画像データベースにおける索引構造の効率化

劉　強[†]　　北上 始[‡]　　黒木 進[‡]　　田村 慶一[‡]　森 康真[‡]

[†]広島市立大学大学院 情報科学研究科 〒731-3194 広島市安佐南区大塚東三丁目４番１号
[‡]広島市立大学 情報科学部 〒731-3194 広島市安佐南区大塚東三丁目４番１号
E-mail: {liuqiang,kitakami, kuroki,ktamura, mori}@db.its.hiroshima-cu.ac.jp

あらまし　大量の画像データベースからユーザの希望する画像の検索を高速化するために，著者らは，索引 SR-Tree に改良を加える方法について提案している. この方法では, SR-Tree の内部ノードに対して，ある効率的な情報表現が行われるので，同一ページ内にできるだけたくさんのエントリが格納される．これにより，キャッシュメモリアクセスの最適化が期待される．提案方法の有効性を評価するために，3000 枚の画像データから特徴量を抽出し，検索能力の評価を行った．その結果，提案方法が従来方法に比べて優れた検索能力をもっていることがわかった.

キーワード　画像処理，特徴ベクトル，索引，最近傍検索，多次元データ

Efficiency improvement of index structure in image-database

Qiang LIU[†]　Hajime KITAKAMI[‡]　Susumu KUROKI[‡]　Keiichi TAMURA[‡]

[†]Graduate School of Information Sciences,

[‡]Faculty of Information Sciences,

Hiroshima City University, 3-4-1 Ozuka Higashi, Asaminami-ku, Hiroshima 731-3194 Japan
E-mail: {liu,kitakami,kuroki,ktamura}@db.its.hiroshima-cu.ac.jp

Abstract　In order to efficiently find images needed by user from a large amount of image database, the authors propose a method for improving SR-Tree which is created as an index in multi-dimensional databases. Since the method is carried out an efficient information expression on each internal node included in SR-Tree, it is achieved to store entries as much as possible into the same page. Therefore, the authors expect to be optimized cash memory access using the method. Finally, they evaluate the capability of search for an image database that includes feature vectors extracted from 3000 images. The capability resulted in the proposed method being excellent more than the existing method.

Keyword　Image data processing, Feature vector, Index, Nearest Neighborhood Search, Multi-dimensional data

1. はじめに

ディジタルカメラや DVD の普及に従って，画像情報，映像情報の電子化はもはや日常的な存在であり，従来に無い画像や映像のデータベースが実現可能となりつつある．特に，ここ数年のプロセッサ技術，ストレージ技術の発展につれて，数年前には到底実現できなかった規模のデータベースを構築できるようになっている．それに伴い，キーワードによる検索機能だけでは十分ではなく，画像と映像の内容に基づく類似検索が求められている.

本稿では，大規模な画像のような多次元データを対象とした検索を高速化する索引技術について提案する. 画像や映像の大規模のデータベースを対象とした類似検索では，あらかじめ画像や映像を解析し特徴量を抽出しておくことで，特徴量どうしの比較によって類似しているものが検索される(図 1). 特徴量空間による類似性の判定は，距離の計算に過ぎないので比較的簡単な処理である．しかし，すべての特徴量を機械的に比較したのでは，データベースの規模に比例した処理が必要となる．たとえば，データベースの規模が 100 倍になると，同じく 100 倍の処理が必要となり，システムの処理効率(レスポンスやスループット)に深刻な影響を与える．そのため，特徴量空間に対する索引を構築し，類似検索を高速化することが必要になる.

内容に基づく検索で使われる特徴量空間は，次元数が高く，また，データ数の規模もはるかに大きいという特徴がある．1996 年ごろから，特徴量空間に特化した索引技術の提案が活発に行われるようになっている. 特徴量ベクトルを対象とした索引構造の技術的課題

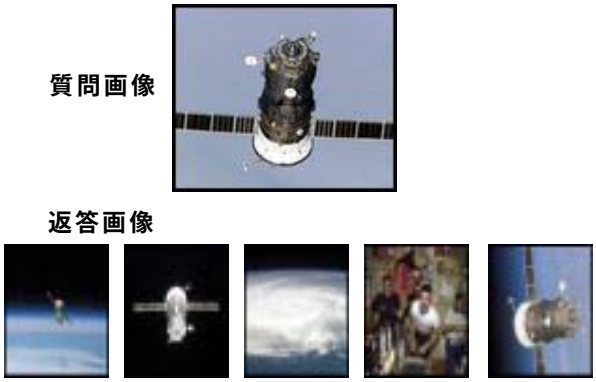


図 1 類似検索の例

としては以下の点が挙げられる.

- 多次元空間での最近接点検索を効率よく行えること.
- スケーラブルであること(大規模なデータセットに効率がよいこと).
- データの追加・削除を効率よくできること.
- 二次記憶を効率的に利用できること.
- 並行処理環境において効率的であること.

本稿では，二次記憶を効率的に利用するために，SR-Tree[8]の内部ノードにおける情報表現の効率化の方法を提案する．情報表現の効率化により，キャッシュメモリアクセスの最適化を可能にし，同一ページ内にできるだけたくさんのエントリを格納できるようになる．提案する方法の有効性を検証するために，画像データベースに対して，類似検索を行っている．

本稿の構成は以下のとおりである．2 章で従来の索引技術とキャッシュメモリの最適化技術について紹介を行う．3 章で SR-Tree および検索アルゴリズムについて詳しく説明を行う．4 章では従来のインデックスに対して，評価実験を行い，それらの問題点を明らかにする．5 章では提案する情報表現の効率化方法を説明し，またこれに対しての結果を解析する．次の６章で，情報表現の効率化を行った索引の性能を評価し，類似検索の高速化を検証する．最後の７章で，本論文のまとめを行う.

2. 関連研究

近年の RAM の低価格化に伴い，巨大なメインメモリを持つ計算機の入手が容易になった．そのため，メインメモリデータベースはさまざまな分野に適用されることが期待されており，メインメモリデータベースに関連する研究はますます盛んになってきた.

CPU の処理速度とメインメモリアクセス時間の差は大きく，その差は今後も更に拡大すると言われている．そのため，メインメモリデータベースでは，従来のデータベースのボトルネックであったディスク I/O に加え，メインメモリへのアクセスが新たなボトルネックとなる．メインメモリへのアクセス回数を減らすためのひとつの手段として，キャッシュメモリの利用が一般的に知られている．キャッシュメモリの記憶容量は小さいが，主記憶装置に比較してかなり高速にアクセスできるというメリットがある．CPU から要求されたデータがキャッシュメモリ中に存在する場合は，高速なキャッシュメモリからデータを読み込むことができる．しかしながら，データがキャッシュメモリ中に存在しない場合(キャッシュミス)は，目的のデータを取得するために，キャッシュに比べて低速なメインメモリにアクセスしなければならず，メインメモリへのアクセス時間がかかることになる．したがって，キャッシュミスの回数を減らすことで，処理全体の効率を向上させることができる.「キャッシュコンシャス」とは，キャッシュの効果的な利用を考慮して効率の向上を目指すことである.

キャッシュの動作を考えることは処理の高速化を考える上で重要な課題である．そのため，キャッシュを効果的に利用するための研究が盛んに行われている．木構造を配列により表現しているため，子ノードへのポインタを保持しないキャッシュコンシャスな木「Arrar-Based Cache conscious Trees(ABC-Tree)」[7]が提案されている．1999 年に Rao,J.と Ross,K.A.が提案しているキャッシュコンシャスな木「Cache Sensitive B+-Tree(CSB+-Tree)」[11]は子ノードへのポインタを削除する方法を適用している．リレーショナルデータベースにおいて，ページを分割し，分割された領域に属性のデータを格納することでキャッシュコンシャスなデータの格納方法を実現した．2002 年に Kihong Kim らが開発されているキャッシュコンシャスな木「Cache-conscious R-Tree(CR-Tree)」[4]では，R-Tree のキーである MBR(Minimum Bounding Rectangles)を相対化・量子化により圧縮し，キャッシュラインあたりの参照可能なキーの数を増加させることで，処理の効率化を実現している.

3. SR-Tree

3.1. インデックスの構造

SR-Tree(Sphere/Rectangle-Tree) [8]は従来のインデックスよりも更に高速な検索を実現するため，提案されたものである．木の特徴は，包囲球(bounding sphere)と包囲矩形(bounding rectangle)を併用する点にある．これまでにも，包囲球は SS-Tree[5]で，包囲矩形は R*-Tree[3]で使われている．しかし，次元が高くなった場合，包囲矩形を用いる方法では１辺の長さと対角線の長さの差は，次元が高くなるにつれて大きくなる．たとえば，１辺の長さが１である超立方体の場合，次元数が D のとき，対角線の長さは $\sqrt{D}$ であ

り，2次元の正方形の場合には $\sqrt{2}$ であるが，16次元の超立方体の場合には4になる．そのため包囲矩形で領域を分割した場合，体積としては小さくても，領域内の点相互の距離が大きくなるという問題がおきる．また体積については，R*-Treeのほうが SS-Treeよりも小さく，およそ 1/50 になっていることがわかる．ところが，最大径については，SS-Treeがおよそ 1.5 であるのに対して R*-Treeでは 2.5 と，逆に R*-Treeの方が大きくなっている．つまり，包囲球を用いる方法では距離的に近い空間だけを取り出すことができるが包囲長方形よりも体積が大きくなりやすいため，領域間の重なりが大きくなる可能性が高くなる．これは最近接検索や範囲検索の性能を下げる原因となる．したがって，包囲球がすべての点で包囲矩形に勝っているわけではなく，包囲球にも体積が大きくなるという問題点がある．

そこでSR-Treeではこれらを同時に使用することによって，多次元空間を R*-Tree や SS-Tree よりも効果的に分割することを可能にする．このように二つの形状を併用した場合，インデックス構造の管理に必要なデータの大きさが増えるため，ノードあたりの枝の数が小さくなり，木が高くなるという欠点がある．しかし，片山氏らの評価実験の結果，CPU時間，ディスクアクセス回数，いずれの面についても，SR-Tree が R*-Tree ならびに SS-Tree を上回る性能を持つことが確認されている[8].

SR-Tree では包囲矩形と包囲球を組み合わせて用いる(図 2)．したがって，SR-Treeでは，このように包囲矩形と包囲球の共通部分によって,領域が区分される．

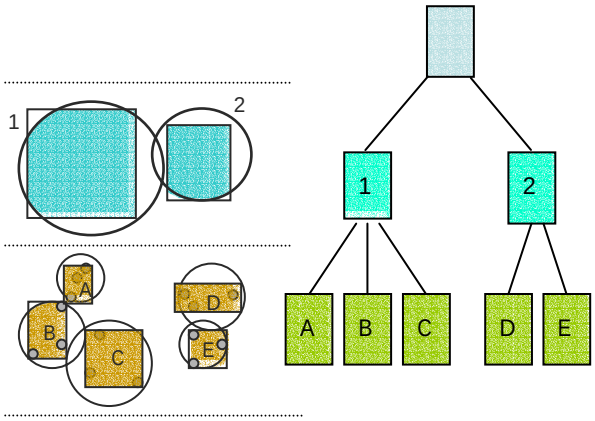


図2 SR-Tree構造

まず，リーフの構造を示す．

$L:(E_1,\ldots,E_n)$　ただし，$(m_L \le n \le M_L)$である．

$E_i:(p,data)$

リーフ L は，n個 $(m_L \le n \le M_L)$ のエントリ $E_1,\ldots,E_n$ を格納したものである．m_L，M_L は，1 リーフあたりに格納するエントリの数の下限ならびに上限である．それぞれのエントリには，キーとなる点の座標 p とそれに付与するデータ data を格納する．

次に，ノードの構造を示す．

$N:(C_1,\ldots,C_n)$　ただし，$(m_N \le n \le N_N)$である．

$C_i:(CD,S,R,w,\mathrm{child\_pointer})$

ノード N は，n個 $(m_N \le n \le M_N)$ のエントリ $C_1,\ldots,C_n$ を格納したものである．ひとつのエントリがひとつの子供に相当し，m_N，M_N は，1 ノードあたりに格納するエントリの数の下限ならびに上限である．個々のエントリには，子供へのポインタ child_pointer に加えて，その重心 CD，その包囲球 S，包囲矩形 R，そして，その子供を頂点とする部分木が格納している点の総数 w を格納する．この構造は，SS-Tree と比べると包囲矩形 R が加わった構造になっており，R-Tree と比べて包囲球 S とデータ数 W が加わった構造になっている．また，各エントリに含まれているすべての点の重心を付け加えている．

3.2. 検索アルゴリズム

最近接検索は，与えられた質面点に最も近い点を探す検索であり，画像などから得られた特徴ベクトルを類似検索する場合など，マルチメディアアプリケーションにおいて広く使われている．検索アルゴリズムの詳細は，文献[9]で説明されているので，ここでは，その概略だけを述べる．多次元インデックスを利用しながら最近接検索を行う方法は二種類あり，深さ優先の探索法[9]と幅優先の探索法[10]が提案されている．基本的な考え方はどちらも同じであり，質問点に近い領域から順に検索結果の候補を集めながら探索し，集めた候補よりも近いも探索領域が無くなった時点で探索を終える．そして，探索を終えた時点での候補集合が検索結果になる．このような探索を行う場合，中間ノードや葉ノードを質問点から近い順に整列した優先順位付き待ち行列が必要になる．根ノードから出発し，中間ノードを訪れるたびに子ノードを待ち行列に加え，待ち行列の先頭から順に探索していくのである．深さ優先の探索法と幅優先の探索法の相違点は，この待ち行列の作り方であり，その結果，木のたどり方が深さ優先と幅優先になっている．

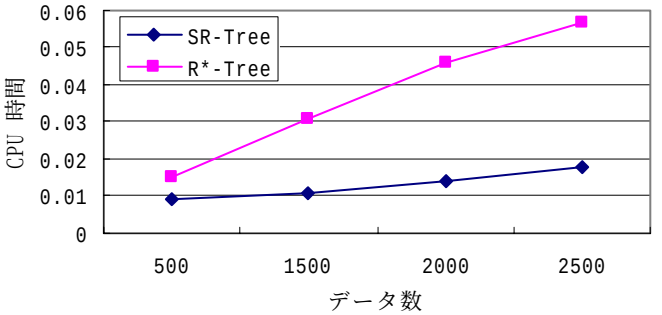
これらの探索法は一長一短であり，必ずしも一方が優れているとはいえない．訪れるノードの数については，幅優先の探索法が最適であり，必ず最小限のノードしか訪れないことが明らかになっている．ところが幅優先の探索法には待ち行列が長くなるという問題がある．一方，深さ優先の探索法では，中間ノードごろに待ち行列を作るので，待ち行列の長さは高々，中間ノードあたりの子ノードの数にしかならない．そのため，演算コストの面では，むしろ幅優先の探索の方が有利なのである．

そこで，SR-Tree では深さ優先の探索法を用いている．探索は，2 段階に分かれており，まず，指定された数に達するまで点を集め，検索結果の候補集合を作る．そして，候補集合に含まれている点の存在範囲を探索範囲として，より近い点が無いかどうか調べる．より近い点が見つかった場合には，そのつど，候補集合を作り直し，探索範囲を絞りながら探索を続ける．最後に，探索範囲に含まれるリーフをすべて探索し終わった時点で，検索が終了する．

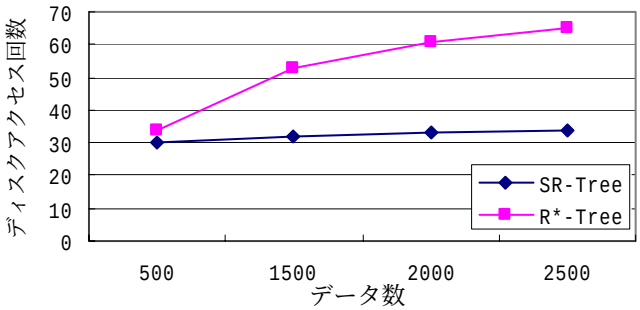
4. 従来方法に対する評価実験

従来のインデックスを多次元データに適用した場合の問題点を明確にするために評価実験を行った．実験には，R*-Tree と SR-Tree の二つのインデックスを用いた．R*-Tree は，包囲長方形を用いる空間インデックスの中で代表的なものであり，R-Tree よりも性能がよいとされている．評価実験に使うる実データは，画像データから実際に抽出した特徴ベクトルであり，色相を分割して生成したヒストグラムデータである．

これらのデータに対してインデックスを構築し，最近接検索に要する CPU 時間ならびにディスクアクセス回数を測定した．測定した検索は，データセット中の一つのデータを検索点として，それに近接する 20 個のデータを探すというものである．無作為に選んだ 30 個の点について検索を行い，それらの平均値を測定結果とした．最近接検索のためのアルゴリズムは，文献[9]のものを用いた．



(a) CPU 時間



(b) ディスクアクセス回数

図 3 最近接検索の実験

実験に使用した計算機環境は，Intel celeron2.4GHz，メモリー：258MB，HDD：20.0GB，OS：WindowsXP，ツール：Cygwin である．ディスクブロックの大きさは 8192Byte とし，リーフのエントリに付随するデータ領域の大きさは 512Byte である．最近接検索の性能についての実験結果を図 3 に示す．

これらの結果から，どちらのデータに対しても，SR-Tree が R*-Tree を上回っていることがわかる．データの数が 2000 のとき，CPU 時間が 25%，ディスクアクセス回数が 51%になっている．以上の結果から，SR-Tree は，R*-Tree よりも最近接検索については，高い性能を持つことがわかる．

しかし，多次元点データを扱う場合，ノードのエントリの大きさが大きくなり，ノードあたりの枝の数(fanout)が小さくなってしまうという問題点が指摘されている.SR-Tree の場合，包囲球と包囲長方形の両者をエントリに格納するため，ノードエントリのサイズが，包囲球のみを格納する SS-Tree に対して約 3 倍，包囲矩形のみを格納する R*-Tree に対して約 1.5 倍になる．したがって，ノードあたりの枝の数が R*-Tree に比べて 2/3 になり，木の高さが高くなったり，検索の際に読み込むノード数が増加することが考えられる．

5. 索引構造の効率化

以上のことから，データベースシステムにおいて索引技術は，データ処理の効率を向上させる手法の一つであり，メインメモリデータベースにおいても同様の効果を期待することができる．近年の研究において，キャッシュコンシャスな索引構造は優れた性能を示している．従来のデータベースシステムでは，ディスク I/O のボトルネックに注目し，ディスク I/O を減らす研究が行われてきた．しかし，メインメモリデータベースシステムでは，キャッシュミスがボトルネックとなる．そこで，索引技術にキャッシュコンシャスの概念を取り入れることで処理効率の向上を望むことができる．

キャッシュコンシャスな木を実現する手法の一つとしてポインタの削除がある．削除したポインタの変わりにデータを格納することで，ノードあたりのデータ数は増加する．これにより，ノードを参照する際の計算に不必要となるポインタをメインメモリからキャッシュメモリに読み込まずにすむ．そして検索に必要となるデータのみがキャッシュに読み込まれることでキャッシュミスの回数は減り，結果的に高速に処理することが可能となる．また，ノードあたりのデータ数は増加する．その結果として，木全体の高さは低くなる.これにより，検索においてノードを参照する回数，つまりノードをキャッシュに読み込む回数が減り，キャッシュミスの回数も減る．

多次元データの索引として提案された SR-Tree のフ

ァンアウトを解消し，検索速度の向上を行うために，内部ノードにおける情報表現の効率化を行う．情報表現の効率化によりキャッシュメモリアクセスの最適化が行われ，同一ページ内にできるだけたくさんのエントリを格納できるようになると考える．片山氏らが考案された索引木の内部ノードでは図 4(a) で示している．

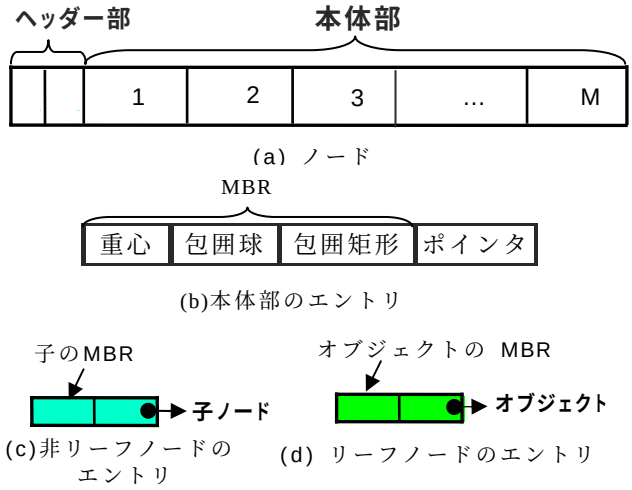


図 4 ノードの構造

ノードの本体部では，図に示しているように M 個のエントリを格納することができる．本体部エントリは MBR とポインタから構成される．MBR は図 4(b)で示されるように重心と包囲球と包囲矩形の三つから成る．ヘッダー部にノードのタイプ(リーフまたは非リーフ)とエントリ数が含まれる．ノードのタイプに対応して，本体部のエントリでは非リーフの場合(図 4(c))とリーフの場合(図 4(d)が二つある．エントリの大きさは，そのエントリに属するすべて点の重心の大きさと包囲矩形の大きさと包囲球と子ノードへのポインタの大きさを足したものである．

entry を本体部のエントリとし，Cent をエントリの重心とし，Shp をエントリの包囲球とし，Rect を包囲矩形とし，pをエントリのポインタとし，coord を座標とし，radius を中心の直径とする場合，一つのエントリの大きさは次式で表すことができる．

entry.サイズ=Cent.サイズ+Shp.サイズ
+Rect.サイズ+p.サイズ
Cent.サイズ=coord.サイズ*次元数
Sph.サイズ=coord.サイズ*次元数+radius.サイズ
Rect.サイズ=coord.サイズ*次元数*2

たとえば，特徴量ベクトルの次元数を 8 とし，各座標のサイズを 8 バイトとする場合，重心の大きさが 64 バイト，包囲球の大きさが 72 バイト，包囲矩形の大きさが 128 バイトとなる．子ノードへのポインタの大きさと合わせると一つのエントリのサイズは，272 バイトとなる．

SR-Tree の実験から，ほとんどのノードに格納されている重心が包囲球の中心とほぼ同じであることをわかっている．そこで，我々はノードにある重心を包囲球の中心と同じに見なし，それを取り除くことによって，MBR の情報表現の効率化を図っている．

次に，情報表現の効率化を具体的に解析する．重心を削除する場合，上記に示しているように簡単に計算すると,一つのエントリでは 64 バイトの空間を獲得することができる．つまり，一つのノードに格納できるエントリ数が 10 である際,得られる空間が 640 バイトとなり，すなわち，2 個のエントリを格納できる空間が得られる．一般に示すと，以下のようになる．各座標のサイズを m とし，次元数をｄとし，各葉ノードに格納できる最大エントリの数をｆとし，オブジェクトの総数を N とすると得られる空間 S が次に，

$$S = m \times d \times (N / f^2 + N / f)$$

となる．式を見ると，得られる空間 S が次元数とオブジェクトの総数 N の増加につれて増加し，平均なエントリ数ｆにより減少することがわかる．また，情報表現を効率化することは，重心と包囲球の中心とを同じみなすことである．重心を削除した後では，データ挿入と削除に使われる重心が包囲球の中心のように得られる．従って，重心を取り除くことによる問題が生じないと考える．

6. 性能評価

提案した方法に対して 3 章で行われた実験と同じ実験を行い，結果を測定した．検索手法は広く使われているカラーヒストグラムに基づく方法であり，画像データは NASA(アメリカ航空宇宙局)で公開されているスペースシャトルの静止画像を 3000 枚使用した.色空間としては，HSL カラーモデルを使い，色相で分割した 24 色を用いてヒストグラムを求めた.特徴ベクトル間の類似度はユークリッド距離によって計算した．ここでは，提案した方法により作成される木を CSR-Tree と呼ぶ．この実験における最大エントリ数並びに木の高さは，表 1，表 2 に示すとおりである．

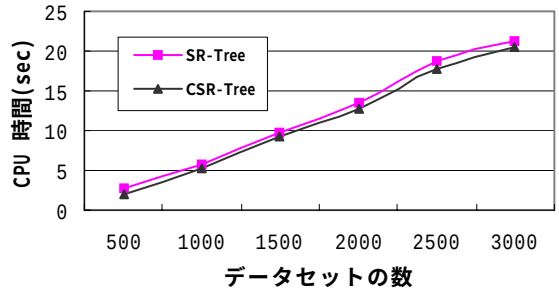
表 1 木の高さ(24 次元)

		データ数			
		500	1500	2000	2500
R*-Tree	高さ	3	3	3	4
SR-Tree	高さ	3	3	3	4
CSR-Tree	高さ	2	3	3	4

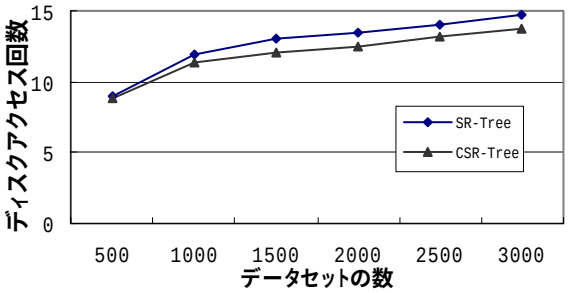
表 2 最大ノード数と最大リーフ数(24 次元)

Index	Node	Leaf
R*-Tree	24	10
SR-Tree	13	25
CSR-Tree	14	25

まず，木の構築に要するコストについて評価する．図 6 には，データの挿入に要した CPU 時間ならびにディスクアクセス回数(ブロック読み込み回数とブロック書き込み回数の合計)を示している．また，時間を評価するには，同じ実験を数回行って，それらの平均を算出したものである．ディスクアクセス回数については，データの挿入に対しての一回あたりの平均値を示している．



(a) CPU 時間

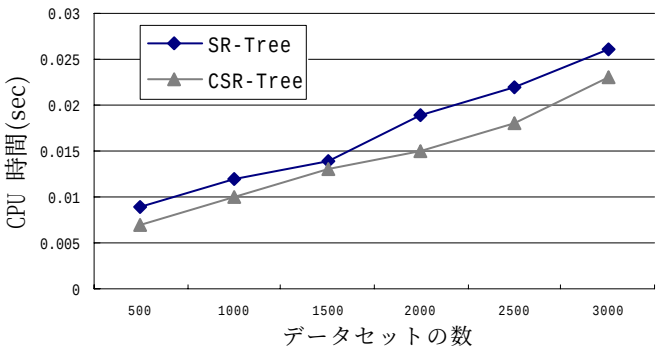


(b) ディスクアクセス回数

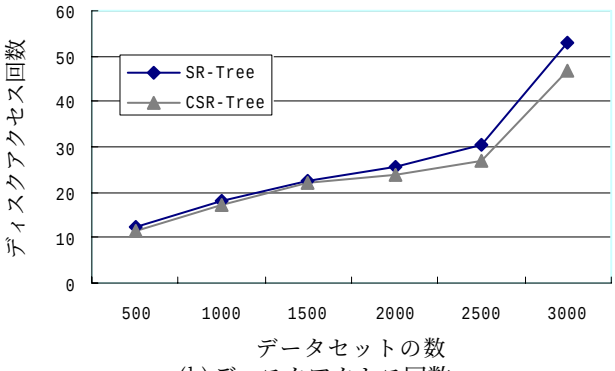
図 6 木の構築

これらの図から，CSR-Tree は，SR-Tree よりも短い CPU 時間で木を構築できることがわかる．また，CSR-Tree と SR-Tree を比べると，CSR-Tree の方が SR-Tree よりもわずかだが少ない CPU 時間とディスクアクセスを必要としていることがわかる．これは，内部ノードの情報表現の効率化によりノード内に格納できるエントリの数が増え，ノードの分割やノードの再挿入を起こる回数を減らしたことによるものと考えられる．ひとつのノードあたりに格納できるエントリの数が増えたことから，キャッシュメモリに蓄積できるエントリの数が増えることが考えられる．これにより，キャッシュアクセスのミスも減らすことができる．前述したように，キャッシュアクセスのミスを減らすことで，メインメモリへのアクセス回数も減少することため，性能の向上につながったといえる．なお，データの総量が増えることにつれ，CPU 時間とディスクアクセス回数の増加率が高くなっていることもわかる．

次に，最近接検索の性能について評価する．実験結果を，図 7 に示す．測定した検索は，データセット中の一つのデータを検索点として，それに近接する 20 個のデータを探すというものである．無作為に選んだ 30 個の点について検索を行い，それらの平均値を測定結果とした．最近接検索のためのアルゴリズムは，文献[9]のものを用いた．



(a) CPU 時間



(b)ディスクアクセス回数

図 7　検索の性能

これらの結果から，どちらのデータに対しても，CSR-Tree の性能が SR-Tree 性能を上回っていることがわかる．これは，CSR-Tree が内部ノードの情報表現の効率化により格納できるエントリの数を増やしたからである．つまり，ノードのエントリが増えたことで，キャッシュメモリに蓄積できるノードのエントリ数も増加すると考えられる．その結果，キャッシュメモリにアクセスするだけですむ処理が増え，メインメモリへのアクセス回数が減少する．これによって CSR-Tree が SR-Tree よりも良い性能を得られたと考えられる．

7. まとめ

本稿では，多次元データのためのインデクシング手法について性能評価を行い，従来方法の問題点を指摘するとともに，多次元インデックスとして提案された

SR-Tree[8]を用いて，索引構造の情報表現を効率化し，その性能評価を行った．

本稿は SR-Tree の内部ノードに着目し，情報表現を効率化する方法を提案した．内部ノードの情報表現を効率化することにより，同じ内部ノードに格納できるエントリの数が増えた．そして，検索時のディスクアクセス回数を減らすことで検索の性能を向上させることができた.画像から抽出した 24 次元データについて評価実験を行った．データ数を変化させた場合では，インデックス構築に要する CPU 時間に対して CSR-Tree が SR-Tree よりも少なかった．SR-Tree より CPU 時間が 20%,ディスクアクセス回数がおよそ 10%減少した．類似検索の性能でも SR-Tree の性能を上回って, CPU 時間とディスクアクセス回数が 10%減少した．次元数を変えた場合でも SR-Tree の性能を上回っていたことが確認された．これらの結果から提案手法である CSR-Tree が SR-Tree より性能が良いと言える．

このように索引構造の情報表現を効率化することにより，ひとつのノードあたりに格納できるエントリの数が増えたことで，キャッシュメモリに蓄積できるエントリの数が増えることが考えられる.これにより，キャッシュアクセスのミスも減らすことができると考えられる．前述したように，キャッシュアクセスのミスを減らすことで，メインメモリへのアクセス回数も減少し，性能の向上につながると考えられる．以上により,CSR-Tree が SR-Tree を上回っている性能を持ち，最近接検索を高速化することに成功した原因と考えられる．

謝辞

本研究の一部は広島市立大学･特定研究費(一般研究費(コード番号:3106))，文部科学省科学研究費補助金(課題番号:16700114)の支援により行われた．

文　献

[1] H.D.Wactlar, T. Kanade, M.A. Smith, and S.M. Stevens, “Intelligent access to digital video: Informedia project,” IEEE Computer, Vol.29, no.5, pp.46-52, May 1996.

[2] 金出武雄，佐藤真一，“ Informedia: CMUディジタルビデオライブラリプロジェクト，”情報処理，Vol.37 , no.9, pp.841-847, Sept. 1995.

[3] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, “The R*-Tree: An efficient and robust access method for oints and rectangles,” Proc. ACM SIG-MOD, Atlantic City, USA, pp.322-331, May 1996.

[4] K. Kim, S.K.Cha, and K. Kwon, “Optimizing multidimensional index Trees for main memory access”, Proc. ACM SIGMOD, pp.475-586 (2001).

[5] D.A. White and R. Jain, “Similarity indexing with the SS-Tree”, Proc, of the 12th Int. Conf. on Data Engineering, New Orleans, USA, pp.516-523, Feb. 1996.

[6] J.T. Robison, “The K-D-B-Tree: A search structure for large multidimensional dynamic indexes,” Proc. ACM SIGMOD, Ann Arbor, USA, pp.10-18, April, 1981.

[7] 高見澤秀久，有次正義，“配列を用いたキャッシュコンシャスな索引木の提案，” 日本データベース学会Letters Vol.1, No.1 pp.11-14，2002年10月.

[8] Norio Katayama and Shin‘ ichi Satoh, “ The SR-Tree: An Index Structure for High-Dimensional Nearest Neighbor Queries, ” Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data (May 1997) pp. 369-380.

[9] N. Roussopoulos, S. Kelley, and F. Vincent, “Nearest nerighbor queries,” Proc. ACM SIGMOD, San Jose, USA, pp.71-79, May 1995.

[10] G. Hjaltason and H. Samet, “Ranking in Spatial Databases,” 4th International Symposium, SSD’95, Portland, USA, pp.83-95, August 1995.

[11] J. Rao, K. A. Ross, “Making B+-Trees Cache Conscious in Main Memory”, Proceedings of ACM SIGMOD Conference. 2000. pp.475-576.