位置依存 Top-k 検索のための効率的なインデックス構造の提案

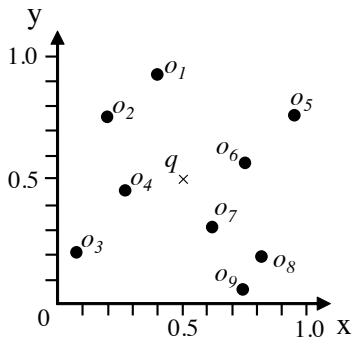
佐々木 勇和[1,a)]　Wang-Chien Lee[2,b)]　原 隆浩[1,c)]　西尾 章治郎[1,d)]

概要：近年，位置情報を含むデータを検索する機会が増加している．これに伴い，位置情報を考慮した Top-k 検索が注目されている．本稿では，特に，指定された位置とデータとの距離，およびその他の属性値に基づいて，データを検索する位置依存 Top-k 検索について着目する．位置依存 Top-k 検索のためのインデックス技術は，筆者らの知る限り，これまでに提案されておらず，既存のインデックス構造では効率的に検索できない．そこで本稿では，R 木とスカイラインを組み合わせた SKY-R 木を提案する．SKY-R 木では，位置情報と属性値の両方の観点から検索領域を効果的に削減することができる．さらに，SKY-R 木を拡張して，属性値間の類似度を考慮して構築する，S2KY-R 木も提案する．実験結果より，提案インデックス構造は，入出力コストおよび計算時間を削減できることを確認した．

1. はじめに

位置測量装置と移動端末の普及に伴い，位置情報を考慮した検索が多くのアプリケーションにとって必要不可欠となっている．例えば，位置依存ソーシャルサービスである，Yelp や Foursquare[*1] などが例としてあげられる．これらのサービスでは，位置や嗜好に合わせた検索をユーザに提供している．図 1 において，左のグラフは 9 個のデータを含む空間（地理的な距離）を示し，右の表はそれぞれのデータの 2 つの属性値（値段と人気）を示す．指定位置 q およびユーザの嗜好が与えられた時，データと指定位置の距離および嗜好を考慮した属性値をもとに，データを順位付けする．[*2] このとき，ユーザが指定する位置や，ユーザの嗜好（値段が重要など）により，最もよいデータは変化する．

近年，空間データベースにおける Top-k 検索は，頻繁に研究されている ([3], [4] など)．本稿では，文献 [12] で定義されている，位置と属性値に基づいてデータを順位付けする，位置依存 Top-k 検索に着目する．位置依存 Top-k 検索では，ユーザは，検索データ数 k，指定位置 q，および嗜好を指定する．つまり，位置依存 Top-k 検索は，“データと q の距離” と “ユーザの嗜好に合わせた属性値” に基づいて



	price	rating
o_1	0.1	1.0
o_2	0.8	0.1
o_3	0.3	0.8
o_4	0.3	0.4
o_5	0.9	0.3
o_6	0.2	0.7
o_7	0.3	0.3
o_8	0.6	0.2
o_9	0.6	0.6

図 1: データと指定位置を含む空間およびデータの属性値

データをスコア付けし，上位 k 個のスコアとなるデータを返信する．検索領域を削減するためには，データとの距離と属性値が重要となる．例えば，R 木 [5] により，指定位置に最も近いデータを検索し，そのデータに属性値に基づいてスコア付けすることが可能である．より効率的な方法として，Max aR 木 [8] は，位置と属性値の両方を同時に考慮しながら，検索領域を削減することが可能である．しかし，属性値の情報が不正確であるため，絞込みの精度はそれほど高くない．

本稿では，位置依存 Top-k 検索処理のための新しいインデックス構造を提案する．この提案インデックス構造は，R 木 [5] とスカイライン [1] を基にするため，*SKY-R* 木とよぶ．R 木は空間データベースにおいて，効率的にデータを検索できるインデックス構造として知られている．一方，スカイラインでは，位置を考慮しない Top-k 検索にお

[1] 大阪大学情報科学研究科マルチメディア工学専攻
[2] Pennsylvania State University
[a)] sasaki.yuya@ist.osaka-u.ac.jp
[b)] wlee@cse.psu.edu
[c)] hara@ist.osaka-u.ac.jp
[d)] nishio@ist.osaka-u.ac.jp
[*1] http://www.yelp.com/，および https://foursquare.com/
[*2] この例では，小さい属性値をよりよいものとする．

いて，スカイラインに属するデータ (スカイライン点) のみが最もスコアがよいデータとなるため，属性値の観点から検索領域を削減することが可能である．そのため，SKY-R 木は，位置と属性値の両方の観点から検索領域を削減することが可能である．SKY-R 木では，それぞれのノードは，そのノードを根とする部分木に属するデータの位置情報およびスカイライン集合の概要を保持している．クエリ処理アルゴリズムでは，位置情報およびスカイライン集合の概要より，それぞれデータの位置および属性値を推定し，検索すべきデータを選定する．さらに，SKY-R 木を拡張し，属性値間の類似度を考慮した構造，*S2KY-R* 木を提案する．SKY-R 木では，構築時に位置情報のみを考慮するが，S2KY-R 木では，属性値間の類似度と位置を考慮するため，葉ノードが似たようなデータを保持しやすくなり，効率的な位置依存 Top-k 検索処理を実現する．実データを用いた実験により，提案インデックス構造は，入出力コストおよび計算時間において，効率的に位置依存 Top-k 検索を処理できることを確認する．

以下では，まず，2. において，予備知識を説明する．その後，3. および 4. において，提案するインデックス構造，および拡張インデックス構造について，それぞれ説明する．その後，5. において，実験結果を示す．最後に，6. において，関連研究を述べ，7. で本稿をまとめる．

2. 予備知識

2.1 問題定義

データ $o \in O$ は，位置情報 *o.loc* と属性値 *o.att* を保持している．*o.loc* は，緯度と経度によって表される 2 次元平面とし，*o.att* は d 次元のベクトル $o.att_i \in [0,1](i = 1, \cdots, d)$ で表される．属性値は，小さいほどよいものとする．各データのスコアは，クエリ q に基づいて決定される．クエリ q は，クエリの指定位置 *q.loc* と重み付け係数 *q.w* ($q.w_i \geq 0$ $(i = 1, \cdots, d)$，かつ $\sum_{i=1}^{d} q.w_i = 1$) で構成される．クエリ q における，データ o のスコア $score(q, o)$ は，以下の式に従って計算される．

$$score(q,o) = \alpha \frac{D(q.loc, o.loc)}{maxD} + (1-\alpha)\frac{\sum_{i=1}^{d} q.w_i \cdot o.att_i}{maxA}. \quad (1)$$

この式において，$\alpha \in [0,1]$ は，距離と属性値のスコア付けにおける重要性を表すパラメータ，$D(q.loc, o.loc)$ は q と o のユークリッド空間上の距離，および $maxD$ と $maxA$ は正規化のための空間上の最大距離と属性値上の最大距離をそれぞれ示す．このスコアリング関数に基づいて，位置依存 Top-k 検索を以下のように定義する．

定義 1 (位置依存 Top-k 検索). クエリ q およびデータ要求数 k が与えられたとき，位置依存 Top-k 検索の検索結果 $TOP_k(q)$ は，$TOP_k(q) \subset O, |TOP_k(q)| = k$，かつ $\forall o_i, o_j : o_i \in TOP_k(q), o_j \in O - TOP_k(q)$ において，$score(q, o_i) \leq score(q, o_j)$ となる．

本稿での研究目的は，効率よく $TOP_k(q)$ を探すことである．

例: 図 1 において，クエリを q (*q.loc*={0.5, 0.5}, *q.w*={0.5, 0.5}) , $\alpha = 0.5$，および $k = 2$ とする．それぞれのデータのスコアを式 (1) で計算すると，1 位および 2 位のデータは，o_7 および o_4 となる．

2.2 スカイライン

位置依存 Top-k 検索では，データのスコアは位置と属性値によって決定する．属性値のみに着目した場合，スコアリング関数が単調増加であるため，1 位のデータはスカイラインに含まれることが知られている．以下に，スカイラインの定義，および Top-k 検索との関連を述べる．

定義 2 (スカイライン). あるデータ $o_i \in O$ が，データ $o_j \in O$ に対して，全ての次元の属性値において $o_i.att \geq o_j.att$，かつ少なくとも 1 つの次元の属性値において $o_i.att > o_j.att$ となる場合，o_i は o_j にドミナントされているという．スカイライン集合 $SKY \subseteq O$ は，O 内のいずれの点にもドミナントされていないデータ集合である．また，SKY 内のデータをスカイライン点とよぶ．

定理 1. スコアリング関数が単調増加の場合，*1* 位のデータは必ずスカイライン集合に含まれる．

証明: ある点 o_x と，点 o_x をドミナントする点 o_y を考える．つまり，全ての次元の属性値において $o_y.att_i \leq o_x.att_i$，かつ少なくとも 1 つの次元の属性値において $o_y.att_i < o_x.att_i$ が成り立つ．単調増加関数 f では，$f(o_x) > f(o_y)$ が常に成立する．そのため，他の点にドミナントされている点は，上位 1 位のデータに含まれることはない．つまり，スカイライン集合内のデータのみが上位 1 位のデータとなりえる．□

スカイライン点により，位置を考慮しない Top-k 検索における 1 位のデータを検索することができる．そのため，スカイラインは属性値の観点から検索領域を削減することが可能である．

例: 図 2 における右のグラフは，図 1 におけるデータの属性値を 2 次元平面にプロットしたものである．例えば，o_4 は o_7 にドミナントされているため，o_4 はスカイライン点に含まれない．この例では，スカイライン点は o_1，o_2，o_6，o_7，および o_8 となる．

2.3 Max aR 木

Max aR 木 [8] は，位置依存 Top-k 検索処理に用いることができる．Max aR 木では，各ノードは子孫ノードの位置情報と属性値情報の概要を保持している．まず，位置情報として，各ノードは自身を根とする部分木に含まれるデータを

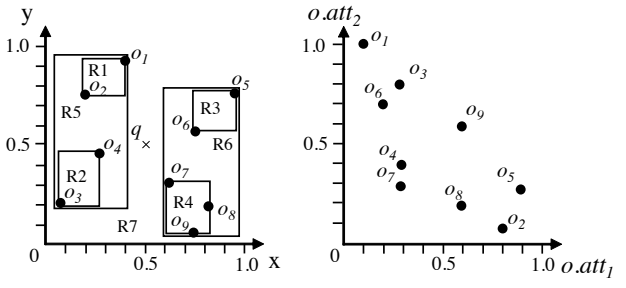


図 2: 最小矩形と属性値

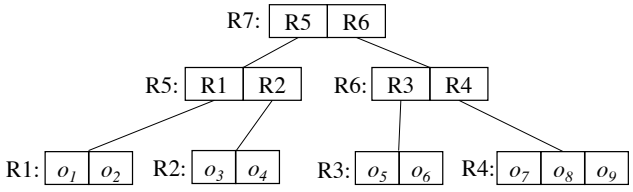


図 3: R 木におけるデータ構造

表 1: Max aR 木における各ノードが保持する属性値情報

Node	Best value
R7	(R5:0.1,0.1), (R6:0.2,0.2)
R6	(R3:0.2,0.3), (R4:0.6,0.2)
R5	(R1:0.1,0.1), (R2:0.3,0.4)
R4	(o_7:0.3,0.3), (o_8:0.6,0.2), (o_9:0.6,0.6)
R3	(o_5:0.9,0.3), (o_6:0.2,0.7)
R2	(o_3:0.3,0.8), (o_4:0.3,0.4)
R1	(o_1:0.1,1.0), (o_2:0.8,0.1)

全て包含する最小矩形 *Minimum Bounding Rectangle (MBR)* を保持している．加えて，各ノードは自身を根とする部分木に含まれるデータの属性値の各次元における最小値を保持している．Max aR 木は，位置と属性値によって検索領域を削減することができるが，各ノードは子孫データの各次元の最小値の属性値を保持しているため，検索時に子孫データの実際の属性値よりよいスコアを推定してしまう．そのため，属性値の観点からの絞込みの精度は，それほど高くない．

例: 図 3 は，図 1 における 9 個のデータにおける Max aR 木の構造，および図 2 は MBR と属性値を表す．また，表 1 は，中継ノードにおける最小属性値情報，および葉ノードにおけるデータの属性値情報を示す．この例では，R5 が保持している R1 の最小属性値は，(R1:0.1, 0.1) であるが，R1 が実際に保持しているデータの属性値は，(o_1:0.1, 1.0) および (o_2:0.8, 0.1) である．これらの値は R5 が保持しているものと大きく離れている．

3. R 木とスカイラインを組み合わせたインデックス構造

本節では，R 木とスカイラインを組み合わせたインデックス構造 SKY-R 木，および位置依存 Top-k 検索処理アルゴリズムについて説明する．

3.1 SKY-R 木

SKY-R 木は，基本的に R 木と同じ構造だが，各ノードはスカイライン点を保持している．

SKY-R 木は，属性値による検索領域の削減を目的とする．各ノードが自身の全ての子孫データの属性値情報を保持すれば，検索領域を大幅に削減することが可能である．しかし，R 木では，ディスクに対する入出力数（入出力のコスト）を抑制するため，1 つのノードに対して，1 回の入出力で全ての子ノードの情報を取得できるように子ノードの数が制限されている．つまり，ノードが保持する情報のサイズに応じて子ノードの数が変化するため，各ノードが保持する情報量が多すぎる場合，効率が低下する．そのため，どのような情報を保持するかが，効率的に位置依存 Top-k 検索を処理する上で重要となる．保持する情報における条件は，正確な結果を保証すること，さらに小さいサイズで検索領域を削減することである．そこで，スカイライン点はデータの結果を保証でき，かつ検索領域を削減できるため，SKY-R 木の各ノードは，スカイライン点を保持する．

位置依存 Top-k 検索処理では，スカイライン点は非常に有益であるが，スカイライン点の数を制御できないといった問題がある．もしスカイライン点の数が多くなると，ノードが保持する情報量が大きくなり，入出力のコストが大きくなってしまう．そこで，SKY-R 木では，概要スカイライン [13] により，スカイライン点の数を制限しつつ，結果を保証する．スカイライン点の数が最大スカイライン点の数 $|\overline{SKY}|$ を超えた場合，以下の処理によりスカイライン点を集約する．

検索領域削減の効率性が損なわれないように，概要スカイライン（アルゴリズム 1）では，まず全てのスカイライン点の中から $|\overline{SKY}|$ 個の代表スカイライン点を選択する．まず，エントロピーが最も小さいスカイライン点を代表スカイラインとする（1 行目）．その後，他の代表スカイライン点との距離が最大となるスカイライン点を代表スカイライン点とする（2 から 4 行目）．代表スカイライン点以外のスカイライン点を，そのスカイライン点との距離が最小となる代表スカイライン点に割り当てる（5 から 8 行目）．最後に，同じ代表スカイライン点に割り当てられた点を集約し（各次元で最小の属性値を選択），“仮想スカイライン点” を作成する（10 から 12 行目）．概要スカイラインにより，スカイライン点数を効果的に削減することができる．

SKY-R 木では，各葉ノードはエントリ *(o, o.loc, o.att)* を保持し，一方，各中継ノードはエントリ *(np, rectangle, sp)* を保持する．*np* は子ノードへのポインタ，*rectangle* は MBR，および *sp* はスカイライン点を表す．

例: 図 3 は，図 1 における SKY-R 木を表す．インデック

Algorithm 1 Abstract skyline

Input: SKY and $|\overline{SKY}|$
Output: $\overline{SKY}$
1: $p_1 \leftarrow \text{argmin}_{\forall p \in SKY}(\sum_{i=1}^{d} \ln(p.att_i + 1))$
2: **for** $\forall p_i : i = 2, \cdots, |\overline{SKY}|$ **do**
3: $p_i \leftarrow \max_{\forall p \in SKY - p_1, \cdots, p_{i-1}} \text{dist}(p, \{p_1, \cdots p_{i-1}\})$
4: **end for**
5: **for** $\forall p \in SKY$ **do**
6: $i \leftarrow \text{argmin}_{\forall p_i \in [1, |\overline{SKY}|]} \text{dist}(p, p_i)$
7: $C_i \leftarrow C_i \cup \{p\}$
8: **end for**
9: **for** $1 \le i \le |\overline{SKY}|$ **do**
10: **for** $1 \le i \le d$ **do**
11: $q.att_j \leftarrow \min_{\forall p' \in C_i}(p'.att_j)$
12: **end for**
13: $\overline{SKY} \leftarrow \overline{SKY} \cup q$
14: **end for**

ス構造は，R 木と同じであるが，各ノードが保持する情報が異なる．ここでは，最大スカイライン点数を 2 とする．表 2 は，各中継ノードが保持するスカイライン点，および各葉ノード保持する属性値情報を示す．例えば，R5 におけるスカイライン点は 3 つ (<0.1,1.0>, <0.8,0.1>, <0.3,0.4>) あるが，R7 は 2 つのスカイライン点しか保持することができない．そこで，R7 は <0.1,1.0> と <0.3,0.4> を集約し，仮想スカイライン点 <0.1,0.4> を作成し，<0.8,0.1> とともに記録する．図 4 では，線上の円は R7 が保持する R5 および R6 におけるスカイライン点，および図内の白円は仮想スカイライン点を表す．

表 2: 各ノードが保持するスカイライン点

ノード	スカイライン点
R7	(R5:<0.1,0.4>,<0.8,0.1>), (R6:<0.2,0.7>,<0.3,0.2>)
R6	(R3:<0.9,0.3>,<0.2,0.7>), (R4:<0.3,0.3>,<0.6,0.2>)
R5	(R1:<0.1,1.0>,<0.8,0.1>), (R2:<0.3,0.4>)
R4	(o_7:0.3,0.3), (o_8:0.6,0.2), (o_9:0.6,0.6)
R3	(o_5:0.9,0.3), (o_6:0.2,0.7)
R2	(o_3:0.3,0.8), (o_4:0.3,0.4)
R1	(o_1:0.1,1.0), (o_2:0.8,0.1)

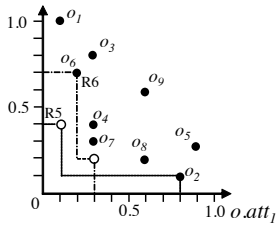


図 4: R7 が保持する R5 と R6 のスカイライン点

次に，SKY-R 木の構築方法について説明する．基本的に，R 木 [5] と構築方法は同様で，Insert を繰り返す（アルゴリズム 2）．ChooseLeaf と Split は，標準的なものと同じ方法で行う．ChooseLeaf は，新しいデータをノードに追加した場合に，MBR の増加が最小となると葉ノードを選択する関数，および Split は，ノードを MBR が最も離れるように 2 つのノードに分割する関数である．

Algorithm 2 Insert algorithm

1: $N \leftarrow$ ChooseLeaf($data$)
2: Add $data$ to N
3: **if** exceed maximum number of child nodes of N **then**
4: $\{N_1, N_2\} \leftarrow N$.Split()
5: **if** N is root **then**
6: Initialize new root NR and add N_1 and N_2 to NR
7: **else**
8: Ascend from N to root, and update MBR and skyline points
9: **end if**
10: **else if** N is not root **then**
11: Update the MBR and skyline points of the ancestor node of N
12: **end if**

アルゴリズムにおける，R 木と SKY-R 木の違いは，祖先ノードのスカイライン点を更新する点である（11 行）．ノードが保持するスカイライン点が $|\overline{SKY}|$ を超えた場合，アルゴリズム 1 によりスカイライン点の数を減らす．

3.2 位置依存 Top-k 検索処理

位置依存 Top-k 検索処理では，best-first 検索アルゴリズム（文献 [6] など）を用いる．best-first 検索アルゴリズムでは，ヒープ内で最も小さいスコアのエントリから順に検索する．データのスコアは式 (1) によって定義されている．ここでは，ノード N のスコアを以下の式で定義する．

$$score(q, N) = \alpha \frac{MinD(q.loc, N.MBR)}{MaxD} + (1-\alpha)\text{argmin}_{\forall p \in N.sp} \frac{\sum_{i=1}^{d} q.w_i \cdot p.att_i}{MaxA}. \quad (2)$$

$MinD(q.loc, N.MBR)$ は，$q.loc$ と N の MBR の最小距離を示す．式 (2) により，N の子孫データのうち最小のスコアとなりうるスコアを計算する．

アルゴリズム 3 において，MINHEAP は，格納しているデータとノードをスコアの昇順にソートしているヒープである．そのため，MINHEAP の先頭の要素がデータの場合，スコアが最小のデータとなる（4 から 9 行目）．$TOP_k(q)$ に格納しているデータ数が k となったとき，処理を終了する（6 行）．ここで，データおよびノードを MINHEAP に挿入する際，MINHEAP 内のデータのスコアが挿入予定のものより小さいものが，$(k - TOP_k(q))$ 個ある場合，挿入を行わない（11 行目）．MINHEAP 内のデータおよびノードの数を削減することにより，ソートのための計算時間を削減することができる．

例: 図 1 において，クエリを q ($q.loc$=$\{0.5, 0.5\}$, $q.w$=$\{0.5, 0.5\}$)，$\alpha = 0.5$，および $k = 2$ とする．ノードおよびデータのスコアを，それぞれ式 (2)，および式 (1) で計算する．上位 2 位のデータを検索する場合における，ヒープ内の要

Algorithm 3 Search algorithm

Input: k, q and index structure
Output: $TOP_k(q)$

```
 1: MINHEAP.insert(root, 0)
 2: while MINHEAP.size() ≠ 0 do
 3:     N ← MINHEAP.first()
 4:     if N is object then
 5:         TOP_k(q).insert(N)
 6:         if |TOP_k(q)| ≥ k then
 7:             return TOP_k(q)
 8:         end if
 9:     else
10:         for ∀n_i ∈ N.entry do
11:             if number of data with smaller score in MINHEAP than
                score(q, n_i) < (k − |TOP_k(q)|) then
12:                 if N is leaf node then
13:                     MINEHEAP.insert(Data, score(q, n_i))
14:                 else
15:                     MINHEAP.insert(Node, score(q, n_i))
                        //If score is a same value, a smaller MBR is better.
16:                 end if
17:             end if
18:         end for
19:     end if
20: end while
```

素をトレースしたものを以下に示す.

(1) MINHEAP{(R7,0)}.
(2) MINHEAP{(R5,0.3),(R6,0.3)}.
(3) MINHEAP{(R6,0.25),(R2,0.453),(R1,0.5846)}.
(4) MINHEAP{(R4,0.412),(R2,0.556),(R1,0.719),(R3,0.719)}.
(5) MINHEAP{(o_7,0.4118),(R2,0.556),(o_8,0.615),(R1,0.719),(R3,0.719)}. //(o_9, 0.853) は，MINHEAP に挿入されない.
(6) o_7 を検索結果に挿入.
(7) MINHEAP{(R2,0.556),(o_8,0.615),(R1,0.719),(R3,0.719)}.
(8) MINHEAP{(o_4,0.556),(o_8,0.615),(R1,0.719),(R3,0.719)}. //(o_3, 0.904) は，MINHEAP に挿入されない.
(9) o_4 を検索結果に挿入; 終了.

この例では，確実に検索結果に入らないデータを検索することなく，図 3 における木の一部のみを検索することにより，正確な検索結果を取得できる.

4. 拡張インデックス構造

SKY-R 木は，R 木と同様の構築方法を用いるため，構築時にデータの位置のみを考慮する．しかし，データの属性値も位置と同様に検索領域の削減に有効であるため，インデックス構造の構築時に考慮すべき要素である．そこで，構築時に属性値間の類似度も考慮にいれた，*S2KY-R* 木 (similarity skyline R 木) を提案する.

4.1 S2KY-R 木

S2KY-R 木は，SKY-R 木と同様に，各ノードはスカイライン点を保持しているが，構築時に考慮する点が異なる．つまり，アルゴリズム 2 における，ChooseLeaf と Split が異なる．具体的には，SKY-R 木（R 木 [5] も同様）では，MBR の面積のみを考慮するが，S2KY-R 木では，データを挿入する際，MBR の面積と属性値間の類似度を考慮する．まず，ChooseLeaf において，新しいデータに対して挿入パスを適切に決定するために，現在のノードからどの子端末を選択するかを以下のように決定する.

現在のノードの p 個のエントリ $E_1, \cdots, E_p$，および挿入されるデータ o を考える．R 木では，MBR の増加分を以下の式で計算する.

$$AreaIncrease(o, E_x) = Area(o + E_x) - Area(E_x) \quad (3)$$

$Area(x)$ は，x を含む MBR の面積，E_x $(1 \leq x \leq p)$ は，現在のノードのエントリを表す．より小さい $AreaIncrease$ は，新しいデータが子ノードの子孫データに距離的に近いことを意味する.

S2KY-R 木は，以下の定義に基づいて，属性値間の類似度を計算する.

定義 3 (属性値間の類似度). 類似度は，新しく挿入されるデータの属性値とエントリ E_x のスカイライン点との最小距離とする.

$$Similarity(o, E_x) = \min_{\forall p \in E_x.sp} dist(o.att, p). \quad (4)$$

$dist(x, y)$ は，属性値 x と y の距離を示す.

最後に，$AreaIncrease$ と $Similarity$ の合計により，$Increase$ を計算する.

$$Increase(o, E_x) = \beta \frac{AreaIncrease(o, E_x)}{MaxArea} + (1 - \beta) \frac{Similarity(o, E_x)}{MaxSim}. \quad (5)$$

$MaxArea$ と $MaxSim$ は，正規化のための全体の領域および最大の類似度，β はパラメータとなる．$Increase$ が最小となるノードは，距離が近く，属性値が類似しているため，次のノードして選ばれる．この計算は，葉端末に行き着くまで続けられる（アルゴリズム 4）．ここで，β が 1 の場合，S2KY-R 木は SKY-R 木と同じになる.

Algorithm 4 ChooseLeaf algorithm

```
1: N ← root
2: while 1 do
3:     if N is leaf node then
4:         return N
5:     end if
6:     N ← np in the entry in N with the smallest increase
7: end while
```

さらに，Split において，スカイライン集合間の距離を以

下の式で定義する．

$$Sdist(SKY_A, SKY_B) = \frac{\sum_{p \in SKY_A} \min_{\forall q \in SKY_B} dist(p,q)}{|SKY_A| + |SKY_B|} + \frac{\sum_{q \in SKY_B} \min_{\forall p \in SKY_A} dist(q,p)}{|SKY_A| + |SKY_B|}. \quad (6)$$

この式は，スカイライン間の最小の距離の平均を計算する．$Sdist(x,y)$ が小さい場合，スカイライン集合 x と y の距離が近いことを表す．

Algorithm 5 Split algorithm

```
1: for ∀ pair of E_i and E_j in entries do
2:     d_ij ← area(E_i + E_j) − area(E_i) − area(E_j)
3:     sim_ij ← Sdist(SKY_Ei, SKY_Ej)
4:     dif_ij = βd_ij + (1 − β)sim_ij
5: end for
6: Choose the pair with the largest dif value to be the first elements of
   the two group
7: while not all entries in N belong a group do
8:     if one group needs to include all remaining entries then
9:         Assign all remaining entries to it and break
10:    end if
11:    Calculate Increase for all remaining entries to two groups
12:    Choose the entry with the largest Increase and add it to the other
       group
13: end while
```

アルゴリズム 5 では，まず，2 つのエントリを最初の要素として選択する（1 から 6 行目）．他のエントリはどちらかのグループに割り当てられるが，$Increase$ が大きいエントリの割り当てを防ぐために，最大の $Increase$ をもつエントリをもう片方のグループに割り当てる（グループが 2 つあるため，各エントリは 2 つの $Increase$ をもつ）（11 から 12 行目）．さらに，片方のグループへの偏りを防ぐため，最小の子ノード数を事前に決定しておく（基本的に，最大子ノード数の半分となる）（8 から 9 行目）．

例: 図 5 は，図 1 において，S2KY-R 木を構築した際の各ノードの MBR を表す．この例では，o_4 と o_7 が似ている属性値をもつため，同じノード R3 に保持される．

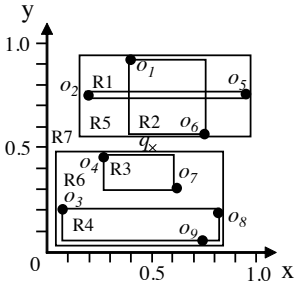


図 5: S2KY-R 木における最小矩形

4.2 検索処理

S2KY-R 木においても，アルゴリズム 3 を用いて位置依存 Top-k 検索を処理する．

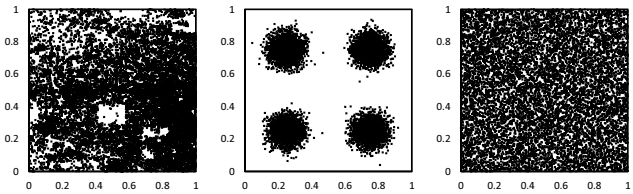
S2KY-R 木では，葉ノードがより似ているデータを保持しやすくなる．それにより，スコアが比較的小さい（上位 k 位とは限らない）k 個のデータを MINHEAP に SKY-R 木より早く挿入しやすくなるため，MINHEAP 内のデータ数およびノード数が少なくなる．結果として，計算時間が削減される．

5. 性能評価

位置依存 Top-k 検索処理における, SKY-R 木および S2KY-R 木の性能を評価する．また，比較手法として，Max aR 木を用いる．

データセット． データの位置情報として，実データと人工データを用いる．実データは，Foursqure から抽出した 10km×10km の東京の施設，および人工データは一様分布およびクラスタ分布となる（図 6 参照）．人工データの属性値として，d 次元の一様分布および非相関分布を用いる．実データの Foursquare から抽出したデータには，属性値が付与されてなかったため，一様分布属性値を与えた．

実データは 45,219 個，および人工データは 100,000 個のデータが存在する．



(a) 実データ　(b) クラスタ分布　(c) 一様分布

図 6: 位置情報

設定. 全てのインデックスは，メモリ内に存在し，ページサイズを 4KB とした．ノードのサイズは，SKY-R 木および S2KY-R 木では $(16 + 4d \cdot |\overline{SKY}|)$B，Max aR 木では，$(16 + 4d)$B となる．つまり，SKY-R 木および S2KY-R 木の最大子ノード数は $\lceil 4,096/(16 + 4d \cdot |\overline{SKY}|) \rceil$，および Max aR 木の最大子ノード数は $\lceil 4,096/(16 + 4d) \rceil$ となる．また，SKY-R 木および S2KY-R 木における，$|\overline{SKY}|$，および β を，予備実験の結果より，それぞれ 5，および 0.8 とする．全てのアルゴリズムを c++で実装し，Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz with 8.00 GB RAM を搭載するサーバによって実験した．

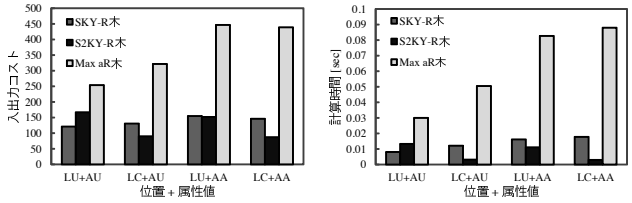
実験においては，ランダムなクエリ（$q.loc$ と $q.w$ がランダム）を 100 個生成し，“入出力コスト”（探索ノード数）および “計算時間” の平均値を調べた．表 3 はパラメータを示し，太字はデフォルト値とする．

表 3: パラメータ設定

パラメータ	値
要求データ数 k	1, 5, 10, **20**, 50, 100
α	0, 0.2, **0.3**, 0.4, ,0.6, 0.8, 1.0
次元数 d	1, 2, **3** , 4
位置情報	実データ，一様分布，クラスタ分布
属性値情報	一様分布，非相関分布

5.1 人工データ

まず，人工データにおける結果を，図 7a および 7b に示す．これらの図において，“LU”, “LC”, “AU”, および “AA” は，それぞれ一様分布位置情報，クラスタ分布位置情報，一様分布属性値，および非相関分布属性値を示す．SKY-R 木および S2KY-R 木は，Max aR 木よりも，全ての組合わせにおいて，高性能であることがわかる．この結果より，スカイライン点が検索領域削減に有効であることがわかる．また，SKY-R 木は，一様分布属性値の場合において，一方，S2KY-R 木は，クラスタ分布位置情報の場合において，性能がよくなる．これは，位置情報のみを用いるインデックス構造の場合，属性値に偏りのない方が有効であり，一方，属性値の類似度を用いるインデックス構造の場合，位置に偏りのあるほうが有効であるためである．



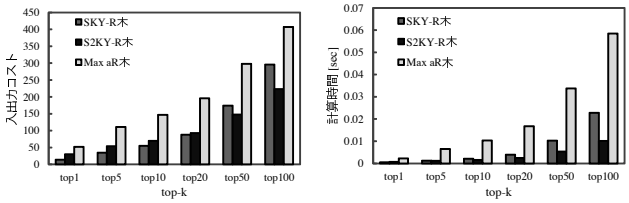
(a) 入出力コスト　　(b) 計算時間

図 7: 位置情報と属性値情報分布の影響

5.2 実データ

次に，実データを用いた場合の結果を示す．

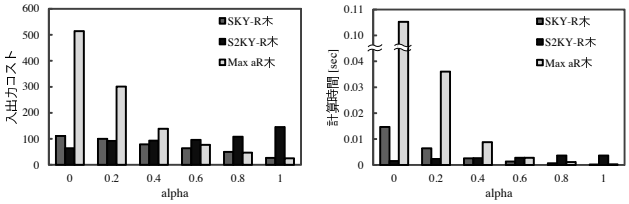
k の影響． k の増加に伴い，検索領域が拡大するため，効率的にクエリを処理することが重要となる．図 8a，および 8b に k を変化させたときの評価結果を示す．SKY-R 木および S2KY-R 木は，全ての k において，Max aR 木よりよい性能を示している．入出力コストにおいて，k が小さい場合，SKY-R 木は S2KY-R 木より小さく，k が大きい場合は，S2KY-R 木の方が小さい．これは，SKY-R 木の場合の各ノードの MBR の方が S2KY-R 木の場合より小さいため，S2KY-R 木の方が検索領域が若干大きくなってしまうためである．一方，k が大きい場合では，S2KY-R 木では，一度に検索結果に入るデータを取得しやすいため，入出力コストが小さくなる．さらに，S2KY-R 木では，計算時間が SKY-R 木より非常に小さい．これは，S2KY-R 木では，一度の探索で比較的スコアがよいデータを MINHEAP に挿入するため，MINHEAP 内に挿入されるデータおよびノードの数が減少し，計算時間が削減するからである．



(a) 入出力コスト　　(b) 計算時間

図 8: k の影響

α の影響． パラメータ α は，ユーザにおけるデータの位置の重要性を表しており，α が大きいほど，より近いデータを優先して検索する．図 9a，および 9b に α を変化させたときの評価結果を示す. S2KY-R 木の性能は，α が小さいとき，他のインデックス構造よりもよいが，α が大きいときには悪くなる．これは，S2KY-R 木は属性値を考慮することで検索領域を削減できるが，MBR が若干大きくなってしまうためである．よって，α が大きくなるにつれて，入出力コストが徐々に増加する．一方，SKY-R 木は，全ての α において，Max aR 木よりよい性能を示す．これは，SKY-R 木の方が検索領域をより削減できるためである．



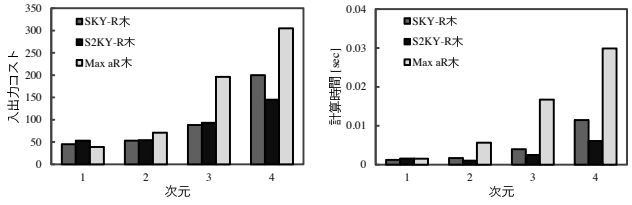
(a) 入出力コスト　　(b) 計算時間

図 9: α の影響

d の影響． 属性値の次元数が増加するに伴い，属性値の多様性が増加する．そのため，属性値による検索領域の削減が難しくなる．図 10a，および 10b に d を変化させたときの評価結果を示す．SKY-R 木および S2KY-R 木は，d が 1 の場合を除き，Max aR 木より性能がよい．d が 1 の場合，スカイライン点は常に 1 つとなるため，デフォルト値を 5 としている SKY-R 木および S2KY-R 木は無駄に子ノードの数を減らしてしまう．そのため，Max aR 木の方が性能がよくなる．d が増加すると，属性値の類似度を考慮する点がより有効に働くため，S2KY-R 木の性能がよくなる．

6. 関連研究

Top-k 検索． Top-k 検索処理は幅広く研究されている [7]. Tao らは [11]，branch-and-bound アルゴリズムを提案している．branch-and-bound アルゴリズムは，R 木 [5] を基として，最もよいスコアの可能性があるデータを保持するノードを探索する．一方，データのタプルから階層化インデックスを構築し，少ない階層のみの検索により結果を保証す



(a) 入出力コスト　(b) 計算時間

図 10: d の影響

る Top-k 検索手法も提案されている [2], [9].

これらの手法は，位置依存 Top-k 検索処理に用いることもできるが，非効率的である．属性値の上位 1 位のデータを検索し，そのデータの位置によるスコアを付与することを繰り返すことにより，検索結果を取得可能であるが，指定位置から非常に遠いデータを検索してしまう可能性が高い．

位置依存 Top-k 検索を定義している文献 [12] では, branch-and-bound アルゴリズム [11] により，データを検索している．しかし，branch-and-bound アルゴリズムを用いて位置依存 Top-k 検索を行う場合，全てのクエリに対して，木構造を構築する必要があり，コストが非常に大きい．文献 [12] では，連続的逆 Top-k 検索について提案しているが，構築コストを考慮していない．

k 最近傍検索. 空間データベースにおける k 最近傍検索 [10] では，指定位置に近い k 個のデータを検索できるため，位置依存 Top-k 検索に応用することができる．しかし，スコアが非常に大きいデータも検索してしまうため，非効率的である．

Spatial keyword top-k 検索. Spatial keyword top-k 検索では，キーワードに関連があり，指定位置に近いデータを検索する．Felipe ら [4] は，signature file と R 木を組み合わせた手法を提案した．この手法では，限られたキーワードにのみ対応し，キーワードとの適合数に応じて，データを順位付けする．Cong ら [3] は，文章とキーワードの関連度および位置を考慮して，データを順位付けする手法を提案した．Spatial keyword top-k 検索では，ドキュメント検索に着目しており，属性値を着目した位置依存 Top-k 検索とは，順位付けの方法が異なる．

7. まとめ

本稿では，位置依存 Top-k 検索処理のための新しいインデックス構造である SKY-R 木を提案した．SKY-R 木では，R 木とスカイラインを組み合わせることにより，位置および属性値から検索領域を削減することができる．また，効率的な位置依存 Top-k 検索処理アルゴリズムを提案した．インデックス構造および位置依存 Top-k 検索処理アルゴリズムにより，効率的に検索結果を取得することができる．さらに，属性値の類似度を考慮してインデックス構造を構築する，S2KY-R 木も提案した．実験により，提案インデックス構造は，入出力コストおよび計算時間のどちらの観点からも有効であることがわかった．SKY-R 木および S2KY-R 木の性能差は，検索条件やデータの分布に依存する．

本稿で提案したインデックス構造は，様々な拡張が考えられる．まず，キーワード検索との組合わせ (例えば，ある位置に近い人気の “中華” レストラン) が考えられる．次に，移動端末のための連続的位置依存 Top-k 検索への拡張も有益と思われる．さらに，本稿では，データと指定位置の距離をユークリッド距離で計算したが，道路ベースの距離に拡張することも重要な研究課題である．

[謝辞]

本研究の一部は，文部科学省研究費補助金・基盤研究 S（21220002），基盤研究 B（24300037），および特別研究員奨励費（24-293）の研究助成によるものである．ここに記して謝意を表す.

参考文献

[1] Borzsony, S., Kossmann, D. and Stocker, K.: The skyline operator, *ICDE*, pp. 421–430 (2001).

[2] Chang, Y.-C., Bergman, L., Castelli, V., Li, C.-S., Lo, M.-L. and Smith, J. R.: The onion technique: indexing for linear optimization queries, *ACM SIGMOD Record*, Vol. 29, No. 2, pp. 391–402 (2000).

[3] Cong, G., Jensen, C. S. and Wu, D.: Efficient retrieval of the top-k most relevant spatial web objects, *VLDB Journal*, Vol. 2, No. 1, pp. 337–348 (2009).

[4] De Felipe, I., Hristidis, V. and Rishe, N.: Keyword search on spatial databases, *ICDE*, pp. 656–665 (2008).

[5] Guttman, A.: R-trees: a dynamic index structure for spatial searching, *SIGMOD*, pp. 47–57 (1984).

[6] Hjaltason, G. R. and Samet, H.: Distance browsing in spatial databases, *ACM TODS*, Vol. 24, No. 2, pp. 265–318 (1999).

[7] Ilyas, I. F., Beskales, G. and Soliman, M. A.: A survey of top-k query processing techniques in relational database systems, *ACM CSUR*, Vol. 40, No. 4, p. 11 (2008).

[8] Lazaridis, I. and Mehrotra, S.: Progressive approximate aggregate queries with a multi-resolution tree structure, *ACM SIGMOD Record*, Vol. 30, No. 2, pp. 401–412 (2001).

[9] Lee, J., Cho, H. and Hwang, S.-w.: Efficient dual-resolution layer indexing for top-k queries, *ICDE*, pp. 1084–1095 (2012).

[10] Roussopoulos, N., Kelley, S. and Vincent, F.: Nearest neighbor queries, *ACM SIGMOD Record*, Vol. 24, No. 2, pp. 71–79 (1995).

[11] Tao, Y., Hristidis, V., Papadias, D. and Papakonstantinou, Y.: Branch-and-bound processing of ranked queries, *Information Systems*, Vol. 32, No. 3, pp. 424–445 (2007).

[12] Vlachou, A., Doulkeridis, C. and Nørvåg, K.: Monitoring reverse top-k queries over mobile devices, *MobiDE*, pp. 17–24 (2011).

[13] Vlachou, A., Doulkeridis, C. and Nørvåg, K.: Distributed top-k query processing by exploiting skyline summaries, *Distributed and Parallel Databases*, Vol. 30, No. 3-4, pp. 239–271 (2012).