修士論文 要旨

高次元直交領域探索問題に対する省空間データ構造

数理情報学専攻 48-166203　**石山 一樹**

指導教員　**定兼 邦彦 教授**

1　はじめに

直交領域探索問題とは，d 次元空間上の n 個の点から成る集合 P と軸並行な直交領域 Q が与えられたときに，Q に含まれる P の点に関する情報を求める問題である．出力するものはクエリの種類によって異なり，領域 Q に含まれる点を列挙する reporting クエリや点の数を出力する counting クエリなどがあり，データベース内の検索などに応用がある．この問題では実際の応用を考えて，先に点集合 P からデータ構造を構築しておいて，領域 Q が与えられたらデータ構造を用いてクエリに答える，という問題設定を考える．したがって，大量のデータを扱う場合には，クエリ時間計算量だけでなくデータ構造の空間計算量も重要となる．

2　既存研究と本研究の成果

直交領域探索問題に対する簡潔データ構造[*1]を考える場合は，点集合 P として $[n]^d$ 上のものを考える．一般の $\mathbb{R}^d$ 上の問題を $[n]^d$ 上の問題に帰着できることが知られており [3]，このような仮定は一般性を失わない．

2 次元の場合は wavelet tree が $n\lg n+\mathrm{o}(n\lg n)$ ビットの空間計算量で，reporting クエリに $\mathrm{O}((1+k)\lg n)$ 時間[*2][*3]で答えられることが知られている [4]．さらに，Bose らが提案したデータ構造ではクエリ時間計算量を $\mathrm{O}(\lg\lg n)$ 倍改善している [2]．一方，多次元の場合は KDW-tree が $dn\lg n+\mathrm{o}(dn\lg n)$ ビットの空間計算量で，reporting クエリに $\mathrm{O}\left(\left(n^{\frac{d-2}{d}}+k\right)\lg n\right)$ 時間[*4]で答えられることが知られている [6]．しかし，この解析には誤りがあり，本研究では $\Omega\left(\left(dn^{\frac{d-2}{d-1}}+dk\right)\lg n\right)$ 時間かかる場合が存在することを示し，新しい計算量の上界として $\mathrm{O}\left(\left(d^2n^{\frac{d-2}{d-1}}+dk\right)\lg n\right)$ を与えた．

本研究では，3 つの新しい省空間データ構造を提案する．提案手法 1 では KDW-tree と同じ空間計算量で，reporting クエリの時間計算量を $\mathrm{O}\left(\left(d^3n^{\frac{d-2}{d}}+dk\right)\frac{\lg n}{\lg\lg n}\right)$ 倍に改善した．

提案手法 2 では空間計算量を改善した．$[n]^d$ 上の点集合を表現する際の情報理論的下限は $(d-1)n\lg n+\mathrm{O}(n)$ ビットであり，d を定数と見なす場合には，KDW-tree や提案手法 1 は簡潔とは言えない．提案手法 2 では，空間計算量を $(d-1)n\lg n+\mathrm{o}((d-1)n\lg n)$ ビットに抑えつつ，reporting クエリに $\mathrm{O}\left(d^2n^{\frac{d-2}{d-1}}+k\lg n\right)$ 時間で答えられる．

提案手法 3 は実用的に高速であることを目指して作ったものである．実際のデータベースへの応用を考えると，データの次元数 d は大きいが，探索に使う次元数 d' は小さいという場合が考えられ，そのようなときに高速にクエリに答えられることが期待される．データ構造の空間計算量は $(d-1)n\lg n+\mathrm{o}((d-1)n\lg n)$ ビットであり，漸近的に情報理論的下限を達成する．

3　提案手法の概要

ここでは提案手法 2 について簡単に説明する．

以下では，d 個の次元を次元 0，次元 1，...，次元 $d-1$ と名付ける．この手法では各点の z-value [5] を用いる．ここで点 p の z-value $z(p)$ は，点 p の第 i 座標値が 2 進数で $b_i^0b_i^1\cdots b_i^{l-1}$ と書けるとき，$z(p)=b_0^0b_1^0\cdots b_{d-1}^0b_0^1b_1^1\cdots b_{d-1}^1\cdots b_0^{l-1}b_1^{l-1}\cdots b_{d-1}^{l-1}$ で定義される．提案手法では，各点 $p\in P$ の第 1 座標値から第 $d-1$ 座標値だけを使った長さ $(d-1)\lg n$ の z-value $z(p)$ を考え，$z(p)$ を第 0 座標値の昇順に並べた整数列から wavelet tree を構築する．

ここでは wavelet tree が具体的にどのようなデータ構造であるかは説明しないが，このデータ構造を用いることで次のような探索が行える．今，クエリ領域が $Q=\left[l_0^{(Q)},u_0^{(Q)}\right]\times\cdots\times\left[l_{d-1}^{(Q)},u_{d-1}^{(Q)}\right]$ と与えられているとする．このとき，最初に領域 $R=\left[l_0^{(Q)},u_0^{(Q)}\right]\times[0,n-1]\times\cdots\times[0,n-1]$ に着目し，次元 1 から次元 $d-1$ までの $d-1$ 個の次元について R を 2 等分していくことを考える．つまり，最初のステップでは次元 1 を $[0,\frac{n}{2}-1]$ と $[\frac{n}{2},n-1]$ に分割した 2 つの領域を考え，その次の段階では，分割して得られた 2 つの領域それぞれについて次元 2 を 2 等分した 4 つの領域を考える，ということを続ける．次元 $d-1$ まで分割して 2^{d-1} 個の領域ができたら，再び次元 1 の分割に戻る．

[*1] あるデータを表現するのに最低限必要な量（情報理論的下限）が Z ビットのとき，そのデータを表現するデータ構造の空間計算量が $Z+\mathrm{o}(Z)$ ビットで，クエリを効率的にサポートするとき，そのデータ構造は簡潔であるという．

[*2] k は列挙する点の数を表す．

[*3] $k=0$ を代入すると counting クエリの時間計算量になる．

[*4] d を定数と仮定した場合の結果．

これは，z-value を 1 桁目から順に決定していくことに対応している．このような探索を行うと，注目する領域 R が次第に小さくなり，R と Q の交わりがなくなった場合には，それ以上 R を分割する必要がなくなる．また，R が Q に完全に含まれるようになった場合には，R に含まれる点は Q にも含まれると判断できる．提案するデータ構造を用いると，R を分割した後に分割後に含まれる点の数を定数時間で計算できる．したがって，counting クエリの時間計算量は考える領域 R の数に比例し，その個数は $\mathrm{O}\left(d^2 n^{\frac{d-2}{d}}\right)$ 個である．また，reporting クエリの場合は，Q に含まれる R が見つかった後，R に含まれる点の座標値を $\mathrm{O}((d-1)\lg n)$ 時間で求めることができる．したがって，reporting クエリの時間計算量は $\mathrm{O}\left(d^2 n^{\frac{d-2}{d}} + dk \lg n\right)$ 時間である．

4 数値実験

提案するデータ構造が実用性を検証するために数値実験を行った．比較するデータ構造は提案する 3 つの手法の他に，点の座標値を配列で持ち線形探索する naïve，空間計算量が線形のデータ構造としてよく知られている kd-tree [1]，そして KDW-tree である．KDW-tree については，文献 [6] の著者らが公開しているものを使用する．ただし，公開されている実装は KDWtree に加えて点の座標値の配列も併用することで高速化を図っている．そのためデータ構造の空間計算量は理論的な値よりも大きくなっている*5．また，提案手法の実装には succinct data structure library を使用し，wavelet tree 内のビット列として通常のビットベクトルを用いた場合と RRR ベクトルを用いた場合も比較する．

まずデータ構造のメモリ使用量を比較すると，ほとんどの場合で 2 つ目の提案手法と 3 つ目の提案手法が小さかった．これは理論的な結果と一致している．

次に counting クエリの実行時間を比較する．今回はクエリ領域の体積を空間全体の体積で割った値（selectivity）を変化させて実験した．3 次元の場合の結果（図 1）から，低次元の場合は 2 つ目の提案手法が速いことがわかる．一方で 24 次元の場合，$d' = 24$ のとき（図 2）は提案手法はそれほど速くないが，$d' = 3$ の場合（図 3）は 3 つ目の提案手法が速いことがわかる．ただし，提案手法においても KDW-tree の実装と同じように点の座標値の配列を併用することで高速化できる可能性がある．

*5 おおよそ 2 倍程度大きいと考えられる．

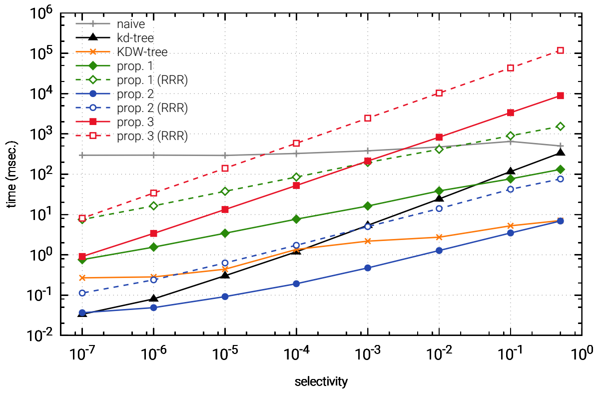

図 1. $n = 2^{26}, d = 3, d' = 3$ の場合．

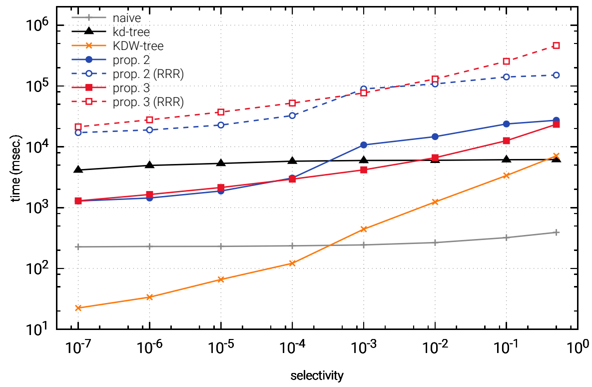

図 2. $n = 2^{24}, d = 24, d' = 24$ の場合．

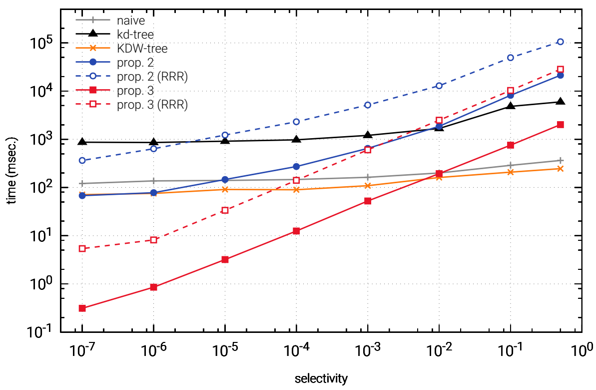

図 3. $n = 2^{24}, d = 24, d' = 3$ の場合．

参考文献

[1] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, Vol. 18, No. 9, pp. 509–517, 1975.

[2] P. Bose, M. He, A. Maheshwari, and P. Morin. Succinct orthogonal range search structures on a grid with applications to text indexing. In *Proc. of WADS*, pp. 98–109, 2009.

[3] Harold N. Gabow, Jon Louis Bentley, and Robert E. Tarjan. Scaling and related techniques for geometry problems. In *Proc. of STOC*, pp. 135–143, 1984.

[4] V. Mäkinen and G. Navarro. Position-restricted substring searching. In *Proc. of LATIN*, pp. 703–714, 2006.

[5] Guy M. Morton. *A computer oriented geodetic data base and a new technique in file sequencing*. International Business Machines Company New York, 1966.

[6] Y. Okajima and K. Maruyama. Faster linear-space orthogonal range searching in arbitrary dimensions. In *Proc. of ALENEX*, pp. 82–93, 2015.