異なる長さをもつ表形式データの転置†

佐藤隆士†† 津田孝夫†††

著者らは，関係データベースほかの応用のため固定長データからなる表形式データの転置および任意並べかえアルゴリズムについて研究してきた．近年，知識工学の発展により，データベース，知識ベースに複雑なデータ(complex data)の扱いが必要になってきている．このような状況から，本論文では，データ量が大きく計算機の主記憶に納まりきらないような表形式データについて，1) 同一列は同じ長さのデータからなるが，列ごとに長さが異なる場合，2) 各要素のデータ長がすべて異なる場合，を扱う．1行を1ブロックとし，コストをブロック転送回数で評価し，転置を能率よく行う方法を提案する．本論文の手法では，表形式データを列方向に分割することを繰り返し，最終的に一列ごと別ファイルにし転置を完了する．その際，上記 1) については，分割の過程を最適にすることができ，2) についても同じアルゴリズムで能率よく転置できることがわかった．

1. はじめに

記憶階層下でのデータ並べかえ問題についてはFloyd の文献[1]にみられる．Floyd は，ページ化された2次記憶と，2ページ分の容量をもつ主記憶間で，データを並べかえるのに必要な手間の下界を，特別に定めた演算の回数で求めている．著者らは，Floyd の方法を拡張，改良し，関係データベースほかへ応用を試みた[2)~4)]．固定長の要素をもつ表形式データを対象とし，その転置および任意並べかえについて研究した．

近年，知識工学の発展により，データベース，知識ベースに複雑なデータ (complex data) の扱いが必要になってきている．このような状況から，本論文では，データ量が大きく計算機の主記憶に納まりきらないような表形式データについて，

1) 同一列は同じ長さのデータからなるが，列ごとに長さが異なる場合，

2) 各要素のデータ長がすべて異なる場合，

を扱う．1行を1ブロックとし，コストをブロック転送回数で評価し，転置を能率よく行う方法を提案する．

本論文の手法では，表形式データを列方向に分割することを繰り返し，最終的に一列ごと別ファイルにし転置を完了する．アルゴリズムは，あらかじめ列分割の過程を求める部分と，その過程に従って，2次記憶を効率的に使用しながら実際に転置を行う部分とからなる．提案のアルゴリズムにより，上記 1) については，その分割過程を最適にすることができる．また，2) についても同じアルゴリズムで能率よく転置できることがわかった．

2. 諸定義

本論文では，主記憶，2次記憶からなる2階層モデルを扱う．転置すべきデータは2次記憶上にあるが，データ量が大きく全体を主記憶におくことはできない．主記憶，2次記憶間のデータ転送単位をブロックという．1ブロックの大きさを p，主記憶のデータ容量を w ($\geqq 2$) ブロック，転置の対象となる全データを d ブロック分とする．先の仮定から $d>w$ である．1ブロックは，1個の論理レコード（以下，単にレコード）からなっている．そして，各レコードは m 個のフィールドをもっている．各フィールドは0から $m-1$ の番号で呼ばれ，異なる長さをもつ．行方向に1ブロックを書くと，**図1**のような表の形になるので，転置対象のデータを**表形式データ**という．レコードは行，フィールドは列，フィールド値は要素に対応する．

転置とは，レコード順に同一フィールド値を集める操作である．たとえば，“氏名，住所，Tel,…” からなるレコードをフィールドごとに，氏名のみのファイル，住所のみのファイル，Tel のみのファイル，…を作る操作に相当する．なお，転置されたファイルはトランスポーズ形ファイルとも呼ばれる[5)]．

1レコード/ブロックを仮定したが，現実には1ブロックに複数レコードが存在する場合もあるし，1レコードが複数ブロックに分割されて格納されることもある．しかし，次のように考えると，1レコード/ブ

† Transposition of Tabular Data Structures Having Different Column Size by TAKASHI SATO (Department of Information Engineering, Takuma National College of Technology) and TAKAO TSUDA (Department of Information Science, Kyoto University).

†† 詫間電波工業高等専門学校情報工学科
††† 京都大学工学部情報工学科

ロックの問題に帰着できる．

まず，1ブロックが r (≧1) レコードをもっている場合だが，このときはもとの並びで同一ブロックにはいっている r 個の同じフィールドの値は，転置後も同じブロックに移動させられるので，これら r 個をまとめて1つの要素と考える．つぎに1レコードが b (≧1) ブロックに分割される場合だが，このときは b ブロックおきに同じフィールドが格納されるので，b ブロックおきの組ごとに転置を行えば，1レコード/ブロックのアルゴリズムを b 回繰り返すことで全体の転置ができる．

アルゴリズムの**コスト**は，主記憶，2次記憶間のブロック転送回数とする．提案のアルゴリズムでは，読込み（2次記憶から主記憶へのブロック転送）回数と書出し（主記憶から2次記憶へのブロック転送）回数が等しいので，コストの見積には読込み回数を用いている．

次に木に関した用語の定義をしておく．**t-分木**とは，節の有限集合であって，空集合であるか，もしくは，根と t 個の異なる t-分木（もとの木に対して部分木ともいう）とからなるものをいう．根はその部分木の根に対して親といい，逆の関係を子という．t-分木を拡張して，もとの木でその部分木が空になっているところに特別な印を付けたものを**拡張 t-分木**という．たとえば，**図2**で（a）の2分木の拡張2分木を（b）に示す．特別な印は図では□であり，外部節点という（○印は内部節点）．

各節点から根までの距離をその節点の深さという．内部節点を上から下に，同じ深さでは左から右に順につめた拡張 t-分木を完全 t-分木という．

各外部節点に荷重を与え，外部節点からの深さと荷重の積を，外部節点すべてにわたって和をとったものを**荷重路長**という．荷重路長最小の t-分木とは，同数で同じ荷重の外部節点をもつ拡張 t-分木のうち荷重路長が最小のものをいう．たとえば，荷重の組を 2, 3, 6, 14, 17, 19 とするとき，荷重最小の2分木は**図3**のようになる．内部節点には，部分木の荷重の和が書き込んである．その総和は荷重路長になっている．荷重路長最小の2分木を求めるアルゴリズムは，D. Huffman が見つけ，Huffman 符号として有名である．

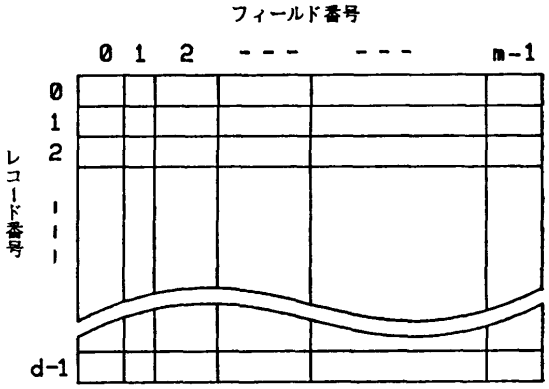


図1 表形式データ
Fig. 1 Tabular data structure.

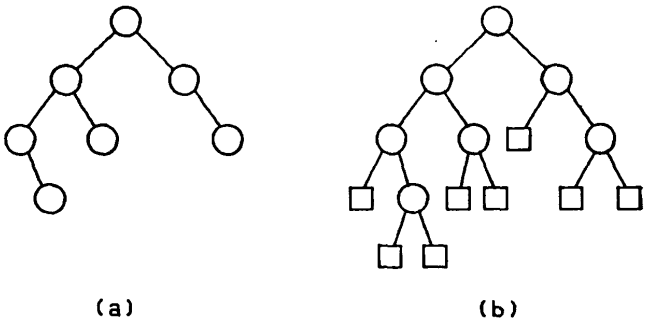


図2 2分木と拡張2分木
Fig. 2 Binary tree and extended binary tree.

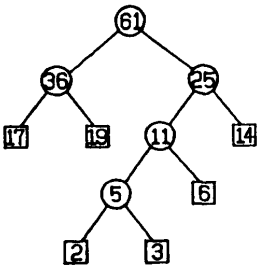


図3 荷重路長最小の2分木
Fig. 3 Binary tree which has a minimum weighted path length.

3. 列ごとに異なる長さをもつ表形式データの転置

本章では，フィールドごとに異なる長さをもつが，同一フィールドでは同じ長さをもつ場合の転置を扱う．$m \leqq w$ の場合は，フィールドごとに主記憶のバッファを割り当てることにより簡単に転置できるので，興味の対象は $m > w$ の場合である．提案のアルゴリズムは，列の集合（グループ）を主記憶のデータブロック数 (w) だけの互いに素な部分集合（サブグルー

プ）に分割する操作を繰り返す．サブグループごとに主記憶のデータブロック1つを割り当てる．別に主記憶に1ブロック分のバッファを用意し，グループのブロックを2次記憶から読み込み，サブグループごとのブロックに分配する．すべての列集合の要素数が1になったとき，アルゴリズムは終了する．このとき，同じフィールドの要素が2次記憶の連続領域を占めるようになるので，転置が完了したとみなす．$d=8$, $w=2$, $m=4$, 各フィールドの長さの比を 4：2：1：1 とした場合の例を**図4**, **図5**に示す．図4は論理レベルで分割過程を示したものであり，図5は1ブロック1行で表した物理レベルの記憶配置である．このアルゴリズムの要点は，列集合の分割過程の決定と2次記憶のブロック管理である．以下，これらを順に説明し，コストの上界を示す．

3.1 列集合の分割

列集合の分割過程は拡張 w-分木で表現できる．根は全体の列集合に対応し，親子関係はグループ，サブグループの関係すなわち，列分割を表す．外部節点は表形式データの各列を表すが，拡張 w-分木の外部節点数 m' は分割回数を s とすると，

$$m'=w+s(w-1)$$

となるので，m' が列数 m と同じにできないときは，$w-2$ 個以内の長さ0の列を擬似的に加え，$m'=m$ となるようにする．外部節点に荷重として対応する列のデータ量をブロック数で書いておく．各列の長さを l_i $(i=0,1,\cdots,m-1)$ とすると各列のデータ量はブロック数で，

$$d_i \fallingdotseq dl_i \Big/ \sum_{j=0}^{m-1} l_j$$

である．等号でなく近似になっているのは，1つのフィールド値が2つのブロックに分割されないようにしていること，1レコード/ブロックだが，レコードがブロックより短く，空白があるかもしれないこと，などの理由により，各列の長さだけではブロック数を正確に決められないからである．しかし，実用的にはこの近似値で十分である．

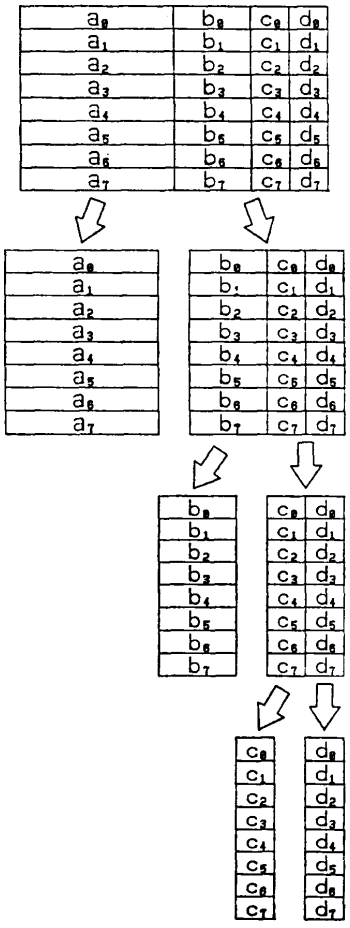


図4 列の分割過程例（論理レベル）
Fig. 4 Partition process of columns (logical level).

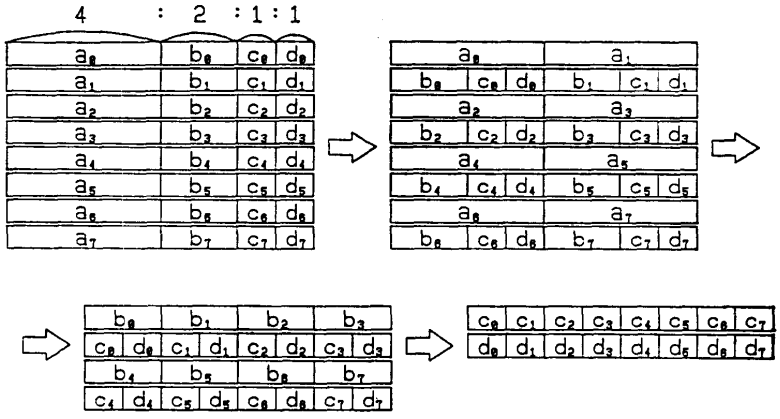


図5 列の分割過程例（物理レベル）
Fig. 5 Partition process of columns (physical level).

[補題1] 主記憶の w ブロックを用いて，列集合の分割に要するコストを最小とする分割過程を求める問題は，各列のブロック数を荷重とした荷重路長最小の拡張 w-分木を求める問題に帰着する．

（証明） 各列のブロック数を荷重とした拡張 w-分木を作る．下にある内部節点から順に上（根）方向に，

子に記入された数の和を書いていく．この値は，その内部節点と子が表す列分割に要するコストになっている．したがって，内部節点に与えられた値の総和は全列分割過程のコストである．しかも，この値は，定義から拡張 w-分木の荷重路長にも等しいので，補題1を得る． □

荷重路長最小の拡張 w-分木を求めるには，荷重の列から小さい順に w 個取り去り，その和をもとの列に加える操作を列の要素数が1になるまで繰り返せばよい[6]．したがって，列集合の最適な分割過程を求めるアルゴリズムは，**アルゴリズム1**のようになる．

[定理1] アルゴリズム1は，列集合の最適な分割過程を与える．

(証明) アルゴリズム1は，ラベルに関する操作を除き，文献6)の荷重路長最小の w-分木を求める方法と同じである．したがって，補題1からアルゴリズム1は，コスト最小つまり最適な列集合の分割過程を与える． □

なお，ラベルは分割過程のトレースを容易にするためである．ラベルにより，節点の親子間の列分割の関係を直接的に表現できる．

[例] 主記憶のデータブロック数 $w=3$ で，列数 $m=8$ の表形式データを転置する．列番号は $0,\cdots,7$ とする．各列のデータ量をブロック数で，順に 10, 3, 4, 20, 6, 7, 4, 2 としたときの最適な分割過程を求める．まず，$m'=9$ となるので，長さ0の列を1個付け加えて考えることになる．荷重を値とする節点を作り，列番号をラベルとして付けると，**図6**(a)のようになる．次に値の小さい順に 0, 2, 3 の節点をまとめると値が5で，ラベルが {1, 7} の根をもつ木ができる (同図(b))．この木と，値4の節点2つをまとめる (同図(c))．次に値 6, 7, 10 の節点をまとめ (同図(d))，最後にこれと(c)でできた根の値13の木，残っている値20の節点をまとめて1本の木にする (同図(e))．できあがった拡張 w-分木の荷重路長は，内部節点の値の和だから97である． □

```
begin
  m から m' を計算し，m'−m 個の荷重0の列を付け加える；
  荷重を値（コスト）とする節点を作り，列名を集合の型でラベルとして
  付ける．ただし，長さ0の列は空集合としておく；
  /* 節点が1個の木のみからなる森（木の集合）ができる */
  while 森に含まれる木が1本より多いかぎり do begin
    森の中から根のコストが小さい順に w本の木をみつける．その根を
    n1,…,nw とする；
    n1,…,nw のコストの和を値としてもち，n1,…,nw のラベルの和集合
    をラベルとする節点（fとする）を作る；
    n1,…,nw をfの子にする
    end
  end
```

アルゴリズム1 列集合の最適な分割過程を求める
Algorithm 1 Algorithm to compute an optimum partition process of columns.

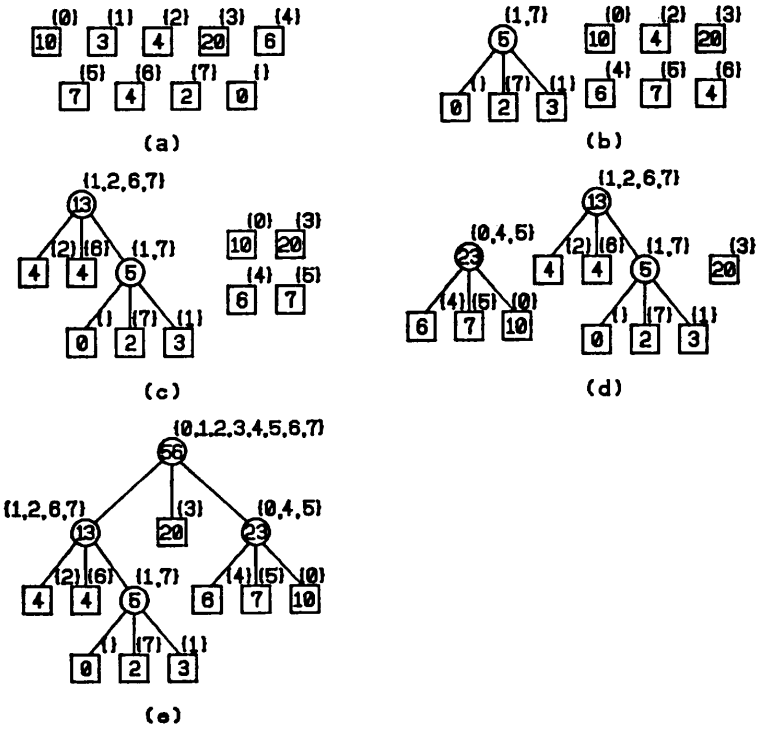


図6 列集合の分割過程を決める w-分木の作成
Fig. 6 Construct w-ary tree which determines a partition process of w-ary tree.

3.2 列集合の分割方法

前節で列集合の分割過程を決める拡張 w-分木ができているので，次に2次記憶上の表形式データを列方向に分割することにより転置を行う．分割の際，2次記憶上のメモリ空間を有効に使用するため，2次記憶のブロック管理に次のような工夫をする．分割過程における列集合（グループ）に含まれる2次記憶上のブロックは物理的に連続した領域でなくてもよいよう，

現存する各グループに対して，そのグループに含まれるブロック番号の列を主記憶にリストの形でもっておく．これをブロックリストという．また，2次記憶上の空ブロック番号をリストにつないだものを Free としておく．グループをサブグループに分割する際，グループのブロックのうち，2次記憶から主記憶に読み込まれたものは，その跡の領域は再利用可能（すなわち，別のデータで書き換えられてもかまわない状態）なので，そのブロック番号を Free につないでおく．主記憶上では，w 個のサブグループが主記憶の w 個のブロックを用いて組み立てられるが，主記憶のそれぞれの組立て用ブロックがいっぱいになったら，Free から1ブロック番号を取り出し，そのブロックに書き出す．そして，そのブロック番号をそれが属するサブグループのブロックリストにつないでおく．このように2次記憶上のブロックを再利用することにより，必要な2次記憶の領域を転置操作の最初から最後までほぼ一定にできる．いわば**その場での転置**ができるわけで，もともと主記憶に収まらないような大量データを扱っているので，このことは特に重要である．

図7に図6(d)を例に2次記憶のブロック管理の様子を示す．図では，ブロックリストに2次記憶のブロック番号を書くかわりに，ブロックへのポインタ（矢印）を用いている．

以上をまとめると列分割のアルゴリズムは**アルゴリズム2**のようになる．

3.3 コストの上界

3.1 節でも述べたように，ブロックの読込み回数は列集合の分割過程を決める拡張 w-分木の荷重路長である．本節では，アルゴリズム1で求まる分割過程で転置を行った場合に要するコストの上界を示す．

各列の長さが同じで外部節点の荷重が等しい場合，次の補題が成立する．

［補題2］ 外部節点の荷重が等しい場合，拡張 w-分木の荷重路長最小は完全 w-分木で実現できる．外部節点数を m，各外部節点の荷重を d_i とすると最小値 E は，

$$E=d_i\left\{m\lfloor\log_w m\rfloor+\frac{w}{w-1}(m-w^{\lfloor\log_w m\rfloor})\right\}$$

となる．

（証明） 文献 6) 参照． □

上式右辺 { } 内は，関数 $m\log_w m$ の m が w のべき乗の点をむすんだ折れ線関数になる．また，各列の長さが異なる場合に関して，次の補題が成立する．

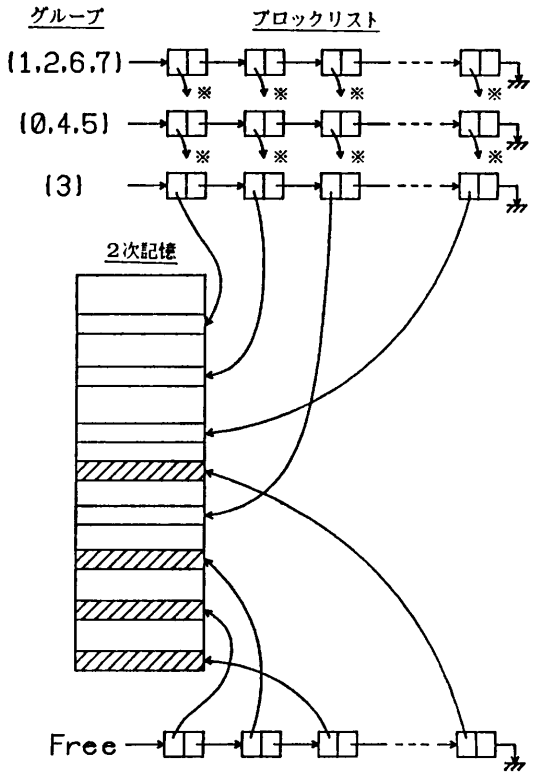


図7 2次記憶のブロック管理（※は2次記憶のブロックへのポインタ，斜線は空ブロック）
Fig. 7 Block management of secondary storage. Each ※ indicates a different block of secondary storage.

［補題3］ 拡張 w-分木の荷重の総和が同じならば，荷重が均等な場合より，不均等な場合の方が荷重路長を小さくできる．

（証明） 深さ l_1 にある荷重 m_1 と，深さ l_2 ($\geq l_1$) にある荷重 m_2 の2個の外部節点の荷重をそれぞれ Δ だけ増減して $m_1+\Delta$，$m_2-\Delta$ とした場合を考える．この操作で，荷重の総和は変化しないが，荷重路長は $(l_2-l_1)\Delta$ だけ減少する．一方，任意の不均等な荷重の組は，総和が等しく値が均等な荷重の組から出発して，2個の外部節点間の荷重の増減を繰り返すことで実現できる．したがって，荷重の降順に根に近い外部節点を埋めることにより，荷重が均等な場合より荷重路長を小さくできる．このことは完全 w-分木を含む任意の拡張 w-分木についていえるので，補題2と組み合わせて補題3を得る． □

定性的には，総データ量（荷重の総和）が同じなら，各列の長さに偏りが大きいほどコスト（荷重路長）は

```
procedure partition(p);
begin
  if  pが拡張w-分木の外部節点  then return; /* 列が抽出された */
  /*pが指す内部節点の列集合をpの子の列集合に分解する*/
  for  pの列集合の全ブロックについて  do begin
    処理対象ブロックを id とする; /*pのブロックリストをたどる*/
    B⇐Mid; /*2次記憶の id ブロックを主記憶バッファBに読み込む*/
    ブロック番号 id を Free につなぐ;
    for  B中の各データについて  do begin
      データをそれが属するサブグループ用の主記憶のブロック (Wiw
      とする) に入れる;
      if  Wiw がいっぱい  then begin
        Free からブロック番号 (id' とする) を1つとってきて, Wiw の
        サブグループのブロックリストにつなぐ;
        Mid'⇐Wiw
        end
      end
    end;
  /*後処理と再帰呼出し*/
  for  pのすべての子sについて  do begin
    if  s用の主記憶のブロック Wiw が空でない  then begin
      Free からブック番号 (id' とする) を1つとってきて, Wiw のサ
      ブグループのブロックリストにつなぐ;
      Mid'⇐Wiw
      end;
    partition(s)
    end
end; /*partition*/
begin
  partition (root) /*拡張w-分木の根について呼出す*/
end
```

アルゴリズム 2 列集合の分割
Algorithm 2 Algorithm to partition column sets.

減少する傾向にあるといえる.

[定理2] アルゴリズム1で求まる分割過程で表形式データの転置を行った場合に要するコストの上界は,

$$E=d\left\{\lfloor\log_w m\rfloor + \frac{w}{w-1}(1-w^{\lfloor\log_w m\rfloor}/m)\right\}$$

である. ここで, w ($\geqq 2$) は主記憶のブロック数, d ($>w$) は全データのブロック数, m ($>w$) は列数である.

(証明) $d_i=d/m$ とおけば, 補題2および3から明らかである. □

$d\lceil\log_w m\rceil$ のほうがわかりやすい上界だが, 定理の式のほうがより精密である.

4. 可変長データの転置

前章では, 同一列のデータは同じ長さに制限していた. しかし, 前章のアルゴリズムで, 同一列のデータ長も異なる場合, すなわち表形式データのすべての要素が異なる可変長データの場合でも転置可能である. さらに, ブロック長とレコード長の関係も任意にできるが, 可変長ブロックのファイルは直接アクセスできないので, ブロック長は固定でなければならない. これらの場合, 拡張 w-分木の荷重となる各列のブロック数は, 各列の平均データ長から推定することになる.

3章では証明できた分割過程の最適性については, この場合あてはまらない. たとえば転置前の表形式データが前半 (上半分) と後半 (下半分) で列の長さの平均値に大きなかたよりがある場合を考えるとよい. このような場合, 全体を一度に転置するよりもむしろ前半と後半に分けて別々に分割過程を求めて転置したほうが転置コストを小さくできる. この考えをすすめると, 列の長さの平均値が得られるならできるだけ少ない行数ごとに転置を行ったほうがよいことになる. しかし, あまり小さすぎると転置後のブロックに空白が多くなるので, この単位は最小の長さの列が1ブロック分を占めるだけのレコード数以上を目安にする.

5. おわりに

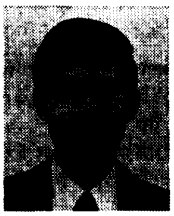
本論文では, 2次記憶に格納された異なる長さをもつ表形式データを効率よく転置する方法を提案した. そして, 2次記憶から主記憶へのブロック転送回数を手間の単位にとりコストを見積った. 表形式データは, はじめ行方向に連続してレコードが格納されている. 列集合の分割の繰り返しにより転置を行うが, 同一列内のデータ長が同じ場合は, この分割過程を最適にできることがわかった. そして, その際のコストの上界を求めることができた. また, 同一列内のデータ長も異なる場合についても同一アルゴリズムで能率よく転置可能であるが, 必ずしも最適にはならないことを述べた.

参考文献

1) Floyd, R. W.: Permuting Information in Idealized Two-Level Storage, in Miller, R. and Thatcher, J. (eds.), *Complexity of Computer Computations*, pp. 105-109, Plenum Press, New York (1972).

2) Tsuda, T. and Sato, T.: Transposition of Large Tabular Data Structures with Applications to Physical Database Organization (Part 1) Transposition of Tabular Data Structures, *Acta Inf.*, Vol. 19, No. 1, pp. 13-33 (1983).
3) Tsuda, T., Sato, T. and Tatsumi, T.: Minimizing Page Fetches for Permuting Information in Two-Level Storage (Part 1) Generalization of the Floyd Model, *J. Inf. Proc.*, Vol. 6, No. 2, pp. 74-77 (1983).
4) 佐藤, 津田：2階層記憶における効率のよいデータ並べかえアルゴリズム，情報処理学会論文誌, Vol. 27, No. 9, pp. 845-852 (1986).
5) Batory, D. S.: On Searching Transposed Files, *ACM Trans. Database Syst.*, Vol. 4, No. 4, pp. 531-544 (1979).
6) Knuth, D. E.: *The Art of Computer Programming, Vol. 1 Fundamental Algorithms*, pp. 399-405, Addison-Wesley, Reading, Mass. (1973).

(昭和62年6月15日受付)
(昭和63年6月24日採録)



佐藤 隆士（正会員）

昭和28年9月生．昭和53年岡山大学大学院修士課程(電子工学専攻)修了．工学博士．同年詫間電波高専助手．現在，同校助教授．データベース，記憶階層の研究に従事．電子情報通信学会，CAI 学会各会員．

津田 孝夫（正会員）

1932年生．1957年京都大学工学部電気工学科卒業．現職は京都大学工学部情報工学科教授．工学博士．現在の主要研究テーマは，メモリ階層間データ転送量の下界とそれによるアルゴリズムの最適化，ベクトル計算機のための自動ベクトル化と自動並列化，実時間オペレーティングシステムなど専用 OS の構成と実現法など．