距離索引 *MI-tree*

石川雅弘[†1] 能登谷淳一[†2]
陳漢雄[†3] 大保信夫[†4]

距離の定義されたデータ空間における効率的な近傍検索を支援するための索引木，*MI-tree(Metric Index Tree)* を提案する．MI-tree は動的な更新およびディスクへの格納を考慮に入れたページ化された索引木であり，検索時の距離計算回数および I/O コストの削減を目的としている．MI-tree は距離空間中の点を一次元空間にマッピングし数値キーを割り当て，そのキーにより B+tree を利用する．また空間の分割基準として二点からの距離の差を利用し，それによる解候補の絞り込みを提案している．本稿では MI-tree の構造と挿入，近傍検索の手順および静的構築法を示し，検索時の距離計算回数とアクセスノード数を実験により評価する．

A metric index *MI-tree*

Masahiro Ishikawa ,[†1] Junichi Notoya ,[†1] Hanxiong Chen [†3]
and Nobuo Ohbo[†4]

We propose a metric index *MI-tree(Metric Index tree)* which supports efficient near neighbour searches in an arbitrary metric space. MI-tree is a paged index tree and allows dynamic updates. Its main purpose is to reduce the number of distance calculations and I/O accesses at search time. MI-tree maps each point in metric space into one-dimensional space. By such mapping, each point is assigned with a numeric key and inserted into B+tree. The tree exploits the difference between two distances from two referential points to divide the target metric space. This value is also used for candidate filtering in novel fashion. In this article, we present its structure, insertion/retrieval and static construction procedures for MI-tree. We show experimental results on synthetic data and comparison with MVP-tree and GNAT.

1. 背　景

近年，文書をはじめ音声や画像などのマルチメディアデータの利用と蓄積が進んでおり，その効率的な管理および処理方法の研究が活発になっている．ユーザの要求に迅速に応えるためには，大量のデータからユーザが興味を持つ部分を効率的に抽出する事が必要であり，特にあるデータが示された時それに類似したデータを取り出すための類似検索への要求は大きい．

†1 筑波大学 博士課程工学研究科
Doctoral Program of Engineering, University of Tsukuba, Japan
†2 秋田県立大学電子情報システム学科
Akita Prefectural University, Department of Electronics and Information Systems, Japan
†3 つくば国際大学 産業情報学科
Institute of Industorial Information, Tsukuba International University, Japan
†4 筑波大学 電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba, Japan

文書や画像データなどでは類似度を測る目的でデータの多次元特徴ベクトルを利用する事が多いため，R-tree などの多次元索引木を利用する事も考えられる[9)6)]．しかし多次元索引木では暗に距離が L_2 距離 (ユークリッド距離) である事を前提としているため，各次元の相関を考慮に入れたヒストグラム距離などを採用している場合の利用は困難である[10)]．文字列間の類似度として利用される InsDel 距離，Edit 距離なども同様である．

このような背景から，距離公理の成立のみを前提とし，データ間の相対距離のみに基づいた距離索引の研究が行なわれており，GNAT[7)], MVP-tree[3)] などの索引木が提案されている．しかしこれらの距離索引は構築開始時点に全データが参照可能な場合にのみ適用できる静的手法であり，データの動的な更新は考慮されていない上，木のディスクへの格納も考えられていない．動的更新にも対応した距離索引木としては R-tree に類似した構造の M-tree[5)] が提案されており，ページ化されディスクへの格納も考慮されている[3)]．しか

し M-tree の各ノードに対応する部分空間は (超) 球であり空間の重複が起き易い．またデッドスペースも大きくなりがちであり，検索効率の低下要因と考えられ，静的手法に比べると検索時の距離計算回数が多くなる [10]．

本稿では動的な更新にも対応しディスクへの格納も考慮した距離索引木，*MI-tree(Metric Index tree)* を提案する．MI-tree の特徴は空間の分割法と解候補の絞り込み法にある．解候補の絞り込みは二点からの距離の差に基づいており，データ空間もこの値に従って分割される．またデータにはこの値に基づいた数値キーを割り当て，これにより順序索引である B+tree の利用を可能としている．B+tree を利用する事で，ディスクへの格納が可能となるだけでなく同時実行制御などの B+tree 上で開発されてきた様々な技術をそのまま利用できるというメリットが得られる．MI-tree の目的はディスクへの格納と動的更新を可能としながら静的手法並の距離計算回数での検索を実現する事である．

本稿は以下のように構成されている．2 で距離索引の説明を行ない，3 で MI-tree の構造と挿入・近傍検索の手順を述べる．4 では MI-tree の静的構築法を示す．5 で 合成データにおける実験結果を示し，6 でまとめを述べる．

2. 距離索引

データ集合 O 上に以下の条件 (距離公理) を満す関数 $d : O \times O \to \mathcal{R}$ が定義されているとき，$d(\cdot,\cdot)$ を O 上の距離関数，$\langle O, d(\cdot,\cdot)\rangle$ を距離空間 (metric space) と呼ぶ:

$$\forall x, y \in O; \quad d(x,y) = d(y,x) \tag{1}$$
$$\forall x \in O; \quad d(x,x) = 0 \tag{2}$$
$$\forall x, y \in O; x \neq y \to 0 < d(x,y) < \infty \tag{3}$$
$$\forall x, y, z \in O; d(x,z) \leq d(x,y) + d(y,z) \tag{4}$$

距離索引 (*metric index*) とは，距離公理の成立のみを前提とし，距離空間中のデータに対し距離に基づく検索を支援する索引である．一般的に文書や画像データ間に定義される距離 (類似度) は属性間の相関なども考慮に入れられ，計算コストが大きくなる傾向にある．そのため距離索引の主要な目的は検索時に必要な距離計算回数を削減する事にある．また，距離計算回数がデータ数 N に対し充分小さければ距離計算回数のオーダが $O(N)$ であっても距離索引のメリットは大きい．

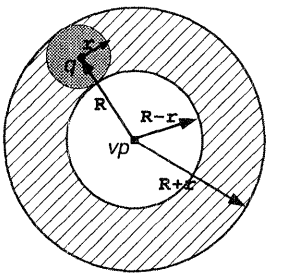


図 1 距離索引における解候補の絞り込み
Fig. 1 Candidate filtering used by metric indices

2.1 近傍検索

距離索引により支援される検索の代表的なものに r-近傍検索，k-最近傍検索がある．それぞれ次のように定義される．

定義 2.1 (r-近傍検索)
データ q と半径 r が与えられた時，q の r-近傍に含まれるデータの集合

$$NN_r(q,r) = \{o \mid o \in O \wedge d(q,o) \leq r\}$$

を求める検索を r-**近傍検索** (*r-neighbour search*, r-N 検索) と呼ぶ．□

定義 2.2 (k-最近傍検索)
データ q と正整数 k が与えられた時，q からの距離が最も小さい k 個のデータの集合

$$NN_k(q,k) = \{o \mid o \in O \wedge (\exists r; \forall r' < r; \#NN_r(q,r') < k \wedge o \in NN_r(q,r))\}$$

を求める検索を k-**最近傍検索** (*k-nearest neighbour search*, k-NN 検索) と呼ぶ．ここで # は集合の要素数を得る演算子とする．□

2.2 解候補の絞り込み

距離索引における r-近傍検索では，一般に式 (4)(三角不等式) を利用して次のように解候補を絞り込む．

データ集合から代表点 vp をとる．データ集合 O の各要素から vp への距離は予め計算されているとする．このとき点 q の r-近傍内にあるデータ $q' \in NN_r(q,r)$ から vp への距離 $d(vp,q')$ の取り得る値の範囲は $R-r \leq d(vp,q') \leq R+r$ である．ここで $R = d(vp,q)$ である．従ってこの条件を満さないデータは q の r-近傍に含まれ得ないと判断でき，解の候補から除外され，q からの距離を計算する必要がない．この様子を**図 1** に示す．

O の各要素から vp への距離が索引の構築時に計算され保存されていれば，この絞り込みで必要な距離計算は q, vp 間のみである．

2.3 空間の分割法

距離索引は一般的に階層的索引木であり，空間 (データ集合) を再帰的に分割する事により検索の際の探

索空間の縮小を図る．空間の分割法は大きく二つに分けられ，それぞれによる最も基本的な索引木が *vp-tree(vantage point tree)* と *gh-tree(generalized-hyperplane tree)* である[8]．

vp-tree は超球により空間を分割する．データ集合 O から代表点 (vantage poin) 一点 vp を取り，それから他のデータへの距離の中央値 d_m を求める．空間 (データ集合) は d_m を半径とする超球の内側と外側に二分される．各部分空間は再帰的に同様の方法で分割され，階層木が構成される．データは等分割されるため vp-tree はバランスした二分木となる．

MVP-tree は一つの代表点でいくつの部分空間へ分割するかをパラメータ化し，また複数の (部分) 空間の分割に同一の代表点を利用する事を許容したものであり，vp-tree を一般化したものと言える．MVP-tree ではさらに構築時に計算した距離を保存し，検索時に解の候補の絞り込みに利用する事で検索効率の向上を図っている．

gh-tree ではデータ集合 O から代表点二点 p_1, p_2 を取る．他のデータは p_1, p_2 のうち近い方へ分配されるが，これは二点から等距離にある超平面による分割に他ならない．各部分空間は再帰的に同様の方法で分割され，二分木が構成される．gh-tree は必ずしもバランスしない．

GNAT は代表点の数をパラメータ化したものであり，gh-tree を一般化したものと言える．GNAT ではさらに各代表点から各部分空間への最大・最小距離を保存し，検索時に解の候補の絞り込みに利用している．

3. MI-tree

MI-tree はディスクへの格納および距離計算回数の削減を目的としたページ化された距離索引木であり，動的更新も可能である．MVP-tree などと同様，構築時 (データ挿入時) に求めた距離を保存し，検索時に解の候補の絞り込みに利用する事で距離計算回数の削減を図っている．また各データには相対距離に基づいた数値を割り当て，それをキーとする事で B+tree の利用を可能としている．そのため近傍検索においてはキー値によるデータノード (B+tree の葉ノード) のトラバースが可能であり，バックトラックなどのためのインデクスノードへの余分なアクセスが抑制される．

3.1 MI-tree の構造

2.3 で述べたように，距離索引は一般的に階層的索引木であり，空間を再帰的に分割することで検索時の探索空間の縮小を図る．

MI-tree も同様であるが，それに加え各部分空間内を

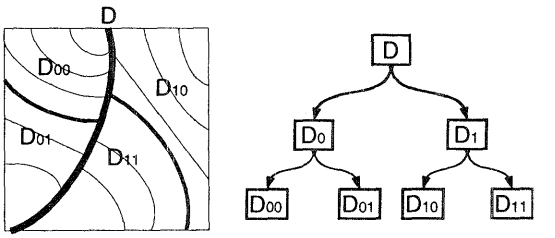


図 2 空間の再帰的分割とスライス
Fig. 2 Recursive division of space and slicing

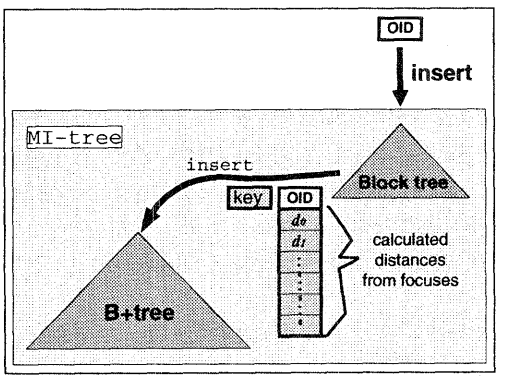


図 3 MI-tree における挿入処理のイメージ
Fig. 3 Illustration of insertion process in MI-tree

スライスする事で探索空間のさらなる縮小を図っている．また検索範囲とスライス範囲を対応させる事で近傍検索での便宜を図る．**図 2**(a) に二次元平面 D を部分空間 $D_{00}, D_{01}, D_{10}, D_{11}$ に分割し，さらにその内部をスライスした様子を示す．図 2(b) はその階層構造を表す部分空間の木 (block tree) である． 空間の分割はデータの挿入に伴ない動的に行なうため，部分空間の木は必ずしもバランスしない．

MI-tree はこの部分空間の木と実際にデータを格納する索引木本体で構成される．部分空間の木は各データに数値キーを割り当てるために利用され，データは数値キーおよびキーを求めるために計算された焦点からの距離と共に索引木本体に格納される．ここで保存される距離は検索時の解候補の絞り込みに利用される．索引木本体の静的構造は B+tree のそれと同じであり，キー値による効率的検索を支援する．挿入処理のイメージを**図 3** に示す．

3.2 キーの構造

キーはデータ空間の分割状況とデータ間の相対距離から生成される数値であり，**図 4** のような内部構造を持つ． 上位数ビット (ブロックビット) を占める**ブロック値** key_b は部分空間のうちの一つを特定するのに用い，下位ビット (スライスビット) の**スライス値**

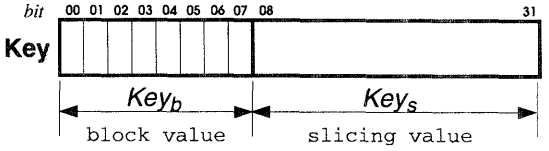


図 4 キーの構造 (例)
Fig. 4 Key structure(example)

key_s はその部分空間内でのスライス位置を表す．このような内部構造を持つ事で一つの部分空間のキーの取り得る値の範囲は他の部分空間のそれと重複しなくなり，全空間のデータを一つの B+tree に格納する事ができ，同じ部分空間に対応するノードであればデータノードのトラバースが可能となる．

図 4 でブロックビットを 8 ビットとしているのは例であるが，これが部分空間の木の高さおよび部分空間数の上限を規定する．部分空間の木は各部分空間の分割情報を格納するため，部分空間の数に比例した格納領域が必要となる．一つのノードに格納する分割情報は数十バイトであるため，ブロックビットが 8 ビットであれば部分空間の木全体の分割情報は数 K ～10 数 K バイトに収まる．

3.3 空間の分割と解候補の絞り込み

一般的に距離索引では空間の分割による探索空間の縮小と共に三角不等式を利用した解候補の絞り込みが行なわれる．これは索引の構築時に計算し保存しておいた距離を用いて行なわれるため，余分な距離計算を必要としない．従来の距離索引における解候補の絞り込み方法は 2.2 で述べた．

MI-tree ではそれに加えてスライス値を利用した新たな絞り込み方法を採用している．空間の分割もスライス値に従って行なわれる．

3.3.1 スライス値

図 4 のようなキー構造を可能とするにはスライス値の上限・下限を知る事ができなければならない．そのような基準として二つの代表点からの距離の差を利用する．

対象データ集合 O から**焦点** (*focus*) 二点 $fp_0, fp_1 \in O$ をとる．二焦点間の距離を

$$R = d(fp_0, fp_1)$$

とおく．点 $q \in O$ の二焦点からの距離およびその差をそれぞれ

$$r_0(q) = d(fp_0, q)$$
$$r_1(q) = d(fp_1, q)$$
$$d_r(q) = r_0(q) - r_1(q)$$

とする．この時次式が成り立つ:

$$-R \leq d_r(q) \leq R \qquad (5)$$

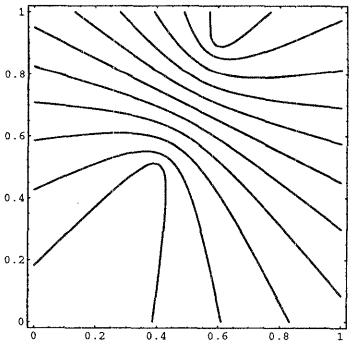


図 5 分割規準となる等値曲線の例
Fig. 5 Example of level surfaces used for space division

$d_r(q)$ を $[0, 2^S - 1]$ に正規化した値

$$Sv(q) = \left\lfloor (2^S - 1)\frac{d_r(q) + R}{2 \cdot R} \right\rfloor$$

を q の焦点 fp_0, fp_1 に対する**スライス値**と呼ぶ．ここで S はスライスビット数である．

このスライス値による分割は d_r の等値曲面 (双曲面) による空間分割であり，部分空間同士は重ならない．二次元平面における L_2 距離の場合の等値曲面 (等値曲線) の様子の例を**図 5** に示す．

3.3.2 r-近傍とスライス範囲の対応

MI-tree では 距離空間 $\langle O, d(\cdot,\cdot)\rangle$ における r-近傍検索と k-最近傍検索を支援する．k-最近傍検索は r-近傍検索を基本としており，r-近傍検索における解候補の絞り込みではスライス値を利用している．

r-近傍内のデータが取り得るスライス値の範囲は次のように決定できる☆.

系 3.1 (r-近傍のスライス範囲)
データ q の二焦点からの距離の差を $d_r(q)$ とする．q の r-近傍に含まれるデータ $q' \in NN_r(q, r)$ について次式が成り立つ:

$$d_r(q) - 2r \;\leq\; d_r(q') \;\leq\; d_r(q) + 2r$$

□

図 6 に二次元平面の場合の r-近傍と対応するスライス範囲の例を示す．q を中心とする円内が問合せ範囲 (q の r-近傍) であり斜線部が対応するスライス範囲である． 図中，問合せ半径を示す r が距離の目盛であるのに対し，スライス範囲を示す $2r$ の目盛は距離の差である事に注意されたい．

3.3.3 空間の分割

MI-tree ではスライス値によりデータ空間を分割する．データ集合 O から焦点二点 fp_0, fp_1 を取り，各

☆ 証明は付録 A.1 を参照

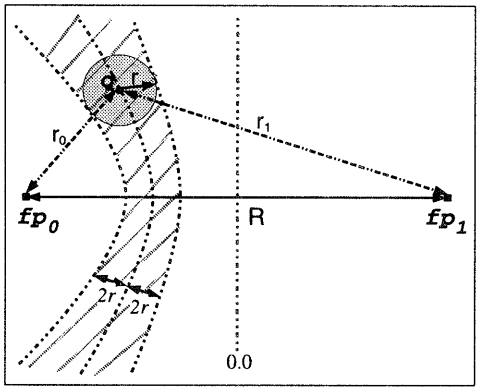


図 6 スライス範囲による解候補の絞り込み
Fig. 6 Candidate filtering using slicing values

データのスライス値を計算しその中央値 d_m を求める．空間 (データ集合) はスライス値が d_m 未満の部分と d_m 以上の部分に二分される．各部分空間は再帰的に同様に分割され，階層木 (部分空間の木) が構成される．分割が中央値によって行なわれるためこの階層木はバランスした二分木となる☆．

3.4 近傍検索の手順

3.4.1 r-近傍検索の手順

中心データ q と半径 r が与えられた時，r-近傍検索は次の手順で行なう:

(1) **交差部分空間とスライス範囲の決定**
部分空間の木を利用して q の r-近傍と交差する部分空間を決定し，各部分空間において r-近傍内の点が取り得るスライス値の最小・最大値の対を得る．交差する部分空間は **図 7** に示す手順で発見できる．

(2) **候補集合の決定 (スライス値による絞り込み)**
得られたキー範囲により検索木本体で範囲検索を行なう．これは B+tree による範囲検索そのものである．

(3) **フィルタリング (従来の方法による絞り込み)**
データを挿入する際に保存しておいた距離を利用し，2.2 で述べた方法で候補集合の絞り込みを行なう．

(4) **リファインメント**
q から各候補への距離を計算し，r-近傍に含まれるデータ集合を得る．

距離計算が必要となるのはスライス範囲の決定のため q と焦点との距離を求める時とリファインメントにおいてである．概念的な手順としては MVP-tree などと同様であるが，候補集合の決定をスライス値を用いた B+tree 上での範囲検索により行なう点が異なる．

☆ 本稿では MI-tree における空間の分割はスライス値の中央値による二分割としているが，必ずしもそうである必要はない．

```
## B - 探索するブロック (部分空間)
## q - 問合せ中心データ
## r - 問合せ半径
## dist(・,・) - 距離関数
blockSearch(B, q, r)
{
    if (B が未分割ブロック) {
        B を解に追加;
    } else {
        ## (q,r) のスライス範囲を求める
        r0 ← dist(q, B の第 0 焦点);
        r1 ← dist(q, B の第 1 焦点);
        d  ← r0 - r1;
        minSlice ← d - 2r;
        maxSlice ← d + 2r;
        ## 交差する子ブロックを探索
        if (minSlice < B の分割スライス値)
            blockSearch(B の第 0 子ブロック);
        if (maxSlice >= B の分割スライス値)
            blockSearch(B の第 1 子ブロック);
    }
}
```

図 7 解空間と交差する部分空間の検索手順
Fig. 7 Procedure for block search

3.4.2 k-最近傍検索の手順

中心データ q と正整数 k が与えられた時，k-最近傍検索は r-近傍検索を利用して次の手順で行なう:

(1) **中心データの検索**
部分空間の木を利用して中心データ q の属すべき部分空間を決定し，q のキー値を得る．属すべき部分空間の決定は 図 7 の手順で検索半径 r を 0.0 とする事で行なう．

(2) **候補集合と半径 r の決定**
q のキー値を中心にデータノードを前後にトラバースし，q の属する部分空間における k-最近傍データの候補集合を求める．そのうち最も q から遠いデータへの距離を次での検索半径 r とする．

(3) **r-近傍検索によるリファインメント**
q の r-近傍検索を行なう．得られたデータ集合のうち q からの距離が小さいデータ k 個を解とする．

ただし複数の部分空間を r-近傍検索する場合，次第に検索が必要な半径が小さくなる事があるため，実際には一括した r-近傍検索は行なわない．すなわち各部分空間を r-近傍検索する時には，それまでに得られている解候補のうち k 番目に近いデータへの距離 r' を用いる．

3.5 データの挿入とノードの分割

部分空間の分割状況と相対距離によりキーを生成する事を除けば，MI-tree におけるデータの挿入は

B+tree での挿入とほぼ同じである.
検索木本体は空のデータノード一つから始まり，分割とキーのプロモートにより成長する．B+tree と異なるのはデータノードの分割が生じた場合である.

3.5.1 部分空間の分割が伴なう場合

一つのデータノードに格納されるデータはすべて同じ部分空間に属する．データノードが分割されると可能ならば対応する部分空間も分割される．**図 8** にはじめてデータノードおよびデータ空間が分割される時の様子を示す.

図 8(a) でデータノードの下に付した数値はブロックビットであり，図 8(b) のどの部分空間に対応するかを示す．ここではブロックビットは 2 ビットとした．図 8(a) のように部分空間の分割を伴なう場合は右側のノードのブロック値をキーとしてプロモートする．ブロックビットの下線は，そのビットが空間分割に使用されている事を示す．またインデクスノード内に示した数値 "10.00" はプロモートされたキーであるが，整数部はブロックビットを，小数部はスライス値を表現している.

図 9 にデータノードおよびデータ空間がさらに分割される様子を示す．部分空間の木はアンバランスになったが索引木本体はバランスしている.

部分空間の分割の際にブロックビットを上位ビットから使用しているため，その後の分割によってもそれまでに付けたキーの順序は変化しない．ただしデータノードを分割する時右側の新ノードに分配されたエントリについてはその時点でブロック値が更新される.

3.5.2 部分空間の分割が伴なわない場合

図 10 に示すのは部分空間の分割を伴なわない場合のデータノードの分割の様子である．部分空間が分割されないのは，対象となるデータノードに対応する部分空間 (この場合 10 空間) が既にブロックビットを使い切っておりそれ以上の分割ができないからである．この場合データノードの分割はスライス値 (キー値) を用いて行なわれ，通常の B+tree と同様のキーのプロモートが行なわれる.

3.6 焦点の選択基準

MI-tree の構築で最も重要なのは，各部分空間でどの二点を焦点とするかである.
式 (5) と系 3.1 に示すように，スライス値の取り得る値の上限・下限は焦点間の距離 R により定まり，その幅に対する r-近傍のスライス範囲の比率も焦点間距離に依存する．そのため同じ検索半径 r でも焦点間距離 R が小さければ検索範囲が二つの部分空間に跨がる可能性が高くなる.

よって焦点はその部分空間においてなるべく離れた二点である事が望ましい.

しかしこのような二点を選択するためには全組合せの距離を計算しなければならない．また部分空間の分割はデータノードの分割のタイミングで行なうため，参照できるのは分割対象ノードに含まれるデータのみである．そこで実際には次のようなヒューリスティクスによりデータノード中でなるべく遠い二点を選択する:

(1) 分割すべきデータノード $\mathcal{N}$ に含まれるデータ集合を $O_{\mathcal{N}}$ とする.
(2) $O_{\mathcal{N}}$ から一点 fp_0 を選択する.
(3) fp_0 から $O_{\mathcal{N}}$ の各データへの距離を求め，そのうち最も距離の遠い点を fp_1 とする.
(4) 新たに fp_1 を fp_0 とし (2) へ戻る.

(2) から (4) を一定回数繰り返して得られた二点を焦点とする.

4. MI-tree の静的構築

3.5 ではデータの逐次挿入による MI-tree の動的構築について述べたが，現在のところ動的構築にはデータ集合の距離分布と挿入順序によっては部分空間の木がアンバランスとなり，結果として検索効率が低下するという問題がある．これは焦点が一つのデータノード内からしか選択されないため，全データの分布を反映できない事に起因している.
しかし既に多量のデータが蓄積されている場合，全データを参照しより適切な焦点を選択する事で，より適切な構造の MI-tree が構築できる．そのようにして構築した MI-tree も構築後は動的な更新が可能であり，実用的なメリットは大きい.

本節では MI-tree の静的構築法について述べる.

4.1 キー，データの生成と挿入

データの有限集合 O とその上の距離関数 $d(\cdot,\cdot)$ が与えられた時，O に対する MI-tree は次のように構築できる:

(1) O と部分空間の木の根ノードを対応させる．また各点から各焦点への距離のを保存するための配列 $D[]$ を空にする.
(2) O の要素数がデータノードの容量以内であるか O に対応する部分空間が分割不可能の場合，現在の部分空間の木に基づいてキーを生成する．各データ $o \in O$ を，生成したキーとこれまでに計算した焦点からの距離 $D[o]$ と共に B+tree へ挿入し，終了する.
(3) そうでなければ O から焦点二点 fp_0, fp_1 を

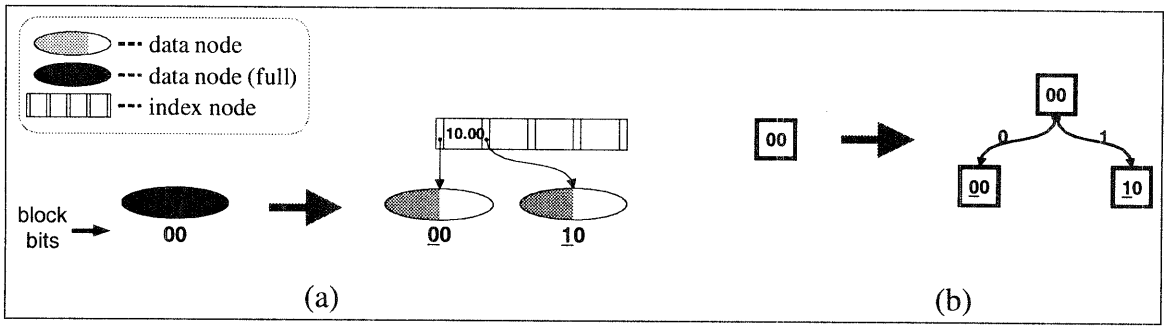


図 8 データノードと部分空間の分割 (1)
Fig. 8 Data node split and space division (1)

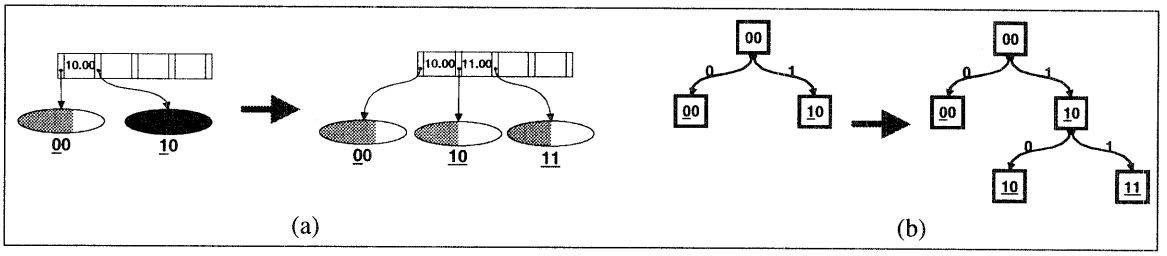


図 9 データノードと部分空間の分割 (2)
Fig. 9 Data node split and space division (2)

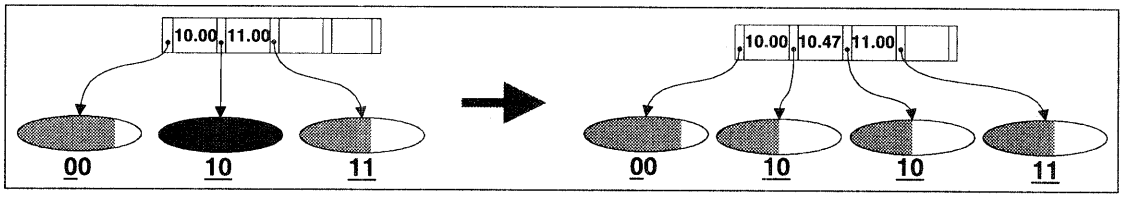


図 10 データノードの分割 (部分空間が分割できない場合)
Fig. 10 Data node split (without space division)

選択する．ここで各点 $o \in O$ から焦点への距離は $D[o]$ に追加保存する．

(4) 各点の二焦点への距離の差の中央値を d_m とし，O を排他的 (かつ補完的) 部分集合

$$O_0 = \{o \mid o \in O \wedge d(o, fp_0) - d(o, fp_1) \leq d_m\}$$
$$O_1 = \{o \mid o \in O \wedge d(o, fp_0) - d(o, fp_1) > d_m\}$$

に分割する．この時対応する部分空間も分割し，それぞれを O_0, O_1 と対応付ける．

(5) O_0, O_1 に対し以上の操作を繰り返す．

焦点の選択は 3.6 で述べた動的構築時の選択方法における O_N を全データ O とするだけである．以上の手順の概略を **図 11** に示す．この手順により構築される MI-tree は部分空間の木も含めてバランスしたものとなる．

5. 実　　験

MI-tree による近傍検索実験を GNU/Linux (Pentium-III, 500MHz) 上で行なった．実装は C++ で行ない，まず B+tree を作成し，その派生クラスとして MI-tree を定義した．

ページサイズは 4KB，ブロックビットは 8 ビット固定とした．データは各次元の幅を $[0, 1)$ とするユークリッド空間中の点とし，文献 1) の Appendix H の方法で生成したクラスタデータを用いた．これは一点を中心としたガウス分布に従うデータ群をクラスタとし，各クラスタの中心はランダムに選択したものである．データ生成のパラメータである分散は $\sigma = 0.1$ とし，クラスタ数は 10 に固定，各クラスタは均等サイズとした．二次元でデータ数 10000 の場合の例を **図 12** に示す．距離は L_2 距離を用いた．

```
## B   - ブロック (部分空間)
## O   - データ集合
## D[] - 各データと焦点との距離の配列の配列
construct(B, O, D[])
{
    if ((O の要素数がデータノードの容量以下)
       or (B が分割できない))
    {
        for o in O {
            key  ← o のキーを生成;
            data ← o と D[o] をまとめる;
            (key, data) を B+tree へ挿入;
        }
    } else {
        focus0, focus1 ← O から焦点を選択;
        ## 各点から焦点への距離を計算し，保存
        for o in O {
            D[o].append(dist(o, focus0))
            D[o].append(dist(o, focus1))
        }
        O0, O1 ← 焦点により O を分割;
        B0, B1 ← 焦点により B を分割;
        construct(B0, O0, D[])
        construct(B1, O1, D[])
    }
}
```

図 11 MI-tree の静的構築の手順
Fig. 11 Procedure for static construction of MI-tree

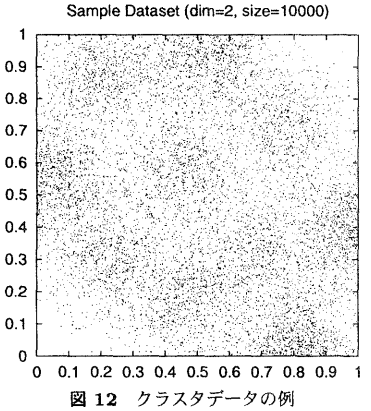


図 12 クラスタデータの例
Fig. 12 Sample of clustered data

5.1 データ数に対する変化

データ数の増加による影響を評価するため，データ空間の次元が 2, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, データ数が 10000, 20000, 40000, 60000, 80000 のそれぞれの場合について k-最近傍検索における距離計算回数とアクセスノード数を計測した．検索データ数 (k) は 10, 20, 50, 100 とした．なお MI-tree は静的に構築したものを用いた．

20 次元の場合の結果を **図 13**, **図 14** に示す．図 14 における “#total node” は全ノード数を示している．

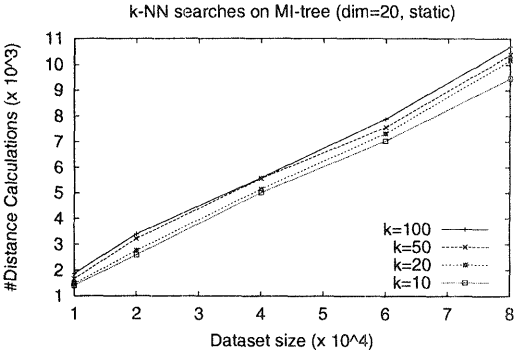


図 13 k-NN 検索での距離計算回数
Fig. 13 Distance calculations in k-NN searches

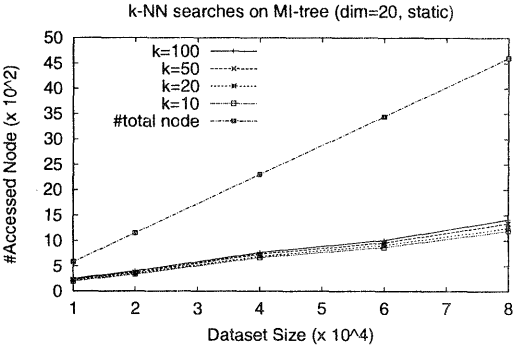


図 14 k-NN 検索でのアクセスノード数
Fig. 14 Accessed nodes in k-NN searches

距離計算回数，アクセスノード数ともにデータ数の増加にほぼ比例して増加する事が分かる．グラフのゆらぎは，生成されたクラスタデータのゆらぎを反映したものと考えられる．

5.2 データ空間の次元に対する変化

5.1 の実験結果の，次元に対する距離計算回数とアクセスノード数の変化を**図 15**, **図 16** に示す．データ数は 20000 である． 両者ともに次元の増加に対し対数的に増加する傾向にある事が分かる．

5.3 ランダム順挿入による構築の場合との比較

静的に構築した MI-tree とクラスタデータをランダム順に逐次挿入して構築した MI-tree における k-最近傍検索での距離計算回数とアクセスノード数の比較を行なった．

距離計算回数とアクセスノード数は同様の傾向にあるので，ここでは**図 17**, **図 18** に 20 次元の場合とデータ数 20000 の場合の，10-NN 検索での距離計算回数を示す．

静的構築の場合ランダム順挿入による構築の場合に比べて 3 ～ 20 % 程度距離計算回数が減少している

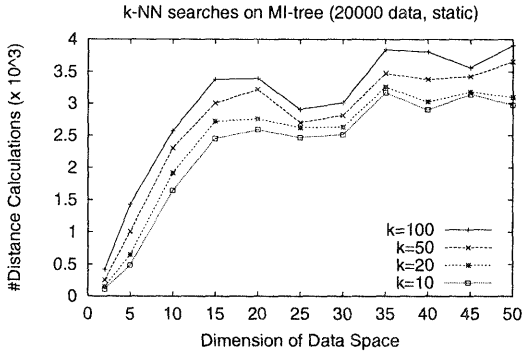


図 15 k-NN 検索での距離計算回数
Fig. 15 Distance calculations in k-NN searches

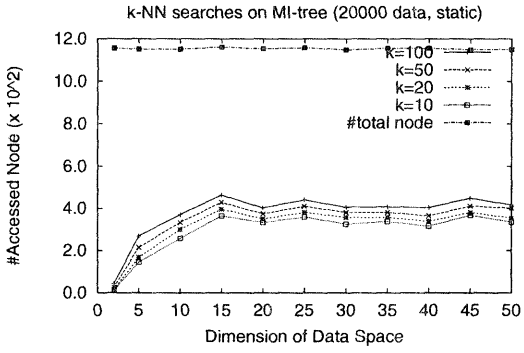


図 16 k-NN 検索でのアクセスノード数
Fig. 16 Accessed nodes in k-NN searches

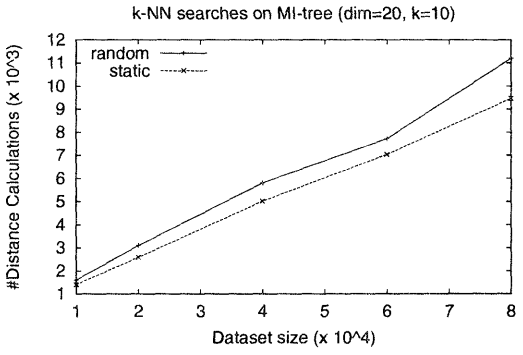


図 17 10-NN 検索での距離計算回数
Fig. 17 Distance calculations in 10-NN searches

事が分かる．差が大きくないのは静的構築のメリットが小さいからではなく，むしろランダム順が動的挿入の順序として良い性質を持つからと考えられる．

5.4 一様分布データの場合

R-tree などの多次元索引においては，データ空間が 15 次元程度以上になると近傍検索の効率が著しく低下し，ほぼ全データへのアクセスが必要となる事が知ら

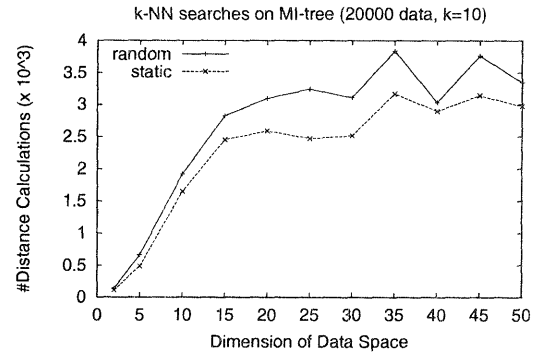


図 18 10-NN 検索での距離計算回数
Fig. 18 Distance calculations in 10-NN searches

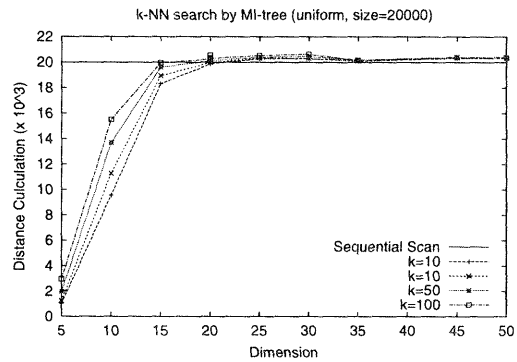


図 19 次元と距離計算回数 (一様分布データの場合)

れている [2]．同様の減少が見られるかを観察するため一様分布データにおける検索実験を行なった．**図 19** に 20000 データの時の次元の変化に伴なう距離計算回数の変化の様子を示す．図 19 から，MI-tree においても 15 次元程度になるとほぼ全データとの距離計算が必要となっている事が分かる．

5.5 MVP-tree, GNAT との比較

検索時の距離計算回数について MVP-tree と GNAT との比較を行なった．文献 3), 7) には k-最近傍検索の手順が示されていないため，ここでは r-近傍検索による比較を行なった．MVP-tree は各ノードにおける代表点 (vantage point) の数および分割数がパラメータ化されているが，ここではそれらを (2,2), (3,2), (3,13), (3,80) とする組合せで実験を行なった．GNAT では代表点 (division point) の数がパラメータ化されているが，ここでは 50 および 100 とした．MI-tree は静的に構築したもの，ランダム順挿入で動的に構築したものの双方で実験した．データセットは 5, 10, ..., 50 次元，10000, 20000, 40000, ...80000 データのそれぞれのものを用いた．

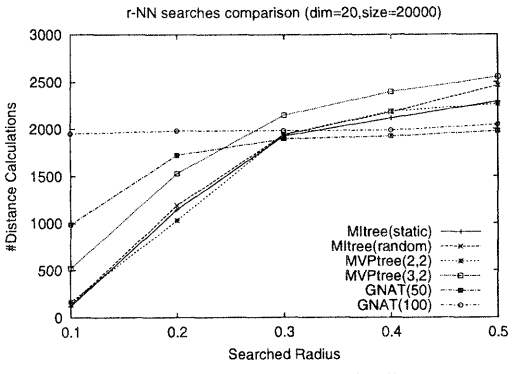


図 20 r-N 検索での距離計算回数
Fig. 20 Distance calculations in r-N searches

図 20 に 20 次元，20000 データの場合の結果を示す．ただし MVP-tree については最も距離計算回数が少なかった (2,2), (3,2) の結果のみを示した☆．

距離計算回数は検索半径に対しておよそ対数的に増加する事が分る．MI-tree についてはアクセスノード数も同様の傾向であった．全体的に MI-tree は MVP-tree(2,2) と同程度の距離計算回数で検索できる事が分かった．図 20 で，検索半径 0.3 以上では GNAT(50) が最も計算回数が少ないが，距離計算回数がほとんど変化していない事が興味深い．GNAT(100) については半径が変化しても距離計算回数はおよそ 2000 回でほとんど変化していない．データセットが 10 個の均等サイズクラスタから構成されている事から，これは一つのクラスタのサイズに対応したものと考えられ，この数値はデータセットの性質を強く反映したものと考えている．

また多少の変動はあるものの，動的に構築した MI-tree と静的に構築した MI-tree の差が k-最近傍検索の場合と比べると少ない事や，MVP-tree と同程度の距離計算回数であるという傾向は全次元全データ数で見られた．

☆ 他のパラメータでは二倍以上の距離計算回数が必要であった．代表点や分割数を大きくした時に却って距離計算回数が多くなったのは，分割数 (すなわち木における子ノードの数) が増加した分木の高さが低くなり，解候補の絞り込みに利用できる代表点との距離の数が少なくなったためと考えられる．また分割数の増加による枝刈りのメリットが得られにくいデータセットであったためとも考えられる．
なお GNAT で GNAT(100) が GNAT(50) よりも距離計算回数が多くなっているのは，代表点が増加してもそれに見合った絞り込み力の増加が得られていないためと考えられる．すなわち，代表点からの距離による絞り込みは，ある程度の数までは増加した分だけ絞り込みの効果も増すが，それ以上ではあまり効果は増さず，かえって代表点との距離コストがマイナス要因となると考えられる．

6. まとめと今後の課題

ディスクへの格納および動的な更新が可能な距離索引木 MI-tree を提案し，その構造と挿入，近傍検索手順および静的構築法を示した．
MI-tree を実装し，静的構築とランダム順挿入による動的構築の場合の k-最近傍検索における距離計算回数とアクセスノード数を実験により計測した．
静的に構築した場合は動的に構築した場合に比べ距離計算回数，アクセスノード数共 3 ～ 20 % 程度削減される事が分った．アクセスノード数は線型走査に比べ数分の一である．なおランダム順動的構築に対して静的構築による改善率が少ないのは，ランダム順が挿入順序としてはむしろ良い性質を有するためと考えられる．

MVP-tree, GNAT と r-近傍検索における距離計算回数の比較を行ない，MVP-tree と同程度の距離計算回数で検索できる事を確認した．これは MI-tree は MVP-tree と同様に参照点 (vantage point もしくは focus) からの距離による解のフィルタリングを行なっており，その効果が同程度であるためと考えられる．MI-tree にはディスクへの格納が容易であるなど B+tree を利用する事によるメリットがあり，また動的な更新も可能であるという点で有利である．

現在のところ MI-tree には，空間的に偏った順序でのデータ挿入による動的構築の場合には部分空間の木がアンバランスとなり，検索効率が低下するという問題がある．しかし既に多量のデータが蓄積されている場合には静的に構築する事でこのような問題を回避でき，構築後の動的更新も可能である．

今後は完全に動的な環境でも性能が低下しないよう，動的環境でのバランス回復法について検討する．
なお，部分空間の分割タイミング，同一焦点での分割回数などは本稿で述べたように固定である必要はない．今後可変化可能性を検討する．スライスについての上限・下限が分からなければならないという制約は，固定ビット数 (スライスビット数) で整数表現している事から来る制約であるため，浮動小数点表現を用いる事により除く事ができる．よって，例えば FastMap[4] など他の手法を利用したスライス値の採用も考えられる．

参 考 文 献

1) A.K.Jain and Dubes, R.: *Algorithms for Clustering Data*, Prentice-Hall (1998).
2) Berchtold S., Bohm C., K. D. K. H.-P.: A Cost Model For Nearest Neighbor Search in High-

Dimensional Data Space, *ACM PODS Symposium on Principles of Database Systems, 1997, Tucson, Arizona* (1997).
3) Bozkaya, T. and Ozsoyoglu, M.: Distance-based indexing for high-dimensional metric spaces, *Proceedings of the ACM SIGMOD Conference on Management of Data, Tucson, Arizona, May 1997*, pp. 357–368 (1997).
4) Faloutsos, C. and Lin, K.-I. D.: FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets, *Proceedings of the ACM SIGMOD Conference on Management of Data, San Jose, CA, May 1995*, pp. 163–174 (1995).
5) PaoloCiaccia, M.P. and Zezula, P.: M-tree: An efficient access method for similarity search in metric spaces, *Proceedings of the 23rd VLDB Conference, Athenes, Greece, 1997*, pp. 357–368 (1997).
6) Ravi Kanth, D. A. and Singh, A. K.: Dimensionality reduction for similarity searching in dynamic databases, *Proceeding of ACM SIGMOD Conference on Management of Data, Seattle, Washington, 1998*, pp. 166–176 (1998).
7) S.Brin: Near neighbour search in large metric spaces, *Proceedings of the 21st VLDB Conference, Zurich, Switzerland, 1995*, pp. 574–584 (1995).
8) Uhlmann, J. K.: Satisfying general proximity/similary queries with metric trees, *Information Processing Letters*, Vol. 40, pp. 175–179 (1991).
9) Yasushi Sakurai, M. Y. and Uemura, S.: High-dimensional nearest neighbour search based on virtual bounding rectangles, *Proceeding of International Conference on FODO'98*, pp. 258–267 (1998).
10) 岩崎雅二郎: 類似画像検索を実現する距離空間インデックスの実装および評価, 情報処理学会論文誌 データベース, Vol. 40, pp. 24–44 (1999).

付　　録

A.1 系 3.1 の証明

点 q の r-近傍に含まれる点 q' から fp_0, fp_1 への距離をそれぞれ $r_0(q'), r_1(q')$ とする．この時三角不等式から

$$(0 \leq)\ r_0(q') \leq r_0(q) + r \tag{6}$$
$$(0 \leq)\ r_1(q') \leq r_1(q) + r \tag{7}$$
$$(0 \leq)\ r_0(q) \leq r_0(q') + r \tag{8}$$
$$(0 \leq)\ r_1(q) \leq r_1(q') + r \tag{9}$$

といえる．式 (6),(9) および (7),(8) からそれぞれ

$$r_0(q') - (r_1(q') + r) \leq (r_0(q) + r) - r_1(q)$$
$$r_0(q) - (r_1(q) + r) \leq (r_0(q') + r) - r_1(q')$$

であり，したがって

$$r_0(q') - r_1(q') \leq (r_0(q) - r_1(q)) + 2r$$
$$(r_0(q) - r_1(q)) - 2r \leq r_0(q') - r_1(q')$$

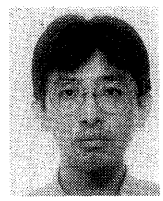
である．よって

$$d_r(q) - 2r \leq d_r(q') \leq d_r(q) + 2r \tag{10}$$

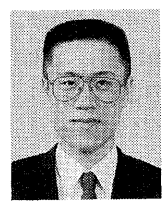
が成り立つ．

(平成 1999 年 3 月 20 日受付)
(平成 1999 年 6 月 27 日採録)

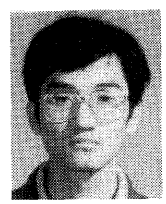
(担当編集委員 **吉川 正俊**)



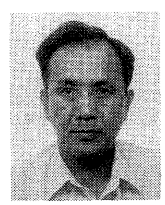
石川　雅弘（学生会員）
1995 年 筑波大学第三学群情報学類卒業．1997 年 同大大学院工学研究科博士前期課程修了．現在 同大学院工学研究科博士後期課程在学中．



能登谷淳一（正会員）
1999 年 筑波大学大学院工学研究科修了．同年 秋田県立大学電子情報システム学科 助手．博士 (工学)．



陳　　漢雄（正会員）
1993 年 筑波大学大学院工学研究科修了．同年 同大電子・情報工学系助手．1994 年 つくば国際大学産業情報学科講師．データベースシステムに関する研究に従事．工博．



大保　信夫（正会員）
1968 年 東京大学大学院修士課程修了．同年 同大理学部助手．1980 年 筑波大学電子・情報工学系講師．1995 年 同大電子・情報工学系教授．データベースシステムに関する研究に従事．理博．