決定化されたグラフパターントライの学習アルゴリズム

Graph Classification Based on Graph Pattern Tries

坂上 陽規 [1*]　栗田 和宏 [1]　瀧川 一学 [1]　有村 博紀 [1 †]
Haruki Sakagami[1]　Kazuhiro Kurita[1]　Ichigaku Takigawa[1]　Hiroki Arimura[1]

[1] 北海道大学大学院情報科学研究科
[1] Graduate School of Information Science and Technology, Hokkaido University

Abstract: In this paper, we study efficient learning of graph decision trees from massive graph data without costly complete enumeration of subgraph patterns. We propose the class of graph-fragment decision trees, which is a decision trees whose paths are graph operation codes used in frequent graph mining algorithm gSpan. Then, we prensent an efficient top-down algorithm, called GFDT, euipped with a special unbalanced split rule to discover large graph patterns.

1　はじめに

1.1　背景と研究目的

機械学習においては分類の精度だけでなく，学習した分類規則を人間が解釈が可能であるかという部分も実用において重要視される点の一つである [2].

グラフ分類規則: 本稿では，構造機械学習の代表的な問題の一つであるグラフ分類問題を考察する [3,5,9,11]. 規則の族として，頻出グラフマイニング手法の一つ gSpan [10] で用いられているグラフコードをパターンとして採用し，それらをパスに持つような木であるグラフトライを考察する．このような分類規則は，トライの各パスがただ一つのグラフパターンを表現していることから，人間が規則を解釈しやすいと考えられる．

従来研究: 一方で，グラフパターンを用いてグラフを分類する場合は，gSpan などの頻出部分グラフの列挙なを用いて，事前に分類に用いるグラフパターンを発見することが一般的である [5,9,11]. しかし，網羅的なパターンの発見は，計算量の大きな処理であり，パターンの最大サイズや最小支持度に対して実行時間が指数的に増大しうる．そのため，多様な構造学習応用のボトルネックになっている．

これに対して、本稿では，事前にグラフマイニングアルゴリズムによるグラフパターンの発見を行わず，通常のトップダウン決定木構築に似たプロセスで，グラフトライのトップダウン構築を行いながら，同時に必要なパターンを探索することを考える．

1.2　研究結果

主結果として，初めにグラフトライの長男次弟表現による二分木表現である**グラフ断片決定木**（graph fragment decision trees）の族を提案する．次に，C4.5 や CART に代表されるトップダウン決定木学習方式を拡張して，グラフ断片決定木のトップダウン型学習アルゴリズム GFDT を与える．

GFDT では，gSpan が用いているグラフコードの末尾拡張法 は，通常の決定木のトップダウン構築に自然に組み込まれており，決定木の利得スコアによる分割テストの探索とサンプルの再帰的分割を用いながら，同時に分類に必要なグラフパターンを発見していく．

さらに，通常の決定木構築の利得スコアによるバランスした分割に加えて，大きなグラフパターンに対応する長いパスを見つけるための飽和探索と呼ぶ手法を提案する．これは，ランダムにアンバランスな分割を選択することで，ルール集合発見アルゴリズムにおける貪欲集合被覆のように，パターンを繰り返し追加することを可能にする．

実データ上の実験では，gSpan を前処理に用いたグラフ決定木学習アルゴリズムと比較した小さな最小頻度や大きなパターンサイズなどの現実的なパラメータ設定において，提案手法は比較手法に比べて，同等に近い分類精度を保ちながら，高速に学習を行うことを観察した．

2　準備

本節では，以降で必要な概念と定義を導入する．可変長の要素のリストを $L = [a_1, \ldots, a_n]$，集合を $A = \{a_1, \ldots, a_n\}$ と表す．リスト L と，添え字 $i \leq n$，要素 b に対して，要素 $L[i] = a_i$ および接頭辞 $L[1..i] =$

*連絡先：北海道大学 大学院情報科学研究科
〒 060-0814 札幌市北区北 14 条西 9 丁目
E-mail: Sakagami@ist.hokudai.ac.jp
†連絡先：E-mail: arim@ist.hokudai.ac.jp

$a_1, \ldots, a_i$，連結 $L \cdot b = [a_1, \ldots, a_n, b]$ を定義する．キーと値の組 $a : v$ と書く．ハッシュ表 $H = \{a_1 : v_1, \ldots, a_n : v_n\}$ に対して，キー a_i に対する値を $H[a_i] = v_i$ で表す．リスト等の内包記法を通常通り定義する．以降で説明のない用語については，教科書 [7,8] を参照されたい．

2.1 グラフデータベース

$\mathcal{L} =$ をラベルの可算集合とする．$\mathcal{L}$ 上の**頂点ラベル付きグラフ**は，組 $G = (V(G), E(G), L_G)$ である．ここに，$V(G)$ は頂点の集合であり，$E(G) \subseteq V^2$ は無向辺の集合，$L_G : V \to \mathcal{L}$ は各頂点にラベルを割り当てるラベル関数である．

$\mathcal{G}$ と $\mathcal{Y} = \{0, \ldots, K-1\}$ で，それぞれ，$\mathcal{L}$ 上のラベル付きグラフ全体と**分類ラベル**全体を表す．サイズ m の**グラフデータベース**は，頂点ラベル付きグラフのリスト $GDB = [\, G_i \mid i = 1, \ldots, m \,]$ と分類ラベルのリスト $Y = [\, y_i \mid i = 1, \ldots, m \,]$ の組 $(GDB, Y) \in \mathcal{G}^m \times \mathcal{Y}^m$ である．各 $i = 1, \ldots, m$ に対して，$GDB[i] = G_i$，$Y[i] = y_i$ を，それぞれ，i 番目のグラフと分類ラベルと呼ぶ．頂点集合を $V(GDB) = \bigcup_i V(G)$ とする．GDB の**サンプル**は，任意の添え字集合 $S \subseteq \{1, \ldots, m\}$ である．

2.2 グラフコード

任意の $K \geq 0$ に対し，長さ K の**グラフコード**（graph code，コードとも呼ぶ）とは，リスト $code = [op_1, \ldots, op_k]$ である．ここに，各要素 $op_i \in \mathcal{OP}$ は**グラフ演算子**（graph operator）は，与えられたグラフパターンを変更する操作を表し，以下のいずれかの形をとる．

- OP1 $op = (node_0, node_1, label_1)$: 連結頂点追加演算子は，グラフパターンにラベル $label_1$ を持つノード $node_1$ を加え，既存のノード $node_0$ とノード $node_1$ の間を辺で結ぶ操作を表す．
- OP2 $op = (node_0, node_1)$: 辺追加演算子は，グラフパターン上の既存のノードの対 $node_f$ と $node_t$ の間を辺で結ぶ操作を表す．

コード $code$ **が表現するグラフパターン** $H(G)$ は，空のグラフ $(\emptyset, \emptyset, \emptyset)$ に対し，$code$ 中の各演算子を先頭から順に適用して得られるラベル付きグラフ H である．

例 1 ラベル集合を $\mathcal{L} = \{A, B, C, D, E, F\}$ とする．図 1 にグラフコード $P = [(\perp, 1, A), (1, 2, B), (1, 3, C), (3, 2)]$ と，その接頭辞 $P_1, P_2, P_3, P_4 = P$，それらに対するグラフパターン $G_4 = H(code_4), G_3 = H(code_3), G_2 = H(code_2), G_1 = H(code_1)$ の例を示す．

グラフコード $code, code'$ に対して，$code$ が $code'$ の**接頭辞**ならば，$code \preceq code'$ と書く．ラベル付グラフ H, H' に対して，H が H' の**部分グラフに同形**ならば，$H \sqsubseteq H'$ と書く．二項関係 $\preceq$ と $\sqsubseteq$ はそれぞれ推移的である．今，$code \preceq code'$ と仮定する．このとき，$H(code) \sqsubseteq H(code')$ である．さらに，任意のグラフ G に対して，$H(code') \sqsubseteq G$ ならば $H(code) \sqsubseteq G$ である．

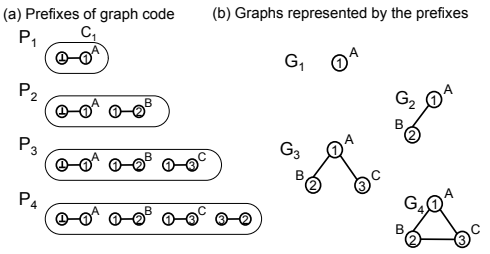

図 1: 例 1 のグラフパターン $G_4 (= G(P_4))$ の接頭辞 $P_i = C_1[1..i]$ と対応するグラフパターン $G_i = G(P_i)$ を示す $(1 \leq i \leq 4)$.

3 グラフトライとグラフ断片決定木

グラフ断片決定木は，図 2 に示したような二分木の形をした決定規則である．これは，これは図 3 に示した辺に，グラフ演算子をもつトライ（trie）[1] を，いわゆる長男次弟表現（leftmost-child right-sibring encoding）をもちいて二分木表現したものである．しかし，その決定規則としての意味はトライとはかなり異なる．

定義 1 (GF 決定木) グラフ断片決定木（graph fragment decision tree，GF 決定木）は，次の条件を満たす完全二分木 T の形をした決定規則である．(i) 根 $root$ は，親を持たない唯一の頂点である．(ii) 各内部頂点 v は 1 子 $v.1$ と 0 子 $v.0$ のちょうど 2 個の子をもち，頂点ラベルとして，グラフ演算子 $op(v) \in \mathcal{I}$ をラベルにもつ．v から 1 子 $v.1$（0 子 $v.0$）へ出る向辺を 1 辺（0 辺）と呼ぶ．(iii) 葉 v は，子をもたない頂点であり，分類ラベル $y(v) \in \mathcal{Y}$ をラベルにもつ．

GF 決定木 T の頂点数 $numnodes(T)$ と，深さ $depth(T)$，葉の数 $size(T)$ を通常通り定義する．図 2 に，グラフ断片決定木の例を示す．今後，図では，1 枝を左下に，0 枝を右横に水平に向けて描く．

GF 決定木のすべての頂点 v に対して，次のように再帰的に，v **のグラフコード** $code(v) \in \mathcal{I}^*$ を対応づける: (i) 根 $root$ には，空列 $code(root) = []$ を対応づける．(ii) もし深さ $k-1$ の内部ノード v にコード $code(v) = [op_1, \ldots, op_{k-1}]$ が対応付けられたならば，深さ k の子である 1 子には $code(v.1) = code(v) \cdot op(v)$ を，0 子には $code(v)$ を対応づける．

定義 2 (予測) GF 決定木 T の**予測アルゴリズム**は，与えられた入力グラフ $G \in \mathcal{G}$ に対して，T の根からスタートし，つぎのように**予測値** $y = \texttt{predict}_T(G) \in \mathcal{Y}$ を出

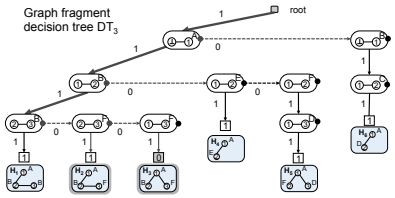

図 2: グラフ断片二分決定木の例

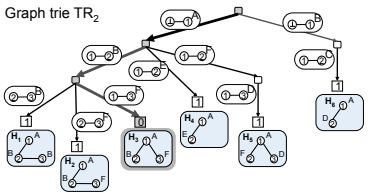

図 3: グラフトライの例

力する：各内部ノード v において，ノードに対応付けられたグラフコード $code(v)$ が定義するグラフパターン $H(code(v))$ を考え，照合条件"$H(code(v)) \sqsubseteq G$?" をテストする．もしテストが成功すれば 1 子へ進み，失敗すれば 0 子へ進む．以上を繰り返して，最後に葉 w に到達したら，その分類ラベル $y = y(w) \in \mathcal{Y}$ を出力する．このとき，G が**葉** w **へ到達した**という．

トライとの違い: GF 決定木は全ての葉たち $w_1, \ldots, w_\ell$ がグラフパターンの列 $H_1, \ldots, H_\ell$ をもつようなトライであるとみなせる．実際，G がある葉 w_i に到達したとき，定義から $H_i = H(code(w_i)) \sqsubseteq G$ が成立する．ただし，GF 決定木は完全二分木なので，非分岐枝（non-branching branch）を直接表すことができない．また，もし非分岐枝を導入した場合には，予測関数 $\texttt{predict}_T$ が部分関数になってしまうので，これを回避するエスケープ予測の扱いが必要になる．

そこで，現在の実装では簡易策として，非分岐ノードを 0 枝が直接に葉をさすような分岐頂点（横路葉, sideway leaf）で表現している．また，過剰学習をさけるために同じ長い非分岐枝に属する横路葉たちには，枝の最上部のノードで定まる同じ出力ラベルを割り付ける．代案として，枝の途中で失敗したら右隣の枝へ「失敗枝」でジャンプする仕組みも考えられる．

3.1 出現リスト

部分グラフ照合 $H(code) \sqsubseteq G$ のテストには，本来，グラフの同形性判定が必要であるが，これは計算困難な問題であり，できるだけ避けたい．そこで，代わりに提案アルゴリズムでは，完全な出現リストを帰納的に更新する方法で，照合条件のテストを行う．GF 決定木におけるパターン照合の再帰的性格から，空列から長

Algorithm 1 学習アルゴリズム GFDT

1: **Algorithm** GFDT$(GDB, minsup, \alpha, \varepsilon)$
2: Input: サイズ m のグラフデータベース GDB, 最小支持度 $minsup \geq 0$，飽和しきい値 $\alpha \in [0, 1]$，利得しきい値 $\varepsilon \geq 0$ の組 $\Theta = (minsup, \alpha, \varepsilon)$
3: Output: 決定木の根ノード $root$
4: $S \leftarrow \{1, \ldots, m\}$
5: $Occ \leftarrow \{\, i : [\,] \mid 1 \leq i \leq m \,\}$
6: **return** RecGFDT$(S, Occ, [\,], GDB, \Theta)$

いグラフコードへ拡張すると同時に，出現リストを更新していくことができる．

グラフ におけるグラフパターン H の**出現リスト**（occurrence list）は，リスト $occlist = [occ_1, \ldots, occ_\ell]$ である．各 $1 \leq i \leq \ell$ に対して，occ_i は，**出現**または**照合**と呼ばれる H の頂点を GDB の頂点を対応づける照合写像 $occ_i : V(H) \to V(G)$ であり，次を満たす: (i) occ_i は 1 対 1 である; (ii) occ_i は頂点ラベルを保存する; (iii) occ_i は辺を保存する．

グラフデータベース GDB における H の**出現リスト表**は，ハッシュ表 $OT = \{\, i : occlist_i \mid i = 1, \ldots m \,\}$ である．ここに，$OT[i] = occlist_i$ は，G_i における H の出現リストである．H から G への全出現を含んでいるとき，出現リスト $occlist$ は**完全**といい，全ての G_i に関して $OT[i]$ が完全なとき，表 OT は**完全**だという．

4 学習アルゴリズム

本節では，グラフ断片決定木の学習アルゴリズム GFDT を提案する．

4.1 基本アルゴリズム

GFDT 学習アルゴリズムは，トライの長男次弟表現による二分決定木表現を利用して，トップダウン型の貪欲探索を用いて，グラフ断片決定木の学習を行う．頻出グラフマイニングアルゴリズム（gSpan 等）を用いたグラフ決定木と異なり，パターンと決定木の探索を融合して，TDIDT 型学習を行うことが特色である．

Algorithm1 にトップレベルの手続き GFDT を，Algorithm2 に主要な再帰手続き RecGFDT を示す．

入力のグラフデータベース GDB と学習パラメーターの組 Θ が与えられると，再帰手続き RecGFDT は，すべてのグラフからなるサンプルをもって，根だけの決定木から計算を開始する．各ノードにおいて，RecGFDT は，(a) **葉生成演算**を行うか，それとも，(b) 内部ノードの下に二つの子を生成する**ノード分割演算**を行うか，(c) アンバランスなサンプル分割を伴う**枝伸長演算**（**飽和演算**とも呼ぶ）を行うかの 3 つの選択を行う．

4.2 3 つの演算

(a) 葉生成演算: **停止条件**として，(a.i) S のサイズが 0 であるか（非空性制約），(a.ii) S の不純度スコアが 0

であるか（不純度制約），(a.iii) S の最良分割での利得が ε 以上であるか（最小利得制約），(a.iv) S の最小支持度がしきい値 $minsup$ 未満であるか（最小支持度制約），のいずれかの条件が満たされたときに，葉の生成が直ちに行われる．分類の場合は，ラベル分布から

$$y_* := \arg\max_{\hat{y}\in Y} \sum_{xy\in S} 1[\hat{y} = y] \tag{1}$$

で定義される多数派ラベルを割り当てる．

(b) ノード分割演算: ノードの分割では，親から受け継いだ $(k-1)$-グラフコード $code$ から，新たに探索したグラフ演算子 op を用いて拡張された k-グラフコード $code_1 = code \cdot op$ を構築する．こに，拡張されたコードが表すパターングラフ $H(code_1)$ によるサンプル S の分割を $S_1 = \{\, i \in S \mid H(code_1) \sqsubseteq GSB[i] \,\}$ と $S_0 = \{\, i \in S \mid H(code_1) \not\sqsubseteq GSB[i] \,\}$ とする．**グラフ演算子 op の探索**は，非負値をとる利得スコア

$$gainScore(S, S_1, S_0) := \tag{2}$$
$$g(S) - (|S_1|/|S|) * g(S_1) + (|S_0|/|S|) * g(S_0)$$

が最大になるように選ぶ．確率ベクター $\mathbf{p} = (p_i)_{i=1}^k$ に対して非負実数値を返す不純度関数には，Gini スコア $gini(\mathbf{p}) := 1 - \sum_i (p_i)^2 \in [0,1]$ を用いる [1]．候補となる演算子 op は，$H(code \cdot op)$ が一つ以上の入力グラフにマッチするような適用可能な全ての演算子の集合 $\mathcal{OP}(S, code)$ から選ぶ．

(c) 枝伸長演算（飽和演算）: ノード分割演算は，バランスした分割を優先する傾向がある．そのため，大きなパターングラフに対応する長い非分岐パスの発見に失敗することが，しばしばある．この改善のため，**飽和演算**（saturated search）と呼ぶノード分割の変種を提案する．飽和演算はノード分割に優先して実行され，与えられた正実数 $r \in [0,1]$ に対して，選択した候補グラフ演算子 op に対して，ノード v でのサンプル分割 S_1, S_0 において，$|S_1| \geq |S|r$ を満たすとき，演算子 op の選択を確定して，それ以上の最適演算子の探索を枝刈りして，直ちに S_1 と S_0 にノード分割を実行する．

4.3 その他の詳細

再帰の非対称性: 通常の TDIDT アルゴリズムと異なる点として，演算 (b) と (c) の再帰は，1 枝側と 0 枝側で非対称である．1 枝側は，op による更新で新たに得られた $code_1$，Occ_1, S_1 を引き渡し，0 枝側は，親から引き継いだ $code$, Occ，照合分を除去した S_0 を引き渡して再帰する．これは，断片二分決定木（図 2）が，グラフトライ（図 3）を長男次弟表現を用いて模倣していることを考えると理解しやすい．この非対称な探索により，GFDT は，ルール集合学習アルゴリズム (例

Algorithm 2 再帰手続き RecGFDT

```
1: Algorithm RecGFDT(S, OT, code, GDB, Θ)
2: Output: 決定木のノード v
3: if (a) の停止条件が成立 then {(a) 葉の生成 }
4:    ŷ ← S の出力ラベル分布で最頻のラベル
5:    return 新しい葉 v
6: end if
7: OP(S, code) ← S に適用可能なグラフ演算全ての集合
8: op_best ← None, Δ_best ← 0
9: for 全ての op ∈ OP に対して do { 演算子の探索 }
10:    S_1, S_0 ← S の code_1 := code · op による分割
11:    if |S_1| ≥ |S|α then {(c) 枝の伸長（飽和演算）}
12:       if |S_0| = 0 then 葉 v.0 を生成し，ラベル付与.
13:       break
14:    else
15:       Δ_op ← gainScore(S, S_1, S_0) {(b) }
16:       if Δ_op > Δ_best then
17:          op_best ← op, Δ_best ← Δ_op
18:       end if
19:    end if
20: end for
21: if Δ_best < ε then {(a) 葉の生成 }
22:    ŷ ← S の出力ラベル分布で最頻のラベル
23:    return 新しい葉 v
24: else {(b) ノードの分割 }
25:    S_1, S_0 ← S の code_1 := code · op による分割
26:    新しい分岐頂点 v を生成する
27:    Occ_1 ← Occ を code_1 で更新
28:    v.1 ← RecGFDT(S_1, Occ_1, code_1, GDB, Θ)
29:    v.0 ← RecGFDT(S_0, Occ, code, GDB, Θ)
30:    return v
31: end if
```

[6]) のように，1 枝側へのパターンの逐次追加による集合被覆戦略を模倣している可能性がある．

枝伸長演算におけるゼロ確率問題: 枝伸長演算で 0 枝側の葉 $v.0$ を生成する際に，$minsup > 0$ の場合，(b) のノード分割演算は 0 枝側 $v.0$ のサンプル S_0 が空となる分割を決して行わないのに対して[2]，(c) の飽和演算では，S_0 が空ならば，伸ばした 1 枝 $v.1$ はすべてのサンプル $S_1 = S$ を受け継ぎ，0 枝 $v.0$ は直ちに (a.i) の非空性制約から伸長を止められて葉となる．

このとき 0 枝側は，空サンプル $S_0 = \emptyset$ を受け取るので，そのままでは葉の出力ラベル y を選択できないという「ゼロ確率問題」が生じる．これには，エスケープ確率（バックアップの事前分布）として， ラベルの一様分布や，親 v のサンプル S から得られた分布に基づいてラベルを選択する方略をとる．

5 実験

4 章で提案した断片決定木の学習アルゴリズム GFDT を実装し，既存手法（頻出グラフ発見をサブルーチンに用いたグラフ決定木）と比較する計算機実験を，実際のグラフデータ上で行った．

[1] 各サンプル S を出力ラベル分布の確率ベクターとみなしている．

[2] その場合，最小支持度制約から v は強制的に葉となる．

表 1: 実験に用いたデータセット

データセット	MUTAG	NCI1	ENZYMES	MSRC9
データ数	188	4110	600	221
クラス数	2	2	6	8
平均頂点数	17.93	29.87	32.63	40.58
平均辺数	19.79	32.30	62.14	97.94
頂点ラベルの種類数	7	36	3	10
最頻クラスの割合 (%)	66.49	50.05	16.67	13.57

5.1 データ

実験に用いたデータセットは，MUTAG, NCI1, ENZYMES, MSRC9 の 4 つのベンチマークデータを用いる．これらは [4] からダウンロードした．表 1 に，各データセットに関する情報をまとめた．辺ラベルの情報を含むデータセットに関しても，辺ラベルの情報を用いなかった．

5.2 実験方法

使用したプログラムは次のとおりである．

- GFDT: 第 4 章で説明した，提案手法である決定化グラフトライを，Python で実装したもの．
- GFDT-NOSAT: GFDT から飽和探索 (saturation) を除いたもの．
- gS-DT: gSpan を用いて列挙した頻出部分グラフ特徴を用いた決定木を，次項の gSpan アルゴリズムと機械学習ライブラリ scikit-kearn[3]を用いて，Python で実装したもの．
- gSpan: 頻出グラフ発見手法 gSpan の gBolt [4]と呼ばれる C++による高速な実装．

アルゴリズムの分類精度の計測は，ランダムな分割による 10 分割交差検定を 10 通り繰り返し，得られたテスト精度の平均値を用いた．実行時間については，GFDT は上記と同じく 10 回の平均値を計測したが，gSpan および gS-DT は時間がかかることと決定性から 1 度だけ計測した．特に指定しなければ，GFDT の飽和しきい値は $\alpha = 1$，$\varepsilon = 0$ を用いた．計算機環境として，PC (CPU Intel Corei7 3.3GHz，Memory 64GB, OS Ubuntu 16.04 LTS) と Python 3.5.2 を用いた．

5.3 実験 1：飽和探索の評価

本実験では，4 章で説明した飽和探索の分類制度に関する効果を調べる．図 4 に，MUTAG と ENZYEMS のデータセットに対して，飽和探索のしきい値 $0 < \alpha \leq 1$ の範囲で変更させて，プログラム GFDT を用いて飽和探索を行う GFDT と行わない GFDT-NOSAT の分類精度を計測した結果のグラフを示した．グラフには，参考として，実験 3 における各データセットでの gS-DT の精度の最良値と，ベースラインの精度を加えた．

[3]scikit-learn.org/
[4]ttps://github.com/Jokeren/DataMining-gSpan

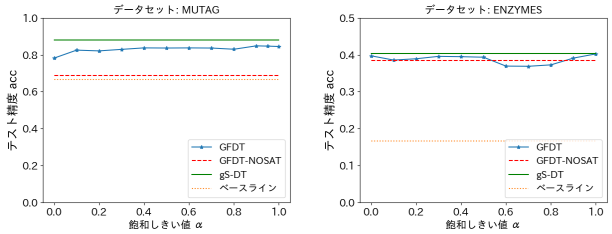

図 4: 実験 1: 飽和しきい値 α に対する分類精度

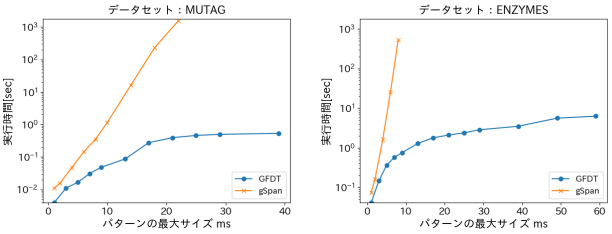

図 5: 実験 2: パターンの最大サイズに対する実行時間（秒）の比較

実験の結果，MUTAG については，GFDT は，GFDT-NOSAT より，精度が約 10%以上高かった．一方 ENZYMES では，GFDT と GFDT-NOSAT で，精度の差異はほとんど見られなかった．飽和しきい値 α について，ノードの演算子のランダムな選択が多くなるような α が小さな値でも，大きな精度の低下が見られなかった．

5.4 実験 2：gSpan の実行時間との比較

図 5 に，MUTAG と ENZYMES に対し，探索対象となるパターンの最大サイズを，MUTAG は $ms = 1 \sim 39$ の間で，ENZYMES は $ms = 1 \sim 59$ の間で変えながら，GFDT と gSpan の実行時間を計測した結果を示す．GFDT では ms は，木の 1 パス長さの上限から 1 引いた値とし，gSpan パターンのサイズはパターンの辺の数で定義した．gSpan では，$minsup = 1$ を用いた．なお，いずれのプログラムも，実行時間が 1800 秒を超えたものは計測しなかった．

結果のグラフから，gSpan の実行時間はパターンサイズ ms に対して指数的に増加しているのに対して，GFDT の方は ms が増加しても，増加率は大きくないことが観察された．これは，gSpan が網羅的にパターン列挙を行うのに対して，提案の GFDT は貪欲法を用いた局所探索であるので高速に動作すると考えられる．

5.5 実験 3：分類精度の評価

表 2 に，MUTAG, NCI1, ENZYMES, MSRC9 のデータセットを用いて，GFDT と gS-DT の分類精度を計測した結果を示す．表では，各データセットに対する精度の最高値を太字で示した．さらに上記の表 2 の実験の補助情報として，この実験に関して，表 3 に gS-DT の最

表 2: 実験 3: GFDT と gS-DT の精度 (%) の比較

	MUTAG	NCI1	ENZYMES	MSRC9
GFDT	84.31	75.62	**40.47**	**91.82**
gS-DT-5	82.27	75.90	40.37	89.60
gS-DT-10	87.76	**79.33**	33.01	22.21
gS-DT-inf	**87.98**	75.67	32.83	22.21
Baseline	66.49	50.05	16.67	13.57

表 3: 実験 3: gS-DT に用いた minsup と gS-DT の実行時間（秒）

	MUTAG		NCI1	
項目	minsup	実行時間	minsup	実行時間
gS-DT-5	1	0.09	1	2.20
gS-DT-10	1	1.24	1	698.86
gS-DT-inf	9	66.06	411	71.42

	ENZYMES		MSRC9	
項目	minsup	実行時間	minsup	実行時間
gS-DT-5	1	6.21	1	27.46
gS-DT-10	420	18.68	110	1.09
gS-DT-inf	420	18.73	110	1.07

小支持度パラメータと実行時間を示し，表 4 に GFDT の実行時間を示した．

実験において，GFDT では，標準のパラメータ設定を用い，MSRC9 に対してのみ $ms = 15$ とした．gS-DT については，前処理の gSpan のパラメータ ms の各値 $\mu \in \{5, 10, \texttt{inf}\}$ を用いたものを，gS-DT-μ と表記した．さらに，パラメータ $minsup$ については，絶対値 $minsup = 1$ および相対値 $minsup = 0.01 \sim 0.9$ の範囲で 11 点を設定し，上限時間 100 秒以内で計算が終了するような最も小さな minsup を一つ選択した．

表 2 のグラフから，精度の面では，ENZYMES と MSRC9 では GFDT の方が高い分類精度を得ており，逆に MUTAG と NCI1 では gS-DT の方が高い精度を示している．前者のデータは最頻クラス割合が低く、辺密度が高いという特性があり、後者は逆であった。

この中で ENZYMES や MSRC9 では、gS-DT はサイズ制限 ms を 10 や無制限にしたときには，最小支持度 $minsup$ を高く設定しなければ，前処理の gSpan を実行できなかった．これに対して，GDFT はこれらのデータセットに対しても高速であり，サイズの大きなパターンを効率的に発見できた。これが GDFT が高い精度を示した理由の可能性がある．

6 おわりに

本稿では，グラフコードを用いたグラフの分類規則である，グラフ断片決定木と，列挙を用いないそのトップダウン学習アルゴリズムを提案した．実験では，いくつかのデータセットについては，gSpan による網羅的なパターン探索を用いた決定木学習と比較して同等以上の精度が得られた．アルゴリズムの高速化やアンサンブル学習への応用は今後の課題である．

表 4: 実験 3: GFDT のサイズ制限と実行時間（秒）

	MUTAG	NCI1	ENZYMES	MSRC9
実行時間（秒）	0.56	52.87	6.27	1.65
パターンサイズ制限 ms	`inf`	`inf`	`inf`	15

謝辞

第一著者の坂上は，本研究に関して北海道大学大規模知識処理研究室の岡崎文哉氏と横山侑政氏に貴重な議論とご教示を頂きました．本研究は，JSPS 科研費 基盤研究 JPA16H01743，挑戦的萌芽研究 JP15K12022 と，北大 GI-CoRE GSB 拠点の助成と支援を部分的に受けました．ここに謝意を表します．

参考文献

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2 edition, 2001.

[2] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics Springer, Berlin, 2001.

[3] Warodom Geamsakul, Tetsuya Yoshida, Kouzou Ohara, Hiroshi Motoda, Hideto Yokoi, and Katsuhiko Takabayashi. Constructing a decision tree for graph-structured data and its applications. *Fundamenta Informaticae*, 66(1-2):131–160, 2005.

[4] Kristian Kersting, Nils M. Kriege, Christopher Morris, Petra Mutzel, and Marion Neumann. Benchmark data sets for graph kernels, 2016.

[5] Taku Kudo, Eisaku Maeda, and Yuji Matsumoto. An application of boosting to graph classification. In *Advances in neural information processing systems*, pages 729–736, 2005.

[6] Bing Liu, Wynne Hsu, and Yiming Ma. Integrating classification and association rule mining. In *Proc. KDD 1998, ACM*, pages 80–86. AAAI Press, 1998.

[7] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012.

[8] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[9] Hiroto Saigo, Sebastian Nowozin, Tadashi Kadowaki, Taku Kudo, and Koji Tsuda. gboost: a mathematical programming approach to graph classification and regression. *Machine Learning*, 75(1):69–89, 2009.

[10] Xifeng Yan and Jiawei Han. gspan: Graph-based substructure pattern mining. In *Proceedings 2002 IEEE International Conference on Data Mining.*, pages 721–724. IEEE, 2002.

[11] 瀧川一学 横山侑政. 全部分グラフ指示子に基づく決定木学習. *SIG-FPAI*, B5(02):75–80, Jan 2016.