DEIM Forum 2022 I44-1 (day4 p20)

グラフ編集距離に基づく部分グラフ探索

小出　智士† 　河野　圭祐† 　塩川　浩昭†† 　天笠　俊之††

† 株式会社 豊田中央研究所　〒 480–1192 愛知県長久手市
†† 筑波大学 計算科学研究センター　〒 305–0006 茨城県つくば市
E-mail: †{koide,kawano}@mosk.tytlabs.co.jp, ††{shiokawa,amagasa}@cs.tsukuba.ac.jp

あらまし　本研究ではグラフデータベースを探索し，与えられたクエリグラフと類似する部分グラフを探索する「類似部分グラフ検索問題」を取り扱う．類似度関数として**グラフ編集距離** (graph edit distance) を用いる．グラフ編集距離を与えられた 2 つのグラフに対して計算すること自体が計算量的に困難な問題であり，したがって編集距離の意味で類似する部分グラフをデータベースから発見する問題は計算量の意味で大きなチャレンジを伴う．この問題を解くためのアルゴリズムの実現可能性を探るため，本研究ではアルゴリズムおよびデータ構造の検討を行った．具体的には部分グラフ探索時の深さ優先探索を列挙し，トライ木に格納することでラベル列が重複する DFS の探索を共通に行うことで計算量の削減を行う，というアルゴリズムを提案する．小規模なグラフデータベースに対して提案アルゴリズムを適用し，アルゴリズムをスケールさせるための課題の抽出を行った．

キーワード　グラフデータベース, 類似度検索, グラフ編集距離

1 はじめに

グラフデータベースに対する検索は基本的かつ重要な問題である．クエリグラフ h およびターゲットグラフ g が与えられたとき，g が h と同型な部分グラフを含むかどうか，を判定する部分グラフ同型 (subgraph isomorphism) 問題は NP-hard であることが知られている．一方で部分グラフ同型問題は，例えば「高分子グラフ g があるモチーフグラフ h を含むか？」というような応用上重要なタスクを含むものである．

本研究では **類似部分グラフ検索問題** を取り扱う．これは部分グラフ同型問題の一般化とみなすことができる問題である．すなわち，クエリグラフ h およびターゲットグラフ g が与えられたときに，h と**類似するような g の部分グラフを発見する問題**である (具体的には 2 章で定義する)．同型性でなく類似性に着目することで，構造的に類似したグラフに対する統計分析の実施や，データやクエリにノイズが含まれている場合でも検索が可能になるなど，より高度な処理が可能になる．

一方で，類似部分グラフ検索問題が挑戦的な問題であることは以下のように理解することができる．グラフ間の距離関数を $d(g,h)$ と書いた場合，典型的な類似度検索は $d(g,h) \leqq \tau$ となる部分グラフを検索する，というものになるが，$\tau = 0$ としてみると，この問題は部分グラフ同型問題 (NP-hard) に帰着してしまう [1]．

本研究では距離関数として**グラフ編集距離** (graph edit distance; GED) を用いる．グラフ編集距離はノードやエッジのリラベリング，挿入，削除操作によってグラフ g を h に変換するのに必要な最小の操作回数として定義される．これは文字列の編集距離同様，直感的かつ自然なものであり，距離の公理を満たすような筋の良いものである．このことより，グラフ編集距離は similarity join [7] や supergraph similarity search [5], などのタスクで広く研究されている．一方で距離計算自体が NP-hard であることが知られており [6]，計算コストの削減がグラフ編集距離に基づくマイニングアルゴリズムでは重要になる．

本研究の貢献は以下の通りである．

- グラフ編集距離ベースの類似部分グラフ検索問題を定義した (2 章).
- いくつかのベースラインアルゴリズムを考え (3 章)，それらの問題を解決する形で，効率的なデータ構造およびアルゴリズムを提案する (4 章).
- 実験によって，小規模なグラフデータベースに対しては十分に実用的な処理時間で，問合せ処理を実現可能であることを示した (5 章).

2 問題定義

2.1 グラフ編集距離

本研究において，**グラフ** はノードラベルおよびエッジラベルをもつ無向グラフを意味するものとする．グラフ g のノード集合を $V(g)$，エッジ集合を $E(g)$ と記す．グラフ編集距離は，あるグラフ g を別のグラフ h に変換するためのノードのラベルの置換，およびノード挿入・削除操作の最小の回数として定義される．ノード数が異なる場合 ($|V(h)| < |V(g)|$ である場合)，h に空ラベル ε を持つ孤立ノードを追加して数を揃えておくことでグラフ編集距離 $ed(g,h)$ は以下のように書くことができることが知られている．

1：距離の公理 $d(g,h) = 0 \Rightarrow g = h$ (同一律) を仮定している．逆に言えば NP-hard でないような類似部分グラフ検索アルゴリズムがあるとすれば，それは距離の公理を満たさないような擬距離に基づくものである．

$$ed(g,h) = \min_{\varphi \in \Phi(|V(g)|)} dist(g,h,\varphi) \quad (1)$$

$$\text{where } dist(g,h,\varphi) = \sum_{i=1}^{|V(g)|} c_n(i,\varphi(i)) + \sum_{i=1}^{|V(g)|-1} \sum_{j=i+1}^{|V(g)|} c_e\big((i,j),(\varphi(i),\varphi(j))\big) \quad (2)$$

ここで $\Phi(m)$ は $\{1,\cdots,m\}$ の置換全体の集合，c_n, c_e はそれぞれノードラベル，エッジラベルを書き換えるための編集コストであり，本研究ではラベルが同じ場合は 0，異なる場合は 1 である最も単純なケースのみを考える．置換を用いたこの定義の意味合いは「あるノードマッチングを考えた場合の編集コストを考え,」「すべてのノードマッチングを考えてコスト最小となるマッチングを採用する」というものである (ノード数が異なる場合に追加された空ラベルとマッチングしたノードは編集によって「削除」されるとみなされる).

計算量 グラフ編集距離を上記の定義に従って計算するためにはグラフのノード数 $n = \max\{|V(g)|, |V(h)|\}$ に対して $O(n!)$ の置換を考慮する必要があり，巨大なグラフに対しては計算困難である．実際，グラフ編集距離の計算は NP-hard であることが既に知られている [6]．したがって巨大なグラフに対するグラフ編集距離の計算は困難であるが，計算効率を向上させるための様々なアプローチが検討されており，ある程度のサイズまでのグラフまでは計算可能である．具体的には，上述した置換 (ノードマッピング) φ を A* アルゴリズムを用いて最適化するアルゴリズムが利用可能である [7]．また，必ずしも最小の編集回数でなく，真の編集距離の上限を与える近似アルゴリズムも研究されている (例えば [3]) が，今回は近似は行わない．

2.2 類似部分グラフ検索問題

定義 1 (類似部分グラフ検索問題)．グラフデータベース $\mathcal{G} = \{g_i\}_{i=1}^N$，およびクエリグラフ h，およびしきい値 τ が与えられたとき，以下で定義される類似部分グラフの集合 R を発見する問題を**類似部分グラフ検索問題**と定義する:

$$R = \{g \mid \exists g \subseteq g_i \text{ s.t. } ed(g,q) \leqq \tau, g \text{ is connected}, |V(g)| \leqq k\}.$$

ここで $g \subseteq h$ は g が h の (ノード) 誘導部分グラフであることを意味する．最後の条件 $|V(g)| \leqq k$ は探索する部分グラフのサイズを制限するものである (この条件は技術的な制約から導入されたものであり，可能な限り大きく取りたい．この問題については 4.4 節で議論する).

3 ベースライン

我々の知る限り，編集距離に基づく類似部分グラフ検索問題はこれまで考えられてこなかった問題であるが，グラフ編集距離に関する既知の事実を用いることで (効率性は度外視した) いくつかの解法を構成することは可能である．ここではベースラインとなるアルゴリズムを 2 つ述べる (深さ優先探索，および q-gram 索引)．また，それらの問題点について議論し，(次節に) 問題の部分的な解決を図る．

3.1 深さ優先探索

連結な部分グラフ g のうちで $ed(g,q) \leqq \tau$ となるものを探索するためのシンプルなアルゴリズムとして深さ優先探索 (DFS) を用いることができる (Algorithm 1)．このアルゴリズムは初期ノード $S = \{v \in V(g)\}$ を選び (Line 4)，(2) S の近傍ノード集合 $\mathcal{N}(S)$ 内のノードを一つ選んで S に追加して再帰する (Line 10)，という手続きである．この再帰の途中では，現状のノード集合 S のサイズが k 以下であれば S の誘導部分グラフ g を生成し (Line 6)，編集距離 $ed(g,q) \leqq \tau$ をチェックする (満たす場合は結果として出力を行い，再帰を続ける)．この手続をすべての $\mathcal{G}$ のグラフ g_i およびそのノード $v \in V(g_i)$ に対して実行する (Line 2–3) ことでサイズ k 以下のすべての連結な部分グラフ g に対して $ed(g,q) \leqq \tau$ をチェックしたことになるため，このアルゴリズムは正しく動作する．その一方で，このアルゴリズムは極めて多数の編集距離計算を必要とするため現実的ではない．

Algorithm 1 Baseline-1: DFS-based similarity search

```
1: function SearchBaseline-1(G, q, τ)
2:     for all g_i ∈ G do
3:         for all v ∈ V(g_i) do
4:             S ← {v}
5:             DfsNaive(g_i, q, τ, S)
6: function DfsNaive(g_i, q, τ, S)
7:     g ← DerivedSubgraph(g_i, S)
8:     if |S| ≦ k and ed(g, q) ≦ τ then
9:         output (i, g)      ▷ Graph ID and its derived subgraph
10:    for all v ∈ N(S) do
11:        DfsNaive(g_i, q, τ, S ∪ {v})
```

3.2 q-gram 索引

Algorithm 1 における編集距離計算の回数を減らすために，Zhao et al. で用いられた q-gram 索引 [7] を用いることができる．文字列の編集距離に対する q-gram 索引 [2], [4] と同様に索引対象となるグラフ g からノード数 q の path graph をすべて列挙する．この path graph のラベル列 (ノードラベル q 個，エッジラベル $q-1$ 個の合計 $2q-1$ 個の文字からなる文字列) の multiset $\Pi(g)$ を考えたとき，$ed(g,h) \leqq \tau$ ならば $\Pi(g)$ と $\Pi(h)$ は少なくとも

$$\max\big(|\Pi(g)| - \tau \cdot D(g), |\Pi(h)| - \tau \cdot D(h)\big) \quad (3)$$

個の q-gram を共有する，ということが知られている [**?**]．ここで $D(g) = \max_{v \in V(g)} |\Pi(g,v)|$ であり，$\Pi(g,v)$ はグラフ g のノード v を含むような q-gram の multiset である．したがって，$g_i \in \mathcal{G}$ の誘導部分グラフ g で $ed(g,h) \leqq \tau$ を満たすものが存在するならば $\Pi(g_i)$ と $\Pi(h)$ は少なくとも

$$LB_{\text{path}}(h) := |\Pi(h)| - \tau \cdot D(h) \quad (4)$$

個の q-gram を共有する必要がある [2]．さらに prefix filtering 原理 [?] より $\Pi(h)$ の $(\tau \cdot D(h)+1)$-prefix に含まれる q-gram を少なくとも一つもつようなグラフ $g_i \in \mathcal{G}$ とその部分ノード集合 $S \subseteq V(g_i)$ を対象にすればよいことがわかる [3]．

これより，Algorithm 1 の亜種を構築することができる．g_i に対する q-gram 転置索引を作っておき，クエリ h の q-gram $s \in \Pi(h)$ の転置索引 L_s からラベル列 s を持つようなグラフ g_i および対応するノード集合 $S \subseteq V(g_i)$ を逆引きし，それを Algorithm 1 における初期シードノード集合 S として用いて，同様に深さ優先探索することで探索範囲を限定することができる．

Algorithm 2 Baseline-2: q-gram-based similarity search

```
1: function SearchBaseline-2({L_s}, h, τ)
2:     α ← |Π(h)| − τ · D(h)
3:     for all s ∈ α-prefix of Π(h) do
4:         for all (g_i, S) ∈ L_s do            ▷ Postings-list of s
5:             DfsNaive(g_i, h, τ, S)           ▷ See Algorithm 1
```

4 提案手法

4.1 概　　要

前章で導入したベースラインはデータベース $\mathcal{G}$ の各グラフ g_i それぞれについて DFS によって部分グラフを探索するものであった．一方，データベース中には同型な部分グラフが複数回出現する (例えば，化合物データベース中のベンゼン環など)．ベースラインでは，これらの同型なグラフに対して重複してグラフ編集距離を計算してしまうが，これを省略できれば計算コストの削減につながると期待できる．ところがグラフ同型性の判定自体が計算量の大きな操作であるため，このアイデアを実装するためには工夫が必要になる．

提案手法では Yamada et al. で提案された AcGM コード [5] を用いる．これはグラフ同型性の判定を文字列比較の問題に変換する方法である．得られた文字列を trie に格納することで同型な部分グラフが trie の一つの節点に対応づけることができるため，計算の重複を回避することができる．以下では，まず必要なコンセプトを導入し，続いて提案手法について述べる．

4.2 DFS と AcGM コード

ここではグラフ上の部分グラフを列挙する DFS と関連する概念である prefix-connected permutation を導入する．さらに，ノードに順序が定まっているような部分グラフを文字列として表現する AcGM コードについて述べる．これらの概念は Yamada et al. [5] で導入されたものである．

2：ここでは式 (3) のように max を取ることはできない．なぜなら，比較対象の g_i の誘導部分グラフ g は多数あり，g_i の情報のみから $|\Pi(g)| - \tau \cdot D(g)$ の値を確定させることはできないためである．

3：Multiset Π の α-prefix とは，集合要素に global ordering を仮定してソートしたときの長さ α の prefix と定義される．どのような global ordering を仮定してもよいが，探索コストを下げるために出現頻度の低い q-gram が辞書順で小さくなるような順序を仮定するのがよい．

a） Prefix-connected permutation

$\Omega^+(g)$ を g のノード集合の全ての置換の集合と定義する．例えば $V = \{v_1, v_2, v_3\}$ としたとき，

$$\Omega^+(g) = \{(v_1,v_2,v_3),(v_1,v_3,v_2),(v_2,v_1,v_3),\\ (v_2,v_3,v_1),(v_3,v_1,v_2),(v_3,v_2,v_1)\}$$

となる．*Prefix-connected permutation* (PCP) とは $\Omega^+(g)$ のうちで，任意の prefix による誘導部分グラフが連結であるものと定義する．グラフ g の PCP の集合を $\Omega(g)$ と書く．定義より $\Omega(g) \subseteq \Omega^+(g)$ である (等号成立は g が全連結のときに限る)．たとえば $v_1 \leftrightarrow v_2 \leftrightarrow v_3$ という鎖グラフを考えた場合，$\Omega(g) = \{(v_1,v_2,v_3),(v_2,v_1,v_3),(v_2,v_3,v_1),(v_3,v_2,v_1)\}$ となる．定義より，ある $\Omega(g)$ の要素 $(v_{i_1},\cdots,v_{i_m})$ の長さ k の prefix $(v_{i_1}\cdots,v_{i_k})$ から誘導される部分グラフは連結である．また， PCP を一つ選ぶことは $v_{i_1} \to v_{i_2} \to \cdots \to v_{i_k}$ という順序のグラフ上の DFS をエミュレートしている，と見ることができる．したがって $\Omega(g)$ 全体を考えることで g 上で可能な DFS のバリエーションをすべて列挙している，と考えることができる．同様に $\Omega(g;k)$ をサイズ $k \leqq n := |V(g)|$ までの PCP の集合と定義し，k-PCP と呼ぶ．すなわち $\Omega(g;k) = \{(v_{i_1},\cdots,v_{i_k}) \mid (v_{i_1},\cdots,v_{i_n}) \in \Omega(g)\}$ と定義される．($n < k$ のときは $\Omega(g;k) = \Omega(g)$ とする)

b） AcGM code

グラフ g およびそのノード集合 $V(g)$ を考え，ノード集合上に順序 $\pi = [v_1, v_2, \cdots, v_n]$ があたえられているものとする．

定義 2. g および π が与えられたとき，その AcGM コードは $c(g;\pi) = f_1 f_2 \cdots f_n$ のように文字列 $f_1, f_2, \cdots f_n$ を連結したものとして定義される．ここで f_i は **AcGM fragment** と呼ばれる長さ i の文字列 $f_i = l(v_i)\phi_{1,i}\phi_{2,i}\cdots\phi_{i-1,i}$ と定義され，$l(v_i)$ はノード v_i のノードラベル，

$$\phi_{i',i} = \begin{cases} l(v_i,v_{i'}) & (v_i,v_{i'}) \in E(g) \\ \varepsilon & (v_i,v_{i'}) \notin E(g) \end{cases} \tag{5}$$

である．ただし，$l(u,v)$ はエッジ $(u,v) \in E(g)$ のエッジラベルである．

AcGM コード c が与えられたとき，そこからグラフを一意に復元することができる．したがって AcGM コード c, c' が文字列として等しいとき，対応するグラフ g, g' は同型である．

4.3 木構造による PCP の索引化

グラフデータベース $\mathcal{G}$ の各グラフ g_i に対して以下のように索引を構築する．

（1） g_i の k-PCP を列挙し，各 PCP $(v_{i_1},\cdots,v_{i_k}) \in \Omega(g)$ に対して AcGM コードを計算する．ここで k はユーザが決定するパラメータである．

（2） Trie に AcGM コードを格納する．具体的には AcGM コード $c = f_1 \cdots f_n$ に対して，深さ i の枝に f_i を割当てるパトリシア木の構造を採用する．Leaf にはグラフの id i および

PCP $(v_{i_1}, \cdots, v_{i_k})$ を格納する．すでに leaf が存在している場合はリスト形式ですべてのデータを保持するようにする．

このようにして構築される trie を T と記す．このようなデータ格納によって T のある leaf に格納された部分グラフは，連結かつサイズ k 以下であり，かつすべて同型となる．また，root から leaf への経路上にある深さ d の接点は leaf に格納された部分グラフの連結なサイズ d の部分グラフに対応する．

4.4 類似検索の実現

クエリ q に対して trie T を検索することで類似検索を実現する．T の root からある leaf までの経路は DFS をエミュレートしていることに注意すると，途中の接点 u で出現する部分グラフ g_u に対して q との編集距離を計算することで検索を実現することができる．フィルタリングなどの高速化処理を行わない単純なアルゴリズムは以下のようになる．

（1） ω を T のルートノードとする．

（2） ω に対応する誘導部分グラフを考え，それを g_ω とする．$ed(h, g_\omega) \leqq \tau$ ならば結果を出力する．[4]

（3） ω の子ノード ω' に対して同様の処理を再帰的に繰り返す．

この処理を具体化したものが Algorithm 3 である．誘導部分グラフ g_ω はノード ω からルートノードまでの AcGM fragment を結合して得られる AcGM code c_ω から復元することができる (Line 5).

Algorithm 3 A simple trie-based similarity search

```
1: function SearchTrieSimple(T, h, τ)
2:     (ω, c_ω) ← (T.root, ε)   ▷ Trie root, empty AcGM fragment
3:     DfsTrie(ω, c_ω, h, τ)
4: function DfsTrie(ω, c_ω, h, τ)
5:     g_ω ← GraphFromAcGM(c_ω)
6:     if ed(g_ω, h) ≦ τ then output ω   ▷ Descendants of trie node include the corresponding graph ids and the subgraph information.
7:     for all (ω', f) in ω.children do   ▷ Iteration over all the child nodes and its AcGM fragment
8:         DfsTrie(ω', c_ω f, h, τ)   ▷ c_ω f: Extended AcGM code
```

このアルゴリズムは以下の問題がある．

[**問題 1**] 訪れた各ノード ω で $ed(h, g_\omega)$ を計算する必要があり，計算量が大きい．

[**問題 2**] DFS の打ち切り条件がないため，trie T の全ノードを訪れることになる．

[**問題 3**] Trie の深さ k よりも大きなサイズの類似部分グラフを発見することができない．

以下ではこれらの問題について解決策などを議論する．

4：ノード ω の子孫の葉ノードにグラフ ID および部分グラフの情報が格納されているため，そこから復元することができる．

a） 編集距離計算の軽量化 (**問題 1**)

訪れた各ノード ω における $ed(h, g_\omega) \leqq \tau$ のチェックのための編集距離計算をできるかぎり回避するため，編集距離の下限を用いた計算コストの削減を行う．具体的には $LB(h, g_\omega) \leqq ed(h, g_\omega)$ となる (計算量の小さな) 関数 $LB(h, g_\omega)$ を計算し，$LB(h, g_\omega) > \tau$ となれば実際に $ed(h, g_\omega)$ を計算することなく $ed(h, g_\omega) \leqq \tau$ ではありえないことがわかる．

このような下限として，本研究はラベル分布に基づくものを考える．これはグラフ h と g_ω の編集距離が小さければ，ラベル分布も近いことが期待できる，という考えに基づくものである．例えば，ある 2 つのグラフのノードラベルが $\{\mathsf{A}, \mathsf{A}, \mathsf{A}\}$ および $\{\mathsf{B}, \mathsf{B}\}$ であるとする．このとき (エッジの接続関係に関係なく) 少なくともラベル A のノードをを一つ削除し，残りの 2 つのラベル A のノードのラベルを B に変更する必要がある．したがって編集距離はどんなに小さくても 3 である，と見積もることができる．

以上の考え方を一般化する．グラフ g のノードラベルおよびエッジラベルの multiset を $L^{\mathrm{n}}(g)$ および $L^{\mathrm{e}}(g)$ と書くことにする．ここでノードラベル分布 $L^{\mathrm{n}}(g)$ を $L^{\mathrm{n}}(h)$ に変更するには $\chi(A, B) := \max\{|A|, |B|\} - |A \cap B|$ として $\chi(L^{\mathrm{n}}(g), L^{\mathrm{n}}(h))$ 回の編集操作が必要である．エッジラベル分布に関しても同様に $\chi(L^{\mathrm{e}}(g), L^{\mathrm{e}}(h))$ 回の編集操作が必要であり，合計すると編集距離の下限を

$$LB_{\mathrm{label}}(g, h) := \chi(L^{\mathrm{n}}(g), L^{\mathrm{n}}(h)) + \chi(L^{\mathrm{e}}(g), L^{\mathrm{e}}(h)) \quad (6)$$

と見積もることができる．これを用いて $LB_{\mathrm{label}}(g, h) > \tau$ の場合は編集距離を計算することなく $ed(g, h) > \tau$ を確定することができる．

b） DFS の打ち切り (**問題 2**)

もう一つの高速化の戦略として，Algorithm 3 ではすべての子ノードに対する再帰計算を行うが，いくつかの子ノードを枝刈りすることで計算量を削減することが考えられる．あるノード ω においてこのような枝刈りが可能となるためには，ω に対応する誘導部分グラフ g_ω を含むような任意のスーパーグラフ g' (i.e., $g_\omega \subseteq \forall g' \subseteq g$) に対して $ed(h, g') > \tau$ が確定している必要がある．これは上述した $LB(h, g_\omega) > \tau$ よりも強い条件である．このような枝刈りを実現するための下限を $ELB(h, g_\omega)$ と書くこととする [5]．形式的には $ELB(h, g_\omega)$ は以下を満たす関数である．

$$ELB(h, g_\omega) > \tau \implies ed(h, g') > \tau \ (g_\omega \subseteq \forall g' \subseteq g) \quad (7)$$

このような性質を持つ ELB として，本研究ではラベル分布に基づく方法を考える．

命題 1. $\Gamma(A; B) := |A \setminus B| = |A| - |A \cap B|$ とすると

$$ELB_{\mathrm{label}}(h, g_\omega) := \Gamma(L^{\mathrm{n}}(g_\omega); L^{\mathrm{n}}(h)) + \Gamma(L^{\mathrm{e}}(g_\omega); L^{\mathrm{e}}(h))$$

は ELB である (すなわち式 (7) を満たす).

5：ELB は extended lower bound の略.

証明. $ELB_{\mathrm{label}}(h, g_\omega) > \tau$ とし，$\forall g'$ s.t $g_\omega \subseteq g' \subseteq g$ を考える．$L^{\mathrm{n}}(g_\omega) \subseteq L^{\mathrm{n}}(g')$ であるので，$\Gamma(L^{\mathrm{n}}(g_\omega); L^{\mathrm{n}}(h)) \leqq \Gamma(L^{\mathrm{n}}(g'); L^{\mathrm{n}}(h))$ である．明らかに $\Gamma(A; B) \leqq \chi(A, B)$ であるので，$\Gamma(L^{\mathrm{n}}(g_\omega); L^{\mathrm{n}}(h)) \leqq \chi(L^{\mathrm{n}}(g'), L^{\mathrm{n}}(h))$ である．同様に $\Gamma(L^{\mathrm{e}}(g_\omega); L^{\mathrm{e}}(h)) \leqq \chi(L^{\mathrm{e}}(g'), L^{\mathrm{e}}(h))$ が成り立つ．式 (6) より $ed(h, g') \geqq \chi(L^{\mathrm{n}}(g'), L^{\mathrm{n}}(h)) + \chi(L^{\mathrm{e}}(g'), L^{\mathrm{e}}(h))$ であることより，ELB_{label} は 式 (7) を満たす． □

c） Trie の深さ制約への対処 (**問題 3**)

Trie T は深さが k までに制約されていた (4.3 節)．これは検索可能な部分グラフのノード数が k に制限されていることを意味している．Trie T には permutation を格納するので [6]，そのサイズは k に対して爆発的に大きくなる．索引される部分グラフのサイズを k に制限する場合，以下の命題のようにクエリグラフ h のノード数が小さければ厳密な結果を返すことが保証される．

命題 2. $|V(h)| \leqq k - \lfloor \tau/2 \rfloor$ の場合，*Algorithm 3* は $\mathcal{G}$ から $ed(g, h) \leqq \tau$ となる連結な部分グラフを見落としなく発見することができる．

証明. $ed(g, h) \leqq \tau$ となるような g が trie T に索引化されていることを示せばよい．T にはデータベース $\mathcal{G}$ 内のサイズ k 以下のすべての連結部分グラフが列挙されているので，$ed(g, h) \leqq \tau \Rightarrow |V(g)| \leqq k$ であることを示せば十分である．

$$(|V(g)| - |V(h)|) + (|E(g)| - |E(h)|) \leqq LB_{\mathrm{label}}(g, h) \leqq \tau$$

である [7]．これより g のノード数は h よりも高々 $\lfloor \tau/2 \rfloor$ 個多いだけであることが以下のようにしてわかる．まず g は連結であるので h に $\lfloor \tau/2 \rfloor$ 個のノードを追加した場合，同数のエッジも追加する必要がある．もし，g のノード数が h よりも $\lfloor \tau/2 \rfloor + 1$ 個以上多い場合，g を h に変換するには，少なくとも $2\lfloor \tau/2 \rfloor + 2$ 個のノード・エッジが追加操作が必要になり，これは τ よりも大きいため，$ed(g, h) \leqq \tau$ に反するためである．以上より $|V(g)| - |V(h)| \leqq \lfloor \tau/2 \rfloor \leqq k - |V(h)|$，すなわち $|V(g)| \leqq k$ である． □

したがって，最も単純な解決策は k を増やすことである．すでに述べた通り，索引サイズの問題で増やせる k には限界があるが，例えば省スペースな trie を用いるなどの対策も考えられる．他の解決策としては trie の leaf まで到達したときに，さらに DFS する必要があれば，対応する実際のグラフをロードして，各グラフに対してそれぞれ DFS の続きを DfsNaive 関数 (Algorithm 1) を用いて実行する，という方法もありえる．この問題に関しては本研究では深入りせず，今後の課題とする．

6：PCP $\Omega(g)$ は全 permutation $\Omega^+(g)$ の部分集合なので，サイズは一般には $\Omega^+(g)$ よりは小さいが (特に g がスパースな場合)，それでもやはり k に対して指数関数的な増加となる．

7：一般に 2 つの集合 A, B に対して，$\max\{|A|, |B|\} \geqq |A|$ および $|A \cap B| \leqq |B|$ であるので $\chi(A, B) = \max\{|A|, |B|\} - |A \cap B| \geqq |A| - |B|$ が成り立つ．

d） 提案アルゴリズム

ここまでの議論に基づいた提案アルゴリズムを Algorithm 4 に示す．このアルゴリズムは Algorithm 3 と類似しているが，下限 LB_{label} を用いて編集距離 $ed(g_\omega, h)$ の計算を省略する部分 (Line 6)，および ELB_{label} を用いてそれ以上 DFS せずに再帰を打ち切るための条件 (Line 9) が追加されている．

Algorithm 4 Proposed trie-based similarity search

```
1: function SearchTrieProposed(T, h, τ)
2:     (ω, c_ω) ← (T.root, ε)   ▷ Trie root, empty AcGM fragment
3:     DfsTrieModified(ω, c_ω, h, τ)
4: function DfsTrieModified(ω, c_ω, h, τ)
5:     g_ω ← GraphFromAcGM(c_ω)
6:     if LB_label(g_ω, h) ≦ τ then
7:         if ed(g_ω, h) ≦ τ then
8:             output ω
9:     if ELB_label(h, g_ω) ≦ τ then
10:        for all (ω', f) in ω.children do
11:            DfsTrieModified(ω', c_ω f, h, τ)
```

5 評 価 実 験

本章では MUTAG データセット [1] を用いた評価実験を行う．MUTAG は芳香族ニトロ化合物からなるデータセットであり，188 のラベル付き無向グラフからなる．各グラフの平均ノード数は 17.9，平均次数は 2.2，最大次数は 4 である．

MUTAG データセットに対して索引構築を行い，問合せ処理時間や索引構築時間などの計測を行った．Trie の深さパラメータは $k = 9$ をデフォルト値として用いた．

クエリグラフ h は以下の手順で生成した．MUTAG データセットに含まれる 188 個のグラフのそれぞれからノード数 $n = 6$ の連結な部分グラフを以下の手順でランダムサンプリングした．(i) 一様ランダムにノード v を一つ選び，$S = \{v\}$ とする．(ii) S の近傍ノード v を一様ランダムに選び，$S \leftarrow S \cup \{v\}$ とする．(iii) $|S| = n$ ならばノード集合 S に関する誘導部分グラフをクエリグラフ h とする/そうでなければ (ii) に戻る．

実装は C++ (gcc 7.5.0) を用い，`-O3` 最適化を行った．実行環境として Intel Xeon Gold 6142 CPU (2.60 GHz, 64 core), 768 GB RAM のワークステーションを用いた．ただし，実行はシングルスレッドで行った．

5.1 結　　果

[τ **の影響**] 類似度のしきい値を $\tau \in \{1, 2, 3\}$ の範囲で変化させた場合の平均問合せ処理時間を表 1 に示す．問合せグラフのサイズは $|V(h)| = 6$，trie 深さ $k = 9$ であるので，$6 \leqq 9 - \lfloor \tau/2 \rfloor$ が成り立ち，したがって命題 2 よりこれらの設定のもとではアルゴリズムは正しい結果を返す．

表 1 に示した結果より，しきい値 τ が 1 増えると処理時間が数倍程度増えることがわかる．この要因として考えられるのは

- τ の増加により Algorithm 4 の $LB_{\mathrm{label}}(h, g_\omega) \leqq \tau$ (Line 6) を通過するケースが増加し，Line 7 の編集距離計算の

回数が増加すること，

- τ の増加により Algorithm 4 の $ELB_{\mathrm{label}}(h, g_\omega) \leqq \tau$ (Line 9) を通過するケースが増加することで訪問する trie のノード数が増加すること，

の二つである．再帰が深くなると trie のノードの子の数の分だけ再帰呼び出しが増えるため，指数関数的な計算時間の増加となる．これらの計算時間の増加を緩和する手段として，LB や ELB としてよりよいものを選択することが考えられる (今後の課題)．

表 1　問い合わせ処理時間の τ 依存性

τ	1	2	3
Proc. time	188 ms	1042 ms	2885 ms

[$|V(h)|$ の影響] 問合せグラフ h のノード数を $|V(h)| \in \{4,5,6,7\}$ と変化させた場合の平均問合せ処理時間を示す．類似度しきい値は $\tau = 2$ とする．この場合も $|V(h)| \leqq 9 - \lfloor 2/2 \rfloor$ が成り立つので，命題 2 より提案アルゴリズムは正しい結果を返す．表 2 に示した結果より，グラフのノード数の増加とともに問合せ処理時間が増加する傾向が見られる．これはノード数が大きい場合には trie T をより深く探索する必要があるためである，と理解できる．($k = 7$ のケースでは 188 クエリ中およそ 5%にあたる 9 クエリで 30 秒以内に処理が完了しなかったため，平均問い合わせ処理時間の集計からは除外している)

表 2　問い合わせ処理時間の $|V(h)|$ 依存性

$|V(h)|$	4	5	6	7
Proc. time	14 ms	131 ms	1042 ms	10757 ms*

[グラフ数 $|\mathcal{G}|$ の影響] データベース $\mathcal{G}$ に含まれるグラフの数を $|\mathcal{G}| \in \{47, 94, 188\}$ と変化させたときの平均問合せ処理時間を表 3 に示す．

表 3　問い合わせ処理時間の $|\mathcal{G}|$ 依存性

$|\mathcal{G}|$	47	94	188
Proc. time	574 ms	739 ms	1042 ms

[索引サイズ・索引構築時間] 最後に索引サイズおよび索引構築時間を示す．索引サイズとして，trie T のノード数を用いて評価を行った．索引サイズの k 依存性を示したものが表 4 である．Trie T に格納する部分グラフのサイズ k が大きくなると指数関数的に T が大きくなることがわかる．

表 4　Trie T に格納する部分グラフの最大サイズ k と T のノード数の関係 (MUTAG データセット)

k	5	6	7	8	9
#nodes	13,729	102,855	769,824	5,700,114	6,858,094

一方で索引サイズの $|\mathcal{G}|$ 依存性を示したものが表 5 である．グラフ数が 47 から 188 と 4 倍になっても，trie T のノード数の増加は 3 倍程度に抑えられている．これは直接的には AcGM コードの接頭辞がグラフ間で共有されていることを意味しており，すなわち $\mathcal{G}$ 中に同型な部分グラフが多く存在していることを示唆している．一般には同型な部分グラフを多く含むようなデータベースでは trie T のサイズが小さくなることが期待される．

表 5　グラフ DB に含まれるグラフ数 $|\mathcal{G}|$ と T のノード数の関係 (MUTAG データセット) / $k = 8$ のときの結果

$|\mathcal{G}|$	47	94	188
#nodes	1,907,450	3,015,207	5,700,114

[注意] MUTAG データセットは平均次数 (および最大次数) が小さく，PCP がそれほど爆発的に大きくならないようなデータセットである．次数が大きなデータセットへの適用も課題．

6 ま と め

本研究ではグラフ編集距離に基づく類似部分グラフ検索問題について考察した．この問題は本質的に計算量の大きな問題である．いくつかのベースラインアルゴリズムを考え，それらの問題を (部分的に) 解決するアルゴリズムを提案した．提案したアルゴリズムを比較的小規模なグラフデータベースに適用したところ，現実的な計算時間で検索を完了できることが確認できた．一方で，提案手法は索引が巨大になる，という課題があり，より大きなサイズのクエリグラフや大きな類似度しきい値に対応するためにはアルゴリズムおよびデータ構造の改良が必須であり，これらが今後の課題である．

文　献

[1] Asim Kumar Debnath, Rosa L. Lopez de Compadre, Gargi Debnath, Alan J. Shusterman, and Corwin Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry*, 34(2):786–797, 1991.

[2] Dong Deng, Guoliang Li, and Jianhua Feng. A pivotal prefix based filtering algorithm for string similarity search. In *Proc. SIGMOD*, 2014.

[3] Stefan Fankhauser, Kaspar Riesen, and Horst Bunke. Speeding up graph edit distance computation through fast bipartite matching. In Xiaoyi Jiang and Andrea Ferrer, Miquel nd Torsello, editors, *Graph-Based Representations in Pattern Recognition*, pages 102–111. Springer Berlin Heidelberg, 2011.

[4] Chuan Xiao, Wei Wang, and Xuemin Lin. Ed-Join : An Efficient Algorithm for Similarity Joins With Edit Distance Constraints. *Proc. VLDB Endow.*, 1(1):933–944, 2008.

[5] Masataka Yamada and Akihiro Inokuchi. Similar supergraph search based on graph edit distance. *Algorithms*, 14(8), 2021.

[6] Zhiping Zeng, Anthony K. H. Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou. Comparing stars: On approximating graph edit distance. *Proc. VLDB Endow.*, 2009.

[7] Xiang Zhao, Chuan Xiao, Xuemin Lin, and Wei Wang. Efficient graph similarity joins with edit distance constraints. In *Proc. ICDE*, 2012.