Vol.2020-DBS-172 No.9
2020/12/22

複雑ネットワークにおける効率的な反復走査のための経路索引構築法

楠 和馬[1,2,a)]　波多野 賢治[1,b)]

概要：グラフ問合せにおいて，部分グラフを抽出することは必要不可欠な処理であり，その処理の高速化が必須になる．本研究では，グラフ問合せ内のパタン指定によく利用され，グラフデータベースにおける問合せ計画の 1 過程である，反復的な経路導出の効率化を目的とする．現在までに効率的な部分グラフ抽出手法として，経路索引構築やグラフ走査手法が提案されてきたが，どちらか一方の手法では現実的なグラフ管理において，辺の次数が外れ値的に多いハブから受ける影響により非効率な処理が発生する．本研究では，隣接節点を効率的に走査可能なネイティブグラフ処理とハブを起点とする反復経路の索引を組み合わせることにより，効率的に反復的なグラフ走査を可能にする経路索引構築法を提案する．

キーワード：グラフデータベース, 経路索引, インデックスフリー隣接性, グラフ走査

1. はじめに

あらゆる分野において多種多様な関係を持つデータ（ネットワーク）が扱われている．例えば，ソーシャルネットワークや論文の共著・引用関係などが存在する．このようなネットワークに対して実体間の関係性に着目したデータ問合せや分析が行われている [1]．このようなネットワークを表現可能なデータ構造にグラフがあり，実体を表現する節点と関係性を表現する辺で構成される．節点と辺のようなグラフの基本的な構成要素だけではなく，データに応じて多様な形状のグラフが提案されてきた [2]．

世の中に普及しているデータベースシステムには，リレーショナルデータベース (RDB) が存在し，表形式のデータモデル（リレーション）によるグラフ管理方法が確立されてきた [3]．しかし RDB 上での管理では，実体間の関係性を辿る処理（グラフ走査）は基本的に辺を格納したリレーションに基づいて，リレーション同士の自己結合を実行するため高コストになる．

グラフ問合せの特性を考慮して，汎用的にグラフを管理が可能で，効率的にグラフ走査を実行可能なグラフデータベース (GDB) が多数開発され研究されている [4]．GDB は各節点が隣接節点へのポインタを持つという特性（インデックスフリー隣接性）を持ち，この特性によりデータ量に影響を受けず一定のコストでグラフ走査が可能になる．

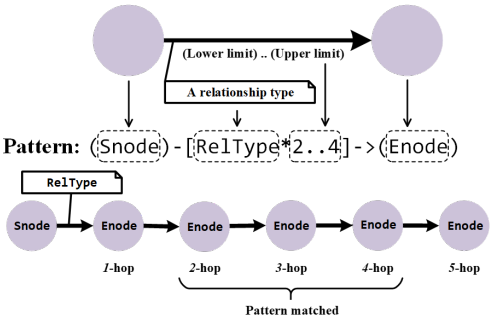


図 1: 問合せ内の反復的なグラフ走査とその反復経路

しかし，インデックスフリー隣接性はどのような節点に到達可能か把握していないため，GDB は節点に接続されている辺の数（節点次数）に関係なくグラフ走査を行う．また，反復的にグラフ走査を行えば，次数の多い節点に到達する機会も増えるため，節点次数を考慮できていないことは非効率性にもつながる．本研究では，反復的なグラフ走査により得られる部分グラフを反復経路と呼ぶ．グラフ問合せで節点・辺の組合せ（パタン）を指定する際に，反復経路は図 1 のように “`-[RelType*2..4]->`” と表記する．このパタンは，`RelType` という関係性の辺（関係辺）を 2 から 4 回繰り返しグラフ走査をする，という指定になる．

ネットワーク科学では，節点次数が外れ値的に多い節点はハブと呼ばれており [1]，実社会のネットワークではハ

[1] 同志社大学
Doshisha University
[2] 日本学術振興会 特別研究員 DC2
Research Fellow of Japan Society for the Promotion of Science
[a)] kkusu@acm.org
[b)] khatano@mail.doshisha.ac.jp

ブは全体の中は極少数である．しかし，現実世界のネットワークにはスモールワールド性という性質があり [5]，反復的にグラフ走査を行えばどの節点からでもほとんどの節点に到達可能になることが知られている．また，グラフ問合せにおいては関係性指向の問合せ性質上，反復的なグラフ走査はあらゆる分野のネットワークにおいて必要とされる．そのため，反復的なグラフ走査おいてハブから受ける処理コストへの影響が大きいことは容易に推測できる．

そこで本研究では，ハブか否か区別して管理し，索引に基づくハブを起点とする反復経路の管理方法を提案する．また，ハブに対する処理を回避しながらグラフ走査を可能にするために，インデックスフリー隣接性と反復経路索引を組み合わせた走査アルゴリズムも提案する．

2. 関連研究

グラフ走査の高速化のために，グラフの管理方法や索引に関する研究が行われてきた．本研究に大きく関わるインデックスフリー隣接性を持つグラフ管理方法と，特定パタンの経路索引方法について紹介する．

2.1 インデックスフリー隣接性とネイティブ GDB

ネイティブ GDB (NGDB) は，各節点が隣接節点への小さな索引を持つようにグラフ管理を行う GDB であり，インデックスフリー隣接性を持ち合わせている．NGDB 索引計算が必要なく，その節点の次数に依存したコストでグラフ走査を実現している．そのため，グラフ走査コストはデータ量の影響を受けにくく，膨大なグラフにおいても一定のコストで問合せが可能になる [6]．

しかし，各節点が保持しているのは隣接節点リストのみであることから，隣接していない節点は把握できていない．また，愚直に隣接節点リストに基づいてデータを次々と問い合わせるため，ハブのような次数が多い節点に対しても同様に隣接節点リストに従ってグラフ走査せざるを得ない．

以上のように，インデックスフリー隣接性は局所的な問合せに有効であるが，節点群を起点とした反復的なグラフ走査のような大域的な問合せに弱い特性を持つ．

2.2 経路索引

逐次的に節点を辿る方法ではなく，問合せ頻度に基づき特定のグラフパタンに対して索引を構築する手法も存在する [7, 8]．GDB のデータ管理事例には，一つの巨大なグラフを管理する場合と膨大な数の小さいグラフを管理する場合に分類できる [7]．前者の管理事例に該当する現実のネットワークはソーシャルネットワークや論文の共著・引用関係が挙げられ，後者に該当する事例は化学構造やタンパク質構造が挙げられる．従来のグラフ索引は膨大な数の小さいグラフの中から特定のパタンを検索することに焦点が当てられてきた．一つの巨大なグラフを管理する場合，データ量が大きくなるほど辺の数も増加するため，各節点の節点次数が増える傾向にある．それに伴い，そのグラフ索引は膨大に成長し，問合せ対象となるパタンの導出コストはデータ量に依存することになる．

2.3 本研究の目標

部分グラフ抽出の従来手法は，2.1 項と 2.2 項に述べた通り，隣接節点リストに基づくグラフ走査や経路索引によって実現されている．一つの巨大なグラフ管理の場合には，インデックスフリー隣接性の特性を活かし局所的に部分グラフ抽出を実施することが効率的ではあるが，NGDB は，膨大な次数を持つハブに起因する非効率性を考慮できていない．一方，同じ状況で経路索引は，膨大なデータ量に影響を受けて索引対象が増加するため，索引構築やグラフ走査が非効率になる．このように，膨大な一つのグラフを管理する場合に，どちらか一方の方法では大域的なグラフ走査において非効率な点がある．

したがって本研究では，非効率な処理の原因となるハブの存在を考慮することにより，反復経路に対するグラフ走査の高速化を目的とする．隣接節点リストに基づくグラフ走査におけるハブに対する処理を回避する必要があるため，グラフの中でハブを対象に反復経路を収集することで，索引サイズを大きくしすぎず，索引を参照することでハブに対する処理を抑制したグラフ走査法を提案する．

3. 提案手法

汎用的なグラフ管理を可能にする GDB において，2.3 項で述べた通り，非効率性の原因であるハブの情報も把握するべきである．そのため本研究では，ハブからの反復経路を収集することにより，効率的に反復経路を取得するための索引構造を提案する．

したがって 3.1 項では，GDB に格納されたグラフの中でどの節点がハブであるかを決定する方法について説明する．次に 3.2 項では，ハブからの反復経路を事前に収集し，グラフ走査時に反復経路を取得可能な索引の構築方法について説明する．3.3 項では，3.1 項と 3.2 項で説明した索引を利用して反復経路の導出方法について説明する．

3.1 ハブの決定方法

節点群のうちハブがどれかを決定するためには，実用上で繰り返し走査される関係辺を検討する必要がある．そのため，関係性を一種類だけ選択し，関係辺の連なりである反復経路を収集する．

関係性を選択した後に，次のようにハブを決定する．

(1) 該当する関係辺を GDB から取得
(2) 該当する辺に接続されている節点群を取得
(3) 各節点の辺の接続数（次数）を計上
(4) x 軸を節点次数，y 軸を各節点次数の発生確率とする

次数分布を作成
(5) ハブとそれ以外の節点を区別するための閾値を決定
(1) から (4) の間では，対象の関係辺で構成された部分グラフの次数分布を作成する．最後の (4) において，各節点がハブであるか否かを区別するために，次数に対する閾値は，パレートの法則 [9] に従って次数の多い節点から上位 20 %とした．このようにベキ分布は，パレートの法則に基づいて分析対象の細分化が行われる．パレートの法則はネットワーク科学においても現実ネットワークに対して存在することは報告されている [10]．その結果，n が部分グラフ内の節点数としたとき，節点次数を計上するため，上記の時間計算量は $\mathcal{O}(n)$ である．

3.2 ハブを起点とする反復経路の管理方法

NGDB は到達済みもしくは隣接節点に関する情報のみを参照できることを考え，反復経路に対するグラフ走査中に利用可能な索引を構築する必要がある．NGDB は到達した節点の固有 ID や何回辺を辿ったか把握しているため，任意長の反復経路を取得できるように収集した反復経路をその長さで分けて管理する．

ハブを起点とする反復経路索引の構築アルゴリズムを Algorithm 1 に示す．指定されたハブから反復経路の収集を実行するために，グラフ走査中に取得可能な 2 種類の入力データが必要である．一つは 3.1 項で決定したハブ集合であり，もう一つは，実用上必要となる最大走査回数である．また，collectRecursivePaths 関数を定義しており，Algorithm 1 の 10 行目で，ハブを起点として幅優先探索（BFS[*1]）ベースのグラフ走査 traverse を実行する．traverse は BFS ベースのグラフ走査であるため，巡回グラフを効果的に取り扱うことができ，到達済の節点に対する処理を回避する．さらに 10 行目で，収集した反復経路をその経路長ごとに分類する．

以上の処理を各ハブに対して行い，Algorithm 1 の 3 行目のように，各ハブの ID ごとに分類する．したがって，ハブ ID と経路長の対を指定することで，各ハブからの反復経路を即座に取得可能になる．

3.3 経路索引を利用したグラフ走査方法

本研究で提案した反復経路索引を利用したグラフ走査アルゴリズムについて説明する．節点次数が少ない節点の場合は隣接節点リストに基づくグラフ走査が適しており，ハブに対しては隣接節点リストの参照を避ける必要がある．したがって，ハブに到達するまで隣接節点リストに基づき辺を辿り，ハブに到達した際にハブ ID と経路長を指定することで反復経路を経路索引から取得する．Algorithm 2 は

[*1] David Eppstein: Breadth First Search algorithm (Python). https://www.ics.uci.edu/~eppstein/PADS/BFS.py, 2020 年 11 月 7 日閲覧.

Algorithm 1 ハブを起点とする反復経路の索引構築

Input H: ハブ集合, $maxhop$: 最大反復経路長.
Output I: 反復経路索引

```
 1: I ← {}
 2: for each h ∈ H do
 3:     I[h] ← collectRecursivePaths({h}, maxhop)
 4: end for
 5: return I
 6: function collectRecursivePaths(P_rp)
 7:     hop2rp[0] ← P_rp
 8:     hop ← 1
 9:     for hop ← 1 to maxhop step by 1 do
10:         hop2rp[hop] ← traverse(hop2rp[hop - 1])
11:     end for
12:     return hop2rp
13: end function
14: function traverse(P_rp)
15:     P_rp ← {}
16:     for each rp ∈ P_rp do
17:         N_visited ← rp.nodes
18:         N_adj ← rp.adjacentNodes
19:         for each n_adj ∈ N_adj do
20:             if n_adj ∉ N_visited then
21:                 rp.addNode(n_adj)
22:                 P_rp.append(rp)
23:             end if
24:         end for
25:     end for
26:     return P_rp
27: end function
```

Algorithm 2 反復経路索引を利用したグラフ走査法

Input I: 反復経路索引, H: ハブ集合, N_s: 起点節点 ID, hop: 走査回数
Output P: 結果集合

```
 1: h ← 0
 2: P ← N_s
 3: P_pending ← [ ]
 4: for 1 to hop step by 1 do
 5:     P ← BFS(P)
 6:     for each rp ∈ P do
 7:         if rp.endNode ∈ H then
 8:             P_pending.append(rp)
 9:             P.delete(rp)
10:         end if
11:     end for
12: end for
13: for each rp ∈ P_pending do
14:     hub ← rp.endNode
15:     hop ← hop − rp.length
16:     rp_I ← I.get(hub, hop)
17:     rp ← rp.join(rp_I)
18:     P.append(rp)
19: end for
```

ハブに対する非効率な処理の抑制を考慮しながら，任意の節点からのグラフ走査を可能にする．

Algorithm 2 の 4 から 12 行目では，隣接節点リストに基づいて開始節点集合 N_s から一つずつグラフ走査を行

う．問合せで指定された回数まで反復的に辺を辿る最中にハブに到達した場合，13 から 19 行目でまとめて索引により経路を導出するために走査を中断し，$P_{pending}$ へ追加する．隣接節点リストに基づくグラフ走査が完了した後，$P_{pending}$ に追加された経路からハブ ID と経路長を取得し，索引を利用して残りの反復経路を取得する．以上の処理で，反復経路に対するグラフ走査を実現する．

4. 評価実験

本研究では，反復経路に対するグラフ走査の効率性を評価するために，既存の NGDB との比較実験を行う．比較対象とする GDB には，グラフ走査が高速である [11] Neo4j ver. 4.0.0 を利用する．また実験環境は，OS が CentOS 7.6.1801 (x86-64)，CPU が Intel Xeon Silver 4114 (2.20 GHz, 10 core) × 2，RAM が 256 GB である．

4.1 ベンチマーク

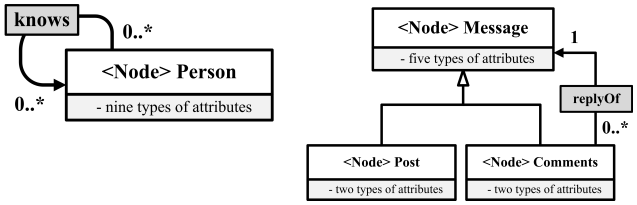
本実験では，反復経路に対するグラフ走査の処理時間とそのスケール性を評価する．したがって，あらゆるデータ量のグラフを生成可能な，Linked Data Benchmark Council (LDBC) が開発した Social Network Benchmark (SNB) を利用する．

LDBC SNB データセットは，ソーシャルネットワークをモデル化したデータセットであり，スキーマの詳細はマニュアル [12] に記載されている．また，LDBC SNB は，スケールファクタ (SF) に応じてデータ量を調整可能であり，GDB のスケーラビリティを評価することができる．さらに，Online Transaction Processing (OLTP) と Online Analytical Processing に対する性能評価用の問合せ例が二種類用意されており，それぞれ Interactive Workload [13] と Business Intelligence Workload [14] である．本実験では，グラフ走査は OLTP に該当するため，Interactive Workload を参考にするが，この Workload には二つのタイプがある．一つは短い経路を問い合わせる Interactive Short (IS) と，もう一つは長い経路の問合せであり，グラフ構成要素の属性に対する条件や集約関数を含む問合せを提供する Interactive Complex (IC) である．IS は反復経路の問合せ例が無いため，問合せ 14 例の内 12 例に反復経路を含む IC のクエリを参考に実験を設定する．

この実験を準備するために，次のタスクを実行した．

- SF 値が 1, 10, 100 の LDBC SNB データセットを，データセット生成プログラム[*2]を用いて作成，
- 格納およびベンチマーク実行用プログラム[*3]を用いて，データセットを Neo4j へ格納，
- IC の問合せに指定されている反復回数を参考にし，索引構築には 10-hop までの反復経路を収集．

[*2] LDBC: ldbc/ldbc_snb_datagen.git, https://github.com/ldbc/ldbc_snb_datagen, 2020 年 11 月 7 日閲覧.
[*3] LDBC: ldbc_snb_implementations.git, https://github.com/ldbc/ldbc_snb_implementations, 2020 年 11 月 7 日 閲覧.



(a) Person-knows-Person (b) Message-replyOf-Message

図 2: LDBC SNB 内に含まれている 2 種類の反復経路

表 1: Person，knows，ハブの数

SF	# Person	# knows	# ハブ
1	7,731	180,623	1,832
10	49,897	1,938,516	12,474
100	344,926	19,941,198	86,231

4.2 反復経路に対するグラフ走査性能の評価方法

グラフ走査の高速化が行えているか確認するため，無作為に選択した 200 個の節点に対して反復的なグラフ走査を実行する．この比較実験を通じて，隣接節点リストに基づくグラフ走査と，提案手法によるグラフ走査のどちらがより効率的かを示す．

IC の問合せは，図 2 に示すように，2 種類の反復経路を含んでいる．一つ目の反復経路は，図 2(a) に示したように，2 人のユーザが知り合いか否かを表す， Knows 関係辺で構成されている．もう一方の反復経路は，Post や Comment に対する返信を表す，ReplyTo 関係辺で構成されている．本実験では，多対多の関係である Knows による反復経路は使用するが，1 対多の関係である ReplyTo 関係による反復経路は使用しない．この理由は，図 2(a) に示すように，Message 節点の中の Comment 節点のみが ReplyTo 関係を持つことにより複雑ではなくなるためである．

したがって本実験では，Person-knows-Person パタンのみを用いて次数分布を作成した．各 SF ごとに作成した次数分布を図 3 に示す．破線はハブか否かを区別する閾値であるため，赤丸，黒丸に該当する節点次数をそれぞれハブとその他の節点として扱うことにした．

4.3 結果および考察

本研究の提案手法はハブの決定やそのハブからの反復経路の収集を行う必要があるため，その収集に費やした時間を計測した．表 2 に，索引構築に費やした時間を示す．

次に，反復経路に対するグラフ走査において，本研究で提案した経路索引の有効性を示すため，グラフ走査時間を計測した．図 4(a) は，反復的グラフ走査の平均消費

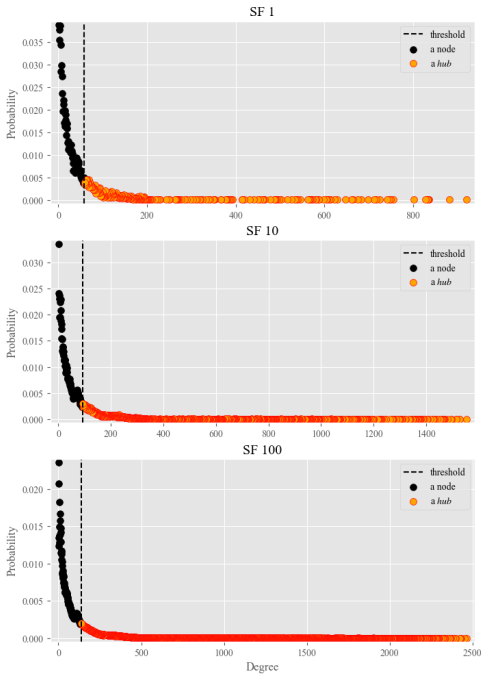


図 3: LDBC SNB の次数分布 (SF = 1, 10, 100)

表 2: 反復経路索引の構築時間 [sec.]

SF	ハブの決定	反復経路の収集	累計時間
1	1.89	1.01	2.90
10	4.94	4.53	9.47
100	38.12	35.71	73.83

時間を n-hop 毎に示しているため，x 軸は n-hop，y 軸は n-hop 毎の消費時間 [ms] の平均値である．図 4(b) は，n-hop ごとに消費時間の比率を示しており，ベースラインと比較して，提案手法がグラフ走査時間をどの程度短縮しているかを確認するのに役立つ．ベースラインは索引を利用せずにグラフ走査を行った場合であり，この手法により費やした時間を基準としたため，全て 1 である．

図 4 の SF 1 では，全 Hop においてグラフ走査速度を劣化させる結果となっているが，これはデータ量が小規模であることにより，ハブか否かを決定する節点次数に対する閾値が小さすぎることが原因である．実際に SF 10 以降は，グラフ走査の高速化が実現できており，SF 10 の 3-hop 以上で 1.18 倍，SF 100 の 3-hop 以上で 1.19 から 1.38 倍の高速化を達成できた．以上のことから，本研究の提案手法は，データが大きいほど反復経路に対するグラフ走査の性能向上に貢献している，といえる．

しかし，SF 1 で起きた非効率性の問題は，他の SF でも例外なく同様の問題が発生する可能性が大いに考えられる．ハブの選定基準は，簡易的にパレートの法則を採用したことで，SF 1 では節点次数が少なくなるため，隣接節点リストに基づく反復経路に対するグラフ走査は低コストになる．また Algorithm 2 は，グラフ走査の効率化を実現することができたが，より効率的に実行するためには，コストベースのアルゴリズムを定義し，隣接節点リストに基づくグラフ走査と反復経路索引を使い分けることが必要である．また図 4(a) に示すように，反復経路をグラフ走査する消費時間は，5〜6-hop 付近で増加が止まり，6-hop 以上で飽和している．ネットワーク科学分野では，ソーシャルネットワークにおける 6 次の隔たりと呼ばれる仮説を実証している [1, 10]．図 4(a) に示す実験の結果は，BFS による反復的にグラフ走査を行っている間は，6-hop 付近でほぼ全ての節点が到達済みになるという仮説に沿ったものである．このように，未到達節点が残り少なくなった場合には，本研究で提案した隣接節点リストに基づくグラフ走査と反復経路索引の二つの計算コストが均衡している可能性が考えられる．さらに，SF 100 のデータセットを対象とした実験では，図 4(b) に示すように，グラフ走査時間は SF 1 や SF 10 に比べて不安定であった．これは，反復経路索引がインメモリベースの実装であり，SF 100 においては残された少量のメモリ空間で計算したため非効率になったことが考えられる．最終的には，反復経路を効率的にグラフ走査するためのメモリ空間が残らなかったため，メモリ効率の良い索引構造を考える必要がある．

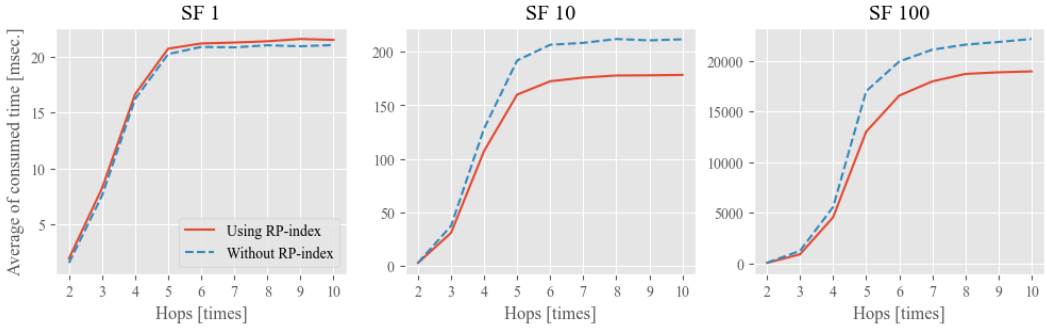
5. おわりに

本研究では，NGDB が反復的にグラフ走査を行う際に，節点次数を考慮していないことで引き起こされる非効率性に着目し，その問題を解決するために反復経路索引の提案を行った．また，その索引を利用したグラフ走査方法も提案し，ハブに対する処理を回避できるようにした．評価実験では，グラフ走査時間を計測し，既存 GDB と処理性能を比較した結果，18% から 38% の高速化を実現できた．

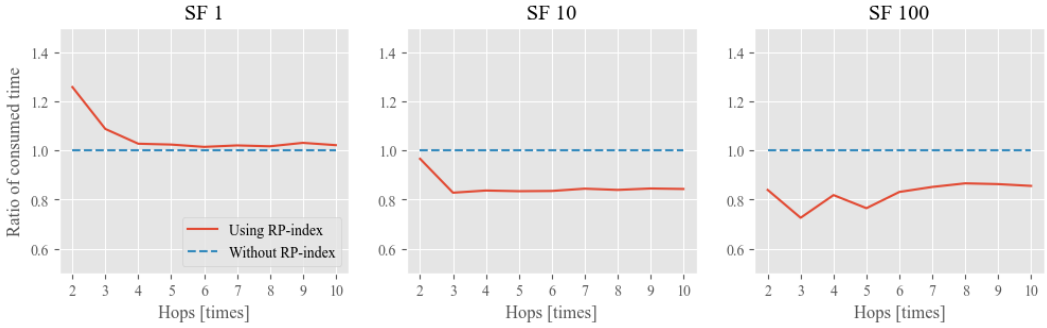
今後の課題として，反復経路索引やグラフ走査アルゴリズムの拡張を行う必要があることについて 4.3 項で述べた．具体的には，以下のような拡張が必要だと考えている．

- 経由されやすさを考慮したハブの決定法
- 反復経路索引の圧縮手法
- コストベースグラフ走査法
- 複数の節点からグラフ走査時の索引計算法

本研究で提案した反復経路索引の構築で重要なハブの決定方法について，パレートの法則により簡易に決めた閾値ではなく，他の節点から到達されやすいかどうかといった観点から，ハブの選定方法を調整する必要がある．また，メモリ上での反復経路索引の圧縮方法を提案し，メモリ上でグラフ走査の処理をより効率的に行えるようにする必要がある．さらに，隣接節点リストに基づくグラフ走査と反復経路索引の計算コストを事前に見積もり，どちらの処理を行うべきか推定した計算コストに基づき切り替える必要がある．加えて今回の Algorithm 2 は，一つの開始節点から処理を行う方法であるが，複数の開始節点から反復的に



(a) 平均グラフ走査時間



(b) 平均消費時間の比率

図 4: 反復経路に対するグラフ走査時間

グラフ走査を行えば同一のハブに到達する可能性が高いため，同一のハブに到達する場合は最終的にまとめて処理を行えるように拡張することで高速化を図る必要がある．

謝辞 本研究の一部は日本学術振興会特別研究員奨励費（課題番号：JP19J15498），JSPS 科研費 JP18H03342, JP19H01138 の研究助成を受けて遂行された．

参考文献

[1] Barabási, A.-L. and Pósfai, M.: *Network Science*, Cambridge University Press (2016).

[2] Rodriguez, M. A. and Neubauer, P.: Constructions from Dots and Lines, *Bulletin of the American Society for Information Science and Technology*, Vol. 36, No. 6, pp. 35–41 (online), DOI: 10.1002/bult.2010.1720360610 (2010).

[3] Celko, J.: *Joe Celko's Trees and Hierarchies in SQL for Smarties*, Morgan Kaufmann, second edition edition (2012).

[4] Sakr, S. and Pardede, E.: *Graph Data Management: Techniques and Applications*, IGI Global (2011).

[5] Newman, M.: *Networks: An Introduction*, Oxford University Press (2010).

[6] Sakr, S. and Pardede, E.: The Graph Traversal Pattern, *Graph Data Management: Techniques and Applications* (Rodriguez, M. A. and Neubauer, P., eds.), IGI Global, chapter 2, pp. 29–46 (online), DOI: 10.4018/978-1-61350-053-8.ch002 (2012).

[7] Sakr, S. and Al-Naymat, G.: The Overview of Graph Indexing and Querying Techniques, *Graph Data Management: Techniques and Applications* (Sakr, S. and Al-Naymat, G., eds.), IGI Global, chapter 3, pp. 71–88 (online), DOI: 10.4018/978-1-61350-053-8.ch004 (2012).

[8] Luaces, D., Viqueira, J. R., Pena, T. F. and Cotos, J. M.: Leveraging Bitmap Indexing for Subgraph Searching, *22nd International Conference on Extending Database Technology (EDBT)*, OpenProceedings.org, pp. 49–60 (online), DOI: 10.5441/002/edbt.2019.06 (2019).

[9] Pareto, V. F. D.: La courbe des revenus, *Cours d'économie politique*, Vol. (II), No. (Ⅲ), Librairie Droz, chapter (Ⅰ), pp. 299–345 (1964).

[10] Barabási, A.-L. and Frangos, J.: *Linked: The New Science Of Networks Science Of Networks*, Perseus Books Group (2002).

[11] Lissandrini, M., Brugnara, M. and Velegrakis, Y.: Beyond Macrobenchmarks: Microbenchmark-based Graph Database Evaluation, *Proceedings of the VLDB Endowment*, Vol. 12, No. 4, pp. 390–403 (online), DOI: 10.14778/3297753.3297759 (2018).

[12] Linked Data Benchmark Council: *LDBC The graph & RDF benchmark reference The LDBC Social Network Benchmark (version 0.3.1)*.

[13] Erling, O., Averbuch, A., Larriba-Pey, J., Chafi, H., Gubichev, A., Prat, A., Pham, M.-D. and Boncz, P.: The LDBC Social Network Benchmark: Interactive Workload, *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, ACM, pp. 619–630 (online), DOI: 10.1145/2723372.2742786 (2015).

[14] Szárnyas, G., Prat-Pérez, A., Averbuch, A., Marton, J., Paradies, M., Kaufmann, M., Erling, O., Boncz, P., Haprian, V. and Benjamin, J. A.: An Early Look at the LDBC Social Network Benchmark's Business Intelligence Workload, *Proceedings of the 1st ACM SIGMOD Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA)*, GRADES-NDA '18, ACM, pp. 9:1–9:11 (online), DOI: 10.1145/3210259.3210268 (2018).