ラベル付き有向グラフにおける 部分グラフ同型問題を解くための外部記憶アルゴリズム

王　志華† 鈴木　伸崇††

† 筑波大学大学院 図書館情報メディア研究科　〒 305-8550 茨城県つくば市春日 1-2
†† 筑波大学 図書館情報メディア系　〒 305-8550 茨城県つくば市春日 1-2
E-mail: †{s1521649,nsuzuki}@slis.tsukuba.ac.jp

あらまし　本稿では，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部分グラフ同型問題を解くためのアルゴリズムを提案した．提案アルゴリズムは，二つのアルゴリズム（アルゴリズム 1，アルゴリズム 2）として構成する．アルゴリズム 1 では，データグラフを先頭から一部分ずつ主記憶に読み込み，パターングラフとマッチングを行い，完全解と部分解を求める．アルゴリズム 2 では，アルゴリズム 1 で得られた部分解に対して，部分解がなくなるまで「拡大」操作を行う.「拡大」操作の後，完全解になったら出力し，完全解にならなかったグラフは削除する．評価実験を行い，概ね所望の効率でアルゴリズムが動作することが分かった．

キーワード　部分グラフ同型問題，外部記憶アルゴリズム，RDF

1. はじめに

グラフとは，人，モノ，場所などの多様な情報のつながりを表現するデータ構造である．グラフは，データおよびデータ同士の関連性をそれぞれ「ノード」と「エッジ」で表現するデータ構造を持ち，このデータ構造は Web ページのリンク関係や SNS の交友関係，商品の購買関係，交差点と道路の接続関係など私たちの周りに多く存在している．

部分グラフ同型問題（subgraph isomorphism）とは，発見したいパターンを定義したグラフ（以下，パターングラフ）と探索対象のグラフ（以下，データグラフ）が与えられたときに，データグラフに含まれるパターングラフと同型の部分グラフを全て発見する問題である. 部分グラフ同型問題の判定は，グラフに関する問題の中で最も基本的な問題の一つであり，画像における特徴量抽出 [1]，化学式における類似構造の検出 [2] など様々な分野に応用できる．

部分グラフ同型問題における先行研究としては，多くのアルゴリズムがある．具体的には，Ullmann [3] の手法は部分グラフ同型問題を解くための最初の手法であり，検索範囲を減少させる手続きであるバックトラック法を使って部分グラフ同型判定を行う．近年，Ullmann の手法に基づいて発展した VF2 [4]，QuickSI [5]，GraphQL [6]，GADDI [7]，SPath [8] のアルゴリズムもある．これらのアルゴリズムを比較する先行研究 [9] では，ラベル付き無向グラフを対象として共用のフレームワークを作って，各アルゴリズムを比較している．

これまで提案されてきた部分グラフ同型問題を解くためのアルゴリズムの多くはデータを主記憶に格納し処理することを前提としており，データが主記憶に収まらない場合には適用困難である．主記憶のサイズを超えるデータを効率良く処理するアルゴリズムとして，外部記憶アルゴリズム (external memory algorithm) が考えられている．部分グラフ同型問題を解くための外部記憶アルゴリズムとしては [15] がある．しかし，このアルゴリズムはラベル無し単純グラフを対象としており，I/O コストがパターングラフのサイズに関して指数的に増加する．一方，本アルゴリズムはラベル付きグラフを対象としており，I/O コストはパターングラフのサイズに関して線形である．グラフに関する他の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 [10]，到達可能性を判定するもの [11]，三角形分割問題を解くもの [12] がある．

そこで本稿では，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部分グラフ同型問題を解くための手法を提案する．提案手法は，まず全てのデータグラフをファイルとして外部記憶に格納し，一部ずつ主記憶に読み込み，検索や計算などの処理を行う．具体的には，次のような二つのアルゴリズムとして構成する．アルゴリズム 1 では（図 1），データグラフを先頭から $|S_1|$ 大きさのデータを一部分ずつ主記憶に読み込み，パターングラフとマッチングを行い，完全解と部分解を求める．ここで，パターングラフと同じ形になるグラフは完全解と呼び，パターングラフと部分的に一致し，現時点で完全解になるか否か不明のグラフは部分解と呼ぶ．また，S_1 と S_2 はそれぞれ読み込んだデータグラフと部分解を格納する主記憶上の領域である．アルゴリズム 2 では（図 2），アルゴリズム 1 で得られた部分解に対して，部分解がなくなるまで「拡大」操作を行う．ここで，パターングラフとデータグラフのノードとエッジを参照し，部分解において不明だったノードとエッジの情報を追加するという処理を「拡大」操作と呼ぶ.「拡大」操作の後，完全解になったら出力し，完全解にならなかったグラフは削除する．

2. 諸定義

本章では，グラフと部分グラフ同型問題に関する定義について述べる．

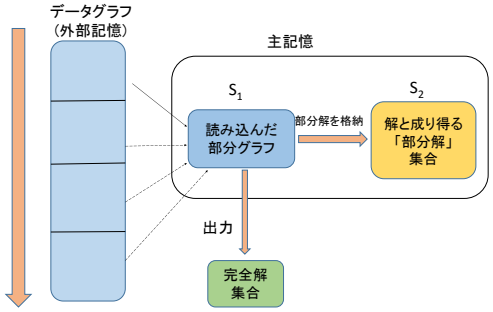


図 1: アルゴリズム 1 の概要

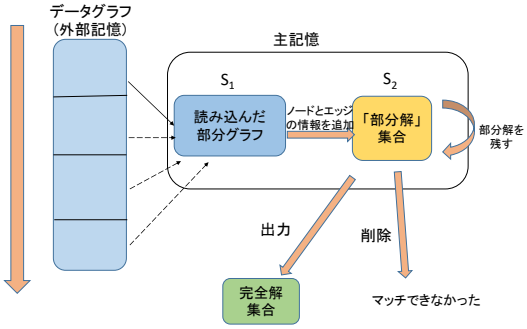


図 2: アルゴリズム 2 の概要

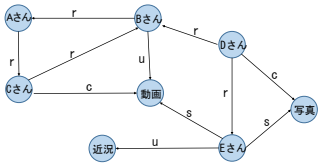
2.1 グラフの概要

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 (以下，単にエッジ) の集合である．エッジは両端にノードを持ち，それらを端点と呼ぶが，両端のノードは同じ場合もある．ノード $v \in V$ を始点とし，ノード $v' \in V$ を終点とする有向エッジ $e \in E$ のラベルが l であるとき，$e = v \xrightarrow{l} v'$（または $e = v' \xleftarrow{l} v$）と表す．このようにエッジ e で v, v' が結ばれているとき，v と v' は隣接していると言い，エッジ e は v と 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．$v \in V$ の入力辺のラベル集合と出力辺のラベル集合をそれぞれ $L_{in}(v)$，$L_{out}(v)$ と表す．

また，グラフ $G = (V, E)$ と $G' = (V', E')$ に対して，$V' \subseteqq V$ かつ $E' \subseteqq E$ を満たす時，G' は G の部分グラフであるという．

例として，図 3 に Facebook の友人関係を表すグラフ示す．このグラフにおいて，各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や行動を表し，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．

グラフをファイルに格納する方法いくつか考えられるが，本稿では，グラフを格納するファイルとして，各行に 1 つのエッジが書かれているものを考える．この記法では，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．図 3 のグラフをこの記法で表現した例を図 4 に示す．



(a) Facebook の友人関係

r	友達申請
u	アップロード
c	コメント
s	シェア

(b) ラベルの意味

図 3: グラフの例

A さん	r	C さん
B さん	r	A さん
C さん	r	B さん
D さん	r	E さん
D さん	r	B さん
B さん	u	動画
C さん	c	動画
D さん	s	写真
E さん	s	動画
E さん	u	近況

図 4: グラフを格納するファイルの例

2.2 部分グラフ同型問題

パターングラフを $P = (V_p, E_p)$，データグラフを $G = (V, E)$ と表す．グラフデータにおける部分グラフ同型問題を次のように定義する．もし

$\forall u \xrightarrow{l} u' \in E_p \Rightarrow \exists f(u) \xrightarrow{l} f(u') \in E$

という条件を満たす単射 $f : V_p \to V$ が存在するならば，P は G において部分グラフ同型であるという．部分グラフ同型問題とは，上記の条件を満たす単射をすべて求める問題である．

3. 提 案 手 法

本研究のアルゴリズムでは，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，部分グラフ同型問題を解くためのアルゴリズムを提案する．提案アルゴリズムでは，ファイルとして外部記憶に格納されたグラフデータを一部ずつ主記憶に読み込み，解を発見するための処理を行う．提案アルゴリズムは，二つのアルゴリズムとして構成する．本章では，この二つのアルゴリズムについて述べる．

3.1 準　　備

3.1.1 前　処　理

本アルゴリズムの処理では，グラフの各ノードに対して，出力エッジ情報と入力エッジ情報両方を確認できる必要があるため，そのようなグラフデータを前処理で用意する．

パターングラフとデータグラフ両方とも，出力辺ファイルと入力辺ファイル二つのファイルを用意する．出力辺ファイルは，「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．入力辺ファイルは，出力辺ファイルの「終点」と「始点」を逆にして，「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である．ノー

ドをシーケンシャルな読み込みで特定するために，出力辺ファイルと入力辺ファイルいずれも左側のノードのアルファベット順に外部ソートする．出力辺ファイルと入力辺ファイル二つのファイルを並列して読み込むことによって，一部分ずつ読み込んでも，各ノードの出力エッジ情報と入力エッジ情報を容易に確認できる．

例でこれらの操作を示す．グラフ（図 5）に対して，出力辺ファイルと入力辺ファイルは図 6 のようになる．例えば，ノード v_1 に対して，出力辺ファイルと入力辺ファイルを同時に読み込む際に，一行目を読み込むだけで，$v_1 \xrightarrow{l} v_2$ と $v_1 \xleftarrow{l} v_3$ のような情報が得られる．本アルゴリズムの処理では，主記憶のサイズを超えないように，一度に読み込む行数を制限するが，この方法で，主記憶に読み込まれるデータのサイズが制限されても，各ノードに対する入出力エッジを得ることができる．

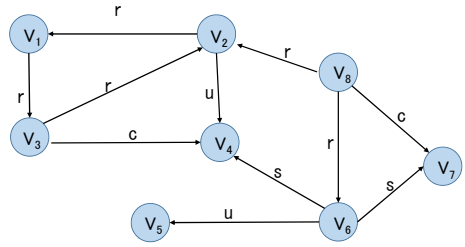


図 5: グラフ

v_1 r v_3
v_2 r v_1
v_2 u v_4
v_3 r v_2
v_3 c v_4
v_6 s v_4
v_6 u v_5
v_6 s v_7
v_8 r v_2
v_8 r v_6
v_8 c v_7

(a) 出力辺ファイル

v_1 r v_2
v_2 r v_3
v_2 r v_8
v_3 r v_1
v_4 c v_3
v_4 s v_6
v_4 u v_2
v_5 u v_6
v_6 r v_8
v_7 s v_6
v_7 c v_8

(b) 入力辺ファイル

図 6: アルゴリズムの入力データ

3.1.2 基本的な考え方

本アルゴリズムでは，データグラフを外部記憶から主記憶に一部分ずつ読み込む．この主記憶上の領域を S_1 とする．各エッジと接続している両端のノードについて，S_1 への読み込まれ方に応じて三種類に分けてアルゴリズムの処理を考える．例（図 7）では，現在外部記憶から主記憶に読み込まれた部分は S_1 で示された上から二番目の領域である．

a） 内部ノード

S_1 に読み込まれたノードのことである．境界ノード（後述）または同じ内部ノード同士と隣接する可能性がある．内部ノード集合は V_{in} で表す．内部ノードと接続しているエッジ及び隣接しているノードは同じ行で読み込まれるので，これらの情報は S_1 を参照することで確認できる．

図 7 では，$V_{in} = \{v_1, v_2, v_6\}$ のようなノードは内部ノードである．これらの内部ノードによって，$\{(v_2\ r\ v_1), (v_2\ c\ v_3), (v_5\ r\ v_6), (v_6\ s\ v_1), (v_7\ r\ v_6)\}$ のような情報が実際に得られる．

b） 境界ノード

S_1 に読み込まれたデータの境界にあるノードのことである．つまり，S_1 の外にあるが，少なくとも S_1 の中にある内部ノードと隣接するノードである．それ以外，外部ノード（後述）または同じ境界ノード同士と隣接する可能性がある．境界ノード集合は V_{bn} で表す．境界ノードは内部ノードと隣接している場合だけ，接続しているエッジの情報が確認できる．

図 7 では，$V_{bn} = \{v_3, v_5, v_7\}$ のようなノードは境界ノードである．これらの境界ノードによって，$\{(v_2\ c\ v_3), (v_6\ s\ v_1), (v_7\ r\ v_6)\}$ のような情報が得られる．

c） 外部ノード

S_1 に読み込まれていないノードのことである．つまり，S_1 の外にあり，かつ，S_1 の中にある内部ノードと隣接しないノードである．境界ノードまたは外部ノード同士と隣接し，v_{out} で表す．解のマッチングにおいて，境界ノードまでパターングラフとマッチングできるが，これ以上の情報は現時点のデータで確認できないため，未知のラベルを x にして，未知のノードを v_{out} にしておく．

図 7 では，v_4 のようなノードは外部ノードである．v_4 と v_5 のエッジ情報は，$\{(v_5\ x\ v_{out})\}$ のように記述する．

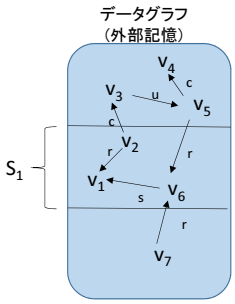


図 7: ノードの分類

3.2 アルゴリズム 1

本節では，アルゴリズム 1 の詳細及びその動作例を示す．アルゴリズム 1 では，データグラフを先頭から K ずつ主記憶の S_1 に読み込み，パターングラフとマッチングを行い，完全解と部分解を求める．ここで，パターングラフと同じ形になるグラフは完全解と呼び，パターングラフと部分的に一致し，現時点で完全解になるか否か不明のグラフは部分解と呼ぶ．マッチングリスト M と M' をペアとして完全解と部分解を格納する．また，S_2 は部分解を格納する主記憶上の領域である．

3.2.1 アルゴリズム 1 の詳細

まず Algorithm 1 でアルゴリズム 1 の全体の流れを示す．3 行目では，主記憶のサイズを超えないように，データグラフを読み込む際に，読み込む行数を K 行に制限し，「一部分ずつ読み込む」を実現する．5 行目内部ノード集合を求める．6 行目で

は，GetBNnodes 関数で境界ノード集合を求める．その詳細は Algorithm 2 で示す．本アルゴリズムは，パターングラフの一つ目のノード（u_1 とする）から操作を始めるため，7〜10 行目では，u_1 とマッチングする可能性があるノード v が発見できたら，新しい M を作成し，ノードペア (u_1, v) を M に追加してから，次の SubgraphSearch で照合を開始する．SubgraphSearch の詳細は Algorithm 3 で示す．

Algorithm 1 アルゴリズム 1 全体の流れ

入力: パターングラフ $P = (V_P, K, E_P)$，データグラフ $G = (V, E)$
出力: 完全解の M と M'

```
    begin
 1: S_2 = ∅
 2: while G の終端まで次の操作を行う do
 3:     G ファイルの K 行を S_1に読み込む
 4:     V_in = V(S_1)
 5:     V_bn = GetBNnodes(S_1, P)
 6:     for each v ∈ V(S_1) do
 7:         if L_out(v) = L_out(u_1) or L_in(v) = L_in(u_1) then
 8:             新しい M を作成し，(u_1, v) を M に追加する
 9:             M' = ∅
10:             SubgraphSearch(P, S_1, M, M')
    end
```

Algorithm 2 GetBNnodes

入力: パターングラフ $P = (V_P, K, E_P)$，データグラフ $G = (V, E)$
出力: V_{bn}

```
    begin
 1: for each v -l-> v' ∈ V(S_1) do
 2:     if v' ∈ V(S_1) but  v ∉ V(S_1) then
 3:         v を V_bnに追加する
 4:     else if v ∈ V(S_1) but v' ∉ V(S_1) then
 5:         v'を V_bnに追加する
 6: return V_bn
    end
```

SubgraphSearch では，1〜4 行目は完全解と部分解を判断する．完全解の場合，M と M' は v_{out} と x を含まない．そこで，1〜2 行目の条件を満たす場合，M と M' を出力する．3〜4 行目の条件を満たす場合，部分解になるので，M と M' を主記憶の S_2 に残し，次のアルゴリズム 2 で「拡大」操作を行う．6 行目でパターングラフのまだマッチング操作していないノード u を一つ選択する．7〜20 行目では，データグラフのノードとエッジの照合を行う．データグラフのノードは内部ノード，境界ノードと外部ノード三つの種類があるため，パターングラフとのマッチング操作をそれぞれ行う．8〜11 行目は内部ノードに対する処理である．IsJoinable_I の詳細は Algorithm 4 で示す．12〜16 行目は境界ノードに対する処理である．IsJoinable_B の詳細は Algorithm 5 で示す．17〜20 行目は外部ノードに対する処理である．IsJoinable_O の詳細は Algorithm 6 で示す．選択された u に対して，これらの三つ関数では，適切な v を決定する．パターングラフとデータグラフのマッチするノードのペアを M に格納する．v に関するラベルとノードの情報を M' に格納する．

Algorithm 3 SubgraphSearch

入力: パターングラフ $P = (V_P, K, E_P)$，データグラフ $G = (V, E)$
出力: 完全解の M と M'

```
    begin
 1: if M の中に外部ノード v_outがない，かつ，M'の中にラベル x がない then
 2:     M と M'を出力
 3: else if M の中に外部ノード v_outがある，または，M'の中にラベル x がある then
 4:     M と M'を S_2に格納する
 5: else
 6:     まだマッチングされていない u ∈ V_pを一つ選ぶ
 7:     for each v ∈ V_in do
 8:         if IsJoinable_I(P, S_1, M.M', u, v) の戻り値 M'_tmpは空ではない then
 9:             (u, v) を M に，v -l-> v' ∈ M'_tmpを M'に追加する
10:             SubgraphSearch(P, S_1, M, M')
11:             (u, v) を M から，v -l-> v' ∈ M'_tmpを M'から削除
12:     for each v ∈ V_bn do
13:         if IsJoinable_B(P, S_1, M.M', u, v) の戻り値 M'_tmpは空ではない then
14:             (u, v) を M に，v -l-> v' ∈ M'_tmpまたは v -x-> v' ∈ M'_tmp を M'に追加する
15:             SubgraphSearch(P, S_1, M, M')
16:             (u, v) を M から，v -l-> v' ∈ M'_tmpまたは v -x-> v' ∈ M'_tmp を M'から削除
17:     if v = v_outかつIsJoinable_O(P, S_1, M.M', u, v) の戻り値 M'_tmpは空ではない then
18:         (u, v) を M に，v -x-> v' ∈ M'_tmpを M'に追加する
19:         SubgraphSearch(P, S_1, M, M')
20:         (u, v) を M に，v -x-> v' ∈ M'_tmpを M'から削除
    end
```

IsJoinable_I では，u の出力エッジと入力エッジそれぞれに対して，マッチングできる内部ノード v を探索する．内部ノードと隣接するノードは境界ノードと内部ノードなので，ラベル情報は確認できる．出力エッジ $u \xrightarrow{l} u' \in E_p$ と $(u', v') \in M$ に対して，u にマッチする適切な v 発見をできたら．v に関する出力エッジとノードの情報 $(v\ l\ v')$ を M'_{tmp} に追加する．同様に，入力エッジ $u' \xrightarrow{l} u \in E_p$ と $(u', v') \in M$ に対して，u にマッチする適切な v を発見できたら．v に関する入力エッジとノードの情報 $(v'\ l\ v)$ を M'_{tmp} に追加する．

IsJoinable_B では，u の出力エッジと入力エッジそれぞれに対して，マッチングできる境界ノード v を探索する．境界ノードと隣接するノードは内部ノード，外部ノードまたは境界ノードであるが，内部ノードと隣接する場合だけラベル情報は確認できる．ラベル情報が確認できる場合，3〜5 行目と 12〜14 行目のように IsJoinable_I と同じ操作を行う．ラベル情報が確認できない場合，6〜8 行目と 15〜17 行目で示すように，ラベルを x にする（x は未知のラベルを表す）．出力エッジ $u \xrightarrow{l} u' \in E_p$ に対して，u にマッチする適切な v を発見でき

たら．v に関する出力エッジとノードの情報 $(v\ x\ v')$ を M'_{tmp} に追加する．同様に，入力エッジ $u' \xrightarrow{l} u \in E_p$ の場合，u にマッチする適切な v を発見できたら．v に関するラベルとノードの情報 $(v'\ x\ v)$ を M'_{tmp} に追加する．

Algorithm 4 IsJoinable_I

入力: パターングラフ $P = (V_P, K, E_P)$，データグラフ $G = (V, E)$，ノード $u \in V_p$，ノード $y \in V(S_1)$
出力: M'_{tmp}
begin
1: $M'_{tmp} = \emptyset$
2: **for each** 出力エッジ $u \xrightarrow{l} u' \in E_p$, $(u', v') \in M$ **do**
3: **if** $v' \in V_{in} \cup V_{bn}$，かつ $v \xrightarrow{l} v' \in E(S_1)$ **then**
4: $v \xrightarrow{l} v'$を M'_{tmp}に入れる
5: **continue**
6: **else**
7: **return** $\emptyset$
8: **for each** 入力エッジ $u' \xrightarrow{l} u \in E_p$, $(u', v') \in M$ **do**
9: **if** $v' \in V_{in} \cup V_{bn}$，かつ $v' \xrightarrow{l} v \in E(S_1)$ **then**
10: $v' \xrightarrow{l} v$ を M'_{tmp}に入れる
11: **continue**
12: **else**
13: **return** $\emptyset$
14: **return** M'_{tmp}
end

Algorithm 5 IsJoinable_B

入力: パターングラフ $P = (V_P, K, E_P)$，データグラフ $G = (V, E)$，ノード $u \in V_p$，ノード $y \in V(S_1)$
出力: M'_{tmp}
begin
1: $M'_{tmp} = \emptyset$
2: **for each** 出力エッジ $u \xrightarrow{l} u' \in E_p$, $(u', v') \in M$ **do**
3: **if** $v' \in V_{in}$，かつ $v \xrightarrow{l} v' \in E(S_1)$ **then**
4: $v \xrightarrow{l} v'$を M'_{tmp}に追加する
5: **continue**
6: **else if** $v' \in V_{bn} \cup \{v_{out}\}$ **then**
7: $v \xrightarrow{x} v'$を M'_{tmp}に追加する
8: **continue**
9: **else**
10: **return** $\emptyset$
11: **for each** 入力エッジ $u' \xrightarrow{l} u \in E_p$, $(u', v') \in M$ **do**
12: **if** $v' \in V_{in}$，かつ $v' \xrightarrow{l} v \in E(S_1)$ **then**
13: $v' \xrightarrow{l} v$ を M'_{tmp}に追加する
14: **continue**
15: **else if** $v' \in V_{bn} \cup \{v_{out}\}$ **then**
16: $v' \xrightarrow{x} v$ を M'_{tmp}に追加する
17: **continue**
18: **else**
19: **return** $\emptyset$
20: **return** M'_{tmp}
end

IsJoinable_O では，u の出力エッジと入力エッジを別々に対応する．外部ノードは境界ノードまたは外部ノードと隣接するので，いずれの場合もラベル情報は確認できない．3〜5 行目と 9〜11 行目で示すように，エッジのラベルを x にする．出力エッジ $u \xrightarrow{l} u' \in E_p$ に対して，u にマッチする適切な v を発見できたら．v に関する出力エッジとノードの情報 $(v\ x\ v')$ を M'_{tmp} に追加する．同様に，入力エッジ $u' \xrightarrow{l} u \in E_p$ の場合，u にマッチする適切な v を発見できたら．v に関するラベルとノードの情報 $(v'\ x\ v)$ を M'_{tmp} に追加する．

Algorithm 6 IsJoinable_O

入力: パターングラフ $P = (V_P, K, E_P)$，データグラフ $G = (V, E)$，ノード $u \in V_p$，ノード $y \in V(S_1)$
出力: M'_{tmp}
begin
1: $M'_{tmp} = \emptyset$
2: **for each** 出力エッジ $u \xrightarrow{l} u' \in E_p$, $(u', v') \in M$ **do**
3: **if** $v' \in V_{bn} \cup \{v_{out}\}$，かつ $v \xrightarrow{l} v' \in E(S_1)$ **then**
4: $v \xrightarrow{x} v'$を M'_{tmp}に追加する
5: **continue**
6: **else**
7: **return** $\emptyset$
8: **for each** 入力エッジ $u' \xrightarrow{l} u \in E_p$, $(u', v') \in M$ **do**
9: **if** $v' \in V_{bn} \cup \{v_{out}\}$，かつ $v' \xrightarrow{l} v \in E(S_1)$ **then**
10: $v' \xrightarrow{x} v$ を M'_{tmp}に追加する
11: **continue**
12: **else**
13: **return** $\emptyset$
14: **return** M'_{tmp}
end

3.2.2 アルゴリズム 1 の動作例

本節では，アルゴリズム 1 の動作例を示す（図 8）．パターングラフ P とデータグラフ G に対して，P のノード u_1 とマッチングできるノードが発見できたら, 照合を開始する．S_1 で示された二番目の領域では，v_4 と v_6 は u_1 と同じ入力ラベルを持っているので，その領域が主記憶に読み込まれてからマッチング操作を始める．ここでは，新しい M を二つを作成する．$M_1 = \{(u_1, v_4)\}$ と $M_2 = \{(u_1, v_6)\}$ である．この時 M' はまだ空であるが，SubgraphSearch で M' が作成される．S_1 に読み込まれたグラフデータにおいて，内部ノード集合 V_{in} と境界ノード集合 V_{bn} を作成する．ここで，$V_{in} = \{v_1, v_3, v_4, v_5, v_6\}$, $V_{bn} = \{v_2, v_7, v_{10}\}$ である．S_1 に読み込まれたデータは

$$\{v_1 \xrightarrow{r} v_4, v_1 \xrightarrow{s} v_{10}, v_2 \xrightarrow{c} v_3, v_2 \xrightarrow{r} v_4, v_5 \xrightarrow{r} v_6, v_5 \xrightarrow{u} v_7, v_7 \xrightarrow{r} v_6\}$$

である．M_1 の場合，パターングラフの各ノードとマッチングできたノードは全て S_1 にあるので，最後に完全解になり，M'_1 とペアで出力する．M_2 の場合，$v_5 \xrightarrow{l} v_6$ と $v_7 \xrightarrow{l} v_6$ は得られたたが，それ以外の情報は現在読み込まれたデータだけで判断できないので，まだ不明のラベルとノードをそれぞれ x と v_{out} にして，M'_2 とペアで部分解として主記憶の S_2 に残す．完全解 (M_1, M'_1) と部分解 (M_2, M'_2) は次のように記述する．

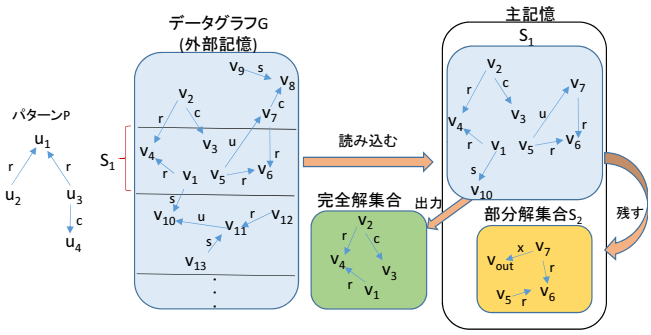


図 8: アルゴリズム 1 動作例

完全解

M_1={(u_1, v_4), (u_2, v_1), (u_3, v_2), (u_4, v_3)}

M_1'={$(v_1\ r\ v_4)$, $(v_2\ r\ v_4)$, $(v_2\ c\ v_3)$}

部分解

M_2={(u_1, v_6), (u_2, v_5), (u_3, v_7), (u_4, v_{out})}

M_2'={$(v_5\ r\ v_6)$, $(v_7\ r\ v_6)$, $(v_7\ x\ v_{out})$}

3.3 アルゴリズム 2

本節では，アルゴリズム 2 の詳細及び動作例を述べる．アルゴリズム 2 では，アルゴリズム 1 で得られた部分解に対して，部分解がなくなるまで「拡大」操作を行う．

3.3.1 アルゴリズム 2 の詳細

Algorithm 7 でアルゴリズム 2 の「拡大」操作の詳細を示す．アルゴリズム 2 では，データグラフを先頭から一部分ずつを読み込む．アルゴリズム 1 と同じく主記憶のサイズを超えないように，3 行目で読み込む行数を K 行に制限する．5，6 行目で,「拡大」したグラフは完全解であるかを判断する．完全解の場合，M と M' には，v_{out} と x が含まないため，この条件を満たす場合は完全解になるので出力する．そうでない場合，引き続き 8〜14 行目の「拡大」を行う．8〜14 行目では，S_2 にある部分解を記述する M と M' に対して，その中にある外部ノード v_{out} と不明なラベル x 情報を適切な v と l に置き換える．具体的には，9〜11 行目では，もし M において u とペアである v が v_{out} であり，v と隣接する v' が v_{out} でない場合，新しく S_1 に読み込まれたグラフデータの中で，v_{out} と置き換えるノードを探索する．もし $v'' \xrightarrow{l} v'$ のようなエッジ発見できたら，v'' で v を置き換えて，$v'' \xrightarrow{l} v$ で $v \xrightarrow{x} v'$ を置き換える．12〜14 行目は，もし M において u とペアである v が v_{out} でないが，v と隣接する v' が v_{out} である場合に対応する．もし $v \xrightarrow{l} v''$ のようなエッジ発見できたら，$(u'v'')$ で $(u'v')$ を置き換えて，$v \xrightarrow{l} v''$ で $v \xrightarrow{x} v'$ を置き換える．最後に,「拡大」の後，完全解にならなかった M と M' を削除する．

Algorithm 7 アルゴリズム 2

入力: パターングラフ $P = (V_P, K, E_P)$，データグラフ $G = (V, E)$，S_2に格納されている M と M'

出力: 完全解の M と M'

begin

1: 「拡大」できる部分解がなくなるまで，2〜15 行目の操作を行う
2: **while** G の終端まで次の操作を行う **do**
3: G ファイルの K 行を S_1 に読み込む
4: **for** each $(M, M') \in S_2$ **do**
5: **if** M の中に外部ノード v_{out}がない，かつ，M'の中にラベル x がない **then**
6: M と M'を出力
7: **else**
8: **for** each $u \xrightarrow{l} u' \in E_p$ **do**
9: **if** $(u, v) \in M$, $v = v_{out}$, $(u', v') \in M$, $v' \neq v_{out}$, $v'' \xrightarrow{l} v' \in E(S_1)$ **then**
10: $(u, v) \in M$ を (u, v'') で置き換える
11: $v \xrightarrow{x} v' \in M'$を $v'' \xrightarrow{l} v$ で置き換える
12: **else if** $(u, v) \in M$, $v \neq v_{out}$, $(u', v') \in M$, $v' = v_{out}$, $v \xrightarrow{l} v'' \in E(S_1)$ **then**
13: $(u', v') \in M$ を (u', v'') で置き換える
14: $v \xrightarrow{x} v' \in M'$を $v \xrightarrow{l} v''$で置き換える
15: S_2に残った M と M'を削除

end

3.3.2 アルゴリズム 2 の動作例

本節では，アルゴリズム 2 の動作例を示す（図 9）．ここで，アルゴリズム 1 の動作例で得られた部分解 (M_2, M_2') に対して,「拡大」操作を行う．データグラフ G を先頭から S_1 に読み込む．先頭の領域が主記憶の S_1 に読み込まれることで，以下のノードとエッジの情報が得られる．

$$\{v_2 \xrightarrow{c} v_3, v_2 \xrightarrow{r} v_4, v_5 \xrightarrow{u} v_7, v_7 \xrightarrow{r} v_6, v_7 \xrightarrow{c} v_8, v_9 \xrightarrow{s} v_8\}$$

これらのノードとエッジの中で $v_7 \xrightarrow{x} v_{out}$ と置き換えることの可能な情報を探索する．具体的な操作は Algorithm 7 の 8〜14 行目を参照する．パターングラフの $u_3 \xrightarrow{c} u_4$ に対して，(u_3, v_7) が M に含まれている．しかも，現在の部分解の M' に $v_7 \xrightarrow{x} v_{out}$ も含まれているので，12〜14 行目を参照する．ここで，$v_7 \xrightarrow{c} v_8$ が発見できたため，(u_4, v_{out}) を (u_4, v_8) で置き換えて，$v_7 \xrightarrow{c} v_8$ で $v_7 \xrightarrow{x} v_{out}$ を置き換える．以上で，部分解 (M_2, M_2') に対する「拡大」操作を終了する．今回の例では,「拡大」後の部分解は完全解になったため，(M_2, M_2') を完全解として出力する.「拡大」前と「拡大」後の (M_2, M_2') は次のように記述する．

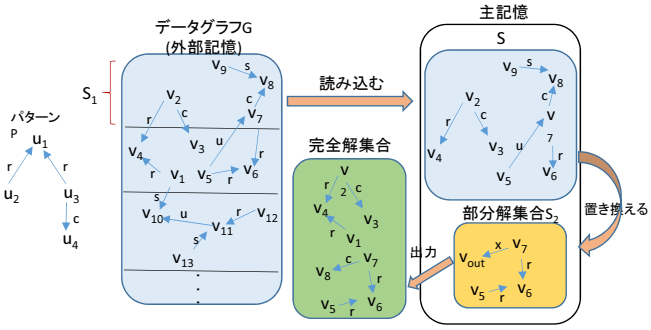


図 9: アルゴリズム 2 動作例

部分解「拡大」前

M_2={(u_1, v_6), (u_2, v_5), (u_3, v_7), (u_4, v_{out})}

M_2'={(v_5 r v_6)，(v_7 r v_6)，(v_7 x v_{out})}
部分解「拡大」後
M_2={(u_1, v_6)，(u_2, v_5)，(u_3, v_7)，(u_4, v_8)}
M_2'={(v_5 r v_6)，(v_7 r v_6)，(v_7 c v_8)}

3.4 I/O コスト

提案アルゴリズムの I/O コストを考える．一般的な外部記憶アルゴリズムのモデルにしたがって，外部記憶と主記憶間ではサイズ B のブロック毎にデータ転送を行うものとする [13]．

まず，前処理の I/O コストを考える．前処理では，出力辺ファイルと入力辺ファイルの外部ソートを行うので，前処理の I/O コストは $O(sort(|E|))$ である．

次に，アルゴリズム本体（アルゴリズム 1 とアルゴリズム 2）の I/O コストを考える．まず，アルゴリズム 1 について，Algorihtm 1 の 3 行目で K 行のデータを読み込んでいる．このサイズを $S \leqq |S_1|$ とする．このとき，K 行読み込むための I/O コストは $O(S/B)$ なので，アルゴリズム 1 の I/O コストは次のようになる．

$$O\left(\frac{S}{B} \cdot \frac{|E|}{S}\right) = O\left(\frac{|E|}{B}\right)$$

次に，アルゴリズム 2 の I/O コストを考える．Algorithm 8 の 3 行目に要する I/O コストは，グラフデータ全体を 1 回読み込むごとに，次の I/O コストを要する．

$$O\left(\frac{S}{B} \cdot \frac{|E|}{S}\right) = O\left(\frac{|E|}{B}\right)$$

ここで，Algorithm 7 の 1 行目の「拡大」が繰り返される回数を考える．1 回の「拡大」で少なくとも 1 つの辺が部分解に追加されるので,「拡大」は最大で $|E_p| - 1$ 回繰り返される．よって，アルゴリズム 2 の I/O コストは次のようになる．

$$O\left(\frac{|E||E_p|}{B}\right)$$

以上から，アルゴリズム本体（アルゴリズム 1 およびアルゴリズム 2）の I/O コストは次のようになる．

$$O\left(\frac{|E||E_p|}{B}\right)$$

4. 評 価 実 験

本稿では，提案アルゴリズムを実装し，(1) 完全解と部分解の数，(2) 実行時間，(3) 主記憶使用量の 3 点を評価する．

4.1 評価用データ

評価実験では，SP^{2}Bench [14] を用いてデータグラフを作成する．SP^{2}Bench は Notation3 形式のラベル付き有向グラフを生成するベンチマークソフトである．SP^{2}Bench によって，グラフデータを作成する際に，n の値を指定することによって，任意のサイズのデータが作成できる．生成されるデータは，コンピュータサイエンス分野の書誌情報データベースである DBLP のデータ構造に沿ったものである．そのため，生成されるファイルは論文や書籍，修士論文や博士論文，議事録などの書誌情報が格納されたグラフデータとなる．

評価実験では，SP^{2}Bench による 559MB（n=5,000,000）から 5,644MB（n=50,000,000）まで 5 つのデータグラフを作成した．作成したデータグラフの詳細を表 1 に示す．

表 1: SP^{2}Bench で作成されたデータグラフ

n	データサイズ (MB)	エッジ数
5,000,000	559	5,000,632
10,000,000	1,105	10,000,457
20,000,000	2,259	20,000,629
30,000,000	3,386	30,025,978
50,000,000	5,644	50,042,632

今回の実験では，エッジ数 3 本のパターングラフ（図 10）利用し，評価を行った．

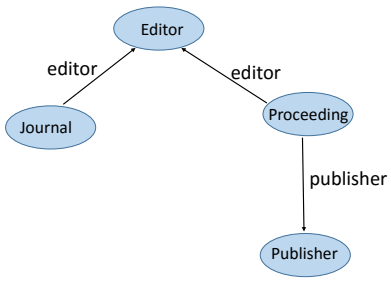


図 10: パターングラフ

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．
CPU: Intel Xeon E5-2623 v3 3.0GHz
主記憶: 16GB RAM
OS: Linux CentOS 7 (64bit)
使用言語: C++

主記憶について，最大使用量を ulimit コマンドで 4G に制限し，これを超えるサイズのファイルを処理可能か否かを確認した．生成した 5 つのデータグラフによる作成した入力ファイル（出力辺ファイルと入力辺ファイル）とエッジ数 3 本のパターングラフを用いて，以下の実験を行った．データグラフの読み込む行数は 20 万行（K=200,000）とした．主記憶に読み込んだデータはそのままページキャッシュとして再利用することで次回以降 CPU からは高速に読み書きができるため，実行時間に影響がある．このような状況を回避するために，$sudo sh -c "echo 3 > /proc/sys/vm/drop_caches"コマンドでページキャッシュと Slab キャッシュの解放した後，データを読み込む．

まず，入力ファイルのサイズを変化させたときの部分解と完全解の数を集計した．その結果を図 11 に示す．この結果から，入力ファイルのサイズを大きくすることで，部分解数と完全解数いずれも概ね線形に増加することが分かった．部分解と完全解の数は主記憶使用量と関連するため，主記憶使用量も線形に増加することが推測できる．

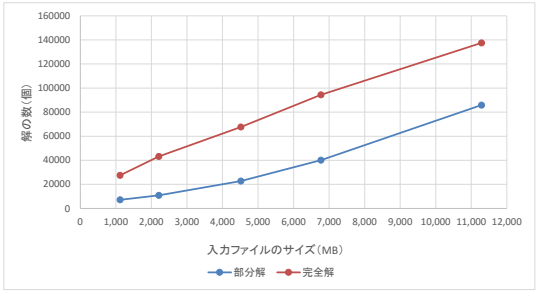


図 11: 部分解と完全解の数

次に，入力ファイルのサイズを変化させたときアルゴリズムの実行時間を time コマンドで計測する．その結果を図 12 に示す．この結果から，入力ファイルのサイズを大きくすることで，実行時間は概ね線形に増加すると言える．しかし，より多くのパターングラフを用いて検証する必要がある．

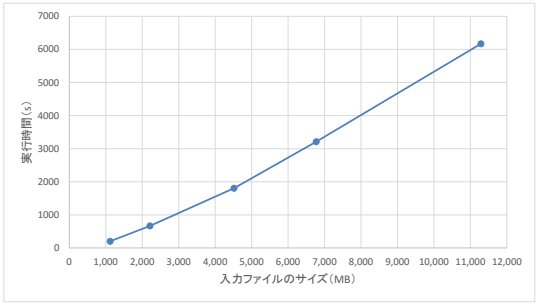


図 12: 実行時間

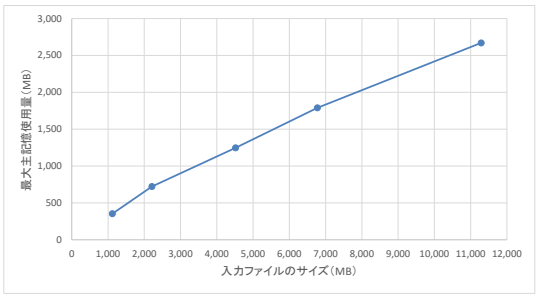


図 13: 主記憶使用量

更に，入力ファイルのサイズを変化させたときの主記憶最大使用量を/usr/bin/time -f "%M KB"コマンドで計測した．その結果を図 13 に示す．入力ファイルのサイズを大きくすることで，主記憶使用量は概ね線形に増加していることが分かった．主記憶最大使用量を 4G に制限した状態で，少なくとも 11G のデータを処理可能であった．このことから，提案アルゴリズムは主記憶使用量より大きいデータを処理可能であると言える．ただし，主記憶使用量の増加を抑えるための更なる工夫も必要と考えられる．

5. むすび

本稿では，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部分グラフ同型問題を解くためのアルゴリズムを提案した．

今後の課題として，主記憶使用量を抑える手法を考案する．主記憶使用量を抑えることで，より大きなサイズのデータを用いて，アルゴリズムが動作できることが期待される．さらに，パターングラフのサイズを変化させる際に，アルゴリズムの全体を評価したいと考えている．

文　献

[1] 下野弘貴, 佐藤正実, 北澤仁志, “特徴空間中の部分グラフ間距離の高速計算による実時間行動識別,” 信学技報, 104(669), PRMU2004-197, pp.109–114, 2005.
[2] 高橋由雅, 藤島悟志, 加藤博明, “化学物質の構造類似性に基づくデータマイニング”, J.Comput. Chem. Jpn., 2(4), pp.119–126, 2003.
[3] J.R. Ullmann, “An algorithm for subgraph isomorphism,” J. ACM, 23(1), pp.31–42, 1976.
[4] L.P. Cordella, P. Foggia, C. Sansone, and M. Vento, “A (sub) graph isomorphism algorithm for matching large graphs,” IEEE Trans. PAMI, 26(10), pp.1367–1372, 2004.
[5] H. He and A. K. Singh, “Closure-tree: An index structure for graph queries,” Proc. ICDE, pp.38–49, 2006.
[6] Huahai He, Ambuj K. Singh. “Graphs-at-a-time: query language and access methods for graph databases,” Proc. SIGMOD, pp.405–418, 2008.
[7] Shijie Zhang, Shirong Li, Jiong Yang, “GADDI: distance index based subgraph matching in, biological networks,” Proc. EDBT, pp.192–203, 2009.
[8] Peixiang Zhao, Jiawei Han, “On graph query optimization in large networks,” PVLDB, 3(1), pp.340–351, 2010.
[9] Lee, Jinsoo, et al. “An in-depth comparison of subgraph isomorphism algorithms in graph databases,” PVLDB, 6(2), 2012.
[10] Zhiwei Zhang, Jeffrey Xu Yu, Lu Qin, Lijun Chang, and Xuemin Lin. “I/O efficient: Computing sccs in massive graphs,” VLDB Journal, 24(2), pp.245–270, 2015.
[11] Zhiwei Zhang, Jeffrey Xu Yu, Lu Qin, Qing Zhu, and Xiaofang Zhou. “I/O cost minimization: reachability queries processing over massive graphs,” Proc. EDBT, pp.468–479, 2012.
[12] Xiaocheng Hu, Yufei Tao, and Chin-Wan Chung. “Massive graph triangulation,” Proc. SIGMOD, pp.325–336. 2013.
[13] Jeffrey Scott Vitter. “Algorithms and Data Structures for External Memory,” Foundations and Trends in Technology, Now Publishers, 2008.
[14] Michael Schmidt, Thomas Hornung, Georg Lausen, and Christoph Pinkel. “SP2Bench: a SPARQL performance benchmark,” Proc. ICDE, pp.222–233, 2009.
[15] Francesco Silvestri. “Subgraph enumeration in massive graphs,” CoRR, abs/1402.3444, https://arxiv.org/abs/1402.3444, 2015.