D-007

RDBMS 向けグラフ縮約検索処理の予備評価
Preliminary Evaluation of Contraction-Based Information Retrieval on RDBMS

横井 一仁† 西山 博泰†
Kazuhito Yokoi Hiroyasu Nishiyama

1. はじめに

近年、企業内にて様々な形式のデータが扱われている。一般的に RDBMS にデータを格納する際は、表構造を事前に決める必要がある。本格納方法では、運用中に格納形式の変更を行うのが容易でなく、企業内で扱われる形式の変化に柔軟に対応することが難しい。このような用途には、事前にデータの格納形式を決めずに格納できるグラフ形式が向いている。しかし、グラフ形式によるデータ格納は検索の際、ノードを探索するため、検索性能が低下する欠点がある。

我々は、グラフ形式でデータを管理する RDF(Resource Description Framework)ストア Apache Jena[1]において、縮約グラフを用いた検索処理高速化技術について報告した[2,3]。代表的な RDF ストアでは、主語(Subject)、述語(Predicate)、目的語(Object)の 3 つの値を 1 組としてデータを管理する。RDF ストアに対し、検索を行う際は、SPARQL(SPARQL Protocol and RDF Query Language)と呼ばれるグラフ形式専用のクエリ言語を用いて問い合わせる。SPARQL の言語仕様は、W3C によって標準化されており、RDF を扱うデータベースも存在する[4,5]。しかし、グラフ形式のデータを扱うシステムを開発する際に、既存データとの連携がしやすく開発者が扱いやすい SQL を用いて開発したいケースや、実績が豊富な RDBMS 製品を選択したいケースがある。このような RDBMS 上でグラフ形式のデータを扱うケースにおいても、前述したように検索性能が低下する問題が生じるため、RDBMS 上にて縮約グラフが利用できることが望まれる。

そこで本研究では、グラフを表現するためのデータを付与した RDBMS に対し、縮約グラフを用いることで、検索性能を向上させる方式を評価した。縮約グラフとは、グラフ形式のデータの接続関係を要約した小規模なグラフ形式のデータであり、元のグラフ形式のデータへの対応関係を持つ。検索の際に、まず縮約グラフを参照し、元のデータの検索すべき範囲を限定することで、検索処理時間を短縮できる。

縮約グラフを用いた検索性能の評価を行ったところ、グラフ形式のデータを直接検索した場合比較し、最大 24 倍の検索性能向上を確認した。

2. RDBMS 向けグラフ縮約検索

グラフ縮約検索とは、グラフ形式のデータを要約した縮約グラフを用い、元のグラフデータの検索範囲を限定することで、検索処理時間を短縮する検索方式である。

2.1 縮約グラフ

グラフ形式のデータとは、例えば図 1 のようなノード同士が接続された構造を持つデータである。本データには表 1 のように各ノードに、情報を付与することもある。本研究では、グラフ形式のデータを記述する方法として隣接リストモデルを用いて、RDBMS 上に格納した。隣接リストモデルは表 2 のような 2 つのノード間の接続関係を 1 行ずつ記述するモデルである。

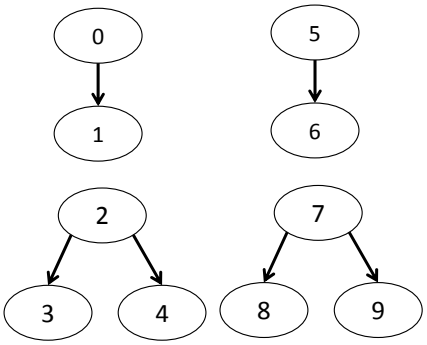


図 1 グラフ形式のデータ

表 1 ノード管理 (NAME 表)

ID	VAL
0	“Name0”
1	“Name1”
2	“Name2”
3	“Name3”
4	“Name4”
5	“Name5”
6	“Name6”
7	“Name7”
8	“Name8”
9	“Name9”

表 2 ノード接続管理表 (LINK 表)

SRC	DEST
0	1
2	3
2	4
5	6
7	8
7	9

縮約グラフは、図 2 の左側に示すようなグラフ形式のデータの中で、同一の接続関係を持つ構造を統合し、サイズを小さくした隣接リストモデルのデータである。縮約グラフの各ノードは、元のグラフ形式のデータへの対応関係を持つ。

縮約グラフは、元のグラフ構造のデータのうち、接続先のない末端のノードを起点とし、同じ接続関係を持つノードを 1 つにまとめる操作を繰り返すことで、作成される。例えば、図 2 中のノード番号 0,1 から成るグラフ構造と、ノード番号 5,6 から成るグラフ構造は、同じ構造である。縮約グラフを作成する際は、末端のノード番号 1 と 6 を起点として構造を比較し、同じ構造の場合は統合したグラフ構造を縮約グラフとして登録する。統合により作成された縮約グラフは図 2 中のノード番号 100,101 から成る構造である。縮約グラフを作成する際、ノードが元のグラフ構造のどのノードに対応するか分かるよう、接続関係(点線矢

† 日立製作所 横浜研究所
Yokohama Research Laboratory, Hitachi, Ltd.

印)が付与される。同様にノード番号 2,3,4 とノード番号 7,8,9 から成る構造も縮約グラフ作成処理によって統合され、縮約グラフ上にノード番号 102,103,104 から成る構造が生成される。本構造おいても元グラフ上の構造との接続関係が付与される。

本研究の評価では、表 2 のノード接続管理表を縮約し、表 4 のようなノード接続管理表を作成した。また、表 3 のノード管理表には、元のグラフデータと縮約グラフを対応づけるために、縮約グラフ用のノード番号を格納した CID という列を追加した。

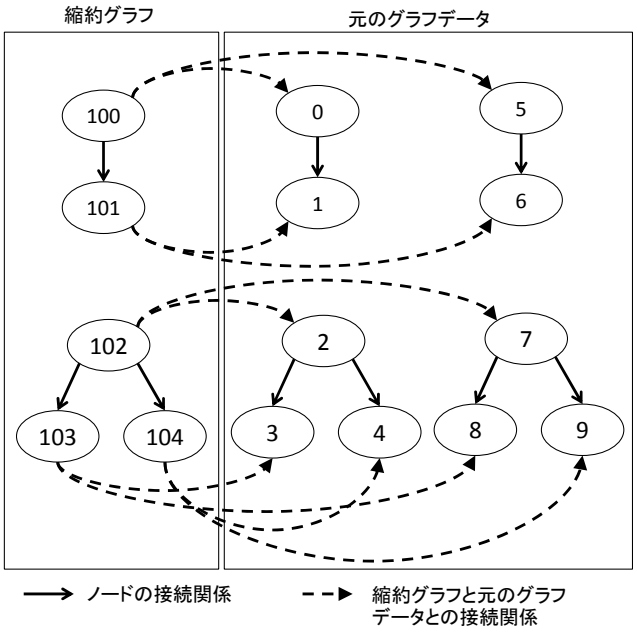


図 2　縮約グラフ

表 3 縮約検索で用いる CID を付与したノード管理表 (CG_NAME 表)

CID	ID	VAL
100	0	“Name0”
101	1	“Name1”
102	2	“Name2”
103	3	“Name3”
104	4	“Name4”
100	5	“Name5”
101	6	“Name6”
102	7	“Name7”
103	8	“Name8”
104	9	“Name9”

表 4 縮約したノード接続管理表 (CG_LINK 表)

SRC	DEST
100	101
102	103
102	104

2.2 グラフ縮約検索の手順

縮約グラフを用いた検索手順は、下記のとおりである。

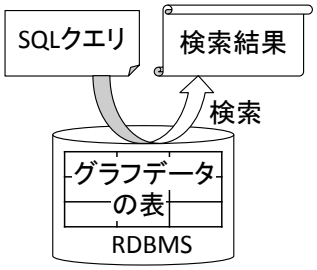
(1) 検索クエリを縮約グラフ向けの検索クエリに変換する。
(2) 縮約グラフ向けの検索クエリを用いて、縮約グラフを検索し、元のグラフデータの検索範囲を限定する情報(CID)を得る。
(3) 検索クエリの条件に、(2)で得た検索範囲を限定する情報(CID)を加える。
(4) (3)で作成したクエリを用いて元のグラフデータを検索する。

縮約グラフを用いた検索の処理時間は、縮約グラフの処理時間と、その結果により検索範囲を絞り込んだ元のデータの検索処理時間を足し合わせた時間である。

縮約グラフのデータサイズは、元のグラフデータよりも小さいため、縮約グラフに対する検索は、比較的短い時間で処理が完了する。元のグラフデータに対する検索は、検索範囲を限定する情報を加えることによって、処理時間を短縮できる。

つまり、縮約グラフの検索時間より、検索範囲を絞り込むことで従来検索から短縮できた時間が多くなることで、高速化の効果を得ることができる。

(a) 通常の検索



(b) グラフ縮約検索

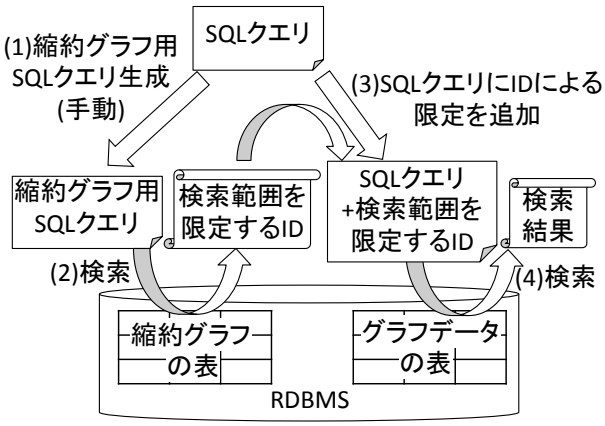


図 3　縮約グラフの検索手順

3. 評価

3.1 評価環境

表 5 の評価環境にて、グラフ縮約検索処理の性能評価を行った。

表 5　評価環境

#	項目	値
1	CPU	Intel Core i7-980 Processor 3.33GHz, 6Core, 12 Thread (Hyper threading)
2	メモリ	24GB
3	ストレージ	300GB (Intel SSDSA2CW300G3)
4	OS	Cent OS 5.7
5	データベース	MySQL 5.0.95

3.2 グラフ形式のデータと検索クエリ

評価では、グラフ形式ならではのデータと、検索クエリを用いた。

1 つ目は、図 4 に示す組織の階層構造を表現したデータである。3 階層の組織の中に、検索対象として 4 階層のデータを 1 件のみ追加した。本データに対し、図 6 に示す 4 階層の深さに存在するデータを検索するクエリを用いた。通常の検索では、1 つずつノードを辿り階層数をカウントする必要があるため、検索処理に時間がかかる。一方、縮約グラフを用いた検索では、縮約グラフ上で指定階層に存在するデータを特定するため、検索処理時間の高速化が期待できる。

2 つ目は、図 5 に示す作業順序を表現したワークフロー図のデータである。ノード間の接続は、直前の作業が完了しないと次の作業が開始できないことを表す。本データには、検索対象として 3 つの作業がループ構造となっているデータを 1 件のみ追加した。このループ構造の作業は、直前の作業が完了しないため、作業を開始できない作業である。通常の検索では、1 ずつノードを辿り、元のノードに戻っていないか確認する処理が必要なため、検索処理に時間がかかる。一方、縮約グラフを用いた検索では、サイズの小さな縮約グラフのノードのみを辿り、ループ構造を検出するため、検索処理時間の高速化が期待できる。本検索で用いた検索クエリは、図 7 である。

上記 2 つのデータ共に、10 万ノード、100 万ノード、1000 万ノード用意し、データを直接検索した場合の検索処理時間と、縮約グラフを用いた検索処理時間を測定した。

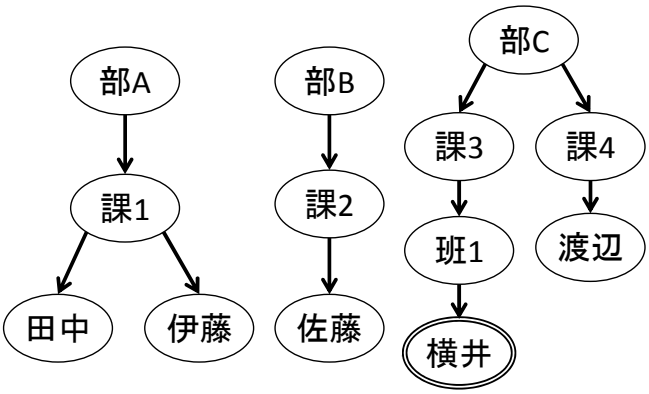


図 4 階層構造のデータ例
(二重線は、クエリにより検索されるノード)

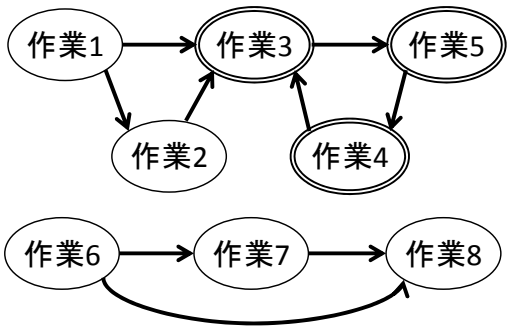


図 5 作業手順のデータ例
(二重線は、クエリにより検索されるノード)

```
select distinct NAME.ID, NAME.VAL
from LINK L1, LINK L2, LINK L3, NAME
where L1.DEST = L2.SRC and
      L2.DEST = L3.SRC and
      NAME.ID = L3.DEST
```

(a) 通常の検索クエリ

```
select distinct NAME.ID, NAME.VAL
from LINK L1, LINK L2, LINK L3, NAME,(
  select distinct CG_NAME.CID AS IDO
  from CG_LINK CG_L1, CG_LINK CG_L2, CG_LINK CG_L3,
       CG_NAME
  where CG_L1.dest = CG_L2.src and
        CG_L2.dest = CG_L3.src and
        CG_NAME.cid = CG_L3.dest ) as SUB
where NAME.CID = SUB.IDO and
      L1.DEST = L2.SRC and
      L2.DEST = L3.SRC and
      NAME.ID = L3.DEST
```

(b) 縮約グラフを用いた検索クエリ

図 6 階層構造データから
4 階層のノードを検索するクエリ

```
select distinct NAME.ID, NAME.VAL
from LINK L1, LINK L2, LINK L3, NAME
where L1.DEST = L2.SRC and
      L2.DEST = L3.SRC and
      L3.DEST = L1.SRC and
      NAME.ID = L1.SRC
```

(a) 通常の検索クエリ

```
select NAME.ID, NAME.VAL
from LINK L1, LINK L2, LINK L3, NAME,(
  select distinct CG_NAME.CID AS IDO
  from CG_LINK CG_L1, CG_LINK CG_L2, CG_LINK CG_L3,
       CG_NAME
  where CG_L1.dest = CG_L2.src and
        CG_L2.dest = CG_L3.src and
        CG_L3.dest = CG_L1.src and
        CG_NAME.cid = CG_L1.src ) as SUB
where NAME.CID = SUB.IDO and
      L1.DEST = L2.SRC and
      L2.DEST = L3.SRC and
      L3.DEST = L1.SRC and
      NAME.ID = L1.SRC
```

(b) 縮約グラフを用いた検索クエリ

図 7 作業手順データから
ループ構造を検索するクエリ

なお、縮約グラフを検索した結果、元グラフを参照すべきデータが複数見つかる場合がある。この際は、各データを並列して参照することで、元データを検索する処理時間を短縮できる可能性があるが、縮約グラフ単体の性能向上効果を確認するために、今回は並列処理を用いずに測定した。

3.3 評価結果

図 8 に評価結果を示す。階層構造のデータから、4 階層のデータを検索するケースにて 6〜6.8 倍の検索性能向上を確認した。縮約グラフには、234 ノードのサイズが存在し、元データと比べサイズが小さいため、比較的短時間で処理が完了した。また、縮約グラフを参照したことで、参照す

べき元データの検索範囲が 234 分の 1 になった。この 2 つの効果により、処理時間が短縮した。

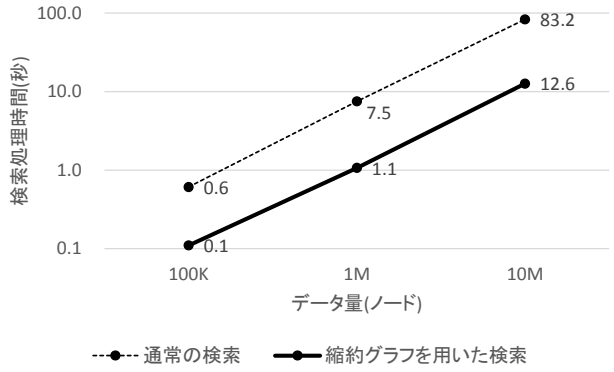


図 8　階層構造のデータに対する性能測定結果

図 9 は、作業手順のデータに対して、ループを検出するクエリを用いて検索した結果である。通常の検索処理時間と縮約グラフを用いた検索処理時間を比較すると、12〜24 倍性能向上した。縮約グラフは、194 ノードのデータサイズであった。また、縮約グラフを参照したことで、元のグラフデータの検索範囲を 3/194 に絞り込めた。作業手順のデータに関しても 2 つの効果により、処理時間を短縮した。

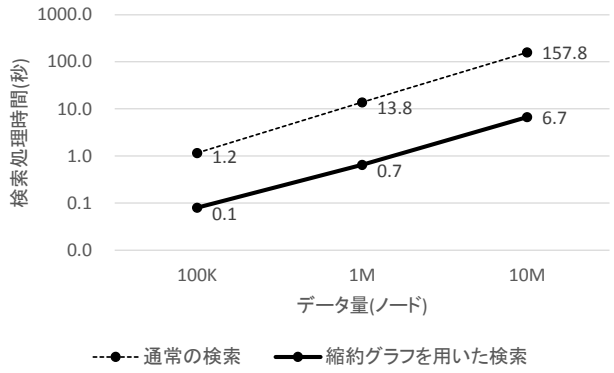


図 9　作業手順のデータに対する性能測定結果

4. 関連研究

これまでに我々は、RDF 形式でデータを格納する RDF ストアにて縮約グラフを用いた検索処理の性能向上について報告してきた[2,3]。今回これらの研究を基とし、より広く普及している RDBMS 上で本技術を用いた評価を行った。

RDF ストアの検索処理を向上されるためには、複数のサーバにデータを分割して格納し、検索を並列に行う方式がよく用いられる。このクラスタ構成での運用は、代表的な RDF ストアである Virtuoso, Apache Jena においても利用できる[5,6]。クラスタ構成を前提とし、検索性能を向上させる研究が、多数報告されている。例えば、Huang らは、クラスタ上のサーバに RDF データを分割しておき、検索の際にクエリの条件節を分割したクエリを並列して検索することで高速化する方式を提案している[7]。本方式ではデータは、近いノードを同じサーバに格納されるように配置するという工夫がされている。しかし、クラスタ構成を用いて検索をした場合、サーバを跨ぐ処理が必要となると、サーバ間の通信がボトルネックとなり、サーバ台数に比例して性能向上させることが難しい。

本研究にて提案した縮約グラフを用いた検索は、縮約グラフを参照することで、検索範囲を限定する方式であるため、1 台のサーバでも検索性能を向上できる方式である。そのため、ハードウェアを大きく増強することなく、性能向上が可能である。また、縮約グラフを検索し、参照すべき元データの範囲が複数得た場合は、各範囲を並列して検索できるため、クラスタ構成とも相性がよい。

5. 結論

本研究によって、縮約グラフを用いることで、検索処理性能を最大 24 倍向上できることを確認した。縮約グラフは、データ量が多いケースや、ノードの接続が多様であるケースで有利という特徴がある。

データ量が多いケースで特徴を確認するために、今回 3 つのデータサイズで測定した。これによって、検索対象のデータのサイズが増加しても、一定の性能向上の効果があることが分かった。その要因は、縮約グラフの検結果により、限定された参照が必要なデータ量の割合が、全てのデータサイズで一定であり、性能向上効果もデータサイズに依存していないためであると考えられる。この特徴は、グラフ形式のデータベースを、大量データの格納に対応させるために、重要である。

ノードの接続が多様であるケースでの評価結果は、示してしないが、縮約グラフを用いた検索では、データベースに様々なノードの繋がりを持ったグラフ形式のデータが入っているほど、性能向上効果が高くなるという特徴がある。なぜなら、ノードの繋がりのパターンが多いほど、縮約検索によって検索範囲を絞り込める範囲が狭くなり、元のグラフデータを検索する処理時間を短縮できるためである。

今後は、DBpedia[8]などのノードの接続関係が多様なグラフ形式のデータに対するベンチマークが必要である。技術面では値の範囲による絞り込み、検索処理の並列化など更なる性能向上ができる仕組みを用い、ベンチマークを今後行う必要がある。また、データベースには常にデータ更新が行われるため、縮約グラフの差分作成処理など処理を効率化する方法も検討していく予定である。

参考文献

[1] Apache Jena, http://jena.apache.org/ (2014)
[2] 千代英一郎, 宮田康志, 西山博泰, “グラフ縮約にもとづく SPARQL クエリ並列化方法の設計および予備評価”, 情報処理学会論文誌, 53 巻 12 号 (2012)
[3] 千代英一郎, 宮田康志, 西山博泰, 横井一仁, “値域分割にもとづく SPARQL クエリ分散処理方法 “, コンピュータソフトウェア, 30 巻 3 号 (2013)
[4] Oracle Database 12c Oracle Spatial and Graph Option, http://www.oracle.com/technetwork/database-options/spatialandgraph/overview/rdfsemantic-graph-1902016.html (2014)
[5] Virtuoso Universal Server, http://virtuoso.openlinksw.com/ (2014)
[6] Alisdair Owens, Andy Seaborne, Nick Gibbins, Schraefel, Mc, “Clustered TDB: A Clustered Triple Store for Jena”, Technical report, University of Southampton (2008)
[7] Jiewen Huang, Daniel J. Abadi, Kun Ren, “Scalable SPARQL Querying of Large RDF Graphs”, Proceedings of International Conference on Very Large Data Bases (2011)
[8] DBpedia, http://ja.dbpedia.org/ (2014)