個人用データベース構築ツール TRIAS の開発†

山本米雄†† 柏原昭博††
川岸圭介†† 塚本信宏†††

個人用データベース構築ツール TRIAS を論じる．個人用データベースは，個人的作業環境が重視される現在のコンピュータ利用形態では重要である．本システムの目的は，個人用データベースを構築するツールを開発することである．TRIAS は，パーソナルコンピュータなど低機能なコンピュータを対象として，データアクセスの多機能性，高速性と，ユーザ・システム間の対話性を実現する．データアクセスに関しては，データ構造に多分岐平衡木を用いて高速性を実現する．また，データを連想三重組の形で管理することで，種々のアクセス法を可能にする．三重組でデータを表現するのは，最も単純でありながら多様の事象を表現でき，しかもデータを高速にアクセスするためである．開発したシステムで実験を行った結果，TRIAS は個人用データベース実現に必要な要素を備えている構築用ツールであることが確認できた．

1. まえがき

コンピュータ利用形態において，個人的な環境を構築することはますます重要になっている．この個人的作業環境において，個人のためのデータベースは，利用価値が大きい．以後，我々は個人のために用いるデータベースを個人用データベースと呼ぶ．

個人用データベースは，従来の大型コンピュータで使用するデータベースとは一線を画する．従来のデータベースは複数の利用者が共有し，データの供給はシステムの作成者か専門家の集団が行う．それに対し，個人用データベースは一個人が専有し，個人の目的のために使用する．そして，データの管理・維持は利用者である個人が行う．

個人用データベースの特徴を以下に示す．

1) 扱うデータは，個人的であるため種類は多種多様である．
2) 個人によって目的は異なり，種々のデータアクセス法が要求される．
3) パーソナルコンピュータなどの比較的低機能なコンピュータを用いても高速なデータアクセスが要求される．
4) 対話的にデータアクセスが行える．

本論文では，以上の個人用データベースを構築するためのツールとして TRIAS (TRIple Associative System) を提案する．

TRIAS は，すべてのデータを連想三重組の形式でハンドリングし，データベースを構築・管理する．連想三重組は，(e, a, v) の形式をとり，図的に表現すると図1になる．e は物 (entity)を表し，v は値(value)を表す．a は属性 (attribute) であり，e から v への参照 (reference) を示す．

三重組でデータを表現する考えは，文献 1) の LEAP にさかのぼることができる．LEAP では，AI 的研究指向から，データを2項関係で表現するために三重組を考え，その三重組を連想記憶装置上におくものであった[2]．一方，TRIAS は，個人用データベースの観点から記憶されたデータの実用的利用法に重点をおく．LEAP と TRIAS の具体的な違いは，連想の概念と，実現法の2点にある．連想は，LEAP では e から v への参照を意味し，TRIAS では図2に示すように順次 e_1 から $v_1, \cdots, v_k$ への遷移を指す．実現法は，LEAP では，三重組 (e, a, v) を (a, e), (e, v), (v, a) とし，各々の組の値をキーとしてハッシュ法によりテーブルに格納する．TRIAS では，個人用データのダイナミックな性質に着目し，データアクセスの高速性を実現するために，三重組の集合に対して，多分岐平衡木構造を用いる．

TRIAS では，データベース部に ML-tree (Multi-Linked-tree) 構造を用いた．ML-tree でのディスクアクセス回数は，データ数により最大値が決まり，最大時間を一定時間内に抑えることができる．

以下，ML-tree, 連想三重組，TRIAS, TRIAS の機能，TRIAS の評価について論じる．

† A Tool for Construction of Personal Database: TRIAS by YONEO YAMAMOTO, AKIHIRO KASHIHARA, KEISUKE KAWAGISHI (Department of Information Science, Faculty of Engineering, Tokushima University) and NOBUHIRO TSUKAMOTO (School of Medicine, Keio University).

†† 徳島大学工学部情報工学科
††† 慶応義塾大学医学部

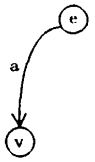

図 1 連想三重組
Fig. 1 Associative triple.

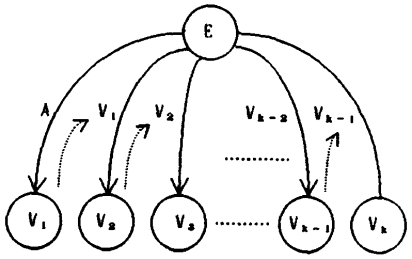

図 2 TRIAS の連想
Fig. 2 Association of TRIAS.

2. ML-tree

多分岐平衡木[3]の代表的なものとして，Bayer らによって提唱された B-tree[4] や，その後 B-tree に改良を加えた B⁺-tree などがある．我々は，B⁺-tree でのデータの登録・削除時の無駄を解消するため，改良を加えて ML-tree を開発した．

2.1 ML-tree 構造

改良点は，以下の2点である．

1. すべての節に親節へのポインタを持たせた．
2. 各レベルにおいて節を双方向にリンクした．

［1の改良について］ 親節へのポインタは，キーの登録，または削除時に親節をアクセスする場合に有効となる．B⁺-tree では，スタックなどを用いて，キーの検索時にたどった節を履歴し，その履歴を用いて親節をアクセスする．しかし，ML-tree では親節へのポインタを用いてアクセスできるため，たどった節の履歴をとる必要がない．

［2の改良について］ 文献5) の B⁺-tree では，葉レベルのみ昇順方向に節をリンクしているが，ML-tree では，すべてのレベルにおいての節を双方向にリンクした．この改良は，キーの削除による節の併合，またはキーの流入時に，隣接節をアクセスする場合に有効となる．B⁺-tree では，隣接節をアクセスするためには親節を経由しなければならず，最悪の場合根までさかのぼる必要がある．しかし，ML-tree では，隣接節がリンクされているために，親節を経由する必要がない．したがって，節の併合，キーの流入のアルゴリズムが簡単になり，かつ高速に行うことができる．

以下に，次数 m の ML-tree の特徴を示す[6]．

①根を除くすべての節は，少なくとも $[m/2]$ 個の子を持つ．
②根は，少なくとも2個の子を持つ．
③k 個の子を持つ節は，k 個のキーのコピーを持つ．
④すべてのキーは，葉に含まれる．
⑤すべての葉は，同一レベルに現れる．
⑥葉以外の節内のキーは，直後の分岐ポインタで作られる部分木中の最大キーに等しい．
⑦レコード本体は節内に含まれず，葉からレコード本体へのポインタとしてリンクされる．
⑧親子節，隣接節は互いにポインタでリンクされる．
⑨各レベルの最右節のみ節内の最大キーより大きいキーへのポインタを持つ．

以上の特徴の中で，①，⑧が B⁺-tree と異なる特徴である．

2.2 データアクセス

ML-tree では，データの登録と削除のアルゴリズムは双対になっている．そこで，登録のアルゴリズムのみを図3に示す．(検索に関しては，簡単であるので省略する．)

図3では，まず登録すべきキー (insert_key) を検索し，キーの存在を調べる．キーが存在しなければ，実際にキー (insert_item) を ML-tree へ登録 (putnode) する．putnode では，登録すべき節 (current_node) の状態を調べる．そして，節に余裕がある場合 (current_node^. num_data<m) は，その節に登録 (e[i+1]:=insert_item) する．余裕がない場合は，節の分割 (split) を行う．分割した節が根であれば，新たに根 (new (new_root)) を割り当てる．分割後，左節の最大キーを親節に対して再び登録 (putnode (current_node^. e[num_data]) する．以後再帰的に処理を行う．

3. 連想三重組

3.1 定　　義

連想三重組は，以下の3成分から構成される．

1) 物 (entity)
2) 属性 (attribute)
3) 値 (value)

連想三重組の表現を以下のように定義する．

(e, a, v)=(物，属性，値)[7]

```
type ptr = ^node;
     item = record key: string;  p: ptr  end;
     node = record num_data: 0..m; parent,left,right,mostright: ptr;
                   e: array[1..m] of item  end;
procedure insert(insert_key: string; var ret:boolean);
  var current_node, new_node, new_root : ptr;
      i: integer;  insert_item : item;
  procedure putnode(insert_item: item);
    begin
      with current_node^ do
        begin
          if num_data < m then
            begin
              i := num_data;
              while e[i].key > insert_item.key and i > 0 do
                begin
                  e[i+1] := e[i];  i := i -1
                end;
              e[i+1] := insert_item;   num_data := num_data +1;
            end else
            begin
              new(new_node);
              split(current_node, new_node, insert_key);
              if parent = nil then
                begin
                  new(new_root);
                  new_root^.parent, new_root^.left,
                  new_root^.right, new_root^.mostright <- nil
                  new_root^.num_data := 1;
                  new_root^.e[1] := e[num_data]
                end else
                begin
                  current_node := parent;  putnode(e[num_data])
                end
            end
        end
    end;
begin
  if search(insert_key, current_node, ret) then ret := false
  else
    begin
      insert_item.key := insert_key; insert_item.p := nil;
      putnode(insert_item);   ret := true
    end
end;
```

図 3 登録のアルゴリズム
Fig. 3 Algorithm for data insertion in ML-tree.

連想三重組の集合に対して，以下の関係を定義する．

$$R(E, A, V)=\text{true if } (E, A, V)\in\{(e, a, v)\}.$$
$$=\text{false if } (E, A, V)\notin\{(e, a, v)\}. \quad (3.1)$$

$$R(E, A, *)=\{(e, a, v)|e=E,\ a=A\}. \quad (3.2)$$

$$R(E, *, V)=\{(e, a, v)|v=V,\ e=E\}. \quad (3.3)$$

$$R(*, A, V)=\{(e, a, v)|a=A,\ v=V\}. \quad (3.4)$$

$$R(E, *, *)=\{(e, a, v)|e=E\}. \quad (3.5)$$

$$R(*, A, *)=\{(e, a, v)|a=A\}. \quad (3.6)$$

$$R(*, *, V)=\{(e, a, v)|v=V\}. \quad (3.7)$$

$$R(*, *, *)=\{(e, a, v)\}. \quad (3.8)$$

ここで，E, A, V は定数，$*$ は変数，e, a, v は束縛変数である．

式(3.1)は，連想三重組 (E, A, V) がデータベースに存在するかどうかの真偽を判定する．式(3.8)は，データベースに存在するすべての連想三重組を表現している．その他の関係は，例えば式(3.2)の場合は，物が E かつ属性が A であるすべての連想三重組の集合を表現している．

3.2 内部表現

TRIAS 内部では，さまざまな検索法を実現するために三重組をコード化し，中間コードを作成する．中間コードは，以下の3種類である．

Key1='1'+ne+na+nv.

Key2='2'+na+nv+ne.

Key3='3'+nv+ne+na.

ここで，ne, na, nv は三重組の各成分 e, a, v のレコード番号（4.1節参照）である．また，＋はビット列の連結を意味する．

3.3 意　味

連想三重組によるデータ表現は，人間が理解する上で最もプリミティブな形態であり，しかもあらゆるデータを表現できる．

連想三重組によるデータ管理は，以下の利点を持つ．

1) 柔軟なデータ表現が可能である．
2) データハンドリングが容易である．

TRIAS での連想三重組の特徴は，属性をユーザが自由に設定できる点にある．これは，物に対する多様な表現が可能であることを意味する．さらに，属性に対する値が複数個存在することを許している．これにより，属性に対して値が複数個存在する世界までも表現可能としている．このような柔軟なデータ表現は，個人用データベースにとって必要不可欠である．

また，データ管理の面から促えると，連想三重組の単純さは，データハンドリングを容易なものとする．

4. TRIAS

TRIAS は，以下の特徴を有する．

①処理データは，多種多様である．

②さまざまなデータアクセス法がある．

③高速なデータアクセスが可能である．

TRIAS のデータ形式が，連想三重組であることから，①，②がいえる．また，データベースのデータ構造に ML-tree を用いている点から，③がいえる．③

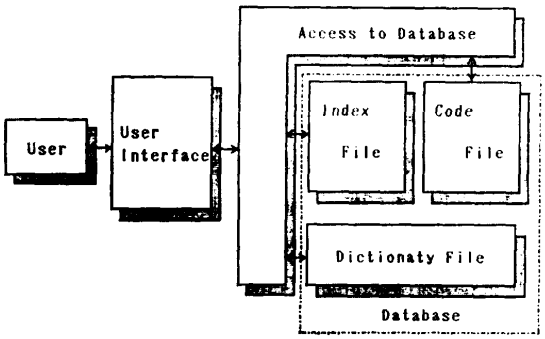

図 4 TRIAS の構成
Fig. 4 TRIAS configuration. (TRIAS is composed of database, access to database, and interface.)

は，対話的にユーザがデータアクセスを行う上で，重要な一要素である．

TRIAS の構成を図 4 に示す．TRIAS は，以下の3つの部分から構成される．

1) データベース部
2) データベースアクセス部
3) ユーザインタフェース部

データベース部は，データとして連想三重組を管理している．ユーザインタフェース部では，ユーザの要求を対話的に処理する．データベースアクセス部では，ユーザインタフェースから得たユーザの要求を変換し，必要なデータをデータベースに対してアクセスする．そして，アクセスされたデータを再変換し，ユーザインタフェースへ返す．

4.1 ファイル構成

データベース部は，以下の3つのファイルから構成される．

a) 索引ファイル (Index file)
b) コードファイル (Code file)
c) 辞書ファイル (Dictionary file)

索引ファイルは，三重組の各成分を格納するための ML-tree 構造のファイルである．ML-tree のキーは三重組の各成分であり，葉レベルにおける情報本体へのポインタは，辞書ファイルのレコードを指し示す．

コードファイルは，中間コードを格納するための ML-tree 構造のファイルである．ML-tree のキーは，中間コードである Key1～Key3 (3.2 節参照) である．これらの中間コードの各レコード番号は，辞書ファイルのレコードを指し示す．また，中間コードには三重組の各成分の関連性の維持という意味がある．コードファイルの葉レベルにおける情報本体へのポインタは，無意味である．

辞書ファイルは，三重組の各成分を格納するランダムアクセス可能なファイルである．このファイルの各レコードは，索引ファイルの対応するキーとポインタでリンクされている．ML-tree では，キーとそのキーの情報本体は異なるのが一般的であるが，TRIAS では，索引ファイルのキーとそれに対応してリンクしている辞書ファイルのレコードの内容は同一である．

4.2 ファイルアクセス

"*" による検索，登録，削除時に，上述した3つのファイルがどのようにアクセスされるかを図 5 に示す．

"*" による検索 (tri_search) では，指定 (input) した各成分から検索用中間コード (s_key) を作成 (set_flag, set_rec) する．この検索用中間コードを，コードファイルで部分検索 (tree_gsearch) する．そして，検索用中間コードとコードファイルの中間コード (tmp) がマッチング (tricmp) する限り，以下の処理を繰り返す (tree_next)．検索された中間コード (tmp) の各成分のレコード番号 (tmp. eavrec[ ]) を用いて，辞書ファイルから三重組の各成分を検索 (dic_get) する．

登録 (tri_insert) では，指定した成分が索引ファイルに存在するかを調べる (inkey)．存在する場合は，そのレコード番号を求める．存在しない場合は，索引ファイルに登録 (tree_insert(ndxfile,…)) し，そのレコード番号を求めて辞書ファイルへの登録 (dic_put) を行う．次に，レコード番号から3種の中間コード (key1, key2, key3) を作成 (make_qkey) し，コードファイルへの登録 (tree_insert (codefile,…)) を行う．

削除 (tri_delete) は，各成分を索引ファイルで検索 (input) し，レコード番号を得て3種の中間コードを作成 (make_qkey) する．そして，コードファイルからこの3種の中間コード (key1, key2, key3) を削除 (tree_delete) する．

5. TRIAS の機能

TRIAS の機能を以下に示す．

1. 検索，2. 登録，3. 削除，4. 連想，5. 単語検索 (先頭)，6. 単語検索 (一部)，7. ファイルからの読み込み，8. ファイルへの書き出し

連想，単語検索は，TRIAS 特有のものである．

連想は，図2で示したように，物を固定して，関連

```
const E = 1; A = 2; V = 3;
  codefile = 0;    ndxfile = 1;   dicfile = 2;
type qkey =    {Difinition of Middle-code.}
       record flag: char;  eavrec: array[1..3] of integer  end;
var key1, key2, key3: qkey;
    key: array[1..3] of string;  rec: array[1..3] of integer;
procedure input(type: integer);
  begin
    read(key[type]);
    if key[type] = '*' then rec[type] := 0
    else
       tree_search(ndxfile, key[type], rec[type], h)
  end;

{Search by "*"}
procedure tri_search;
var s_key, tmp: qkey;  h: boolean; rec: integer;
begin
  input(E); input(A); input(V);
  set_flag(s_key);    set_rec (s_key);
  tmp := s_key;  tree_gsearch(codefile, tmp, rec, h);
  while tricmp(s_key, tmp, n) and h do
    begin
      dic_get(key[E], tmp.eavrec[E]);
      dic_get(key[A], tmp.eavrec[A]);
      dic_get(key[V], tmp.eavrec[V]);
      writeln(key[E], key[A], key[V]);
      tree_next(codefile, tmp,  h)
    end
end;
{Insertion}
procedure tri_insert;
  var dmy: integer;
 procedure inkey(type: integer);
  var h: boolean;
  begin
    read(key[type]);
    tree_search(ndxfile, key[type], rec[type], h);
    if not h then {Not existence}
      begin
        tree_insert(ndxfile, key[type], rec[type]);
        dic_put(key[type], rec[type])
      end
  end;
begin
  inkey(E); inkey(A); inkey(V);
  make_qkey();
  tree_insert(codefile, key1, dmy);
  tree_insert(codefile, key2, dmy);
  tree_insert(codefile, key3, dmy)
end;
{Deletion}
procedure tri_delete;
  var h: boolean;
begin
  input(E); input(A); input(V);
  make_qkey();
  tree_delete(codefile, key1);
  tree_delete(codefile, key2);
  tree_delete(codefile, key3)
end;
```

図 5 ファイルアクセスアルゴリズム
Fig. 5 Algorithm for files access. (Files are index file, code file, and dictionary file.)

ある三重組を遷移的に検索する機能である．この機能により，複雑な事象を表現できる．また，ユーザにとっては，曖昧な検索要求を行える支援的な機能である．

単語検索では，三重組の各成分の一部分（接頭語，中間語，接尾語）を指定することで，その三重組を検索できる．ユーザにとっては，成分の指定が曖昧である場合，語句の一部分でデータを検索できる機能は有効である．一般に，検索語の制限が小さければ，ユーザの検索の自由度はより向上する．TRIAS の単語検索は，そのような柔軟性のある検索法を実現している．

5.1 検　　索

検索は，三重組の各成分を指定して三重組の有無を確認する場合と，“*”記号を用いて行う場合がある．

“*”による検索は，以下の7種類に分けることができる．

group A : $(E, A, *)$ $(E, *, *)$ $(*, *, *)$
group B : $(*, A, V)$ $(*, A, *)$
group C : $(E, *, V)$ $(*, *, V)$

検索される三重組の集合は，3.1 節で述べたそれぞれの関係の“*”の位置が対応する (3.2)～(3.8) のいずれかである．

“*”による検索では，検索用中間コードを作成する必要がある．検索用中間コードは，“*”部分のレコード番号を0として，3種の中間コード Key1～Key3 から，1つを選択して作成する．group A は Key1 を，group B は Key2 を，group C は Key3 を選択する．この選択は，検索用中間コードをコードファイル内で部分検索するため，キーの先頭から一致させる必要があることによる．部分検索とは，キーの先頭が指定したキーとマッチングするキーを検索することである．

5.2 登　　録

三重組の登録について，例を挙げて説明する．

以下の三重組を登録する場合を考える．

（りんご，色，赤い）

索引ファイルを検索した結果，3成分とも検索失敗したとする．まず，索引ファイルに3成分を登録し，以下のようなレコード番号を得た．

りんご　　レコード番号　23
色　　レコード番号　24
赤い　　レコード番号　25

次に，得られたレコード番号によって以下のように辞書ファイルに成分の登録を行う．

第 23 レコードに　　りんご
第 24 レコードに　　色

第 25 レコードに 赤い

最後に，得られたレコード番号から中間コードを以下のように作成し，コードファイルへの登録を行う．

Key1='1'+23+24+25.

Key2='2'+24+25+23.

Key3='3'+25+23+24.

5.3 削 除

削除では，削除する三重組に対して，3種の中間コードをコードファイルから削除して3成分の関連性を消去する．三重組の各成分は，索引・辞書ファイルから消去しない．これは，その成分を他の三重組が使用している場合があるためである．しかし，削除の結果どの三重組も使用していない成分が索引・辞書ファイルに存在する可能性がある．したがって，削除を頻繁に行うと索引・辞書ファイルの使用効率が低下することが考えられる．しかし，5.6 節の機能を用いれば，使用効率の低下を回避できる．

5.4 連 想

2つの三重組として，

1) (E_1, A_1, V_1) 2) (E_1, V_1, V_2)

を考えると，2つの三重組の物が等しく，かつ1）の三重組の値と2）の三重組の層性が等しい．この時，2つの三重組は1)から2)に連結されているという．TRIAS では，連結されている三重組を検索する機能を連想という．

また，2)の三重組が複数個存在したときには，利用者が連想する三重組を選択できる．

連想に対する関係を以下のように定義する．

$$R_{as}(e_1, a_1, v_1)=\{(e_1, v_i, v_{i+1})\mid i=1, 2, \cdots, n-1\}.$$

ただし，e_1, a_1, v_i $(i=2, 3, \cdots n)$ は定数である．

連想機能の内部処理は，1)の三重組の中間コードを使用して，

Key='1'+ne_1+nv_1.

を作成する．そして，このコードをコードファイル内で部分検索することにより，2)の三重組を連想することができる．ただし，ne_1, nv_1 は，各々 E_1, V_1 のレコード番号である．

例として，

(りんご，色，赤い)

(りんご，赤い，値段が高い)

(りんご，値段が高い，おいしい)

の3つの三重組が登録されているとする．まず，"*" 記号を用いて先頭の三重組を検索する．

(りんご，色，*)

>(りんご，色，赤い)

次に，連想機能を用いれば，以下のようになる．

$ 連想

(りんご，赤い，値段が高い)

$ 連想

(りんご，値段が高い，おいしい)

$ 連想

ほかに連結されたデータはない．

5.5 単語検索

"*" による検索では，"*" 以外の文字はすべて一致していなければならないため，指定した単語を含む三重組の検索はできない．そこで，単語検索機能を用意し，指定した単語を含む三重組の検索を可能にした．単語検索には，先頭，一部の2種類ある．

単語検索・先頭は，指定した単語が三重組の各成分のいずれかの接頭語と一致する三重組を検索する機能である．内部処理では，指定した単語を索引ファイルで部分検索する．そして，得られた辞書ファイルへのポインタ (rec) から，以下の3種の中間コードを作成し，コードファイル内で部分検索を行う．

KeyA='1'+rec. KeyB='2'+rec.

KeyC='3'+rec.

部分検索の結果，得た中間コードに基づき目的の三重組を検索できる．

単語検索・一部は，指定したすべての単語を三重組のいずれかの成分の一部として含む三重組を検索する機能である．内部処理では，コードファイルに存在するすべての Key1 に対して，辞書ファイルへのアクセスを通し，指定したすべての単語を含むかどうかを調べる．

5.6 ファイルアクセス

TRIAS では，ファイルの読み込み，ファイルへの書き出しが行える．ファイルの読み込みは，三重組をデータとするファイルの読み込みを行う．この機能は，大量の三重組を登録するときに有効である．ファイルへの書き出しは，必要な三重組だけをデータとするファイルの作成を行う．削除が重なって索引・辞書ファイルに多くの無駄なキーが存在する場合は，この機能を用いてデータベースの再構築を行うことができる．

6. TRIAS の評価

次に，TRIAS に対する評価を行う．

表1 は，三重組の成分に重複がない場合の評価であ

表 1 TRIAS の評価（重複のない場合）
Table 1 Evaluation of TRIAS. (Case of no items overlap of associative triples.)

(単位：10 ms)

	データ数	1,000	2,000	3,000	4,000	5,000	6,000	7,000	8,000	9,000	10,000
	(E, A, V)	22	24	28	28	27	32	33	34	33	32
	$(*, A, V)$	28	29	31	32	31	37	38	38	38	38
検	$(E, *, V)$	29	29	32	32	32	38	38	39	39	38
	$(E, A, *)$	17	17	19	22	21	20	22	22	22	21
	$(*, *, V)$	28	30	32	32	32	37	38	38	38	38
索	$(E, *, *)$	17	18	19	21	21	20	22	22	22	22
	$(*, A, *)$	19	19	19	20	20	20	22	22	22	22
	$(*, *, *)$	3	4	4	4	4	4	4	4	4	4
	登録	70	73	70	73	83	89	84	92	88	91

表 2 TRIAS の評価（3割重複の場合）
Table 2 Evaluation of TRIAS. (Case of 30 percent of items overlap of associative triples.)

(単位：10 ms)

	データ数	1,000	2,000	3,000	4,000	5,000	6,000	7,000	8,000	9,000	10,000
	(E, A, V)	23	26	26	26	26	33	31	33	34	34
	$(*, A, V)$	28	33	34	33	33	41	37	38	42	42
検	$(E, *, V)$	29	32	34	33	33	41	38	40	42	42
	$(E, A, *)$	21	22	22	22	23	24	26	24	24	24
	$(*, *, V)$	29	33	34	33	32	41	37	38	42	42
索	$(E, *, *)$	18	20	21	22	23	24	26	24	24	25
	$(*, A, *)$	20	22	21	23	24	24	25	24	25	26
	$(*, *, *)$	3	3	3	4	4	4	4	4	4	4
	登録	59	64	61	65	68	73	68	70	73	81

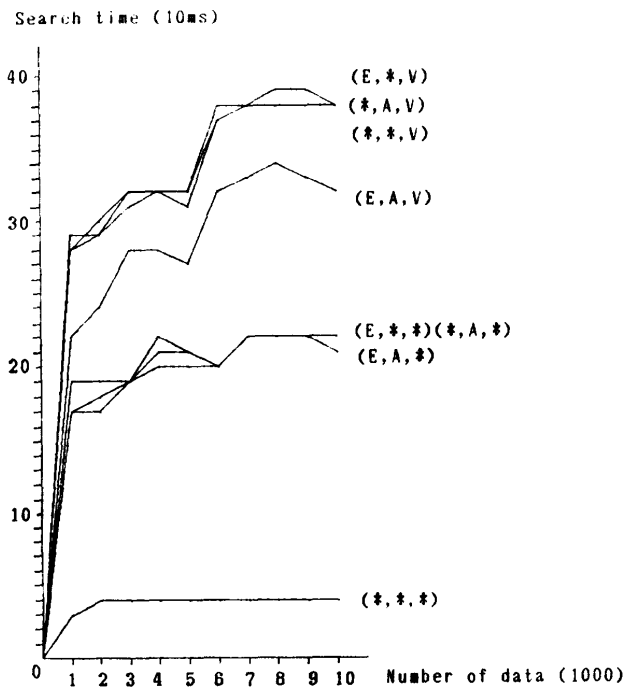

図 6 検索速度（重複のない場合）
Fig. 6 Searching time. (Case of no items overlap of associative triples.)

り，表 2 は，三重組の成分が 3 割重複している場合の評価である．登録，検索時間は，ユーザが値（value）を指定して最初の三重組を登録，検索するまでの時間であり，各データ数において 100 件のデータをアクセスしたときの 1 件あたりの平均値である．削除時間に関しては，登録時間とほぼ同等であるので省略した．

図 6，図 7 は，各々表 1，表 2 の検索速度をグラフ化したものである．図 8，図 9 は，各々表 1，表 2 の登録速度をグラフ化したものである．

以下に評価結果を示す．

〔検索速度に関して〕

1) $(*, *, *)$ がデータ数によらずほぼ一定で他の検索法と比較して非常に高速である．
2) (value) が，"*" の方が定数であるときより検索時間が小さい．
3) (E, A, V) の検索時間は，値（value）が "*" と定数である場合の中間になる．

〔登録速度に関して〕

3 本の傾向線が現れる．

〔検索時間に関して〕 1) は，(*, *, *) はコードファイルの先頭のキーから検索を始めるため，コードファイルのディスクアクセス回数が1回で済むことによる．2) は，ユーザからの値 (value) の入力が "*" でない場合，値を指定した後，索引ファイルにおいてその値のレコード番号を検索するため，その時間だけ検索時間は大きくなるためである．3) は，(*E*, *A*, *V*) の検索が，コードファイルにおいて存在するかどうかの確認が行われるだけで，辞書ファイルへのアクセスが必要でないため，値 (value) が定数である "*" 検索より検索時間が小さい．

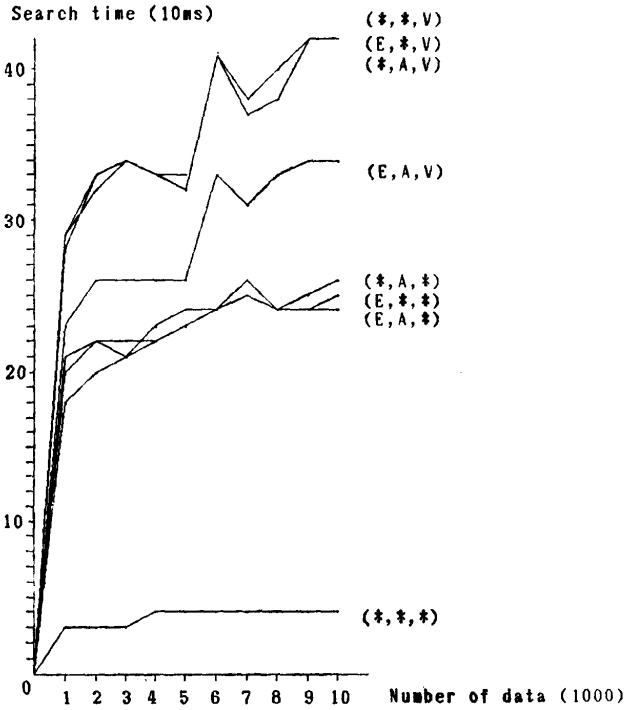

図 7 検索速度（3割重複の場合）
Fig. 7 Searching time. (Case of 30 percent overlap of items of associative triples.)

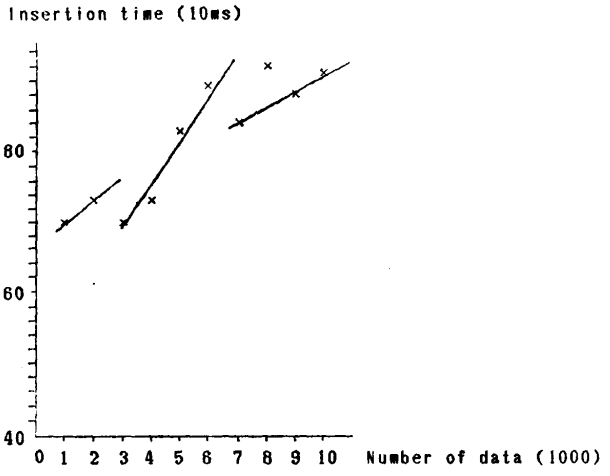

図 8 登録速度（重複のない場合）
Fig. 8 Inserting time. (Case of no items overlap of associative triples.)

〔登録時間に関して〕 データ数が2,000から3,000の間，6,000から7,000の間において，索引ファイルの ML-tree のレベルが1だけ増加した．レベルが増加する以前，つまりデータ数が 2,000，6,000 付近では，節の使用率が高くなり，データ登録時に節の分割が頻繁に起こる．そのため，レベル増加直前では，登録時間が大きい．また，レベル増加直後，つまりデータ数が 3,000，7,000 の時は，節の使用率が低いため，データ登録時に節の分割は起こりにくい．したがって，データ登録は，葉へのデータ挿入だけとなり，登録時間は，レベル増加直前に比べて小さくなる．

以上の結果を総括すると，TRIAS は1万件のデータに対して，1秒以下のデータアクセスを保証することが確認できた．これは，TRIAS が個人用データベースを構築・管理するのに適していることを示す．

7. むすび

本論文では，個人用データベース構築ツール TRIAS を提案した．

TRIAS では，連想三重組の形式でデータを管理する．このため，柔軟なデータ表現が可能であり，様々な角度からデータをアクセスする方法を実現できる．

また，TRIAS のデータベース部には，ML-tree を用いている．このため，個人用データの動的な性質に対処でき，高速データアクセスが実現できる．

以上から，TRIAS は，個人用データベースを構築するためのツールとして有効であるといえる．

応用としては，TRIAS 自体を核として，利用者の使用目的に応じてインタフェースを構築することが考えられる．一例として，CAI への応用がある．CAI では，学習者の個別性を重視し，かつ学習者が能動的に学習するという

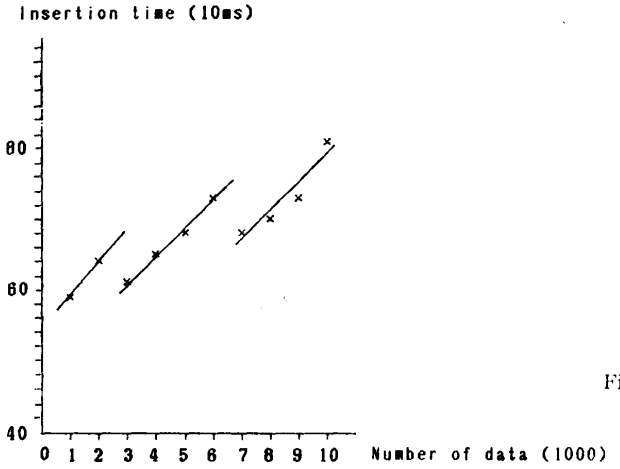

図 9 登録速度（3割重複の場合）
Fig. 9 Inserting time. (Case of 30 percent overlap of items of associative triples.)

ことが重要である．そこで，TRIAS を用いて，学習者の個別的・能動的学習環境を構築すれば，有用な CAI システムが実現できる．TRIAS の CAI に応用したシステムについては，別に報告する．

参 考 文 献

1) Feldman, J. A. and Rovner, P. D.: An ALGOL-based Associative Language, *Comm. ACM*, Vol. 12, No. 8, pp. 439-449 (1969).
2) 有澤 博：データベース理論，情報処理叢書 2, p. 91, オーム社，東京 (1980).
3) Knuth, D. E.: *Sorting and Searching*, *The Art of Computer Programming Vol. 3*, p. 722, Addison-Wesley (1973).
4) Bayer, R. and McCreight, E.: Organization and Maintenance of Large Ordered Indexes, *Acta Informatica*, Vol. 1, No. 3, pp. 173-189 (1972).
5) Maruyama, K. and Smith, S. E.: Analysis of Design Alternative for Virtual Memory Indexes, *Comm. ACM*, Vol. 20, No. 4, pp. 245-254 (1977).
6) 山本米雄，柏原昭博：多分岐平衡木 ML-tree における一括削除方式，電子情報通信学会論文誌，Vol. J 72-D, No. 3, pp. 140-143 (1989).
7) 山本米雄：ヒューマンフレンドリーなパーソナルデータベース：TRIAS，情報処理学会ヒューマンフレンドリーなシステムシンポジウム報告集，pp. 11-14 (1986).

（昭和 63 年 3 月 3 日受付）
（平成 元 年 4 月 11 日採録）

山本 米雄（正会員）

昭和 20 年生．昭和 44 年大阪大学工学部通信工学科卒業．昭和 49 年同大学院博士課程修了．工学博士．同年徳島大学工学部情報工学科助手．現在同大助教授．昭和 54～55 年，57 年米国イリノイ大学の CERL で PLATO の研究に従事．CAI に興味を持っている．著書「新アマチュアプログラミング」（共著．日本ソフトバンク）など．CAI 学会，電子情報通信学会，IEEE などの会員．

柏原 昭博（学生会員）

昭和 39 年生．昭和 62 年徳島大学工学部情報工学科卒業．平成元年同大学院修士課程修了．現在，大阪大学大学院博士課程在学中．人工知能，特に知的 CAI の研究に従事．人工知能学会，CAI 学会各会員．

川岸 圭介

昭和 38 年生．昭和 61 年徳島大学工学部情報工学科卒業．昭和 63 年同大学院修士課程修了．現在，四国システム開発(株)勤務．プログラム言語，CAI に興味を持つ．

塚本 信宏（学生会員）

昭和 33 年生．昭和 61 年徳島大学医学部医学科卒業．慶應義塾大学大学院医学研究科博士課程在学中．現在，同大学工学部電気工学科計算機科学専攻に出張研究中．医用画像処理，知識工学，機械学習に興味を持つ．ACM, AAAI, ソフトウェア科学会各会員．