DEIM Forum 2022 G34-5

GuP: ガードパターンによる高速なサブグラフマッチング

新井　淳也† 　鬼塚　真†† 　藤原　靖宏†,††

† 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
E-mail: †{junya.arai.at,yasuhiro.fujiwara.kh}@hco.ntt.co.jp, ††onizuka@ist.osaka-u.ac.jp

あらまし　クエリグラフと同型なサブグラフを別のグラフの中で探す問題をサブグラフマッチングと呼ぶ．サブグラフの選び方は無数に存在するため，既存手法は同型であり得る部分の頂点を候補として事前に抽出することで探索空間を削減する．しかし候補から構成されながらも同型でないサブグラフは一般に多数存在するため，不要な探索が生じてしまう．本論文では，ガードパターンを用いる効率的なサブグラフマッチングアルゴリズム GuP を提案する．ガードパターンは各候補に付与され，その候補を含む同型なサブグラフが存在しないような探索状態を表現する．GuP はガードパターンにマッチする候補を探索時に除外することで不要な探索を削減する．実験により，最新の既存手法と比べ GuP は処理時間 1 秒以上のクエリの数を約 90%削減することが確認された．

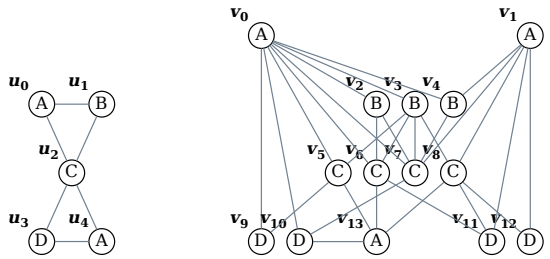
キーワード　サブグラフマッチング，サブグラフ同型，グラフデータベース，バックトラッキング，アルゴリズム

1 序　　論

文書の中で特定のフレーズを検索するのと同じように，グラフにおいては大きいグラフ（データグラフ）の中でクエリグラフと同型な部分を検索することが様々な場面で求められる．クエリグラフの頂点からデータグラフの頂点への同型なマッピング（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．例えば図 1 に示すクエリグラフ Q とデータグラフ G の間には埋め込み $\{(u_0, v_1), (u_1, v_4), (u_2, v_7), (u_3, v_{10}), (u_4, v_0)\}$ が存在する．ここで (u_i, v_j) はクエリグラフの頂点（クエリ頂点）u_i からデータグラフの頂点（データ頂点）v_j へのマッピングである．サブグラフマッチングは NP 困難であり [1]，複雑なグラフにおいては計算コストが高い．そのため効率的なアルゴリズムが長く研究されてきた [1–8]．

効率的なサブグラフマッチングのためには探索空間の削減が重要である．そのために広く用いられているアプローチとして候補フィルタリング [4] がある．同型な埋め込みを探す際はクエリ頂点 u_i $(i = 0, 1, \ldots)$ のマップ先を順番に決定していく．候補フィルタリングはマップ先の候補となるデータ頂点の集合 $C(u_i)$ を事前に抽出するために用いられる．最も基本的なフィルタはラベルの一致を確認することである [2]．さらに候補を絞り込むために局所的な擬似サブグラフマッチング [6, 9] やクエリグラフの全域木とのマッチング [1, 4] が提案されてきた．

しかし候補フィルタリングは依然として多くの不要な探索空間を残す．同型であるためには頂点が候補であること以外にも，マップ先にクエリグラフと同様に辺が存在すること，および複数のクエリ頂点が同一のデータ頂点にマップされていない（単射である）ことが求められる．そのため候補の組み合わせ方の数と比べ同型な埋め込みの数はずっと小さい．このことは同型な埋め込みで使用されるデータ頂点のみを候補に選んだ場合でさえ問題になる．例として Q のうち三角形 $u_2u_3u_4$ に着目する．この三角形の埋め込みは 3 つ存在する：$\{(u_2, v_5), (u_3, v_9), (u_4, v_0)\}$, $\{(u_2, v_7), (u_3, v_{10}), (u_4, v_0)\}$, $\{(u_2, v_8), (u_3, v_{11}), (u_4, v_1)\}$．従って $\{v_5, v_7, v_8\} \subseteq C(u_2), \{v_9, v_{10}, v_{11}\} \subseteq C(u_3), \{v_0, v_1\} \subseteq C(u_4)$ より，候補の組み合わせの数 $|C(u_2)||C(u_3)||C(u_4)|$ は $3 \times 3 \times 2 = 18$ 以上となる．しかし同型な埋め込みは 3 つなので，候補フィルタリングのみでは多くの不要な探索を生じることが分かる．



(a) クエリグラフ Q　　**(b)** データグラフ G

図 1: クエリグラフとデータグラフの例

効率的なサブグラフマッチングを可能にするため，本論文では**ガードパターン**によって探索空間を削減するサブグラフマッチングアルゴリズム *GuP* (Guard Pattern) を提案する．候補フィルタリングはクエリ頂点 u_i をデータ頂点 v にマップ可能かを一対一で判定するが，同型な埋め込みを得るためには埋め込みに含まれる他の頂点のマッピングとの組み合わせを考慮する必要がある．そこで GuP は各候補に対し，その候補を採用できないマッピングの条件をガードパターンとして付与する．そしてクエリ頂点のマップ先を決定していく中で，ガードパターンがマッチする（**ガード**される）候補は除外する．これにより候補の数が減少し効率的に探索できる．ガードパターンは**予約集合**と *deadend prefix* (D-prefix) という 2 種類の条件から成る．予約集合はデータグラフにおける候補間の接続関係を基に生成され，単射でない埋め込みを予め探索空間から削減する．

D-prefix は探索の最中に deadend パターン [10,11] (D-pat) に基づき逐次生成され，一部のマッピングを置き換えながら繰り返し同型でない埋め込みを生成してしまうことを防ぐ．最新の既存手法 [5,7,8] と比べ GuP はより多くのクエリを短時間で処理できることが実験で確認された．

以降の本論文の構成は次の通りである．第 2 節で関連研究を，第 3 節で前提知識を紹介する．第 4 節で手法の詳細を述べ，その評価結果を第 5 節で示す．最後に第 6 節で結論を述べる．なお紙面の制限から定理や補題の証明は省略する．

2 関連研究

頂点ラベル付きグラフに対するサブグラフマッチングのアルゴリズムは Ullmann [2] が最初に提案したとされる．Ullmann 以来サブグラフマッチングではバックトラッキングに基づく手法が主流である．

バックトラッキングによる探索を高速化するためのアプローチとして主なものが 2 つある．1 つ目は候補フィルタリングである．ラベルと次数 [2] や近傍の頂点にのみ着目した擬似的なマッチング [9,12] に基づくフィルタが提案されてきた．2010 年代になると，TurboISO [1], CFL [4], DAF [5] などクエリグラフの全域木とデータグラフの間で幅優先探索 (BFS) によるマッチングにより候補を抽出する手法が提案された．2 つ目のアプローチはマッチング順序の最適化である．クエリグラフで隣接する 2 頂点はデータグラフ上のマップ先でも隣接しなければならないため，片方のマップ先が決定していれば他方のマップ先はその隣接頂点に限られる．このことを利用し，クエリグラフ中のマッチ箇所が少ない部分からその隣接頂点へ向かうようなマッチング順序を生成する手法が研究されてきた [1,4–6,9,13]．しかしマッチング順序の最適化はヒューリスティクスなので，あるクエリグラフとデータグラフに対して良い順序を生成する手法が別の入力では悪い順序を生成することがある [7,14]．これに対し，候補フィルタリングや GuP のガードパターンのように候補を減らすアプローチは確実に探索空間を削減できる．

さらに新しいアプローチとして，探索中に発見された同型でない部分埋め込みから枝刈り条件を生成する手法が提案されている．新井ら [10] は，同型になり得ないマッピングの組み合わせである D-pat を探索中に発見し，それを含む部分埋め込みを枝刈りする手法を提案した．DAF [5] の failing set も似たアイデアに基づく手法である．ただし，D-pat がメモリ上に保存され繰り返し枝刈りに使用されるのに対し，failing set は 1 回の枝刈りを行うための条件を生成する．GuP の D-prefix は D-pat の数値表現 [10] を基に，D-pat のマッチ範囲を広げる辺制約集合（定義 11）や，候補集合全体をまとめて枝刈りする大域 D-prefix（定義 9）の考え方を新しく導入しガードパターンとして再構成したものである．

3 前提知識

本節は本論文に関する基礎的な事項を説明する．表 1 に本論文で使用する表記をまとめて示す．

表 1: 論文中で使用される表記

シンボル	定義
u_i, v	クエリ頂点とデータ頂点
$\ell(v), d(v)$	頂点のラベルと次数
$N(v)$	隣接頂点集合
$N_+(u_i)$	後方隣接頂点集合 $\{u_j \mid j > i\}$
$C(u_i)$	u_i の候補の集合
$M(u_i)$	埋め込み M における u_i のマップ先
$M\|_V$	集合による M の制限 $\{(u_i, v) \in M \mid u_i \in V\}$
$M\|_i$	頂点 ID による M の制限 $\{(u_j, v) \in M \mid j < i\}$
$X\|_i$	頂点 ID による $X \subseteq V_Q$ の部分集合 $\{u_j \in X \mid j < i\}$
$\mathrm{dom}(M)$	埋め込みの定義域 $\{u_i \mid \exists v, (u_i, v) \in M\}$
$\mathrm{ran}(M)$	埋め込みの値域 $\{v \mid \exists u_i, (u_i, v) \in M\}$
$A[u_i, v].R$	(u_i, v) の予約集合
$A[u_i, v].p$	(u_i, v) の（局所）D-prefix
p_g	大域 D-prefix

3.1 問題定義

本論文では頂点ラベル付きの単純な無向グラフであるクエリグラフ $Q = (V_Q, E_Q, \Sigma, \ell)$ とデータグラフ $G = (V_G, E_G, \Sigma, \ell)$ を扱う．V_Q, V_G は頂点の集合，E_Q, E_G は辺の集合，Σ はラベルの集合，ℓ は頂点をラベルに対応付ける関数である．クエリ頂点には通し番号の ID が振られている（つまり $V_Q = \{u_0, u_1, \ldots, \}$）．クエリ頂点は u，データ頂点は v で表記する．文脈から明らかな場合は u と v がそれぞれクエリ頂点とデータ頂点であることを明記しない．また，X から Y への対応関係の集合 $M \subseteq X \times Y$ と写像 $M : X \to Y$ を区別しない．

定義 1 (サブグラフ同型)．次の 3 つの制約を満たす埋め込み $M : V_Q \to V_G$ が存在するとき，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) 辺制約：$\forall (u_i, u_j) \in E_Q,\ (M(u_i), M(u_j)) \in E_G$,
(3) 単射制約：$\forall u_i, u_j \in V_Q,\ i \neq j \implies M(u_i) \neq M(u_j)$.

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．

マッピングの組み合わせ爆発によって埋め込みの数は指数的に増大することがあるため，実際には一定数の埋め込みを列挙した時点で探索を打ち切ることが一般的である [4,5,8]．

本論文では常にクエリ頂点の ID 順にマッチングを行う．他のマッチング順序を使用する場合は事前に頂点 ID を振り直す．ただし u_0 を除く任意のクエリ頂点は自身より若い ID のクエリ頂点と隣接することを前提とする．頂点 ID 順のマッチングでは，部分埋め込み M について $k = |M|$ と置くと，M は常に $u_0, u_1, \ldots, u_{k-1}$ のマッピングを含む．k を M **の長さ**と呼ぶ．

3.2 バックトラッキングによる探索

既存手法の多くはアルゴリズム 1 に示すようなバックトラッキングに基づく再帰的な探索 [7] を行う．再帰手続き Backtrack はクエリ頂点の一部をデータ頂点へ同型にマップする部分埋め込み M を受け取り，M にマッピングを一つ追加して

アルゴリズム 1 単純な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 Q の全ての同型な埋め込み

```
1: procedure Backtrack(M)
2:   if |M| = |V_Q| then M を解として出力し return
3:   k ← |M|
4:   for each v ∈ C(u_k; M) do
5:     if v ∉ ran(M) then
6:       Backtrack(M ∪ {(u_k, v)})
```

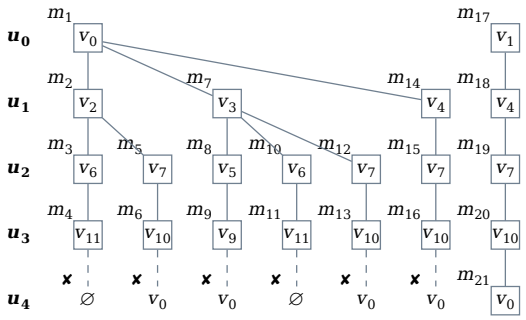


図 2: 図 1 の Q と G に対するバックトラッキングの探索木．$\emptyset$ に X 印は局所候補がない状態を，データ頂点に X 印は単射制約によりマップできない状態を，それぞれ表す．

再帰するという手順を繰り返す．なお M にマッピングを追加した埋め込みを M の**拡張**と呼ぶ [5]．追加できるマッピングがなければ別のマッピングを追加するため呼び出し元へ戻る．アルゴリズム中の $C(u_i; M)$ は，M に含まれるマッピングの下で辺制約を満たす u_i の候補（**局所候補**）の集合である．M の長さを k と置くと，$C(u_i; M)$ は次のように与えられる：

$$C(u_i; M) = \{v \in C(u_i) \mid \forall u_j \in N(u_i)|_k, v \in N(M(u_j))\}. \tag{1}$$

ただし $N(u_i)|_k = \{u_j \in N(u_i) \mid j < k\}$．クエリ頂点の候補 $C(u_i)$ は最も基本的なフィルタである label-and-degree filtering (LDF) [2,7] と neighbor label frequency filtering [7] で求められているものとする．ラベル制約はフィルタで確認されているので，バックトラッキングでは考慮しない．

探索の過程はマッピングの追加操作，すなわち再帰呼び出しをノードとする探索木上の深さ優先探索とみなすことができる．図 1 の Q と G に対応する探索木を図 2 に示す．例えばノード m_3 は (u_2, v_6) の追加を表す．同時に，m_3 は m_3 までのパスで追加されたマッピングから成る部分埋め込み $\{(u_0, v_0), (u_1, v_2), (u_2, v_6)\}$ に対応付けられる．バックトラッキングは最終的に m_{21} で示される埋め込みを同型な埋め込みとして出力する．

図 2 が示すように，ノード m_1 から始まる再帰では同型な埋め込みを発見できない．このような不要な再帰の削減がサブグラフマッチングの高速化における鍵となる．

3.3 Deadend パターン

バックトラッキングによる探索を効率化するため，新井らは D-pat に基づく枝刈りを提案した [10,11]．

定義 3 (Deadend パターン)．$D \subseteq V_Q \times V_G$ をマッピングの集合とする．任意の同型な埋め込み M について $D \not\subseteq M$ であるとき，D を deadend パターン (D-pat) と呼ぶ．

定義より D-pat を包含する部分埋め込みからは同型な埋め込みを得られないため，探索を打ち切ってよい．

ある部分埋め込み M またはその拡張が同型性の制約（定義 1）に違反するとき，M から一部のマッピングを抽出することで D-pat を作成できる．抽出するマッピングは Deadend マスク (D-mask) で選択する．

定義 4 (Deadend マスク)．M を部分埋め込み，K をクエリ頂点の集合とする．$M|_K$ が D-pat であるとき，K を M の deadend マスク (D-mask) と呼ぶ．[*1]

ただし $M|_K = \{(u_i, v) \in M \mid u_i \in K\}$ である．違反する制約ごとの D-mask の選択ルールを以下に示す．

補題 1 (制約違反に対する D-mask)．ある部分埋め込み M について，以下のように与えられる K は M の D-mask である．

ラベル制約違反 $\ell(u_i) \neq \ell(v)$ であるような $(u_i, v) \in M$ が存在するならば，$K = \{u_i\}$.

辺制約違反 $(u_i, u_j) \in E_Q$ かつ $(v, v') \notin V_G$ であるような $(u_i, v), (u_j, v') \in M$ が存在するならば，$K = \{u_i, u_j\}$.

単射制約違反 $(u_i, v) \in M$ かつ $(u_j, v) \in M$ であるような u_i, u_j $(i \neq j)$ が存在するならば，$K = \{u_i, u_j\}$.

実際は計算効率のために同型性の制約を定義とは異なる同値な条件で確認することがあり，その場合 D-mask の選択ルールも異なる．第 4.4 節では GuP における選択ルールを説明する．

4 手　　法

GuP は候補のそれぞれにガードパターンを付与する．ただし一つのデータ頂点が複数のクエリ頂点の候補になり得るため，以降では候補をクエリ頂点とのペア (u_i, v) で表す．バックトラッキング中に (u_i, v) のガードパターンが部分埋め込み M にマッチしたとき，$M \cup \{(u_i, v)\}$ についての再帰は枝刈り（ガード）される．ガードパターンは予約集合と D-prefix という 2 種類の条件から成る．予約集合が考慮できるのは単射制約のみだが，バックトラッキングの開始前に生成されるため探索の最初からガードに使用できる．一方，D-prefix は探索中に生成されるため初期状態では機能しないが，辺制約と単射制約の両方に基づくガードが可能である．D-prefix は D-mask から生成されるため，GuP は埋め込みの探索と同時に D-mask の計算を行う．本節では GuP の全体像を示したあと，2 種類のガードパターンの定義と生成方法を説明する．

4.1 全　体　像

アルゴリズム 2 に GuP の全体像を示す．GuP はバックト

*1：文献 [10] では $K \subseteq \mathrm{dom}(M)$ であることも K が D-mask であるための条件としているが，$\mathrm{dom}(M)$ に含まれないクエリ頂点は $M|_K$ の値に影響しないので実質的に本論文の定義でも違いはない．

アルゴリズム 2 GuP

入力: クエリグラフ Q，データグラフ G
出力: G に含まれる Q の全ての同型な埋め込み

```
1: u_i ∈ V_Q それぞれの候補集合 C(u_i) を抽出
2: 最適化したマッチング順序に Q を書き換え
3: 予約集合 A[u_i, v].R (u_i ∈ V_Q, v ∈ C(u_i)) を計算
4: D-prefix p_g と A[u_i, v].p (u_i ∈ V_Q, v ∈ C(u_i)) を初期化
5: anc ← (0, ..., 0); embedID ← 0; F[u_i] ← ∅ (u_i ∈ V_Q)
6: Backtrack(∅, C, F)                                  ▷ アルゴリズム 3
```

アルゴリズム 3 関数 $\textsc{Backtrack}(M, C_M, F_M)$

```
1: if |M| = |V_Q| then M を出力し return ∅
2: k ← |M|; found ← false
3: anc[k] ← embedID; embedID ← embedID + 1
4: for each v ∈ C_M[u_k] do
5:    K ← ∅; C'_M ← C_M; F'_M ← F_M
6:    if v ∈ ran(M) then
7:       K ← {u_i, u_k}, ただし M[u_i] = v
8:    if K = ∅ then
9:       if p_g, A[u_k, v].p または A[u_k, v].R に M がマッチ then
10:         K ← マッチしたパターンに対応する D-mask（第 4.4 節）
11:   if K = ∅ then
12:      for u_i ∈ N_+(u_k) do
13:         C'_M[u_i] ← C_M[u_i] ∩ N(v)
14:         if |C'_M[u_i]| < |C_M[u_i]| then F'_M[u_i] ← F_M[u_i] ∪ {u_k}
15:         if C'_M[u_i] = ∅ then K ← F'_M[u_i]; break
16:   if K = ∅ then K ← Backtrack(M ∪ {(u_k, v)}, C'_M, F'_M)
17:   if K = ∅ then found ← true; continue
18:   K に基づいて p_g と A[u_k, v].p を更新（補題 4, 3）
19: anc[k] ← 0
20: if found then return ∅
21: return agg(⋃_{v∈C_M[u_k]} A[u_k, v].p.K; k, F_M[u_k])
```

ラッキングの開始前にいくつかの処理を行う．多くの既存手法と同様に，初めに候補のフィルタリングとマッチング順序の最適化を行う．GuP はフィルタや順序に依存しないので任意の既存手法と組み合わせられる．評価（第 5 節）で用いた実装は DAF [5] のフィルタと VC [6] のマッチング順序を基にしている．次にガードパターンを初期化する．予約集合 $A[u_i, v].R$ はバックトラッキングの開始前に計算され，それ以降変化しない．一方 D-prefix は変化する．D-prefix はそれぞれの (u_i, v) に対応する局所 D-prefix $A[u_i, v].p$ に加え，候補によらず適用される大域 D-prefix p_g の 2 種類がある．初期値としてこれらの両方にガード効果のない適当な値を代入する．その後，D-prefix によるガードで使用される大域変数 $anc, embedID, F$ を初期化し，最後にバックトラッキングを開始する．

バックトラッキングの処理をアルゴリズム 3 に示す．アルゴリズム 1 との差分として注目すべき点を 4 つ挙げる．第 1 に，関数 Backtrack が戻り値を持つ．M から同型な埋め込みを得られた場合は空集合を，そうでなければ M の D-mask を返す．第 2 に，候補のループにはインクリメンタルに計算された局所候補集合 $C_M[u_k]$ が用いられる（4 行目）．M にマッピングを追加する度に未マッチのクエリ頂点の候補を再計算することにより，辺制約で候補がなくなる部分埋め込みを早期に発見する．第 3 に，M とガードパターンのマッチ判定が行われる（9 行目）．第 4 に，M またはその拡張から同型な埋め込みが見つからなかった場合，D-prefix が更新される（18 行目）．このようなアルゴリズムについて，以降で詳しく説明する．

4.2 予約集合

部分埋め込みにマッピングを追加するにつれ，辺制約によってその後マップ先となり得る候補が絞り込まれていく．絞り込まれた候補がいずれも既に別のクエリ頂点のマップ先となっていたとすれば，単射制約によりその部分埋め込みの拡張は同型にならない．ところが，従来のバックトラッキングでは実際にマップ先を決めるときまで単射制約の確認を行わない．そのため不要な再帰が発生する．例えば図 2 に示されているように，マッピング (u_2, v_5) を追加（ノード m_8）すると u_4 の局所候補が v_0 のみとなる．そのため，v_0 を他のクエリ頂点のマップ先として使用した状態で (u_2, v_5) を追加する再帰は不要である．予約集合によるガードはこのような再帰を削減する．

4.2.1 予約集合の定義

予約集合は次のように定義される．

定義 5 (予約集合)．集合 $R \subseteq V_G$ が次を満たすとき，R を (u_k, v) の予約集合と呼ぶ：任意の同型な埋め込み M について，$M(u_k) = v \implies \exists i \geqq k,\ M(u_i) \in R$.

定義 6 (予約集合とのマッチ)．M を長さ k の部分埋め込み，R を (u_k, v) の予約集合とする．$R \subseteq \mathrm{ran}(M)$ であるとき，M が R にマッチするという．

このような定義において，予約集合にマッチした部分埋め込みへのマッピングの追加はガードしてよい．

定理 1 (予約集合と同型性)．M を長さ k の部分埋め込み，R を (u_k, v) の予約集合とする．M が R にマッチするとき，$M \cup \{(u_k, v)\}$ およびその拡張は同型な埋め込みでない．

4.2.2 予約集合の生成

(u_i, v) の予約集合は u_i より後のクエリ頂点のマップ先として使用されるデータ頂点の集合なので，次のように再帰的に計算できる．$S(u_j, v')$ を (u_j, v') の予約集合とする．任意の u_i，v，および $u_j \in N_+(u_i)$ について，$S'(u_i, v; u_j)$ は (u_i, v) の予約集合である：

$$S'(u_i, v; u_j) = \bigcup_{v' \in C(u_j) \cap N(v)} S(u_j, v') \setminus \{v\}. \tag{2}$$

予約集合は小さい方がより多くの部分埋め込みへのマッチを期待できる．そのため最小の予約集合を与える $u_j \in N_+(u_i)$ を選択することで (u_i, v) の予約集合を $S(u_i, v; u_j)$ から得られる．しかしながら以下の条件を満たさない予約集合は部分埋め込みに決してマッチせず，ガードの効果がない．

補題 2 (マッチの必要条件)．R を (u_k, v) の予約集合とする．R にマッチする長さ k の部分埋め込み M が存在するならば，任意の正整数 $t < |R|$ について R は以下を満たす：

$$\left|\{u_i \mid i < k \text{ かつ } \exists j \leqq t, R[j] \in C(u_i)\}\right| \leqq t+1. \qquad (3)$$

ただし $R[j]$ $(j \geqq 0)$ は R に含まれる j 番目の要素.

$R \subseteq \mathrm{ran}(M)$ ならば $v \in C(f(v))$ であるような単射な写像 $f : R \to V_Q|_k$ が存在する．補題 3 はこれを近似的に確認する．

これを踏まえ，(u_i, v) の予約集合 $S(u_i, v)$ を次のように定義する．$S'(u_i, v; u_j)$ を (u_i, v) の予約集合 R と置いたときに式 3 を満たすような $u_j \in N_+(u_i)$ が 1 つ以上存在するならば，その中で最も小さい予約集合を与える u_j を選んで $S(u_i, v) = S'(u_i, v; u_j)$ とする．そうでなければ $S(u_i, v) = \{v\}$ とする．この計算は $S(u_i, v)$ の値を $A[u_i, v].R$ にメモ化することで効率化できる．

このように計算された予約集合をバックトラッキング中のガードに使用する．予約集合 R のマッチ判定には $O(|R|)$ 時間を要するが，補題 2 により $|R|$ は小さく保たれる．

4.3 D-prefix

同型になり得ないマッピングの組み合わせである D-pat は効果的な枝刈りを可能にするが，D-pat D が部分埋め込みに含まれることを確認するには $O(|D|)$ 時間を要する．予約集合と異なり D のサイズには上限がないので，このコストは問題になる場合がある．

軽量なガード判定のために，GuP は D-pat の特殊ケースである D-prefix を用いる．D-pat は任意のマッピングを含むが，D-prefix は常に連続なクエリ頂点 $u_0, u_1, \ldots$ のマッピングを含む．D-prefix P が部分埋め込み M に包含されることは M が P の拡張であることと同値である．そして M が P の拡張であることは次のように $O(1)$ で判定できる．まずバックトラッキングの再帰回数を変数 *embedID* でカウントする．*embedID* は再帰で渡された部分埋め込み M に一意に対応する**埋め込み** *ID* とみなすことができる．さらに M と一緒に M の拡張元の埋め込み ID を配列 *anc* に保持する．$anc[i]$ には $M|_i$ の埋め込み ID が代入されている．ただし $M|_i = \{(u_j, v) \in M \mid j < i\}$．ここで D-prefix P の埋め込み ID を $p.m$，長さを $p.k$ と置く．すると $anc[p.k] = p.m$ であることと M が P の拡張であることは同値である．例えば図 2 の探索木において，各ノードの ID m_i はそのノードに対応する部分埋め込みの埋め込み ID である [*2]．なお探索木には現れていないが，長さ 0 の部分埋め込み $\emptyset$ に対応する m_0 が存在するものとする．m_4 において $anc = (m_0, m_1, m_2, m_3, m_4)$ である．m_4 は長さ 2 の埋め込み m_2 の拡張であり，確かに $anc[2] = m_2$ となっている．このように，D-prefix を用いると小さいオーバーヘッドでガードを判定できる．

GuP は局所 D-prefix と大域 D-prefix という 2 種類の D-prefix を用いる．各候補に付与されるガードパターン $A[u_i, v].p$ は局所 D-prefix である．一方，大域 D-prefix p_{g} はアルゴリズム全体でただ一つ存在し，全ての候補に共通で適用される．それぞれの定義と生成方法を以下で説明する．

*2：分かりやすさのためノード ID は m_i と表記するが，実装上 m_i は整数 i と同一である．

4.3.1 局所 D-prefix

局所 D-prefix は次のように定義される．

定義 7 (局所 D-prefix)．部分埋め込み M と，クエリ頂点 u_k，データ頂点 v について，$M|_i \cup \{(u_k, v)\}$ が D-pat であるような $i < k$ が存在したとする．そのような i のうち最小のものを $\underline{i}$ と置いたとき，部分埋め込み $M|_{\underline{i}}$ を (u_k, v) の局所 deadend prefix (D-prefix) と呼ぶ．

定義 8 (局所 D-prefix とのマッチ)．M を長さ k の部分埋め込み，P を (u_k, v) の D-prefix とする．M が P の拡張であるとき，M が P にマッチするという．

「(u_k, v) の」と明示する場合は常に局所 D-prefix なので，単に「(u_k, v) の D-prefix」と書く．アルゴリズム上は前述の通り埋め込み ID によるマッチ判定を行う．具体的には，P の埋め込み ID $p.m$ と長さ $p.k$ を保持し，M の拡張元の配列 anc が $anc[p.k] = p.m$ を満たすならマッチと判断する．

局所 D-prefix にマッチした部分埋め込みへのマッピングの追加はガードしてよい．

定理 2 (局所 D-prefix と同型性)．M を長さ k の部分埋め込み，P を (u_k, v) の D-prefix とする．M が P にマッチするとき，$M \cup \{(u_k, v)\}$ とその拡張は同型な埋め込みでない．

D-prefix は D-mask を基に生成される．

補題 3 (局所 D-prefix の生成)．M を長さ $k+1$ の部分埋め込み，K を M の D-mask とする．さらに，もし $K = \{u_k\}$ ならば $i = 0$，そうでなければ $i = 1 + \max\{i \mid u_i \in K, i < k\}$ と置く．このとき $M|_i$ は $(u_k, M(u_k))$ の D-prefix である．

定義 8 では D-prefix の埋め込み ID $p.m$ を長さ $p.k$ を使用する．これらは補題 3 の i と anc を用いて $p.m = anc[i]$, $p.k = i$ として得られる．

4.3.2 大域 D-prefix

局所 D-prefix はクエリ頂点と候補のペア (u_k, v) に紐づき，それ以外の候補はガードしない．M を長さ k の部分埋め込み，v, v' を u_k の候補，K を $M \cup \{(u_k, v)\}$ の D-mask としたとき，もし $u_k \notin K$ ならば K は $M \cup \{(u_k, v')\}$ の D-mask でもある．つまり M への (u_k, v') の追加もガードしてよい．しかし K から作られる (u_k, v) の局所 D-prefix は (u_k, v') のガードには使用されない．このような局所 D-prefix の欠点を補うため，GuP は大域 D-prefix を用いる．

定義 9 (大域 D-prefix)．部分埋め込み M について，$M|_i$ が D-pat であるような i が存在したとする．そのような i のうち最小のものを $\underline{i}$ と置いたとき，部分埋め込み $M|_{\underline{i}}$ を大域 deadend prefix (D-prefix) と呼ぶ．

定義 10 (大域 D-prefix とのマッチ)．M を部分埋め込み，P を大域 D-prefix とする．M が P の拡張であるとき，M が P にマッチするという．

部分埋め込みが大域 D-prefix にマッチするなら任意のマッピングの追加をガードしてよい．

定理 3 (大域 D-prefix と同型性)．M を部分埋め込み，P を大域 D-prefix とする．M が P にマッチするとき，M とその拡張は同型な埋め込みでない．

補題 4 (大域 D-prefix の生成)．M を部分埋め込み，K を M の D-mask とする．さらに $i = 1 + \max\{i \mid u_i \in K\}$ と置く．このとき $M|_i$ は大域 D-prefix である．

大域 D-prefix もアルゴリズム上は埋め込み ID に基づくマッチ判定と生成を行う．局所 D-prefix とほぼ同様であるため，説明は割愛する．

4.4 D-mask の選択

D-prefix を生成するためには同型にならない部分埋め込みの D-mask を得る必要がある．アルゴリズム 3 において同型にならないことを判定しているのは次の 4 か所である：単射制約違反（6 行目），ガード（9 行目），候補の喪失（15 行目），再帰（16 行目）．このうち単射制約違反のみ補題 1 の D-mask 選択ルールを使用できる．他の 3 か所は定義 1 と異なる条件によって判定しているため，それぞれに対応する D-mask の選択ルールを以下に示す．

4.4.1 ガードの D-mask

予約集合と D-prefix によって D-mask が異なる．

a） 予約集合の D-mask

予約集合とのマッチによってガードされる場合の D-mask は次の通り与えられる．

補題 5 (予約集合に基づく D-mask)．M を長さ k の部分埋め込み，R を (u_k, v) の予約集合とする．M が R にマッチするとき，$\{u_i \mid \exists v \in R, (u_i, v) \in M\}$ は M の D-mask である．

b） D-prefix の D-mask

D-prefix は D-mask から生成されるため，D-prefix にマッチする部分埋め込みにも同じ D-mask を適用できる．

補題 6 (D-prefix に基づく D-mask)．M を部分埋め込み，P を D-prefix，K を P の生成元となった D-mask とする．M が P にマッチするとき，K は M の D-mask である．

D-mask を得るため，アルゴリズム上の D-prefix は埋め込み ID $p.m$ と長さ $p.k$ に加えて生成元の D-mask $p.K$ を保持する．$p.K$ はビットベクトルで表現できるため，メモリ消費は小さい．

4.4.2 候補喪失の D-mask

局所候補がなくなった場合の D-mask は辺制約集合によって与えられる．

定義 11 (辺制約集合)．M を長さ k の部分埋め込み，u_i をクエリ頂点とする．集合 $F \subseteq N(u_i)|_k$ が次を満たすとき，F を M における u_i の辺制約集合と呼ぶ：

$$\{v \in C(u_i) \mid \forall u_j \in F, v \in N(M(u_j))\} = C(u_i; M). \quad (4)$$

式 4 は式 1 の $N(u_i)|_k$ を F で置き換えたものであることに注目されたい．つまり辺制約集合とは，局所頂点集合の計算に影響しない頂点の除外を許した隣接頂点集合である．これを用いて以下のように D-mask を求める．

補題 7 (候補喪失の D-mask)．M を長さ k の部分埋め込みとする．ある u_i $(i \geq k)$ の局所候補が M の下で存在しないならば，M における u_i の辺制約集合は M の D-mask.

アルゴリズム 3 の 12–15 行目はマッピング (u_k, v) を追加した後の局所候補集合 C'_M と辺制約集合 F'_M を計算する．マッピングの追加によってクエリ頂点 u_i の候補が減少する場合，u_i の辺制約集合に u_k を追加する．

4.4.3 再帰の D-mask

関数 BACKTRACK は引数 M を拡張し，そこから同型な埋め込みを得られなかった場合に M の D-mask を返却する．その場合の D-mask は次のように得られる．

補題 8 (D-mask の集約)．M を長さ k の部分埋め込み，$C_M(u_k)$ を M の下での u_k の局所候補集合，F を M の下での u_k の辺制約集合とする．さらに集合 $K \subseteq V_Q$ は任意の $v \in C_M(u_k)$ について $M \cup \{(u_k, v)\}$ の D-mask であるとする．このとき，次のように定義される $\mathrm{agg}(K; k, F)$ は M の D-mask である：

$$\mathrm{agg}(K; k, F) = \begin{cases} K|_k \cup F & \text{if } u_k \in K, \\ K|_k & \text{otherwise.} \end{cases} \quad (5)$$

補題 9 (共通 D-mask)．$\mathcal{M}$ を長さ k の部分埋め込みの集合，$\mathcal{K} = \{M \text{ の D-mask} \mid M \in \mathcal{M}\}$ とする．このとき $\bigcup \mathcal{K}$ は任意の $M \in \mathcal{M}$ の D-mask である．

アルゴリズム 3 の 21 行目では補題 8 と 9 に基づいて M の D-mask を計算する．前述の通り，D-prefix $A[u_k, v].p.K$ にはその生成元となった $M \cup \{(u_k, v)\}$ の D-mask が保存されていることに注意されたい．

以上のように得た D-mask から D-prefix を生成することで GuP は効率的な探索を行う．

5 評　　価

本節では GuP に関する性能評価の結果を紹介する．

5.1 実験設定

実験には Xeon E7-8890 4 基（合計 72 コア）と 2 TB のメモリを備えたマシンを用いた．GuP および比較手法のプログラムはそれぞれ物理 1 コアで逐次実行した．ただし実験時間短縮のため，これらのプログラムを最大 70 プロセスまで同時に起動し，複数のクエリについての実験を並列に行った．性能比較には 2019 年以降の手法である DAF [5][*3]，RapidMatch (RM) [8][*4]，および文献 [7][*5]の GQL-G と GQL-R を用いた．GQL-G と GQL-R は交叉による局所候補集合計算と GraphQL の候補フィルタリングにそれぞれ GraphQL と RI [15] によるマッチング順序最適化を組み合わせた手法である．

データグラフにはサブグラフマッチングの研究で広く用い

*3：https://github.com/SNUCSE-CTA/DAF
*4：https://github.com/RapidsAtHKUST/RapidMatch
*5：https://github.com/RapidsAtHKUST/SubgraphMatching

られている [4–8] Yeast（3,112 頂点，12,519 辺，71 ラベル），Human（4,674 頂点, 86,282 辺, 44 ラベル）と WordNet（76,853 頂点，120,399 辺，5 ラベル）に加え，より大規模なグラフとして Patents（3,774,768 頂点，16,518,947 辺，20 ラベル）[7] を用いた．Patents 以外は元々ラベルを持った実世界のグラフである．Patents のラベルは文献 [7] でランダムに割り当てたものを使用した．クエリグラフは既存研究 [7] と同様にデータグラフから抽出した．具体的にはデータグラフ上でランダムウォークを行い，訪問した頂点の誘導部分グラフをクエリとした．平均次数 3 未満のクエリグラフを sparse (S)，3 以上のクエリグラフを dense (D) に分類し，頂点数を 8 から 32 まで変化させながらクエリセット 8S, 16S, 24S, 32S, 8D, 16D, 24D, 32D をデータグラフごとに生成した．各セットは 5 万個のクエリグラフを含む．既存研究では 100 または 200 クエリを 1 クエリセットとしている [4–8] のに対し，本論文はランダム性による結果のぶれを抑えるため多くのクエリを使用した．既存研究 [4–7] と同様，埋め込みを 10^5 個発見した時点でそのクエリグラフについての探索を打ち切った．さらにクエリグラフとクエリセットのそれぞれにタイムアウトを設定した．各クエリグラフについての探索は 1 時間で打ち切った．クエリセットは 100 クエリごとのグループに分割し，いずれかのグループで 100 クエリ合計の処理時間が 3 時間を超えたとき，クエリセット全体について ‘did not finish’ (DNF) と判定した．

5.2 クエリ処理時間

サブグラフマッチングのクエリ処理時間はミリ秒オーダから計測が困難なほどの長時間まで幅広く，平均値による比較は適当でない．そこで既存研究 [6,7,14] に倣い，本論文では処理時間ごとにクエリを分類しその数を比較した．具体的には秒オーダ（1 秒以上 1 分未満），分オーダ（1 分以上 1 時間未満），時間オーダ（1 時間以上，つまりタイムアウト）の 3 つに分類し，それらをクエリセットごとに数えた．システム利用者の思考を途切れさせない待ち時間の上限が 1 秒とされる [16] ことから，応答が 1 秒以内であることは重要である．

実験の結果を図 3 に示す．これは積み上げ棒グラフなので，棒全体の高さは 1 秒以上を要したクエリの数となる．まず Yeast の結果に注目すると，GuP は 16S と 24S においていずれのクエリも 1 秒以内に処理できた唯一の手法である．また Yeast の 32S をはじめ，WordNet を除くほとんどのクエリセットにおいて GuP は 1 秒以上要したクエリが最も少ない．さらに時間オーダのクエリ数は全てのクエリセットにおいて GuP が最少である．Human の 24S，Patents の 32D においては GuP だけが DNF とならなかった．表 2 に全クエリセットについて処理時間ごとのクエリ数を合計した値を示す．ただし DNF となった手法があるクエリセットは集計から除いた．この集計条件において GuP は全てのクエリを 1 時間以内に処理できた．また秒オーダと分オーダのクエリも最少であり，2 番目に少ない手法と比べそれぞれ 88.7%と 98.7%削減した．このように，GuP は既存手法と比べ顕著に高い性能を示した．

さらに Yeast のクエリセットごとの合計処理時間を図 4 に示す．タイムアウトしたクエリは処理時間が 1 時間だったものとして集計した．GuP にはガードのマッチを確認するためのオーバーヘッドがあるものの，全体的に他の手法に比肩する性能を示した．なお，クエリセットの処理時間はタイムアウトの長さによって大きく変化し得ることに留意されたい．Yeast 以外のデータセットではタイムアウトが多く発生しているため，本論文では処理時間を議論しない．

表 2: 処理時間別のクエリ数．ただしいずれかの手法で DNF となったクエリセットは集計から除外している．

	GuP	DAF	GQL-G	GQL-R	RM
1 秒以上 1 分未満	481	26258	98264	97800	4273
1 分以上 1 時間未満	1	546	80	88	86
1 時間以上	0	167	48	66	51

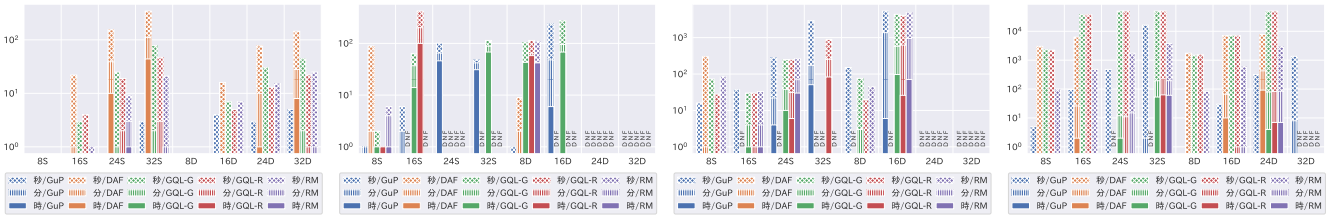
5.3 各ガードの効果

全てのガードを無効化した状態をベースラインとし，予約集合，局所 D-prefix，大域 D-prefix のいずれか 1 つのみを有効にした実装が削減した再帰の回数を図 5 に示す．ガードを無効化するとクエリ処理時間が増大するため，Yeast と Human の一部のクエリセットのみ使用し，さらに 5 万クエリのうち最初の 1000 クエリのみ使用した．図中の ‘GuP’ は全てのガードを有効にした場合の削減数である．全体的に局所 D-prefix の削減数が最も多く，全ガードを有効にした場合に近い．次に大域 D-prefix の効果が大きい．これらと比べ予約集合の効果は限定的である．D-prefix が単射制約と辺制約の両方に起因する不要な再帰を削減できるのに対し，予約集合は単射制約にのみ対応する．このことが削減数の差を生んだ可能性がある．

さらにガードの効果を確認するため，Yeast の 24S についてクエリごとの再帰回数を調査した．GuP の再帰削減数が多い順に表 3 に示す．#Q はクエリセットにおけるクエリの番号である．「割合」は，全ガードが有効な GuP と比較したときの各ガード単体での削減数の割合である．例えば 1 行目のクエリ 601 では局所 D-prefix が 99.9%，大域 D-prefix が 100.0%なので，このいずれかのガードのみ用いた場合でも全ガードを併用する GuP とほぼ同じ削減数を達成できる．このクエリに対して予約集合の削減割合は 0.0%なので，全く性能に貢献していない．これに対し，2 行目のクエリ 541 では予約集合の削減割合が 100.0%である一方，局所 D-prefix の削減割合は 90.2%に低下した．このようにそれぞれのガードパターンは効果を発揮するクエリが異なる．それらの併用により多様なクエリで不要な再帰を削減できることが GuP の高い性能の理由である．

6 結　　論

本論文はサブグラフマッチングのための新しいアルゴリズムである GuP を提案した．GuP は候補のそれぞれにガードパターンを付与する．そして探索中，ガードパターンにマッチする部分埋め込みにはその候補へのマッピングを追加しない．これにより同型な埋め込みを発見できない再帰を削減する．実験



(a) Yeast **(b)** Human **(c)** WordNet **(d)** Patents

図 3: 各クエリセットにおけるクエリ処理時間の分布．縦軸はクエリ数．

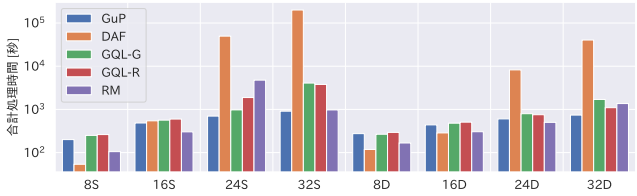
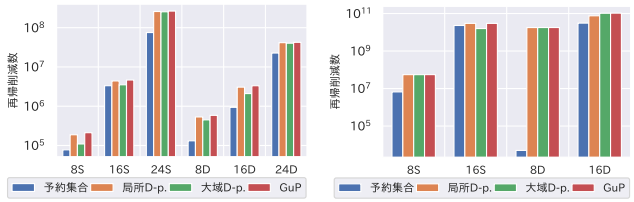


図 4: Yeast における各クエリセット全体の処理時間



(a) Yeast **(b)** Human

図 5: 再帰呼び出しの削減回数

表 3: 各ガードの再帰削減数と GuP の削減数に対する割合

	GuP	予約集合		局所 D-prefix		大域 D-prefix	
#Q	削減数	削減数	割合	削減数	割合	削減数	割合
601	181580k	846	0.0%	181407k	99.9%	181578k	100.0%
541	64210k	64210k	100.0%	57917k	90.2%	64210k	100.0%
728	8031k	8030k	100.0%	8027k	99.9%	0	0.0%
65	3691k	890k	24.1%	3680k	99.7%	3312k	89.7%
103	3430k	523k	15.2%	3429k	100.0%	3364k	98.1%

において GuP は最新の手法と比べて高い性能を示した．

文　　献

[1] W.-S. Han *et al.*, “TurboISO: Towards Ultrafast and Robust Subgraph Isomorphism Search in Large Graph Databases,” SIGMOD, 2013.

[2] J. R. Ullmann, “An Algorithm for Subgraph Isomorphism,” *JACM*, vol. 23, no. 1, pp. 31–42, 1976.

[3] L. P. Cordella *et al.*, “A (Sub)Graph Isomorphism Algorithm for Matching Large Graphs,” *TPAMI*, vol. 26, no. 10, pp. 1367–1372, 2004.

[4] F. Bi *et al.*, “Efficient Subgraph Matching by Postponing Cartesian Products,” SIGMOD, pp. 1199–1214, 2016.

[5] M. Han *et al.*, “Efficient Subgraph Matching: Harmonizing Dynamic Programming, Adaptive Matching Order, and Failing Set Together,” SIGMOD, pp. 1429–1446, 2019.

[6] S. Sun and Q. Luo, “Subgraph Matching with Effective Matching Order and Indexing,” *TKDE*, pp. 491–505, 2020.

[7] S. Sun and Q. Luo, “In-Memory Subgraph Matching: An In-depth Study,” SIGMOD, pp. 1083–1098, 2020.

[8] S. Sun *et al.*, “RapidMatch: A Holistic Approach to Subgraph Query Processing,” *VLDB*, vol. 14, no. 2, pp. 176–188, 2020.

[9] H. He and A. K. Singh, “Graphs-at-a-time: Query Language and Access Methods for Graph Databases,” SIGMOD, pp. 405–418, 2008.

[10] 淳. 新井 *et al.*, “探索失敗履歴を用いた高速サブグラフマッチング,” DEIM, pp. 1–9, 2018.

[11] J. Arai *et al.*, “Fast Subgraph Matching by Exploiting Search Failures,” *CoRR*, 2020.

[12] P. Zhao and J. Han, “On Graph Query Optimization in Large Networks,” *VLDB*, vol. 3, no. 1-2, pp. 340–351, 2010.

[13] H. Shang *et al.*, “Taming Verification Hardness: An Efficient Algorithm for Testing Subgraph Isomorphism,” *VLDB*, vol. 1, no. 1, pp. 364–375, 2008.

[14] F. Katsarou *et al.*, “Subgraph Querying with Parallel Use of Query Rewritings and Alternative Algorithms,” EDBT, pp. 25–36, 2017.

[15] V. Bonnici *et al.*, “A subgraph isomorphism algorithm and its application to biochemical data,” *BMC Bioinformatics*, vol. 14, no. 7, p. S13, 2013.

[16] J. Nielsen, *Usability engineering*. Interactive Technologies, Elsevier, 1994.