部分グラフ同型問題と模倣関係の融合

菅原　知倫†　　鈴木　伸崇††

† 筑波大学 情報学群 知識情報・図書館学類　〒 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: †s1311508@u.tsukuba.ac.jp, ††nsuzuki@slis.tsukuba.ac.jp

あらまし　グラフデータにおける主なパターンマッチの定義として，部分グラフ同型問題（subgraph isomorphism）と模倣関係（graph simulation）の二つがある．これらの概念はそれぞれ次のような特徴がある．(1) 部分グラフ同型問題は NP 完全であり，この問題を解くには最悪の場合指数時間を要する．(2) 模倣関係によるパターンマッチは多項式時間で計算できるものの，解の構造がパターングラフと一致しない．そこで本稿では，部分グラフ同型問題と模倣関係を融合し，両者の利点を活かしたパターンマッチングアルゴリズムを提案する．評価実験では，提案アルゴリズムが概ね所望の動作効率でグラフデータにおけるパターンマッチを行えることを確認した．

キーワード　グラフデータ，部分グラフ同型問題，模倣関係

1. はじめに

グラフデータは情報のつながりを表現することに適したデータ構造であり，人間関係や交通ネットワークの管理など幅広く利用されている．グラフデータの効率的なパターンマッチは，そうした実際の利用に応用できる重要な課題の一つである．

グラフデータにおける主なパターンマッチの定義として，部分グラフ同型問題（subgraph isomorphism）と模倣関係（graph simulation）の二つがある．部分グラフ同型問題は，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見つけるというもので，SNS の分析や化学式の類似構造の発見などに利用できる．部分グラフ同型問題に関する先行研究として，Ullmann によって提案された初の実用的なアルゴリズムがある [1]．また，近年では Ullmann のアルゴリズムを拡張する形で VF2 [2]，QuickSI [3]，GraphQL [4]，GADDI [5]，SPath [6] といった多くのアルゴリズムが提案された．しかし，部分グラフ同型問題は NP 完全であり，上述のアルゴリズムを用いたとしても，この問題を解くには最悪の場合指数時間を要する．また，部分グラフ同型問題はパターングラフと解となる部分グラフのノード間に 1 対 1 の関係があるため，本来は 1 つの解にまとまって欲しいノードが異なる解に分かれてしまい，更に解の数が必要以上に大きくなってしまうという問題がある．

一方，模倣関係は，ウェブサイトの分類やソーシャルネットワーク上の役割の発見などに利用される．この概念によるパターンマッチは Henzinger らの研究 [7] により多項式時間で計算できることが示されている．しかし，模倣関係はパターングラフとデータグラフ間の対応を関係として求めるため，解の構造がパターングラフと一致しないという問題がある．特に，模倣関係の解は単一の関係として得られるため，本来は異なる解に分かれるべきノードが同じ解に含まれてしまうという問題がある．

そこで本稿では，まず部分グラフ同型問題と模倣関係を融合し，両者の利点を活かした新たな概念を定義する．次に，定義した新概念に基づくパターンマッチングアルゴリズムを提案する．具体的には，パターングラフのノードに対して，その中から部分グラフ同型問題の定義を適用するノード（以下，キーノード）を任意に設定する．そして，キーノードに対しては部分グラフ同型問題の定義，それ以外のノードに対しては模倣関係の定義を適用したマッチングを行う．これにより本論文が提案するアルゴリズムは，部分グラフ同型問題と比べて計算コストが低く，模倣関係と比べて解の構造がパターングラフと一致するマッチングを可能とする．ただし，よりパターングラフの構造と一致するノードとマッチさせるために，本研究では一般的な模倣関係の定義を拡張し，制約を強めた模倣関係を扱う．

提案アルゴリズムを実装し，ラベル付き有向グラフを対象とした評価実験を行った．その結果，提案アルゴリズムが所望の動作効率でグラフデータにおけるパターンマッチを行えることを確認した．

本稿の構成は以下の通りである．2 章では，グラフ及びパターンマッチに関する定義を行う．3 章では，部分グラフ同型問題と模倣関係を融合した新たな概念を定義し，本研究で提案するアルゴリズムについて説明する．4 章では，評価実験について述べる．5 章でまとめと今後の課題について述べる．

2. 諸定義

本章では，本研究で扱うグラフの概要と，部分グラフ同型問題および模倣関係の定義について述べる．

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 はラベル付き有向辺の集合である．始点を $u \in V$，終点を $v \in V$，ラベルを l とする有向辺 $e \in E$ を $e = u \xrightarrow{l} v$ と表す．このときノード u と v を有向辺 e の端点と呼び，端点は辺 e に接続しているという．また，ノードに接続する辺のうち，そのノードを終点とする辺を

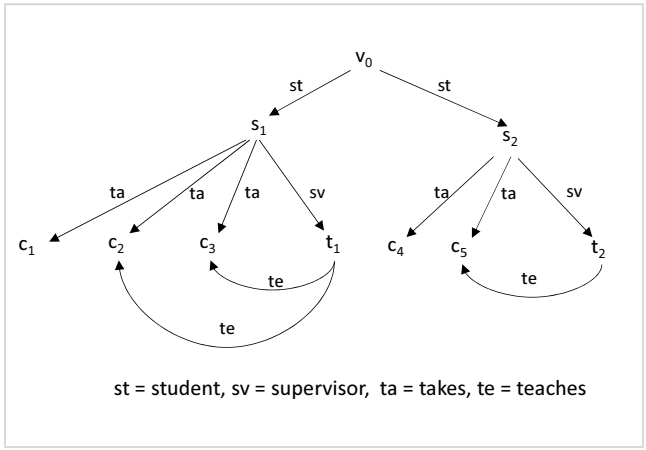

図 1　グラフの例

入力辺，始点とする辺を出力辺と呼び，$v \in V$ の入力辺集合と出力辺集合をそれぞれ $In(v)$，$Out(v)$，入力辺のラベル集合と出力辺のラベル集合をそれぞれ $lab_{in}(v)$，$lab_{out}(v)$ と表す．

G における辺の系列 $e_1, e_2, \cdots, e_n$ について，辺 e_i の端点が v_{i-1} と v_i であり $(1 \leqq i \leqq n)$，$v_0 = v$, $v_n = v'$ とすると，この辺の系列を v から v' への foward/backward パス（以下，fb パス）と呼ぶ．また，v から v' への fb パスが存在するならば，v' は v から fb パスで到達可能であるという．

例として，図 1 に学生や教員と授業の関係を表すグラフ $G = (V, E)$ を示す．ここで，

$$
\begin{aligned}
V &= \{v_0, s_1, s_2, c_1, c_2, c_3, c_4, c_5, t_1, t_2\} \\
E &= \{v_0 \xrightarrow{st} s_1, v_0 \xrightarrow{st} s_2, s_1 \xrightarrow{sv} t_1, s_2 \xrightarrow{sv} t_2, \\
&\quad s_1 \xrightarrow{ta} c_1, s_1 \xrightarrow{ta} c_2, s_1 \xrightarrow{ta} c_3, s_2 \xrightarrow{ta} c_4, \\
&\quad s_2 \xrightarrow{ta} c_5, t_1 \xrightarrow{te} c_2, t_1 \xrightarrow{te} c_3, t_2 \xrightarrow{te} c_5\}
\end{aligned}
$$

である．ノードは学生や教員，授業を表し，ラベル付き有向辺はそれらの関係を表している．ただし，学生の集合を $\{s_1, s_2\}$，授業の集合を $\{c_1, c_2, c_3, c_4, c_5\}$，教員の集合を $\{t_1, t_2\}$ とする．例えば，学生 s_1 は授業 c_1，c_2，c_3 を受講していること，授業 c_2 と c_3 は学生 s_1 の指導教員である教員 t_1 が教える授業であるということがわかる．また，このグラフにおいて s_1 から c_4 への fb パス p_0 は $s_0 \xleftarrow{st} v_0 \xrightarrow{st} s_2 \xrightarrow{ta} c_4$ となり，c_4 は s_1 から p_0 で到達可能であるという．

2.2　パターンマッチの定義

パターングラフを $P = (V_p, E_p)$，データグラフを $G = (V, E)$ とする．グラフデータにおけるパターンマッチを次のように定義する．

部分グラフ同型問題

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．

模倣関係

もし $\forall u \xrightarrow{l} u' \in E_p$ が次の条件を満たすならば，二項関係 $S \subseteqq V_p \times V$ を模倣関係という．

- $(u, v) \in S$ ならば, $(u', v') \in S$ となる辺 $v \xrightarrow{l} v' \in E$ が存在する．
- $(u', v') \in S$ ならば, $(u, v) \in S$ となる辺 $v \xrightarrow{l} v' \in E$ が存在する．

部分グラフ同型問題と模倣関係の定義に基づくパターンマッチの一例を示す．データグラフを図 1，パターングラフを図 2 とする．図 2 のパターングラフは「指導教員の担当する授業を受講する学生」を表す．部分グラフ同型問題の場合，次の 3 つの解が得られる．

- f(学生) $= s_1$, f(授業) $= c_2$, f(教員) $= t_1$
- f(学生) $= s_1$, f(授業) $= c_3$, f(教員) $= t_1$
- f(学生) $= s_2$, f(授業) $= c_5$, f(教員) $= t_2$

一方模倣関係の場合，次の解が得られる．

- $\{$(学生, s_1), (学生, s_2), (授業, c_2), (授業, c_3), (授業, c_5), (教員, t_1), (教員, t_2)$\}$

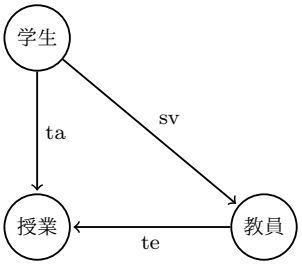

図 2　パターングラフの例

3. 提 案 手 法

本章では，部分グラフ同型問題と模倣関係を融合した新たな概念の定義および定義した新概念に基づくパターンマッチングアルゴリズムについて述べる．

3.1　新概念の定義

新たな概念を定義するために，パターングラフの定義を拡張する（拡張パターングラフ）．拡張パターングラフは $P = (V_p, K, E_p)$ と定義され，ここで，K はキーノードの集合，V_p はそれ以外のノードの集合である．P とデータグラフ $G = (V, E)$ に対して，$f : K \to V$ を単射，$S \subseteqq V_p \times V$ を二項関係とする．もし f と S が次の条件を満たすならば，組 (f, S) を G における P の解と定義する．

- 各辺 $u \xrightarrow{l} u' \in E$ が次の条件を満たす．
  - $u \in K$ の場合:$u' \in K$ ならば，$f(u) = v \wedge f(u') = v'$，$u' \notin K$ ならば，$f(u) = v \wedge (u', v') \in S$ となる辺 $v \xrightarrow{l} v' \in E$ が存在する．
  - $u \notin K$ の場合 : $u' \in K$ ならば，$(u, v) \in S \wedge f(u') = v'$，$u' \notin K$ ならば，$(u, v) \in S \wedge (u', v') \in S$ となる辺 $v \xrightarrow{l} v' \in E$ が存在する．
- 各辺 $u' \xrightarrow{l} u \in E$ が次の条件を満たす．
  - $u \in K$ の場合:$u' \in K$ ならば，$f(u) = v \wedge f(u') = v'$，$u' \notin K$ ならば，$f(u) = v \wedge (u', v') \in S$ となる辺 $v' \xrightarrow{l} v \in E$ が存在する．

– $u \notin K$ の場合：$u' \in K$ ならば，$(u,v) \in S \wedge f(u') = v'$，$u' \notin K$ ならば，$(u,v) \in S \wedge (u',v') \in S$ となる辺 $v' \xrightarrow{l} v \in E$ が存在する．

ただし，G における P の解 (f,S) に対して，もし他のどの組 (f,S') に対しても $S' \subseteqq S$ となるならば，(f,S) は最大であるという．なお，この定義は部分グラフ同型問題と模倣関係を一般化したものである．この定義は $K = V_p$ のとき部分グラフ同型問題と一致し，$K = \emptyset$ のとき模倣関係と一致する．以下，拡張パターングラフのことを単にパターングラフと呼ぶ．

この定義に基づくパターンマッチの一例を示す．データグラフを図 1，パターングラフを図 2 とする．また，図 2 中のノード「学生」をキーノードとすると次の 2 つの解が得られる．

- f(学生) $= s_1$，　$\{$(授業, c_2), (授業, c_3), (教員, t_1)$\}$
- f(学生) $= s_2$，　$\{$(授業, c_5), (教員, t_2)$\}$

3.2　提案アルゴリズム

本節では，3.1 節の定義を用いたパターンマッチングアルゴリズムについて述べる．ただし，$u' \xrightarrow{l} u \in E_p$ と $v \in S(u)$ に対して，$v' \in S(u')$ である辺 $v' \xrightarrow{l} v$ が存在しない場合，v を dangling node とよぶ．同様に，$u' \xrightarrow{l} u \in E_p$ と $v' \in S(u')$ に対して，$v \in S(u)$ である辺 $v' \xrightarrow{l} v$ が存在しない場合，v' を dangling node とよぶ．Algorithm 1 にアルゴリズム MATCH を，Algorithm 2 にアルゴリズム MATCH の 14 行目で用いている INITIALIZE 関数を示す．

Algorithm 1 は 2 行目で，入出力辺が最も多い P のノード u_s を選択する．3，4 行目で，u_s を起点とし，u_s から u_s 以外の各キーノードへの fb パスを求める．5 行目で，u_s にマッチし得る G のノード $C(u_s)$ を求める．6 ～ 31 行目のくり返しで，$C(u_s)$ のノードから fb パスを辿ることで，u_s 以外のキーノードにマッチし得る G のノード $C(u_i)$ を求め，単射 $f : K \to V$ を作成する．もし，fb パスが複数存在する場合，すべての fb パスで到達可能なノードをマッチし得る候補とする．また各くり返しにおいて，作成した単射 $f : K \to V$ に基づき，関係 $S \subseteqq V_p \times V$ を求める．関係 $S \subseteqq V_p \times V$ は，14 行目の INITIALIZE 関数によって解の候補となる関係（dangling node を含む）を求めた後，17 ～ 29 行目で dangling node を削除することで求める．

Algorithm 2 は 2 行目で，始点もしくは終点どちらか片方のみがキーノードの辺を T に追加する．3 ～ 5 行目で，始点と終点が共にキーノードである P の辺に対応する辺が G に存在するかどうかをチェックする．6 ～ 32 行目のくり返しで，すべての $u \in V_p$ について，マッチし得る G のノード $S(u)$ を求める．$S(u)$ は，P のノード u の入出力辺と，u にマッチし得る候補ノードの入出力辺を順々に見ていくことで求める．6 ～ 32 行目のくり返しの対象となる辺 $u \xrightarrow{l} u' \in T$ は，常に始点か終点どちらか一方のノードはマッチし得る候補が求まっているが，もう一方のノードにマッチし得る候補が求まっていない．マッチし得る候補が求まっているノードが始点か終点かによって 7 ～ 19 行目の処理を行うか 20 ～ 32 行目の処理を行うか決定する．どちらの場合も，マッチし得る候補が求まっていないノード $u \in V_p$ に対応する候補を求め，その後 $u \in V_p$ の入出力辺のうち，隣接する頂点にマッチし得る候補が求まっていない辺を T に追加する．

Algorithm 1 MATCH

Input: パターングラフ $P = (V_P, K, E_P)$，データグラフ $G = (V, E)$

Output: 解 (f,S) の集合 R

1: $R \leftarrow \emptyset$
2: K の中から $In(u_s) \cup Out(u_s)$ が最大のノード $u_s \in K$ を選択する．
3: **for** each $u_i \in K \backslash \{u_s\}$ **do**
4: 　P における u_s から u_i までの fb パス p_i を求める．
5: $C(u_s) \leftarrow \{v_s \in V \mid lab_{in}(u_s) \subseteqq lab_{in}(v_s)$ かつ $lab_{out}(u_s) \subseteqq lab_{out}(v_s)\}$
6: **for** each $v_s \in C(u_s)$ **do**
7: 　**for** each p_i **do**
8: 　　$K_i \leftarrow \{v \in V \mid v$ は v_sから fb パス p_iで到達可能 $\}$
9: 　**if** $K_i = \emptyset,\ 1 \leqq i \leqq k$ **then**
10: 　　continue
11: 　**for** each $(v_1, v_2, \cdots, v_k) \in K_1 \times K_2 \times \cdots \times K_k$, **do**
12: 　　$f(u_s) \leftarrow v_s$
13: 　　$f(u_i) \leftarrow v_i,\ 1 \leqq i \leqq k$
14: 　　$S \leftarrow INITIALIZE(P, G, f)$
15: 　　$S \leftarrow S \setminus \{(u,v) \in S \mid v = v_s \vee v = v_i,\ 1 \leqq i \leqq k\}$
16: 　　$Q \leftarrow \emptyset$
17: 　　**for** each $u \in V_p$ **do**
18: 　　　**if** $S(u)$ が dangling node を含む **then**
19: 　　　　$S_{old}(u) \leftarrow S(u)$
20: 　　　　$S(u)$ からすべての dangling node を削除し，Q に u を加える．
21: 　　**for** each $u' \in Q$ **do**
22: 　　　**for** each $v' \in S_{old}(u') \backslash S(u')$and each $v' \xrightarrow{l} v \in E$ s.t. $u' \xrightarrow{l} u \in E_p \wedge v \in S(u)$ **do**
23: 　　　　**if** v が dangling node になる **then**
24: 　　　　　$S(u)$ から v を削除し，Q に u を加える．
25: 　　　　　$S_{old}(u) \leftarrow S(u)$
26: 　　　**for** each$v' \in S_{old}(u') \backslash S(u')$and each $v \xrightarrow{l} v' \in E$ s.t. $u \xrightarrow{l} u' \in E_p \wedge v \in S(u)$ **do**
27: 　　　　**if** v が dangling node になる **then**
28: 　　　　　$S(u)$ から v を削除し，Q に u を加える．
29: 　　　　　$S_{old}(u) \leftarrow S(u)$
30: 　　**if** すべての $u \in V_p$ について，$S(u) \neq \emptyset$ **then**
31: 　　　R に (f,S) を加える．
32: R を返す．

提案アルゴリズムの時間計算量について考える．まず，アルゴリズム MATCH の 14 行目で用いている INITIALIZE 関数の時間計算量を考える．1，12，13 行目は定数時間で実行できる．2 行目は P のすべての辺から，始点もしくは終点どちらか片方のみがキーノードの辺を T に追加するため，$O(|E_p|)$ の時間がかかる．3 ～ 5 行目のくり返しは最大でも $|E_p|$ 回であるた

Algorithm 2 INITIALIZE

Input: パターングラフ $P = (V_p, K, E_p)$，データグラフ $G = (V, E)$，単射 $f : K \to V$

Output: 二項関係 $S \subseteqq V_p \times V$

1: $S \leftarrow \emptyset$
2: $T \leftarrow \{u \xrightarrow{l} u' \mid u \xrightarrow{l} u' \in E_p,\ (u \in K \wedge u' \notin K) \vee (u' \in K \wedge u \notin K)\}$
3: **for** each $u \xrightarrow{l} u' \in E_p$ s.t. $u \in K \wedge u' \in K$ **do**
4: **if** $f(u) \xrightarrow{l} f(u') \notin E$ **then**
5: $\emptyset$ を返す.
6: **for** each $u \xrightarrow{l} u' \in T$ **do**
7: **if** $u \in K \vee (u \notin K \wedge S(u) \neq \emptyset)$ **then**
8: **if** $u \notin K$ **then**
9: $S(u') \leftarrow \{v' \in V \mid v \in S(u), v \xrightarrow{l} v' \in E\}$
10: **else**
11: $S(u') \leftarrow \{v' \in V \mid v \in f(u), v \xrightarrow{l} v' \in E\}$
12: **if** $S(u') = \emptyset$ **then**
13: $\emptyset$ を返す.
14: **for** each $u' \xrightarrow{l} u'' \in Out(u')$ **do**
15: **if** $u'' \notin K \wedge S(u'') = \emptyset$ **then**
16: $u' \xrightarrow{l} u''$ を T に加える.
17: **for** each $u'' \xrightarrow{l} u' \in In(u')$ **do**
18: **if** $u'' \notin K \wedge S(u'') = \emptyset$ **then**
19: $u'' \xrightarrow{l} u'$ を T に加える.
20: **else if** $u' \in K \vee (u' \notin K \wedge S(u') \neq \emptyset)$ **then**
21: **if** $u' \notin K$ **then**
22: $S(u) \leftarrow \{v \in V \mid v' \in S(u'), v \xrightarrow{l} v' \in E\}$
23: **else**
24: $S(u) \leftarrow \{v \in V \mid v' \in f(u'), v \xrightarrow{l} v' \in E\}$
25: **if** $S(u) = \emptyset$ **then**
26: $\emptyset$ を返す.
27: **for** each $u \xrightarrow{l} u'' \in Out(u)$ **do**
28: **if** $u'' \notin K \wedge S(u'') = \emptyset$ **then**
29: $u \xrightarrow{l} u''$ を T に加える.
30: **for** each $u'' \xrightarrow{l} u \in In(u)$ **do**
31: **if** $u'' \notin K \wedge S(u'') = \emptyset$ **then**
32: $u'' \xrightarrow{l} u$ を T に加える.
33: S を返す.

め，2 行目と同様に $O(|E_p|)$ である．6 ∼ 32 行目のくり返しについて考える．8 ∼ 11 行目は，調べる必要がある v の出力辺 $v \xrightarrow{l} v' \in E$ の数が最大 $|E|$ であるため，$O(|E|)$ の時間がかかる．14 ∼ 16 行目および 17 ∼ 19 行目の処理は，6 ∼ 32 行目のくり返し全体でそれぞれ最大 $|E_p|$ 回実行される．そのため，くり返し全体で $O(|E_p|)$ の時間がかかる．また，20 行目以降に分岐した場合の計算コストは 8 ∼ 19 行目と同様である．6 ∼ 32 行目のくり返しは最大 $|E_p|$ 回くり返されるので，INITIALIZE 関数の時間計算量は

$$O(|E_p| \cdot |E|) \tag{1}$$

である．

アルゴリズム MATCH の時間計算量を考える．1，9 ∼ 10，12，13，16 行目は定数時間で実行できる．2 行目はすべてのキーノードを調べる必要があるので，$O(|K|)$ の時間がかかる．3，4 行目は $|K| - 1$ 回くり返されるが，各回において fb パスを深さ優先探索で求めるため，$O(|K| \cdot (|V| + |E|))$ の時間がかかる．5 行目は u_s の入出力辺の数が最悪 $|E_p|$ であり，G のすべての辺を調べる必要があるので，$O(|E_p| + |E|)$ の時間がかかる．6 ∼ 31 行目のくり返しは u_s にマッチするノードの数だけくり返さるので，最大 $|V|$ 回くり返される．7 ∼ 8 行目は fb パスの数だけくり返され，その数は u_i の数が $|K| - 1$ であり，各 u_i までの fb パスの数が最大 $|E_p|$ なので，最大 $(|K| - 1) \times |E_p|$ 回である．また，各くり返しの中で fb パスを深さ優先探索で辿るので，合計で $O(|K| \cdot |E_p| \cdot (|V| + |E|))$ の時間がかかる．11 行目以降のくり返しは，それぞれのキーノードにマッチする候補のノード数が $|V|$ 以下で，それの $|K| - 1$ 個の直積をとるので $|V|^{|K|-1}$ 回くり返される．14 行目の INITIALIZE 関数の時間計算量は式（3.1）のように $O(|E_p| \cdot |E|)$ であり，15 行目は V_p の各ノードに対し，マッチした最大 $|V|$ の候補ノードを調べるので $O(|V_p| \cdot |V|)$ である．17 ∼ 20 行目は v が dangling node かどうか調べるため，各 u とその入出力辺，u にマッチする v とその入出力辺を見る必要があるので，$O(|E_p| \cdot |E|)$ の時間がかかる．21 ∼ 29 行目のくり返しは，Q の大きさが最悪の場合 $|V_p|$ なので，最大 $|V_p|$ 回くり返される．22，26 行目について，$S_{old}(u') \backslash S(u')$ の大きさは最悪の場合 $|V|$ である．また，$u' \xrightarrow{l} u \in E_p \wedge v \in S(u)$ となる各 $v' \xrightarrow{l} v \in E$ を得るため，u と v の入出力辺を調べる必要があり，21 行目以降のくり返し全体で $O(|E_p| \cdot |E|)$ の時間が必要である．更に，23 ∼ 25，27 ∼ 29 行目は v が dangling node になるか判定するため，v の入出力辺を調べる必要があり，v の入出力辺は最大 $|E|$ なので，$O(|E|)$ の時間がかかる．そのため，21 ∼ 29 行目のくり返しは全体で $O(|E_p| \cdot |E|^2)$ の時間がかかる．30 ∼ 31 行目はすべての u を調べるので $O(|V_p|)$ である．よって，提案手法の時間計算量は

$$O(|V|^{|K|}(|E_p| \cdot |E|^2) + |K| \cdot |E_p| \cdot |V| \cdot (|V| + |E|)) = O(|V|^{|K|}(|E_p| \cdot |E|^2)) \tag{2}$$

である．

以上の議論から，以下が成り立つ．

定理 1. 定数 c に対して $|K| \leqq c$ であるとする．このとき，Algorithm 1 は多項式時間で動作する．

$|K|$ が定数で抑えられない場合，上記時間計算量は指数オーダーとなる．しかし，以下の結果から，これを多項式オーダーにするのは困難である．

定理 2. $|K|$ が定数でない任意のサイズを許すとする．P = NP でない限り，本問題を多項式時間で解くことは困難である．

証明. P != NP かつ本問題が多項式時間で解けると仮定する．与えられたパターングラフ P とデータグラフ G に対して，P

に任意にノードを追加して，P を部分グラフとして含む（多項式サイズの）パターングラフ P' を作る．P' のキーノード集合 K を P のノードと一致するように指定する．P' と G に対して本問題を解き，得られた解から K にマッチしないノードを取り除く．これは，P にマッチする部分グラフ同型問題を解くことと同じである．本問題が多項式時間で解けることから，NP 完全である部分グラフ同型問題も多項式時間で解けることになり，P != NP に矛盾する．よって P = NP でない限り，本問題を多項式時間で解くことは困難である． □

3.3 アルゴリズムの動作例

本節では，パターングラフを図 2，データグラフを図 1 としてアルゴリズムの動作例を示す．図 2 中のノード「学生」と「教員」をキーノードとする．

まず，Algorithm 1 の 2 行目で，キーノード集合の中から入力辺と出力辺の合計が最多のキーノードを選択する．この場合，キーノード「学生」とキーノード「教員」の入出力辺の合計は共に 2 であるため，任意に「学生」を選択したとする．3，4 行目で，選択した「学生」から他のキーノード「教員」への fb パスを求める．「学生」から「教員」への fb パスは 学生 $\overset{sv}{\rightarrow}$ 教員 と 学生 $\overset{ta}{\rightarrow}$ 授業 $\overset{te}{\leftarrow}$ 教員 の 2 つである．5 行目で，選択した「学生」にマッチする候補を求める．データグラフの中で，「学生」と同じラベルの入出力辺を持つ s_1 と s_2 が候補となる．こうして求めた s_1 と s_2 それぞれに対して 6 ∼ 31 行目の処理を行う．仮に $v_s \in C(u_s)$ を s_1 として 6 行目以降の説明をする．7，8 行目で，s_1 から先ほど求めた fb パスを辿ることで，選択しなかったキーノード「教員」にマッチし得る候補を求める．この場合，候補は t_1 のみである．こうして求めた候補を用いて，12，13 行目で f(学生) $= s_1, f$(教員) $= t_1$ という単射を作成する．14 行目の INITIALIZE 関数で，作成した単射に基づきキーノードでない P のノード「授業」にマッチし得る候補の関係を求める．

ここで，INITIALIZE 関数の動作について述べる．2 行目で，始点か終点どちらかがキーノードである辺 学生 $\overset{ta}{\rightarrow}$ 授業 と 教員 $\overset{te}{\rightarrow}$ 授業 を集合 T に加える．3 ∼ 5 行目で，始点と終点が共にキーノードである辺 学生 $\overset{sv}{\rightarrow}$ 教員 に対応する辺 f(学生) $\overset{sv}{\rightarrow} f$(教員) がデータグラフ中に存在するかどうかを調べる．2 行目で求めた辺 学生 $\overset{ta}{\rightarrow}$ 授業 と 教員 $\overset{te}{\rightarrow}$ 授業 それぞれに対して 6 ∼ 32 行目の処理を行う．仮に $u \overset{l}{\rightarrow} u' \in T$ を 学生 $\overset{ta}{\rightarrow}$ 授業 として 6 行目以降の説明をする．「学生」はキーノードなので，7 ∼ 19 行目に分岐する．また，8 ∼ 11 行目は 10，11 行目の処理に分岐し，始点が「学生」にマッチし得る候補 s_1 であり，ラベルが「ta」である辺の終点を S(授業) に加える．この場合，c_1，c_2，c_3 が S(授業) に加わる．S(授業) が空でないため 12，13 行目の処理は行われない．また，隣接する頂点がキーノードでなく，マッチし得る候補も求まっていない「学生」の入出力辺が存在しないため，14 ∼ 19 行目で新たに T に加わる辺はない．教員 $\overset{te}{\rightarrow}$ 授業 に対しても同様に 6 ∼ 32 行目の処理を行い，最後に 33 行目で関係 S を返し，INITIALIZE 関数を終了する．

再び Algorithm 1 の動作について述べる．INITIALIZE 関数を実行した結果，{(授業, c_1), (授業, c_2), (授業, c_3)} という関係が求まる．15 行目で，キーノードにマッチし得る候補ノード s_1 と t_1 が求めた関係に含まれていないかを調べる．14 行目の INITIALIZE 関数で求めた関係に含まれる c_1 は dangling node であるため 17 ∼ 20 行目で削除する．もし dangling node を削除したことで，新たな dangling node が発生した場合 21 ∼ 29 行目で削除するが，この場合，新たに発生した dangling node は存在しない．30，31 行目で，求めた単射と関係の組を解の集合 R に加える．s_2 に対しても同様に 6 ∼ 31 行目の処理を行い，最後に 32 行目で解の集合 R を返し，マッチングを終了する．

4. 評価実験

本章では，提案アルゴリズムに関する評価実験について述べる．提案アルゴリズムを実装し，(1) データグラフのサイズを変化させたときのアルゴリズムの処理時間，(2) キーノードの数を変化させたときのアルゴリズムの処理時間，(3) 提案手法による解の数の 3 点を評価する．

4.1 評価用データ

評価実験に用いるデータグラフは，任意のサイズ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 [9] 及び BSBM(Berlin SPARQL Benchmark) [10] を用いて作成する．SP^{2}Bench で作成したグラフの概要を表 1 に，BSBM で作成したグラフの概要を表 2 に示す．実験に用いるパターングラフは，それぞれのベンチマークソフトが生成するグラフデータの構造に沿って人工的に作成する．SP^{2}Bench が生成するグラフに対するパターングラフとして図 3 と図 4 を，BSBM が生成するグラフに対するパターングラフとして図 5 を作成した．

表 1 SP^{2}Bench で作成したデータグラフの概要

辺数	ノード数	データサイズ (KB)
10,303	6,533	1,088
30,029	18,607	3,225
50,168	30,794	5,400
70,270	43,008	7,588
90,061	55,085	9,730

表 2 BSBM で作成したデータグラフの概要

辺数	ノード数	データサイズ (KB)
40,177	14,205	10,174
115,387	37,683	29,370
190,496	60,668	48,548
260,922	80,784	66,611
331,385	100,817	84,713

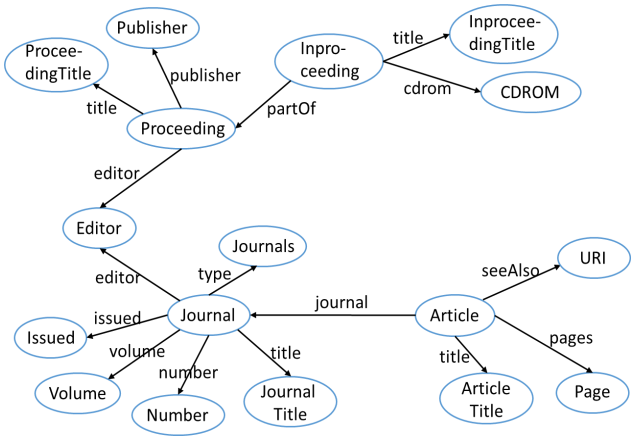

図 3　作成したパターングラフ (1)

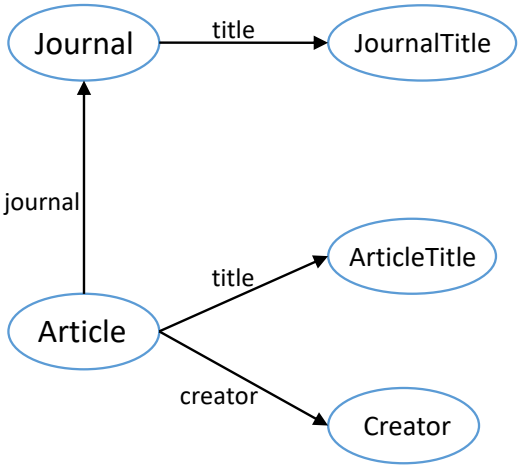

図 4　作成したパターングラフ (2)

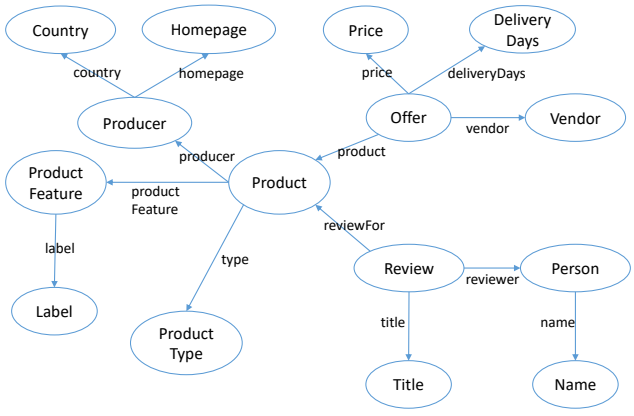

図 5　作成したパターングラフ (3)

4.2　評価実験の詳細

評価実験の環境は以下の通りである．

OS:　Windows 10 Home (64bit)

CPU:　Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz

主記憶:　8.00GB

使用言語:　Ruby 2.3.1

以下，得られた結果について述べる．

4.2.1　実験 1:データグラフのサイズを変化させたときのアルゴリズムの処理時間

はじめに，SP^{2}Bench で作成した 5 つのデータグラフ（表 1）を対象に，パターングラフを図 3，キーノードの数を 3 としてマッチングにかかる時間を計測した．処理時間の計測は Windows PowerShell の Measure-Command を用いた．計測した結果を図 6 に示す．また，BSBM で作成した 5 つのデータグラフ（表 2）を対象に，パターングラフを図 5，キーノードの数を 1 としてマッチングにかかる時間を計測した．計測した結果を図 7 に示す．実験の結果から，キーノードの数を定数とすると，提案アルゴリズムの処理時間はデータグラフのサイズ（辺数）に関してほぼ線形であることがわかる．

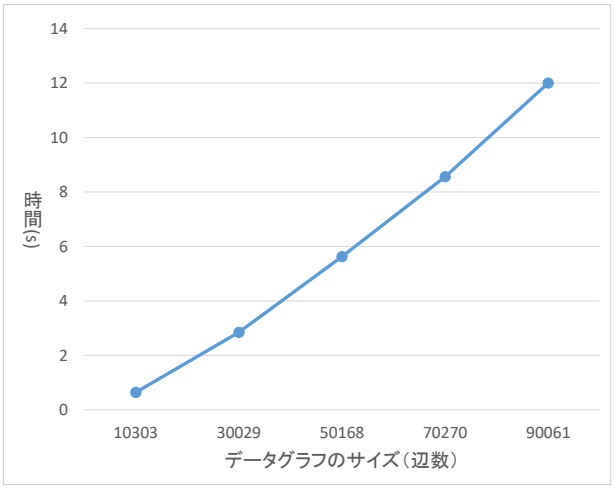

図 6　データグラフの変化と処理時間 (1)

4.2.2　実験 2:キーノードの数を変化させたときのアルゴリズムの処理時間

SP^{2}Bench で作成した 5 つのデータグラフ（表 1）のうち辺数 50168 のグラフを対象に，パターングラフを図 3 としてキーノード数 1 から 6 までのマッチングにかかる時間を計測した．計測した結果を図 8 に示す．また，BSBM で作成した 5 つのデータグラフ（表 2）のうち辺数 40177 のグラフを対象に，パターングラフを図 5 としてキーノード数 1 から 5 までのマッチングにかかる時間を計測した．計測した結果を図 9 に示す．実験の結果から，キーノード数を変数とすると，提案アルゴリズムの実行時間は線形より大きくなることがわかる．

4.2.3　実験 3:提案手法と部分グラフ同型問題の解の数

SP^{2}Bench で作成した 5 つのグラフ（表 1）のうち辺数 50168 のグラフを対象に，パターングラフを図 4 としてマッチングを

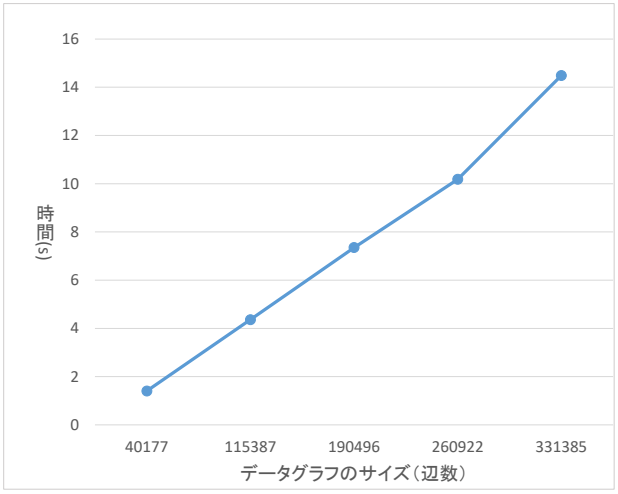

図 7　データグラフの変化と処理時間 (2)

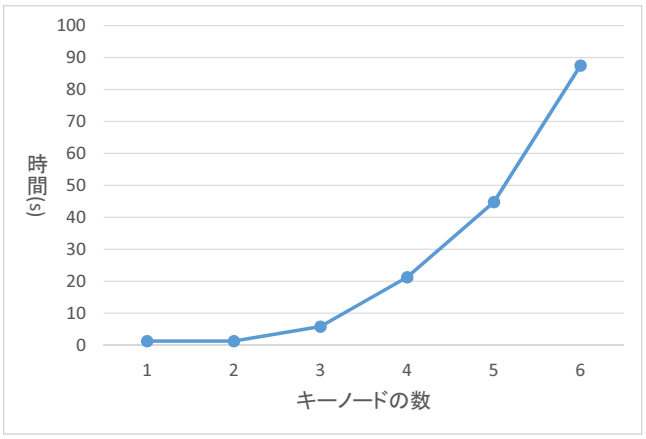

図 8　キーノード数の変化と処理時間 (1)

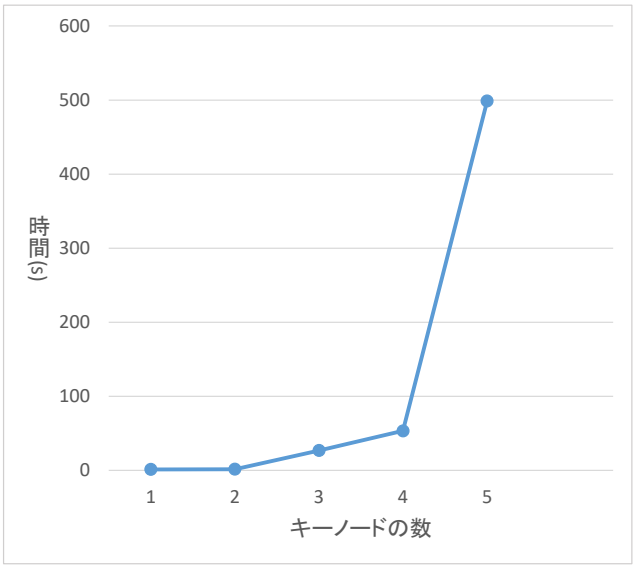

図 9　キーノード数の変化と処理時間 (2)

行い，提案手法と部分グラフ同型問題の解の数を計測した．ただし，検索目的を「雑誌ごとに，雑誌に含まれる論文のタイトルと著者の一覧（タイトルと著者の対応は求めない）」とし，提案手法におけるキーノードは「Journal」のみとして計測した．得られた結果を図 10 に示す．部分グラフ同型問題による解の数が 4811 個に対し，提案手法による解の数は 104 個であった．対象としたグラフに含まれている雑誌ノードの数が 104 個なので，提案手法による解の数は検索目的を満たす必要最小限の数だと言える．実験の結果から，部分グラフ同型問題の解の数が必要以上に大きくなるのに対し，提案手法は検索目的に適したキーノードを指定することで解の数を抑えることが可能であるとわかる．

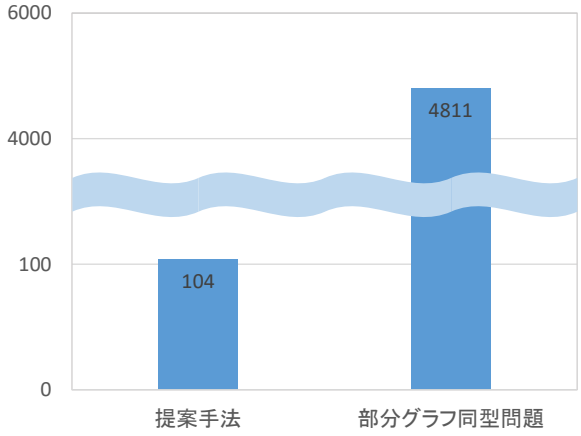

図 10　提案手法と部分グラフ同型問題の解の数

4.3　考　　察

前章の式 (2) より，Algorithm 1 の時間計算量は線形よりも大きくなっている．しかし，式 (2) の時間計算量は最悪の場合のものであり，実験 1 の結果から，実際にはより効率よく動作することが期待される．

また，実験 2 の結果から，実行時間はキーノード数に関して指数関数的に増加する．定理 2 の結果から，キーノード数を変数とした場合は本問題は計算困難であり，実験 2 の結果はこのことを反映したものといえる．このことから，キーノードを増加させた場合の，実行時間の上昇を抑えるのは本質的に困難であると考えられる．

5.　む　す　び

本稿では，部分グラフ同型問題と模倣関係を融合した新たな概念を定義し，定義した新概念に基づくパターンマッチングアルゴリズムを提案した．具体的には，パターングラフのノードに対して，その中からキーノードを任意に設定し，キーノードに対しては部分グラフ同型問題の定義，それ以外のノードに対しては模倣関係の定義を適用してマッチングを行うアルゴリズムである．また，提案アルゴリズムの処理時間を評価するため Ruby 言語を用いて実装し，評価実験を行った．その結果，(1) キーノードの数を定数とすると，データグラフのサイズに対して線形の処理時間でマッチングを行えること，(2) キーノードの数を変数とすると，キーノード数の増加に従い，処理時間が指数関数的に増加することを確認した．

今後の課題としては，本稿の提案アルゴリズムではマッチし

ない情報を検索するため，融合させるパターンマッチの定義を修正あるいは拡張することが考えられる．また，本稿が提案したアルゴリズムはすべての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，データグラフのサイズが主記憶のサイズを大幅に超える場合，本稿の提案アルゴリズムを適用することは困難である．そこで，主記憶のサイズを大幅に超える大規模グラフに対してもマッチングが行えるようアルゴリズムを拡張することが重要な課題であると考えられる．

文　　献

[1] J.R. Ullmann, “An algorithm for subgraph isomorphism,” J. ACM, 1976, 23(1), pp. 31–42.

[2] 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.

[3] H. He and A. K. Singh, “Closure-tree: An index structure for graph queries,” Proc. ICDE, pp.38–49, Atlanta, USA, April. 2006.

[4] Huahai He, Ambuj K. Singh. “Graphs-at-a-time: query language and access methods for graph databases,” Proc. SIGMOD, pp. 405–418, Vancouver, Canada, June. 2008.

[5] Shijie Zhang, Shirong Li, Jiong Yang, “GADDI: distance index based subgraph matching in biological networks,” Proc. EDBT, pp. 192–203, Bordeaux, France, March. 2009.

[6] Peixiang Zhao, Jiawei Han, “On graph query optimization in large networks,” Proc. VLDB, pp. 340–351, Singapore, Sept. 2010.

[7] Monika R. Henzinger, Thomas A. Henzinger, Peter W. Kopke, “Computing Simulations on Finite and Infinite Graphs,” Foundations of Computer Science, 1995, pp. 453–462.

[8] Milner, R. Communication and Concurrency. Prentice Hall, 1989.

[9] Michael Schmidt, Thomas Hornung, Georg Lausen, and Christoph Pinkel, “SP2Bench: a SPARQL Performance Benchmark,” Proc. ICDE, pp. 222–234, Shanghai, China, 2009.

[10] Chris Bizer, Andreas Schultz, “Berlin SPARQL Benchmark (BSBM) - Benchmark Rules,” http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/spec/ BenchmarkRules/index.html, 参照 Feb. 3, 2017.