DBSJ Japanese Journal
Vol. 15-J, Article No. 12, March 2017

外部記憶アルゴリズムを用いた大規模グラフデータからのスキーマ抽出手法[1]

An Algorithm for Extracting Schemas from External Memory Graphs

関根 吉紀♡ 池田 光雪♢ 鈴木 伸崇♠

Yoshiki SEKINE Kosetsu IKEDA Nobutaka SUZUKI

本論文では，グラフデータからスキーマ抽出を行うための外部記憶アルゴリズムを提案する．ここで，外部記憶アルゴリズムとは，主記憶上での処理が困難な大規模なデータを効率良く処理するためのアルゴリズムのことをいう．提案アルゴリズムは，増分クラスタリング法（**incremental clustering method**）に基づいて，各ノードが属するクラスの抽出と，それらクラス間のエッジを抽出することにより行われる．評価実験では，使用したデータは限られているものの，入力データサイズに対してほぼ線形の実行時間でスキーマ抽出が可能であり，スキーマ抽出中の主記憶使用量は入力データサイズよりも十分小さいという結果が得られた．

In this paper, we propose an algorithm for extracting a schema from a large graph. In order to handle large graphs, our algorithm is designed as an external memory algorithm. Our algorithm is based on the incremental clustering method and consists of two steps: extracting classes of nodes and extracting edges between classes. The experiments show that the execution time is almost linear and the memory usage of the schema extraction is much smaller than the input graph size.

1. はじめに

データを保存，管理する一般的な手段として関係データベース（RDB）がある．RDB を使用するときには，データを格納する前に必ずスキーマを設計し，そのスキーマと一致するようにデータを格納しなければならない．スキーマはこのようにデータの構造を規定するほか，ユーザが問合せ式を記述する際の手助けとなる，問合せ処理系の問合せ実行効率を向上させるという 2 つの重要な恩恵を与える．

一方，人間同士の関係や交通ネットワークのような，RDB では扱いにくいデータを保存，管理する手段としてグラフデータが広く用いられている．グラフデータにおいては RDB のようなスキーマを事前に定めることはほとんどないため，スキーマが付与されているものは稀である．しかし，そのようなグラフデータにおいても適切にスキーマを抽出できれば先述の恩恵を受けることができる．たとえば，正規経路問合せ式 q に対して，q とスキーマとの積オートマトン q' が得られれば，q' は q よりも効率よく実行できる [2].

近年，計算機の処理能力の向上や HDD 等の外部記憶装置の低価格化と大容量化によって，計算機上で処理・蓄積されるデータが大幅に増加している．このため，主記憶のサイズを大幅に超えるデータを処理することのできるアルゴリズムが必要とされている．しかし，これまで提案されてきたスキーマ抽出アルゴリズムは，データを主記憶に格納し処理することを前提としているため，データが非常に大きい場合には適用困難である．

そこで本研究では，サイズが大きく全体を主記憶上で処理できないグラフデータに対応したスキーマ抽出手法を提案する．本手法においてスキーマ抽出は，各ノードが属するクラスの抽出と，それらクラス間のエッジを抽出することにより行われる．具体的には，増分クラスタリング法（incremental clustering method）を用いた作成法である先行研究 [3] のアルゴリズムを外部記憶アルゴリズムとして拡張し，さらにクラスの抽出法にも改良を加える．先行研究では，グラフデータも含む，スキーマ抽出に必要なすべての情報を主記憶に格納し処理していたのに対し，本研究では，グラフデータをシーケンシャルに読み込み，必要なデータのみを主記憶上に置き逐次的にクラス抽出を行う．また，スキーマ抽出に必要だが主記憶に乗り切らない情報は外部ファイルに書き出し，それらも同様にシーケンシャルに読み込んで分析・処理することによってスキーマ間のエッジ抽出を実現している．外部記憶アルゴリズムは通常 C 言語のような比較的低水準な言語で実装されることが多いが，このようにデータを処理することで，Ruby のような高水準言語でも大規模データ処理を実現できる．評価実験では，使用したデータは限られているものの，入力データサイズに対してほぼ線形の実行時間でスキーマ抽出が可能であり，スキーマ抽出中の主記憶使用量は入力データサイズよりも十分小さいという結果が得られた．

グラフデータからスキーマを抽出する先行研究として，グラフ上のあるノードから同じパスで到達する 1 つ以上のノードを，スキーマの 1 つのノードとして写像してスキーマを求めるもの [4]，その近似解を求めるもの [5]，各ノードについて，接続しているエッジラベルが類似しているノードを 1 つのクラスにまとめるもの [3]，グラフの要約を求めるもの [6] などがある．[4] は処理コストが高く，閉路を含むグラフに対応困難である．[5] は [4] に比べて処理コストが低いが，閉路を含むグラフに対応できない場合があり，近似解しか得られない．[3] はパスの探索を行わないため，[4] や [5] に比べて効率がよく，閉路を含むグラフにも対応可能である．[6] はエッジのラベルや向きがないグラフデータに対する手法である．以上のアルゴリズムはいずれも外部記憶アルゴリズムではないため，主記憶のサイズを超えるデータは扱うことができない．グラフに関する外部記憶アルゴリズムとしては，グラフの強連結成分を求めるもの [7]，到達可能性を判定するもの [8]，三角形分割問題を解くもの [9] がある．しかし，グラフデータからスキーマを抽出する外部記憶アルゴリズムは我々の知る限り存在しない．

本論文の構成は以下の通りである．第 2 章ではグラフ関連の定義を述べる．第 3 章では提案手法について述べる．第 4 章では評価実験について述べる．第 5 章では本論文のまとめを述べる．

2. 諸定義

本章では，本研究で扱うグラフとそのスキーマに関する定義について述べる．

2.1 グラフ

ラベル付き有向グラフ（以下，単にグラフ）を $G = (V, E)$ と表す．ここで V はノードの集合，E はラベル付き有向エッジの集合である．エッジは両端にノードをもち，それらを端点と呼ぶ．ノード $u \in V$ を始点とし，ノード $v \in V$ を終点とする有向エッジ $e \in E$ の

[1] 本論文の一部は [1] に基づく．

♡ 学生会員　筑波大学大学院図書館情報メディア研究科
ysekine@klis.tsukuba.ac.jp
♢ 正会員　千葉大学アカデミック・リンク・センター
lumely@chiba-u.jp
♠ 正会員　筑波大学図書館情報メディア系
nsuzuki@slis.tsukuba.ac.jp

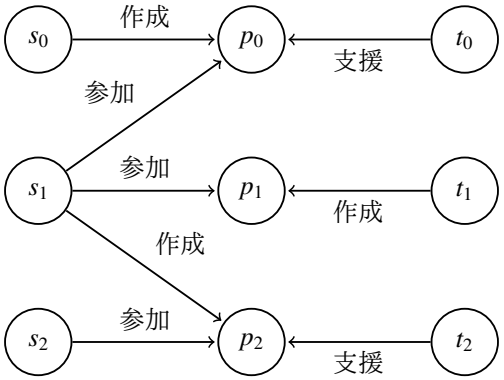

図 1: グラフの例

s_0 作成 p_0
s_1 参加 p_0
s_1 参加 p_1
s_1 作成 p_2
s_2 参加 p_2
t_0 支援 p_0
t_1 作成 p_1
t_2 支援 p_2

図 2: グラフを格納するファイルの例

ラベルが l であるとき，$e = u \xrightarrow{l} v$ と表す．このようにエッジ e で u, v が結ばれているとき，u, v は隣接していると言い，エッジ e はノード u または v に接続していると言う．ノードに接続しているエッジのうち，そのノードを終点とするエッジを入力エッジ，始点とするエッジを出力エッジと呼ぶ．

グラフの例として，図 1 に学生や先生と活動プランの関係を表すグラフ $G = (V, E)$ を示す．ここで，$V = \{s_0, s_1, s_2, t_0, t_1, t_2, p_0, p_1, p_2\}$，$E = \{s_0 \xrightarrow{\text{作成}} p_0, s_1 \xrightarrow{\text{参加}} p_0, s_1 \xrightarrow{\text{参加}} p_1, s_1 \xrightarrow{\text{作成}} p_2, s_2 \xrightarrow{\text{参加}} p_2, t_0 \xrightarrow{\text{支援}} p_0, t_1 \xrightarrow{\text{作成}} p_1, t_2 \xrightarrow{\text{支援}} p_2\}$ である．学生の集合を $\{s_0, s_1, s_2\}$，先生の集合を $\{t_0, t_1, t_2\}$，活動プランの集合を $\{p_0, p_1, p_2\}$ とする．このグラフにおいて，ノードは個々の人間や活動プラン，有向エッジはそれらの関係を表し，エッジにつけられたラベルは始点を主語，終点を目的語とする動詞である．

本研究では，グラフを格納するファイルとして，Resource Description Framework（RDF）の記法の一つである Notation3 のように，各行に 1 つのエッジの情報が書かれているものを考える．具体的には，各行が「始点」「端点を結ぶエッジのラベル」「終点」の 3 つで構成されているファイルである．図 1 のグラフデータをこの記法で表現した例を図 2 に示す．

2.2 グラフにおけるスキーマ

本研究において，グラフにおけるスキーマは元のグラフを要約した概形であり，スキーマ自体もグラフである．以下，スキーマのノードをクラスと呼ぶ．スキーマ抽出ではグラフのすべてのノード（葉ノードを除く場合もある）をクラスに写像するため，クラスは元のグラフの 1 つ以上のノードと対応している．クラス間のエッジには，対応している元のグラフのノード間のエッジが張られる．図 1 のグラフから抽出したスキーマの例を図 3 に示す．s_0, s_1, s_2 は $\bar{v}_0$ に，p_0, p_1, p_2 は $\bar{v}_1$ に，t_0, t_1, t_2 は $\bar{v}_2$ に対応している．このとき，s_0, s_1, s_2 はクラス $\bar{v}_0$ に属すると言う．

本研究では，スキーマは 2 つのファイルで構成される．1 つ目はクラス間のエッジを格納するファイルである．Notation3 のように，各行に「始点」「端点を結ぶエッジのラベル」「終点」を記述する．2 つ目は，元のグラフデータのノードとその属するクラスを格納するファイルである．各行に「グラフのノード」と「それが属するクラス」を記述する．図 3 のスキーマをこれらの記法で表現した例を図 4 に示す．

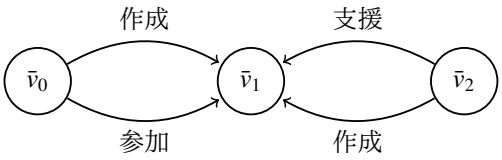

図 3: 抽出されたスキーマの例

$\bar{v}_0$ 作成 $\bar{v}_1$
$\bar{v}_0$ 参加 $\bar{v}_1$
$\bar{v}_2$ 支援 $\bar{v}_1$
$\bar{v}_2$ 作成 $\bar{v}_1$

(a) クラス間のエッジを格納するファイルの例

s_0 $\bar{v}_0$
s_1 $\bar{v}_0$
s_2 $\bar{v}_0$
p_0 $\bar{v}_1$
p_1 $\bar{v}_1$
p_2 $\bar{v}_1$
t_0 $\bar{v}_2$
t_1 $\bar{v}_2$
t_2 $\bar{v}_2$

(b) ノードとそれが属するクラスを格納するファイルの例

図 4: スキーマを格納するファイルの例

3. 提案手法

スキーマ抽出は，各ノードが属するクラスを決定すること（クラス抽出）と，それらクラス間にエッジを張ること（エッジ抽出）の二段階からなる．我々は増分クラスタリング法を用いたスキーマ抽出アルゴリズム [3] におけるクラス抽出手法を拡張し，さらに全体を外部記憶アルゴリズムとした．本章ではまず，クラス抽出時に用いる効用値の定義を行い，次に提案する外部記憶アルゴリズムについて述べる．

3.1 準備

本手法は，ラベル付き有向グラフを入力とし，入出力エッジのラベルが類似しているノードを同じクラスにまとめる手法である．なお，軽微な修正でラベル付き無向グラフにも適用可能である．

クラス抽出を行う際，文献 [3] では入力エッジと出力エッジを区別せずに効用値（utility value）を計算するため，分けられるべきノードが同じクラスになるという問題があった．そこで提案手法では効用値の定義を拡張し，入力エッジと出力エッジそれぞれの効用値を計算し，その線形和をクラス抽出に用いる．

ノード v_i の入力エッジの集合を $R(v_i)$，出力エッジの集合を $A(v_i)$ と表す．クラス $\bar{v}_i$ の入力エッジの集合を $R(\bar{v}_i) = \bigcup_{v_i \in \bar{v}_i} R(v_i)$ とし，同様にクラス $\bar{v}_i$ の出力エッジの集合を $A(\bar{v}_i) = \bigcup_{v_i \in \bar{v}_i} A(v_i)$ とする．

クラス $\bar{v}_i$ に属するノードのうち，入力エッジにラベル l をもつノードの割合を $P_{in}(l|\bar{v}_i)$ と表す．これが高いほど，クラス $\bar{v}_i$ に属しているノードは入力エッジにラベル l をもつことになる． これに対して，すでにクラスに属するノードの中で，入力エッジにラベル l をもつノードのうち，クラス $\bar{v}_i$ に属するノードの割合を $P_{in}(\bar{v}_i|l)$ と表す．これが高いほど，入力エッジにラベル l をもつノードがクラス $\bar{v}_i$ に属していることになる． 入力エッジのラベル l とクラス $\bar{v}_i$ の関係性の高さ $T_{in}(l, \bar{v}_i)$ と，クラス $\bar{v}_i$ におけるその平均 $E_{in}(\bar{v}_i)$ を次のように定義する．

$$T_{in}(l, \bar{v}_i) = P_{in}(l|\bar{v}_i) \cdot P_{in}(\bar{v}_i|l)$$
$$E_{in}(\bar{v}_i) = \frac{1}{|R(\bar{v}_i)|} \sum_{l \in R(\bar{v}_i)} T_{in}(l, \bar{v}_i)$$

入力エッジに対する効用値 U_{in} は各クラスにおける $E_{in}(\bar{v}_i)$ の平均値である．K をクラスの総数（スキーマのノード数）としたとき，U_{in} を次のように定義する．

$$U_{in} = \frac{1}{K} \sum_{i=1}^{K} E_{in}(\bar{v}_i)$$

出力エッジに対しても同様に，クラス $\bar{v}_i$ に属するノードのうち，出力エッジにラベル l をもつノードの割合を $P_{out}(l|\bar{v}_i)$ と表し，すでにクラスに属するノードの中で，出力エッジにラベル l をもつノードのうち，クラス $\bar{v}_i$ に属するノードの割合を $P_{out}(\bar{v}_i|l)$ と表す．出力エッジのラベル l とクラス $\bar{v}_i$ の関係性の高さ $T_{out}(l, \bar{v}_i)$ と，クラス $\bar{v}_i$ におけるその平均 $E_{out}(\bar{v}_i)$，出力エッジに対する効用値 U_{out} を入力エッジと同様に次のように定義する．

$$T_{out}(l, \bar{v}_i) = P_{out}(l|\bar{v}_i) \cdot P_{out}(\bar{v}_i|l)$$
$$E_{out}(\bar{v}_i) = \frac{1}{|A(\bar{v}_i)|} \sum_{l \in A(\bar{v}_i)} T_{out}(l, \bar{v}_i)$$
$$U_{out} = \frac{1}{K} \sum_{i=1}^{K} E_{out}(\bar{v}_i)$$

入出力エッジに対する効用値 U を，入力エッジに対する効用値と出力エッジに対する効用値の線形和と定義する．

$$U = \alpha U_{in} + \beta U_{out}, \quad \alpha + \beta = 1$$

効用値は，接続するエッジのラベルが類似したノードが同じクラスにまとまっているほど大きい値となる．この値が高いほどスキーマの質が高いとみなし，クラス抽出の際に用いる．

3.2 提案手法

[3] のアルゴリズムは，ノードの属するクラスの情報などのスキーマ抽出に必要となるすべての情報を主記憶上に格納し，データの任意の場所にアクセスして処理することを前提としている．しかし，データサイズが非常に大きい場合，必要となるすべての情報は主記憶に格納できないため，このアルゴリズムをそのまま用いて処理することは困難である．

そこで提案手法では，グラフデータをシーケンシャルに読み込み，必要最低限のデータのみを主記憶上に格納し逐次的にクラス抽出を行う．また，クラス間のエッジ抽出に必要だが（サイズの都合上）主記憶に格納することが困難な情報は外部ファイルに書き出す．それらをシーケンシャルに読み込んで分析・処理することによってクラス間のエッジ抽出を実現する．

本手法は（1）前処理，(2）クラス抽出，(3）エッジ抽出の 3 段階からなり，図 2 のようなグラフデータを与えると，そのスキーマを抽出する． 節で述べた通り，入出力エッジを区別する効用値を用いてクラス抽出を行う．なお， [3] では指定したルートノードから深さ優先順にクラスを抽出するが，本研究でのクラス抽出はノード名のアルファベット順である．

処理手順の概要を以下に示す．

入力: グラフ（各行が「始点」「端点を結ぶエッジのラベル」「終点」で構成されるファイル．以下，ファイル 1）

出力: 入力グラフから抽出したスキーマ

処理内容:

1. 前処理
   (a) ファイル 1 を 1 行ずつ読み込み，始点と終点を入れ換え,「終点」「端点を結ぶエッジのラベル」「始点」の順にしてファイル 2 に書き出す．この操作をファイル 1 の最終行まで行う
   (b) ファイル 1 とファイル 2 を外部ソートする．その結果をそれぞれファイル 3，ファイル 4 とする
2. クラス抽出
   (a) ファイル 3 とファイル 4 を並行してシーケンシャルに読み込む
   (b) 効用値を計算し，1 ノードずつクラスを抽出する
   (c) クラス抽出を終えるたびに，ノードとクラスの対応関係をファイル 5 に書き出し，そのノードの出力エッジの情報をファイル 6 に書き出す
3. エッジ抽出
   (a) ファイル 6 を外部ソートする．その結果をファイル 7 とする
   (b) ファイル 5 と，ファイル 7 を並行してシーケンシャルに読み込む
   (c) クラス未特定のノードのクラスをファイル 5 から特定する
   (d) クラス間のエッジを張る

以下，上記処理手順の詳細を述べる．

3.2.1 前処理

スキーマ抽出の対象となるグラフデータは,「始点」「端点を結ぶエッジラベル」「終点」の順で 1 行が構成されているファイルである（ファイル 1）．ノードに接続するすべての出力エッジをシーケンシャルな読み込みで特定するために，ファイル 1 をアルファベット順に外部ソートする（ファイル 3）．ファイル 3 は同じ始点をもつエッジが連続して出現するので，これを 1 行目からシーケンシャルに読み込むことで出力エッジは特定できる．しかし，ノードの入力エッジを特定するためにはファイル 3 全体を参照する必要が生じる．これを回避するため，ファイル 1 の始点と終点を入れ換え「終点」「端点を結ぶエッジのラベル」「始点」の順で各行が構成されるファイル（ファイル 2）を作成し，外部ソートする（ファイル 4）．ファイル 4 は同じ終点をもつエッジが連続して出現するので，各ノードの入力エッジをシーケンシャルな読み込みで特定できる．

3.2.2 クラス抽出

効用値の計算に必要で，主記憶に格納し続けるデータは,「各クラスに属するノードの数」,「各クラス $\bar{v}$ と $\bar{v}$ の各ラベル l に対して，$\bar{v}$ において l をもつノードの数」の 2 つである．後者は，入力エッジと出力エッジを分けて保存する．なお，主記憶にはノードに接続するエッジのラベルや出力先の隣接ノードも格納するが，そのノードのクラス抽出が終わった時点で破棄される．

クラス抽出の流れを Algorithm 1 に示す．前処理でソート済みの 2 つのファイル（ファイル 3，4）はアルファベット順に並んでいるため，両ファイルの先頭を確認，比較して小さい方のノードからクラス抽出を行う（7〜9 行目）．ただし，葉ノードのクラスは *NIL* とし，クラス抽出は行わない（4〜6 行目）．そのノードに接続するエッジがすべて得られたら，そのノードのクラスを抽出する．

クラス抽出は元のアルゴリズム [3] の通り，既存のクラスがない場合は新規クラスを作成し割り当て，既存のクラスがある場合

Algorithm 1 クラス抽出

1: ファイル 3 から 1 行読み込む.「始点」を v とする
2: ファイル 4 から 1 行読み込む.「終点」を u とする
3: **while** ファイル 3 とファイル 4 の少なくとも一方が EOF でない **do**
4: **while** u が葉ノードの場合 **do**
5: ファイル 4 を 1 行読み込む.「終点」を u とする
6: **end while**
7: $current = \min\{u, v\}$
8: ファイル 3 とファイル 4 をシーケンシャルに読み，$current$ の入力エッジと出力エッジを集める
9: 効用値を計算し，$current$ のクラス c を抽出
10: $current$ とそれが属するクラスの対応をファイル 5 に出力
11: $current$ の各出力エッジに対して,「始点」を c に置き換えたものをファイル 6 に出力
12: ファイル 3 の現在行の「始点」を v，ファイル 4 の現在行の「終点」を u とする
13: **end while**

は既存の各クラスに割り当てたときの効用値と，新しいクラスを作成し割り当てたときの効用値を計算し，効用値が最も大きいクラスにそのノードを割り当てる（9 行目）.

ノードのクラス抽出を終えるたびに，次のエッジ抽出で必要な情報を書き出す．クラス間にエッジを張るためには,「クラスを始点とし,（クラス未特定の）ノードを終点とするエッジ」と,（クラス未特定の）ノードのクラスを特定するための「ノードとそれが属するクラスの対応」の 2 つが必要である．そこで,「ノードとそれが属するクラスの対応」をファイル 5 に書き出し，さらに,「クラスを始点とし,（クラス未特定の）ノードを終点とするエッジ」をファイル 6 に書き出す（10，11 行目）．なお，ファイル 6 は「終点（クラス未特定のノード)」「クラスから終点に向かうエッジのラベル」「始点（クラス)」の順で各行が構成されるようにする．これは，次のエッジ抽出において終点でソートを行うことによる．

3.2.3 エッジ抽出

エッジ抽出の流れを Algorithm 2 に示す．まずファイル 6 を外部ソートする（ファイル 7，1 行目）．ファイル 7 は同じ終点をもつエッジが連続して出現するので，そのノードのクラスをファイル 5 からシーケンシャルな読み込みで特定できる．具体的には，ファイル 7 を 1 行読み，あるクラスを始点とし，クラス未特定のノードを終点とするラベル付きエッジを得る．その終点ノードの属するクラスをファイル 5 から特定することによってエッジ抽出を行う（5〜17 行目）．なお，クラスから葉ノードに向かうエッジの出力先のクラスは NIL とする (5〜6 行目)．また，同一のエッジがすでに存在した場合は張らないこととする．

3.3 I/O コスト

Algorithm 1，2 の I/O コストを考える．$G(V, E)$ のノード数を $|V|$，エッジ数を $|E|$，B を主記憶と外部記憶との間で行われるブロック転送のサイズとする．また，$O(sort(|E|))$ はマージソートのオーダーを表している．例えば，2-way マージの場合

$$O(sort(|E|)) = O\left(\frac{|E|}{B}\log_2\frac{|E|}{B}\right)$$

Algorithm 2 エッジ抽出

1: ファイル 6 を外部ソートする．得られたファイルをファイル 7 とする
2: $tmp \leftarrow NIL$
3: **while** ファイル 7 が EOF でない **do**
4: ファイル 7 を 1 行読み込む．その行を $v, l, \bar{v}_i$ とする
5: **if** v が葉ノード **then**
6: エッジ $\bar{v}_i \xrightarrow{l} NIL$ を張る
7: **else if** $v = tmp$ **then**
8: エッジ $\bar{v}_i \xrightarrow{l} \bar{v}_j$ を張る
9: **else**
10: **while** ファイル 5 が EOF でない **do**
11: ファイル 5 を 1 行読み込む．その行を $tmp, \bar{v}_j$ とする
12: **if** $v = tmp$ **then**
13: break
14: **end if**
15: **end while**
16: エッジ $\bar{v}_i \xrightarrow{l} \bar{v}_j$ を張る
17: **end if**
18: **end while**

である [10].

1. 前処理
   2 つのエッジのファイルを外部ソートする I/O コスト: $O(sort(|E|))$
2. クラス抽出
   (a) 2 つのエッジのファイルの読み込み: $O(|E|/B)$
   (b) ノードとクラスの対応をファイルに書き出し: $O(|V|/B)$
   (c) クラス未特定のエッジの書き出し: $O(|E|/B)$
   (d) 書き出したエッジファイルのソート: $O(sort(|E|))$
3. エッジ抽出
   (a) 2 つのエッジのファイルの読み込み: $O(|E|/B)$
   (b) 1 で書き出した，クラス未特定のエッジファイルの読み込み: $O(|E|/B)$
   (c) ノードとクラスの対応ファイルの読み込み: $O(|V|/B)$

（2）と（3）を合わせて，本処理での I/O コストは次のようになる．

$$O\left(\frac{|E|}{B} + \frac{|V|}{B} + sort(|E|)\right) = O\left(\frac{|V|}{B} + sort(|E|)\right)$$

3.4 アルゴリズムの動作例

本節では，図 5 のグラフを用いてアルゴリズムの動作例を示す．このグラフに対応するファイルをファイル 1 とし，図 6a に示す．以下，葉ノードは " で囲み表記する．

まず，ファイル 1 を入力として前処理を行う．前処理では，第 1 列と第 3 列を入れ換え，ファイル 2（図 6b）を作成する．次に，ファイル 1 とファイル 2 をそれぞれ外部ソートし，ファイル 3，4（図 7）を得る．

続いて，Algorithm 1 を用いてクラス抽出を行う．ファイル 3，4 を 1 行読み込み，$v = v_1$，$u =$ "v_6" を得る．しかし，"v_6" は葉

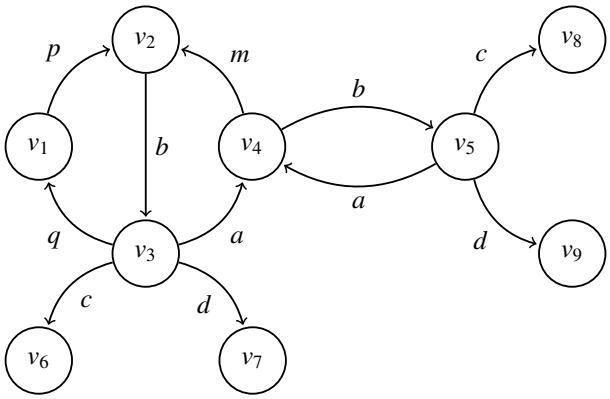

図 5: 入力グラフの例

v_1 p v_2
v_2 b v_3
v_3 q v_1
v_3 a v_4
v_3 c “v_6”
v_3 d “v_7”
v_4 m v_2
v_4 b v_5
v_5 a v_4
v_5 c “v_8”
v_5 d “v_9”

(a) 入力グラフデータ（ファイル 1）

v_2 p v_1
v_3 b v_2
v_1 q v_3
v_4 a v_3
“v_6” c v_3
“v_7” d v_3
v_2 m v_4
v_5 b v_4
v_4 a v_5
“v_8” c v_5
“v_9” d v_5

(b) 第 1 列と第 3 列を入れ換えたもの（ファイル 2）

図 6: ファイル 1 とファイル 2

ノードで，クラス抽出対象外であるため，クラス抽出対象ノードが現れるまでファイル 4 を読み込み，v_1 を u とする．ここで，v と u を比較すると，$v = u = v_1$ なので，この v_1 に接続する出力エッジのラベル p をまずファイル 3 から読み込み，次に入力エッジのラベル q をファイル 4 から読み込む．接続するエッジのラベルがすべて得られたので，クラス抽出を行う．既存のクラスはないため，新規クラス $\bar{v}_1$ を作り，v_1 を $\bar{v}_1$ に割り当てる．ここで，ファイル 5 にノードとそれが属するクラスの対応 v_1 $\bar{v}_1$ を書き出し，ファイル 6 にこのクラス $\bar{v}_1$ から出力するエッジの情報 v_2 p $\bar{v}_1$ を書き出す．次のノード以降も同様に，クラス抽出と情報の書き出しを続ける．これを両ファイルの終端に達するまで繰り返し，クラス抽出を終了する．最終的にファイル 5，6（図 8a，8b）を得る．

最後に，Algorithm 2 を用いてエッジ抽出を行う．まず，ファイル 6 を外部ソートし，ファイル 7（図 8c）を得る．ファイル 7 を 1 行読み込み，$v =$ “v_6”, $l = c, \bar{v}_i = \bar{v}_3$ を得る．このエッジはクラスから葉ノードへ向かうエッジであるため，空のクラス（*NIL*）へ向かうエッジとしてエッジを張る．エッジ情報は $\bar{v}_3$ c *NIL* と書き出す．続いてファイル 7 を 1 行読み込み，$v =$ “v_6” を得る．再び葉ノードなので，先ほどと同様に処理を行う．ファイル 7 を読み進めていくと，$v = v_1, l = q, \bar{v}_i = \bar{v}_3$ を得る．葉ノード以外のノードなので，そのノードのクラスをファイル 5 を読み込んで特定する．ファイル 5 の 1 行目を読み込むことで，v_1 は $\bar{v}_1$ に属すると特定できたので，クラス間にエッジ $\bar{v}_3 \xrightarrow{q} \bar{v}_1$ を張る．これをファイル 7 の終端まで繰り返し，エッジ抽出を終了する．抽出されたスキーマを図 9 に，スキーマのエッジを図 10 に示す．

v_1 p v_2
v_2 b v_3
v_3 a v_4
v_3 c “v_6”
v_3 d “v_7”
v_3 q v_1
v_4 b v_5
v_4 m v_2
v_5 a v_4
v_5 c “v_8”
v_5 d “v_9”

(a) ファイル 1 をソートした結果得られたファイル（ファイル 3）

“v_6” c v_3
“v_7” d v_3
“v_8” c v_5
“v_9” d v_5
v_1 q v_3
v_2 m v_4
v_2 p v_1
v_3 b v_2
v_4 a v_3
v_4 a v_5
v_5 b v_4

(b) ファイル 2 をソートした結果得られたファイル（ファイル 4）

図 7: ファイル 3 とファイル 4

4. 評価実験

本章では（1）入出力エッジの区別の有無から抽出されるスキーマを比較，(2）データサイズを変化させたときのアルゴリズムの処理時間，(3）アルゴリズムの主記憶使用量の 3 点から提案アルゴリズムを評価する．

4.1 評価用データ

評価実験で用いるグラフデータの作成には SP^{2}Bench [11] を用いる．これは，DBLP のデータ構造に沿ったスキーマに基づいて，任意のサイズのラベル付き有向グラフ（Notation3 形式）を作成するベンチマークソフトである．作成されるファイルの各行は「始点」「端点を結ぶエッジのラベル」「終点」で構成される．始点および終点に当たるノードは，一般のノード，匿名ノード，RDF クラス，文字列リテラルである．一般のノードは <>で囲まれており，匿名ノードは _:が先頭に付加される．そして，RDF クラス（これは，スキーマのクラスとは別物である）に当たるノードは，その種類に応じて名前空間接頭辞をつけて表される．例えば，人物クラスは `foaf:`が先頭に付加され，`foaf:Person` と表される．文字列リテラルノードは ” で囲み表される．

クラス抽出の対象となるノードは，葉ノードを除くすべてのノードである．提案アルゴリズムでは，文字列リテラルノードを葉ノードとみなし，クラス抽出の対象となるのは一般のノード，匿名ノード，RDF クラスを表すノードとする．また，各ノードが判別できるように，意味のある文字列のアルファベットに付随する<，>，_:，`foaf:`などもノード名の一部とする．すなわち，`<nodename>`，`_:personname`，`foaf:Person` などをノード名とする．このため，ソートした際には一般のノード，匿名ノード，RDF クラスを表すノードの順に並ぶ．

実験で使用するグラフデータは，約 10MB から約 10GB までの 5 つのサイズのグラフデータである（表 1）．ノード数は葉ノードを含むすべてのノード数である．

4.2 評価実験の詳細

評価実験を行った環境は以下の通りである．

OS: Linux CentOS 7（64bit）

CPU: Intel Xeon E5-2623 v3 3.0GHz

主記憶: 16GB

使用言語: Ruby 2.3.0

以下，得られた結果について述べる．

$v_1\ \bar{v}_1$
$v_2\ \bar{v}_2$
$v_3\ \bar{v}_3$
$v_4\ \bar{v}_2$
$v_5\ \bar{v}_3$

(a) ノードとそれが属するクラスの対応のリスト（ファイル 5）

$v_2\ p\ \bar{v}_1$
$v_3\ b\ \bar{v}_2$
$v_4\ a\ \bar{v}_3$
“v_6” $c\ \bar{v}_3$
“v_7” $d\ \bar{v}_3$
$v_1\ q\ \bar{v}_3$
$v_5\ b\ \bar{v}_2$
$v_2\ m\ \bar{v}_2$
$v_4\ a\ \bar{v}_3$
“v_8” $c\ \bar{v}_3$
“v_9” $d\ \bar{v}_3$

(b) クラスを始点とし，クラス未特定のノードを終点とするエッジのリスト（ファイル 6）

“v_6” $c\ \bar{v}_3$
“v_7” $d\ \bar{v}_3$
“v_8” $c\ \bar{v}_3$
“v_9” $d\ \bar{v}_3$
$v_1\ q\ \bar{v}_3$
$v_2\ m\ \bar{v}_2$
$v_2\ p\ \bar{v}_1$
$v_3\ b\ \bar{v}_2$
$v_4\ a\ \bar{v}_3$
$v_4\ a\ \bar{v}_3$
$v_5\ b\ \bar{v}_2$

(c) ファイル 6 をソートした結果得られたファイル（ファイル 7）

図 8: ファイル 5，ファイル 6，ファイル 7

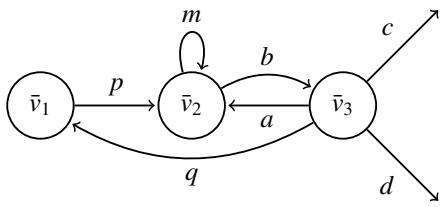

図 9: 図 5 から抽出したスキーマ

4.2.1 評価実験 1: 抽出されたスキーマの比較

[3] の効用値の定義をそのまま用いた場合，入出力エッジを区別せずにクラス抽出を行う．これに対して本研究での効用値の定義を用いる場合は，入力エッジと出力エッジを区別してクラス抽出を行う．出力エッジがノードの特徴を決める傾向があるため，効用値計算においては，出力に重きを置くように $\alpha = 0.2$，$\beta = 0.8$ とした．

5 つのグラフデータ（表 1）に対して前述の 2 通りのスキーマ抽出を行った結果を表 2, 3 に示す．まず，スキーマのサイズについては，どちらの手法もクラス数とクラス間のエッジ数は元のグラフデータと比べて大幅に減少した．[3] の効用値の定義をそのまま用いた場合，抽出されたスキーマのクラス数は 4 であり，データサイズを変えてもクラス数に変化はなかった．本研究での効用値を用いた場合は，クラス数が 8 のものと 9 のものが抽出された．

どのようにクラス抽出されたかについて詳しく見るために，図 11，12 に元のデータ [11] の一部と抽出されたスキーマの比較を示した．[3] の効用値の定義をそのまま用いた場合，`_:references` のごく一部と，ノード `<http://localhost/misc/UnknownDocument>` が同じクラスとなり，`_:Aaarti_Arvelo` といった人名を表すノードは 1 つのクラスになった．`Article`，`Inproceeding`，`Journal` が同じクラスに属しており，`bench`，`foaf`，`rdf` という接頭辞の付く，RDF クラスを表すノードは同じクラスにまとまった．しかし，`Article` や `_:references` が同じクラスとなってしまっている．

本研究での効用値を用いた場合は入出力エッジを区別したことにより，より細かくクラスが抽出された．具体的には，RDF クラスを表すノードがまとまり，さらに `foaf:Document` のみで 1 クラスとなった．`Article` のみのクラスや，`Proceeding` と `Book` がまとまったクラスができた．9 クラスに分かれたものは，`Journal` のみが 1 つのクラスとなったり，`_:references` がさらに分かれたりするなど，サイズによって細かい部分に多少の違いが見られた．

$\bar{v}_3\ c\ NIL$
$\bar{v}_3\ d\ NIL$
$\bar{v}_3\ q\ \bar{v}_1$
$\bar{v}_3\ a\ \bar{v}_2$
$\bar{v}_2\ m\ \bar{v}_2$
$\bar{v}_2\ b\ \bar{v}_3$
$\bar{v}_1\ p\ \bar{v}_2$

図 10: スキーマのエッジファイル

表 1: SP^{2}Bench で作成されたグラフデータ

	サイズ（MB）	ノード数	エッジ数
1	10.5	61,107	100,073
2	108.6	593,379	1,000,009
3	1,105.2	5,582,764	10,000,457
4	5,511.5	27,452,918	50,000,869
5	10,946.1	55,182,878	100,000,380

4.2.2 評価実験 2: アルゴリズムの処理時間

用意した 5 つのグラフデータ（表 1）を対象に前処理，クラス抽出，エッジ抽出にかかる処理時間を計測した．前処理とエッジ抽出での外部ソートには，UNIX の sort コマンドを用いた．処理時間の内訳については表 4 に示し，処理時間の合計のグラフを図 13 に示す．この結果から，処理時間はほぼ線形であることがわかる．

4.2.3 評価実験 3: アルゴリズムの主記憶使用量

主記憶サイズを超えるデータで処理できるかを確認するため，実行中の主記憶使用量を計測した．5 つのファイルのうちで最大のもの（約 10GB）に対して前処理，クラス抽出，エッジ抽出を実行中における主記憶使用量を計測した．sort コマンドを実行する際「-S」オプションをつけ，sort の主記憶使用量を 1GB に制限した．

まず，前処理においては外部ソートの間，オプションで主記憶使用量を 1GB に制限しているため，使用量が最大 1.1GB まで上昇した．この値が前処理中の最大の使用量であった．次に，クラス抽出中の主記憶使用量を計測した．得られた結果を図 14 に示す．実行開始から 7,225 秒まで主記憶使用量は約 150MB で推移し，そこから最大 1,722MB まで上昇した．つまり，入力データサイズ（10,946.1MB）に対して，約 16%の主記憶使用量でクラス抽出が完了している．最後に，エッジ抽出中の結果のグラフを図 15 に示す．前処理と同様に，外部ソートの間はオプションで主記憶使用量を 1GB に制限しているため，最大約 1.1GB まで上昇し，その後のエッジを張る処理の間は約 9MB で推移した．

クラス抽出の終盤で主記憶使用量が増えた理由は，入力エッジを非常に多くもつ「RDF クラス」を表すノードが最後に処理され，このとき多くの情報が主記憶に格納されたためである．クラス抽出はアルファベット順に行われるため，最初に一般ノード，次に匿名ノード，最後に「RDF クラス」を表すノードが処理される．

表 2: 入出力エッジを区別しない場合

	クラス数	エッジ数
1	4	142
2	4	188
3	4	220
4	4	244
5	4	254

表 3: 入出力エッジを区別した場合

	クラス数	エッジ数
1	9	181
2	8	197
3	8	205
4	9	243
5	8	224

この結果から，莫大な数のエッジをもつノードが主記憶使用量の増加につながることがわかった．この問題への対処は今後の課題である．

5. まとめと今後の課題

本研究では，大規模グラフデータに対応したスキーマ抽出手法を提案した．具体的には，増分クラスタリング法を用いた作成法である先行研究 [3] の効用値の定義を拡張し，アルゴリズム全体を外部記憶アルゴリズムとした．提案するスキーマ抽出手法は，(1) ノードに接続するエッジをシーケンシャルに得るために必要なファイルを作成する前処理，(2) ノードが属するクラスを逐次的に効用値を計算して求めるクラス抽出，(3) クラス間にエッジを張るエッジ抽出の 3 段階からなる．

評価実験を行い，(1) 入出力エッジの区別の有無から抽出されるスキーマを比較，(2) データサイズを変化させたときのアルゴリズムの処理時間，(3) アルゴリズムの主記憶使用量の 3 点から提案アルゴリズムを評価した．その結果，先行研究 [3] では分けられなかったノードをクラス分けすることができ，抽出に要する時間はほぼ線形であった．そして，主記憶使用量は最大で入力データサイズの約 16%であるという結果が得られた．

今後の課題として，より包括的な評価実験を行うこと，主記憶使用量を抑える手法を考案することが挙げられる．なお，RDF のクラスを表すノードのように，多くのノードから参照されるノードのみクラス抽出の計算を行わずに別個のクラスとしてまとめることで，主記憶使用量の増大を抑えることができると考えられる．また，提案手法ではグラフデータに変更があった場合，スキーマを作り直さなければならないため，スキーマを更新する手法の開発も試みる予定である．

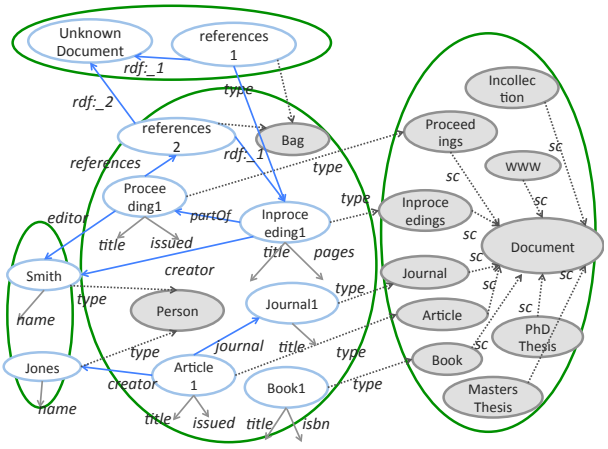

図 11: 入出力エッジを区別しない場合

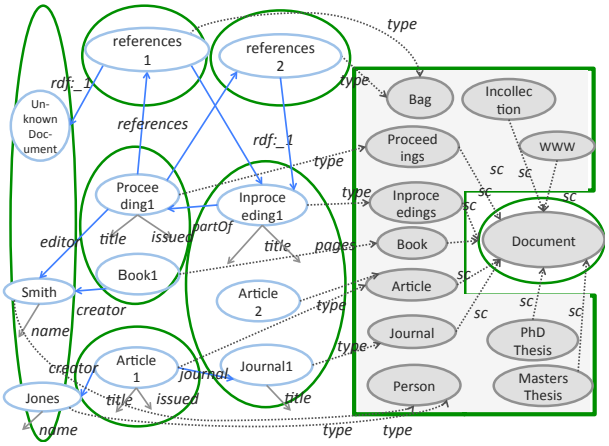

図 12: 入出力エッジを区別した場合

[文献]

[1] Y. Sekine, K. Ikeda, and N. Suzuki, “An algorithm for extracting schemas from external memory graphs,” The First Workshop on Big Network Analytics (in conjunction with CIKM 2016), 2016.

[2] M. Fernandez and D. Suciu, “Optimizing regular path expressions using graph schemas,” Proceedings of the Fourteenth International Conference on Data Engineering, pp.14–23, 1998.

[3] Q.Y. Wang, J.X. Yu, and K.-F. Wong, “Approximate graph schema extraction for semi-structured data,” Advances in Database Technology—EDBT 2000, pp.302–316, Springer, 2000.

[4] R. Goldman and J. Widom, “Dataguides: Enabling query formulation and optimization in semistructured databases,” Technical Report 1997-50, Stanford InfoLab, 1997. http://ilpubs.stanford.edu:8090/264/

[5] R. Goldman and J. Widom, “Approximate Dataguides,” Proceedings of the Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats, vol.97, pp.436–445, 1999.

[6] S. Navlakha, R. Rastogi, and N. Shrivastava, “Graph summarization with bounded error,” Proceedings of the 2008 ACM SIGMOD international conference on Management of dataACM, pp.419–432 2008.

[7] Z. Zhang, J.X. Yu, L. Qin, L. Chang, and X. Lin, “I/O efficient: Computing sccs in massive graphs,” The VLDB Journal—The International Journal on Very Large Data Bases, vol.24, no.2, pp.245–270, 2015.

[8] Z. Zhang, J.X. Yu, L. Qin, Q. Zhu, and X. Zhou, “I/O cost minimization: reachability queries processing over massive graphs,” Proceedings of the 15th International Conference on Extending Database TechnologyACM, pp.468–479 2012.

[9] X. Hu, Y. Tao, and C.-W. Chung, “Massive graph triangulation,” Proceedings of the 2013 ACM SIGMOD international conference on Management of dataACM, pp.325–336 2013.

[10] N. Sitchinava, “Memory hierarchies,” http://algo2.iti.kit.edu/download/mem_hierarchy_01_unedited.pdf. Accessed: 2015-12-14.

表 4: 処理時間の内訳

	前処理（s）	クラス抽出（s）	エッジ抽出（s）	合計（s）
1	1.53	7.16	0.86	9.55
2	6.99	63.80	3.86	74.65
3	62.64	644.42	32.82	739.88
4	386.52	3535.44	170.81	4092.77
5	787.66	7339.29	315.60	8442.55

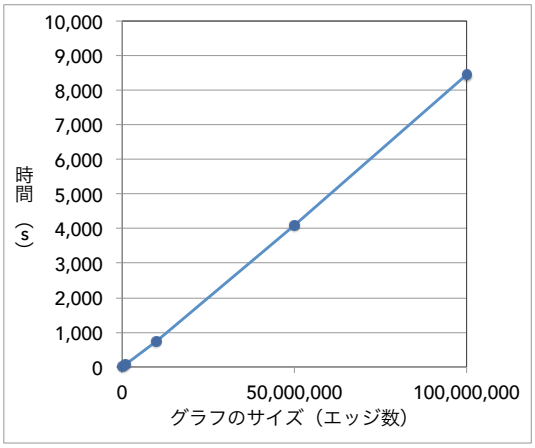

図 13: 処理時間の合計

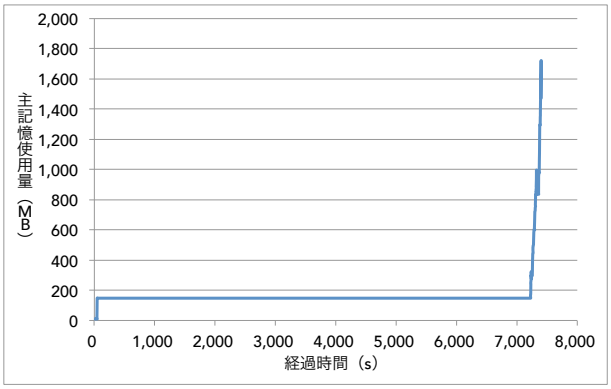

図 14: クラス抽出中の主記憶使用量の推移

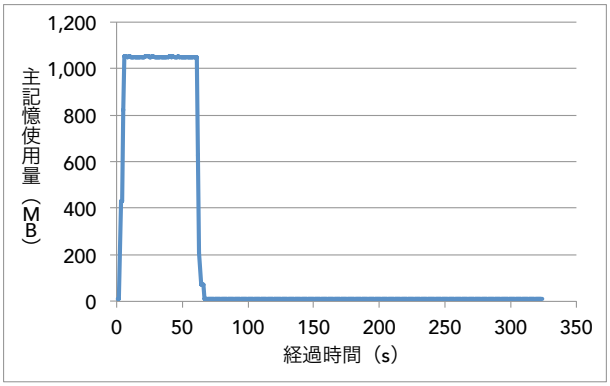

図 15: エッジ抽出中の主記憶使用量の推移

[11] M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel, “SP^{2}Bench: a SPARQL Performance Benchmark,” Data Engineering, 2009. ICDE’09. IEEE 25th International Conference onIEEE, pp.222–233 2009.

関根 吉紀　Yoshiki SEKINE
筑波大学大学院図書館情報メディア研究科博士前期課程在学中．グラフ理論の研究に従事．

池田 光雪　Kosetsu IKEDA
千葉大学アカデミック・リンク・センター特任助教．2016 年筑波大学大学院図書館情報メディア研究科博士後期課程修了．博士（情報学）．グラフ理論やクラウドソーシングなどの研究に従事．電子情報通信学会，情報処理学会各会員．

鈴木 伸崇　Nobutaka SUZUKI
筑波大学図書館情報メディア系准教授 1998 年奈良先端科学技術大学院大学情報科学研究科博士後期課程修了．博士（工学）．データベースやアルゴリズムに関する研究に従事．ACM，電子情報通信学会，情報処理学会各会員．