

# グラフ断片決定木を用いたグラフ特徴抽出手法

## Graph Feature Extraction Using Graph Fragment Decision Trees

坂上 陽規      瀧川 一学      有村 博紀  
Haruki Sakagami    Ichigaku Takigawa    Hiroki Arimura

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

Graph-fragment decision trees are decision trees for graph classification, whose paths are graph operation codes. The GFDT algorithm is a top-down graph fragment decision tree learner without costly graph enumeration. In this paper, we study GFDT in the view of graph feature extraction. We compared GFDT based methods with pattern enumeration based methods in experiments on real-world datasets.

### 1. はじめに

#### 1.1 背景と研究目的

機械学習を用いたグラフ分類問題について、近年多くの研究がなされている。グラフ分類ではグラフデータの特徴表現としてグラフパターンを用いるのが一般的であり、多くの場合では、学習の前処理として何らかの基準を満たすようなパターンの列挙を行う。例えば、部分グラフ列挙に基づく  $gSpan$  アルゴリズム ( $gS$  と表記する) [4] と決定木構築を合わせた手法 ( $gS-DT$  と表記する) が考えられる。これらの前処理は基本的に重く、多様な構造学習応用のボトルネックになっている。本稿ではグラフ分類問題に対する新しいアプローチとして、グラフ断片決定木を用いた手法を考察する。

**グラフ断片決定木:** グラフ断片決定木 (GF 決定木) [5] は、グラフパターンの表現にグラフの符号化表現であるグラフコード [4] を利用する。決定木の各ノードはテストとしてグラフコードの拡張演算子を持ち、決定木のパスによって一意なグラフパターンに対応したグラフコードが表現される。学習アルゴリズム GFDT では通常の決定木のトップダウン構築法に基づいて決定木の構築を行う。 $gSpan$  [4] のように部分グラフの探索空間の網羅的な探索を行わないため、GFDT は、冒頭で述べた  $gS-DT$  アルゴリズムに比べて、高速かつメモリ使用量が小さいという利点をもつ。一方で、GFDT 単体の分類精度は、多少  $gS-DT$  アルゴリズムに及ばないという問題がある。

#### 1.2 研究結果

そこで、本稿では、GFDT アルゴリズムをグラフ特徴集合発見として用いて、集約学習手法と組み合わせ、より高精度のグラフ学習アルゴリズムを構成することを目指す。**集約学習** [3] (ensemble learning) は、与えられた弱学習器と呼ばれるアルゴリズムを用いて多数の分類規則を生成し、それらによる多数の予測を統合して、より高精度な予測を行う枠組みである。集約学習の例として、ランダムフォレスト (random forest, 以降では RF と表記する) やブースティングがあり、広く利用されている [1]。

具体的には、GF 決定木学習手法 GFDT を元にして、二つのグラフ分類手法を提案する。一つ目に、一般の RF に関する考察から、ランダム化した GF 決定木学習手法 rGFDT を開発し、これを弱学習器として RF と組み合わせた手法 rGFDT-RF を提案する。二つ目に、グラフ特徴抽出手法として、GFDT が

学習した木のパス全体に対応する部分グラフ集合を取り出す手法  $gfPath$  を開発し、これを弱学習器として RF と組み合わせた手法  $gfPath-RF$  を提案する。

実データに対する実験では、これらの提案したグラフ分類の集約学習手法と、従来の  $gSpan$  を特徴抽出器として抽出した部分グラフパターンを弱学習器として RF と組み合わせた手法  $gS-RF$  およびその変種と比較した。

実験結果からは、次のことがわかった。精度に関しては、提案の特徴抽出手法と RF を組み合わせた  $gfPath-RF$  は少ない特徴数で、頻出部分グラフ特徴を用いた  $gS[sup]-RF$  と  $gS[size]-RF$  に近い分類精度が得られた (実験 1 の表 4)。さらに、使用する特徴数の条件を揃えて比較した場合では、 $gfPath-RF$  の分類精度が大きく上回った (実験 2 の表 6)。

計算時間に関しては、GFDT ベースの提案手法の GFDT と、 $rGFDT-RF$ 、 $gfPath-RF$  のいずれも、一部の例外を除き、 $gSpan$  ベースの RF よりも高速であった (実験 2 の表 5)。

まとめると、GFDT を用いたグラフ特徴抽出手法と集約学習の組み合わせは、軽量で比較的高性能なグラフ分類手法として、有用性があると考えられる。

### 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  $L = [a_1, \dots, a_n]$  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  $L[1..i] = a_1, \dots, a_i$ 、連結  $L \cdot b = [a_1, \dots, a_n, b]$  を定義する。キーと値の組  $a : v$  と書く。ハッシュ表  $H = \{a_1 : v_1, \dots, a_n : v_n\}$  に対して、キー  $a_i$  に対する値を  $H[a_i] = v_i$  で表す。以降で説明のない用語については、教科書 [3] 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、 $L_G : V(G) \rightarrow \mathcal{L}$  は各頂点にラベルを割り当てるラベル関数である。頂点付きグラフ  $G'$  が  $G$  の部分グラフに同型であるとき、 $G' \sqsubseteq G$  と書く。

$GRAPH$  と  $LABEL = \{0, \dots, K-1\}$  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  $\mathcal{G} = [G_i \mid i = 1, \dots, m]$  と分類ラベルのリスト  $\mathcal{Y} = [y_i \mid i = 1, \dots, m]$  の組  $(\mathcal{G}, \mathcal{Y})$  である。 $\mathcal{G}$  のサンプルは、任意の添え字集合  $S \subseteq \{1, \dots, m\}$  である。

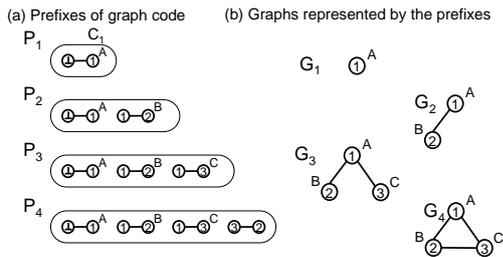


図 1: 例 1 のグラフコード  $P_4$  の接頭辞  $P_i = P[1..i]$  と対応するグラフパターン  $G_i = G(P_i)$  を示す ( $1 \leq i \leq 4$ ).

## 2.2 グラフコード

長さ  $K$  のグラフコード (graph code) とは, リスト  $code = [op_1, \dots, op_K]$  である. ここに, 各要素  $op_i \in \mathcal{I}$  はグラフ演算子 (graph operator) であり, 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  $op = (n_0, n_1, l_1)$ : 連結頂点追加演算子は, グラフパターンにラベル  $l_1$  を持つノード  $n_1$  を加え, 既存のノード  $n_0$  とノード  $n_1$  の間を辺で結ぶ操作を表す.
- OP2  $op = (n_0, n_1)$ : 辺追加演算子は, グラフパターン上の既存のノードの対  $n_0$  と  $n_1$  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  $H(code)$  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頂点ラベル付きグラフである.

例 1 図 1 にグラフコード  $P = [(\perp, 1, A), (1, 2, B), (1, 3, C), (3, 2)]$  と, その接頭辞に対するグラフパターンの例を示す.

## 2.3 ランダムフォレスト

ランダムフォレスト (random forest, RF) [1] は, 複数の弱学習器の決定木で得られた予測を統合して\*, 全体の予測を行う集約学習手法である. RF では, 弱学習器の集合は互いが多様性をもつことが望ましい. そこで, 決定木の場合は次の二つの手法が広く用いられる [1, 3].

**特徴のランダム選択:** 超パラメータを  $\beta \in [0, 1]$  とする. 弱学習器としての決定木構築において, テストとして用いる可能な特徴全体の集合  $\mathcal{OP}$  から,  $\beta$  の割合の部分集合  $\mathcal{OP}'$  をランダムに選択し, テストに用いる特徴を  $\mathcal{OP}'$  の中から選ぶ.

**サブサンプリング:** 弱学習器として決定木を学習する際に, 元の入力訓練データ集合から, 定められたサイズの部分集合をランダムにサンプリングし, これらのデータから学習を行う.

## 3. 提案手法

**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) 内部頂点  $v$  は 1 子  $v.1$  と 0 子  $v.0$  をもち, 頂点ラベルとして, グラフ演算子  $op(v) \in \mathcal{I}$  をもつ. (iii) 葉  $v$  は, 子をもたず, 分類ラベル  $y(v) \in \mathcal{LABEL}$  をラベルにもつ.

図 2 に, GF 決定木と, 入力グラフに対する分類の例を示す.

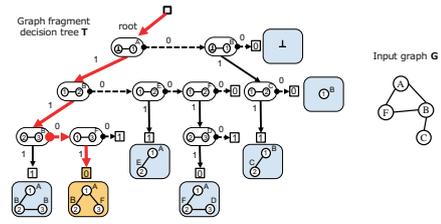


図 2: グラフ断片二分決定木の例

### Algorithm 1 学習アルゴリズム GFDT

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

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  $code(v) = [op_1, \dots, op_{k-1}]$  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  $G \in \mathcal{GRAPH}$  に対して,  $T$  の根からスタートし, 各内部ノード  $v$  において照合条件 " $H(code(v)) \sqsubseteq G?$ " 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. この操作を繰り返し, 到達した葉ラベル  $w$  の持つ分類ラベル  $\hat{y} = y(w)$  を予測値とする. 以後,  $Leaves(T)$  で,  $T$  の葉全体の集合を表す.

**出現リスト:** グラフパターン  $H$  のグラフデータベース  $\mathcal{G}$  におけるマッチングの情報を効率的に保持するために, 出現リストと出現テーブルを導入する.

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  $occlist = [occ_1, \dots, occ_\ell]$  である. 各  $1 \leq i \leq \ell$  に対して,  $occ_i$  は, 出現と呼ばれる  $H$  の各頂点を  $G$  の各頂点に対応づける照合写像  $occ_i : V(H) \rightarrow V(G)$  である.

グラフデータベース  $\mathcal{G}$  における  $H$  の出現リスト表は, ハッシュ表  $OT = \{k : occlist_k \mid k = 1, \dots, m\}$  である.

### 3.1 基本アルゴリズム GFDT

GFDT: GF 決定木の学習は, 貪欲法でのトップダウン構築により行う. Algorithm 1 にトップレベルの手続き GFDT を, Algorithm 2 に主要な再帰手続き RecGFDT を示す.

再起手続き RecGFDT は, 各ノードで (a) 葉生成演算, (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, (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) のいずれかを行う.

(a) 葉生成演算: 停止条件として, (a.i)  $S$  の不純度スコアが 0 であるか, (a.ii)  $S$  の最良分割での利得が  $\varepsilon$  未満であるか, (a.iii)  $S$  の最小支持度がしきい値  $\text{minsup}$  未満であるか, のいずれかの条件が満たされたときに, 葉を生成する. 葉  $v$  の予測ラベルには  $S(v)$  の出力ラベル分布で最頻のラベルを用いる.

(b) ノード分割演算: ノードの分割では, 親から受け継いだグラフコード  $code$  に対し, グラフ演算子  $op$  を用いて拡

\*1 本稿では, 全予測の多数決を用いる.

**Algorithm 2** 再帰手続き RecGFDT

```

1: Algorithm RecGFDT( $S, OT, code, \mathcal{G}, \Theta$ )
2: Output: 決定木のノード  $v$ 
3: if 停止条件が成立 then {(a) 葉生成演算}
4:   return 新しい葉  $v$ 
5: end if
6:  $\mathcal{OP}(S, code) \leftarrow S$  に適用可能なグラフ演算全ての集合
7: 飽和演算 ( $c$ ) により  $\mathcal{OP}(S, code)$  から飽和演算子  $op$  を探す
8: if 飽和演算子が見つからなかったとき then
9:    $\mathcal{OP}(S, code)$  から分割スコア ( $b$ ) が最大の演算子  $op$  を探す
10: end if
11:  $S_1, S_0 \leftarrow S$  の  $code_1 := code \cdot op$  による分割
12:  $OT_1 \leftarrow OT$  を  $code_1$  で更新
13:  $v.1 \leftarrow \text{RecGFDT}(S_1, OT_1, code_1, \mathcal{G}, \Theta)$ 
14:  $v.0 \leftarrow \text{RecGFDT}(S_0, OT, code, \mathcal{G}, \Theta)$ 
15: return 新しい分岐頂点  $v$ 

```

張されたグラフコード  $code_1 = code \cdot op$  を用いて、サンプル  $S$  を  $S_1 = \{i \in S \mid H(code_1) \sqsubseteq GSB[i]\}$  と  $S_0 = \{i \in S \mid H(code_1) \not\sqsubseteq GSB[i]\}$  に分割する。グラフ演算子  $op$  の探索は、利得スコア  $gainScore(S, S_1, S_0) := g(S) - (|S_1|/|S|) * g(S_1) + (|S_0|/|S|) * g(S_0)$  が最大になるように選ぶ。ここに、 $g(S)$  は gini 不順度 [1] である。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) 飽和演算 (枝伸長演算): 飽和演算 (saturated search) では、与えられた正実数  $\alpha \in [0, 1]$  に対して、 $|S_1| \geq \alpha|S|$  を満たす分割を行う演算子  $op \in \mathcal{OP}$  を直ちに選択する。条件を満たすものが複数あれば、ランダムにその中の一つを選択する。飽和演算には長いグラフコードを生成し、パターンの探索範囲を広げる効果が期待される。

**3.2 ランダム化アルゴリズム rGFDT**

本節では、4. 節の実験で RF と組み合わせて用いる弱学習器 rGFDT を説明する。これは、3.1 節の GFDT アルゴリズムを、2.3 節の二つの方法でランダム化したものである。

特徴のランダム選択として、Algorithm 2 の 9 行目において、適用可能なグラフ演算子集合  $\mathcal{OP}$  の代わりに、部分集合  $\mathcal{OP}' \subseteq \mathcal{OP}$  を用いる。ただし、 $\mathcal{OP}'$  は定数  $\beta \in (0, 1]$  に対して  $|\mathcal{OP}'| = \lceil \beta * |\mathcal{OP}| \rceil$  となるように、ランダムに使用する特徴を選択したものをを用いる。サブサンプリングに関しては、2.3 節のサブサンプリング法をそのまま用いる。

**3.3 GFDT を用いたグラフ特徴抽出**

本節では、3.1 節の GFDT アルゴリズムを、グラフデータからの特徴抽出に使う手法 gfPath (graph fragment path) を説明する。GF 決定木の定義から、アルゴリズム GFDT が学習した GF 決定木の任意の葉は、根から葉までのパス (path) に対応するグラフコードを通して、一つの部分グラフパターンを表す。そこで、提案手法 gfPath では、入力の分類ラベル付きグラフデータベース  $(\mathcal{G}, \mathcal{Y})$  から、GFDT を用いて、部分グラフ特徴からなる集合  $\mathcal{H}(\mathcal{G}, \mathcal{Y})$  を次のように抽出する。 $\Theta$  を、GFDT の学習の超パラメータの組とする。

手続き gfPath( $\mathcal{G}, \mathcal{Y}, \Theta$ ):

1. 入力の分類ラベル付きグラフデータベース  $(\mathcal{G}, \mathcal{Y})$  に対し、 $(\mathcal{G}, \mathcal{Y})$  から GF 決定木  $T$  を GFDT で学習する
2. GF 決定木  $T$  の全ての葉  $w \in \text{Leaves}(T)$  が表す部分グラフ  $H(\text{code}(w))$  を特徴として抽出する。以上を繰り返して得られた部分グラフの集合を  $\mathcal{H}(\mathcal{G}, \mathcal{Y})$  として返す。

表 1: 実験に用いたデータセット。

データセット	MUTAG	NC11	ENZYMES
データ数	188	4110	600
クラス数	2	2	6
平均頂点数	17.93	29.87	32.63
平均辺数	19.79	32.30	62.14
頂点ラベルの種類数	7	36	3
最頻クラスの割合 (%)	66.49	50.05	16.67

GFDT アルゴリズムは、gSpan と異なり同型性判定を行っていないので、特徴集合  $\mathcal{H}(\mathcal{G}, \mathcal{Y})$  には互いに同形なグラフパターンが含まれる可能性があり、冗長である。

**4. 実験**

提案の GF 決定木を用いる手法と、既存手法 (頻出グラフ特徴を用いる手法) の比較を、実データ上の実験によって行う。

**4.1 データ**

表 1 に示した MUTAG, NC11, ENZYMES の 3 つのベンチマークデータを、サイト [2] からダウンロードして用いた。

**4.2 実験方法**

実験では、次に示したグラフ特徴抽出器 (グラフの弱学習器) と集約学習手法を組み合わせ、グラフ分類学習器を構成したものをを用いた。以下では、記法  $A-B$  で、グラフに対する弱学習器  $A$  と集約学習手法  $B$  を組み合わせ得られるグラフ分類学習器を表す。

グラフ特徴抽出器は、次の通りである。

- GFDT: 3.1 節のグラフ断片決定木 (GF 決定木) 学習手法の Python による実装。
- rGFDT: 3.2 節の GFDT のランダム化学習手法の Python による実装。
- gfPath: 3.3 節の GF 決定木を用いたグラフ特徴抽出アルゴリズムの Python による実装。
- gS: 頻出グラフ発見手法 gSpan [4] の C++ による高速な実装 gBolt \*2。ただし、並列処理は使用していない。

集約学習手法は次の通りである。

- RF: python の機械学習ライブラリ scikit-learn によるランダムフォレストの実装。

特徴抽出器 gSpan のパラメータである、グラフパターンの最小支持度  $s$  と最大サイズ  $z$  を、それぞれのデータセットごとに表 2 に示した値に設定した。\*3 ここに、 $gS[sup]$  と書かれた群は、 $z$  は制限せず、 $s$  を変化させたものを表し、一方で  $gS[size]$  と書かれた群は、 $s$  は制限せず、 $z$  を変化させたものを表す。

gfPath と rGFDT-RF のパラメータは、飽和しきい閾値を  $\alpha = 1$ 、終了条件しきい値を  $\epsilon = 0$ 、特徴のランダム選択割合を  $\beta = 0.5$  に設定した。集約学習手法について、RF の弱学習器の数は 50 とし、木の深さなどの制限は用いなかった。

分類精度は、異なる 5CV を 5 回繰り返した平均値を計測した。計算機環境として、PC (CPU Intel Corei7 3.3GHz, Memory 64GB, OS Ubuntu 16.04) と Python 3.5.2 を用いた。

\*2 <https://github.com/Jokeren/DataMining-gSpan>

\*3 パラメータ  $s$  および  $z$  は、手作業で、gSpan の計算時間が 200(sec) 以下で生成される特徴が 150 万個を超えないように選択した。

表 2: 実験で使用した gSpan の設定パラメータ.  $s$  と  $z$  は, それぞれ最小支持度と最大サイズを表す.

手法	MUTAG	NCI1	ENZYMES
gS[sup]	$s=5, z=\infty$	$s=205, z=\infty$	$s=360, z=\infty$
gS[size]	$s=1, z=17$	$s=1, z=10$	$s=1, z=7$

表 3: グラフ特徴抽出器による生成した特徴数の比較.

手法	MUTAG	NCI1	ENZYMES
gfPath	38.3	1010.0	237.3
gS[sup]	782816.8	94995.6	276.6
gS[size]	1094054.4	727001.8	106802.8

表 4: 各手法と分類精度の比較.

手法	MUTAG	NCI1	ENZYMES
GFDT	81.10	75.02	39.13
rGFDT-RF	81.30	83.22	53.93
gfPath-RF	84.73	83.08	45.70
gS[sup]-RF	89.28	83.09	37.60
gS[size]-RF	87.04	83.94	51.83
baseline	66.49	50.05	16.67

### 4.3 実験 0: 各手法の分類性能の評価

gSpan と gfPath は, 生成する特徴数に大きな差がある. 予備実験として, 二つのアルゴリズムを各データセットに対して実行し, 生成した特徴数を計測した.

表 3 に得られた結果を示す. 全データセットにおいて, gS[sup] と gS[size] は, 非常に多くの特徴を生成している一方で, gfPath は少数の特徴を生成していることがわかった. これは, gfPath の利点でもあり, 欠点でもある.

### 4.4 実験 1: 各手法の分類性能の評価

グラフ特徴抽出アルゴリズム GFDT, rGFDT, gfPath, gSpan と集約学習手法 RF の組み合わせ手法による分類精度を, 各データセットに対して計測した. ただし, GFDT は RF と組み合わせず, 単一の学習器を用いる. 表 4 に計測した分類精度を, 表 5 に計測した実行時間を示す.

表 4 から, ENZYMES に対しては rGFDT-RF が最も精度が高く, MUTAG に対してはグラフ特徴抽出手法に gSpan を用いた gS[sup]-RF と gS[size]-RF の精度が高かった. gfPath-RF は gS[sup]-RF および gS[size]-RF と比較して全体的にやや精度を下回った. NCI1 では各集約学習手法内で大きな精度の差はなかった. 表 5 からは, 実行時間に関して属性抽出手法 GFDT と rGFDT, gfPath が高速であることが観察される.

### 4.5 実験 2: パターン特徴集合の評価

次に, gSpan で列挙されたグラフ特徴の数を, gfPath と同数に特徴数を削減して実験を行った. 具体的には, gSpan で列挙されたグラフ特徴集合から, gfPath の特徴数との誤差が最小となるような整数  $K$  を選択し, 3 通りの特徴削減手法 (i) 訓練データに対する利得スコアが高い順に  $K$  個選択する方法 (score), (ii) 支持度が高いものから順に  $K$  個選択する方法 (support), (iii) ランダムに  $K$  個選択する方法 (random) を用いた. gSpan の制約パラメータ  $\mu \in \{\text{sup, size}\}$  と特徴選択法  $\nu \in \{\text{score, support, random}\}$  の組み合わせを記法 gS[ $\mu$ - $\nu$ ] で表す.

特徴抽出に gfPath を用いた手法である gfPath-RF と, gSpan の特徴数を削減した手法である gS[ $\mu$ - $\nu$ ]-RF の各データセットに対する精度を比較した. 表 6 に実験結果を示す.

表 5: 各手法の実行時間 (秒) の比較. 括弧内の値は, グラフ特徴発見部分のみの実行時間を示す.

手法	MUTAG	NCI1	ENZYMES
GFDT	0.2	9.0	3.4
rGFDT-RF	10.1	557.6	215.9
gfPath-RF	0.3(0.2)	14.4(13.8)	26.3(26.3)
gS[sup]-RF	129.6(128.1)	222.1(197.4)	142.4(142.2)
gS[size]-RF	118.1(116.7)	123.5(32.8)	112.8(111.1)

表 6: 使用する特徴数を合わせた条件での各手法と分類精度の比較.

手法	MUTAG	NCI1	ENZYMES
gfPath-RF	84.73	83.08	45.70
gS[sup-score]-RF	78.11	73.06	37.17
gS[sup-support]-RF	66.38	80.73	36.93
gS[sup-random]-RF	64.99	79.03	32.83
gS[size-score]-RF	78.11	76.38	41.27
gS[size-support]-RF	66.28	81.21	37.33
gS[size-random]-RF	66.59	72.60	33.17
baseline	66.49	50.05	16.67

実験の結果, gfPath-RF の分類精度が, 全てのデータセットについて最も高かった. この結果から, 少数のグラフ属性しか用いない場合に, gfPath は高い識別能力をもつ特徴集合を発見していると考えられる. これは, 解釈しやすい分類器の学習に有用と思われる.

## 5. おわりに

本稿では, グラフ断片決定木学習手法 GFDT の拡張として, 集約学習器 rGFDT-RF と, グラフ特徴抽出アルゴリズム gfPath, 集約学習器 gfPath-RF を提案し, 実データに対する実験で評価した. 実験結果から, GFDT ベースの手法の, 頻出部分グラフ列挙による手法に対する一定の有用性が観察された. GFDT のグラフ特徴発見手法としての改良は今後の課題である.

## 謝辞

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] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.
- [2] Kristian Kersting, Nils M. Kriege, Christopher Morris, Petra Mutzel, and Marion Neumann. Benchmark data sets for graph kernels, 2016.
- [3] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012.
- [4] 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.
- [5] 坂上 規陽, 栗田 和宏, 瀧川 一学, and 有村 博紀. 決定化されたグラフパターンライの学習アルゴリズム. *SIG-FPAI*, B5(02):63–68, Jan 2018.