THE TRANSACTIONS OF THE INSTITUTE OF ELECTRONICS, INFORMATION AND COMMUNICATION ENGINEERS D-I

# 電子情報通信学会 論文誌 D-I

VOL.J80-D-I NO.8 AUGUST

情報・システムI—コンピュータ 1997

情報・システムソサイエティ

社団法人 電子情報通信学会

THE INFORMATION AND SYSTEMS SOCIETY

THE INSTITUTE OF ELECTRONICS, INFORMATION AND COMMUNICATION ENGINEERS

論　文

# SR-Tree：高次元点データに対する最近接検索のためのインデックス構造の提案

片山　紀生[†*]　　佐藤　真一[†**]

# SR-Tree: *An* Index Structure for Nearest Neighbor Searching of High-Dimensional Point Data

Norio KATAYAMA[†*] and Shin'ichi SATOH[†**]

**あらまし**　画像データに対する内容検索の実現法として，特徴ベクトルを類似検索する方法が広く使われており，その高速化のためのインデックス構造が求められている．これまでにR*-treeを用いる方法や，新たに考案されたSS-treeを用いる方法が提案されているが，本研究では，これらのインデックスよりも更に高速なインデックスとしてSR-tree（Sphere/Rectangle-Tree）を提案する．SR-treeの特徴は，包囲球（bounding sphere）と包囲長方形（bounding rectangle）を併用する点にある．これまでにも，包囲球はSS-treeで，包囲長方形はR*-treeで使われている．しかし，本研究が行った実験によると，次元が高くなった場合，包囲長方形を用いる方法では1辺の長さと対角線の長さの差が大きくなるという問題があり，包囲球を用いる方法では包囲長方形よりも体積が大きくなるという問題があることがわかった．そこで，SR-treeではこれらを同時に使用することによって，高次元空間をR*-treeやSS-treeよりもより効果的に分割することを可能にする．評価実験を行った結果，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を上回ることが確認された．

**キーワード**　多次元インデックス構造，R*-tree，SS-tree，SR-tree，最近接検索

## 1. まえがき

画像データに対する内容検索の実現法として，画像データの特徴ベクトルを類似検索する方法が広く使われており，その高速化が期待されている[1]．例えば，カーネギーメロン大学のディジタルビデオライブラリプロジェクトであるInformediaプロジェクト[2],[3]では，画像照合によるビデオ映像の内容検索機能の導入を進めており，10数次元程度の特徴ベクトルに対する高速な類似検索の実現が求められている．画像検索で用いられる特徴ベクトルは高次元の点データであり，類似検索を高速化するためには最近接検索（nearest neighbor search）を実現する多次元インデックスが必要となる．

これまでにR*-tree [4]などの空間インデックスを利用する方法[1]や，新たに考案されたSS-tree [5]を用いる方法が提案されているが，本研究では，これらのインデックスよりも更に高速なインデックスとしてSR-tree（Sphere/Rectangle-tree）を提案する．SR-treeの特徴は，包囲球（bounding sphere）と包囲長方形（bounding rectangle）を併用する点にある．これまでにも，包囲球はSS-treeで，包囲長方形はR*-treeで使われている．しかし，本研究が行った実験によると，次元が高くなった場合，包囲長方形を用いる方法では1辺の長さと対角線の長さの差が大きくなるという問題があり，包囲球を用いる方法では包囲長方形よりも体積が大きくなるという問題があることがわかった．そこで，SR-treeではこれらを同時に使用することによって，高次元空間をR*-treeやSS-treeよりもより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

† 学術情報センター研究開発部，東京都
Research and Development Department, NACSIS (National Center for Science Information Systems), Tokyo, 112 Japan

* 日本学術振興会による「海外の中核的研究拠点への派遣研究者」として，1996年3月18日より同年4月3日まで，カーネギーメロン大学ロボティクス研究所に訪問研究員として滞在．本論文はこの間の研究成果に基づくものである

** 1995年4月20日より1997年4月19日まで，文部省在外研究員としてカーネギーメロン大学ロボティクス研究所に滞在

の管理に必要なデータの大きさが増えるため，ノード当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，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を上回る性能をもつことが確認された．

本論文の構成は，以下のとおりである．まず，2.で関連研究について説明する．次に3.では，R*-tree, SS-treeなど従来のインデックスに対する性能評価の結果を示すと共に，これらのインデックスを高次元データに適用する上での問題点を指摘する．そして，4.で本論文が提案するSR-treeの説明を行い，5.でSR-treeに対する評価結果について説明する．

## 2. 関連研究

### 2.1 K-D-B-Tree

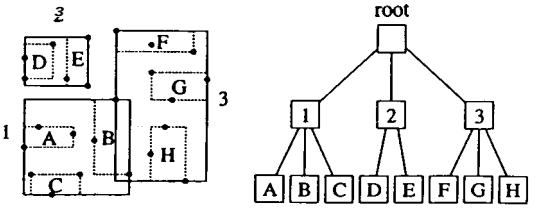
K-D-B-tree [6]は，多次元の点データに対するインデックスであり，$B^+$-treeと同様にバランスされた木構造をもつ．この木構造は，空間を座標軸と直交した平面で再帰的に分割することによって作られる．

K-D-B-treeの特徴は，空間を独立した領域に分割することである．そのため，点が属する領域が唯一決まり，点検索（point search）の場合にたどる枝が1本だけになる．よって，検索時間は木の高さのみで決まり，データ数に対して対数オーダの時間で検索できる．

しかし，K-D-B-treeには，強制分割（forced split）の問題がある．データの投入順序に応じて，既存の領域をまたぐ形で分割しなければならない場合があり，またいでいる領域以下の部分木を強制的に分割するのである．強制分割を行うと，空のリーフや，空に近いノードやリーフが生じる可能性があり，ノードやリーフの最小利用率を保証できなくなってしまう．そのため，木が高くなったり，検索の際にたどるノードやリーフの数が増えるなど，検索性能を下げる要因となる．

### 2.2 R-Tree

R-tree [7]は，長方形をキーとする空間インデックスであり，K-D-B-treeと同様にバランスされた木構造をもつ．しかし，長方形は点と違って広がりがあるため，K-D-B-treeのように互いに独立な領域に分割して格納することができない．そこで，R-treeでは包囲長方形によって領域を管理する．すなわち，リーフにはキーである長方形とそれに付随するデータを格納する．一方，ノードには，下位ブロックへのエントリとして，




□1 R-treeの構造
Fig.1 The R-tree structure.

下位ブロックへのポインタと，それ以下の部分木に格納されているキーに対する包囲長方形を格納する．つまり，個々のノードには，それ以下の部分木に対する包囲長方形が割り当てられるのである．これにより範囲検索（region search）の際，検索範囲と包囲長方形とが重なるかどうかで，たどるべきノードを判定することができる．

R-treeは，もともと空間インデックスとして提案されたものであるが，点データのみを格納して，多次元インデックスとして利用することもできる．この場合，K-D-B-treeとの相違点は，領域を包囲長方形で管理すること，ならびに，分割された領域に重なりを許すことである（図1）．領域に重なりが生じると，点検索の際に複数の枝をたどらなければならなくなる．そのため，R-treeでは，点検索の検索時間は木の高さのみでは決まらない．しかし，強制分割の必要がないので，リーフやノードの最小利用率を保証できるという利点がある．

### 2.3 R*-Tree

R*-tree [4]は，R-treeのアルゴリズムを改良したものである．どのリーフに長方形を挿入すべきか判定するアルゴリズムと，あふれたノードやリーフを分割するアルゴリズムを改良し，更に，ノードやリーフの分割の際にエントリの一部を強制的に再投入する手続き（forced reinsert）を加えることによって，ノードにおける包囲長方形の重なりを小さくすることに成功している．

### 2.4 VAMSplit R-Tree

VAMSplit R-tree [8]は，与えられたデータセットに対して，最適化されたR-treeをトップダウンに生成するためのアルゴリズムである．基本的なアイディアは，主記憶上の多次元インデックスであるk-d tree [9]に対するアルゴリズムに基づいており，座標の分散が最大である軸と直交する平面で，データセットを再帰的に2分していくことによって木構造を構築する．こ

のようなアルゴリズムは既に k-d tree の構築法として使われているが[10]，VAMSplit R-tree ではこれを R-tree の構築に適用すると共に，分割位置を完全な中央値ではなく適切にずらすことによって，ブロックの利用率を高める方法を提案している．

### 2.5 SS-Tree

SS-tree [5] は，高次元点データに対する最近接検索の高速化を目的として提案されたインデックスであり，R*-tree を改良したものである．構造は R*-tree と同様であるが，領域を包囲長方形ではなく，領域内の点の平均位置（重心）を中心とする包囲球によって管理する点に特徴がある（図 2）．

包囲球を用いる利点は二つある．まず，ノード当りの枝の数（fanout）を増やすことができる．長方形の場合，一つの領域を表すのに次元数の 2 倍の座標値が必要だが，球の場合，次元数の座標値と半径の値だけで済む．次に，包囲球を用いた場合，中心からの距離を基準に領域を構成できるので，距離的に近い点を同一の領域に集めることができる．これに対して，包囲長方形の場合，次元が高くなるにつれて 1 辺の長さと対角線の長さに隔たりが生じるため，同一領域に属する点が必ずしも距離的に近いとは限らなくなる．

また，SS-tree には，領域内の点の平均位置（重心）に基づいて木を構築するという特徴がある．すなわち，点をどの部分木に挿入するか選択する際に，点から部分木の重心までの距離が最小であるものを選択する．また，ノードやリーフを分割する際に，それぞれの軸方向についての分散を重心から計算し，VAMSplit R-tree と同様に，分散が最大である軸と直交する平面で分割する．このようにして木を構築することにより，距離的に近い点を同一の領域に集めることが可能になり，最近接検索の高速化が実現されている．

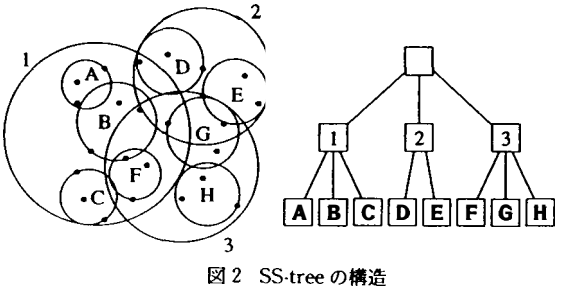



図 2 SS-tree の構造
Fig. 2 The SS-tree structure

## 3. 従来法の問題点の検証

### 3.1 従来法に対する評価実験

本研究では，従来のインデックスを高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，K-D-B-tree，R*-tree，VAMSplit R-tree，SS-tree の四つのインデックスを用いた．K-D-B-tree は，点データに対する多次元インデックスとして代表的なものであり，空間を独立な領域に分割するという特徴がある．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 [4]．VAMSplit R-tree は，R-tree の最適構成法の一つであり，R*-tree や SS-tree を上回る性能が得られたことが報告されている [8]．SS-tree は，最近接検索のための多次元インデックスとして提案されたものであり，R*-tree を上回る検索性能をもつとされている [5]．

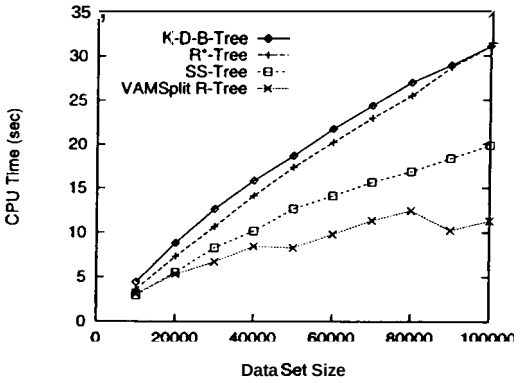
評価実験は以下の 2 種類のデータセットに対して行った．データの次元数はいずれも 16 次元である．

（1） 一様分布データ
（2） 実データ

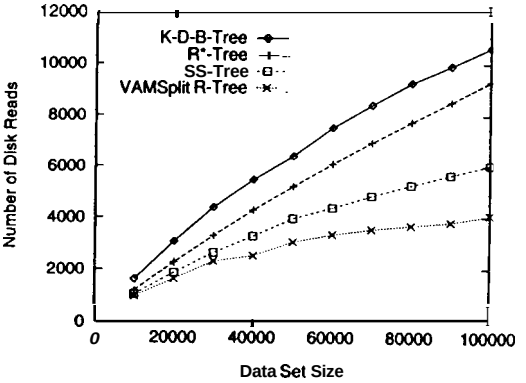
一様分布データは，乱数を用いて生成したデータセットであり，各軸の [0, 1) の範囲に点が一様に分布したものである．一方，実データは，画像データから実際に抽出した特徴ベクトルであり，ニュース映像から取得した画像について，色相を 16 分割して生成したヒストグラムデータである．

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 21 個のデータを検索するというものである．無作為に選んだ 1,000 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 [11] のものを用いた．

実験に使用した計算機は Sun Microsystems 社の SPARCstation-20 (CPU: HyperSPARC 125 MHz，主記憶：224 Mbyte，OS: Solaris 2.4) である．プログラム言語には C++ を用いた．ディスクブロックの大きさは OS のブロックサイズに合わせて 8192 byte とした．リーフのエントリに付随するデータ領域の大きさは 512 byte であり，ノードならびにリーフの最大エントリ数は表 1 に示すとおりである．最小エントリ数については，R*-tree の文献 [4] ならびに SS-tree の文

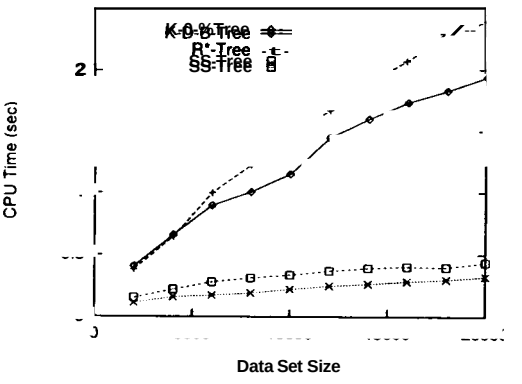



(a) CPU time

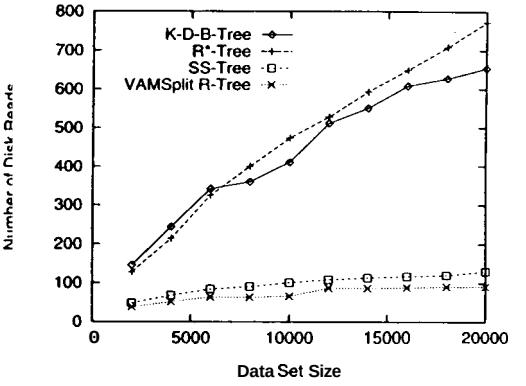



(b) Number of disk reads

図 3 従来法に対する性能評価（一様分布データの場合）
Fig. 3 Performance of indices (uniform distribution data).




(a) CPU time




(b) Number of disk reads

図 4 従来法に対する性能評価（実データの場合）
Fig. 4 Performance of indices (real data).

表 1 ノードならびにリーフの最大エントリ数（16 次元）
Table 1 Maximum number of entries in a node and a leaf (16 dimensions).

| Index | Node | Leaf |
|---|---|---|
| K-D-B-tree | 30 | 10 |
| R*-tree | 3 | 10 |
| VAMSplit R-tree | 3 | 10 |
| ss-tree | 5 | 12 |
| SR-tree | 20 | 12 |

大エントリ数の 40% とした．また，R*-tree，SS-tree で強制的に再投入するエントリの割合についても，これらの文献に合わせて 30%とした．表 2，表 3 に，構築したインデックスの木の高さを示す．

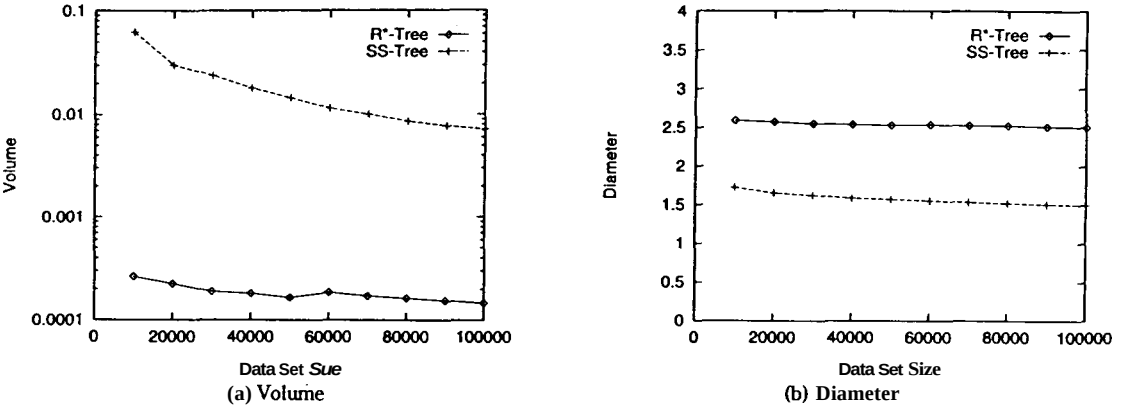
なお，K-D-B-tree の分割アルゴリズムについては．

表 2 木の高さ（一様分布データの場合）
Table 2 Tree heights (uniform distribution data).

| Index | Data Set Size (× 1000) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |
| K-D-B-tree | 4 | 4 | 4 | 4 | 4 | 4 | 5 | 5 | 5 | 5 |
| R*-tree | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| VAMSplit R-tree | 4 | 4 | 4 | 4 | 5 | 5 | 5 | 5 | 5 | 5 |
| SS-tree | 3 | 3 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| SR-tree | 4 | 4 | 4 | 4 | 5 | 5 | 5 | 5 | 5 | 5 |

表 3 木の高さ（実データの場合）

| Index | Data Set Size (× 1000) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 |
| K-D-B-tree | 3 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| R*-tree | 3 | 3 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| VAMSplit R-tree | 3 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| SS-tree | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| SR-tree | 3 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |




(a) Volume　(b) Diameter

図5　リーフでの包囲長方形ならびに包囲球の体積および最大径の平均値（一様分布データ）
Fig. 5　The average volume and average diameter of the leaf-level regions of SS-trees and R'-trees (uniform distribution data).

文献[6]で分割軸を周期的に選択する方法が示されているが，文献[12]で述べられているとおり，この方法には強制分割が起きやすくなるという問題がある．そこで，この実験ではK-D-B-treeの拡張である$R^+$-tree[13]のアルゴリズムを採用し，すべての軸の中から分割面を選択した．

実験結果を図3，図4に示す．いずれの図についても，(a)にCPU時間を(b)にディスクアクセス回数（ブロック読込み回数）を示している．横軸はデータ数であり，図3では，10,000～100,000の範囲で，図4では，2,000～20,000の範囲で変化させている．これらの測定結果は，文献[5],[8]の結果に添うものであり，R*-treeに対するSS-treeの優位性が確認された．また，SS-treeとVAMSplit R-treeの比較は，動的に構築されるインデックスと静的に構築されるインデックスの比較になるので必ずしも適当ではないが，文献[8]の結果どおり，VAMSplit R-treeの性能がSS-treeを上回っている．

### 3.2　包囲長方形による表現の問題点

3.1の実験結果で特徴的なのは，K-D-B-treeとR*-treeの検索時間がデータ数にほぼ比例していることである．これに対して，SS-treeでは，それよりも少ないCPU時間ならびにディスクアクセス回数で最近接検索を実現している．これは，R*-treeが包囲長方形を用いていることが原因であり，次のように説明できる．

包囲長方形の1辺の長さと対角線の長さの隔たりは，次元が高くなるにつれて大きくなる．例えば，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．そのため包囲長方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起きる．この問題を明確にするために，3.1の一様分布データを用いた実験で構築したR*-treeとSS-treeのリーフについて，包囲長方形ならびに包囲球の体積と最大径を測定した．つまり，R*-treeについては，包囲長方形の体積と対角線の長さを，SS-treeについては，包囲球の体積と直径を測定した．測定結果を図5に示す．図5(a)は，包囲長方形ならびに包囲球の体積の平均値を示している．図5(b)は，R*-treeについては包囲長方形の対角線の長さの平均値を，SS-treeについては包囲球の直径の平均値を示している．

これらの結果から，体積についてはR*-treeの方が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．最近接検索では，距離の近い点が同一のリーフに格納されている方が検索時間が短くなるため，領域の最大径が小さいSS-treeの方がR*-treeよりも良い結果をもたらすのである．

以上のことから，包囲長方形を用いた場合，次元が高くなるにつれて1辺の長さと対角線の長さの差が大きくなり，同一領域に属する点が必ずしも距離的に近いとは限らなくなるという問題があることがわかる．

### 3.3 包囲球による表現の問題点

SS-tree では，包囲球を用いることで最近接検索の高速化を実現している．しかし，図 5(a) が示すとおり，体積の点では R*-tree の包囲長方形の方が小さくなっており，およそ 1/50 になっている．体積が大きいと領域間の重なりが大きくなる可能性が高くなるため，最近接検索や範囲検索の性能を下げる原因となる．従って，図 5(a) の結果は，包囲球がすべての点で包囲長方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ことを示している．

そこで，本研究では，この問題点をより明確にするために，3.1 の一様分布データを用いた実験で構築した SS-tree のリーフについて，包囲長方形と包囲球の両方の体積を求め，それらを比較した．結果を図 6 に示す．図 5(a) と同様，横軸はデータ数であり，縦軸は包囲長方形ならびに包囲球の体積の平均値である．比較のために，R*-tree の包囲長方形についての結果も示している．この結果から，SS-tree のリーフにおいて，包囲長方形の体積の方が包囲球の体積よりもはるかに小さくなっていることがわかる．データ数が 100,000 のとき，包囲長方形の体積は包囲球のおよそ 1/900 であり，R*-tree の包囲長方形と比べても，およそ 1/18 になっている．つまり，SS-tree のリーフを包囲長方形によって表すと，R*-tree のおよそ 1/18 の体積になり，同じリーフを包囲球によって表すと，R*-tree のおよそ 50 倍の体積になってしまうのである．

このことから，包囲球はすべての点で包囲長方形に勝っているわけではなく，体積の点では，包囲長方形の方が有利であることがわかる．

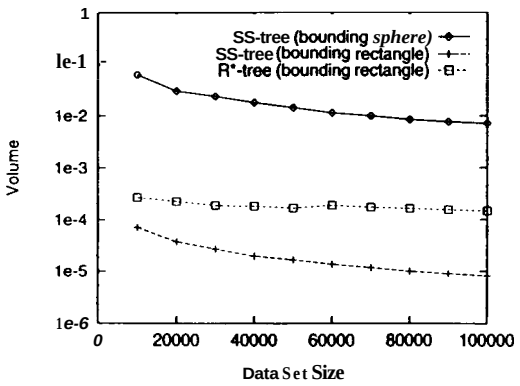



図 6 SS-tree のリーフでの包囲長方形ならびに包囲球の体積の平均値（一様分布データ）
Fig. 6 The average volume of the leaf-level regions of SS-trees (uniform distribution data).

### 3.4 評価結果のまとめ

以上の結果をまとめると次のようになる．

- 包囲長方形は次元が高くなるにつれて，1 辺の長さと対角線の長さの差が大きくなるため，包囲長方形によって空間を分割した場合，距離の近い点だけを集めることが難しくなる．
- 包囲球は，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．

このように，包囲長方形と包囲球は一長一短であり，相補的な関係にあると考えられる．そこで本研究では，これらを組み合わせれば距離的に近い空間を小さな体積で分割できるのではないかと考え，次章に述べる SR-tree を考案した．

## 4. SR-Tree の提案

### 4.1 インデックスの構造

SR-tree の構造は，R*-tree や SS-tree と同じく R-tree に基づくものである．相違点は，R-tree と R*-tree が包囲長方形を，SS-tree が包囲球を用いるのに対して，SR-tree では包囲長方形と包囲球を組み合わせて用いる（図 7）．従って，SR-tree では，図 8 のように包囲長方形と包囲球の共通部分によって，領域が区分される．

まず，リーフの構造を示す．

$$L : (E_1, \ldots, E_n) \qquad (m_L \leqq n \leqq M_L)$$

$$E_i : (\boldsymbol{p}, data)$$

リーフ $L$ は，$n$ 個 $(m_L \leqq n \leqq M_L)$ 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 リーフ当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

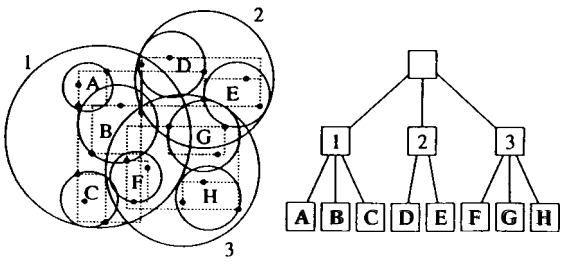
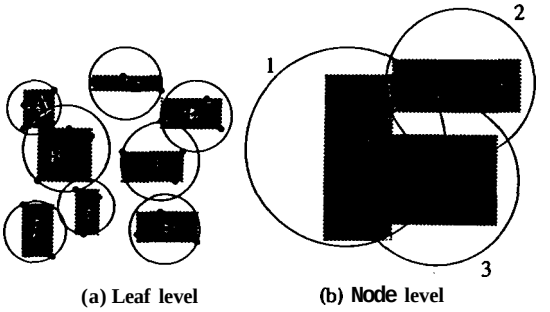



図 7 SR-tree の構造
Fig. 7 The SK-tree structure.




(a) Leaf level　　(b) Node level

図8　包囲長方形と包囲球の共通部分による領域の区分
Fig.8　Regions specified by the intersection of their bounding spheres and rectangles

座標 $\boldsymbol{p}$ とそれに付与するデータ *data* とを格納する．このリーフの構造は，SS-tree と同じものであり，R-tree と比べるとキーが長方形から点に変わっている点が異なっている．

次に，ノードの構造を示す．

$$N : (C_1, \ldots, C_n) \qquad (m_N \leqq n \leqq M_N)$$

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

ノード $N$ は，$n$ 個 $(m_N \leqq n \leqq M_N)$ のエントリ $C_1, \ldots, C_n$ を格納したものである．一つのエントリが一つの子供に相当し，$m_N$，$M_N$ は，1ノード当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，その包囲球 $\boldsymbol{S}$，包囲長方形 $\boldsymbol{R}$，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 と比べると包囲長方形 $\boldsymbol{R}$ が加わった構造になっており，R-tree と比べると包囲球 $\boldsymbol{S}$ とデータ数 $w$ が加わった構造になっている．

### 4.2　データの挿入ならびに削除

データの挿入ならびに削除の手順は SS-tree に準じる．SS-tree の手順は，SS-tree が提案されている文献[5]ならびに SS-tree の原型になっている R-tree，R*-tree の文献[4],[7]を参照することにより明らかなので，ここでは概略だけを述べる．SS-tree では，部分木に格納されている点の平均位置（重心）に基づいて点が挿入される．どのリーフに点を挿入するかは，ルートから順に，重心までの距離が最小である部分木を選択することによって決定する．点の挿入によってリーフがあふれる場合には，重心からの距離が遠い一定割合のエントリを再投入し，既に再投入が行われたリーフがあふれた場合にのみリーフを分割する．リーフの分割は，分割軸の決定と分割位置の決定の2段階に分かれる．まず，個々の次元について座標値の分散を求め，最大の分散をもつ次元を分割軸とする．次に，個々の分割位置について，分割後の二つの領域の分散を求め，それらの和が最小となる位置を分割位置とする．リーフの分割によって上位のノードがあふれる場合には，リーフと同じ方法で再投入ならびに分割が行われる．一方，削除の手順は，R-tree，R*-tree，SS-tree いずれも共通しており，アンダフローが起きない場合には，そのままエントリを削除し，アンダフローが起きる場合には，そのブロックを削除した上で，格納されていたエントリをすべて再投入する．

SR-tree の挿入ならびに削除の手順は SS-tree に準じたものであり，領域の形状の更新方法のみが異なっている．相違点は，領域の包囲長方形ならびに包囲球の更新方法に関するものであり，次の2点である．

まず，SS-tree では包囲球のみをノードのエントリに格納しているが，SR-tree では包囲長方形も格納しているので，包囲球と同時に包囲長方形の更新も行う．このとき，包囲長方形の更新方法は R-tree ならびに R*-tree と同じである．

次に，包囲球については，SS-tree では子供の包囲球をもとに親の包囲球を決定するが，SR-tree では，子供の包囲長方形と包囲球の両方の情報を用いて親の包囲球を決定する．SR-tree で親の包囲球を決定する方法を以下に示す．半径 $r$ を決める際に，包囲球に対する最大距離 $d_s$ と包囲長方形に対する最大距離 $d_r$ のうち，小さい方を半径としている点が SS-tree と異なっている．

（1）　包囲球の中心：$\boldsymbol{x} = (x_1, \ldots, x_i, \ldots, x_D)$

$$x_i = \frac{\sum_{k=1}^{n} C_k.x_i \times C_k.w}{\sum_{k=1}^{n} C_k.w} \qquad (1 \leqq i \leqq D)$$

ここで，$k$ $(1 \leqq k \leqq n)$ はノードのエントリに対する添字，$i$ $(1 \leqq i \leqq D)$ は座標軸に対する添字であり，$C_k.x_i$ は子供の包囲球の中心点の座標，$C_k.w$ は子供を頂点とする部分木が格納している点の総数である．

（2）　包囲球の半径：$r$

$$r = \min(d_s, d_r)$$

$$d_s = \max_{1 \leqq k \leqq n} (\|\boldsymbol{x} - C_k.\boldsymbol{x}\| + C_k.r)$$

$$d_r = \max_{1 \leq k \leq n} (MAXDIST(\boldsymbol{x}, C_k.\boldsymbol{R}))$$

$$MAXDIST(\boldsymbol{p}, \boldsymbol{R}) \equiv \max_{\boldsymbol{q} \in \boldsymbol{R}}(\|\boldsymbol{p} - \boldsymbol{q}\|).$$

ここで，$k$ $(1 \leq k \leq n)$ はノードのエントリに対する添字であり，$C_k.\boldsymbol{x}$，$C_k.r$ は子供の包囲球の中心点と半径，$C_k.\boldsymbol{R}$ は子供の包囲長方形である．また，$MAXDIST(\boldsymbol{p}, \boldsymbol{R})$ は，点 $\boldsymbol{p}$ から長方形 $\boldsymbol{R}$ までの最大距離のことであり，$\boldsymbol{p}$ から最も遠い頂点を見つけることにより，容易に求められる．

### 4.3 データの最近接検索

データの最近接検索については，R*-tree や SS-tree と同様に，文献[11]のアルゴリズムを用いる．アルゴリズムの詳細は，文献[11]で説明されているので，ここでは概略だけを述べる．このアルゴリズムは，与えられた検索点に最も近い点を，指定された数だけ見つけるためのものであり，検索点から領域までの最小距離を基準に深さ優先で探索する．すなわち，検索点からの最小距離でノードのエントリをソートし，最小距離が短いものから順に探索するのである．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，より近い点がないかどうか調べる．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．そして，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．

SR-tree は，R*-tree や SS-tree と同様に，このアルゴリズムを用いて最近接検索を行うが，検索点から領域までの最小距離を計算する方法が R*-tree や SS-tree とは異なる．SR-tree では，包囲球と包囲長方形の共通部分によって領域が区分されるので，検索点から領域までの最小距離 $d$ は，次式のように，包囲球に対する最小距離 $d_s$ と，包囲長方形に対する最小距離 $d_r$ のうち，より大きい方になる．

- 検索点 $\boldsymbol{p}$ からエントリ $C_k$ の領域までの最小距離：$d$

$$d = \max(d_s, d_r)$$

$$d_s = \max(0,\ \|\boldsymbol{p} - C_k.\boldsymbol{x}\| - C_k.r)$$

$$d_r = MINDIST(\boldsymbol{p}, C_k.\boldsymbol{R})$$

$$MINDIST(\boldsymbol{p}, \boldsymbol{R}) \equiv \min_{\boldsymbol{q} \in \boldsymbol{R}}(\|\boldsymbol{p} - \boldsymbol{q}\|).$$

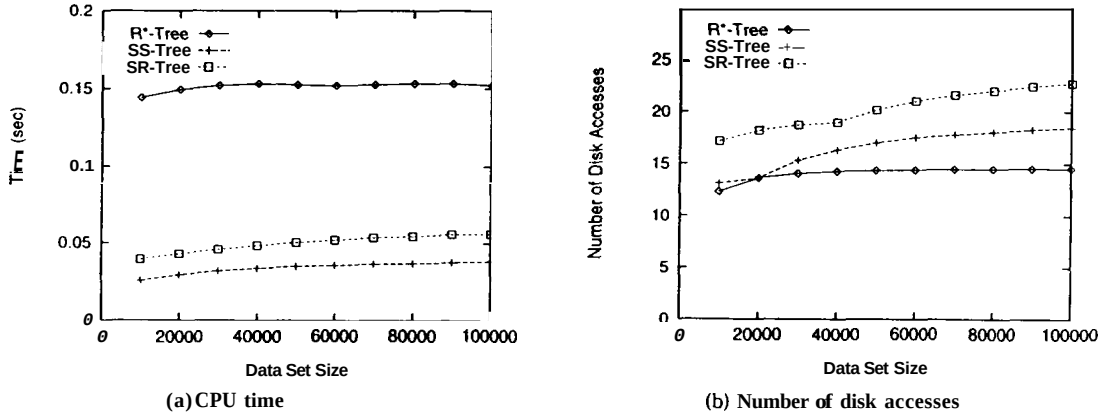
ここで，$C_k.\boldsymbol{x}$，$C_k.r$ はエントリ $C_k$ の包囲球の中心点と半径，$C_k.\boldsymbol{R}$ はエントリ $C_k$ の包囲長方形である．また，$MINDIST(\boldsymbol{p}, \boldsymbol{R})$ は，点 $\boldsymbol{p}$ から長方形 $\boldsymbol{R}$ までの最小距離のことであり，容易に求められる[11]．

## 5. SR-Tree の評価

### 5.1 評価実験

3.1 で行った評価実験と同じ実験を SR-tree に対して行い，SR-tree の性能を測定した．この実験における最大エントリ数ならびに木の高さは，表 1，表 2，表 3 に示すとおりである．

まず，木の構築に要するコストについて評価する．図 9 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（ブロック読込み回数とブロッ

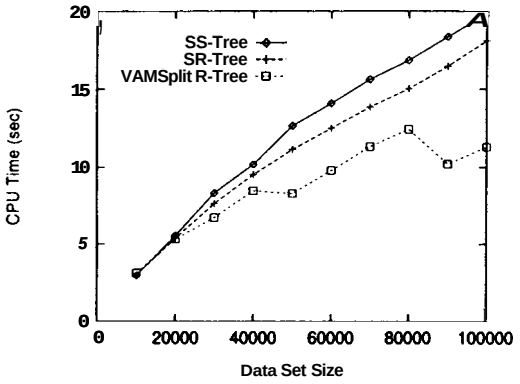



(a) CPU time　　(b) Number of disk accesses

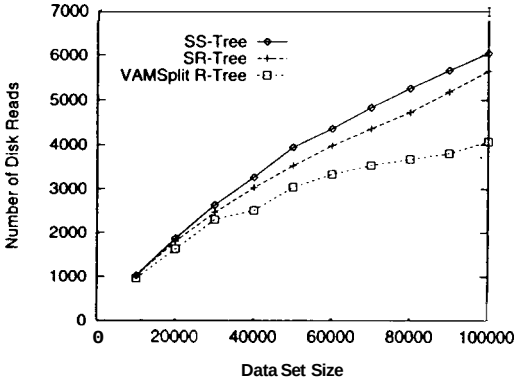
1 9　データの挿入に必要なコストの平均値（一様分布データの場合）
Fig.9 The average *cost* of inserting a new entry (uniform distribution data).

ク書込み回数の合計）の1回当りの平均値を示している．これらの図から，SS-treeならびにSR-treeは，R*-treeよりも短いCPU時間で木を構築できることがわかる．これは，SS-treeとSR-treeが用いている重心に基づくアルゴリズムの方が，R*-treeのアルゴリズムよりも必要な演算量が少ないためである[5]．また，SS-treeとSR-treeを比べると，SR-treeの方がSS-treeよりもより多くのCPU時間とディスクアクセス回数を必要とし，CPU時間でおよそ50%，ディスクアクセス回数でおよそ25%多いことがわかる．これは，SR-treeが包囲球のみならず包囲長方形も同時に格納しているからである．

次に，最近接検索の性能について評価する．実験結果を，図10，図11に示す．図10が一様分布データの場合，図11が実データの場合である．比較のために，SS-treeならびにVAMSplit R-treeの結果も示している．これらの結果から，どちらのデータに対しても，SR-treeがSS-treeを上回っていることがわかる．一様分布データの場合には，データ数が100,000のとき，CPU時間が91%，ディスクアクセス回数が93%になっており，実データの場合には，データ数が20,000のとき，CPU時間が67%，ディスクアクセス回数が68%になっている．また，静的に木が構築されるVAMSplit R-treeと比較すると，一様分布データの場合には，VAMSplit R-treeがSR-treeを上回っているが，実データの場合にはVAMSplit R-treeに匹敵する性能を示している．VAMSplit R-treeが与えられたデータセットに対して静的に構築されるのに対

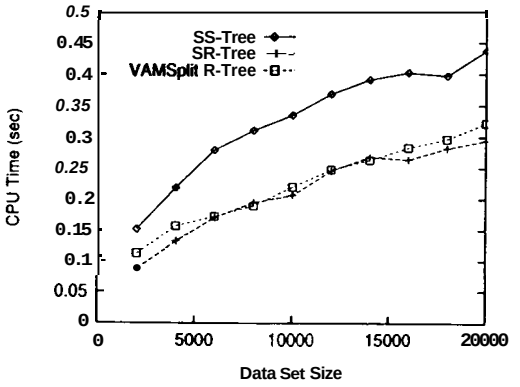



(a) CPU time

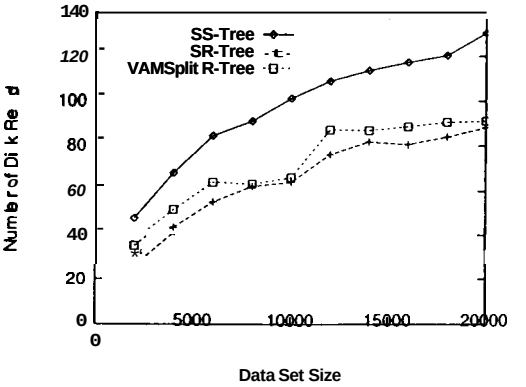



(b) Number of disk reads

図10 SR-treeに対する性能評価（一様分布データの場合）
Fig. 10 Performance of SR-trees (uniform distribution data).

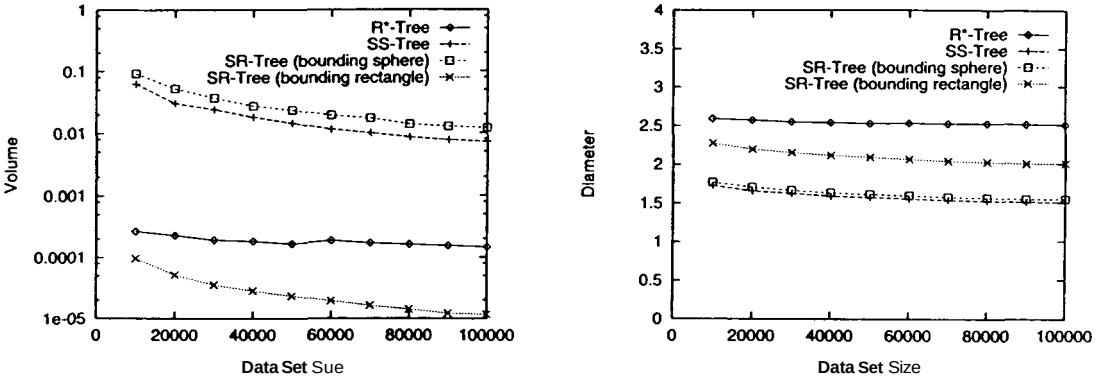



(a) CPU time




(b) Number of disk reads

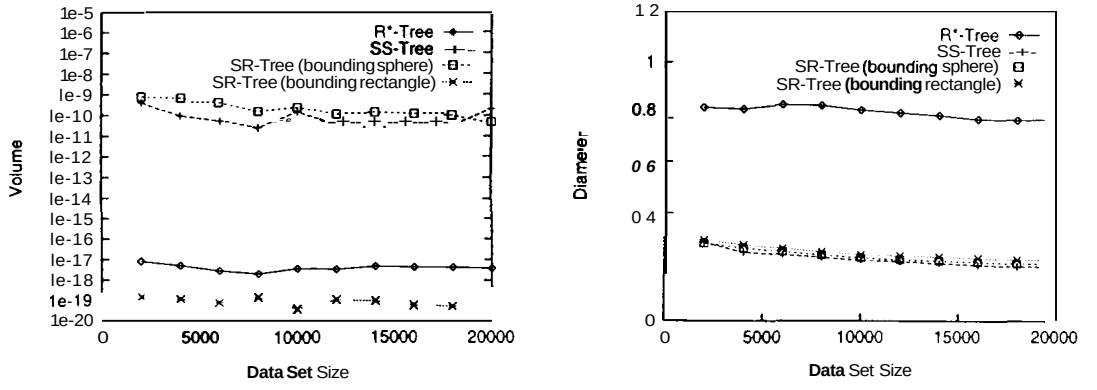
図11 SR-treeに対する性能評価（実データの場合）
Fig. 11 Performance of SR-trees (real data).




(a) Volume (b) Diameter

図 12 リーフでの包囲長方形ならびに包囲球の体積および最大径の平均値（一様分布データ）

Fig. 12 The average volume and average diameter of the leaf-level regions of SS-trees. R*-trees. and SR-trees (uniform distribution data).




(a) Volume (b) Diameter

図 13 リーフでの包囲長方形ならびに包囲球の体積および最大径の平均値（実データ）

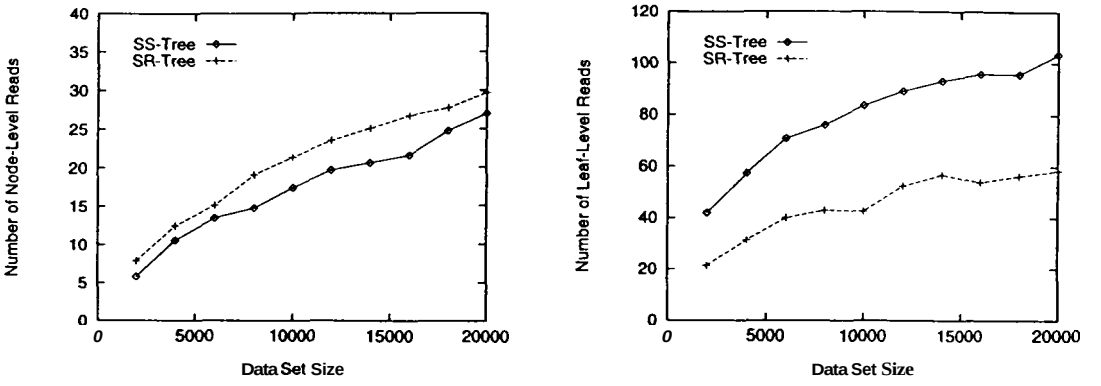
Fig. 13 The average volume and average diameter of the leaf-level regions of SS-trees. R*-trees, and SR-trees (real data).

して，SR-tree が動的に構築されることを考慮すれば，SR-tree が VAMSplit R-tree に匹敵する性能を示すことは注目に値する．

以上の結果から，SR-tree は，SS-tree よりも木の構築に要するコストは大きいが，最近接検索については SS-tree よりも高い性能をもつことがわかる．

### 5.2 SR-Tree の利点

SR-tree が SS-tree を上回る性能を示すのは，包囲球と包囲長方形を併用することにより，空間を SS-tree よりもより小さい領域に分割するからである．この特長を検証するために，評価実験で構築したインデックスについて，リーフレベルでの包囲長方形ならびに包囲球の体積と最大径を測定した．結果を図 12，図 13 に示す．図 12 は一様分布データの場合を，図 13 は実データの場合を示しており，いずれも，(a) に体積の平均値を，(b) に最大径の平均値を示している．それぞれの図には，R*-tree，SS-tree，SR-tree の結果を示してあり，R*-tree については包囲長方形の体積と対角線の長さを，SS-tree については包囲球の体積と直径を示している．SR-tree については，包囲長方形，包囲球それぞれについて，体積と対角線の長さ，体積と直径を示している．SR-tree の領域は，厳密にはこれらの共通部分になるのだが，共通部分は球面と平面が交わった複雑な形状になるため，体積や最大径の計算が




(a) Node level (b) Leaf level

**図 14** ノードおよびリーフレベルでのブロック読込み回数（実データの場合）
Fig. 14 The number of node and leaf block reads (real data).

難しい．そこで，ここでは包囲長方形，包囲球それぞれの体積と最大径を示すことにした．

まず，体積について見てみると，SR-tree の包囲長方形の体積が最も小さくなっていることがわかる．SR-tree の領域は包囲長方形と包囲球の共通部分であるから，領域の厳密な体積は，これと等しいか，これよりも更に小さい．これを SS-tree の体積と比較すると，一様分布データの場合，データ数が 100,000 のとき，およそ 1,000 分の 1 になっており，実データの場合には，データ数が 20,000 のとき，およそ 1,000,000,000 分の 1 になっている．

次に，最大径について見てみると，図 12，図 13 いずれの場合についても，SS-tree と SR-tree の包囲球の最大径がほぼ等しくなっていることがわかる．これは，SR-tree がデータの挿入に関して，SS-tree と同じく，重心に基づくアルゴリズムを用いているからであり，SR-tree が SS-tree と同様に距離的に近い点を一つの領域に集めていることがわかる．

以上のことから，包囲長方形と包囲球の併用によって，SR-tree が距離的に近い点を SS-tree よりもより体積の小さい領域で区分することに成功していることがわかる．これにより最近接検索の際に探索するリーフの数が少なくなり，5.1 の評価実験の結果が示すとおり，最近接検索の更なる高速化が実現されるのである．

### 5.3 SR-Tree の問題点

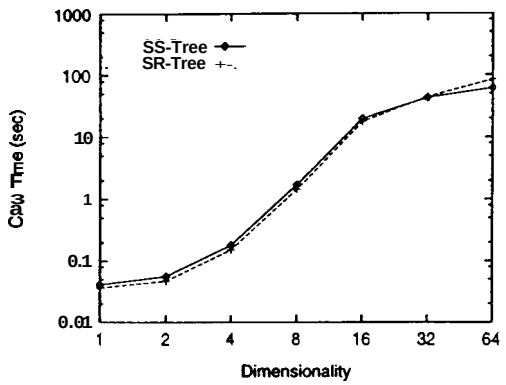
高次元点データを扱う場合，ノードのエントリの大きさが大きくなり，ノード当りの枝の数（fanout）が小さくなってしまうという問題点が指摘されている [5]．SR-tree の場合，包囲球と包囲長方形の両者をエントリに格納するため，ノードエントリのサイズが，包囲球のみを格納する SS-tree に対して約 3 倍，包囲長方形を格納する R*-tree に対しては約 1.5 倍になる．従って，ノード当りの枝の数が SS-tree に比べて 1/3 になり，木の高さが高くなったり，検索の際に読み込むノード数が増加することが考えられる．

そこで，この問題点を検証するべく，5.1 の実データを用いた実験（図 11）におけるノードレベルおよびリーフレベルでのブロック読込み回数を図 14 に示す．予想されるとおり，ノードレベルでのブロック読込み回数は，SR-tree の方が大きくなっている．しかし，それ以上にリーフレベルでの読込み回数が SS-tree よりも少なくなっているために，総合的には図 11 で示されるとおり，SR-tree の方がより少ないディスクアクセス回数で最近接検索を実現しており，枝の数が少なくなるという負の効果よりも，探索範囲を小さくできるという正の効果の方が上回っていることがわかる．

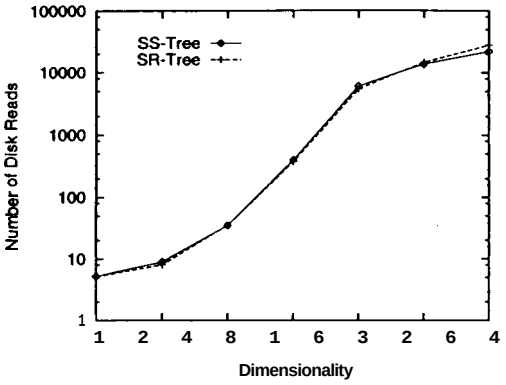
### 5.4 SR-Tree の特性

SR-tree の特性を評価するために，次元数を変えた場合とデータの分布形態を変えた場合について，最近接検索の性能を測定した．

まず，一様分布データの次元数を 1 から 64 まで変えた場合について，SR-tree ならびに SS-tree の性能を測定した．しかし，この測定では SR-tree と SS-tree の比較はできず，一様分布データが高次元における最近接検索の性能評価に適さないという結論しか得られなかった．というのは，一様分布データにおける点の分




(a) CPU time




(b) Number of disk reads

図 15 次元数を変えた場合の SR-tree と SS-tree の性能（一様分布データ）
Fig. 15 Performance of SR-trees and SS-trees with varying dimensionality (uniform distribution data).

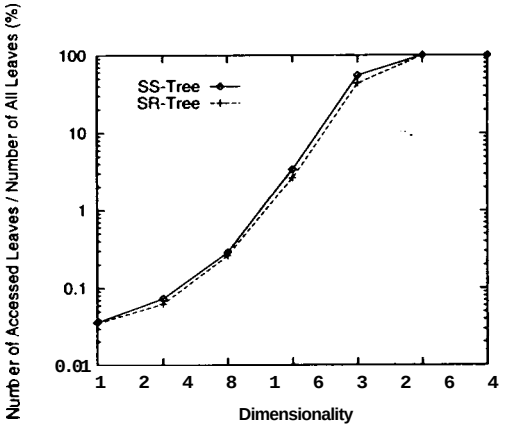



図 16 アクセスしたリーフの割合（一様分布データ）
Fig. 16 The ratio of the accessed leaves (uniform distribution data).

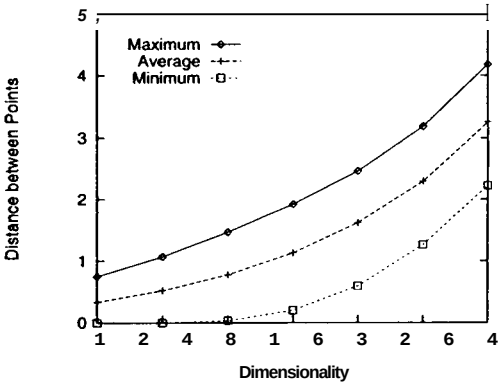
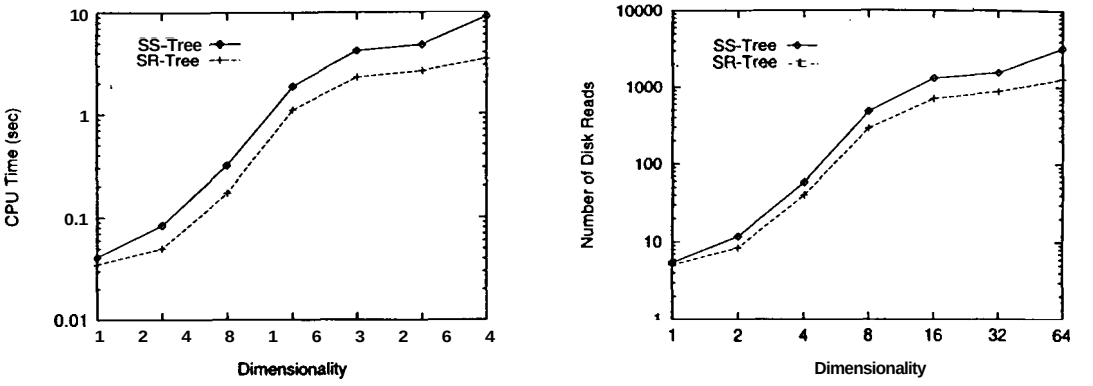



図 17 2 点間の距離の最小値，平均値，最大値（一様分布データ）
Fig. 17 The minimum/average/maximum of distances between points (uniform distribution data).

布が，本質的に最近接検索の高速化を難しくしているからである．まず，測定結果を図 15 に示す．図 15(a) に CPU 時間を，図 15(b) にディスクアクセス回数を示している．データ数はどの次元数についても 100,000 であり，その他の条件は 3.1 ならびに 5.1 の評価実験と同じである．次に，図 16 には，1 回の最近接検索でアクセスしたリーフの割合を，図 17 には，データ中のあらゆる 2 点間の距離について，最小値，平均値，最大値を示している．いずれの図についても横軸が次元数である．

これらの結果の中で注目すべきことは，図 16 が示すように，アクセスしたリーフの割合が，SR-tree, SS-tree どちらについても，32 次元および 64 次元で 100%に達していることである．つまり，どちらのインデックスも探索範囲を絞り込むことに失敗しているのである．しかし，この結果は，SR-tree や SS-tree の性能の限界というよりも，点の分布そのものに問題があると考えられる．というのも，図 17 が示すとおり，一様分布データにおける 2 点間の距離の最小値は次元が増加するにつれて急激に増え，16 次元で 0.6，32 次元で 1.3，64 次元で 2.2 となっている．これは，あらゆる点の組合せの中で最も距離の近い組合せについても，これだけの距離があるということである．しかも，次元数が増加するに従って，最小値と最大値の相対的な




(a) CPU time　　(b) Number of disk reads

図 18 次元数を変えた場合の SR-tree と SS-tree の性能（クラスタデータ）
Fig. 18 Performance of SR-trees and SS-trees with varying dimensionality (cluster data).

差が小さくなっており，最大値に対する最小値の比率は，16 次元で 24%，32 次元で 40%，64 次元で 53%である．これは，距離の分布の幅がどんどん狭くなっていることを意味しており，次元が高くなるにつれて，近い点と遠い点の相対的な差が，どんどん小さくなっているということである．この性質は，最近接検索の高速化を本質的に難しくするものであり，このような分布は，実際のアプリケーションでは意味をなさない．従って，一様分布データは，高次元における最近接検索の性能評価には適さないと考えられるのである．

そこで，本研究では，より実際的なデータとして，複数のクラスタから構成されるクラスタデータを設計し，性能評価に用いることにした．クラスタデータは，複数の球状のクラスタから構成されており，個々のクラスタの中には一定数の点が分布する．従って，点の総数は，クラスタ数にクラスタ当りの点の数を乗じたものになる．クラスタの位置と半径は，単位立方体の内部に無作為に選ぶことにし，個々の点の位置については，まずクラスタの球面上に一様に点を生成したのち，中心からの距離（半径）を一様乱数を用いて無作為に選ぶことによって決定することにした．

図 18 に，クラスタ数 100，クラスタ当りのデータ数 1,000 のクラスタデータについて，次元数を 1 から 64 まで変化させた場合の測定結果を示す．図 18(a) に CPU 時間を，図 18(b) にディスクアクセス回数を示している．データの総数は 100,000 であり，その他の条件は，3.1 ならびに 5.1 の評価実験と同じである．表 4 には，このときのノードならびにリーフの最大エントリ数を，表 5 には，木の高さを示す．これらの結果から，SR-tree は，低次元から高次元に至るまで SS-tree の性能を上回っており，SS-tree に比べておよそ 2 倍の性能が得られていることがわかる．

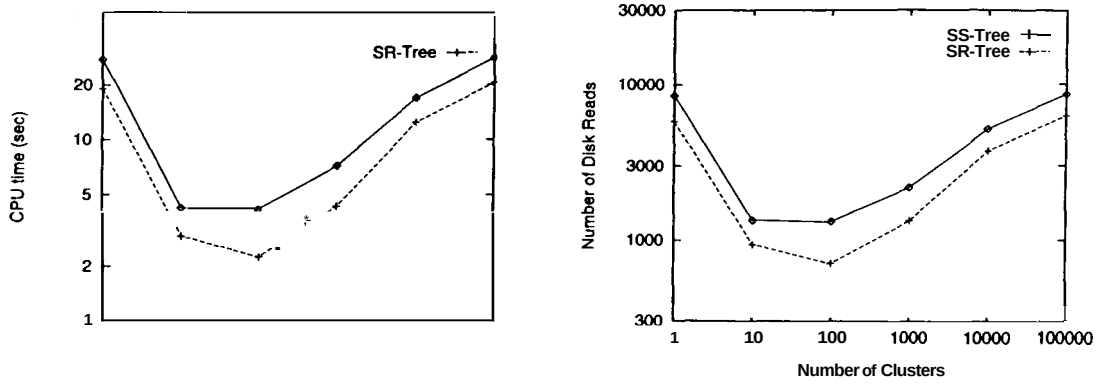
表 4 ノードならびにリーフにおける最大エントリ数
Table 4 Maximum number of entries in a node and a leaf.

| Index | | Dimensionality | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 4 | 8 | 16 | 32 | 64 |
| SS-tree | Node | 341 | 255 | 170 | 102 | 56 | 30 | 15 |
| | Leaf | 15 | 15 | 14 | 14 | 12 | 10 | 7 |
| SR-tree | Node | 204 | 127 | 73 | 39 | 20 | 10 | 5 |
| | Leaf | 15 | 15 | 14 | 14 | 12 | 10 | 7 |

表 5 木の高さ（クラスタデータの場合）
Table 5 Tree heights (cluster data).

| Index | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| SR-tree | | | | | 4 | | 6 | 9 |

次に，データの分布を変えた場合の SR-tree の特性を調べるために，クラスタデータのクラスタ数を 1 から 100,000 まで変化させる実験を行った．次元数は 16 次元であり，データ数は一定で 100,000 である．従って，クラスタ当りの点の数は，100,000 をクラスタ数で割った値である．クラスタデータの場合，クラスタ数を変化させることは，データの分布の一様性を変化させることに相当する．クラスタ数が 1 の場合には，点が単一の球の中に分布することになり，クラスタ数が 1 より大きく 100,000 より小さい場合には，単位立方




(b) Number of disk reads

□ 19 データの分布を変えた場合の SR-tree と SS-tree の性能（クラスタデータ）
Fig. 19 Performance of SR-trees and SS-trees with varying distribution (cluster data).

体の中に配置された複数のクラスタの中に点が分布することになる．そして，クラスタ数が 100,000 の場合には，単位立方体の中に点が一様に分布することになる．従って，クラスタ数が 1 から 100,000 の範囲にあるとき一様性が低く，クラスタ数が 1 または 100,000 に近いとき，一様性が高いことになる．

実験結果を図 19 に示す．図 19(a) に CPU 時間を，図 19(b) にディスクアクセス回数を示している．横軸はクラスタ数である．この結果で特徴的なのは，SR-tree と SS-tree の性能の比が，データの一様性が低いときほど大きくなっていることである．例えば，一様性の高い場合，すなわちクラスタ数が 1 や 100,000 の場合には，SR-tree の CPU 時間は，それぞれ SS-tree の 69%ならびに 73%になっている．これに対して，一様性の低い 100 や 1,000 の場合には，それぞれ 55%ならびに 60%になっているのである．この結果は，一様分布データよりも実データの方が性能の向上が見られた 5.1 の結果とも一致しており，SR-tree が一様性の低いデータに対してより効果的であることを示している．

## 6. むすび

本論文では，高次元点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，従来法の問題点を指摘すると共に，新たなインデックス構造として SR-tree の提案および性能評価を行った．

従来法に対する性能評価では，静的にインデックスを構築する VAMSplit R-tree の性能が最も良く，動的に構築するものの中では SS-tree の性能が最も良いことがわかった．また，次元が高くなった場合に，包囲長方形を用いる方法では 1 辺の長さと対角線の長さの差が大きくなるという問題があり，包囲球を用いる方法では包囲長方形よりも体積が大きくなるという問題があることを，評価実験の結果を用いて検証した．

本論文が提案する SR-tree は SS-tree の拡張であり，これらの問題を包囲長方形と包囲球を組み合わせることによって解決する．SR-tree の特徴は，領域を包囲長方形と包囲球の共通部分によって表現する点である．これにより，距離的に近い点を SS-tree よりも体積の小さい領域で区分することが可能であり，最近接検索の更なる高速化が実現される．一様分布データと実データの 2 種類の 16 次元データについて評価実験を行った結果，SR-tree の性能が R*-tree ならびに SS-tree を上回ることが確認された．最近接検索に要した CPU 時間ならびにディスクアクセス回数を SS-tree と比べると，一様分布データの場合に CPU 時間が 91%，ディスクアクセス回数が 93%となり，CPU 時間で 10%，ディスクアクセス回数で 8%の高速化が実現された．また，実データの場合には CPU 時間が 67%，ディスクアクセス回数が 68%となり，CPU 時間で 47%，ディスクアクセス回数で 48% の高速化が実現され，SR-tree の有効性が確かめられた．また，クラスタデータを用いた実験では，次元数を 1 次元から 64 次元まで変えた場合と，データの一様性を変えた場合について性能評価を行い，SR-tree が低次元から高次元に至るまで有効であること，ならびに，一様性が低い場合により大きな性能の向上が見られることが明らかになった．

このようにSR-treeは，SS-treeを上回る性能をもっており，画像の類似検索など，最近接検索の高速化を必要とするアプリケーションにおいて有用であると考えられる．

**謝辞** 性能評価に用いた実データは，カーネギーメロン大学のディジタルビデオライブラリプロジェクトであるInformediaプロジェクトのデータを使わせて頂きました．ここに感謝致します．また，本研究の機会を提供して下さったカーネギーメロン大学ロボティクス研究所所長金出武雄教授ならびに日本学術振興会に深く感謝致します．

## 文　献

[1] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang. B. Dom, M. Gorkani, J. Hafner, D. Lee. D. Petkovic. D. Steele. and P. Yanker. "Query by image and video content: The QBIC system,'' IEEE Computer. vol.28, no.9, pp.23–32, Sept. 1995.

[2] 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.

[3] 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. 1996.

[4] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An efficient and robust access method for points and rectangles,'' Proc. ACM SIGMOD, Atlantic City, USA, pp.322–331. May 1990.

[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. Robinson, "The K-D-B-tree: A search structure for large multidimensional dynamic indexes." Proc. ACM SICMOD, Ann Arbor, USA, pp.10–18, April 1981.

[7] A. Guttman, "R-trees: A dynamic index structure for spatial searching." Proc. ACM SIGMOD, Boston, USA, pp.47–57, June 1984.

[8] D.A. White and R. Jain. "Similarity indexing: Algorithms and performance,'' Proc. SPIE vol.2670, San Diego, USA, pp.62–73. Jan 1996.

[9] J.L. Bentley. "Multidimensional binary search trees used for associative searching." Comm. of the ACM, vol.18, no.9, pp.509–517, Sept. 1975.

[10] R. Sproull, "Refinements to nearest-neighbor searching in k-dimensional trees." Algorithmica, vol.6, no.4, pp.579–589, 1991.

[11] N. Roussopoulos, S. Kelley. and F. Vincent, "Nearest neighbor queries," Proc. ACM SIGMOD, San Jose, USA, pp.71–79, May 1995.

[12] D. Greene. "An implementation and performance analysis of spatial data access methods." Proc. of the 5th Int. Conf. on Data Engineering, Los Angeles. USA, pp.606–615. Feb. 1989.

[13] T. Sellis, N. Roussopoulos, and C. Faloutsos, "The $R^+$-tree: A dynamic index for multidimensional objects" Proc. of the 13th VLDB Conf.. Brighton, England, pp.507–518, Sept. 1987.

（平成8年11月19日受付）

**片山　紀生**　（正員）

平2東大・工・電気卒．平7同大大学院博士課程了．同年学術情報センター・研究開発部・助手，現在に至る．データベースシステムに関する研究に従事．工博．情報処理学会，IEEE，ACM各会員．

**佐藤　真一**　（正員）

昭62東大・工・電子卒．平4同大大学院情報工学博士課程了．同年学術情報センター助手．平7より平9まで，米国カーネギーメロン大客員研究員としてInformedia映像ディジタルライブラリーの研究に従事．工博．図面・画像理解，画像データベース構築，映像データベース構築などの研究に従事．情報処理学会会員．