データベース・システム 47－6
（１９８５・５・２０）

リレーションのデータベースマシン向きのクラスタリング方法に関する一考察

板倉　一郎　　井上　潮
（ＮＴＴ　横須賀電気通信研究所）

1 はじめに

近年、高水準言語による問合せが可能である、データベースの設計が容易である、等の理由により、リレーショナル・データベースが注目され、汎用マシン上にリレーショナル・データベース管理システムが実現されるようになった。汎用マシン上のシステムでは、転置ファイルに代表されるインデックス技法により検索処理の高速化を行っている。しかし、データの更新によるインデックスの保守時間の増加、インデックスのための記憶域の増加、などの問題を抱えている。そこで、インデックスを使用しなくても、リレーションを高速に検索することが可能なデータベース・マシンが研究・開発されている。この中には、選択・制約演算を専用ハードウェア（サーチプロセッサ）によりリレーション全体をフルサーチすることによって行うものがある。[1] しかし、大規模なデータベースに対して常にフルサーチを行っていたのでは、検索処理時間がデータベース量に比例して長くなるという問題が生じる。

本稿では、サーチ範囲を局所化することにより、専用ハードウェアを用いたフルサーチに基づく検索処理を高速化するためのクラスタリングについての考察結果を述べる。２章では、従来のクラスタリング方法をサーベイし、他の方法と比較し拡張k-d treeが優れていること、実際に適用する際の問題点を述べる。そして、３章では、拡張k-d treeを改良し問題点を解決した改良拡張k-d treeを提案する。４、５章では、個々の改良の効果について机上評価及びシミュレーションによる評価結果を示す。最後に、６章では、改良拡張k-d treeによるクラスタリングを総合的に評価する。

2 従来のクラスタリング方法

2.1 クラスタリングとは

クラスタリングとは、リレーション中のタプルを特定のアトリビュートの値によって、クラスタという論理的な単位にグループ分けすることである。クラスタリングを行うことにより、リレーションを検索するとき、検索条件を満足するタプルが存在する可能性があるクラスタ、すなわち、リレーションの一部分だけをアクセスすればよくなる。このため、リレーション全体をアクセスするフルサーチと比較して、検索処理時間を短縮することが可能になる。ただし、クラスタリングに用いたアトリビュートに対する検索条件が与えられない場合には、フルサーチしなければならない。

2.2 従来方式の分類と比較

クラスタリングを行うための方法は、①クラスタリングの対象となるアトリビュート（以下、キーと呼ぶ）の個数、②リレーションを分割するための情報の管理方法、③データを格納するクラスタのオーバフローの対処方法、によって分類することができる。これらの観点から、従来研究されているクラスタリングを行うための方法とその具体例を分類すると表1 の様になる。

表1 で示した従来方法のうち、マルチ・キーを対象とする方法の中で、ハッシュを用いる方法は値の範囲を指定する検索条件に適用できない。テーブルを用いる方法はオーバフロー域のサーチを行うため検索効率が劣るか、または、テーブルの修正が全体に及ぶため更新効率が劣る。ツリーを用いる方法は値の範囲を指定する検索条件にも適用でき、更新に対しツリーを局所的に修正して対応するため、検索・更新効率が良い。

ツリーを用いる方法（k-d tree）は、各ノードに分割を行うアトリビュート（ディスクリミネータ）とその値の情報を持つ。このノードに持つ情報と構成法の違いから、k-d treeには、（基本）k-d tree、拡張k-d tree、一般化k-d tree、colored binary trieの４つの方法がある。これらの４つの方法のうち、（基本）k-d treeは、検索効率に関して平均アクセスクラスタ数を減らすための対策が施されていない。また、一般化k-d treeは、クラスタリングを行うために必要な情報（検索条件で指定されるアトリビュートの値）の取得が不可能に近く、実現は困難である。colored binary trie は、検索パターンに応じて平均アクセスクラスタ数を減らすことができない。拡張k-d treeは、クラスタリングを行うための情報はアトリビュートが検索条件で指定される頻度のみであり、この情報による平均アクセスクラスタ数の最小化が可能であるため、実現性と検索効率の面で優れている。

拡張k-d treeは、アトリビュートの値の大小関係によってリレーションを２分割し、分割した結果がクラスタに格納できるまで２分割を繰り返す。分割を行うアトリビュート（ディスクリミネータ）の列は、検索条件で指定されるアトリビュートの出現頻度を用い、問合せに対する平均アクセスクラスタ数を最小にするように決められる。拡張k-d treeを使ったクラスタリングの例を図1 に示す。

次節以降、拡張k-d treeに基づくクラスタリングの実現方法の考察結果を述べる。

2.3 拡張k-d treeの問題点

拡張k-d treeによるクラスタリングを実際のシステムに適用する場合、以下の２つの問題点が予想される。

①クラスタの格納率が低い：アトリビュートの値が同一のタプルが分割を行うクラスタ内に複数存在すると、タプル数を均等に分割することができなくなり、格納タプル数の少ないクラスタが生成され、全格納領域に対してタプルが格納されている領域の割合（格納率）が低くなる。

②ディスクアクセス時のオーバヘッドが大きい：ツリー上で論理的に近接するクラスタでも物理的に近接するとは限らず、また、ツリーをサーチした結果得られるクラスタはツリーのリーフに散在し、物理的に近接しない。このため、１クラスタのアクセス毎にディスクのシーク、サーチ動作を伴うことになり、処理時間が増大する。

3 改良拡張k-d treeによるクラスタリング

2.3 で示した拡張k-d treeによるクラスタリングの問題点を解決するため、拡張k-d treeを以下のように改良した改良拡張k-d treeによるクラスタリング法を提案する。

①格納率を上げるため、クラスタの分割をディスクリミネータ変更分割法で行う。

②ディスクアクセス時のオーバヘッドを減らすため、クラスタの物理アドレスでソートしてからディスクへアクセスする。

この２点の改良方法について、格納率に関しては４章で、ディスクのアクセス法に関しては５章で述べる。特に、格納率については、ディスク量の増加によるシステムコストの上昇、フルサーチ時の処理時間の増大等を招くため、重要な問題である。

4 格納率に関する考察

4.1 状態遷移モデルによる解析

この節では、今までに行われた状態遷移モデルに基づく格納率の解析法を紹介し、この方法の拡張k-d treeと改良拡張k-d treeへの適用法について述べ、解析結果を示す。

(1) 状態遷移モデルに基づく解析法

文献[9]では、レコードを追加していき、ブロックがオーバフローした場合、２個のブロックにレコードを均等に分配する分割方法について、ＰＴＤ（Probabilistic Transition Diagram）を導入して格納率の解析を行っている。ＰＴＤは以下に示すものから構成される。

- ラベルの付いたノード｛Ｌｉ｝の集合。
  各Ｌｉはｉ個のレコードを格納したブロックを表わす。
- 向きのあるエッジ｛Ｅｊ｝の集合。Ｌｕから Ｌｖへのエッジ Ｅｊは挿入があるときのｕ個のレコードブロックからｖ個のレコードブロックへのブロック数の遷移を表わす。
- 各Ｌｉに対応してｉ個のレコードを格納したブロック数Ａｉ。
- ＬｕからＬｖへのＥｊに対応して１個のレコードが挿入されたとき、Ａｕを１減らしＡｖを１増やす確率Ｑｊ。

なお、追加するレコードがあるブロックにヒットする確率は、そのブロックに格納されているレコード数に比例すると仮定している。

Ａｉに関する状態遷移方程式はＰＴＤから次の様にして導く。

- Ｌｉに入ってくるエッジＥｊはＡｉをＱｊだけ増やす。
- Ｌｉから出ていくエッジＥｊはＡｉをＱｊだけ減らす。

また、ｎ個のレコードが格納されている状態での格納率（ｓｕｆ）は、次式で定義される。

$$suf = n / \left(b \sum_{i=\lceil b/2 \rceil}^{n} Ai,n \right)$$

(2) 拡張k-d treeへの適用法

(1)でのブロックの分割方法は、オーバフローしたブロックのレコードを２つのブロックに均等に分配するものである。そこでは、オーバフローしたブロックの中の分割を行うキーが全て異なり、完全にレコード数を２等分できると仮定している。しかし、実際のデータベースでは、同一の値を持ったキーが存在することが多いので、キーの重複を考慮したブロックの分割方法について格納率の解析を行う必要がある。拡張k-d treeにおける分割法も２つのクラスタのタプル数を等しくするものである。従って、拡張k-d treeによるクラスタリングの格納率を解析するため、ここでは、レコードが２等分されない様なブロックの分割についての状態遷移を表現できる様にＰＴＤを修正する。

ブロックサイズをｂ（簡単のため、ｂ＝２ｑ、ｑは正の整数）とする。キーの重複を考慮すると、ｂ＋１個のレコードの分割パターンとしては、０とｂ＋１、１とｂ、２とｂ－１、…、ｑとｑ＋１のｑ＋１通りが考えられる。これらの分割パターンは、Ｐ０、Ｐ１、…、Ｐｑ（ブロックサイズｂとキーの平均重複度（同一キーがｘ個存在する確率関数をＫ（ｘ）とすると、ΣｘＫ（ｘ））の関数で表わされる）の確率で発生するとする。また、ブロックが分割できない（０とｂ＋１に分割）場合は、さらに分割を行うものとする。この様にすると、格納タプル数がｉ個のブロックが生成される確率Ｐ'ｉ（ｉ=1,…,n）は次式の様になる。

Ｐ'ｉ＝Ｐｉ／（１－Ｐ０）

以上の様な分割法のＰＴＤを図2に示す。

Ｌ＝｛（Ｌ０，Ａ０），（Ｌ１，Ａ１），…，
（Ｌｂ，Ａｂ）｝

Ｅ＝｛（Ｅ１：Ｌｂ→Ｌｂ，
Ｑ１＝Ｐ'１×ｂＡｂ／ｎ），
（Ｅ２：Ｌｂ→Ｌ０，
Ｑ２＝Ｐ'０×ｂＡｂ／ｎ），
⋮
（Ｅｂ+2：Ｌｂ-1→Ｌｂ，
Ｑｂ+2＝（ｂ－１）Ａｂ-1／ｎ），
⋮
（Ｅ2ｂ+2：Ｌｂ→Ｌｂ-1，
Ｑ2ｂ+2＝Ｐ'２×ｂＡｂ／ｎ）｝

Ａｉに関する状態遷移方程式は、以下の様にな

る。

$$A_{b,n+1} = A_{b,n} + ((b-1)A_{b-1,n} + P'_1 \times bA_{b,n} - bA_{b,n}) / n$$

$$A_{b-1,n+1} = A_{b-1,n} + ((b-2)A_{b-2,n} + P'_2 \times bA_{b,n} - (b-1)A_{b-1,n}) / n$$

$$\vdots$$

$$A_{q,n+1} = A_{q,n} + ((q-1)A_{q-1,n} + P'_q \times bA_{b,n} - qA_{q,n}) / n$$

$$\vdots$$

$$A_{1,n+1} = A_{1,n} + (A_{0,n} + P'_1 \times bA_{b,n} - A_{1,n}) / n$$

$$A_{0,n+1} = A_{0,n} + (P'_0 \times bA_{b,n} - A_{0,n}) / n$$

(3) 改良拡張k-d treeへの適用法

格納率を改善するため、タブル数の少ないクラスタの生成を避けるようにクラスタの分割を行うディスクリミネータ変更分割法を提案する。この方法は、オーバフローしたクラスタを予め決められたディスクリミネータで２つのクラスタに分割したときに、２つのクラスタに格納されるタブル数にある基準以上の差が生じた場合、ディスクリミネータを変更して分割を再び行う。再分割を行う基準比（２つのクラスタに格納されるタブル数の比）をｒ（０＜ｒ＜１）、基準格納タブル数をrb（ｒ×ｂ）とすると、格納タブル数が$\lceil rb \rceil - 1$個以下のクラスタがクラスタの分割により生成された場合は再度分割を行うので、図2 で示したＰＴＤと比較すると、Ｌ0 からＬ$\lceil rb \rceil$-1のノードと、Ｌb からＬb 、Ｌb-1 、…、Ｌb-$\lceil rb \rceil$+2へのエッジが存在しない点が異なる。また、ブロックの分割パターンの発生確率をＰ0 、Ｐ1 、…、Ｐq（4.1 の(2)で定義したものと同じ）とすると、基準を満足しない場合の再分割を繰り返して行うことを考えると、格納タブル数が$\lceil rb \rceil$以上b-$\lceil rb \rceil$+1以下のクラスタが生成される確率Ｐ'i（i=$\lceil rb \rceil$、…、b-$\lceil rb \rceil$+1）は次式の様になる。

$$P'_i = P_i / (1 - (P_0 + P_1 + \cdots + P_{\lceil rb \rceil - 1}))$$

この様なディスクリミネータ変更分割法のＰＴＤを図3 に示す。

$$L = \{(L_{\lceil rb \rceil}, A_{\lceil rb \rceil}), (L_{\lceil rb \rceil+1}, A_{\lceil rb \rceil+1}), \cdots, (L_b, A_b)\}$$

$$E = \{(E1 : L_b \rightarrow L_{\lceil rb \rceil}, Q1 = P'_{\lceil rb \rceil} \times bA_b / n), (E2 : L_{\lceil rb \rceil} \rightarrow L_{\lceil rb \rceil+1}, Q2 = P'_{\lceil rb \rceil+1} \times bA_b / n), \vdots, (Eb\text{-}\lceil rb \rceil+1 : Lb\text{-}1 \rightarrow Lb, Qb\text{-}\lceil rb \rceil+1 = (b-1)A_{b-1} / n), \vdots, (E2b\text{-}3\lceil rb \rceil+2 : L_b \rightarrow Lb\text{-}\lceil rb \rceil+1, Q2b\text{-}3\lceil rb \rceil+2 = P'_{b-\lceil rb \rceil+1} \times bA_b / n)\}$$

Ａi に関する状態遷移方程式は以下の様になる。

$$A_{b,n+1} = A_{b,n} + ((b-1)A_{b-1,n} - bA_{b,n}) / n$$

$$A_{b-1,n+1} = A_{b-1,n} + ((b-2)A_{b-2,n} - (b-1)A_{b-1,n}) / n$$

$$\vdots$$

$$A_{i,n+1} = A_{i,n} + ((i-1)A_{i-1,n} + P'_i \times bA_{b,n} - iA_{i,n}) / n$$

$$A_{\lceil rb \rceil+1,n+1} = A_{\lceil rb \rceil+1,n} + (\lceil rb \rceil A_{\lceil rb \rceil,n} + P'_{\lceil rb \rceil+1} \times bA_{b,n} - (\lceil rb \rceil+1)A_{\lceil rb \rceil+1,n}) / n$$

$$A_{\lceil rb \rceil,n+1} = A_{\lceil rb \rceil,n} + (P'_{\lceil rb \rceil} \times bA_{\lceil rb \rceil,n} - \lceil rb \rceil A_{\lceil rb \rceil,n}) / n$$

また、n1個タブルが格納された時点までのディスクリミネータの変更回数Ｔn1は、

$$T_{n1} = \sum_{n=b}^{n1} (bA_{b,n} / n + (P_0 + P_1 + \cdots + P_{\lceil rb \rceil - 1}) / (1 - (P_0 + P_1 + \cdots + P_{\lceil rb \rceil - 1})))$$

で与えられる。

(4) 解析結果

(2) で示した状態遷移方程式をブロックサイズとキーの平均重複度をパラメータとして、また、(3) で示した状態遷移方程式をブロックサイズ、キーの平均重複度、再分割基準比をパラメータとしてプログラムを作成して解き、格納率とディスクリミネータの変更回数を解析した。その結果を図4 、5 、6 に示す。

評価結果より、次のことが言える。

- キーの平均重複度が1（全てのキーが異なる）のとき、格納率は70%程度である。
- 従来の方法では、キーの平均重複度の増加とともに、同一キー値を持つレコードの存在する確率が高くなるため、レコード数を均等にクラスタを分割できなくなり、格納率は低下する。
- ディスクリミネータ変更分割法は従来の方法と比較して、格納率は 5～15%程度良い。また、再分割を行う回数は、キーの平均重複度と再分割基準比が大きいとき（最悪の条件）には1回のクラスタの分割につき2回程度、また、小さいとき（最良の条件）には0回になる。
- 再分割基準比を大きくすると、格納率は約70%になるが、再分割回数が多くなりオーバヘッドが大きくなる。格納率と再分割回数の関係から、再分割基準比は1/3 、1/4 程度が適当であると考えられる。

4.2 シミュレーションによる評価

(1) 格納率の評価

従来の拡張k-d treeと4.1 で提案したディスクリミネータ変更分割法を取り入れた改良拡張k-d treeについてシミュレーションを行い、格納率の評価を行った。評価結果を図7 に示す。

この結果、4.1 で示した状態遷移方程式による解析結果と同様に、ディスクリミネータ変更分割法が従来の拡張k-d treeの分割法より、格納率に関して優れていることが確認できる。

拡張k-d treeにおけるディスクリミネータは、検索効率が最良（等号による検索条件に対する平均アクセスクラスタ数が最小）になる様に決められている。しかし、ディスクリミネータ変更分割法は、ディスクリミネータを変えるため、検索効率の悪化が予想されるので、検索効率についても評価を行った。

(2) 検索効率の評価

検索効率の評価は、シミュレーションにより、4個のキーに対する15種類の等号による検索条件に対して拡張k-d treeをサーチして、アクセスする平均クラスタ数を求めることによって行った。拡張k-d treeと改良拡張k-d treeそれぞれによるクラスタリングについての評価結果を図8 に示す。

評価結果より、実際のデータベースで使用されることが多いと予想されるタプル長（ 200～ 400 バイト）では、検索効率はほとんど悪化しないことがわかる。

以上の評価結果から、ディスクリミネータ変更分割法は拡張k-d treeの分割法と比較して優れていると言える。

5 ディスクアクセスに関する考察

ディスクアクセス時のオーバヘッドを減らすために、クラスタへのアクセスをディスクの物理的動作特性を考慮して行うことにする。アクセスするクラスタに対するディスクのシーク・サーチ動作を少なくするため、ツリーをサーチして得られたクラスタの順番に直接ディスクへアクセスするのではなく、ツリーのサーチ終了後にクラスタの物理アドレスでソートしてからアクセスする方法を考える。この方法はソート処理に時間がかかるため、ディスクの動作時間短縮の効果とのトレードオフについて机上計算による評価を行った。評

価結果を図9 に示す。

評価結果より、

・ソートした後にディスクアクセスを行う方法はアクセスするクラスタ数が多くなるほど、有効である。

・ソート時間は、ディスク動作時間と比較して小さい。

ことが明かになった。従って、クラスタへのアクセスは、物理アドレスでソートしてから行う方式が優れていると言える。

6 総合評価

本稿で提案した改良拡張k-d treeの総合的な効果を確認するため、改良拡張k-d treeによるクラスタリング、従来の拡張k-d treeによるクラスタリング、クラスタリングを行わないフルサーチの３つの方法について、検索条件で指定されるキーの数をパラメータに、等号による検索条件を満足するクラスタへのディスクアクセス時間について机上計算による評価を行った。評価結果を図10に示す。

評価結果より、以下のことが定量的に明かになった。

・クラスタの分割方法とクラスタへのアクセス方法の改良により、改良拡張k-d treeは従来の拡張k-d treeと比較して、30～40%程度ディスクアクセス時間を短縮する。

・クラスタリングに用いたキーが検索条件で指定された場合、改良拡張k-d treeによるクラスタリングはフルサーチと比較して、ディスクアクセス時間が1/3 ～1/3000程度に短縮される。

・クラスタリングに用いたキーが検索条件で指定されない場合には、クラスタリングを行ったことによる格納率の低下のため、フルサーチと比較して、ディスクアクセス時間が50%程度増加する。

7 まとめ

本稿では、データベース・マシンにおける検索処理を高速化するためのクラスタリング法について述べた。結論をまとめると以下のようになる。

・従来のクラスタリングを行う方法の中では、拡張k-d tree法が優れているが、格納率が低い、ディスク・アクセス時のオーバヘッドが大きい、という問題点がある。

・この問題点を解決するために、クラスタの分割法とディスクへのアクセス法を改良した改良拡張k-d tree法を提案した。

・この改良により、従来の拡張k-d treeによるクラスタリングと比較して、格納率を10%、検索効率を30%程度改善できることを明かにした。

参考文献

[1] E.Babb、" Implementing a Relational Database by Means of Specialized Hardware "、ACMTODS、Vol.4、No.1、1979、1-29

[2] R.L.Rivest、" Partial-match retrieval algorithms "、SIAM J. Comput.、Vol.5、No.1、1976、19-55

[3] J.H.Liou and S.B.Yao、" Multi-dimensional Clustering for Data Base Organization "、Inf.Syst. Vol.2、1977、187-198

[4] J.Neivergelt et al、" The Grid File : An Adaptable,Symmetric Multikey File Structure"、ACMTODS、Vol.9、No.1、1984、38-71

[5] J.L.Bentley、" Multidimensional Binary Search Trees Used for Associative Searching"、CACM、Vol.18、No.9、1975、507-517

[6] J.M.Chang and K.S.Fu、" A Dynamic Clustering Technique for Physical Database Design"、Proc. of ACM SIGMOD Conf.、1980、188-199

[7] 伏見　他、"多次元クラスタリング技法に基づくＧＲＡＣＥ二次記憶系の設計と評価"、信学技報、EC83-49、1984、13-24

[8] Y.Tanaka、" Adaptive Segmentation Schemes for Large Relational Database Machines"、Proc. Int'l Workshop on Database Machine、1983

[9] T.Nakamura and T.Mizoguchi、" An Analysis of Storage Utilization Factor in Block Split Data Structuring Scheme"、Proc. 4th VLDB、1978、489-495

表1　クラスタリングを行うための方法の分類

キーの個数	情報管理	オーバフロー	具　体　例
シングル	ハッシュ	非適応型	通常のハッシング技法
		適応型	dynamic hashing extendible hashing
	テーブル	非適応型	ISAM
	ツリー	適応型	B-tree
マルチ	ハッシュ	非適応型	multi-key hashing[2]
	テーブル	非適応型	multi-dimensional directory[3]
		適応型	grid file[4]
	ツリー	適応型	k-d tree[5] 拡張k-d tree[6] 一般化k-d tree[7] colored binary trie[8]

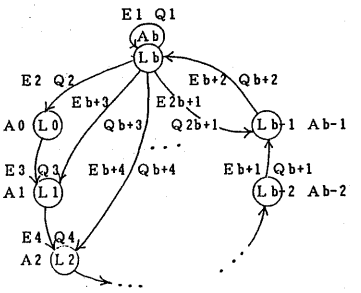

図2　タプル数均等分割法のＰＴＤ

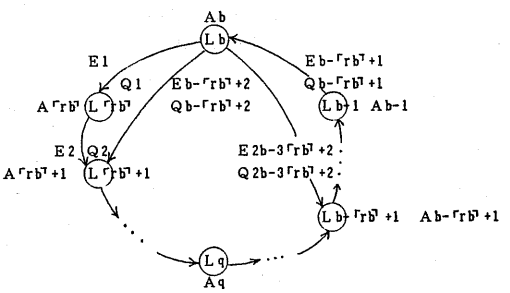

図3　ディスクリミネータ変更分割法のＰＴＤ

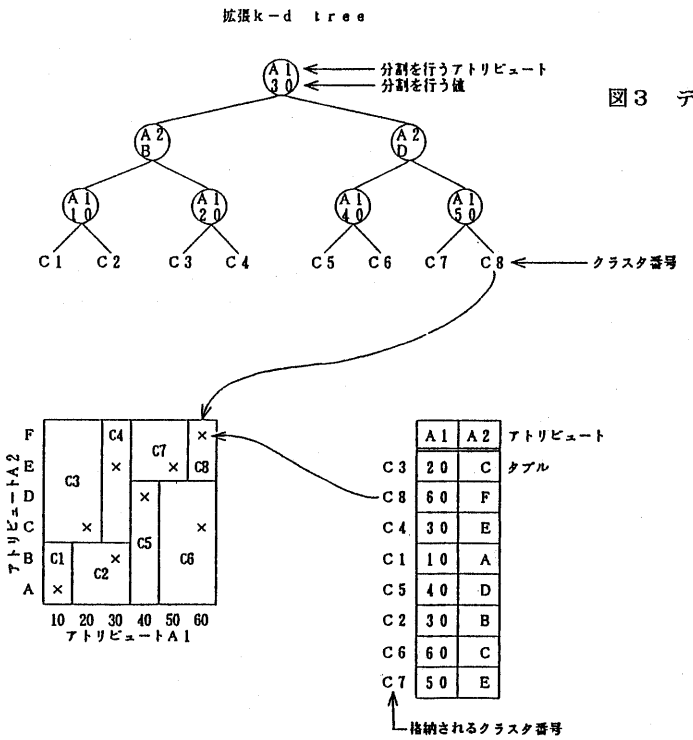

図1　拡張k-d treeによるクラスタリングの例

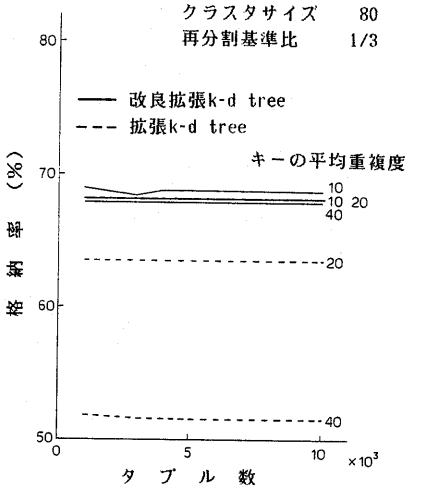

図4　状態遷移方程式による格納率の解析結果

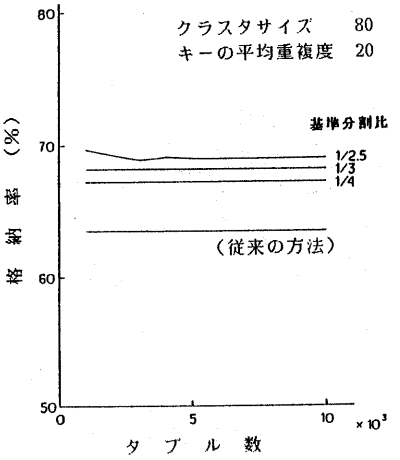

図5　ディスクリミネータ変更分割法の効果

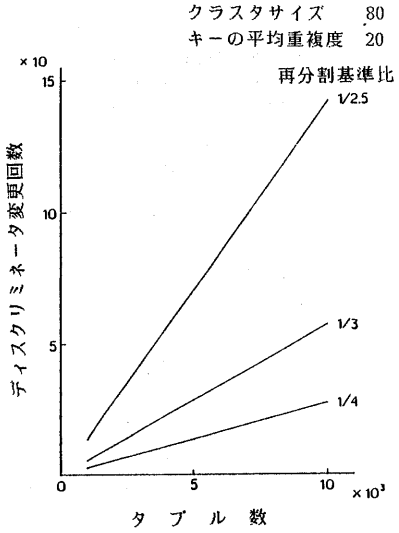

図6　ディスクリミネータの変更回数

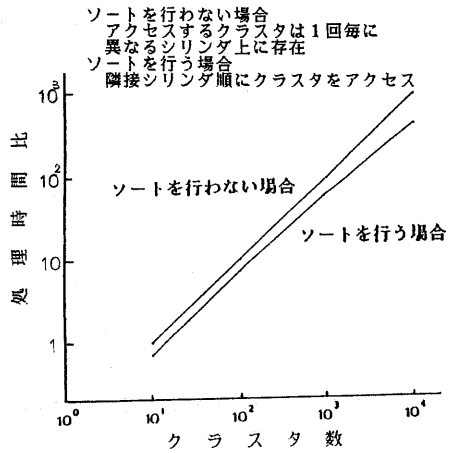

図9　ソートによる処理時間短縮の効果

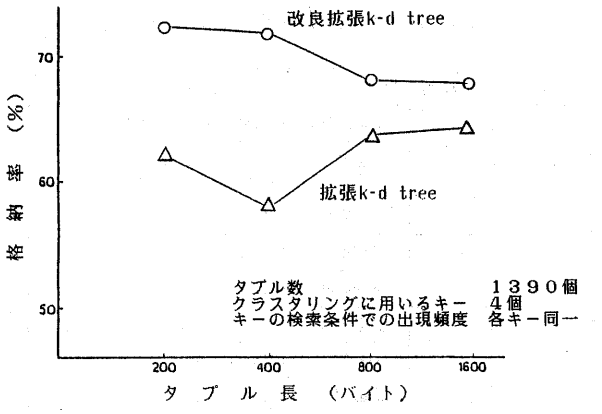

図7　分割方法と格納率

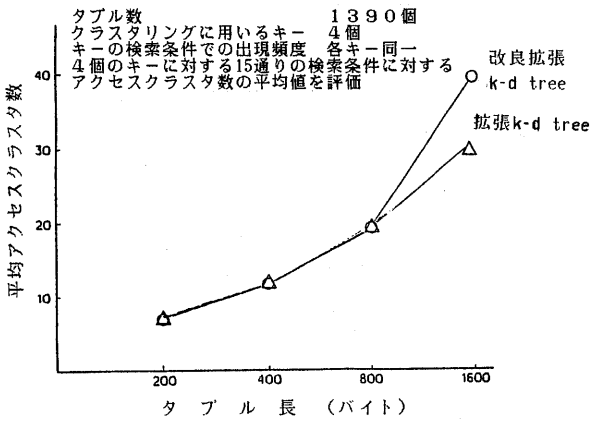

図8　分割方法と検索効率

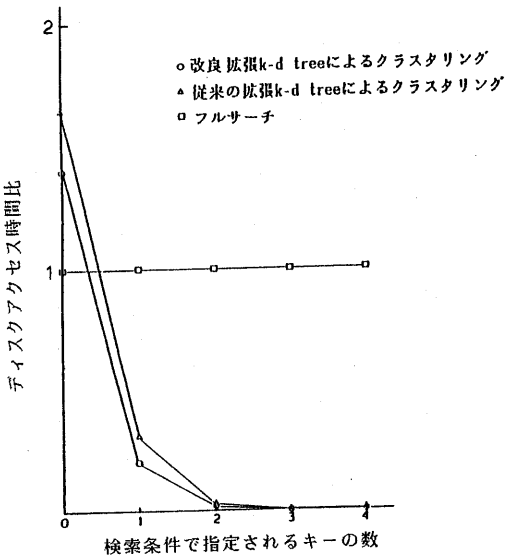

図10　改良拡張k-d treeの効果