第102回人工知能基本問題研究会資料, SIGFPAI, 2016年12月 （2016.11.30改訂2版）

アイテム集合列挙に基づく最適な順序付き決定木の高速発見 *

Fast Discovery of Optimal Ordered Decision Tree Based on Item Set Enumeration

長部 和仁 [1†]　宇野 毅明 [2]　有村 博紀 [1]
Kazuhito Osabe[1], Takeaki Uno[2], Hiroki Arimura[1]

[1] 北海道大学 大学院情報科学研究科
[1] Graduate School of Inf. Sci. & Tech., Hokkaido University
[2] 国立情報学研究所, 情報学プリンシプル研究系
[2] National Institute of Informatics

Abstract: In this paper, we study the problem of finding an optimal decision tree that minimizes the empirical error on an input dataset under constraints such as the maximum size, maximum depth, and minimum frequency of leaves. For this problem, Nijssen と Fromont (DMKD 2010) presented an efficent algorithm $\mathcal{DL}\forall$ based on hash table-based search over a frequent itemset lattice. However, their algorithm requires exponentially large memory to store discovered paths for avoiding duplicates. Thus, it is difficult for the algorithm to handle large data sets. To overcome this problem, we introduce the class of *ordered decision trees* with fixed variable ordering. Then, we present a memory efficient learning algorithm, called ODT, that exactly finds an optimal ordered decision tree under a set of constraints in as large time as DL8 using at most polynomially large memory in the input size. Our algorithm ODT achieved exponential memory reduction by employing depth-first search over the itemset enumeration tree to avoid storing intermediate solutions in a hash table. By experiments, we evaluate the usefulness of our algorithm.

1 はじめに

1.1 研究の背景

近年の機械学習技術の発展と普及を背景として，その応用も大規模化かつ高度化している．とくに，多様で複雑なデータから人間に有用な知識を頑健かつ高速に見つけたいという要求が高まっている．

最近の大規模データからの知識発見では，データから半自動的に発見されたパターンを複合特徴として用いた機械学習手法がふつうになってきた．例えば，バギングやブースティング等の集団学習（ensemble learning）では，このような複合特徴として，決定株（decision stump）や，アイテム集合（itemset），系列パターン，部分グラフなどがよく用いられる．最近の機械学習ツールの一つである XGBoost [1] は，勾配ブースティングに決定木を組合せており，そのような一つの例である．また，深層学習 [2] もある意味ではデータから低次の分類器を生成していると考えられる．また，パターンマイニングから分類学習への接近の一つとして，分類ルール学習やルール集合発見も盛んに研究されている [1,2,4,7]．

複合特徴の発見においては，分類精度に加えて，網羅性や，厳密性，発見した仮説の多様性や，疎性，信頼性の保証が重要となる．また，統計的有意性をもつマイニングや，プライバシー保護マイニングにおいては，与えられた制約をみたす仮説の網羅的な生成と計数が基本的な手続きとして要求されている [6,8]．そこで本研究では，図 1 に示すような決定木の族を対象に，指定された制約の下で，経験誤差を最小化する最適決定木を厳密に見つける問題を考察する．

1.2 既存研究：DL8 アルゴリズム

本研究にもっとも関係する既存研究として，Nijssen と Fromont [5,6] が 2007 年に提案した頻出集合列挙を用いた最適決定木発見アルゴリズム DL8 をとりあげる．

この DL8 アルゴリズムは，決定木のパスと頻出アイテム集合の対応関係に基づいて，一度訪問した頂点を記録するためのハッシュ表を用いて，頻出アイテム集

*本研究は JSPS 科研費基盤 (A)(16H01743), 萌芽研究 (15K12022), 基盤研究 (S)(15H05711) および JST CREST「ビッグデータ統合利活用のための次世代基盤技術の創出・体系化」の助成を受けたものです。

†連絡先：長部和仁、有村博紀、北海道大学情報科学研究科
〒 060-0814 札幌市北区北 14 条西 9 丁目
E-mail: {kz_osabe,arim}@ist.hokudai.ac.jp

[1] https://github.com/dmlc/xgboost

[2] https://www.tensorflow.org

合束上で一種の幅優先探索（実際には表記録型の深さ優先探索）を行う．さらに，木の最大サイズや，最大深さ，葉の最小頻度等の制約を加法性制約として統一的に扱い，効率良い探索を行う．

DL8 の計算時間と使用する領域量は共に $O(MN)$ である．ここに，$N := ||D||$ はデータベースの総サイズであり，M は D 上の頻出アイテム集合の総数である．実験においても，DL8 は元となる頻出アイテム集合アルゴリズムと同等の時間で動作し，いくつかのデータセットでは，C4.5 より精度の高い決定木を発見したと報告されている [6].

DL8 では，変数順序無しの決定木の族 $\mathcal{DT}$ を仮説空間としているため，一つのパス $P_1 = \{x, \overline{y}, z\}$（すなわちアイテム集合）が，$P_2 = \{\overline{y}, x, z\}$ や $P_3 = \{z, \overline{y}, x\}$ のように，最大指数回異なる順序で出現し得る．そのため，既発見のパスを保持するため，集合束全体を保持可能なサイズのハッシュ表が不可欠となる．そのため，入力の指数メモリを必要とするというメモリ効率の問題をもつことが指摘されている [6].

1.3 主結果

そこで本稿では，計算時間とメモリ量の両面で，制約付き最適決定木問題を効率良く解くための理論的性能保証をもつアルゴリズムの研究を行う．はじめに，上記に述べた DL8 の問題点に対応するために，仮説空間としてパスの変数順序を固定した順序付き決定木 (ordered decision tree) の族 $\mathcal{ODT}$ を導入する．そのうえで，族 $\mathcal{ODT}$ に対する最適決定木発見問題を議論する．

主結果として，順序付き決定木の族 $\mathcal{ODT}$ に対するメモリ効率の良い最適決定木発見アルゴリズム ODT を与える．ODT は，入力データベース D と，制約パラメータとして最大サイズ $k \geq 0$ と，葉の最小頻度 $\sigma \in [0..|D|]$ を受け取り，バックトラック型頻出集合発見アルゴリズムが用いるアイテム集合列挙木上で，制約を満たす全ての可能な順序付き決定木の中から，スコア関数を最適化する決定木を発見する．

ODT アルゴリズムは，タプル数 m で総サイズ N の入力データベース D に対して，$O(d(k+m)+k^2)$ 作業領域と $O(MN+Mk^2)$ 計算時間で制約をみたす最適な順序付き決定木を出力する．ここに，$M = |\mathcal{L}_\sigma|$ は頻出アイテム集合の総数である．時間計算量は DL8 と同等である一方で，メモリ量を指数的に改善している．

1.4 本稿の構成

2 節では，順序付き決定木の族 $\mathcal{ODT}$ と最適決定木発見問題を導入する．3 節では，族 $\mathcal{ODT}$ に対する最適な順序決定木発見アルゴリズム ODT を与え，その正当性と計算量解析を議論する．4 節では，ベンチマークデータに対する予備的な評価実験の結果を報告する．5 節では，本稿をまとめ，今後の課題を述べる．なおアルゴリズムと証明の詳細については，手稿 [10] を参照されたい．

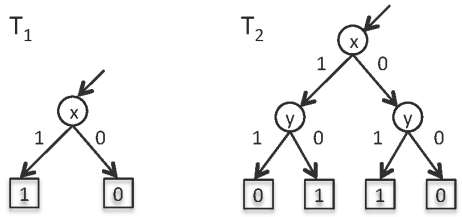

図 1: 決定株 T_1 と順序付き決定木 T_2 の例．ここに，変数順序は $x <_V y$ であり，T_2 は排他的論理和 $x \oplus y$ を表しており，そのパスは左から順に， 4 つの順序付きの拡張アイテム集合（連言）$xy, x\overline{y}, \overline{x}y, \overline{x}\overline{y}$ に対応している．

2 定義

$\mathbb{N} = \{0, 1, 2, \ldots\}$ と $\mathbb{R}$ で、それぞれ、非負整数と実数の全体を表す。任意の実数 $a, b \in \mathbb{R}$ $(a \leq b)$ と非負整数 i, j $(i \leq j)$ に対して、閉区間をそれぞれ $[a, b] := \{\, c \in \mathbb{R} \,|\, a \leq c \leq b\} \subseteq \mathbb{R}$ および $[i..j] := \{i, i+1, \ldots, j\} \subseteq \mathbb{N}$ と定義する。開区間 (a, b) や半開区間を $[a, b)$ などを、通常のとおりに定義する。

2.1 データベースとパターン

任意の正整数を n と m とおく。$V = \{x_1, \ldots, x_n\}$ を n 個の変数からなる変数集合とする。変数順序 (variable ordering) は、変数の添字上の順列 $ord : [1..n] \to [1..n]$ である。ここに、$x_{\mathrm{ord}(1)}$ は第 i 番目の順位の変数であり、ord は $x_{\mathrm{ord}(1)} <_V \cdots <_V x_{\mathrm{ord}(n)}$ のように変数間の全順序 $<_V$ を定める。

データと分類ラベルの全体集合を、それぞれ、$\mathcal{X} = 2^V$ と $\mathcal{Y} = \{0, 1\}$ とおく。各要素 $t \in \mathcal{X}$ は変数の集合であり、タプル (tuple) またはデータと呼ぶ。各要素 $y \in \mathcal{Y}$ は正例または負例を表すブール値であり、分類ラベルと呼ぶ。分類ラベル付きデータベース（またはデータベース）は、組の集合 $D = \{e_1, \ldots, e_m\} \subseteq \mathcal{X} \times \mathcal{Y}$ である。各組 $e = (t, y) \in D$ を分類例（または例）と呼ぶ。以後、変数の数とデータベースのタプル数を、それぞれ、$n = |V|$ と $m = |D|$ で表す。一般に、分類関数（または予測関数）とは、タプルに対して真偽値を返す関数 $\phi : \mathcal{X} \to \mathcal{Y}$ である。

これから具体的な分類関数の族を導入しよう。V 上の論理式であるリテラルとパターンを以下のように定義する。各変数 $x \in V$ に対して、その否定を $\neg x$ で表す。変数とその否定をリテラル（literal）と呼ぶ。集合 $\neg V := \{\neg x \,|\, x \in V\}$ とおくと、$\Sigma := V \cup \neg V$ はリテラルの全体集合である。V 上のサイズ d の拡張アイテム集合

(extended itemset) または (変数順序無し) パターンとは、リテラルの有限集合 $P = \{z_1, \ldots, z_d\} \subseteq (V \cup \overline{V})$ であり、リテラルの論理積 $z_1 \wedge \cdots \wedge z_d$ を表す。拡張アイテム集合は、単にアイテムの全体集合として $\Sigma := V \cup \neg V$ をとったときのアイテム集合 (Σ の部分集合) である。

変数順序 ord に対して、V 上のサイズ d の順序付きパターン (ordered pattern) とは、リテラルの順序列 $P = (z_1, \ldots, z_d) \in (V \cup \overline{V})^*$ で、その変数が変数順序 $<_V$ の昇順 $z_1 <_V, \cdots <_V z_d$ で並んでいるものである。文脈から明らかな時は、順序パターンを非順序パターンと同一視して、$\{x, y\}$ や、$(x) \subseteq (x, y)$ などと書くことがある。以後、$\mathcal{P}_d$ と $\mathcal{OP}_d$ で、それぞれ、V 上のサイズ d の非順序パターンと順序パターンの族を表す。

定義 **1 (**リテラルとパターンの真偽値**)** 任意のタプル $t \in \mathcal{X}$ に対して、t に対するパターン p の真偽値 (または評価値) $\phi_p(t) \in \mathcal{Y} = \{0, 1\}$ を次のように定義する: 変数 $x \in V$ に対して、$\phi_x(t) := 1 \iff x \in t$. 変数の否定 $\neg x \in \neg V$ に対して、$\phi_{\neg x}(t) := \neg \phi_x(t)$. パターン $P = \{z_1, \ldots, z_k\} = z_1 \wedge \cdots \wedge z_k$ に対して、$\phi_P(t) := \phi_{z_1} \wedge \cdots \wedge \phi_{z_k}$. 順序パターンについても無順序パターンと同様に定める。ここに、$\neg 0 := 1$ かつ、$\neg 1 := 0$、$x \wedge y \iff x = y = 1$ と定義する。

データベース D 上に対して、データ $t_i \in D$ の添字 i を TID または単に添字といい、$Tid(D) := [1..m]$ で SD の添字集合を表す。D におけるパターン p の出現リストとは、パターン p の評価値 $\phi_p(t_i)$ が真となる D の組添字の集合 $Occ_D(p) := \{\, i \in [1..m] \,|\, \phi_p(t_i) = 1 \,\}$ である。パターン p の頻度 (frequency) は $freq_D(p) := |Occ_D(p)| \in [0..m]$ である。以後、文脈から明らかならば、D とその添字集合 $Tid(D)$ を区別しない。よって、出現リスト I に対して $Occ_I(p)$ などとも書く。

2.2 決定木の族

本稿では、前ページの図 1 に示すような, 頂点のテストとしてブール変数[3] をもつ決定木を考える。はじめに、変数順序を持たない決定木のクラス $\mathcal{DT}$ を導入する。

定義 **2 (**(順序無し) 決定木**)** 変数集合 V 上の決定木 (decision tree) は、次のように定義される頂点ラベル付き 2 分木 $T = (N(T), E(T), root(T), label_T)$ である。

$N(T)$ は頂点集合であり、$E(T)$ は 1-枝と 0-枝と呼ばれる有向辺の集合である。唯一の入次数 0 の頂点 $root(T) \in N(T)$ は根である。

頂点集合 $N(T)$ は内部頂点と葉からなる。各内部頂点 $v \in N(T)$ は、*1*-枝と *0*-枝と呼ばれる 2 つの有向辺をもち、それぞれ、*1*-子と *0*-子と呼ばれる子 $v.1$ と $v.0$ へと接続する。各葉 $w \in N(T)$ は枝および子をもたない。

[3]頂点テストのブール変数 x は等値制約 "$x = c$" に対応する。一般には、他に不等式制約 "$x \leq c$" がある。またテストとして、ブール変数の否定 $\neg x$ は 0 枝と 1 枝を入れ替えて表せる。

関数 $label_T = test_T \cup class_T$ は、各頂点にラベルを対応づけるラベル関数である。各内部頂点 v に対して、$label_T(v) = test_T(v)$ は変数 $x \in V$ を返す。各葉 w に対して、$label_T(\ell) = class_T(\ell) = c$ は分類ラベル $c \in \mathcal{Y}$ を返す。

決定木 T に対して、そのサイズを T の総頂点数 $k(T)$ と定義し、その深さを最長のパスの長さ $d(T)$ (辺数で数える)、葉数を T に含まれる葉の総数 $l(T)$ と定める。T は完全二分木なので、常に $k(T) = 2l(T) - 1$ となる。$T.1$ と $T.0$ で、それぞれ、根 $root(V)$ の 1 子と 0 子を根とする T の部分木を表し、T の *1* 木と *0* 木と呼ぶ。

以後、V 上の決定木 (decision tree, DT) の族を、$\mathcal{DT} = \mathcal{DT}^V$ で表す。任意の部分族 $\mathcal{C}^V \subseteq \mathcal{DT}^V$ に対して、$\mathcal{C}^V_{k,d,\sigma} \subseteq \mathcal{C}$ で、サイズが k 以下で、深さ d 以下、葉の最小頻度が σ 以上となる $\mathcal{C}$ に属する決定木の族を表す。決定木はタプル上のブール関数を表す。

定義 **3 (**決定木が定義する分類関数**)** 決定木 T が定める分類関数 (または予測関数) はブール関数 $\phi_T : \mathcal{X} \to \mathcal{Y}$ であり、任意のタプル $t \in \mathcal{X}$ に対して $\phi_T(t) := \psi_T(root(T), t)$ と定義される。ここに、任意の頂点 $v \in N(T)$ に対して、$\psi_T(v, t)$ の値は次のように再帰的に定義される:

$$\psi_T(v,t) = \begin{cases} \ell, & \text{if } v \text{ は葉,} \\ \psi_T(v.1, t), & \text{if } v \text{ は内部頂点かつ } x \in t, \\ \psi_T(v.0, t), & \text{if } v \text{ は内部頂点かつ } x \notin t, \end{cases} \tag{1}$$

ここに、内部頂点 v に対して $x := test(v)$ は V の変数であり、葉 v に対して $\ell := class(v) \in \mathcal{Y}$ はクラスラベルである。上記の評価において, ある頂点 v で式 $\psi_T(v, t)$ が評価された場合に, タプル t は葉 v へ到達するという。

例 **1**. 図 1 に, 例として, 深さ 1 でサイズ 3 の決定木 (決定株) T_1 と深さ 2 でサイズ 7 の決定木 T_2 を示す. T_1 は変数 x を表し, T_2 は排他的論理和 $x \oplus y$ を表す.

$D = \{e_1, \ldots, e_m\} \subseteq \mathcal{X} \times \mathcal{Y}$ をデータベースとする。D における決定木 T の頂点 v の頻度とは、v へ到達する D 中のタプルの総数 $\sigma_D(v) \in [1..m]$ をいう。D における T の葉の最小頻度を、全ての葉に対する頻度の最小値 $\sigma_D(T) := \min_{v:T \text{ の葉}} \sigma_D(v) \in [0..m]$ と定義する。

2.3 分類スコア

本節では決定木 T の分類スコアを導入する [3]。ここに、n 個の例を含むデータベース D を仮定する。本小節のみ、例の数が n なので注意されたい。各例 $e = (x, y) \in D$ に対して分類ラベル $y \in \mathcal{Y}$ と決定木 T による予測値 $\hat{y} := \phi_T(t) \in \mathcal{Y}$ を考えて、次のように非負整数 $n_1, n_0, m_1, m_0 \in \mathbb{N}$ を定める: (i) 非負整数 n_1 (または n_0) は、正例の数 (または、負例の数) である。(ii)

非負整数 m_1（または m_0）は、決定木による予測値が 1 となる正例の数（または、負例の数）である。ここに、$n = n_1 + n_0$ である．また，予測値が 0 となる正例と負例の数は，それぞれ $n_1 - m_1$ と $n_0 - m_0$ となる．以下では、任意の分類関数を $\phi : \mathcal{X} \to \mathcal{Y}$ とおく。

例の上の任意の同時分布 $\mathcal{D} : \mathcal{X} \times \mathcal{Y} \to [0, 1]$ における ϕ の誤分類確率を，分類関数 ϕ の**真の誤差**という．ϕ の**真の精度**は，1 から真の誤差を引いた値である．

定義 4 （経験誤差と精度）サイズ $n \geq 0$ の任意のデータベース $D \subseteq \mathcal{X} \times \mathcal{Y}$ に対して、分類関数 ϕ の D における誤分類数を、ϕ が誤分類した例の個数 $\#Err(\phi_T, D) := m_0 + n_1 - m_1 \in [0..n]$ とおき、D における ϕ の経験誤差 (empirical error) を、ϕ が誤分類数の比率

$$Err_n(\phi_T; D) := \frac{\#Err(\phi_T, D)}{n} \in [0, 1] \qquad (2)$$

と定める。さらに、D における ϕ の経験精度 (accuracy) を $Acc_n(\phi_T; D) := 1 - Err_n(\phi_T; D) \in [0, 1]$ とおく。

仮説空間とは分類関数の族 $\mathcal{H} = \{\phi_0, \phi_1, \ldots\}$ である。制約付き決定木の族 $\mathcal{DT}_{k,d,\sigma}$ は仮説空間の一例である。

仮説空間 $\mathcal{H}$ が複雑すぎないとき[4]、学習アルゴリズムはデータ集合上で経験誤差を最小化することで，高い確率で真の誤差を小さくできることが知られている [3]。

2.4 順序付き決定木の族

上記の準備のもとに、順序付き決定木の族 $\mathcal{ODT}$ を導入する。変数順序は、変数の添字の順列 ord であり、V 上の全順序 $<_V$ を定めることを思い出そう。

定義 5 決定木 T が**順序付き** (ordered) であるとは、V 上のある変数順序 ord が存在して、T の根から任意の葉までの任意のパス上の変数が全順序 $<_V$ の昇順で並んでいることをいう。

以後、$\mathcal{ODT}^{V,\mathrm{ord}}$ で、V 上の変数順序 ord にしたがう順序付き決定木 (ordered decision tree, ODT) の族を表す。定義より、任意の ord に対して $\mathcal{ODT}^{V,\mathrm{ord}} \subseteq \mathcal{DT}^V$ である。文脈より明らかな時には添字 V と ord を略する。サイズと深さの制約の下で異なる決定木の個数の上限について、次の定理を示す。

補題 1 $|V| = n$ とする。任意のサイズ $k \geq 1$ と深さ $0 \leq d \leq k$ に対して、$|\mathcal{DT}^V_{k,d,*}| = O(d^{k/2}(2nd)^{dk/2})$ と $|\mathcal{ODT}^{V,ord}_{k,d,*}| = O(d^{k/2}(2n)^{dk/2})$ が成立する。

証明: $\mathcal{DT}$ の異なるパスが非順序パターンに対応することと，$\mathcal{ODT}$ の異なるパスが順序パターンに対応すること，任意の決定木 T は、高々$l(T) \leq \lceil k(T)/2 \rceil$ 個の異なるパスを選んで作れることから示される． □

[4]例えば、仮説空間の VC 次元 [3]VCdim($\mathcal{H}$) が入力パラメータの多項式のときはこの場合である。

Algorithm 1: 変数集合 V と，変数順序 ord，分類ラベル付きのデータベース $D = \{e_1, \ldots, e_m\} \subseteq \mathcal{X} \times \mathcal{Y}$ から、葉の最小頻度 $\sigma_{\min} \in [0..m]$ と，最大サイズ $k_{\max} \geq 0$、最大深さ $0 \leq d_{\max} \leq k_{\max}$ の制約を満たし、経験誤差を最小化する最適な順序付き決定木を発見するアルゴリズム ODT.

```
1 Algorithm ODT(V, ord, D, σ_min, k_max, d_max);
2 Output: 最適木の根へのポインタ root と、そのサイズ k と経験誤差 err からなる三つ組 τ_opt.;
3 begin
4     Θ := (σ_min, k_max, d_max, m := |D|, n := |V|, ord, V, D);
5     X_0 := ∅; Tid(D) := [1..m]; tail_0 := 0, d_0 := 0;
6     Opts := RecODT(X_0, Tid(D), tail_0, d_0, Θ);
7     return τ_opt := arg min_{τ∈Opts} τ.err;
```

補題 1 と，分類関数の有限族 $\mathcal{H}$ の VC 次元 $\mathrm{VCdim}(\mathcal{H})$ の上限が $O(\log |\mathcal{H}|)$ で与えられることから [3]，深さ d とサイズ k を固定した族 $\mathcal{ODT}_{k,d,*}$ の VC 次元は $n = |V|$ の多項式だが，サイズが $k = O(2^d)$ なので，サイズが任意の族 $\mathcal{ODT}_{*,d,*}$ の VC 次元は n の多項式では抑えられない．

2.5 データマイニング問題

本稿で考察する問題は次のとおりである。

定義 6 (制約付き最適順序決定木発見問題) 順序付き決定木の族 $\mathcal{ODT}$ に対して、入力として、変数集合 $V = \{x_1, \ldots, x_n\}$ $(n \geq 1)$ と変数順序 $\leq_V$、V 上のデータベース $D = \{e_1, \ldots, e_m\} \subseteq \mathcal{X} \times \mathcal{Y}$ $(m \geq 1)$、制約パラメータとして最大サイズ $k \geq 0$ と、葉の最小頻度 $\sigma \in [0..m]$ が与えられたとき、与えられた制約を満たす順序付き決定木 $T \in \mathcal{ODT}_{k,d,\sigma}$ すべての中で、経験誤差 $Err_n(T; D)$ を最小化する順序付き決定木 $T_{\min}$ を出力せよ。

上記で，最小経験誤差を達成する木が複数ある場合は、どれか一つを出力すれば良い。

3 提案手法

本節では、順序付き決定木の族 $\mathcal{ODT}$ に対して、DL8 と同一の時間計算量と，入力の多項式領域で、制約を満たす最適決定木を厳密に計算する学習アルゴリズム ODT を提案する。

3.1 アルゴリズムの概要

図 1 に最適な順序決定木を計算する提案アルゴリズム ODT を示し、図 2 にその再帰手続き RecODT を示す。

手続き RecODT は，入力を受け取ると，根となる空アイテム集合 $\emptyset$ から，トップダウンにパスとなる拡張アイテム集合を構築しながら、再帰的にアイテム集合束 $\mathcal{F}$ 上を探索し、動的計画法を用いてボトムアップに最適決定木を求めていく。

各繰り返しでは、手続き RecODT は，現在のパス（＝アイテム集合）X と，その出現リスト Occ，X 中の最大の変数添字 $tail$ を親から受け取り，Occ から二つの子供のための出現リスト Occ_1 と Occ_0 を計算した後に，自分自身をそれぞれに対して再帰的に呼び出す．

親へバックトラックする際には，各サイズ k ごとに，ボトムアップに求めた最適木の集合の情報 $Opts$ を返す（後述）．

ODT は，変数順序 ord から、各パスについてアイテム列としての一意な正規形を仮定できる。そのため、$DL8$ が用いているパスを記録するハッシュ表が不要である。これにより、DL8 の入力サイズに指数的なメモリが不要になる。

3.2 制約を用いた探索の枝刈り

DL9 では、アイテム集合の列挙木上の探索において、トップダウンおよびボトムアップの両方向で枝刈りを行う。根からのトップダウンな計算における葉の最小頻度 $\sigma_{\min}$ の制約については，次が成立する．データベース D における v の頻度 $\sigma_D(v)$ は，すなわち，v に対応づけられた出現リストの長さであることを思い出そう．

補題 2 D を任意のデータベース，T を決定木とし，v を T の任意の内部頂点 v と v を根とする部分木の任意の葉 w に対して，$\sigma_D(v) \geq \sigma_D(w)$ が成立する．

上の補題より，RecODT の繰り返しにおいて，親から受け取った出現リスト Occ の長さが最小頻度 $\sigma_{\min}$ を真に下回ったときには，その子孫の探索をすべて枝刈りしてよいことがわかる．

葉からのボトムアップな計算における最大サイズ $k_{\max}$ と最大深さ $d_{\max}$ の制約については，次が成立する．

補題 3 T を任意のサイズ $k > 1$ の決定木とする．このとき，$size(T) = 1 + size(T.1) + size(T.0) \leq k$ が成立する．さらに，$size(T) \leq k$ ならば，$size(T.i) \leq k - 2$ $(i = 1, 0)$ が成立する．

3.3 最適化プロファイルの計算

提案アルゴリズムでは、現在の繰り返しにおいて、サイズ毎の最適決定木のリスト $Opts := (T^*[1], \ldots, T^*[k_{\max}])$ を求める．これを最適木プロファイルと呼ぶ．定義より、$Opts$ の中で経験誤差が最小のものが、I に関する最適木である。以下では、$k = 1, \ldots, k_{\max}$ に関する帰納法を用いて、最適木プロファイルを特徴付ける。T を任意の決定木とする。また、$|I| \geq \sigma$ と仮定する。

Algorithm 2: パス X と対応する出現リスト I を受け取り、最大サイズ $k_{\max} \geq 0$ と、最大深さ $d_{\max}$、葉の最小頻度 $\sigma_{\min} \in [0..m]$ の制約を満たし、変数順序 ord のもとで、最適決定木プロファイル $Opts = \{\tau_i := (k_i, err_i, v_i)\}_{i=1}^{k_{\max}}$ を計算する再帰的手続き RecODT．ここに、$tail$ は X 中の ord で最大の変数の添字．

```
1  Procedure RecODT(X, Occ, tail, d, Θ);
2  begin
       /* Step1: サイズ k = 1 の最適木を見つける */
3      (σ_min, k_max, d_max, m, n, ord, V, D) := Θ;
4      ℓ_1 := arg max_{ℓ∈𝔏} |I_ℓ|; err_1 := |I| − |I_{ℓ_1}|;
6      Opts := ∅; Opts[1] := (1, err_1, ℓ_1);
7      if d + 1 > d_max then return Opts;
       /* Step2: サイズ k > 1 の最適木を見つける */
8      for i := tail + 1, ..., n do
9          Occ を，変数 x_{ord(i)} のタプルでの評価値が
             1 か 0 かにより，二つの部分リスト Occ_0
             と Occ_1 に分割する．;
10         if (|Occ_1| ≥ σ_min) and (|Occ_0| ≥ σ_min)
             then
11             Opts_1 := RecODT(X∪{x_{ord(i)}}, Occ_1,
                 i, d + 1, Θ);
12             Opts_0 := RecODT(X∪{¬x_{ord(i)}},
                 Occ_0, i, d + 1, Θ);
13             foreach τ_1 ∈ Opts_1 and τ_0 ∈ Opts_0
                 with τ_1.k + τ_0.k < k_max do
14                 τ := a new optimal tree triple;
15                 τ.k := 1 + τ_1.k + τ_0.k;
16                 τ.err := τ_1.err + τ_0.err;
17                 if (Opts[k] = null) or
                     (τ.err < Opts[k].err) then
18                     if (Opts[k] ≠ null) then
19                         DecNodeRefCount(Opts[k].root);
20                     IncNodeRefCount(w_1);
21                     IncNodeRefCount(w_0);
22                     τ.root := a new node v with
                         v.test := x_{ord(i)}, v.1 := w_1,
                         and v.0 := w_0;
23                     Opts[k] := τ;
24             foreach τ ∈ (Opts_1 ∪ Opts_0) do
25                 DecNodeRefCount(τ.root);
26     return Opts;
```

基底ステップ. 初めに、T が葉だけからなるサイズ 1 の木の場合を考えよう。このとき、出現リスト I 中の多数ラベルを $\ell_{MAJ} := \arg\min_{\ell \in \mathcal{Y}} |I_\ell|$ とおく。ここに、$I_\ell := \{\, i \in I \,|\, e_i = (x, y), y = \ell \,\}$ はラベル $\ell \in \mathcal{Y}$ をもつ例の集合である。

補題 **4** 上記のラベル ℓ_{MAJ} をもつ葉だけからなる木は、すべてのサイズ *1* の木の中で I 上の最小経験誤差を与える最適木である。

帰納ステップ. T がサイズ $k > 1$ の場合に，T は変数 x をもつ根 $root(T)$ と、1-木 $T.1$、0-木 $T.0$ からなる. ここで，親からもらった出現リスト Occ を x で分割して得られる出現リストを Occ_1, Occ_0 として，最適木プロファイル $Opts_1$ と $Opts_0$ が ODT の再帰計算により求められていると仮定する。

ここで、$Opts_1$ と $Opts_0$ のそれぞれの木を、1 木と 0 木として組合せ、これに変数 x を頂点ラベルとして根に追加して得られる順序決定木のうちで、サイズが高々 $k_{\max}$ の順序決定木の全体 $CAND(x) \subseteq \mathcal{ODT}_{k_{\max}, d_{\max}, \sigma_{\min}}$ を考える。すなわち、任意の順序決定木 T に対して、

$$
\begin{aligned}
&T := (x, T_1, T_0) \in CAND(x) \iff \\
&\text{(i) } (T_1, T_0) \in Opts_1 \times Opts_0\text{、かつ} \\
&(ii) 1 + size(T_1) + size(T_0) \leq k
\end{aligned}
$$

である。このとき，木の集合 $\cup_{x \in V} CAND(x)$ 中でスコアを最小にする木をサイズ制約を満たす最適決定木として出力すれば良い.

3.4 決定木候補のメモリ管理

手続き RecODT は，1 枝と 0 枝で指されたポインタ構造体として，葉から順にボトムアップに部分的な決定木を構築していく．繰り返しが完了するまでに，手続きは高々 $k_{\max}$ 個の部分木を選択して，最適木プロファイルに登録する．もしこのときに選択されなかった部分木を放置すると，最終的に指数的なメモリを消費してしまう.

そこで，手続き RecODT の各繰り返しで親にバックトラックする前に，$Opts_1 \cup Opts_0$ に含まれる部分木で選択されなかったものすべてについて，参照ポインタを用いてそれらの頂点を削除してメモリを回収する．手続き IncNodeRefCount は参照カウンタを 1 つ増やし，手続き DecNodeRefCount は参照カウンタを一つ減らし，値が 0 になったらそのオブジェクトを除去する．これらの処理は，図 2 に示した手続き RecODT の 19–21 行目と 25 行目で行われる.

3.5 アルゴリズムの正当性

以上の RecODT による最適決定木の再帰的計算に関して、次の補題が成立する。

補題 **5 (**ODT の正当性) 族 $\mathcal{ODT}_{k,d,\sigma}$ に所属するサイズがちょうど k の I 上の最適順序決定木 T^* に対して、次の等式が成立する：

$$
Err(T^*, I) = \min_{x \in V} \min_{\substack{T' \in CAND(x) \\ size(T') = k}} Err(T', I). \quad (3)
$$

証明: 補題の等式の右辺が与える木を $\hat{T}$ とおく。T^* の 1 木 T_1^* と 0 木 T_0^* が，それぞれ $Opts_1$ と $Opts_0$ に含まれることを示す（主張 1)。もし二つの子のどちらかが最適でないならば、T^* 中でその子を、より誤差の小さい木で置き換えると、得られた木の誤差は元の木 T^* よりも小さくなり矛盾するので，主張 1 は成り立つ。このことより、$test(root(T)) = x$ のとき、$T^* \in CAND(x)$ が言える（主張 2)。一方で、任意の木 $T \in CAND(x)$ は正しく $\mathcal{ODT}^V_{k,d,\sigma}$ の決定木であることが言える。全ての $x \in V$ に対し、$\hat{T}$ は $CAND(x)$ 中の最小誤差の木なので、主張 2 より $Err(\hat{T}, I) \leq Err(T^*, I)$ が言える。反対に、T^* は $\mathcal{ODT}^V_{k,d,\sigma}$ 中で最小誤差を与え、$CAND(x) \subseteq \mathcal{ODT}^V_{k,d,\sigma}$ から、$Err(T^*, I) \leq Err(\hat{T}, I)$ である。よって，結果が示された。 □

(3) 全体. 提案アルゴリズムは、補題 5 中の式（3）の右辺で最小値を与える決定木 $\hat{T}$ を各サイズ毎に求めることで、(2) の帰納ステップにおけるサイズ > 1 の最適木を求める。これと、(1) の基底ステップの葉だけからなるサイズ 1 の最適木を合わせて、最適木プロファイルを計算する。

3.6 アルゴリズムの計算量

本小節では，アルゴリズム ODT の計算量を解析する. ODT は、入力データベース D と、制約パラメータとして最大サイズ $k \geq 0$ と、葉の最小頻度 $\sigma \in [0..|D|]$ を受け取り、バックトラック型頻出集合発見アルゴリズムが用いるアイテム集合列挙木上で、制約を満たす全ての可能な決定木の中でスコア関数を最適化する決定木を網羅的に探索する。

定理 **1 (**ODT の正当性と計算量) 順序付き決定木の族に対して、図 1 のアルゴリズム ODT は、変数集合 V と，変数順序 ord、入力データベース D、制約パラメータとして最大サイズ $k \geq 0$ と、葉の最小頻度 $\sigma \in [0..|D|]$ を受け取り、与えられた制約を満たす全ての可能な順序付き決定木の中で、相対誤差を最小化する順序付き決定木 $T_{\text{opt}} \in \mathcal{ODT}^V_{k,d,\sigma}$ を、$O(d(k+m) + k^2) = poly(k, m)$ 作業領域と $O(MN + Mk^2) = O(M \cdot poly(k, m, n))$ 計算時間で出力する。ここに、$n = |V|$ は変数の個数であり，$m = |D|$ は入力タプル数，$N = O(mn)$ は D の総サイズであり、M は頻出アイテム集合の総数である。

定理 1 より，ODT の計算時間は DL8 とほぼ同じである一方で，メモリ使用量は入力サイズの多項式メモリ量を達成し，大幅な改善を行なっている.

表 1: 実験に用いたデータセットの一覧

name	num tuples	classes
chess	3196	2
g-credit	1000	2
mushroom	8124	2
vote	435	2

4 実験

4.1 データと方法

データセットには、Constraint Programming for Itemset Mining Datasets [5] で公開されているものを用いた。これらのデータセットは、UCI machine learning repository [6] で公開されているデータセットを頻出アイテム集合マイニング用に離散化したものである。

表 1 に使用したデータセットとその特徴の一覧を示す。ここで、name はデータセット名であり、num tuples はサイズ（タプル数）、classes は目標属性の数である。

使用したプログラムは次のとおりである。

- ODT: 3 節のアルゴリズム ODT を C++で実装したもの。最適化手法として、3 節に述べた Occurrence deliver と参照カウントを用いたメモリ回収を組み込んだ。
- DL8: 実装が公開されていないため、分類精度についてのみ、文献 [5, 6] に記載されている同一のデータセットにおける実験から、記載の分類精度を使用して比較した。
- LCM basic: 計算時間の参考に、ODT の元となったバックトラック型の頻出集合発見アルゴリズムを C++で実装したもの。これは、LCM [9] から閉包計算をのぞいて、頻出集合発見するものと同等である。実験では，Uno による実装 (uno)[7] と，今回 ODT と共通に新たに実装したもの (ours) の両方を用いた．

実験環境は、PC（CPU Intel Core i7 3.3GHz, Memory 64GB, OS Ubuntu 14.04），コンパイラ g++ ver4.8.4 を用いた。実験では、訓練データ集合上でプログラムを同一パラメータで 1 回ずつ走らせて、計算時間と発見した決定木の精度を計測した。

4.2 実験 1: 発見した最適決定木の例

図 2 に、ODT が発見した最適決定木の例を示す。データセットは mushroom を用いて、「毒性の有無」の属性を分類ラベルとして、最大サイズ $K = 20$、葉の最小頻度 $\sigma = 1200$ として、サイズが $k = 1 \sim 20$ それぞれの最適決定木を計算時間約 14 秒で出力した。図の各行は左からサイズ k，経験精度，変数を番号で，ラベルを [0],[1] で表した木の項表現である．結果として、精度 $acc = 0.970458$ でサイズ $k = 9$ の最適決定木が得られた。

[5]https://dtai.cs.kuleuven.be/CP4IM/datasets/
[6]http://archive.ics.uci.edu/ml/
[7]http://research.nii.ac.jp/ uno/codes.htm

表 2: データセット mushroom から，サイズ $k = 20$ と最小頻度 $\sigma = 1200$ で ODT が発見した最適決定木の例

```
 1, 0.5179, [0]
 3, 0.8867, (38 [0] [1])
 5, 0.9512, (66 [1] (25 [0] [1]))
 7, 0.9581, (66 [1] (47 [0] (25 [0] [1])))
 9, 0.9704, (53 (37 [1] [0])(52 [1] (25 [0] [1])))
11, 0.9192, (39 [0] (53 (37 [1] [0])(52 [1] (19 [0] [1]))))
13.97s user 0.10s system 98% cpu 14.239 total
```

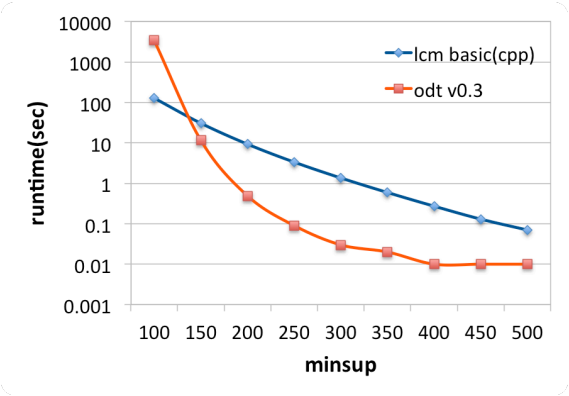

図 2: 実験 2: 葉の最小頻度に対する計算時間

ここから、サイズ $k = 1$ から 9 までは経験精度は単調に向上しているが，$k = 11$ では精度が低下している。この原因として、サイズ制約とパスの辞書順制約によって、サイズ 9 の最適木に対して子を追加することができず、異なる決定木を採用せざるを得なかったと考えられる．

4.3 実験 2: 葉の最小頻度に対する計算時間とメモリ使用量

図 2 と表 3 に，1000 タプルからなる g-credit データセット上で，葉の最小頻度 σ を 100 から 500 の間を 50 刻みで変化させたときの LCM basic と ODT の計算時間とメモリ使用量を示す。表 3 では，最小頻度 $\sigma = 125 \sim 500$ の範囲内で，どのアルゴリズムもメモリ使用量はほとんど変化しなかった．

図 2 のグラフから，x 座標の左側へ最小頻度 σ が減少するにつれて，LCM basic の計算時間は指数的に増加する一方で，ODT の計算時間は二重指数的に増加することがわかる．これは、決定木は，拡張頻出集合であるパスを組合せて作られるからだろう．さらに，大きな σ で LCM より ODT が高速なのは、ODT では木

表 3: アルゴリズムのメモリ量の比較

dataset name	minsup σ	LCM(uno) memory	LCM(ours) memory	ODT memory
g-credit	125	316.6KB	320.3KB	767.6KB
g-credit	250	316.6KB	320.3KB	761.4KB
g-credit	500	316.6KB	320.3KB	760.2KB

表 4: アルゴリズムの分類精度の比較

dataset		C4.5		DL8		ODT	
name	minsup σ	acc	size	acc	size	acc	size
chess	200	0.91	9.0	0.91	8.6	0.9117	15
g-credit	150	0.72	6.4	0.74	7.0	0.7400	9
mushroom	600	0.92	5.0	0.98	13.8	0.9862	15
vote	10	0.96	4.6	0.98	29.6	0.9747	25

の枝刈り条件によって試すべきアイテム集合数が頻出マイニングよりも抑えられるからだと考えられる。

4.4 実験 3: アルゴリズムの分類精度の比較

表 4 に、4 つのデータセット上で、提案アルゴリズム ODT と、既存のアルゴリズム C4.5 と DL8 の訓練データ上での分類精度を比較した。ただし、C4.5 と DL8 の分類精度は，実装が公開されていないため、文献 [5,6] に記載されている同一のデータセットにおける結果を用いた。

この表から、以下が観察される。まず、C4.5 に対し、ODT および DL8 は常により良い精度を示している。これは、C4.5 が貪欲法を用いているのに対し、ODT および DL8 が制約のもとで最適な木を厳密に発見する点による。木のサイズに関しては、C4.5 に対して ODT や DL8 が概して大きくなる傾向がある．これは、今回は精度のみの最適化のため，少しでも精度が改善されるなら大きな木でも採用するためである．辞書順序制約による表現力の影響については，精度とサイズについて ODT は DL8 と同等かやや劣るが，一方で ODT の DL8 に対する精度低下は 1 %未満と小さく，本実験では変数順序は大きな影響を与えていないようにみえる．

5 おわりに

本稿では、決定木の部分族である順序付き決定木の族 $\mathcal{ODT}$ に対して、入力の多項式領域のメモリしか用いずに、DL8 と同じ時間で、制約をみたす最適決定木を厳密に計算する学習アルゴリズム ODT を提案した。

今後の課題として，Nijssen ら [6] が導入したパスとして飽和集合（closed itemset）を許した決定木の族に対して，LCM アルゴリズムを組み込むことで ODT を拡張することがあげられる．

また，提案アルゴリズム ODT に対して，順序属性や連続属性への拡張することは重要である．また，ODT を，Boosting などの集団学習アルゴリズムと組み合わせた場合の有用性評価や，データの前処理による最適木発見の高速化も必要である．

最後に，本稿のアルゴリズム ODT を元にして，分類精度が一定の閾値以上であるような準最適決定木の列挙とランダム生成は興味深い課題である．これについては，別稿で議論する予定である．

謝辞

第三著者の有村は、産業技術総合研究所の瀬々 潤氏、東京大学大学院の津田 宏治氏，寺田 愛花氏，美添 一樹氏には、最適決定木構築とパターン発見の大規模化について，小宮山 純平氏と，石畠 正和氏，瀧川一学氏には、アイテム集合発見と機械学習について貴重なご議論とご教示をいただきました，ここに謝意を表します。また，湊 真一氏以下の湊基盤 (S) プロジェクトのみなさまには，日頃の議論とコメントに深謝します．

参考文献

[1] A. Knobbe, B. Crémilleux, J. Fürnkranz, and M. Scholz. From local patterns to global models: The lego approach to data mining. In *Proc. the ECML/PKDD 2008 workshop ‘From Local Pat-terns to Global Models’ (LeGo’08)*, page 116, 2008.

[2] B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *Proceedings of the fourth international conference on knowledge discovery and data mining*, 1998.

[3] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of machine learning*. MIT press, 2012.

[4] S. Morishita and J. Sese. Transversing itemset lattices with statistical metric pruning. In *Proc. the 19th ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems*, pages 226–236. ACM, 2000.

[5] S. Nijssen and E. Fromont. Mining optimal decision trees from itemset lattices. In *Proc. the 13th ACM SIGKDD int'l. conf. on Knowledge Discovery and Data Mining*, pages 530–539. ACM, 2007.

[6] S. Nijssen and E. Fromont. Optimal constraint-based decision tree induction from itemset lattices. *Data Mining and Knowledge Discovery*, 21(1):9–51, 2010.

[7] P. K. Novak, N. Lavrač, and G. I. Webb. Supervised descriptive rule discovery: A unifying survey of contrast set, emerging pattern and subgroup mining. *Journal of Machine Learning Research*, 10(Feb):377–403, 2009.

[8] A. Terada, M. Okada-Hatakeyama, K. Tsuda, and J. Sese. Statistical significance of combinatorial regulations. *Proceedings of the National Academy of Sciences*, 110(32):12996–13001, 2013.

[9] T. Uno, T. Asai, Y. Uchida, and H. Arimura. Lcm: An efficient algorithm for enumerating frequent closed item sets. In *Proceedings of Workshop on Frequent itemset Mining Implementations (FIMI'03)*, CEUR Workshop Proceedings, 2003.

[10] 長部 和仁, 宇野 毅明, and 有村博紀. アイテム集合列挙に基づく最適な順序付き決定木の高速発見. 手稿, 北海道大学情報科学研究科, 2016 年 11 月. http://www-ikn.ist.hokudai.ac.jp/∼arim/papers/osabe2016fpai102.pdf.