*, Vol. 4, No. 11, pp. 1123–1134 (online), available from <http://www.vldb.org/pvldb/vol4/p1123-huang.pdf> (2011).
- [24] Iwanari, T.: Graph Database における高速な Regular Path Query, 東京大学卒業論文 (2015).
- [25] : The LUBM Benchmark, <http://swat.cse.lehigh.edu/projects/lubm/>.