部分グラフ同型問題と模倣関係の融合的アプローチ

筑波大学
図書館情報メディア研究科
2019年3月
菅原　知倫

目 次

第1章　序章

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．

そこで本論文では，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 1.1）．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．

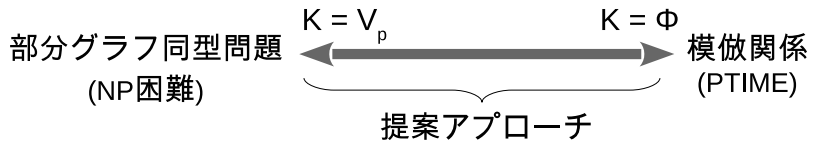

図 1.1: 部分グラフ同型問題および模倣関係と提案アプローチとの関係

簡単な例として，図 1.2 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 1.2 において，u_1 は学生，u_2 は教員，u_3 は科目を表す．図 1.3 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 1.3(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 1.3(a) に示すように c_2 と c_3 は異なる解に属している．

本論文では，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 1.3(c) の 1 つ目の解は，図 1.3(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れた．

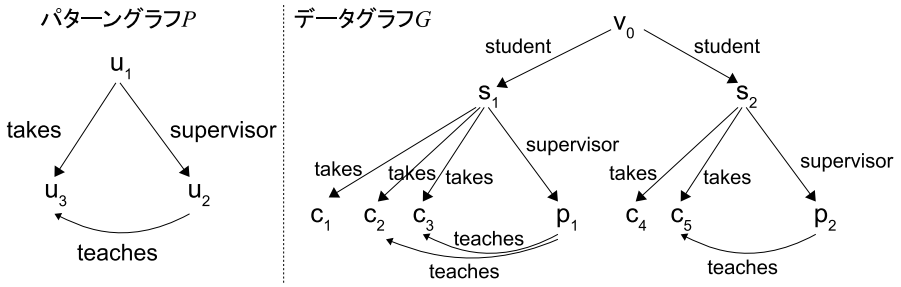

図 1.2: パターングラフとデータグラフの例

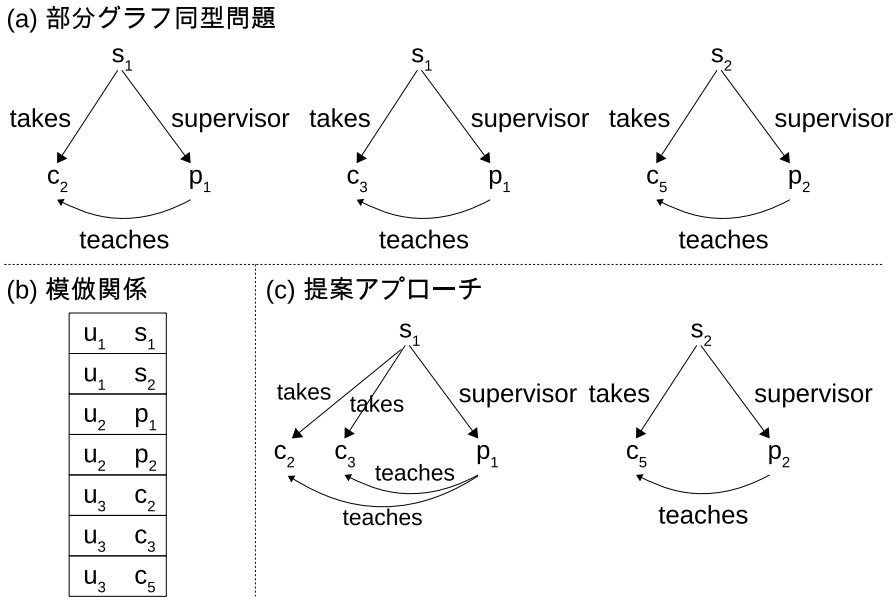

図 1.3: 3 つのアプローチによる解

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ルゴリズムである [10]．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 [7], QuickSI [9], GraphQL [3], 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 [11]．なお，3 章で示す提案アルゴリズムは一部 QuickSI に基づいて記述されているが，バックトラックに基づくアルゴリズムに適用可能である．

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 [4]．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, 6] 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．

論文の構成

本論文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の融合的アプローチ及び提案アルゴリズムについて説明する．4 章では，パターングラフの拡張について述べる．5 章では，評価実験について述べる．6 章でまとめと今後の課題について述べる．

第2章　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 $f : V_p \to V$ 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．

第3章　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．$f : K \to V$ 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．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 $V_p = \emptyset$ 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．

例：改めて図 1.2 のパターングラフ及びデータグラフを考える．u_1 をキーノードとすると，以下の二つの解が得られる．

- $f(u_1) = s_1$，　$\{(u_2, p_1),\ (u_3, c_2),\ (u_3, c_3)\}$
- $f(u_1) = s_2$，　$\{(u_2, p_2),\ (u_3, c_5)\}$

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，本論文では文献 [5] における QuickSI [9] 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 [5] 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 の最小全域木を作成する．ノードの訪問順に最小全域木を利用することで，探索の範囲を削減することができる．作成した最小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 には全域木に出現せず，両端がキーノードである P のエッジ（残余エッジ）を格納する．例えば，図 3.1 の重み付きパターングラフ P に対する SEQ を表 3.1 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．

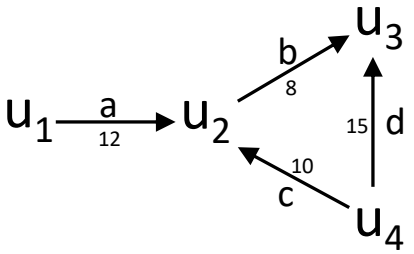

図 3.1: 重み付きパターングラフ P

表 3.1: 図 3.1 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

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: $\quad P := \{e \mid e \in E_p$ such that $e.u \in V_T \wedge e.u' \notin V_T\}$
6: $\quad$ Select an edge $e \in P$ such that $w(e)$ is minimum in P
7: $\quad$ Add e to SEQ, $V_T := V_T \cup \{e.u'\}$
8: **for** each $e \in E_p \backslash SEQ$ such that $e.u \in K \wedge e.u' \in K$ **do**
9: $\quad$ 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)

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）．初めに，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否かをチェックし，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 $C(u)$ を以下のようにして求める（11 行目）．

- d が 0 の場合，FilterCandidates は $lab_{in}(u) \subseteq lab_{in}(v) \wedge lab_{out}(u) \subseteq lab_{out}(v)$ となる v の集合を返す（3〜5 行目）．
- $v' \in V$ を u_{par} にマッチするノードとする．d が 0 より大きい場合，FilterCandidates は v', v 間のエッジの向きとラベルが u_{par}, u 間と一致するような v の集合を返す（6〜16 行目）．

$C(u)$ が求まったならば，Search に処理が戻る．もし d 番目のノード u がキーノードであった場合，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）．

最後に DeleteDangling について簡単に説明する．まず，各パターンノード $u \in V_p$ について，無効なノードを $S(u)$ から削除する（2〜5 行目）．ただし，無効なノード v を $S(u)$ から削除することで，v に隣接するノードが新たに無効なノードとなる場合がある．そのため，新たな無効なノードが発生しなくなるまで，無効なノードの削除をくり返す（6〜15 行目）．

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

if $d = |V_p| + |K|$ **then**
  for each extra edge $u \xrightarrow{l} u' \in SEQ$ **do**
    if $f(u) \xrightarrow{l} f(u') \notin E$ **then**
      return
  $S := S \backslash \{(u, v) \in S \mid f(u') = v \text{ for every } u' \in K\}$
  DeleteDangling(P, G, f, S)
  if $S(u) \neq \emptyset$ for every $u \in V_p$ **then**
    report (f, S)
else
  $u := T_d.node \in SEQ$
  $C(u) :=$ FilterCandidates(P, G, SEQ, f, S, d)
  if $u \in K$ **then**
    for each $v \in C(u)$ such that v is not yet matched **do**
      $f(u) := v$
      Search$(P, G, SEQ, f, S, d + 1)$
      $f(u) := nil$
  else
    $S := S \cup \{(u, v) \mid v \in C(u)\}$
    Search$(P, G, SEQ, f, S, d + 1)$
    $S := S \backslash \{(u, v) \mid v \in C(u)\}$

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．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． □

証明の概略． (f_m, S_m) を P と G の解とする．もし $f_m = f$ かつ $S_m \subseteq S$ ならば，(f, S) は (f_m, S_m) を覆うという．

⇐) HybridMatch は全ての $u \in K \cup V_p$ に対して，u にマッチし得る V のノードを FilterCandidates によって集める．それゆえ，どの P と G の最大解 (f_m, S_m) についても，(f_m, S_m) を覆う (f, S) が DeleteDangling が呼び出される直前に，Search によって発見される．DeleteDangling によって S から削除されたノードは，(f, S) に覆われる解の一部にはなり得ない．そのため，(f, S') を (f, S) に DeleteDangling を適用した結果とすると，以下が得られる．

$$S_m \subseteq S' \subseteq S. \tag{3.1}$$

(f, S') は無効なノードを含まないため，S' は f に関して適切な模倣関係を形成する．すなわち，(f, S') は P と G の解である．(f_m, S_m) は最大であるため，(3.1) と合わせて，S' が S_m と一致する．

⇒) 全ての無効なノードは DeleteDangling によって削除されるので，どの $(f, S) \in R$ も P と G の解となる．また，DeleteDangling は (f, S) で覆われる，解には含まれない無効なノードのみを削除するので，(f, S) は最大である． □

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 $f : K \to V$, 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: $\quad C(u) := \{v \in V \mid lab_{in}(u) \subseteq lab_{in}(v) \wedge lab_{out}(u) \subseteq lab_{out}(v)\}$
5: $\quad$ **return** $C(u)$
6: **if** $u_{par} \in K$ **then**
7: $\quad$ **if** $u \xrightarrow{l} u_{par} \in E_p$ **then**
8: $\quad\quad C(u) := \{v \mid v \xrightarrow{l} f(u_{par}) \in E\}$
9: $\quad$ **else if** $u \xleftarrow{l} u_{par} \in E_p$ **then**
10: $\quad\quad C(u) := \{v \mid v \xleftarrow{l} f(u_{par}) \in E\}$
11: **else**
12: $\quad$ **if** $u \xrightarrow{l} u_{par} \in E_p$ **then**
13: $\quad\quad C(u) := \{v \mid v \xrightarrow{l} v' \in E \wedge (u_{par}, v') \in S\}$
14: $\quad$ **else if** $u \xleftarrow{l} u_{par} \in E_p$ **then**
15: $\quad\quad C(u) := \{v \mid v \xleftarrow{l} v' \in E \wedge (u_{par}, v') \in S\}$
16: **return** $C(u)$

次に，HybridMatch の時間計算量について考える．HybridMatch は 13 行目で呼び出される Search が処理の大部分を占めている．そのため，Search の時間計算量が HybridMatch の時間計算量を決定する．Search は G のノードを再帰的に探索し，その再帰の深さは $|V_p| + |K|$ である．11 行目の $C(u)$ サイズを考える．$V^{in}(l)$ をラベル l の入力エッジをもつノードの集合とし，Σ_P を P に出現するラベル集合として，

$$V_{PL}^{in} = \max_{l \in \Sigma_P} |V^{in}(l)|.$$

同様に，$V^{out}(l)$ をラベル l の出力エッジをもつノードの集合とし，

$$V_{PL}^{out} = \max_{l \in \Sigma_P} |V^{out}(l)|.$$

最後に，

$$V_{PL} = \max(V_{PL}^{in}, V_{PL}^{out})$$

と定義する．FilterCandidates はパターングラフの各ノード u について，u と u_{par} 間のラベル l のエッジを集める．それゆえ，$|C(u)| \in O(V_{PL})$ が得られる．もし $u \in K$ ならば，15 行目の Search は $O(V_{PL})$ 回呼び出され，Search の総探索空間は $O(V_{PL}^{|K|})$ である．また，Search は 11 行目において，$O(V_{PL})$ で動作する FilterCandidates を呼び出す．よって，Search は 6 行目の DeleteDangling を除いて，$O(V_{PL} \cdot V_{PL}^{|K|})$ で動作する．

DeleteDangling について考える．v が無効なノードかどうかは v に接続するエッジによって決定する．そのため，6 行目の while ループは $O(|E|)$ 回くり返す．8 行目及び 12 行目の for ループは $O(V_{PL})$ 回くり返し，ノード v が無効かどうかチェックするのに $O(V_{PL})$ の時間がかかるので，8 から 15 行目は $O(V_{PL}^2)$ で動作する．よって DeleteDangling は $O(|E| \cdot V_{PL}^2)$ の時間を必要とする．結果として，HybridMatch の時間計算量は，

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

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 $v' \in S_{old}(u') \backslash S(u')$ 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 $v' \in S_{old}(u') \backslash S(u')$ 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

である．このことから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

$$O(V_{PL}^{|K|+1} + |E| \cdot V_{PL}^2) \tag{3.2}$$

である．

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 困難となる．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．

第4章　パターングラフの拡張

本研究が対象としている部分グラフ同型問題及び模倣関係は，常にエッジとエッジのマッピングを行う．そのため，一つのパターングラフで異なる階層の情報を検索したい場合などにおいて，パターングラフの図形的制約が問題となる場合がある．そこで，より柔軟なパターンマッチを可能とするため，パターングラフを拡張し，マッチするノード間距離を規定する新たなラベル（以下，距離ラベル）を導入する．拡張後のパターンマッチではエッジと，長さが距離ラベルの値以下であるパスのマッピングを行う．

パターングラフに距離の概念を導入した先行研究には文献 [2] がある．しかし，文献 [2] はノードラベル付き有向グラフを対象とした模倣関係の拡張である．それゆえ，エッジラベル付き有向グラフを対象に，部分グラフ同型問題と模倣関係の融合解を求める本研究にそのまま適用することは難しい．本章では文献 [2] を参考に，本研究に即した距離ラベルの定義及び解を求めるアルゴリズムについて述べる．

4.1　定義

距離ラベル k を付与したパターングラフのエッジを $u \xrightarrow[k]{l} u' \in E_p$ とする．このとき，$u' \in V_p$ に限定する．つまり，距離ラベルが指定されたエッジの終点はキーノードに指定することができない．これは，部分グラフ同型問題の場合，パターングラフのノードと解のノードとの間に必ず 1 対 1 の関係があるため，それに反しないための制限である．距離ラベル k は，正の整数もしくはシンボル「$*$」（無制限を意味する）が指定できる．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，パターングラフのエッジに距離ラベル k が指定されている場合，3 章の定義に加え，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 $u \xrightarrow[k]{l} u' \in E_p$ に対して，次の条件が成り立つ．
  - $u \in K$ の場合：$f(u) = v$，かつ，「$(u', v') \in S$」を満たす，長さ k 以内で全てのエッジラベルが l であるパス $(v, \ldots, v')$ が存在する．
  - $u \in V_p$ の場合：各 $(u, v) \in S$ に対して，「$(u', v') \in S$」を満たす，長さ k 以内で全てのエッジラベルが l であるパス $(v, \ldots, v')$ が存在する．
- 各エッジ $u' \xrightarrow[k]{l} u \in E_p$ に対して，次の条件が成り立つ．
  - 各 $(u, v) \in S$ に対して，「$f(u') = v'$ ($u' \in K$ のとき) または $(u', v') \in S$ ($u' \in S$ のとき)」を満たす，長さ k 以内で全てのエッジラベルが l であるパス $(v', \ldots, v)$ が存在する．

4.2 アルゴリズム

提案アプローチに距離ラベルを導入した場合の解を求めるアルゴリズムについて示す．ただし，処理の基本的な流れは 3 章で示したアルゴリズムと変わらないため，差分のみ記述する．

FilterCandidates(Algorithm 3)：以降の処理において，パターングラフのエッジに距離ラベル k が指定されていない場合，$k = 1$ として処理をする．9〜10 行目を以下の処理に置き換える．

---

else if $u \xleftarrow[k]{l} u_{par} \in E_p$
  $C(u) := \{v \mid$ there is a path $p = (f(u_{par}), \ldots, v)$ in G such that p.length $\leq k\}$

12〜15 行目を以下の処理に置き換える．

---

if $u \xrightarrow[k]{l} u_{par} \in E_p$ **then**
  $C(u) := \{v \mid$ there is a path $p = (v, \ldots, v')$ in G such that p.length $\leq k \wedge (u_{par}, v') \in S\}$
  $S := S \cup \{(u_{par}, v') \mid$ there is a path $p = (v, \ldots, v', \ldots, v'')$ in G such that p.length $\leq k \wedge (u_{par}, v'') \in S\}$
else if $u \xleftarrow[k]{l} u_{par} \in E_p$ **then**
  $C(u) := \{v \mid$ there is a path $p = (v', \ldots, v)$ in G such that p.length $\leq k \wedge (u_{par}, v') \in S\}$

Search(Algorithm 2)：4 行目と 5 行目の間に以下の処理を追加する．

---

for each edge $u \xrightarrow[k]{l} u' \in E_p$ that does not appear in minimumm spanning tree **do**
  if $u \in K$ **then**
    $S := S \cup \{(u', v) \mid$ there is a path $p = (f(u), \ldots, v, \ldots, v')$ in G such that p.length $\leq k \wedge (u', v') \in S\}$
  else
    $S := S \cup \{(u', v') \mid$ there is a path $p = (v, \ldots, v', \ldots, v'')$ in G such that p.length $\leq k \wedge (u, v) \in S \wedge (u', v'') \in S\}$

DeleteDangling(Algorithm 4)：アルゴリズム自体の記述に変化は無いが，無効なノードとなる条件を追加する．$u' \xrightarrow[k]{l} u \in E_p$ に対して，$v \in S(u)$ であるが，$v' = f(u')$ もしくは $v' \in S(u')$ なる v' から v へ長さが k 以内かつ全てのエッジラベルが l であるパスが存在しない場合 v は無効であるという．

第5章　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．

5.1　実験１：距離ラベルなし

距離ラベルを考慮しない通常のパターンマッチについて，提案アルゴリズムを評価する．データセットとして，以下の 2 つを用いた．

- SP2Bench [8] 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，異なるサイズの 5 つのデータ（表 5.1a）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異なる本数のエッジを取り出すことで 4 個のデータ（表 5.1b）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 個づつ（計 40 個）生成した．

表 5.1: SP2Bench 及び DBPedia データセットの概要

(a) SP2Bench

	$\|V\|$	$\|E\|$	size (MB)
1	30,794	50,168	5.1
2	61,107	100,073	10.2
3	91,223	150,010	15.4
4	121,201	200,125	20.6
5	150,921	250,128	26.0

(b) DBPedia

	$\|V\|$	$\|E\|$	size (MB)
1	32,310	50,000	9.9
2	60,831	100,000	19.8
3	167,188	300,000	58.5
4	265,844	500,000	97.0

5.1.1　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：提案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 [9] 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 [6] の Dual Simulation を実装した．

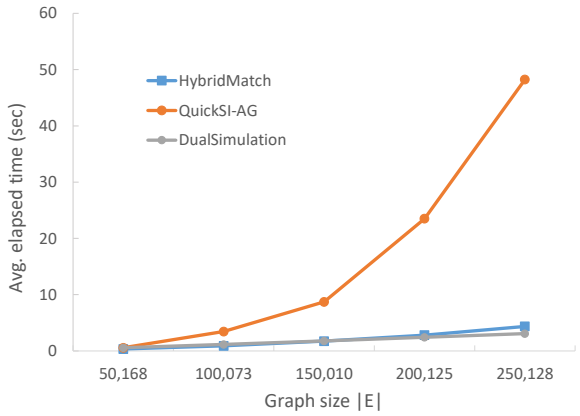

図 5.1: データサイズに関する実行時間の変化 (SP2Bench)

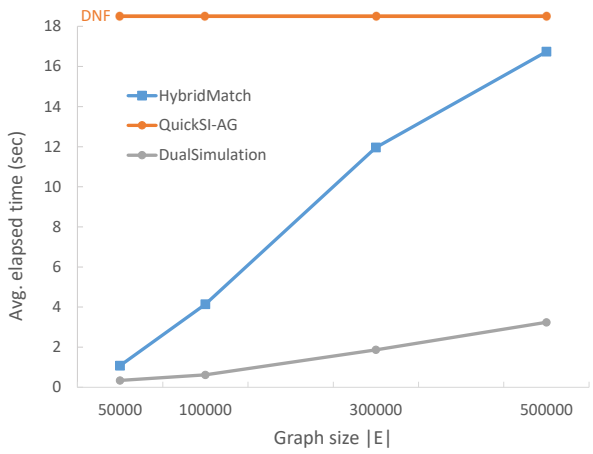

図 5.2: データサイズに関する実行時間の変化 (DBPedia)

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，DualSimulation 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 5.1 と図 5.2 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 よりは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．

5.1.2 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．実験は以下のように行った．

- SP2Bench データセット：エッジ数 250,128 のグラフを用いて，$|K|$ の値を 0 から 10 まで 2 づつ変化させて 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 50,000 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 5.3 と図 5.4 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 (3.2) に沿ったものと考えられる．

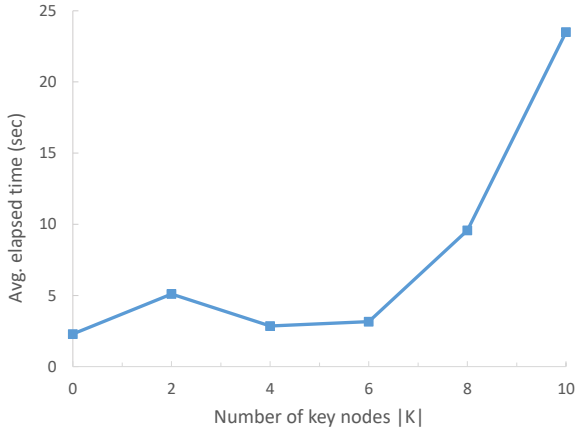

図 5.3: キーノード数に関する実行時間の変化 (SP2Bench)

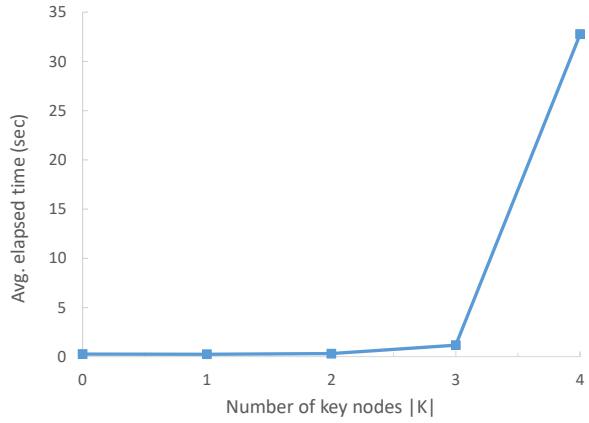

図 5.4: キーノード数に関する実行時間の変化 (DBPedia)

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 秒以内で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 時間以内で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．

5.2 実験 2：距離ラベルあり

パターングラフに距離ラベルが指定された場合の提案アルゴリズムについて評価する．使用したデータセットは以下の通りである．

- 実験 1 と同じく，SP2Bench で 5 つのデータを作成した．ただし，SP2Bench が生成するデータは同じエッジラベルが連続して出現することがないので,「Article」ノード間に「reference」ラベルを持つエッジをランダムに追加したものを利用した．5 つのデータの最終的なエッジ数は，それぞれ 11825，33965，56448，78850，100725 である．
- `http://netsg.cs.sfu.ca/youtubedata/` から YouTube の動画ネットワークデータを 4 つダウンロードし，エッジラベル付き有向グラフの形に整形したものを利用した．4 つのデータのエッジ数は，それぞれ 10651，51894，93362，139141 である．

問合せに用いたパターングラフを図 5.5 と図 5.6 に示す．図 5.5 のパターンは各論文の参照文献ならびに参照文献の参照文献を 2 階層に渡って発見するためのものである．一方，図 5.6 のパターンは各動画の評価，投稿者，関連動画ならびに関連動画の関連動画を 2 階層に渡って発見する．キーノードをそれぞれ「Article」ノードと「Video」ノードに指定し，データグラフサイズを変化させたときのアルゴリズムの動作時間を計測した．ただし，YouTube データセットは構造が画一的であり，図 5.6 のパターンで問合せをすると全 Video ノード及びその関連情報がマッチする．情報探索としての意味を持たせるため，本実験では解を「Rate」の値が 4.8 以上のものに限定して計測した．図 5.7 と図 5.8 に結果を示す．

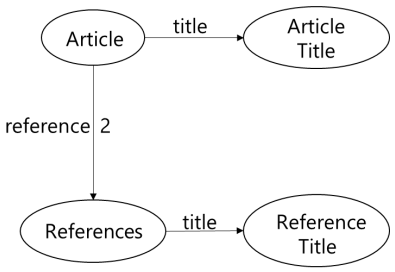

図 5.5: SP2Bench に対するパターングラフ

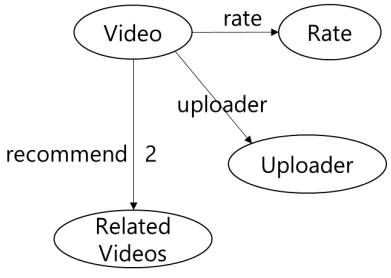

図 5.6: YouTube に対するパターングラフ

両データセットに対し，HybridMatch は概ね線形時間で処理を完了した．この結果は，図 5.1 と図 5.2 に示した距離ラベルがない場合の処理時間の傾向と非常に近しい．それゆえ，パターングラフに距離ラベルを導入するコストは小さいと考えられる．

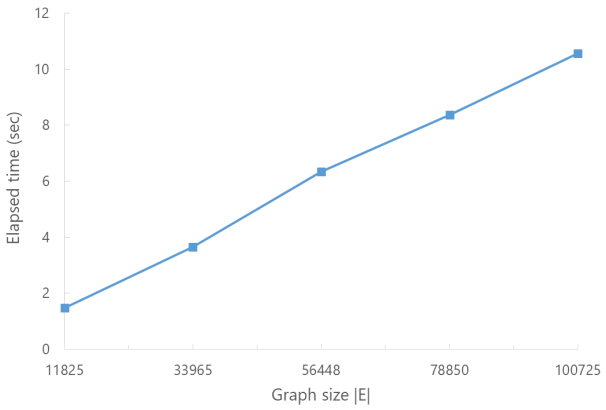

図 5.7: データサイズに関する実行時間の変化 (SP2Bench)

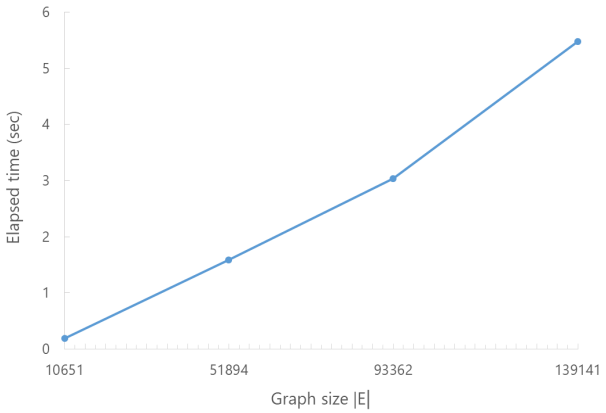

図 5.8: データサイズに関する実行時間の変化 (YouTube)

第6章　むすび

本論文では，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．提案アプローチにおいて，パターングラフのノードはキーノードとその他のノードに分けられる．キーノードに対しては部分グラフ同型問題の概念を適用し，その他のノードに対しては模倣関係の概念を適用する．評価実験の結果，提案アプローチは部分グラフ同型問題の集約手法よりも効率のよい結果を示した．

今後の課題として，距離ラベルを導入したパターンマッチの更なる評価が挙げられる．また，提案アルゴリズムの効率を改善するため，QuickSI 以外の部分グラフ同型問題のアルゴリズムについても検討したい．

謝辞

本研究を進めるに当たって，多くのご指導・御助言をいただいた鈴木伸崇准教授に深く感謝いたします.

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] He, H. and Singh, A. K.: Graphs-at-a-time: Query Language and Access Methods for Graph Databases, *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD'08, pp. 405–418 (2008).

[4] 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).

[5] 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).

[6] 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).

[7] P. 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).

[8] Schmidt, M., Hornung, T., Lausen, G. and Pinkel, C.: SP2Bench: A SPARQL Performance Benchmark, ICDE'09, pp. 222–233 (2009).

[9] 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).

[10] Ullmann, J. R.: An Algorithm for Subgraph Isomorphism, *J. ACM*, Vol. 23, No. 1, pp. 31–42 (1976).

[11] 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).