アルゴリズム 76−10
(2001. 1. 19)

検索確率をもつ二分探索木の探索長の最適期待値を与える領域分割の生成

大西 建輔, 星 守

電気通信大学 大学院 情報システム学研究科
〒 182-8585 東京都調布市調布ヶ丘 1-5-1
E-mail: {onishi, hoshi}@hol.is.uec.ac.jp

あらまし: 点集合とその検索確率が与えられた場合に, 検索路長の最適期待値を与える二分探索木を構成する問題を考える. 本研究では, n 点の点が与えられた場合に, どのような形状の木が最適であるかを, n 次元離散確率分布を表現する n 次元領域の分割をアレンジメントを用いて生成することにより任意の確率分布に対する最適木を与える.

Generation of subdivision for binary search tree with optimal path length of search probability

Kensuke Onishi, Mamoru Hoshi

Graduate School of Information Systems, The University of Electro-Communications,
1-5-1 Chofugaoka, Chofu-shi, Tokyo, 182-8585, Japan
E-mail: {onishi, hoshi}@hol.is.uec.ac.jp

Abstract: In this paper it is dealt with to find the best possible tree for searching table of keys with given probability. Our approach is that all binary search trees and its corresponding region of the parametric space of discrete probability distribution are constructed. In other words, for given n (number of set of points), we generate a subdivision of the parametric space of n-dimensional discrete probability distribution by hyperplane arrangement. Each region corresponds to a binary search tree and in the region such tree has optimal path length.

1 始めに

語とその語の探索確率が与えられた場合に, 探索路長の期待値を最小にする木 (最適木) を与える研究は, 辞書等の語の格納やデータベースでのデータの格納にも深く関係し, 古くから研究がなされている. 探索確率が与えられた場合には, 語数の 2 乗に比例する時間で最適木を与える算法が Knuth [2, pp.436–442] により与えられている.

今, n 個の語の集合 $A = \{a_1, a_2, \ldots, a_n\}$ とそれぞれの検索確率 $(p_1, p_2, \ldots, p_n)(\sum_i p_i = 1)$ が与えられている. 語の集合 A には, 添字順の順序があるとする. このように問題の設定を行った場合には, 二分探索木での語 a_i は, 中順序 (inorder) で数えると i 番目となる. つまり, 二分木の形状 T を決めることにより, 各語までの根からの距離 (路長) が決定されることになる. $l(a_i)$ を語 a_i までの根からの路長とすると, 木 T に対する路長の期待値は,

$$E(T) := \sum_{i=1}^{n} l(a_i)p_i$$

となり, この値を最適化することになる. $E(T)$ の値は 2 種類の変数により定義されている. 一つは, 検索確率 p_i であり, もう一つは, 枝長 $l(a_i)$ である. Knuth は, 前者あ与えられたとき, $E(T)$ を最適化する T を構成することを考え, 単調性の原理を用いた $O(n^2)$ 時間のアルゴリズムを提案した. 我々は, 木の構造を決定することにより, $l(a_i)$ を決め, その木が最適となるような確率分布の領域を考え, n 次元離散確率分布全体の n 次元空間

$R = \{(p_1, p_2, \ldots, p_n) \mid \sum_i p_i = 1\}$ を分割することを行う．つまり，期待値

$$f(\boldsymbol{p}, \boldsymbol{l}) := E(T)$$

を $\boldsymbol{p} = (p_1, \ldots, p_n)$,$\boldsymbol{l} = (l(a_1), \ldots, l(a_n))$ の関数とみなし．R を n 点からなる二分木の集合の濃度 (有限) に分割し，それぞれの領域では，ある二分木が最適木となるという対応を考えた．Knuth は，$\boldsymbol{p}$ を定数とし，f を最適とする二分検索木を生成した．本稿では，そのような分割を生成するアルゴリズムを述べる．

まず，2 章では，すべての二分木の生成を行うために，二分木と 2 進数列の対応を述べ，そのコードの生成アルゴリズムを示す．次に，3 章で 2 進数列からその木に対する期待値 $E(T)$ を計算するためのアルゴリズムを示す．4 章では，R の分割を生成するためのアルゴリズムを提案し，その解析を行う．

2 二分木の生成 (生成木の構築)

本章では，n 点からなる語の集合 A に対するすべての二分木を生成するためのアルゴリズムを説明する．このため，二分木の生成木を定義する．この生成木では，根から葉までの 1 つの路が 1 つの二分木を表す．

まず，二分木と 2 進数列の対応について説明をする．次のような 2 進数列の集合を考える．

$$B_n = \left\{ (b_1 \cdots b_{2n+1}) \;\middle|\; \begin{array}{l} b_i = 0 \text{ or } 1. \\ \textbf{制約 (1)}\ 0,1 \text{ の数はそれぞれ } n, n+1 \text{ 個づつ} \\ \textbf{制約 (2)}\ 2 \text{ 進数列の先頭から任意の場所までの } 1 \text{ の数は } 0 \text{ の数を超えない} \end{array} \right\}$$

$n = 3$ の場合には，次のような集合となる．

$$B_3 = \left\{ \begin{array}{c} (0101011), (0011011), (0001111), \\ (0100111), (0010111) \end{array} \right\}$$

二分木と 2 進数列の対応は，次のように与えることが可能である．

二分木から 2 進数列への対応: 木のノードを前順序 (preorder) で走査する．内点の場合は，0 を対応させ，葉の場合には，1 を対応させる．

2 進数から二分木への対応: 2 進数列を左から順に見ていく．0 のときは，子ノードとして，内点を，作り，1 のときは，葉を作る．内点が既に 2 つの子供を持っている場合は，その親に戻り，同じ操作を繰り返す．

つまり，二分木を数え上げる問題は，上の 2 進数列をすべて生成するという問題となる．$n = 3$ の場合の対応は図 1 に示した．

まず，この 2 進数列の集合の構造を考えると，制約条件から次の性質がわかる．

1. 先頭から数えて，0 の数が n になると，残りは全て 1 である (制約 (1) より)．
2. 先頭のビットは，必ず 0 である．最後の 2 ビットは，必ず 1 である (制約 (2) より)．

ここで，次のような二分木を構成することを考える．根に 0 を与え，左の枝に 0，右の枝に 1 を与える．根から葉に至る路が B_n に含まれる 1 つの 2 進数列，つまり 1 つの二分木を表現する．木全体としてみた場合には，全ての内点が n 点からなる二分木の全体を含んでいるため，n 点からなる二分木の**生成木**と呼ぶ．$n = 3$ の場合の生成木を構成すると，図 2 を得る事ができる．根から黒丸の葉へ到達する路が，内点数が 3 の 二分木を表現する 2 進数列を表現していることになる (アルファベットはそれぞれ図 1 の二分木と対応している)．また，四角は，これ以上木を辿る必要のない部分木を表現している．この四角には制約により，2 種類ある．一つ (薄い四角) は，0 の数が n になり，これ以上 0 を辿る事ができない部分木であり，その数はちょうど黒い葉の数と一致する (上の**性質** (1) より)．またもう一つ (濃い四角) は，**制約** (2) より，これ以上辿る事の出来ない部分木である．

このように生成木を深さ優先探索を用いて，辿ることにより，全ての木を構成することは可能であるが，ここでこの 2 進数列の性質を利用することにより，より速いアルゴリズムの構築が可能である．

補題 1 二分木の内点 *(2 進列では 0)* の個数 n が与えられているとする．$b_1 b_2 \cdots b_i$ で始まる *2* 進数列を表現する部分木を T とする．また，$b'_1 b'_2 \cdots b'_i$ で始まる *2* 進数列を表現する部分木を T' とする．ただし，$b_1 b_2 \cdots b_i$ と $b'_1 b'_2 \cdots b'_i$ に含まれる 1 の数

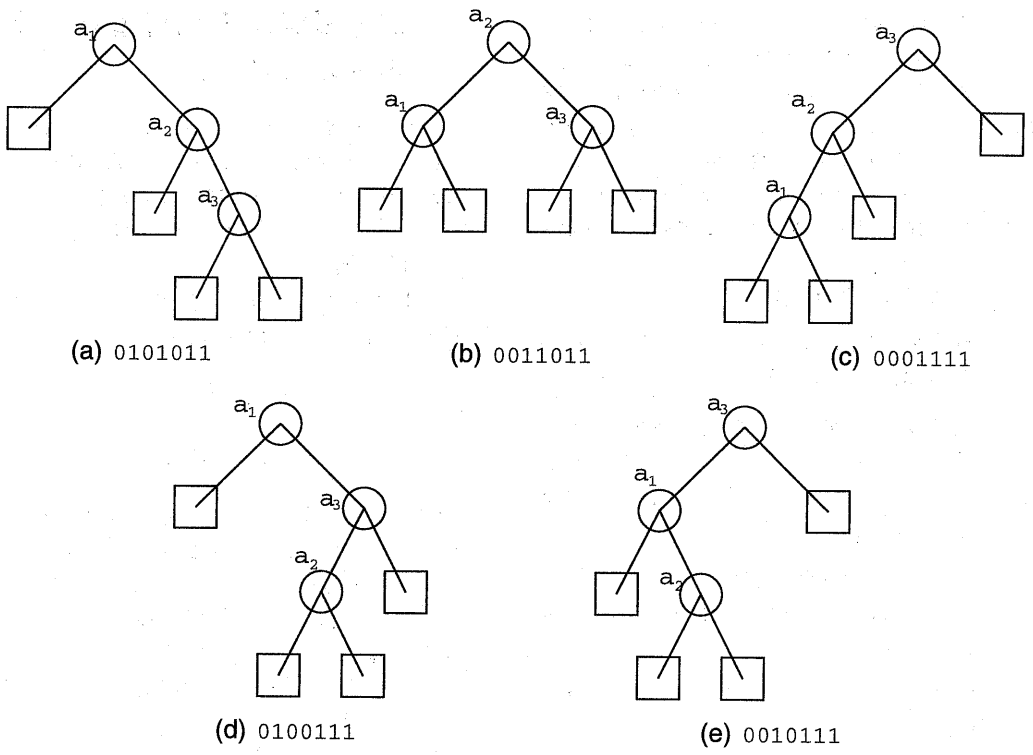

図 1: 3 点のノードからなる二分木の全体

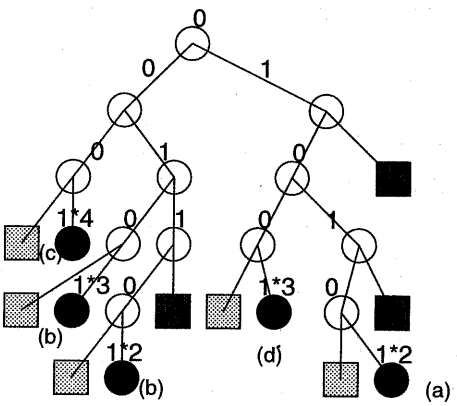

図 2: 内点数が 3 の場合の生成木

は, 等しいとする. このとき, T と T' の間には, 枝のラベルも含めて, *1* 対 *1* の対応が存在する.

証明: 今, $b_1 \ldots b_i$ に含まれる 0 の数を j, 1 の数を k とする $(j+k=i)$. このとき, 2 進数列の残りの部分に含まれる 0,1 の数は, $n-j$ 個, $n+1-k$ 個である. $b_1 \ldots b_i$ に続き, **制約 (1),(2)** を満たし, 0,1 の数が $n-j$ 個, $n+1-j$ 個の 2 進数列を全て生成することになる. これらの 2 進数列を全て含むような二分木は一意に定まる.

また, $b'_1 b'_2 \cdots b'_i$ に含まれる 0,1 の数も, T と同じであるため, この部分列に続く 2 進数列に対する制約は, $b_1 \cdots b_i$ の場合に対する制約と同一である. つまり, 生成されるべき 2 進数列は, 一致する. 結局, これら 2 進数列を含む部分木も, ラベルも含め一致する. □

例 1 図 *2* に着目する. 001, 001 に続く部分木は, ラベルも含め同じ部分木である. ここで, 100 となる部分木が存在しないのは, **制約** *(2)* が存在するためである.

2 進数列を $b_1 \cdots b_{j+k} \cdots b_n$ を $b_1 \cdots b_{j+k}$ と $b_{j+k+1} \cdots b_n$ の 2 つに分割することを考える. こ

のとき, 前の部分列に含まれる 0 の数を j, 1 の数を k とする. 全体の 0 の数 n と, そのときの j,k を決めたときに, 生成される部分 2 進数列の集合を $T(n,j,k)$ $(j=1,\ldots,n, k=1,\ldots,j)$ と表す. この j,k に関する表を 1 度作成することにより, 同じ計算をせずに済むことが 補題 1 よりわかる. つまり, 次のようにアルゴリズムを設計すればよいことがわかる.

[アルゴリズム設計の指針]

- 部分木の探索に入るかどうかは, $T(n,j,k)$ の j,k の部分が空であるかどうかで判定を行う.
- 空であれば、部分木を辿り, そうでない場合は, $T(n,j,k)$ を参照し, その要素に, そこまで辿ってきた部分の接頭文字を付加し, 2 進数列として, 出力を行う.
- 一つの 2 進数列を見付けると, その全ての接尾文字 (suffix) を $T(n,j,k)$ の該当部分に登録を行う.

この指針に従い設計を行ったアルゴリズムを図 3 に示す.

ADD(j, k, $\{B[0]\cdots B[2n]\}$) は, 対応する $T(n,j,k)$ に $B[0]\cdots B[2n]$ の全ての接尾字を登録する操作である. つまり, その関数は図 4 のように書くことができる.

[アルゴリズム解析] ADD に関しては, 2 進数列の長さだけ, つまり, $2n=O(n)$ だけの時間がかかる. さらに, 全ての 2 進数列を生成するアルゴリズムでは, 全ての内点と葉の数を足した時間に比例することがわかる. 葉の数は, 黒丸と, 2 種類の四角を足しただけ存在する. 黒丸は, カタラン数, 薄い四角も、カタラン数だけ存在する. 濃い四角は, $(n-1)$ までのカタラン数を足した数だけ存在する. よって, 葉の数は,

$$\binom{2n}{n}/(n+1)+\binom{2n}{n}/(n+1)+\sum_{i=1}^{n-1}\binom{2i}{i}/(i+1)$$

また, 内点数は, (葉の数) -1 となる. よって, このアルゴリズムは,

$$O\left(\binom{2n}{n}/(n+1)\right)$$

時間かかることになる. この値は, 出力サイズに比例しており, 最適であるといえる. また, $T(n,0,0)$ と B_n が等しいことから, メモリとしても, ほぼ同じ出力サイズのメモリが必要となる.

3 二分探索木の期待値を計算するアルゴリズム

前節で, 2 進数列の生成ができることを示した. 次に, この 2 進数列から, その二分探索木の路長の期待値 (コスト) を計算することを考える. このアルゴリズムは, 2 進数列から, 各内点までの路長を計算を行う.

まず, 前節で得られた 2 進数列に対して, 次の 2 つを決定しなければならない.

- どの 0 が a_i に相当するか.
- 各 a_i までの根からの路長.

前者は, 二分木を中間順走査 (inorder traverse) をおこなった場合に, 数え挙げられる順に, a_1 から a_n を割り振っていけばよい. なぜならば, それぞれのもともと a_i には全順序があるため, その順に沿ってしか, 二分木に登録できないためである. 後者では, 内点に語 a_i が与えられたときに, 路長が決定される. この 2 つの操作と同時に, 2 進数列から, 二分探索木を生成するアルゴリズムを以下に示す. 二分探索木を出力する場合には, 2 進数列に前順走査を行うことになるので, 前順序, 中間順序の走査を同時に扱うアルゴリズムとなる.

アルゴリズムの詳細は, 図 5 に示す. このアルゴリズムは, 与えられた 2 進数列を前から走査していき, 0 の場合は, スタックに 0 を積み, 1 の場合には, 0 を取り出し, そのノードに二分探索木での内点 a_i までの路長を割り振ることを行う。この路長は $C[i]$ $(i=0,\ldots,n)$ に保持され, この値から, 二分探索木に対する期待値を計算することになる.

アルゴリズム中では, 二分木の走査を行う場合以外のことを実行している部分が存在する. 二分木の走査では, 内点と葉を順にたどることになるが, その部分木のノードを全てたどった後に, 枝を部分木の根まで戻らなければならない. しかし, 2 進数列には, その後戻り部分は存在しない. これは, 2 進数列で表現するには, 冗長な部分であり, 表現されていない. そのため, (スタックの深さ)=(木の深さ) とはならず, スタックの深さが 0 の場合の処理が必要となる.

このアルゴリズムで, 2 進数列から, 木とそれぞれの内点までの路長が計算可能となる. 実際の期

Input 点数 n;

Output $2n+1$ 点からなる 2 進数列で, 二分木に対応する列の集合 B_n

1 $T(n,j,k)=\emptyset$ for $j=1,\cdots,n,\ k=1,\ldots,j$;

2 $B[i]=0(i=0,\cdots,2n)$; /* $2n+1$ の 2 進数列を保存する */

3 $B_n=\emptyset$;

4 MakeBinarySequences$(n,0,0)$;

```
MakeBinarySeqences(n,j,k) {
    if (T(n,j,k) ≠ ∅) then
        for (B[j+k]···B[2n] ∈ T(n,j,k)) do
            B_n := B_n ∪ {B[0]B[1]···B[2n]};
            ADD(j,k,{B[0]···B[2n]});
        endfor
    else
        if (j == n) then
            for (i = j+k to 2n) B[i] = 1; endfor
            B_n := B_n ∪ {B[0]B[1]···B[2n]};
            ADD(j,k,{B[0]···B[2n]});
            End;
        endif
        if (j <= k) then
            End;
        endif
        B[j+k] = 0;
        MakeBinarySeqences(n,j+1,k);
        B[j+k] = 1;
        MakeBinarySeqences(n,j,k+1);
    endif
}
```

図 3: アルゴリズム (n 内点の二分木の生成木を構築)

待値は, $C[i]$ を用いて,

$$\sum_{i=1}^{n} C[i]\cdot p_i$$

と表現することが可能である. $\sum_i p_i=1$ という制限があるため, $p_n=1-\sum_{i=1}^{n-1} p_i$ と置き換えて次元を下げると結局次の式が期待値として得ることができる.

$$\sum_{i=1}^{n-1}(C[i]-C[n])\cdot p_i + C[n]. \qquad (3.1)$$

[アルゴリズム解析] アルゴリズムは, **for** を $2n+$

```
ADD(j, k, {B[0] ··· B[2n]}){
    for j + k to 0 do
        T(n, j, k) := T(n, j, k) ∪ {B[j + k] ··· B[2n]};
        if (B[j + k − 1] == 0) then j := j − 1 else k := k − 1; endif
    endfor
}
```

図 4: アルゴリズム ($T(n,j,k)$ に接尾字を登録)

Input 2 進数列 $b_1 b_2 \cdots b_{2n+1} \in B_n$.

Output 内点に $a_1, \cdots, a_n$ が割り振られた 2 分木と各内点までの根からの枝長 $C[i](i = 1, \cdots, n)$.

Main 2 進数列を前から順に調べ, その値が 0 か 1 かによって, それぞれ処理を行う.

```
Initialization Index := 0; Depth := 0; Level := 0;
for j = 1 to 2n + 1
    if (b_j == 0) then
        Push 0 to Stack;
        Depth := Depth + 1; Level := Level + 1;
    else if (b_j == 1) then
        Pop 0 from Stack;
        Index := Index +1;
        Label the poped node 0 to a_Index;
        Depth := Depth - 1; Level := Level - 1;
        C[Index] := Level;
        if (Depth == 0)
         Level := Level + 1;
        endif
    endif
endfor
```

Comment

Index: 現在訪れている内点の番号.

Level: 現在訪れている内点の根からの枝長.

Depth: 現在のスタックの深さ.

図 5: アルゴリズム (2 分木の生成と内点のラベル決定, 路長決定)

1 回動かすので, $O(n)$ のアルゴリズムとなる.

先に, $n=3$ の場合には, 5 種類の木が生成できることを述べた (図 1). それらに対するそれぞれの期待値は, 次のように計算が可能である.

木	期待値	
(a)	$0\cdot p_1+1\cdot p_2+2\cdot p_3$	$2-2p_1-p_2$
(b)	$1\cdot p_1+0\cdot p_2+1\cdot p_3$	$1-p_2$
(c)	$2\cdot p_1+1\cdot p_2+0\cdot p_3$	$2p_1+p_2$
(d)	$0\cdot p_1+2\cdot p_2+1\cdot p_3$	$1-p_1+p_2$
(e)	$1\cdot p_1+2\cdot p_2+0\cdot p_3$	p_1+2p_2

4 分割生成のアルゴリズム

前節では, 2 進数列から, 二分探索木の形状を計算し, 二分探索木の路長の期待値を計算するアルゴリズムを与えた.

全ての二分探索木に対応する 2 進数列を入力することにより, 路長期待値が二分探索木の個数分だけ生成されることになる. 本節では, これらの期待値から分割を生成するアルゴリズムを与え, その解析を行う.

まず, $p_1,\ldots,p_n$ を与えた場合に, それぞれの二分探索木の期待値が決まるということから, p_i を変数とみなし, その期待値を考える. 与えられた $p_1,\ldots,p_n$ に対して最適な期待値として, それぞれの二分探索木の期待値の比較をおこない, 1 つ以上の二分探索木が最適値を与える木として得られる.

これを幾何的に考る. p_i は変数であり, x_i $(0\le x_i\le 1)$ と置き換える. さらに期待値を x_i の関数とみなすと, 式 (3.1) より,

$$y=\sum_{i=1}^{n-1}(C[i]-C[n])\cdot x_i+C[n]. \qquad (4.1)$$

という超平面が二分探索木の数だけ得られることになる. この超平面は $R:=\{(x_1,x_2,\ldots,x_{n-1})\mid x_i\ge 0,\sum_i x_i\le 1\}$ という領域でのみ, 定義されている. これらの超平面の中で最も値の小さい部分を計算することになる. つまり, (4.1) を用いた超平面アレンジメント $\mathcal{A}$ を構成し, y 軸に関して最も下側部分のみを R へ射影すればよいことがわかる.

これをアルゴリズムの形式で書くと図 6 となる.

このアルゴリズムにより, 3 次元の場合に得られる分割は図 7 となる.

アルゴリズム (領域分割の生成) の解析を行なう. まず, Step1 の二分木の列挙であるが二分木の数はカタラン数と対応がつき, 2 節で述べたように, $\binom{2n}{n}/(n+1)\approx 4^n/(\sqrt{\pi}n^{3/2})$ となる. この列挙に必要な計算量は,

$$O\left(\binom{2n}{n}/(n+1)\right)$$

である.

Step2 では, 全ての二分木に対して, アルゴリズムが適用されるので, $O(n)\times$ (木の個数) となる. 二分木の個数は, カタラン数であるため, $\binom{2n}{n}/(n+1)\approx 4^n/(\sqrt{\pi}n^{3/2})$ となる. よって, 全ての木に上記のアルゴリズムを適用することによりかかる計算時間は,

$$O(4^n/(\sqrt{\pi}n^{1/2}))$$

となる.

Step3 では、n 次元で、$4^n/(\sqrt{\pi}n^{3/2})$ の超平面からなる超平面アレンジメントを構成することになる. この構成には,

$$O((4^n/(\sqrt{\pi}n^{3/2}))^n)$$

の計算時間がかかる. このアレンジメントの下側を射影する操作は, 面数に比例するため、$O(4^n/(\sqrt{\pi}n^{3/2}))$ となる. よって, このアルゴリズムの計算時間は,

$$O((4^n/(\sqrt{\pi}n^{3/2}))^n)$$

である.

定理 1 n 個の全順序をもった語の集合に対する最適期待値を与える領域分割は,

$$O\left(\left(\frac{4^n}{\sqrt{\pi}n^{3/2}}\right)^n\right)$$

時間で計算することが出来る. また, 一つの二分木に対して,

$$O\left(\left(\frac{4^n}{\sqrt{\pi}n^{3/2}}\right)^{n-1}\right)$$

時間で領域分割を得ることが可能となる.

5 まとめ

本稿では, 各ノードまでの路長の期待値が最適な二分探索木を与える領域分割をおこなうための

1. n 点からなる二分木 (2 進数列) の列挙 (二分木に対する生成木の利用).

2. 図 5 のアルゴリズムを全ての二分木 (2 進数列) に適応し, $C[i]$ を計算 (二分探索木の生成).

3. 式 (4.1) で表現される超平面をによる超平面アレンジメント $\mathcal{A}$ を計算.

4. 超平面アレンジメントの下側部分を元の空間 R に射影.

図 6: アルゴリズム (領域分割の生成)

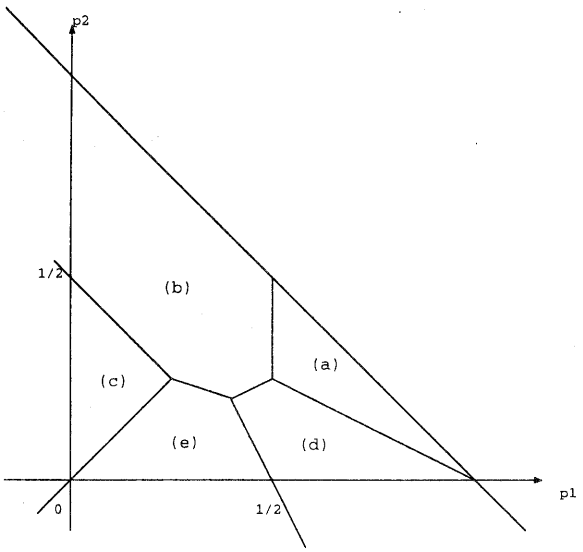

図 7: $n = 3$ の場合の領域分割

アルゴリズムを提案した. このアルゴリズムには, 1) 二分木に対応する 2 進数列の列挙 (生成木の利用); 2) その木に対する期待値を表現する式の生成; 3) 超平面アレンジメントの計算と射影; という 3 つのアルゴリズムが必要となる. 1),2) に関しては, その詳細なアルゴリズムをそれぞれ 2 節, 3 節にて示した. また, 3) に関しては, 既存のアルゴリズムを使うことにより, 生成が可能である. 既存のアルゴリズムに関しては, Edelsbrunner[1] などを参考にしてもらいたい.

また, 1),2) で用いたアルゴリズムを同時に実行することにより, さらなるアルゴリズムの効率化や二分木の各ノードまでの路長が与えられた場合にそれを満たす二分木が存在するのか. 問題の解決も可能であると思われる.

参考文献

[1] H. Edelsbrunner : *Algorithms in Combinatorial Geometry*. Springer-Verlag, Berlin, 1987.

[2] D.E. Knuth: *The Art of Computer Programming*, Vol. 3, Second Edition, Addison-Wealey, 1998.