

# 部分グラフ同型問題と模倣関係の融合的アプローチ

菅原 知倫<sup>1</sup> 鈴木 伸崇<sup>1</sup>

**概要：**グラフデータにおけるパターンマッチの主な方法として、部分グラフ同型問題と模倣関係がある。前者は、与えられたパターンと同型の部分グラフを解として求める。しかし、解はパターンと正確に同型である必要があるため、本来は同じ解に属するべき、互いに関連するノードが異なる解に分かれてしまう場合がある。一方、後者では、解をより効率よく求めることが可能であるが、互いに関連の無いノードが1つの解（関係）に属してしまうことがある。本稿では、部分グラフ同型問題と模倣関係の融合的アプローチを提案する。本手法の特徴は、互いに強く関連するノードは1つの解として得ることが可能で、かつ、関連の無いノードが同じ解には属さない、という点にある。

**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

グラフデータにおける最も基本的かつ重要なパターンマッチの方法として、部分グラフ同型問題 (subgraph isomorphism) と模倣関係 (graph simulation) がある。前者は、与えられたパターンと同型の部分グラフを解として求める。しかし、解はパターンと正確に同型である必要があるため、本来は同じ解に属するべき、互いに関連するノードが異なる解に分かれてしまう場合がある。さらに、この問題は NP 困難であり、解を求めるには大きな計算コストを要する。一方、後者は解をより効率よく求めることが可能であるが、互いに関連の無いノードが1つの解（関係）に属してしまうという問題がある。

そこで本稿では、部分グラフ同型問題と模倣関係の融合的アプローチを提案する。本アプローチでは、パターングラフは2種類のノード（キーノード、および、それ以外のノード）から成る。ユーザは、パターングラフのノードから任意の数のノードをキーノードとして指定する。キーノードは部分グラフ同型問題の概念が適用されるノードであり、キーノードでないノードには模倣関係の概念が適用される。 $P = (V_p, E_p)$  をパターングラフ、 $K \subseteq V_p$  をキーノードの集合とする。 $K$  は任意のノードを  $V_p$  から選択できるため、提案アプローチは部分グラフ同型問題と模倣関係の中間的な概念を表すことができる (図 1)。本アプローチの特徴は、互いに強く関連するノードは1つの解として得ることが可能で、かつ、関連の無いノードが同じ解には

属さない、という点にある。

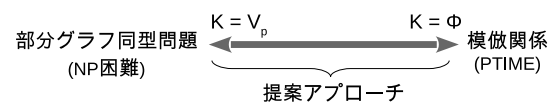


図 1 部分グラフ同型問題および模倣関係と提案アプローチとの関係

簡単な例として、図 2 に示されるパターングラフ  $P$  とデータグラフ  $G$  を考える。図 2 において、 $u_1$  は学生、 $u_2$  は教員、 $u_3$  は科目を表す。図 3 に (a) 部分グラフ同型問題、および、(b) 模倣関係、(c) 提案アプローチ ( $u_1$  をキーノードとする)、により得られる解集合を示す。 $u_1$  をキーノードとして指定することにより、提案アプローチではそれぞれの学生ノードに対して解が得られていることがわかる。この例では、解は  $s_1$  および  $s_2$  によりグループ化されており、 $s_1$  によって履修されている科目  $c_2$  と  $c_3$  は同じ解にまとめられている (図 3(c))。このような解は、部分グラフ同型問題を解くことでは得られず、図 3(a) に示すように  $c_2$  と  $c_3$  は異なる解に属している。

本稿では、提案アプローチに基づいて、パターングラフとデータグラフから解を求めるためのアルゴリズムを示す。提案アプローチに基づく解を求める別の方法として、まず部分グラフ同型問題を解き、得られた解をキーノードでグループ化するという方法が考えられる。例えば、図 3(c) の1つ目の解は、図 3(b) の1つ目と2つ目の解をグループ化することにより得られる。しかし、そのような手法は明らかに非効率である。実際、評価実験の結果、提案手法は、上記の方法よりも効率よく解が得られるという結果が得ら

<sup>1</sup> 筑波大学  
University of Tsukuba

れた。

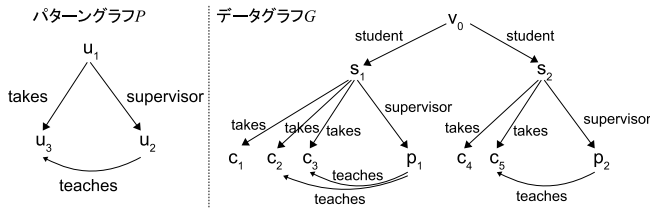
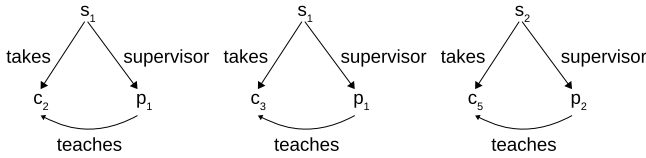


図 2 パターングラフとデータグラフの例

(a) 部分グラフ同型問題



(b) 模倣関係

$u_1$	$s_1$
$u_1$	$s_2$
$u_2$	$p_1$
$u_2$	$p_2$
$u_3$	$c_2$
$u_3$	$c_3$
$u_3$	$c_5$

(c) 提案アプローチ

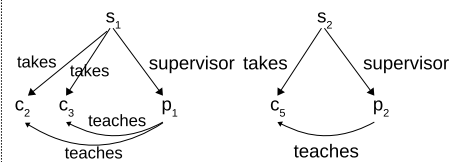


図 3 3つのアプローチによる解

## 関連研究

部分グラフ同型問題を解くアルゴリズムは数多く提案されている。Ullmann アルゴリズムは、この問題を解く最初のアプローチである [9]。この問題は NP 困難であるため、計算の効率化を目的として様々なアルゴリズムが提案されている (VF2 [6], QuickSI [8], CFL-Match [1] など)。これらに共通する特徴は、バックトラックを用いる点である。すなわち、部分解をノードにより拡大し、拡大された部分解が解となり得ない場合、直前の部分解に戻り、直前に拡大したノードの「次のノード」により部分解を拡大する、という手順で解を探索する。また、無駄なバックトラックを減らし効率化を図った手法も提案されている [10]。

模倣関係について、多項式時間で解が得られることが示されている [3]。さらに、模倣関係に対する拡張がいくつか提案されている (文献 [2], [5] など)。しかし、著者の知る限り、部分グラフ同型問題と模倣関係の融合的なアプローチは提案されていない。

## 2. 諸定義

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。ノード  $u$  からノード  $v$  へのラベル  $l$  をもつエッジを  $u \xrightarrow{l} v$  と表す。

ノード  $v$  へ入力するエッジの集合を  $In(v)$ 、 $v$  から出力するエッジの集合を  $Out(v)$  と表す。 $v$  へ入力するエッジのラベルから成る集合を  $lab_{in}(v)$  と表す。すなわち、 $lab_{in}(v) = \{l \mid u \xrightarrow{l} v \in In(v), u \in V\}$  である。同様に、 $v$  から出力するエッジのラベルから成る集合を  $lab_{out}(v)$  と表す。すなわち、 $lab_{out}(v) = \{l \mid v \xrightarrow{l} u \in Out(v), u \in V\}$  である。

パターンと問合せ対象のデータをいずれもグラフとして表す。 $P = (V_p, E_p)$  をパターングラフ、 $G = (V, E)$  をデータグラフとする。部分グラフ同型問題と模倣関係は次のように定義される。

部分グラフ同型問題： 「任意の  $u \xrightarrow{l} u' \in E_p$  に対してエッジ  $f(u) \xrightarrow{l} f(u') \in E$  が存在する」という条件を満たす単射  $f: V_p \rightarrow V$  を発見せよ。

模倣関係： 二項関係  $S \subseteq V_p \times V$  で、任意の  $u \xrightarrow{l} u' \in E_p$  に対して次の条件を満たすものを発見せよ。

- もし  $(u, v) \in S$  ならば、 $(u', v') \in S$  なるエッジ  $v \xrightarrow{l} v' \in E$  が存在する。
- もし  $(u', v') \in S$  ならば、 $(u, v) \in S$  なるエッジ  $v \xrightarrow{l} v' \in E$  が存在する。

## 3. 提案手法

### 3.1 部分グラフ同型問題と模倣関係の融合的アプローチ

まず、パターングラフの概念をキーノードを用いて拡張する。形式的には、パターングラフは三次組  $P = (V_p, K, E_p)$  として定義される。ここで、 $((V_p \cup K), E_p)$  はグラフ、 $V_p \cap K = \emptyset$ 、かつ、 $K$  はキーノード集合である。部分グラフ同型問題の概念は  $K$  に属するノードに適用され、模倣関係の概念は  $V_p$  に属するノードに適用される。 $f: K \rightarrow V$  を単射、 $S \subseteq V_p \times V$  を二項関係とする。パターングラフ  $P = (V_p, K, E_p)$  とデータグラフ  $G = (V, E)$  に対して、組  $(f, S)$  が次の条件を満たすとき、 $(f, S)$  を  $P$  と  $G$  の解と呼ぶ。

- 各エッジ  $u \xrightarrow{l} u' \in E_p$  に対して、次の条件が成り立つ。
  - $u \in K$  の場合：  $f(u) = v$ 、かつ、「 $f(u') = v'$  ( $u' \in K$  のとき) または  $(u', v') \in S$  ( $u' \in V_p$  のとき)」を満たすエッジ  $v \xrightarrow{l} v' \in E$  が存在する。
  - $u \in V_p$  の場合： 各  $(u, v) \in S$  に対して、「 $f(u') = v'$  ( $u' \in K$  のとき) または、 $(u', v') \in S$  ( $u' \in V_p$  のとき)」を満たすエッジ  $v \xrightarrow{l} v' \in E$  が存在する。
- 各エッジ  $u' \xrightarrow{l} u \in E_p$  に対して、次の条件が成り立つ。
  - $u \in K$  の場合：  $f(u) = v$ 、かつ、「 $f(u') = v'$  ( $u' \in K$  のとき) または  $(u', v') \in S$  ( $u' \in V_p$  のとき)」を満たすエッジ  $v' \xrightarrow{l} v \in E$  が存在する。
  - $u \in V_p$  の場合： 各  $(u, v) \in S$  に対して、「 $f(u') = v'$  ( $u' \in K$  のとき) または  $(u', v') \in S$  ( $u' \in V_p$  のとき)」を満たすエッジ  $v' \xrightarrow{l} v \in E$  が存在する。

$P$  と  $G$  の解  $(f, S)$  に対して、もし  $P$  と  $G$  の任意の解  $(f, S')$

に対して  $S' \subseteq S$  ならば,  $(f, S)$  は  $f$  に関して最大であるという.

### 3.2 提案アルゴリズム

提案アプローチに基づいて解を求めるアルゴリズムを示す. 提案アプローチは, 部分グラフ同型問題を解くための特定のアルゴリズムに依存しないが, アルゴリズムを具体的に記述するために, 本稿では文献 [4] における QuickSI [8] の記述法に基づいて提案アルゴリズムを記述する. QuickSI を用いたのは, 記述が簡潔であり, 文献 [4] で比較されているアルゴリズムの中では良好なパフォーマンスを示しているからである.

#### 3.2.1 パターングラフのノード訪問順

$P$  をパターングラフ,  $G$  をデータグラフとする.  $P$  におけるノードの訪問順を決めるために,  $P$  の最小全域木を用いる (この部分は QuickSI に準じている).  $P$  の各エッジに,  $G$  におけるエッジの出現頻度に基づいて重みを割り当て, Prim のアルゴリズムを用いて  $P$  の全域木を作成する. 全域木を格納するために表  $SEQ$  を用いる.  $SEQ$  において,  $T_i.node$  はノード,  $T_i.parent$  は  $T_i.node$  の親ノード,  $T_i.dir$  は  $T_i.parent$  と  $T_i.node$  間のエッジの方向,  $T_i.l$  はそのエッジのラベルを表す.  $SEQ$  は追加的な行  $R_j$  を含む場合がある.  $R_j$  には全域木に出現しない  $P$  のエッジ (残余エッジ) を格納する. 例えば, 図 4 の重み付きパターングラフ  $P$  に対する  $SEQ$  を表 1 に示す. この表  $SEQ$  は, 提案アルゴリズムがノードを  $u_1, u_2, u_3, u_4$  の順に訪れることを示している.

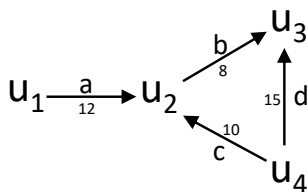


図 4 重み付きパターングラフ  $P$

表 1 図 4 のグラフに対する表  $SEQ$

Type	$T_i.node$	$[T_i.dir, T_i.l, T_i.parent]$
$T_0$	$u_1$	
$T_1$	$u_2$	[in, a, $u_1$ ]
$T_2$	$u_3$	[in, b, $u_2$ ]
$T_3$	$u_4$	[out, c, $u_2$ ]
$R_1$	$u_4$	[out, d, $u_3$ ]

#### 3.2.2 アルゴリズムの詳細

Algorithm 1 に提案アルゴリズム HybridMatch を示す.  $P = (V_p, K, E_p)$  をパターングラフ,  $G = (V, E)$  をデータグラフとする. まず, ノード  $u \in K \cup V_p$  を選択し  $V_T$  と  $SEQ$  に追加する (3 行目). 次に, エッジの重み  $w(e)$  に基

#### Algorithm 1 HybridMatch

**Input:** 重み付きパターングラフ  $P = (V_p, K, E_p)$ , データグラフ  $G = (V, E)$

**Output:**  $P$  と  $G$  の最大解  $(f, S)$

```

1: Initialize  $SEQ$  to an empty table
2:  $V_T := \emptyset$ 
3: Choose a node  $u \in K \cup V_p$ , add  $u$  to  $V_T$ , and add  $u$  to  $SEQ$ 
4: while  $|V_T| \neq |V_p| + |K|$  do
5:    $P := \{e \mid e \in E_p \text{ such that } e.u \in V_T \wedge e.u' \notin V_T\}$ 
6:   Select an edge  $e \in P$  such that  $w(e)$  is minimum in  $P$ 
7:   Add  $e$  to  $SEQ$ ,  $V_T := V_T \cup \{e.u'\}$ 
8: for each  $e \in E_p \setminus SEQ$  such that  $e.u \in K \wedge e.u' \in K$  do
9:   Add  $e$  to  $SEQ$  as an extra edge
10: Let  $f$  be an empty function
11:  $S := \emptyset$ 
12:  $d := 0$ 
13: Search( $P, G, SEQ, f, S, d$ )

```

づいて  $P$  の最小全域木を求める. ここで, エッジが全域木に追加された順にエッジを  $SEQ$  に追加する (4~7 行目). そして, 残りのエッジを残余エッジとして  $SEQ$  に追加する (8~9 行目). 最後に, 後述の Search を呼び出し,  $P$  と  $G$  の全ての最大解  $(f, S)$  を発見する (13 行目).

次に, 解を発見するための手続き Search を示す (Algorithm 2). 二項関係  $S$  に対して,  $(u, v) \in S$  のとき,  $v \in S(u)$  と書く. Search では無効なノードという概念を用いる. 無効なノードとは,  $S$  に候補として追加されたものの, 模倣関係の条件を満たしておらず,  $S$  から削除すべきノードのことである. 形式的には, もしあるエッジ  $u' \xrightarrow{l} u \in E_p$  に対して  $v \in S(u)$  であるが  $E$  が  $v' \in S(u')$  なるエッジ  $v' \xrightarrow{l} v$  をもたない場合,  $v$  は無効であるという. 同様に, もしあるエッジ  $u' \xrightarrow{l} u \in E_p$  に対して,  $v' \in S(u')$  であるが  $E$  が  $v \in S(u)$  なるエッジ  $v' \xrightarrow{l} v$  をもたない場合,  $v'$  は無効であるという.

Search は,  $G$  を再帰的に探索して  $P$  にマッチするノードを発見する.  $d$  は再帰呼び出しの深さを表す. まず,  $d = |V_p| + |K|$ , すなわち, 最も深いノードまで探索を終えている場合, Search は以下の処理を行う (1~8 行目). まず,  $SEQ$  の各残余エッジに対して, 対応する  $G$  のエッジが存在するかどうかをチェックし, 対応するものがない残余エッジが存在した場合, 解として空を返す (2~4 行目). 次に,  $f$  の定義域 (キーノード) を  $S$  から削除し, DeleteDangling (Algorithm 4) を用いて  $S$  から無効なノードを削除する (5~6 行目). 最後に, 得られた  $(f, S)$  を解として返す (8 行目). 次に,  $d < |V_p| + |K|$  の場合, まず FilterCandidates (Algorithm 3) を呼び出し,  $SEQ$  における  $d$  番目のノード  $u$  にマッチし得る候補ノードの集合  $C(u)$  を求める (11 行目). もし  $d$  番目のノード  $u$  がキーノード

---

**Algorithm 2** Search( $P, G, SEQ, f, S, d$ )

---

```

1: if  $d = |V_p| + |K|$  then
2:   for each extra edge  $u \xrightarrow{l} u' \in SEQ$  do
3:     if  $f(u) \xrightarrow{l} f(u') \notin E$  then
4:       return
5:    $S := S \setminus \{(u, v) \in S \mid f(u') = v \text{ for every } u' \in K\}$ 
6:   DeleteDangling( $P, G, f, S$ )
7:   if  $S(u) \neq \emptyset$  for every  $u \in V_p$  then
8:     report ( $f, S$ )
9: else
10:   $u := T_d.node \in SEQ$ 
11:   $C(u) := \text{FilterCandidates}(P, G, SEQ, f, S, d)$ 
12:  if  $u \in K$  then
13:    for each  $v \in C(u)$  such that  $v$  is not yet matched do
14:       $f(u) := v$ 
15:      Search( $P, G, SEQ, f, S, d + 1$ )
16:       $f(u) := nil$ 
17:  else
18:     $S := S \cup \{(u, v) \mid v \in C(u)\}$ 
19:    Search( $P, G, SEQ, f, S, d + 1$ )
20:     $S := S \setminus \{(u, v) \mid v \in C(u)\}$ 

```

---

であった場合、各候補ノード  $v \in C(u)$  に対して以下を行う： $f(u) = v$  と設定し、Search を再帰的に呼び出して解を探索し、 $f$  の状態を元に戻す (14~16 行目)。一方、 $u$  がキーノードでなかった場合、 $C(u)$  に基づいて  $S$  を更新し、Search を再帰的に呼び出して解を探索し、 $S$  の状態を元に戻す (18~20 行目)。

### 3.3 アルゴリズムの正当性と時間計算量

本節では、アルゴリズムの正当性と時間計算量について簡単に述べる。紙面の都合上、証明は省略する。まず、次の定理が成り立つ。

**定理 1**  $R$  を、パターングラフ  $P = (V_p, K, E_p)$  とデータグラフ  $G = (V, E)$  に対する HybridMatch の出力とする。このとき、 $(f, S) \in R$  であることと、 $(f, S)$  が  $P$  と  $G$  の  $(f$  に関する) 最大解であることは同値である。□

次に、HybridMatch の時間計算量について考える。次の定理が成り立つ。

**定理 2** パターングラフ  $P = (V_p, K, E_p)$  とデータグラフ  $G = (V, E)$  に対して、HybridMatch の時間計算量は

$$O(V_{PL}^{|K|+1} + |E| \cdot V_{PL}^2) \quad (1)$$

である。ここで、 $V_{PL} = \max(V_{PL}^{in}, V_{PL}^{out})$  であり、 $V_{PL}^{in}(l)$  をラベル  $l$  の入力エッジをもつノードの集合としたとき、 $V_{PL}^{in} = \max_{l \in \Sigma_P} |V_{PL}^{in}(l)|$  と定義され、 $V_{PL}^{out}$  は出力エッジに関して  $V_{PL}^{in}$  と同様に定義される。特に、次の条件が成り立つ。

(1) 定数  $c$  に対して  $|K| \leq c$  のとき、HybridMatch は多項

---

**Algorithm 3** FilterCandidates( $P, G, SEQ, f, S, d$ )

---

**Input:** 重み付きパターングラフ  $P = (V_p, K, E_p)$ , データグラフ  $G = (V, E)$ , 単射  $f : K \rightarrow V$ , 二項関係  $S \subseteq V_p \times V$ , 深さ  $d$

**Output:**  $u$  にマッチし得る候補ノードの集合  $C(u)$

```

1:  $T := T_d \in SEQ$ 
2:  $u := T.node, u_{par} := T.parent$ 
3: if  $d = 0$  then
4:    $C(u) := \{v \in V \mid lab_{in}(u) \subseteq lab_{in}(v) \wedge lab_{out}(u) \subseteq lab_{out}(v)\}$ 
5:   return  $C(u)$ 
6: if  $u_{par} \in K$  then
7:   if  $u \xrightarrow{l} u_{par} \in E_p$  then
8:      $C(u) := \{v \mid v \xrightarrow{l} f(u_{par}) \in E\}$ 
9:   else if  $u \xleftarrow{l} u_{par} \in E_p$  then
10:     $C(u) := \{v \mid v \xleftarrow{l} f(u_{par}) \in E\}$ 
11:  else
12:    if  $u \xrightarrow{l} u_{par} \in E_p$  then
13:       $C(u) := \{v \mid v \xrightarrow{l} v' \in E \wedge (u_{par}, v') \in S\}$ 
14:    else if  $u \xleftarrow{l} u_{par} \in E_p$  then
15:       $C(u) := \{v \mid v \xleftarrow{l} v' \in E \wedge (u_{par}, v') \in S\}$ 
16:  return  $C(u)$ 

```

---



---

**Algorithm 4** DeleteDangling( $P, G, f, S$ )

---

```

1:  $Q := \emptyset$ 
2: for each  $u \in V_p$  do
3:   if  $S(u)$  contains a dangling node then
4:      $S_{old}(u) := S(u)$ 
5:     Delete every dangling node from  $S(u)$ , and add  $u$  to  $Q$ 
6: while  $Q \neq \emptyset$  do
7:   Delete a node  $u'$  from  $Q$ 
8:   for each  $v' \in S_{old}(u') \setminus S(u')$  and each  $v' \xrightarrow{l} v \in E$  such that  $u' \xrightarrow{l} u \in E_p \wedge (u, v) \in S$  do
9:     if  $v$  becomes dangling then
10:       $S_{old}(u) := S(u)$ 
11:      Delete  $v$  from  $S(u)$  and add  $u$  to  $Q$ 
12:   for each  $v' \in S_{old}(u') \setminus S(u')$  and each  $v \xrightarrow{l} v' \in E$  such that  $u \xrightarrow{l} u' \in E_p \wedge (u, v) \in S$  do
13:     if  $v$  becomes dangling then
14:       $S_{old}(u) := S(u)$ 
15:      Delete  $v$  from  $S(u)$  and add  $u$  to  $Q$ 

```

---

式時間で動作する。

(2)  $|K| \in O(\log |E_p|)$  のとき、HybridMatch は準多項式時間で動作する。

□

この定理から、提案アルゴリズムの時間計算量は  $K$  に依存する。 $K$  が最も大きい、すなわち、 $K$  が  $P$  のノード

集合と一致する ( $V_p = \emptyset$ ) 場合、この問題は部分グラフ同型問題と等しく NP 完全となる。しかし、上記の条件は、 $K$  が著しく大きくならない限り HybridMatch は効率よく動作することを示唆している。多くの場合、 $K$  には少数のノードを指定すれば十分であると考えられる。例えば、もしユーザが文献データベースからジャーナルの情報を得たい場合、「ジャーナル」ノードをキーとして指定すれば十分である。したがって、多くの場合において、HybridMatch は効率よく動作することが期待される。

#### 4. 評価実験

本章では評価実験について述べる。実行環境は、Intel Xeon E5-2623 v3 3.0GHz CPU, 16GB RAM, 2TB SATA HDD, Linux CentOS 7 64bit であり、アルゴリズムの実装には Ruby を用いた。データセットとして、以下の 2 つを用いた。

- SP2Bench [7] は DBLP に基づいた RDF データを生成するツールである。本評価実験では、5 個のデータ (エッジ数約 50K, 100K, 150K, 200K, および、250K) を生成して利用した。
- DBPedia は、Wikipedia から抽出した構造化情報から構成されるデータである。本評価実験では、<http://benchmark.dbpedia.org/> からファイルをダウンロードし、ファイルの先頭から 50K, 100K, 300K, および、500K 個のエッジを取り出すことにより 4 個のデータを作成した。

問合せとして用いるパターングラフについて、SP2Bench と DBPedia それぞれに対して、生成されたグラフデータからエッジをランダムに選びパターングラフを自動生成するツールを作成した。このツールを用いて、SP2Bench と DBPedia それぞれに対して 4, 6, 8, 10 個のエッジをもつパターングラフをそれぞれ 10 個ずつ (計 40 個) 生成した。

##### 4.1 データグラフのサイズと実行時間

まず、データグラフのサイズを変化させた場合に、アルゴリズムの実行時間がどのように変化するかを計測した。比較対象として用いたアルゴリズムは以下の 2 つである。

- ベースライン手法：我々のアプローチに基づく解は、まず部分グラフ同型問題を解き、その解集合をキーノードで集約してグループ化することでも得られる。これをベースライン手法として実装した。この手法において、部分グラフ同型問題を解く部分は QuickSI [8] に基づいて実装を行った。これは、提案アルゴリズムの部分グラフ同型問題を解く部分が QuickSI に基づいており、両手法間の差異を明確化するのに適していると考えられるからである。以下、ベースライン手法のことを QuickSI-AG と呼ぶ。
- Dual Simulation: 模倣関係との比較を行うため、文

献 [5] の Dual Simulation を実装した。

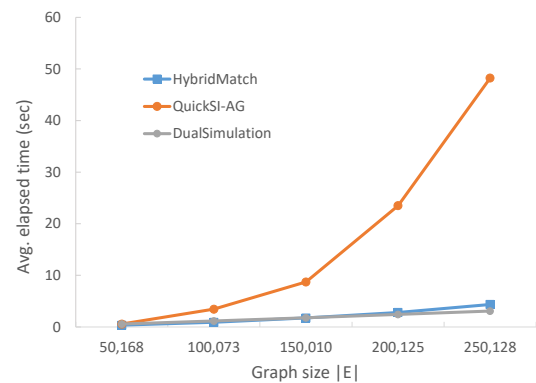


図 5 データサイズに関する実行時間の変化 (SP2Bench)

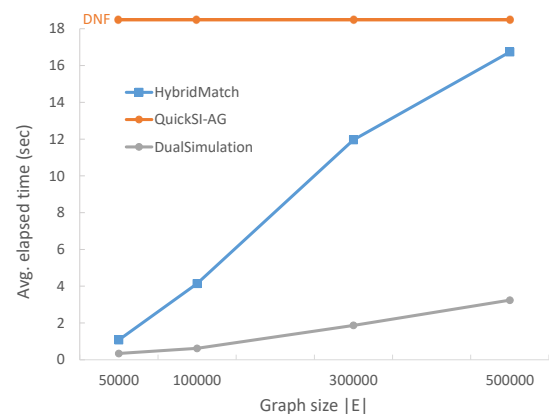


図 6 データサイズに関する実行時間の変化 (DBPedia)

この評価実験では、 $K$  のサイズを固定し ( $|K| = 3$ ), 生成された 40 個のパターングラフを用いて HybridMatch, QuickSI-AG, および, Dual Simulation を実行し、平均実行時間を求めた。  $|K| = 3$  としたのは、多くの場合、 $K$  のサイズは小さいと考えられるからである。図 5 と図 6 に結果を示す。x 軸はグラフのサイズ ( $|E|$ ) を表し、24 時間以内に終了しなかったパターンがある場合は DNF と表記している。SP2Bench データセットに対しては、HybridMatch は常に QuickSI-AG よりも高速であり、Dual Simulation と非常に近い実行時間を記録している。DBPedia データセットに対しては、QuickSI-AG はすべての場合において DNF であり、提案手法は Dual Simulation よりも実行時間を要しているものの、概ね線形時間で処理を完了している。

##### 4.2 キーノード数と実行時間

次に、 $K$  のサイズを増加させた場合に、実行時間がどのように変化するかを調べた。実験は以下に行った。

- SP2Bench データセット：エッジ数 250K のグラフを用いて、 $|K|$  の値を 0 から 10 まで 2 ずつ変化させて

HybridMatch を動作させ、40 個のパターングラフに対する平均実行時間を求めた。

- DBPedia データセット：エッジ数 500K のグラフを用いて、 $|K|$  の値を 0 から 4 まで変化させて HybridMatch を動作させ、40 個のパターングラフに対する平均実行時間を求めた ( $|K| \geq 5$  のとき、このグラフに対して HybridMatch は 24 時間以内に処理を完了できなかった)。

図 7 と図 8 に結果を示す。ここで、x 軸はパターングラフ  $P = (V_p, K, E_p)$  における  $|K|$  の値を示している。両データセットに対して、 $|K|$  が大きくなるにつれて平均実行時間も増加しており、これは概ね式 (1) に沿ったものと考えられる。

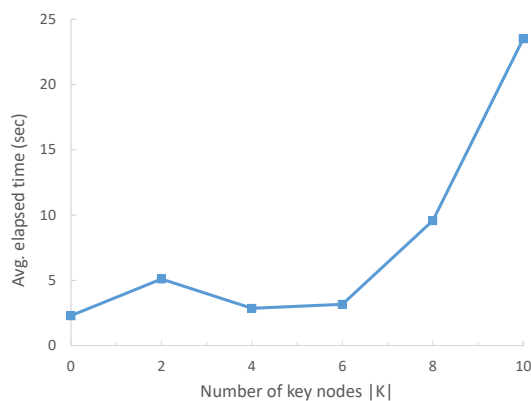


図 7 キーノード数に関する実行時間の変化 (SP2Bench)

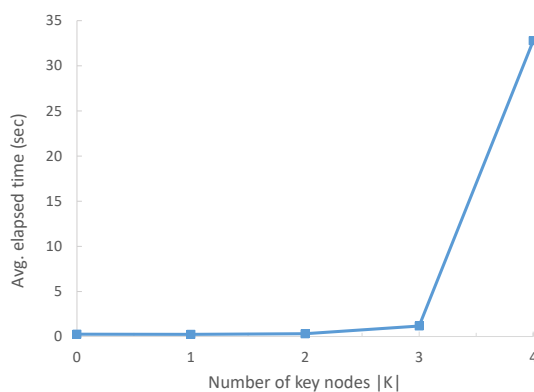


図 8 キーノード数に関する実行時間の変化 (DBPedia)

両データセットを比較すると、SP2Bench のグラフは DBLP に沿ったもので、規則正しい構造をもつ。一方、DBPedia のグラフはより複雑であり、多くの種類のノードを有している。両者のこのような特徴が、HybridMatch の実行時間に影響を与えていると考えられる。SP2Bench のグラフに対しては、HybridMatch は 25 秒以内に処理を終えている一方、DBPedia のグラフに対しては  $|K| \geq 5$  の場

合には 24 時間以内に処理が完了できなかった。したがって、HybridMatch は、データグラフがある程度規則正しい構造をもつか、 $K$  のサイズが小さい場合は DBPedia のような複雑なグラフに対しても効率よく動作するが、 $K$  のサイズが大きい場合、複雑なデータグラフでは効率の良い処理が困難な場合があると考えられる。そのような場合において、より効率よく動作するようにアルゴリズムの改善を図る必要がある。

## 5. むすび

本稿では、部分グラフ同型問題と模倣関係の融合的アプローチを導入し、解を発見するためのアルゴリズムを提案した。今後の課題として、パターングラフの表現力を拡張すること、および、提案アルゴリズムの効率を改善することを考えている。

## 参考文献

- [1] Bi, F., Chang, L., Lin, X., Qin, L. and Zhang, W.: Efficient Subgraph Matching by Postponing Cartesian Products, *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pp. 1199–1214 (2016).
- [2] Fan, W., Wang, X. and Wu, Y.: Incremental Graph Pattern Matching, *ACM Trans. Database Syst.*, Vol. 38, No. 3, pp. 18:1–18:47 (2013).
- [3] Henzinger, M. R., Henzinger, T. A. and Kopke, P. W.: Computing Simulations on Finite and Infinite Graphs, *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, FOCS'95, pp. 453–462 (1995).
- [4] Lee, J., Han, W.-S., Kasperovics, R. and Lee, J.-H.: An in-depth comparison of subgraph isomorphism algorithms in graph databases, *Proceedings of the 39th international conference on Very Large Data Bases*, PVLDB'13, pp. 133–144 (2013).
- [5] Ma, S., Cao, Y., Fan, W., Huai, J. and Wo, T.: Strong Simulation: Capturing Topology in Graph Pattern Matching, *ACM Trans. Database Syst.*, Vol. 39, No. 1, pp. 4:1–4:46 (2014).
- [6] Cordella, L., Foggia, P., Sansone, C. and Vento, M.: A (Sub)Graph Isomorphism Algorithm for Matching Large Graphs, *IEEE Trans. Pattern Anal. Mach. Intell.*, Vol. 26, No. 10, pp. 1367–1372 (2004).
- [7] Schmidt, M., Hornung, T., Lausen, G. and Pinkel, C.: SP2Bench: A SPARQL Performance Benchmark, *ICDE'09*, pp. 222–233 (2009).
- [8] Shang, H., Zhang, Y., Lin, X. and Yu, J. X.: Taming verification hardness: an efficient algorithm for testing subgraph isomorphism, *Proceedings of the VLDB Endowment*, Vol. 1, No. 1, pp. 364–375 (2008).
- [9] Ullmann, J. R.: An Algorithm for Subgraph Isomorphism, *J. ACM*, Vol. 23, No. 1, pp. 31–42 (1976).
- [10] 新井淳也, 鬼塚真, 藤原靖宏, 岩村相哲: 探索失敗履歴を用いた高速サブグラフマッチング, *DEIM 2018*, I7-4, (2018).