

探索失敗履歴を用いた高速サブグラフマッチング

新井 淳也^{†,††} 鬼塚 真^{††} 藤原 靖宏^{†,††} 岩村 相哲^{†††}

[†] 日本電信電話株式会社 〒180-8585 東京都武蔵野市緑町 3-9-11

^{††} 大阪大学大学院情報科学研究科 〒565-0871 大阪府吹田市山田丘 1-1

^{†††} 北海道大学 産学・地域協働推進機構 〒001-0021 北海道札幌市北区北 21 条西 11 丁目

E-mail: [†]{arai.junya,fujiwara.yasuhiro}@lab.ntt.co.jp, ^{††}onizuka@ist.osaka-u.ac.jp,

^{†††}s-iwamura@mcip.hokudai.ac.jp

あらまし 大きなグラフに含まれる、クエリグラフと同型なサブグラフへ埋め込みを列挙する問題はサブグラフマッチングと呼ばれる。この問題はデータ分析などにおいて広く利用される一方、計算コストの大きさから高速化が必要とされている。サブグラフマッチングの解法としてはバックトラッキングによる探索が一般的である。即ち前のステップで得られた部分的な埋め込みを基に新しい埋め込みを生成する手続きを再帰的に適用することで完全な埋め込みを発見する。探索中、完全な埋め込みに到達しないことが判明した部分埋め込みは探索失敗として破棄される。既存のサブグラフマッチング手法はグラフの構造を分析する前処理によってバックトラッキングの探索範囲を狭めることで失敗を削減してきた。しかし前処理は単純なルールに基づいているため、削減可能な失敗は限定的である。本論文では、「失敗から学ぶ」サブグラフマッチングアルゴリズムを提案する。提案手法はバックトラッキングの最中に探索失敗の履歴から失敗となる部分埋め込みのパターンを抽出する。そしてパターンに符合した部分埋め込みを早期に破棄することで余分な探索を枝刈りする。実験の結果提案手法は既存手法比で最大 4 桁高速であることが確認された。

キーワード グラフアルゴリズム, クエリ処理, サブグラフマッチング

1. はじめに

Web ページ間のハイパーリンク, 社会的な交友関係, 金融取引など, 様々な情報において事物間のつながりを表すグラフデータが中心的な役割を果たしている。グラフ中に存在する特定のサブグラフ構造を発見する問題はサブグラフマッチングと呼ばれ, グラフデータに対する最も典型的な問合せ処理の一つである。具体的には, サブグラフマッチングとは大きいグラフ(データグラフ)中に存在するクエリグラフと同型なサブグラフへの埋め込みを列挙する問題である。サブグラフマッチングはグラフデータベースにおいて代表的なグラフクエリ言語である SPARQL, Cypher, Gremlin などによって基本的なクエリとしてサポートしている [15]。さらにデータ分析の分野においても, 人間関係の分析 [5], 営業活動の支援 [9], マルウェアの検出 [11] など様々なアプリケーションにサブグラフマッチングは利用されている。特に頂点が表現する事物の種類(人物, 企業, 商品など)を頂点ラベルとして持つようなラベル付きグラフに対する適用が盛んである [5, 9, 11]。しかしながらサブグラフマッチングは NP 困難問題であるため, 計算に長い時間を要することが課題となっている [7]。

このような背景からサブグラフマッチングの高速化を目指して多くの研究が行われてきた [1, 4, 7, 8, 12, 13, 16]。これらの手法はいずれもバックトラッキングによる探索 [14] を行う。バックトラッキングは既に得られている部分的な解から新しい解を生成することで探索を行う再帰的な手法である。サブグラフマッチングの場合, クエリグラフの頂点(クエリ頂点)の一部のみがデータグラフの頂点(データ頂点)へ割当てられた部分

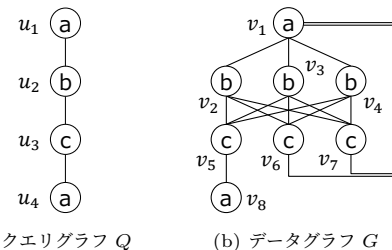


図 1 クエリグラフとデータグラフの例。 u_i, v_i は頂点の識別子を, 頂点内のアルファベットは頂点ラベルを表す。

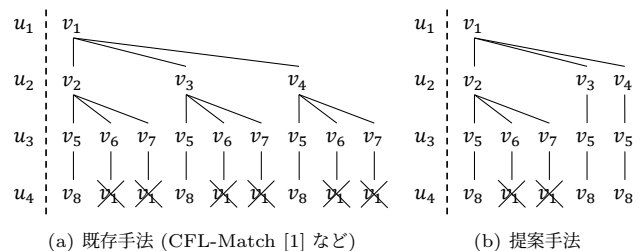


図 2 図 1 の Q と G に対して行われるバックトラッキングの過程を示すツリー。バツ印は探索失敗を表す。

埋め込みが部分的な解に相当する。未割当てのクエリ頂点からデータ頂点への割当てを部分埋め込みに追加することで新しい埋め込みが生成される。新しい埋め込みが全てのクエリ頂点の割当てを含む完全な埋め込みならば解として報告する。一方で新しい埋め込みから完全な埋め込みを得られないことが判明した場合は探索失敗となる。失敗した埋め込みを破棄し, 前のス

テップへ戻り別の部分埋め込みについて探索を再開する。バックトラッキングによる探索は深さ優先探索のように進行するため、探索の過程をツリーで表現することができる。例えば図1のクエリグラフ Q とデータグラフ G のサブグラフマッチングでは、図2(a)のような探索過程が得られる。最も左側の過程 $v_1-v_2-v_5-v_8$ は、 u_1 から u_4 を順にそれぞれのデータ頂点へ割当てて探索を表す。ここでは完全な埋め込みを得られる。次に u_3 の割当てまで戻り、割当て先を v_6 へ変更する。 v_6 の隣接頂点である v_1 は u_1 の割当て先として使用済みであるため、探索失敗となる。そこで u_3 の割当て先を再度変更し、以降も同様に探索を行う。サブグラフマッチングはこのような再帰的な手続きによって行われる。

バックトラッキングにおいて解に至らない部分埋め込みに対する処理は無駄になるため、サブグラフマッチングの性能を向上させるためには探索失敗の削減が必要である。そこで既存手法はバックトラッキングの前処理としてクエリグラフとデータグラフの構造を分析することで探索空間を狭めている。例えば頂点の度数や隣接頂点のラベルなど局所的な情報を比較することで、クエリ頂点 u_i の割当て先となり得るデータ頂点を候補頂点集合 $C[u_i]$ として事前に抽出する手法は一般的に用いられている [1, 7, 8, 16]。より多くの探索失敗を削減できるよう前処理を改良することでサブグラフマッチングは高速化されてきた。

しかしながら、前処理を用いてもバックトラッキング時の探索失敗は発生する。前処理はバックトラッキングに追加されるオーバーヘッドとなるため複雑な処理を用いることが難しく、探索空間の削減範囲に限られる。例えば既存手法の前処理では、実際にはクエリ頂点 u_i の割当て先となることができないデータ頂点が偽陽性として候補頂点集合 $C[u_i]$ に含まれてしまう場合がある。ナイーブにはサブグラフマッチングの探索空間は各クエリ頂点 $u_1, u_2, \dots, u_n$ に対する候補頂点の組合せ $C[u_1] \times C[u_2] \times \dots \times C[u_n]$ である。そのため組合せ爆発により僅かな偽陽性でも性能に大きく影響する。実際、図1の Q と G において、既存手法 [1, 7, 8, 16] は頂点ラベル以外の条件により候補頂点を絞り込むことができない。そのため図2(a)と同様の探索失敗が発生する。この探索では v_1 の重複使用による探索失敗を u_2 と u_3 の割当て先を変更しながら繰返し発生させている。既存手法ではこのように無駄な探索が発生してしまう。

以上のような問題点を踏まえ、本論文では前処理ではなくバックトラッキングの効率化に着目した新しいサブグラフマッチングアルゴリズムを提案する。提案手法の中心的なアイデアは「失敗から学ぶ」ことによる枝刈りである。具体的には、提案手法は探索失敗となった部分埋め込みから失敗の原因となる頂点の割当てを抽出し、失敗パターンとして記録する。そして後続のバックトラッキングにおいて処理中の部分埋め込みが既出の失敗パターンと符合したとき、その探索を打ち切る。提案手法の動作を図1の Q と G を例にとって示す。図2(b)は提案手法による探索の過程を表すツリーである。まずクエリ頂点をそれぞれ v_1, v_2, v_6, v_1 へ割当てた場合、前述の通り v_1 の重複使用により探索失敗となる。ここで v_6 に隣接するラベル a の頂点は v_1 のみであるから、 (u_1, v_1) という割当てと同時に (u_3, v_6) という割当てを行った時点で探索失敗となることは確定してい

る。そこで提案手法は $\{(u_1, v_1), (u_3, v_6)\}$ を失敗パターンとして記録する。同様に u_3 を v_7 へ割当てた場合も探索失敗となるため、失敗パターン $\{(u_1, v_1), (u_3, v_7)\}$ を記録する。次にバックトラッキングにより u_2 の割当て先が v_3 へ変更される。ここで u_3 を v_6 または v_7 に割当てた部分埋め込みは記録された失敗パターンに符合するため、探索が枝刈りされる。これにより探索失敗が減少する。 u_2 を v_4 へ割当てた場合も同様である。既存手法が候補頂点の全組合せを生成する(図2(a))のに対し、提案手法は探索失敗を繰り返さないため高速である。

提案手法の特長は次の3つである：

(1) **ロバスト**：既存手法がクエリグラフとデータグラフの構造を分析することで探索失敗を予防するのに対し、提案手法はバックトラッキング中の情報を利用して高速化する。そのため、図1のように既存手法が機能しないグラフにおいても探索失敗を削減することができる。つまり、処理効率が与えられたグラフに左右されにくいという意味でロバスト性が高い。

(2) **スケーラブル**：クエリグラフとデータグラフが大規模であるほど埋め込みの組合せ数は増加するため、処理時間の増大が急激である。提案手法はロバスト性が高いことから、グラフの規模に対して時間計算量的にスケーラブルである。

(3) **厳密**：提案手法は無駄な探索のみを枝刈りすることが証明されている。つまり全ての埋め込みを列挙する厳密手法である。

実験の結果、提案手法は既存手法と比べ最大4桁以上高速であることが確認された。既存手法では処理に1日以上を要するクエリセットに対しても提案手法は高速に応答可能であるため、これまで事実上不可能だったクエリにも対応することができる。

本論文の構成は次の通りである。第2節で関連研究を紹介し、本研究の前提知識を第3節で説明する。第4節で手法の詳細を述べ、第5節で実験結果を示す。最後に第6節で結論を述べる。

2. 関連研究

サブグラフマッチングの研究はいずれもサブグラフ同型問題を扱うが、その問題設定は大きく2つに分類される。

1つ目は単一の大きなデータグラフからクエリグラフと同型なサブグラフを列挙する問題である。本論文ではこれらの問題設定を扱う。この分野ではバックトラッキングに基づく古典的なアルゴリズムである Ullmann [14] がよく知られている。Ullmann 以降、さらに高速化するために候補頂点枝刈りとマッチング順序最適化という2つのテクニックを中心に多くの研究がなされてきた。1つ目の候補頂点枝刈りとは、クエリ頂点 u_i の候補頂点集合 $C[u_i]$ に含まれるデータ頂点を削減するためのフィルタリング手法である。Ullmann [14] では頂点のラベルと度数に基づくフィルタを行う。GraphQL [8] と SPPath [16] はさらにクエリグラフとデータグラフの局所的な隣接関係を用いた軽量なマッチングをフィルタに用いる。2つ目のテクニックであるマッチング順序最適化とは、バックトラッキングにおいて部分埋め込みへクエリ頂点を追加していく順番を変更することで高速化する手法である。バックトラッキングでは前のステップで生成された部分埋め込みから新しい部分埋め込みを生成するため、部分埋め込みの生成数が少ないクエリ頂点から順に割当て先を決定することで探索空間を狭められる。VF2 [4]

は割当て済みクエリ頂点が常に1つの連結成分になるようにマッチング順序を最適化し、QuickSI [13] はさらにデータグラフにおけるラベルの出現頻度を元に希少度の高い頂点とエッジから順に割当てを行う。近年は候補頂点枝刈りとマッチング順序最適化を同時に行う手法も提案されている。TurboISO [7] と CFL-Match [1] はクエリグラフの全域木に基づく近似的なマッチングにより候補頂点を枝刈りし、さらに候補頂点の接続関係を踏まえクエリグラフ中の推定埋め込み数が少ない部分から割当てを行う。以上の既存手法はいずれもバックトラッキングの探索空間を縮小するためにその前処理を工夫している。これに対し、提案手法はバックトラッキングの最中に枝刈りを行うことにより、前処理が十分に機能しないようなクエリグラフとデータグラフの組合せにおいても処理を高速化することができる。また既存手法と提案手法はほぼ直交しており、両者を組み合わせ使用することも可能である。

2つ目の問題設定は多数の(大抵は小規模な)データグラフに対する、クエリグラフと同型なサブグラフを含むかどうかの判定、あるいは同型なサブグラフの列挙である。この問題設定においては、まずデータグラフのインデックス構造を用いて確実にクエリグラフを含まないデータグラフを除外し(filter)、次に残ったデータグラフがクエリグラフを含むことを確認する(verify)という2段階の処理が一般的である。インデックスの情報としてはデータグラフに含まれるパス [2, 6] や頻出サブグラフ [3] の利用が提案されている。この問題設定においても verify 処理に提案手法を用いて高速化することは可能である。

3. 背景知識

本節では本論文で扱う問題、および以降の説明で用いる表記法を定義する。

3.1 問題定義

本論文では頂点がラベルを持つ無向グラフ $G = (V_G, E_G, \Sigma, \ell)$ を扱う。 V_G は頂点の集合、 $E_G \subseteq V_G \times V_G$ はエッジの集合、 Σ はラベルの集合、 ℓ は頂点とラベルを対応づける関数である。 G を特に**データグラフ**と呼ぶ。同様に**クエリグラフ** $Q = (V_Q, E_Q, \Sigma, \ell)$ を考える。クエリグラフの頂点は $u_1, u_2, \dots, u_n$ というように番号付けされているものとする。

定義 1 (サブグラフ同型). 次の3つの制約^(注1)を満たす埋め込み $M: V_Q \rightarrow V_G$ を定義できるとき、 Q は G に対して**サブグラフ同型**であるという:

- (1) **ラベル制約**: $\forall u_i \in V_Q, \ell(u_i) = \ell(M[u_i])$
- (2) **エッジ制約**: $\forall (u_i, u'_i) \in E_Q, (M[u_i], M[u'_i]) \in E_G$
- (3) **単射制約**: $\forall u_i, u'_i \in V_Q, u_i \neq u'_i \Rightarrow M[u_i] \neq M[u'_i]$

定義 2 (サブグラフマッチング). クエリグラフ Q とデータグラフ G が与えられたとき、 G に含まれる Q のサブグラフ同型な埋め込みを全て列挙する問題を**サブグラフマッチング**と呼ぶ。

本論文はサブグラフマッチングを高速に解く手法を与える。ただし実際には組合せ爆発によって膨大な数の埋め込みが生じ

表 1 本論文で用いる主なシンボル

シンボル	定義
Q, G	クエリグラフ, データグラフ
V_Q, V_G	クエリグラフとデータグラフの頂点集合
E_Q, E_G	クエリグラフとデータグラフのエッジ集合
u_i, v	クエリ頂点 ($u_i \in V_Q$) とデータ頂点 ($v \in V_G$)
$C[u_i]$	u_i の候補頂点集合
$M, \hat{M}$	完全な埋め込みと部分埋め込み
D, Γ	Dead-end パターンと dead-end マスク
$N(\cdot)$	頂点の隣接頂点集合
$\ell(\cdot)$	頂点のラベル
$\text{dom}(\cdot), \text{ran}(\cdot)$	埋め込みの定義域と値域
$\text{Dead}(\cdot)$	述語: 埋め込みは dead-end パターンであるか?

場合があるため、その全てを列挙することは現実的でない。そこで一定数、例えば 1,000 個の埋め込みを列挙した時点でクエリ処理を打ち切ることが一般的である [1, 7, 10]。

3.2 諸定義

本論文で使用する主なシンボルを表 1 に示す。さらにいくつかの定義を行う。

定義 3 (埋め込みの表現). データグラフ G に含まれるクエリグラフ Q の埋め込み $M: V_Q \rightarrow V_G$ を、クエリ頂点 u_i とデータ頂点 v のペアの集合で表す。 u_i の割当て先が v である (つまり $v = M[u_i]$) とき $(u_i, v) \in M$ である。

定義 4 (埋め込みの定義域と値域). 埋め込み M の定義域と値域をそれぞれ次のように書く: $\text{dom}(M) = \{u_i \mid (u_i, v) \in M\}$, $\text{ran}(M) = \{v \mid (u_i, v) \in M\}$ 。

例えば図 1 の Q と G において、 $M = \{(u_1, v_1), (u_2, v_2), (u_3, v_5), (u_4, v_8)\}$ は埋め込みである。また $\text{dom}(M) = \{u_1, u_2, u_3, u_4\}$, $\text{ran}(M) = \{v_1, v_2, v_5, v_8\}$ である。

定義 5 (完全な埋め込み). 埋め込み M について、 $\text{dom}(M) = V_Q$ であるときに限り、 M を**完全な埋め込み**と呼ぶ。

なお埋め込みが完全な埋め込みでない (かもしれない) ことを強調する文脈においては、埋め込みを $\hat{M}$ と表記し、 $\hat{M}$ を**部分埋め込み**と呼ぶ。完全な埋め込みも部分埋め込みの一種として扱うことに注意されたい。

3.3 バックトラッキングによるサブグラフマッチング

サブグラフマッチング手法は近年提案されたものも含めバックトラッキングによる探索 [10, 14] を行うものが主流である。ナイーブなバックトラッキングによるサブグラフマッチングアルゴリズムをアルゴリズム 1 に示す。再帰関数 SEARCH は部分埋め込み $\hat{M}$ と候補頂点集合 C を受け取る。 C の抽出方法としては、最も単純には次のように u_i とラベルが一致するデータ頂点を候補頂点とする方法がある [14]:

$$C[u_i] = \{v \in V_G \mid \ell(v) = \ell(u_i)\}. \quad (1)$$

関数呼び出し SEARCH($\emptyset, C$) によって探索が開始される。関数は最初に $\hat{M}$ が完全な埋め込みかどうかを確認し、そうであれば完全な埋め込みとして報告する (3, 4 行目)。 $\hat{M}$ が完全な埋め込みでない場合、次の再帰呼び出しで使用するため C から

(注1): 3つの制約への命名は以降の説明のために本論文で便宜上行っていることであり、一般的ではない。

アルゴリズム 1 バックトラッキングによる探索

入力: クエリグラフ Q , データグラフ G

出力: G に含まれる全ての Q の埋め込みを報告する.

```

1 function SEARCH( $\hat{M}, C$ )
2    $k \leftarrow |\hat{M}|$ 
3   if  $k = |V_Q|$  then
4      $\hat{M}$  を完全な埋め込みとして報告する
5   return
6    $C' \leftarrow C$  のうち  $\hat{M}$  についてエッジ制約を満たす候補頂点 (式 2)
7   for each  $v \in C'[u_{k+1}]$  do
8     if  $v \notin \text{ran}(\hat{M})$  then
9       SEARCH( $\hat{M} \cup \{(u_{k+1}, v)\}, C'$ )

```

エッジ制約を満たさない候補頂点を除外する (6 行目). エッジ制約は, クエリ頂点 u_i と u'_i が隣接するならばそれらの割当て先 $M[u_i]$ と $M[u'_i]$ も隣接していることを要求する. 従ってクエリ頂点 u_i の割当て先は, $\hat{M}$ に含まれる全ての隣接頂点 u'_i の割当て先と隣接する候補頂点でなければならない. つまり,

$$C'[u_i] = C[u_i] \cap \bigcap_{u'_i \in N(u_i) \cap \text{dom}(\hat{M})} N(\hat{M}[u'_i]). \quad (2)$$

$\hat{M}$ には $u_1, u_2, \dots, u_k$ の割当てが含まれるので, 次は u_{k+1} の候補頂点 v についてループする (7 行目). v が $\hat{M}$ で使用されていないならば (8 行目), $\hat{M}$ に u_{k+1} と v の割当てを追加した部分埋め込み $\hat{M} \cup \{(u_{k+1}, v)\}$ と C' を引数として関数 SEARCH を再帰呼び出しする. このようにラベル制約 (式 1), エッジ制約 (6 行目), 単射制約 (8 行目) を満たす割当てを再帰的に部分埋め込みへ追加していくことにより完全な埋め込みを探索する.

4. 提案手法

本節ではまず提案手法の中心的なアイデアとアルゴリズムの全体像を説明し, その後アルゴリズムの詳細を述べる.

4.1 提案手法のアイデア

候補頂点枝刈りやマッチング順序最適化のようなバックトラッキングの前処理によって探索範囲を狭めることができるが, 依然として多くの探索失敗が発生し処理時間を増大させる. 我々のアイデアはバックトラッキング中に発生した探索失敗から学び, 失敗を繰り返さないことによって探索を高速化することである. 具体的には, 探索失敗は部分埋め込みに含まれる頂点の割当てが定義 1 で示した 3 つの制約のいずれかに違反することによって発生する. 違反の原因となる割当てを部分埋め込みが含んでいる限り, それ以外のクエリ頂点の割当てをどのように変更しても探索失敗となる. そこで提案手法では制約に違反する割当ての組合せ (パターン) を探索失敗時に部分埋め込みから抽出し, それに符合する部分埋め込みを枝刈りすることによって以降の探索失敗を削減する. 既存手法による探索範囲の絞り込みと提案手法を組合せることによって, 既存手法では削減しきれなかった探索失敗を削減し高速化することが可能である.

4.2 Dead-end パターンによる枝刈り

本節ではナイーブなバックトラッキングによる探索 (アルゴリズム 1) を参照しつつ提案手法による探索の枝刈りを説明する. まず以降の説明で使用するため次の通り定義を行う.

定義 6 (Dead-end パターン). D を部分埋め込みとする. $D \subseteq M$

であるような完全な埋め込み M が存在しないとき, D を *dead-end パターン* と呼ぶ. 記述を簡潔にするため, 単に「 D は dead-end である」とも書く. また D が dead-end である場合に限り $\text{Dead}(D)$ が真になるような述語 Dead を導入する.

Dead-end (行き止まり) とは図 2 のような探索ツリーにおける探索失敗 (図中バツ印) を指す. Dead-end パターンを利用した枝刈りを行うため, 提案手法は関数 SEARCH に対して (i) 関数末尾における dead-end パターンの抽出と (ii) 再帰呼び出しの前 (8, 9 行目の間) における dead-end パターンとの照合という 2 つの変更を加える. 1 つ目の dead-end パターンの抽出では部分埋め込み $\hat{M}$ から一部の頂点割当てを dead-end パターン D として抽出し, dead-end パターンの集合 $\mathcal{D}$ へ追加する. 具体的な抽出方法は後述する. Dead-end パターンの抽出は関数呼び出し $\text{SEARCH}(\hat{M}, C)$ またはそこからの再帰呼び出し内において埋め込みの報告がなかった場合にのみ行われる. もし埋め込みの報告があったならば $\hat{M}$ に関する探索は成功したことになるため, $\hat{M}$ から dead-end パターンを抽出することはできない. 2 つ目の dead-end パターンとの照合では, 新しい部分埋め込み $\hat{M} \cup \{(u_{k+1}, v)\}$ が $D \in \mathcal{D}$ のいずれかに符合するかどうかを確認する.

定義 7 (Dead-end パターンとの符合). 部分埋め込み $\hat{M}$ と dead-end パターン D が与えられたとき, $D \subseteq \hat{M}$ であるならば「 $\hat{M}$ は D に符合する」と言う.

もし dead-end パターンに符合するならば再帰を行わないことで探索を打ち切る. このような dead-end パターンの抽出と照合がナイーブなバックトラッキングに対する差分である.

4.3 Dead-end パターンの抽出

前述の通り部分埋め込み $\hat{M}$ が dead-end パターン抽出の対象となると, $\hat{M}$ を含む完全な埋め込みは存在しない. 従って定義上 $\hat{M}$ 自体も dead-end パターンである. しかし dead-end パターンとしての $\hat{M}$ に符合する部分埋め込みは関数呼び出し $\text{SEARCH}(\hat{M}, C)$ において探索済みであるため, $\hat{M}$ に枝刈り効果はない. 以降の探索において多くの部分埋め込みを dead-end パターンと符合させるため, パターンは少数のクエリ頂点に対する割当てのみを含むことが望ましい. しかし $\hat{M}$ から一部の割当てのみを取り出した部分埋め込みが必ずしも dead-end であるとは限らない. そこで dead-end パターンの抽出では, $\hat{M}$ の部分集合であり, かつ依然として dead-end でもあるような部分埋め込みを作成する. その際次のような dead-end マスクという考え方をを用いる.

定義 8 (Dead-end マスク). $\hat{M}$ を dead-end である部分埋め込みとする. このとき次式を満たすクエリ頂点の集合 Γ を $\hat{M}$ の *dead-end マスク* と呼ぶ.

$$\Gamma \subseteq \text{dom}(\hat{M}) \wedge \text{Dead} \left(\left\{ (u_i, v) \in \hat{M} \mid u_i \in \Gamma \right\} \right) \quad (3)$$

$\hat{M}$ から抽出された Dead-end パターン D は, $\hat{M}$ と Γ を用いて $D = \left\{ (u_i, v) \in \hat{M} \mid u_i \in \Gamma \right\}$ によって得られる.

Dead-end マスクの選択方法は探索失敗の原因によって異なる. アルゴリズム 1 において探索失敗となる原因は次の 3 つが

ある: (i) ラベル制約 (式 1) とエッジ制約 (式 2) を満たす候補頂点 $v \in C'[u_{k+1}]$ が存在しない (7 行目), (ii) 新しい部分埋め込み $\hat{M} \cup \{(u_{k+1}, v)\}$ が単射制約を満たさない (8 行目), (iii) 再帰呼び出しした関数 SEARCH が探索に失敗する (9 行目). さらに提案手法では dead-end パターンとの照合を 8, 9 行目の間で行うため, 探索失敗となる原因は合わせて 4 つ存在する. 以降ではこれら 4 つの原因それぞれについて dead-end マスクの選択方法を説明し, 最後に失敗の原因別に得られたマスクを集約し $\hat{M}$ のマスクを得る方法を述べる.

4.3.1 候補頂点が存在しない場合

候補頂点が存在しない状態とは $C'[u_{k+1}] = \emptyset$ である. 実際には u_{k+1} に限らず, いずれかのクエリ頂点が候補頂点を失った時点で $\hat{M}$ の探索失敗が確定する. このとき, dead-end マスクは次のように得られる.

補題 1 (候補頂点が存在しない場合の dead-end マスク). クエリ頂点 u_i について $C'[u_i] = \emptyset$ であるとき, $\Gamma = N(u_i) \cap \text{dom}(\hat{M})$ は $\hat{M}$ の dead-end マスクである.

証明. Γ が dead-end マスクでない, 即ち式 3 を満たさないと仮定する. このとき $\Gamma \subseteq \text{dom}(\hat{M})$ は自明なので, $\{(u_j, v) \in \hat{M} \mid u_j \in \Gamma\} \subseteq M$ であるような埋め込み M が存在する. M はエッジ制約を満たす埋め込みなので, 式 2 の $\hat{M}$ に M を代入して $C'_M[u_i] = C[u_i] \cap \bigcap_{u'_i \in N(u_i)} N(M[u'_i])$ と置くと $M[u_i] \in C'_M[u_i]$. 一方 $\Gamma \subseteq N(u_i)$ 及び $\forall u_j \in \Gamma, M[u_j] = \hat{M}[u_j]$ より, $C'_M[u_i] \subseteq C[u_i] \cap \bigcap_{u'_i \in \Gamma} N(\hat{M}[u'_i]) = C'[u_i] = \emptyset$ である. 従って $M[u_i] \in C'_M[u_i]$ と矛盾する. ゆえに Γ は式 3 を満たす dead-end マスクである. $\square$

4.3.2 単射制約を満たさない場合

説明のため新しい部分埋め込みを $\hat{M}_+ = \hat{M} \cup \{(u_{k+1}, v)\}$ と置く. アルゴリズム 1 では $v \notin \text{ran}(\hat{M})$ (8 行目) によって $\hat{M}_+$ が単射制約を満たすことを確認する. $\hat{M}_+$ が単射制約を満たさない場合の dead-end マスクは次のように得られる.

補題 2 (単射制約を満たさない場合の dead-end マスク). $\hat{M}_+$ を単射制約を満たさない部分埋め込みとする. 単射でないので $\hat{M}_+[u_{i_1}] = \hat{M}_+[u_{i_2}]$ であるようなクエリ頂点 u_{i_1}, u_{i_2} が存在する. ここで $\Gamma_+ = \{u_{i_1}, u_{i_2}\}$ は $\hat{M}_+$ の dead-end マスクである.

証明. $\{(u_i, v) \in \hat{M}_+[u_i] \mid u_i \in \Gamma_+\} \subseteq M$ であるような埋め込み M は単射制約を満たさない. また $\Gamma_+ \subseteq \text{dom}(\hat{M}_+)$. ゆえに Γ_+ は $\hat{M}_+$ の dead-end マスクである. $\square$

補題 1 が $\hat{M}$ の dead-end マスクを与えるのに対し, 補題 2 は $\hat{M}_+$ のマスクを与えることに注意されたい. $\hat{M}_+$ のマスクを $\hat{M}$ のマスクへ変換する方法は後述する.

4.3.3 Dead-end パターンに符合する場合

新しい部分埋め込み $\hat{M}_+$ が記録済みの dead-end パターン D に符合する場合, dead-end マスクは D より得られる.

補題 3 (Dead-end パターンに符合する場合の dead-end マスク). D を dead-end パターンとする. $D \subseteq \hat{M}_+$ であるとき, $\Gamma_+ = \text{dom}(D)$ は $\hat{M}_+$ の dead-end マスクである.

証明. $D \subseteq \hat{M}_+$ より $\Gamma_+ \subseteq \text{dom}(\hat{M}_+)$. D は dead-end パターンなので $\text{Dead}(D)$. ゆえに Γ_+ は $\hat{M}_+$ の dead-end マスクである. $\square$

4.3.4 再帰呼び出しが探索失敗となる場合

アルゴリズム 1 は新しい部分埋め込み $\hat{M}_+$ について単射制約と dead-end パターンへの照合を行ったのち, 関数 SEARCH を再帰呼び出しする. 再帰呼び出しから戻るまでに埋め込みの報告がなかった場合, $\hat{M}_+$ は dead-end であったことが分かる. このとき, $\hat{M}_+$ からの dead-end パターンの抽出と記録は再帰呼び出しされた関数内で完了している. つまり, 再帰呼び出しから戻ったときには $\hat{M}_+$ と符合する dead-end パターンが記録されている. 従って補題 3 と同様にして $\hat{M}_+$ の dead-end マスクを得られる.

4.3.5 Dead-end マスクの集約

関数 SEARCH は部分埋め込み $\hat{M}$ から dead-end パターンを抽出するために $\hat{M}$ の dead-end マスクを必要としている. 補題 1 では直接 $\hat{M}$ のマスクを得られるのに対し, 補題 2 及び 3 で得られるのは新しい部分埋め込み $\hat{M}_+$ のマスクである. 式 3 より $\hat{M}$ のマスク Γ は $\Gamma \subseteq \text{dom}(\hat{M}) = \{u_1, u_2, \dots, u_k\}$ を満たすが, $\hat{M}$ に u_{k+1} の頂点割当てを追加することで作られた $\hat{M}_+$ のマスクは u_{k+1} を含む可能性がある. 従って $\hat{M}_+$ のマスクから $\hat{M}$ のマスクを得るためには変換が必要である.

補題 2 と 3 は $v \in C'[u_{k+1}]$ についてのループ (7 行目) の内部で適用されるため, ループ内で生成される新しい部分埋め込み $\hat{M}_+$ のそれぞれに対応するマスクが得られる. 従って次のような集合 Γ_* を得ることが可能である:

$$\Gamma_* = \bigcup_{v \in C'[u_{k+1}]} (\hat{M} \cup \{(u_i, v)\} \text{ の dead-end マスク}). \quad (4)$$

ここで Γ_* について次のことが言える.

補題 4 (Dead-end マスクの集約). $C'[u_{k+1}] \neq \emptyset$ であるとき, 次の Γ は $\hat{M}$ の dead-end マスクである:

$$\Gamma = \begin{cases} (\Gamma_* \cup N(u_{k+1})) \cap \text{dom}(\hat{M}) & \text{if } u_{k+1} \in \Gamma_* \\ \Gamma_* & \text{if } u_{k+1} \notin \Gamma_* \end{cases} \quad (5)$$

証明. $u_{k+1} \notin \Gamma_*$ の場合は自明なので, $u_{k+1} \in \Gamma_*$ の場合について考える. $\Gamma' = \Gamma \cap \text{dom}(\hat{M})$, $D' = \{(u_i, v) \in \hat{M} \mid u_i \in \Gamma'\}$ と置く. 7 行目でのループでは u_{k+1} に $v \in C'[u_{k+1}]$ をそれぞれ割当てた新しい埋め込みを生成し全て dead-end であったことから, $D' \times \{(u_{k+1}, v) \mid v \in C'[u_{k+1}]\}$ に含まれる埋め込みはいずれも dead-end である. ここで Γ が $\hat{M}$ の dead-end マスクでないとは仮定すると, $\{(u_i, v) \in \hat{M} \mid u_i \in \Gamma'\} \subseteq M$ であるような完全な埋め込み M が存在する. $N(u_{k+1}) \cap \text{dom}(\hat{M}) \subseteq \Gamma'$ であるから, 式 2 の $\hat{M}$ に M を代入して得られる $C'_M[u_{k+1}]$ と $C'[u_{k+1}]$ は等しい. ゆえに $M[u_{k+1}] \in C'[u_{k+1}]$. さらに $\Gamma' \subset \Gamma$ であるから, $M \in D' \times \{(u_{k+1}, v) \mid v \in C'[u_{k+1}]\}$. しかしこの集合に含まれる埋め込みは dead-end であるため矛盾する. ゆえに Γ は $\hat{M}$ の dead-end マスクである. $\square$

関数 SEARCH の末尾では以上のように抽出された dead-end マスク Γ を用いて dead-end パターンを $\hat{M}$ から抽出する.

4.4 Dead-end パターンの管理

簡単のため、4.2 節では dead-end パターンを集合 $\mathcal{D}$ に記録し、照合時には $\mathcal{D}$ に部分埋め込みと符合するパターンが存在するかどうかを確認すると述べた。しかしながら実際には空間的、時間的制約がそれぞれ存在し、実装は現実的でない。まず空間的には、dead-end パターンは最悪の場合可能な部分埋め込みと同数 (即ち $|C[u_1]| |C[u_2]| \dots |C[u_n]|$) 存在するため、メモリに収まらない可能性がある。さらに時間的には、性能向上のために部分埋め込み $\hat{M}$ と符合する dead-end パターンを高速に検索できなければならない。そこで本節では、実用的な枝刈りを可能にするための 2 つのテクニックを説明する。

4.4.1 連想配列への dead-end パターンの格納

探索中に抽出された全ての dead-end パターンを保存しておくことは、メモリ消費量とパターンの検索速度の両面において現実的でない。そこで dead-end パターンの一部をキーとして使用し、連想配列へ保存する方法を取る。キーが一致したパターンは上書き保存することによって、一定数以上のパターンが保存されないようにする。具体的には、探索に失敗した部分埋め込みへ最後に追加されたクエリ頂点とその割当て先データ頂点のペアをキーとする。つまり、 Δ を連想配列、 $\hat{M} = \{(u_1, v_{i_1}), (u_2, v_{i_2}), \dots, (u_k, v_{i_k})\}$ を dead-end である部分埋め込みだとすれば、 $\hat{M}$ から抽出されたパターンは $\Delta[u_k, v_{i_k}]$ に格納される。これにより、保存される dead-end パターンの数は候補頂点数の合計 $\sum_{u_i \in V_Q} |C[u_i]|$ 以下に保たれる。最後に割当てたクエリ頂点とデータ頂点をキーにするのは、その組み合わせが最も上書きされる頻度が少ないためである。枝刈りのために dead-end パターンを検索する際は、部分埋め込みへ最後に割当てたクエリ頂点とデータ頂点をキーとして連想配列を参照する。この処理は $O(1)$ で可能である。

4.4.2 Dead-end パターンの整数表現

Dead-end パターンを連想配列に保存することでメモリ消費と検索コストは実用レベルまで低下する。しかしながら、dead-end パターン $\mathcal{D}$ には最大で $|V_Q|$ 個の割当てが含まれることから、部分埋め込み $\hat{M}$ との符合を確認する ($\mathcal{D} \subseteq \hat{M}$ をテストする) ためには $O(|V_Q|)$ の時間計算量を要する。符合の確認は頻繁に行われるためこの計算コストの影響は大きく、枝刈り処理がかえって処理時間の増加を招く原因となる。

そこで提案手法では dead-end パターンを整数によって表現することで符合の確認を $O(1)$ 時間で行う。紙面の都合上ここでは基本的なアイデアのみ説明する。Dead-end パターンの整数表現のアイデアはバックトラッキングの再帰呼び出しの性質に基づく。具体的には、関数 SEARCH に引数として渡される部分埋め込み $\hat{M}$ が呼び出しごとに必ず異なることを利用する。これにより、 $\hat{M}$ を一意に識別する ID 番号 (埋め込み ID) として SEARCH の通算呼び出し回数を使用することができる。部分埋め込み $\hat{M}$ からその埋め込み ID を対応付ける関数を $\nu(\hat{M})$ と置くと、例えば図 2(b) の探索において、 $\nu(\{(u_1, v_1)\}) = 1$, $\nu(\{(u_1, v_1), (u_2, v_2)\}) = 2$, $\nu(\{(u_1, v_1), (u_2, v_2), (u_3, v_5)\}) = 3$, ... となる。

しかしながら、埋め込み ID を使った dead-end パターンの記録にはさらに 2 つの課題がある。1 つ目は、埋め込み ID で表現可能な部分埋め込みが必ず $u_1, u_2, \dots$ と連続したクエリ頂点に

対する割当てを含むことである。そのため、dead-end マスクによって一部のクエリ頂点のみを抽出した dead-end パターンを表現できない。この問題の影響を緩和するため、dead-end パターンを連想配列へ格納していることを利用する。Dead-end パターンの連想配列 Δ において、 $\hat{M}$ から抽出されたパターンの保存先は $\Delta[u_k, \hat{M}[u_k]]$ (ただし $k = |\hat{M}|$) に固定されている。その場所より自明であるため、dead-end パターンが $(u_k, \hat{M}[u_k])$ を含んでいることを明示する必要はない。そこで $\hat{M}$ の dead-end マスク Γ から u_k を除いて dead-end パターンを作成し、その埋め込み ID を連想配列へ格納する。2 つ目の課題は、部分埋め込みから埋め込み ID への変換である。埋め込み ID は関数 SEARCH と同じ数だけ存在するため、全てを記録しておくことはできない。そこで現在探索中の部分埋め込み $\hat{M}$ に含まれる埋め込みについてのみ ID を保持する。具体的には、 $\hat{M}$ と別に埋め込み ID の配列 Φ を定義し、 $\Phi[\mu] = \nu(\{(u_i, v) \in \hat{M} \mid i \leq \mu\})$ となるように保持する。これには関数呼び出し $\text{SEARCH}(\hat{M}, C)$ において $\Phi[u_k]$ へその時点の関数呼び出し回数を記録するだけでよい。

これら 2 つの課題とその対応策により、dead-end である部分埋め込み $\hat{M}$ とその dead-end マスク Γ が与えられたとき、dead-end パターンは (i) 埋め込み ID, (ii) 埋め込み ID が表す部分埋め込みの要素数, (iii) オリジナルの dead-end マスクの 3 つ組により表現される。具体的には次のように連想配列へ dead-end パターンを保存する:

$$\Delta[u_k, \hat{M}[u_k]] = (\Phi[\mu], \mu, \Gamma). \quad (6)$$

ただし $\mu = \max_{u_i \in \Gamma, i < k} i$ である。3 つ組に含まれる dead-end マスク Γ は補題 3 に基づき dead-end パターンを抽出するために必要である。部分埋め込み $\hat{M}$ ($k = |\hat{M}|$) とそれに対応する埋め込み ID の配列 Φ が与えられたとき、 $\hat{M}$ が dead-end パターンに符合するかどうかは次のように確認できる:

$$\Phi[\mu] = \phi, \text{ ただし } (\phi, \mu, \Gamma) = \Delta[u_k, \hat{M}[u_k]]. \quad (7)$$

Δ の参照と埋め込み ID の比較は共に $O(1)$ なので、照合処理全体としても $O(1)$ である。紙面の都合上次に示す詳細なアルゴリズムでは dead-end パターンの整数表現について具体的に示していないが、内部で以上のような dead-end パターンの表現を用いることで枝刈りのオーバーヘッドを削減している。

4.5 詳細なアルゴリズム

ここまで説明した dead-end パターンの抽出及び管理手法を統合した具体的な探索アルゴリズムをアルゴリズム 2 に示す。なおアルゴリズムは dead-end パターンの連想配列 Δ を大域変数として持つものとする。関数 SEARCH は部分埋め込み $\hat{M}$ と候補頂点集合 C を受け取り、 $\hat{M}$ が dead-end である場合は $\hat{M}$ の dead-end マスクを、そうでない場合は空集合を返却する。探索は関数呼び出し $\text{SEARCH}(\emptyset, C)$ によって開始される。アルゴリズム 1 と比較しアルゴリズム 2 の差分は、dead-end マスクの抽出 (8 行目など)、dead-end パターンとの照合 (14 行目)、記録 (20 行目) の 3 つを行う点である。1 つ目の dead-end マスクの抽出は探索失敗の確認とセットで行う。補題 1 の候補頂点が存在しない場合 (7, 8 行目)、補題 2 の単射制約を満たさない場

アルゴリズム 2 探索アルゴリズムの詳細

入力: クエリグラフ Q , データグラフ G .

出力: G に含まれる全ての Q の埋め込みを報告する.

```

1 function SEARCH( $\hat{M}, C$ )
2    $k \leftarrow |\hat{M}|$ 
3   if  $k = |V_Q|$  then
4      $\hat{M}$  を埋め込みとして報告する
5     return  $\emptyset$ 
6    $C' \leftarrow C$  のうち  $\hat{M}$  についてエッジ制約を満たす候補頂点 (式 2)
7   if  $C'[u_i] = \emptyset$  であるクエリ頂点  $u_i$  が存在する then
8      $\Gamma \leftarrow N(u_i) \cap \text{dom}(\hat{M})$                                 ▷ 補題 1
9   else
10     $\Gamma_* \leftarrow \emptyset$ 
11    for each  $v \in C'[u_{k+1}]$  do
12      if  $v \in \text{dom}(\hat{M})$  then
13         $\Gamma_* \leftarrow \Gamma_* \cup \{u_i \in \text{dom}(\hat{M}) \mid \hat{M}[u_i] = v\}$     ▷ 補題 2
14      else if  $\Delta[u_{k+1}, v] \subseteq \hat{M}$  then
15         $\Gamma_* \leftarrow \Gamma_* \cup \Delta[u_{k+1}, v]$                         ▷ 補題 3
16      else
17         $\Gamma_* \leftarrow \Gamma_* \cup \text{SEARCH}(\hat{M} \cup \{(u_{k+1}, v)\}, C)$ 
18     $\Gamma \leftarrow$  式 5 により変換された  $\Gamma_*$                                 ▷ 補題 4
19  if 再帰呼び出し中に埋め込みの報告がなく, かつ  $\hat{M} \neq \emptyset$  then
20     $\Delta[u_k, \hat{M}[u_k]] \leftarrow \{(u_i, v) \in \hat{M} \mid u_i \in \Gamma\}$ 
21    return  $\Gamma$ 
22  return  $\emptyset$ 

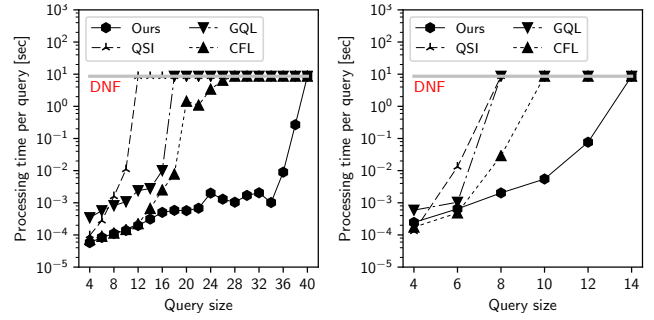
```

合 (12, 13 行目), 補題 3 の dead-end パターンに符合する場合 (14, 15 行目), そして 4.3.4 節で述べた再帰呼び出しが探索失敗となる場合 (17 行目) である. 候補頂点が存在しない場合を除き, 直接得られるのは新しい部分埋め込み $\hat{M} \cup \{(u_{k+1}, v)\}$ の dead-end マスクである. そこでそれらは Γ_* に蓄積し, 候補頂点についてのループを終えた後で式 5 により $\hat{M}$ の dead-end マスクに変換される (18 行目). 2 つ目の dead-end パターンとの照合は, 連想配列の 1 要素 $\Delta[u_{k+1}, v]$ に対してのみ行われるため高速である. なおもし dead-end パターンの記録がまだ行われておらず $\Delta[u_{k+1}, v]$ が未定義であった場合, $\Delta[u_{k+1}, v] \subseteq \hat{M}$ は偽になるものとして扱う. 3 つ目の dead-end パターンの記録は, 再帰呼び出し中に埋め込みの報告がない, 即ち $\hat{M}$ が dead-end であり, かつ $\hat{M}$ が要素を含むときにのみ行う (19 行目). 後者の条件は $\hat{M} = \emptyset$ のとき $\hat{M}[u_k]$ が未定義となってしまうため確認している. 最後に, $\hat{M}$ が dead-end であるときは dead-end マスクを (21 行目), そうでなければ空集合を返却する (22 行目). 以上のようにして実用的な時間計算量と空間計算量の下で dead-end パターンによる枝刈りを実装することが可能である.

また, このアルゴリズムは次の性質を満たす.

定理 1 (完全性). アルゴリズム 2 はデータグラフ G に含まれるクエリグラフ Q の全ての埋め込みを報告する.

証明. ナイーブなバックトラッキング (アルゴリズム 1) からの探索範囲の変化は 14 行目における dead-end パターンとの照合によって発生する. 補題 1, 2, 3, 4 より $\Delta[u_{k+1}, v]$ は (値が代入されているならば) dead-end であるから, $\Delta[u_{k+1}, v] \subseteq \hat{M}$ である部分埋め込み $\hat{M}$ も dead-end である. 定義 6 より dead-end である部分埋め込みは完全な埋め込みを生じないので, 探索を打ち切っても報告される埋め込みは変化しない. $\square$



(a) yeast データセット

(b) human データセット

図 3 各手法の 1 クエリあたりの平均処理時間の比較

5. 評価

本節では dead-end パターンによる枝刈りの効果を評価する. 提案手法の実装ではバックトラッキングによる探索に提案手法を用い, 候補頂点の枝刈りとマッチング順序最適化に最新手法である CFL-Match [1] を組合せる. 比較対象の手法としては CFL-Match に加え, 文献 [10] において高い性能が確認されている QuickSI [13] と GraphQL [8] の合計 3 つを用いる. QuickSI と GraphQL の実装は文献 [10] の著者から, その他は論文の著者から入手した. 実験に用いたマシンは Intel Xeon E5-2697 v2 を搭載し, 128GB のメモリを持つ. グラフは既存研究 [1, 7, 8, 12, 13, 16] で広く利用されている yeast と human を用いる. 共にタンパク質相互作用ネットワークであり, yeast は 3,112 頂点, 12,519 エッジ, 71 種類の頂点ラベルを, human は 4,674 頂点, 86,282 エッジ, 44 種類の頂点ラベルをそれぞれ含む. クエリグラフはランダムウォークによってデータグラフからサブグラフを抽出することで生成する. クエリは頂点数を変化させながら生成し, 頂点数ごとに 1 万クエリを 1 つのクエリセットとする. クエリセットの処理に 1 日以上を要する場合は処理不能 (Do-Not-Finish, DNF) とみなす. クエリによっては膨大な数の埋め込みが存在し全ての列挙が現実的でない場合があるため, 文献 [7] と同様 1,000 個の埋め込みを発見した時点で探索を打ち切る.

5.1 クエリ処理時間

まず dead-end パターンによる枝刈りによって得られる性能を確認するため, 既存手法とクエリの処理時間を比較する. 実験結果を図 3 に示す. 図中の Ours, QSI, GQL, CFL はそれぞれ提案手法, QuickSI, GraphQL, CFL-Match を意味する. yeast データセットにおいては 40 頂点, human データセットにおいては 14 頂点で全ての手法が DNF となった. 全体傾向を見ると, 提案手法は小規模なクエリから大規模なクエリまでバランスよく高速である. その中でも特に大規模なクエリにおいて性能差が顕著である. 特に yeast の 26–36 頂点のクエリでは既存手法比でおよそ 4 桁の高速化がみられる. 図 1 で例示したように, 既存手法は候補頂点枝刈りなどバックトラッキングの前処理が有効でないクエリグラフにおいて網羅的な割当ての試行に陥ることがある. クエリが大規模化すると候補頂点の組合せ数が増大するため, 処理時間が爆発的に増加する傾向にある. これに対し提案手法はクエリグラフの構造によらず dead-end

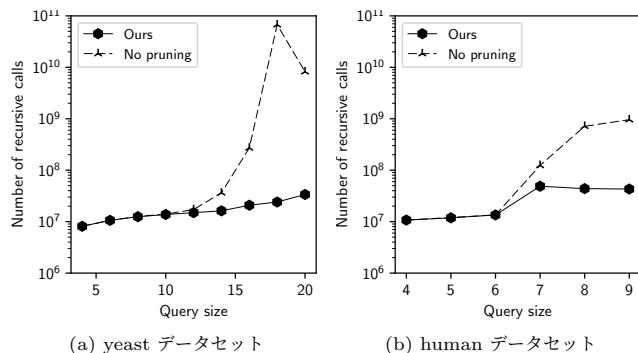


図 4 各クエリサイズにおいて 10,000 クエリの探索で発生した再帰呼び出し回数の合計

パターンに基づく枝刈りによって無駄な探索を削減する。そのため大規模なクエリにおいても処理時間の増加を抑えることができています。

5.2 Dead-end パターンによる枝刈りの回数

次に提案手法の性能が何によってもたらされているかを理解するため、SEARCH 関数 (アルゴリズム 2) の再帰呼び出し回数に着目して評価を行う。本実験では dead-end パターンによる枝刈りを行う通常の提案手法 (“Ours”) と、アルゴリズム 2 の 14, 15 行目をコメントアウトすることで枝刈りを省略した手法 (“No pruning”) について、再帰呼び出しが行われた回数を比較する。これらの再帰回数の差が枝刈りの効果である。結果を図 4 に示す。yeast では 20 頂点, human では 9 頂点より大きいクエリに対して No pruning が DNF となったため、それ以降の頂点数はプロットしていない。再帰回数を見ると、yeast, human と小規模なクエリでは枝刈りがほぼ発生していないものの、クエリが大規模化するに従って枝刈りの効果が増大している。yeast の 18 頂点のクエリにおける再帰回数は Ours が約 2.4×10^7 回, No pruning が約 6.7×10^{10} 回であり、3 桁以上の再帰を削減している。大規模なクエリほど削減率が大きいのは探索失敗の発生頻度が高いためと考えられる。クエリの頂点数が多いほど同一データ頂点への複数のクエリ頂点の割当て (単射制約違反) が発生しやすい。またエッジ数が増加するためエッジ制約違反も発生しやすい。それらによる探索失敗を削減することで提案手法は大幅に性能を向上させている。対照的に小規模なクエリでは探索失敗の削減率が小さい。それでも図 3 の実験結果において既存手法と比肩する性能を示すことができているのは、提案手法が 4.4 節で述べたように dead-end パターンの効率的な管理と照合を行い、枝刈りに要するオーバーヘッドを極めて小さく抑えているためである。

6. 結 論

サブグラフマッチングは様々な場面で利用されるが、NP 困難に属し高速化が課題となっている。そこで本論文では、失敗から学び、失敗を繰り返さないことにより高速化するサブグラフマッチングアルゴリズムを提案した。具体的にはバックトラッキングによる探索中に発生した探索失敗の原因に基づき dead-end パターンを生成し、パターンに符合する部分埋め込みの探索を打ち切る。実験の結果提案手法は既存手法比で最大

4 桁高速であることが確認された。

文 献

- [1] F. Bi, L. Chang, X. Lin, L. Qin, and W. Zhang. Efficient Subgraph Matching by Postponing Cartesian Products. *Proceedings of the 2016 ACM SIGMOD Conference on Management of Data*, 1:1199–1214, 2016.
- [2] V. Bonnici, A. Ferro, R. Giugno, A. Pulvirenti, and D. Shasha. Enhancing Graph Database Indexing by Suffix Tree Structure. In *Proceedings of the 5th IAPR International Conference on Pattern Recognition in Bioinformatics*, PRIB’10, pages 195–203, Berlin, Heidelberg, 2010. Springer-Verlag.
- [3] J. Cheng, Y. Ke, W. Ng, and A. Lu. Fg-index: Towards Verification-free Query Processing on Graph Databases. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, SIGMOD ’07, pages 857–872, New York, NY, USA, 2007. ACM.
- [4] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub)graph isomorphism algorithm for matching large graphs, 2004.
- [5] Y. Fang, W. Lin, V. W. Zheng, M. Wu, K. C. C. Chang, and X. L. Li. Semantic proximity search on graphs with metagraph-based learning, 2016.
- [6] R. Giugno, V. Bonnici, N. Bombieri, A. Pulvirenti, A. Ferro, and D. Shasha. GRAPES: A Software for Parallel Searching on Biological Graphs Targeting Multi-Core Architectures. *PLoS One*, 8(10), oct 2013.
- [7] W.-S. Han, S. Lee, K. Park, J.-H. Lee, M.-S. Kim, J. Kim, and H. Yu. TurboGraph: A Fast Parallel Graph Engine Handling Billion-scale Graphs in a Single PC. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’13, pages 77–85, New York, NY, USA, 2013. ACM.
- [8] H. He and A. K. Singh. Graphs-at-a-time: Query Language and Access Methods for Graph Databases. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD ’08, pages 405–418, New York, NY, USA, 2008. ACM.
- [9] Q. Hu, S. Xie, J. Zhang, Q. Zhu, S. Guo, and P. S. Yu. HeteroSales: Utilizing Heterogeneous Social Networks to Identify the Next Enterprise Customer. In *Proceedings of the 25th International Conference on World Wide Web*, WWW ’16, pages 41–50, Republic and Canton of Geneva, Switzerland, 2016. International World Wide Web Conferences Steering Committee.
- [10] J. Lee, W.-S. Han, R. Kasperovics, and J.-H. Lee. An In-depth Comparison of Subgraph Isomorphism Algorithms in Graph Databases. *Proceedings of the 39th international conference on Very Large Data Bases*, pages 133–144, 2012.
- [11] Y. Park, D. Reeves, V. Mulukutla, and B. Sundaravel. Fast Malware Classification by Automated Behavioral Graph Matching. In *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*, CSIIRW ’10, pages 45:1—45:4, New York, NY, USA, 2010. ACM.
- [12] X. Ren and J. Wang. Exploiting Vertex Relationships in Speeding Up Subgraph Isomorphism over Large Graphs. *Proc. VLDB Endow.*, 8(5):617–628, 2015.
- [13] H. Shang, Y. Zhang, X. Lin, and J. X. Yu. Taming Verification Hardness: An Efficient Algorithm for Testing Subgraph Isomorphism. *Proc. VLDB Endow.*, 1(1):364–375, 2008.
- [14] J. R. Ullmann. An Algorithm for Subgraph Isomorphism. *J. ACM*, 23(1):31–42, 1976.
- [15] H. Voigt. *Declarative Multidimensional Graph Queries*, pages 1–37. Springer International Publishing, Cham, 2017.

- [16] P. Zhao and J. Han. On Graph Query Optimization in Large Networks. *Proc. VLDB Endow.*, 3(1-2):340–351, 2010.