情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2010-SLDM-144 No.49
Vol.2010-EMB-16 No.49
Vol.2010-MBL-53 No.49
Vol.2010-UBI-25 No.49
2010/3/27

ARツリーによる実空間コンテキストの効率的な検出

佐藤 龍[†1] 三次 仁[†2] 鈴木茂哉[†2]
中村 修[†2] 村井 純[†2]

実空間におけるコンテキスト（実空間コンテキスト）を用いて，その時々の状況に応じてコンテンツを配信する活動が期待されている．配信時の気候や配信場所など，実空間において観測されるセンシングデータに基いて実空間コンテキストを検出することは，実空間コンテキストを利用するアプリケーションに不可欠であるため，その効率化は重要な課題の一つである．

本稿では，実空間コンテキストの定義を構成する条件集合の論理関係を構造化するAR-treeにより，実空間コンテキストを検出するために必要な論理比較処理を効率化できることを示す．実空間コンテキストを定義する条件集合には，他の条件との包含関係や交差関係があると予想されることに着目し，このような想定環境におけるコンテキスト検出効率のシミュレーションを行った．その結果，AR-treeは既存研究の一つであるR-treeと比べて，コンテキスト検出を効率化できることを示した．

Efficient Real Space Context Detection using AR Tree

Ryu Sato,[†1] Jin Mitsugi,[†2] Shigeya Suzuki,[†2]
Osamu Nakamura[†2] and Jun Murai[†2]

There are some experiments to delivery content adjust for situation by detecting real-space context. It is challenging to improve system resource efficiency for detection of real-space context from sensing data (which is climate and delivery place etc.) observed in a real-space since all of context awareness applications is used.

This paper shows AR-tree which is a structure of a logical relation etween each set in condition group which composes the definition of the real-space context improve system resource efficiency with reducing the number of compararison condition times to detect a real-space context. Simulation evaluation run in an intended environment that there are an inclusion relation and a crossing relation to other conditions in the condition group to define the real-space context. As the result of evaluation, AR-tree improve system resource efficiency for detection of real-space context compared with R-tree which is one of the most famous.

1. はじめに

実空間コンテキストは，実空間において時々刻々と変化するその時々の状況を示す情報である．コンテキストアウェアアプリケーションなどの，実空間コンテキストを用いる研究が盛んに行われている．また，デジタルサイネージなどの情報表示システム[1)]においては，実空間コンテキストに応じて配信するコンテンツを選択することにより，情報の価値を高められると期待されている．このとき，実空間コンテキストとしては，コンテンツを閲覧しているユーザの状況や，配信時の気候，コンテンツを配信する場所などが想定される．実空間コンテキストは，実空間から観測されるセンシングデータから検出することが必要である．実空間コンテキストの検出は，このようなアプリケーションに不可欠であるため，その効率化は重要な課題の一つである[2)]．

実空間コンテキストは，実空間から観測されるセンシングデータが，予めサービスやユーザごとに定義された実空間コンテキストの条件集合に，合うかを判別することにより検出される．センシングデータは時系列における任意の時点での値を表し，条件は任意の値または範囲を表す．さらに，場所を指定する条件は二次元データであり，気候を温度で指定する条件は一次元データであるなど，次元の異なるデータから構成される．このように，実空間コンテキストの検出とは，複数の値が，複数の中のいずれの条件に合うかを判別し，合った条件から構成される定義を探索する操作と等しい．従って，実空間コンテキストの検出における効率化は，多次元データにおける点の検索を効率化することととらえることができる．

多次元データにおける点の検索を効率化する手法に，空間マッピング手法と空間インデクシング手法がある．これらの手法は，多次元データにおいて任意の点を検索する際に効率化できない場合があり，通常の検索よりも非効率になる場合がある．空間マッピング手法では，マッピング先の空間で連続していないデータを検索対象とする場合に効率化できない．一方，空間インデクシング手法では，検索データにつけられたインデクスが複数のデータにまたがる場合に効率化できない．

このような問題に対し，実空間コンテキストの定義を構成する条件集合の論理関係を構造

†1 慶應義塾大学大学院 政策・メディア研究科
Keio University, Graduate School of Media and Governance.
†2 慶應義塾大学大学院 環境情報学部
Keio University, Faculty of Environment and Information Studies

Vol.2010-SLDM-144 No.49
Vol.2010-EMB-16 No.49
Vol.2010-MBL-53 No.49
Vol.2010-UBI-25 No.49
2010/3/27

化することにより，実空間コンテキストを検出するために必要な論理比較処理を効率化する AR-tree を提案する．AR-tree は多次元のデータにおける点の検索において，検索対象のデータ数が増加する操作がなく，検索対象のデータ範囲を確実に絞ることができるため，先に述べた問題を解決することができる．これを評価するため，特徴を持ついくつかの実空間コンテキストの定義を想定し，実空間コンテキストの検出にかかる効率をシミュレーション評価により示す．

本稿の構成を次に示す．まず第 2 節で実空間コンテキストの検出を多次元データにおける点の検索ととらえ，このときの問題点を述べる．次に第 3 節で第 2 節で示した問題を解決する新たな手法を提案する．第 4 節で提案手法の効率をシミュレーションにより評価する．最後に第 6 節でまとめと今後の課題を述べる．

2. 実空間コンテキストの検出における問題

実空間コンテキストの検出は多次元データにおける点の検索操作と考えることができる．多次元データにおける点の検索を効率化する手法に，空間マッピング手法と空間インデクシング手法がある．空間マッピング手法とは，多次元のデータをより小さい次元のデータで表す手法である．空間マッピング手法には，Z-Ordering Curve[3] や Hilbert Curve[4),5)] などの空間充填曲線がある．空間充填曲線を用いることにより，ある範囲に点が含まれるかを，範囲を表すデータと点を表すデータの部分比較により判別できるため，多次元データにおける点の検索を効率化できる．一方，空間インデクシング手法には，R-tree[6] や R^+-tree, R^*-tree[7]，Quad Tree[8] などがある．空間インデクシング手法を用いることにより，検索対象のデータに対して用意されるインデクスにより，データ検索対象の探索範囲を絞ることができるため，多次元データにおける点の検索を効率化できる．

しかし，これらの手法は多次元データにおいて任意の点を検索する際に効率化できない場合があり，通常の検索よりも非効率になる可能性がある．空間マッピング手法では，マッピング先の空間で連続していないデータを検索対象とする場合に効率化できない（図 1）．これは，マッピングするために正規化処理において，正規形でないデータは分割され，正規形データの組み合わせで表さなくてはならないためである，この場合，検索対象のデータ数がマッピング前よりも増加するため，データ検索を効率化できない．空間充填曲線を用いた空間マッピングアルゴリズムは数多く存在するが，いずれもマッピングにおいて正規化が必要であるため，同じ問題がある．一方，空間インデクシング手法では，同じ検索対象のデータに対してインデクスが用意された場合に効率化できない（図 2）．これは，インデクスが示す全てのデータを検索しなくてはならないためである．この場合，インデクスにより検索対象のデータ範囲を絞ることができないため，データ検索を効率化できない．R+-tree や R∗-tree に代表される R-tree から派生した関連研究は，データ構造は R-tree と同様であるため，いずれも本質的に同じ問題がある．Quad Tree は空間充填曲線によりインデクシングするため，空間充填曲線と同様の問題がある．

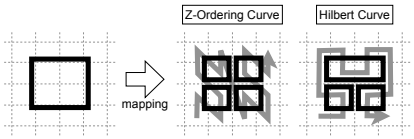


図 1 空間充填曲線において分割が必要な例
Fig. 1 Examples of Division for Mapping to Space Filling Curve

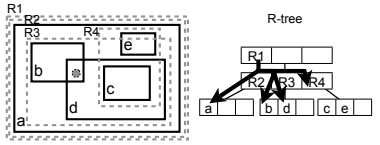


図 2 R-tree において複数範囲の探索が必要な例
Fig. 2 Example of Multipath Search in R-tree

3. AR-tree の提案

実空間コンテキストは多次元データに対する点の検索操作と考えることができるが，既存研究では効率化できない場合があり，通常の検索よりも非効率になる可能性があることを第 2 で述べた．この問題に対し，検索データ同士の論理関係を構造化することにより，多次元データでの点の検索における論理比較処理を効率化する AR-tree を提案する．本節では AR-tree による多次元データ検索の効率化と，AR-tree を構成するアルゴリズムを述べる．

3.1 条件間の論理関係による検索の効率化

複数の条件のうち，必要または十分などの論理関係に着目し，条件間の論理関係を構造化することにより，いくつかの条件を比較する必要を省いて効率化する手法を考える．

3.1.1 条件の論理関係と効率化

いま，任意の条件 A, B を考える．条件 A, B の関係は表 1 に示すように，次の 4 つの場合に分類できる．1）$A \cap B = \emptyset$．2）$A \cap B \neq \emptyset$．3）$A \cap \overline{B} = \emptyset$．4）$\overline{A} \cap B = \emptyset$．ここで，次に示す通り，条件が交差する場合を除く 3 つの場合において，一方の条件の真偽より他方の条件の真偽が分かる場合がある．1）$A \cap B = \emptyset, A \Rightarrow \overline{B}$．2）$A \cap \overline{B} = \emptyset, A \Rightarrow B$．3）$\overline{A} \cap B = \emptyset, B \Rightarrow A$．すなわち，任意の条件集合において，条件同士の関係により，条件比較回数を条件集合の数より少なくしつつ，全体の条件に合うかどうかを調べることができる．従って，実空間コンテキストの定義を構成する条件集合の論理関係を構造化し，比較する条件の順序を最適化することで，実空間コンテキストの検出を効率化できると期待できる．

Vol.2010-SLDM-144 No.49
Vol.2010-EMB-16 No.49
Vol.2010-MBL-53 No.49
Vol.2010-UBI-25 No.49
2010/3/27

$A \cap B = \emptyset$ の場合

A	B
-	-
1	0
0	1
0	0

$A \Rightarrow \overline{B}$ が成立

$A \cap B \neq \emptyset$ の場合

A	B
1	1
1	0
0	1
0	0

$A \cap \overline{B} = \emptyset$ の場合

A	B
1	1
-	-
0	1
0	0

$A \Rightarrow B$ が成立

$\overline{A} \cap B = \emptyset$ の場合

A	B
1	1
1	0
-	-
0	0

$B \Rightarrow A$ が成立

表 1 条件 A,B の関係ごとの真理値表（真を 1，偽を 0 と表記）
Table 1 Truth Table for Case of Relationship between A and B (1 shows true, 0 shows false)

a	b	c	d	e	
0	0	0	0	0	(行 1)
1	0	0	0	0	(行 2)
1	0	0	0	1	(行 3)
1	1	0	1	0	(行 4)
1	1	0	0	0	(行 5)
1	0	1	1	0	(行 6)
1	0	0	1	0	(行 7)

a	b	c	e	d	
1	0	1	0	1	(行 6)
1	1	0	0	1	(行 4)
1	0	0	0	1	(行 7)
1	0	0	1	0	(行 3)
1	1	0	0	0	(行 5)
1	0	0	0	0	(行 2)
0	0	0	0	0	(行 1)

表 2 関係式 1 における条件の真理値表（左側：最適化前，右側：最適化後）
表 3 Truth Table of eq.1 (left side: before construction，right side: after construction

条件同士の論理関係を構造化するには，各条件同士の比較を行えばよい．ここで，論理関係の構造化とは，存在する条件の組み合わせより，効率的に比較操作が可能な条件の比較順序を発見することである．いま，条件集合 C において，C の要素 $a, b, c, d, e (\in C)$ を考える．ここで，a, b, c, d, e は図 3 に表されるように，次に示す式 1 の関係を持つとする．

$$\begin{cases} b \vee c \vee d \vee e \Rightarrow a \\ \neg(\exists b \vee d) \\ c \Rightarrow d \\ e \Rightarrow \overline{b} \vee \overline{d} \end{cases} \quad (1)$$

すると，a, b, c, d, e の起こりうる組み合わせは 7 通り存在する（表 3）．これより，a, b, c, d, e の効率のよい比較順序の発見は表 3（左側）の順序を最適化する問題と考えられる．最適化した結果は表 3（右側）となる．結果，d, e, c, b, a の順で比較すると，平均 4.2 回の比較で組み合わせを調べることができ，効率化できることが分かる．

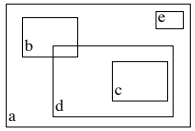


図 3 関係式 1 における条件の論理関係
Fig. 3 Relation between Condition in eq.1

3.1.2 条件比較順序の最適化

条件の論理関係を求め，求めた関係を最適化することにより，条件集合から効率的に比較できる条件の順序を導く．条件関係が最適化されている状態とは，条件集合の真理値表において，一番右の列の上から下に真理値を見たときに，1 が連続した後 0 が一続きになっている．また，一つ左の列を 1 が連続している行と 0 が連続している行とを分けて，列を上から下に真理値表を見たときに，同様に 1 が連続した後に 0 が一続きになっている．このように真理値表が最適化されているとき，条件集合から任意の条件に一致する条件の組み合わせを比較するには，真理値表の一番右の列の条件から比較を始め，比較結果（0 か 1）に応じて調べる対象の行を絞り，1 行に絞り込むまで繰り返せば良い．比較の結果残った行が，任意の条件と一致する条件の組み合わせである．

次に，条件集合の真理値表を最適化する方法について述べる．最適化された真理値表における一番右の列から見た条件の並び順を，最適化された条件順序と呼ぶ．まず，条件集合がただ 1 つの条件 a を含む場合，a の真理値表は，1 と 0 が一つだけある表であり，この状態は最適化されている．ここで，最適化された状態の条件集合の真理値表を基準として，任意の条件 a_m を追加する場合を考える．いま，条件集合を $C(C \ni a_n\{n|0..N\})$ とし，C において存在する値の組み合わせを P とする．真理値表において，C は一番上の行の条件集合を示し，P はそれ以外の行を示す．P は条件集合における全集合であるため，P の各要素と a_m との関係を調べることで，a_m を追加した後の P を導出できる．P の各要素との比較は，最適化された条件順序の順に比較することと等しい．従って，最適化された条件順序に従って a_n と a_m を比較し，重なった条件が存在する場合における a_n の前に a_m を挿入すれば，最適化された条件を保つことができる．

3.2 条件の論理関係に基づく検索ツリー (AR-tree) の提案

条件の論理関係に基づく検索を実現する検索ツリー **Area Relation Tree (AR-tree)** を示す．AR-tree は第 3.1 節で述べた，最適化された真理値表と同等の構造を持つ．AR-tree によるインデクシング例を図 4 に示す．

AR-tree は条件の包含関係による二分木構造を持つインデクスツリーである．AR-tree

Vol.2010-SLDM-144 No.49
Vol.2010-EMB-16 No.49
Vol.2010-MBL-53 No.49
Vol.2010-UBI-25 No.49
2010/3/27

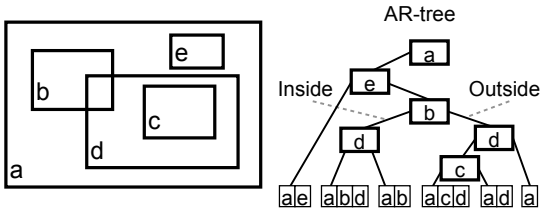


図 4 AR-tree によるインデクシング例
Fig. 4 Example of Indexing by AR-tree

を用いることで任意の値に合う条件が検索できる．AR-tree はインデクスノード（Index Node）とデータノード（Data Node）より構成される．ツリーにおける最上位のインデクスノードをルート（Root）と呼ぶ．Index Node は条件データに加え，条件データ間の包含関係に基づいて，他の Index Node との接続関係を持つ．接続関係は接続相手の Index Node が示す条件を含む場合と含まない場合の二通りである．ここで，接続相手の条件を含む場合は Index Node の左側に，接続相手の条件を含まない場合は Index Node の右側に接続するものとする．任意の条件同士の関係は，含むか，重なるか，含まないかのいずれかに決定できるため，重なる場合における条件を，含む場合と含まない場合の部分条件に分ける事で，AR-tree は任意の条件集合に対して構成可能である．

例えば，a から e の条件がある場合，図 4 の構造を持つ AR-tree を構成できる．a に着目すると，e は a に含まれ，a に含まれない条件は存在しないため a の左側の子として e がつながる．また，b に着目すると，b と d は共有する部分条件を持つため，b は d と共有する部分条件による親子関係と，d と共有しない部分条件による親子関係の二通りの関係を持つ．

3.2.1 AR-tree の特徴

AR-tree は条件集合をインプットとし，Data Node はインプットデータの冪集合を持つ．Data Node が持つデータは，与えられた条件集合による組み合わせの一つである．ツリーにおいては，Root から任意の Data Node までのパスが経由する Index Node の条件の，組み合わせを示す．この特徴は，AR-tree が与えられた条件集合から，条件間の最適な論理関係を構造化できていることを示す．

AR-tree は階層構造を持つ．階層構造とは，任意の節を最上位とするサブツリーが，Root を最上位とするツリーと同じ特徴を再帰的に持つことである．この特徴により，任意の Index Node を最上位とするサブツリーを，ツリー全体と同等に扱うことができる．また，AR-tree により任意の値に合う条件を検索した結果は，Data Node が持つデータにより示される．これはツリーの Root から点の検索を行った結果，比較したインデクスの軌跡がただ一つの線になることを意味する．従って，任意の値に合う条件を検索する過程において，探索の分岐がおこらない．

3.2.2 値に合う条件集合の検索アルゴリズム

値（KEY_POINT）に合う条件集合を検索するアルゴリズムを図 5 に示すとともに以下に述べる． 条件の検索はツリーの Root を始点に，Index Node が示す条件に P が含まれ

SearchValue(AR-tree KEY_POINT)

```
1  if AR-tree is empty then
2    Value KEY_VALUE is not included any areas.
3  else
4    Let TOP_NODE be root of AR-tree.
5    do while TOP_NODE of tree is index
6      if area of TOP_NODE contains KEY_VALUE then
7        Continue loop on subtree whose top is inside child of TOP_NODE.
8      else if area of TOP_NODE and KEY_VALUE are disjoint then
9        Continue loop on subtree whose top is outside child of TOP_NODE.
10     end if
11   end while
12   Value is included areas which have data node.
13 end if
```

図 5 値に合う条件集合の検索アルゴリズム
Fig. 5 Algorithm for Search Matching Conditions with Value

るかを比較し，Data Node を発見するまで繰り返す．Data Node の発見は，比較対象の条件と値の包含関係より二つある子のいずれかを選択し，選択した子を最上位とするサブツリーに対して点の検索を再帰的に試みることで行う．子の選択方法は比較される条件の包含関係より次の通り決定する．ここで，比較対象のうち，Index Node が示す条件を A とし，Index Node が持つ二つの子のうち，A に含まれる条件を持つ子 Index Node を C_I，他方を C_O とする．

- A が P を含む場合：C_I を選択する
- A が P を含まない場合：C_O を選択する

3.2.3 条件の追加アルゴリズム

新規条件（NEW_AREA）を AR-tree に追加するアルゴリズムを図 6 に示すとともに以下

Vol.2010-SLDM-144 No.49
Vol.2010-EMB-16 No.49
Vol.2010-MBL-53 No.49
Vol.2010-UBI-25 No.49
2010/3/27

に述べる．ここで，条件の追加アルゴリズムをツリーまたは任意のサブツリーと NEW_AREA を用いて，AppendArea((SUB_)TREE, NEW_AREA) とおく．

AppendArea((SUB_)TREE NEW_AREA)

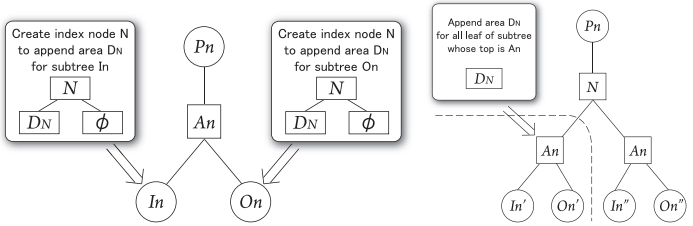
```
1  Let TOP_NODE be root of TREE.
2  begin routine AppendAreaToSubTree(TOP_NODE, KEY_AREA)
3    do while TOP_NODE is index
4      if area of TOP_NODE intersects KEY_AREA then
5        Do AppendArea(subtree which top is inside child of TOP_NODE, KEY_AREA).
6        Do AppendArea(subtree which top is outside child of TOP_NODE, KEY_AREA).
7        Break out of routine.
8      else if area of TOP_NODE and KEY_AREA are disjoint then
9        Continue routine with let TOP_NODE be outside child of TOP_NODE.
10     else if area of TOP_NODE includes KEY_AREA then
11       Create index NEW_INDEX as root with area of NEW_AREA.
12       Place TOP_NODE as inside child of NEW_INDEX.
13       Split subtree O_n into O_n_1 which is inside of area P_n and N
14         and O_n_2 which is outside of area P_n and N.
15       Append NEW_AREA to all data nodes for I_n and O_n.
16     else if area of TOP_NODE contains KEY_AREA then
17       Continue routine with let TOP_NODE be outside child of TOP_NODE.
18     end if
19   end while
20 Create index INDEX as root with area of NEW_AREA.
21 Place NEW_AREA as child of INDEX.
```

図 6 条件の追加アルゴリズム
Fig. 6 Algorithm for Append New Condition

条件の追加は，ツリーの各データに関連する条件群と新規条件（NewArea）により，新たな条件群を作ると同時に対応する検索ツリーを作る操作である．ここで，NewArea を N，追加する対象の AR-tree における任意の Index Node を A_n，Index Node が持つ二つの子のうち，A_n に含まれる条件を持つ子 Index Node を C_I，他方を C_O，A_n の親から Root までの Index Node が持つ条件を結合した条件を P_n．A_{2n} を最上位とするサブツリーを I_n．A_{2n+1} を最上位とするサブツリーを O_n とする．

いま，N を追加する問題は，P_n に対し，$P_n \cap N$ と $P_n \cap A_n$ の 2 つの条件による新たな条件作成の問題と考えられる．従って，追加条件に対するインデクスを構築する方法は，2 つの条件 $P_n \cap N$ と $P_n \cap A_n$ 関係より考えればよい（4 通り）．4 つのそれぞれの場合について N の作成方法を以下に示す．

場合 1 $P_n \cap A_n$ と $P_n \cap N$ が重なる場合
- **パターン 1** N を A_n の子 Index Node として追加する方法
  - (1) I_n と O_n に対して N を追加
- **パターン 2** N を A_n の親 Index Node として追加する方法
  - (1) N を A_n の場所に挿入
  - (2) A_n を N の C_I とし，A'_n を複製し，N の C_O とする
  - (3) I_n を $I'_n(I'_n \subseteq P_n \cap N)$ と $I''_n(I''_n \subseteq \overline{P_n \cap N})$ に分け，I''_n を A'_n の C_I とする
  - (4) O_n を $O'_n(O'_n \subseteq P_n \cap N)$ と $O''_n(O''_n \subseteq \overline{P_n \cap N})$ に分け，O''_n を A'_n の C_O とする
  - (5) I'_n と O'_n の全ての Data Node に D_N を追加



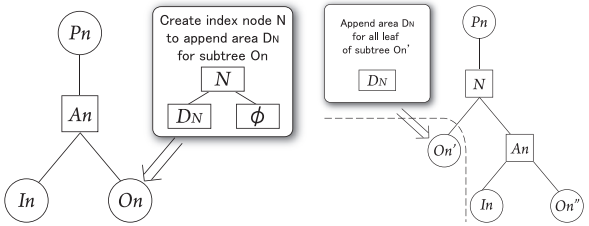
(a) pattern 1　　(b) pattern 2

図 7 $P_n \cap A_n$ と $P_n \cap N$ が重なる場合（場合 1）の条件 N の追加方法
Fig. 7 Append Case 1 ($P_n \cap A_n$ intersects $P_n \cap N$)

場合 2 $P_n \cap A_n$ と $P_n \cap N$ が重ならない場合
- **パターン 1** N を A_n の子 Index Node として追加する方法
  - (1) O_n に対して N を追加
- **パターン 2** N を A_n の親 Index Node として追加する方法
  - (1) N を A_n の場所に挿入
  - (2) N の C_I を I_n とし，C_O を O_n とする
  - (3) O_n を $O'_n(O'_n \subseteq P_n \cap N)$ と $O''_n(O''_n \subseteq \overline{P_n \cap N})$ に分け，O''_n を A'_n の C_O とする
  - (4) O'_n の全ての Data Node に D_N を追加

場合 3 $P_n \cap A_n$ が $P_n \cap N$ を含む場合
- **パターン 1** N を A_n の親 Index Node として追加する方法
  - (1) N を A_n の場所に挿入
  - (2) A_n を N の C_O とする
  - (3) I_n を $I'_n(I'_n \subseteq P_n \cap N)$ と $I''_n(I''_n \subseteq \overline{P_n \cap N})$ に分け，I'_n を N の C_I とし，I''_n を A_n の C_I とする
  - (4) I'_n の全 Data Node に D_N を追加

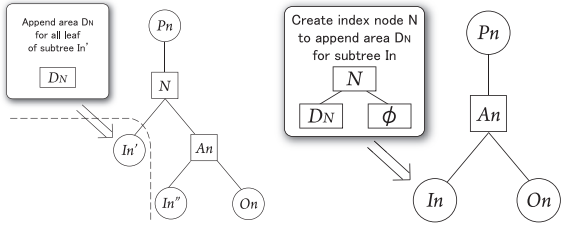
Vol.2010-SLDM-144 No.49
Vol.2010-EMB-16 No.49
Vol.2010-MBL-53 No.49
Vol.2010-UBI-25 No.49
2010/3/27



(a) pattern 1　　(b) pattern 2

図 8　$P_n \cap A_n$ と $P_n \cap N$ が重ならない場合（場合 2）の条件 N の追加方法
Fig. 8　Append Case 2 ($P_n \cap A_n$ does not intersect $P_n \cap N$)

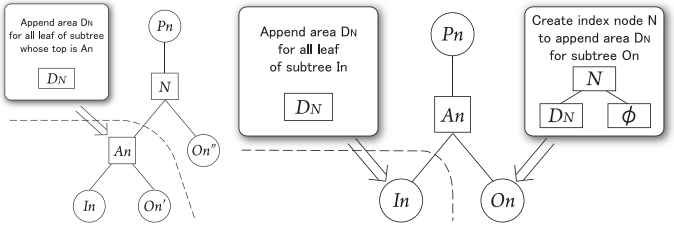
パターン 2　N を A_n の子 Index Node として追加する方法
（1）　I_n に対して N を追加



(a) pattern 1　　(b) pattern 2

図 9　$P_n \cap A_n$ が $P_n \cap N$ を含む場合（場合 3）の条件 N の追加方法
Fig. 9　Append Case 3 ($P_n \cap A_n$ contains $P_n \cap N$)

場合 4　$P_n \cap A_n$ が $P_n \cap N$ に含まれる場合
パターン 1　N を A_n の親 Index Node として追加する方法
（1）　N を A_n の場所に挿入
（2）　A_n を N の C_I とする
（3）　O_n を $O'_n (O'_n \subseteq P_n \cap N)$ と $O''_n (O''_n \subseteq \overline{P_n \cap N})$ に分け，O'_n を A_n の C_O とし，O''_n を N の C_O とする
（4）　I_n と O'_n の全ての Data Node に D_N を追加
パターン 2　N を A_n の子 Index Node として追加する方法
（1）　I_n の全ての Data Node に D_N を追加
（2）　O_n に対して N を追加



(a) pattern 1　　(b) pattern 2

図 10　$P_n \cap A_n$ が $P_n \cap N$ に含まれる場合（場合 4）の条件 N の追加方法
Fig. 10　Append Case 4 ($P_n \cap A_n$ is inside of $P_n \cap N$)

4. 実空間コンテキストの検出効率のシミュレーション評価

第 3 節で提案した AR-tree による，実空間コンテキストの検出効率をシミュレーション評価する．AR-tree や関連研究における検索効率は，対象とする条件集合における条件間の関係により変わる．条件を空間データにおける範囲として考えたとき，条件間の関係は，空間における範囲のサイズと分布の特徴により特徴づけられる．ここで，条件を空間データにおける範囲とし，分布に特徴を持つ空間データセットを用いて効率の変化を測定した．検索効率は検索に必要となった領域の比較回数により評価する．

シミュレーション評価は，Java VM が動作する Mac OS X の PC（CPU は 2GHz が 2 コア，Memory は 2GB 667MHz）において，YourKit Java Profiler[9] を用いて測定した．図 11 に測定に用いた 5 種類の空間データセットを示し，各データセットの分布の特徴を次に述べる．

CONCENT　領域の数：100 個，領域のサイズ：100 種類
階層的に領域同士が包含されるように分布しているデータである．複数サイズの領域の中心が固定されるように作成した．

MIX　領域の数：100 個，領域のサイズ：1 と 5 の二種類 (1：90 個，5：10 個)
二種類のサイズの領域が二次元の空間に一様に分布しているデータである．二次元の空間に一様に分布しているデータを，サイズが 1 である領域を 90 個，サイズが 5 である領域を 10 個作成し，それらをまとめて一つのデータセットにした．

Vol.2010-SLDM-144 No.49
Vol.2010-EMB-16 No.49
Vol.2010-MBL-53 No.49
Vol.2010-UBI-25 No.49
2010/3/27

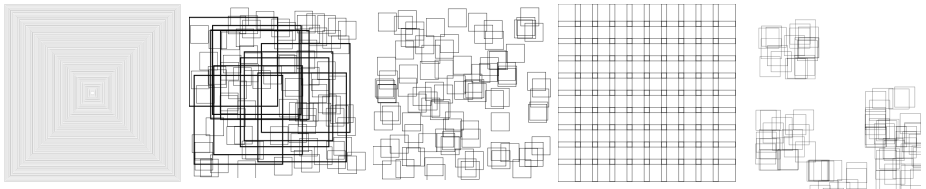


図 11 シミュレーション評価に用いたインプットデータセット
Fig. 11 Input Data Set for Simulation Evaluation

UNIFORM 領域の数：100 個，領域のサイズ：1
固定サイズの領域が二次元の空間に一様に分布しているデータである．

PARCEL 領域の数：100 個，領域のサイズ：1.3，領域の距離：1
固定サイズの領域が一定間隔を空けて交差して分布しているデータである．格子状に分布する領域を，隣の領域と交差するよう拡大して作成した．

CLUSTER 領域の数：100 個，領域のサイズ：1，局所点：5 箇所
5 箇所の局所点を二次元空間に一様に分布させ，それぞれの局所点を中心として，20 個の領域が同心円内に一様に分布するようにして作成した．

AR-tree による検索効率を測定した結果を表 4 にまとめる．表 4 より，全てのデータセットにおいてノード数 100 より小さい比較回数を示し検索が効率化できていることが分かる．CONCENT，MIX においては，他のデータセットに比べ，検索が比較的高く効率化されていることが平均比較回数（Avg）より分かる．MIX においては，検索クエリによらず効率化されていることが標準偏差（StdDev）の値 12.54 から読み取れる．対して，CONCENT においては，検索クエリにより，良く効率化されている場合と，あまり効率化されていない場合があることが標準偏差の値 25.21 から読み取れる．一方，空間データの追加操作は，空間データの数 100 を超える場合が存在した．これは，空間データ同士が交差している場合に，ツリーにおける空間データの追加を複数のサブツリーを対象として再帰的に行ったためである．以上の結果より，AR-tree により検索効率が全体的に向上させることができていると言える．特に CONCENT, MIX において比較的高い効率を実現した．

次に，既存手法の中で最も普及している手法の一つである R-tree と比較することで，AR-tree によるアルゴリズムの検索効率の優位性を示す．検索効率は，検索における比較対象を矩形と点とすることで比較コストを同一にし，検索結果を求めるまでに要した比較回数を比較測定することで行った．検索データセットは図 11 に示した 5 種を用いた．なお，R-tree における効率を左右するパラメータ（各ノードが保持する領域データの最小数と最大数）は，各データセットごとに最も効率の良い値を求めて使用した．

	検索				追加			
	比較回数			時間 (ms)	比較回数			時間 (ms)
	Avg	Max	StdDev	Avg	Avg	Max	StdDev	Avg
CONCENT	33.88	92	25.21	0.09	68.96	212	43.82	0.09
MIX	20.40	42	12.54	0.11	50.50	100	28.87	0.11
UNIFORM	62.56	100	34.72	-	38.39	111	22.20	-
PARCEL	43.90	99	29.62	0.01	56.35	106	29.81	0.01
CLUSTER	74.13	96	30.83	0.08	55.92	127	33.29	0.08

表 4 AR-tree の検索と追加に要した領域比較回数と時間
Table 4 The Number of Comparison And Time to Search And Append about AR-tree

計測結果を図 12 に示す．データセット CONCENT, MIX において，AR-tree は R-tree に比べ，検索に要する平均比較回数が少なかった．R-tree と同一条件で比較を行い，R-tree が最も効率が良くなるパラメータを用いたため，この結果より，AR-tree が検索効率を高めることができたと分かる．一方，データセット UNIFORM, PARCEL, CLUSTER は包含関係を持たないため，包含関係に基づく効率化が有効でなかった．

5. シミュレーション評価結果の考察

AR-tree は包含関係を用いて検索を効率化するため，CONCENT と MIX において，高い効率を示したのは，包含関係を持つためであると考えられる．また，MIX と CONCENT における検索に要した比較回数の標準偏差が異なるのは，ツリーにおける Data Node までの深さにばらつきがあるためと考えられる．実験時にデータを挿入した後のツリーの形状を図 13 に示す．図 13 より，MIX におけるツリーの深さが一様なのに対して，CONCENT では大きなばらつきが見られた．

また，検索に用いたデータに，比較的多くの比較が必要なデータが含まれていたためとも考えられる．実験時にデータを挿入した後のツリーの形状は図 13 に示す通りであり，図より，データセット UNIFORM, PARCEL, CLUSTER において，いずれの領域にも含まれない場合に比較回数がおよそ 80 から 100 まで必要になることが分かる．これが検索効率が高めることができなかった要因と考えられる．従って，ツリーの各 Deta Node の深さの差をできる限りなくして，深さをバランス化させることが今後の課題である．

Vol.2010-SLDM-144 No.49
Vol.2010-EMB-16 No.49
Vol.2010-MBL-53 No.49
Vol.2010-UBI-25 No.49
2010/3/27

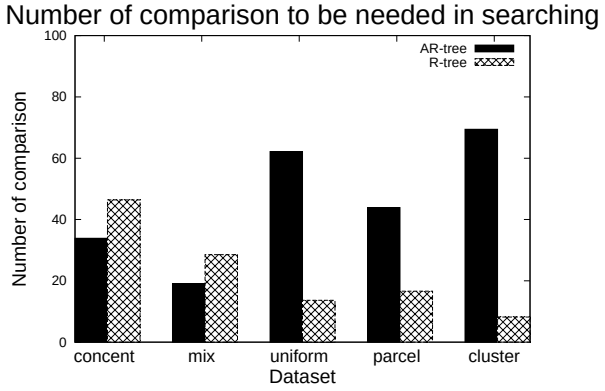


図 12 AR-tree と R-tree において検索に要した領域比較回数
Fig. 12 The Number of Comparison to Search with AR-tree and R-tree

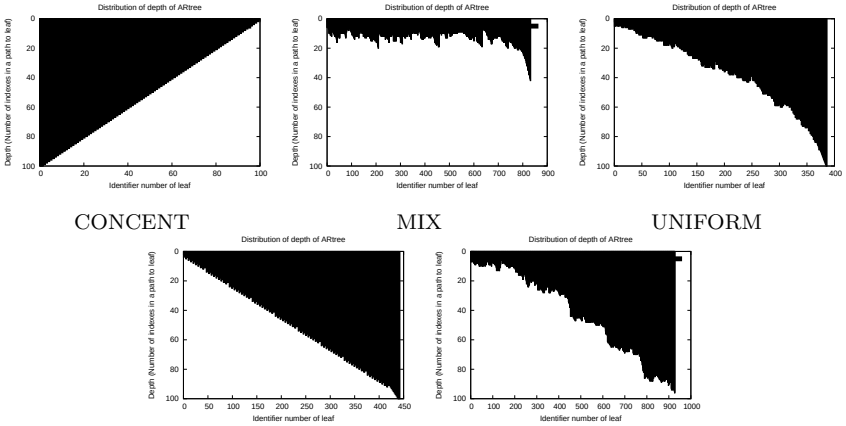


図 13 データ挿入後の AR-tree の形状（縦軸は Root から Data Node までの下向きを正としたノード数，横軸は Data Node の左から数えた番号）
Fig. 13 Shape of AR-Tree after Data Insertion (Y-axis shows depth, X-axis shows number of data node start from left)

6. まとめと今後の課題

本稿では，実空間コンテキストの検出を効率的に行う新たなデータ構造である AR-tree を示した．実空間コンテキストの検索を多次元データにおける点の検索と考え，特徴を持つ 5 種の特徴を持つ領域データセットを用いたシミュレーション実験の結果，全体的に検索効率を高めることができることが分かった．また，検索対象のデータセットにおいて，包含関係が存在する場合には高い効率化を実現できた．このことは，最も普及している関連研究の一つである R-tree と同一データセットを用いて比較した結果，R-tree が最も検索効率を高められるパラメータ調整をした場合よりも効率化できたことで示された．

今後は AR-tree における各 Data Node の深さをバランス化させることが課題である．AR-tree では論理関係により Index Node の親子関係が構成されるため，回転操作が適用できない．そこで，仮想的な包含関係を作ることにより，Data Node の深さの差を減らす方法や，第 3.2.3 節で述べた挿入パターンを，深さの差を考慮して選択することにより，深さをバランス化することができると考えられる．

参 考 文 献

1) 佐藤龍，横石雄大， 三次仁，鈴木茂哉， 中村修， 村井純：移動デジタルサイネージにおけるコンテンツの非同期プリフェッチ，Vol.109, No.326 (2009).
2) ARIMURA, H.: Recent Development of Mining Algorithms for Data Streams, *Trans.Inst. Electron.Inform. Communi.Engnr. Jpn. D-I*, Vol.88, No.3 (20050301).
3) Orenstein, J.A. and Merrett, T.H.: A class of data structures for associative searching (1984).
4) Faloutsos, C. and Roseman, S.: Fractals for secondary key retrieval (1989).
5) Jagadish, H.V.: Linear clustering of objects with multiple attributes, *SIGMOD Rec.*, Vol.19, No.2 (1990).
6) Guttman, A.: R-trees: a dynamic index structure for spatial searching (1984).
7) Beckmann, Kriegel, H., Schneider, R. and Seeger, B.: The R*-tree: an efficient and robust access method for points and rectangles, *ACM SIGMOD Record* (1990).
8) Finkel, R. and Bentley, J.: Quad trees a data structure for retrieval on composite keys, *Acta informatica* (1974).
9) : Java Profiler - .NET Profiler - The profilers for Java and .NET professionals, *http://www.yourkit.com/*.