Vol.2022-HPC-185 No.3
2022/7/27

空間充填曲線を用いたベクトルプロセッサにおける k 近傍法の高速化

小寺 雅司[1,a)]　サハ ソウラブ[1,b)]　荒木 拓也[1,c)]

概要：本稿では，空間充填曲線を用いることでベクトルプロセッサに適した形で k 近傍法を高速化する手法を提案し，SX-Aurora TSUBASA を用いて k 近傍を高速化した結果について報告する．k 近傍法はトレーニングデータとクエリデータの距離計算が計算の大部分を占めている．提案手法では空間充填曲線を用いることでトレーニングデータとクエリデータの距離計算を減らし，ベクトルプロセッサを用いることで計算を高速化する．性能評価の結果，scikit-learn を用いた k 近傍法と比べ，SX-Aurora TSUBASA を用いた k 近傍法計算は最大 7 倍程度高速に計算できることを確認した．

Speeding up k-nearest neighbors search with space-filling curve using vector processors

Abstract: In this paper, we propose a method to perform k-nearest neighbors search with a space-filling curve method using a processing scheme suitable for vector processors, and report the results of speeding up k-nearest neighbors search using SX-Aurora TSUBASA. In the k-nearest neighbors search, the distance between training data and query data accounts for a large part of the computation. The proposed method speeds up the process by reducing the distance computation between training data and query data by using a space-filling curve. The performance evaluation results show that the k-nearest neighbors search using SX-Aurora TSUBASA is up to seven times faster than the k nearest neighbor method using scikit-learn.

1. はじめに

k 近傍法 (k-Neareset Neighbors，k-NN) は，データベースの中からクエリデータに対して最も近くにある k 個のデータを選ぶアルゴリズムであり，回帰やクラス分類に使われる基本的なアルゴリズムである．素朴な実装による k-NN はモデル作成時に時間がかからないが，この実装による k-NN を用いた分類や回帰は，推論時にデータベース全体のデータを参照することから，計算コストが高いという問題がある．しかし高速な予測が必要となる k-NN の応用例もあり，例えばオンライン広告のターゲッティングなどでは判断を 10 ミリ秒程度で求められることがあるため [1]，高速化は重要である．

k-NN を高速化するためには，ハードウェアの特性を活かした高速な演算処理をすることと k 個の近傍点を得るために必要のない計算をしないことが重要である．高速な演算処理を行うためには，ベクトル型プロセッサの活用が有望である．ベクトル型プロセッサでは，ひとつの命令を複数のデータ要素に対して一斉に実行することで効率的に処理を行うことができる．ベクトル型プロセッサの能力を最大限に活用するためには，一斉処理が困難な部分をできるだけ減らし，データ処理プログラム中の大部分をベクトル方式の演算に置き換えることができる工夫が必要である．

k-NN における必要のない計算を省くために kd-tree をはじめとする木構造を用いた手法がよく知られている．しかし木構造を用いた手法は再帰を用いることからベクトル型プロセッサを用いて，ひとつの命令を複数のデータ要素に対して一斉に実行するのが難しいという問題がある．

そこで我々は空間充填曲線による計算量を削減する手法を用いることで，ベクトル型プロセッサによる 高速化と計算量の削減を両立する実装を行った。以下ではまず 2 章で k-NN のアルゴリズムと空間充填曲線，特に Z-curve につ

[1] 日本電気株式会社 (デジタルテクノロジー開発研究所)
NEC Corporation(Digital Technology Development Laboratory)
a) masashi-kotera@nec.com
b) sourav-saha@nec.com
c) takuya_araki@nec.com

いて紹介する．次に 3 章で Z-curve を用いた k-NN の計算を紹介し，4 章で Z-curve を用いた k-NN をベクトル化する手法を提案する．5 章では提案手法と NEC が開発するベクトル型プロセッサである SX-Aurora TSUBASA による計算の高速化の効果を評価する．6 章で本研究の関連研究を紹介する．7 章で本研究のまとめを述べる．

2. 背景

2.1 k-NN のアルゴリズム

文献 [1][2] に記載の例を引用し，古典的な k-NN のアルゴリズムについて説明する．k-NN はトレーニングデータのデータセットの中から注目するクエリデータに最も近い k 個のトレーニングデータを選ぶアルゴリズムである．データセットの中にある全てのトレーニングデータとクエリデータの距離を計算し，計算した距離をソートして，k 番目までのトレーニングデータを選び出す．

k-NN の応用例としては回帰や分類がある．回帰の例としてはワインの価格の予想などがある．これは，ワインの酒齢やアルコール度数，産地などの特徴量からワインの価格を予想するというものである．分類の例としては，クレジットカードのマーケティングの問題などがある．この問題は新しい顧客 Q がクレジットカードの勧誘に応じてくれるかどうかを Q の類似顧客がどう応じたかに基づいて予測する問題である．その顧客のデータが表 1 であったとする．

表 1 顧客 Q は勧誘に応じるのか

顧客	年齢	収入 (千ドル)	Q との距離	反応 (目的変数)
Q	37	50	0	?
A	35	35	$\sqrt{229}$	◯
B	22	100	$\sqrt{2725}$	×
C	63	200	$\sqrt{23176}$	×
D	59	170	$\sqrt{14884}$	×
E	25	45	$\sqrt{169}$	◯

このサンプルでは，以前にクレジットカードの勧誘をしたことがある 5 人 (A, B, C, D, E) のデータが含まれている．この顧客について名前，年齢，収入及び，これらによって決定される距離，勧誘に応じたかどうか (◯ なら応じた，× なら応じなかったとする．) の結果が分かっている．その上で新しい顧客である Q が勧誘に応じてくれるかどうかを予測したい．

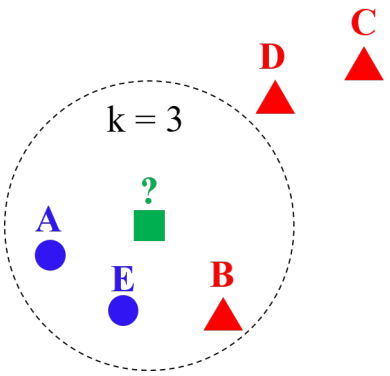

図 1 A, B, C, D, E と Q の位置関係

近傍点の数を 3 にしたとき，すなわち $k = 3$ のとき，近い人から順番に 3 人を並べると E, A, B である．E, A, B の中でクレジットカードの勧誘に応じたか否かの多数決を取ると，B はクレジットカードの勧誘に応じていないが，E, A はクレジットカードの勧誘に応じているため，Q はクレジットカードの勧誘に応じると予想される (図 1)．

k-NN はシンプルなアルゴリズムであるが強力な手法である．しかし，このアルゴリズムはトレーニングデータまたはクエリデータのデータセットが大きいとき，トレーニングデータとクエリデータデータの間の距離を計算するコストが高くなるのが問題である．kd-tree をはじめとする木構造を用いて計算コストを減らす手法もあるが [9]，木構造を用いた手法はベクトル化が難しい．そこで空間充填曲線と範囲検索を用いて距離計算の計算コストを減らすことを考える．

2.2 空間充填曲線

空間充填曲線は，任意の精度で多次元空間を埋め尽くす曲線である．その種類はいくつかあり，Z-curve や Hilbert-curve といったものがある (図 2)．今回の実装では Z-curve を用いる．

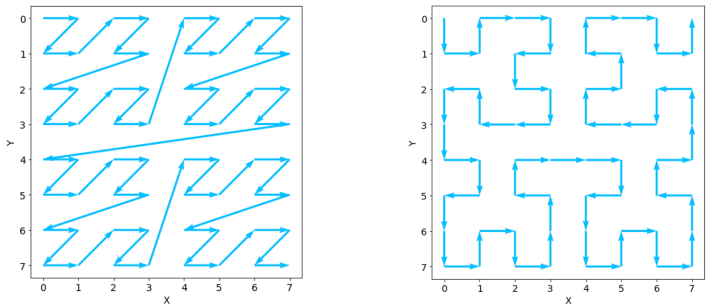

図 2 Z-curve(左) と Hilbert-curve(右)

空間充填曲線の性質について紹介する．空間充填曲線は多次元空間のデータを 1 次元空間に写像することができる．写像をすることで，多次元データを 1 次元空間上でソート

することができる．Z-curve を用いてデータを多次元空間から 1 次元空間へと写像すると，多次元空間上でのデータ同士の局所的な位置関係の情報を保持したまま写像することができる．つまり，元の多次元空間で近傍に存在したデータ同士は写像された先の 1 次元空間でも近傍に存在する (図 3)．

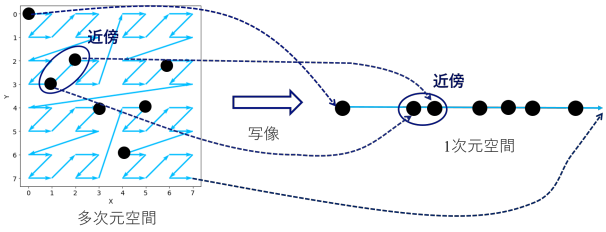

図 3 多次元空間から 1 次元空間への写像

Z-curve の他の性質として，多次元空間での範囲検索を 1 次元空間での範囲検索に置き換えることで，効率よく範囲検索を実現できるという性質がある．すなわち，多次元空間中に矩形を与え，その中に存在する点を検索することを考える．矩形の頂点の中から原点に最も近い点 (三角) を S 点，最も遠い点 (四角) を G 点とする．その矩形の S 点と G 点を結ぶ空間充填曲線は矩形内部の全ての点を通過する．そのため，空間充填曲線上で S 点と G 点の間で範囲検索をすることで 1 次元空間での範囲検索ができる (図 4)．

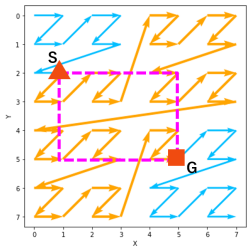

図 4 多次元空間中の矩形
S 点と G 点を通る Z-curve は多次元空間の矩形内部のすべての点を通る．

また，この多次元のデータを 1 次元のデータに写像するときの 1 次元の値を Z 値という．Z 値の計算については以下で例を用いて説明する (図 5)．

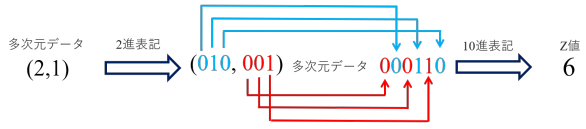

図 5 Z 値の計算

具体的に 2 次元のデータ (2,1) を例に用いて説明する．(2,1) を 2 進表記で (010,001) とする．(説明のために 3bit 表記にしている)，この 2 進表記されたデータについて，交互に 2 進表記された数値を混ぜ合わせて (000110) を得る．これを 10 進表記に戻すことで Z 値 6 を得る．この操作によって，多次元データを 1 次元の値に変換することができる．同様の操作をデータセットの全てのデータについて行い，得られた 1 次元の値をソートすると、多次元データの Z-curve 上での順番になる．

格子点のデータ (0,0),(0,1),(1,0)...,(7,7) について Z 値を計算した後ソートし，格子点を線で繋ぐことで図 2 の例で示した Z-curve を得る (図 6)．

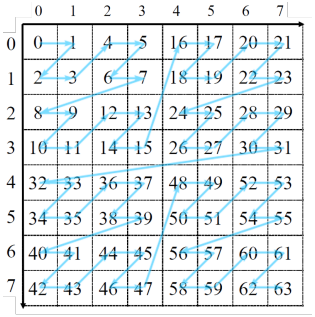

図 6 格子点の Z 値

そして，この Z 値を求める計算は論理和，論理積，論理シフトの bit 演算のみによって行うことができるためデータセットが十分に大きいとき，同時計算による処理を高速に行えるベクトルプロセッサによる高速な計算が可能である．

3. Z-curve を用いた k-NN の計算

この章では，Z-zurve を用いた k-NN の実現方法について説明する．

3.1 基本となる Z-curve を用いた k-NN の計算法

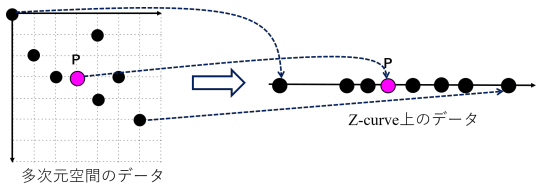

図 7
トレーニングデータとクエリデータを Z-curve 上に写像する．

はじめに，トレーニングデータの Z 値を計算して Z-curve 上でソートする．次にクエリデータの Z 値を計算してソートされたトレーニングデータの中に配置する．このとき Z 値は 1 次元の値なので，二分探索によって高速に配置する場所を決定することができる．Z-curve 上に配置したクエリデータの前後 k 個のトレーニングデータを選ぶ (ここでは前後 k 個のトレーニングデータとしているが計算を実行する上では，クエリデータの前後で合計 k 以上のトレーニ

ングデータを選べば良い)(図 7)．クエリデータとその前後 k 個のトレーニングデータとの間の元の多次元空間での距離を計算する (図 10)．

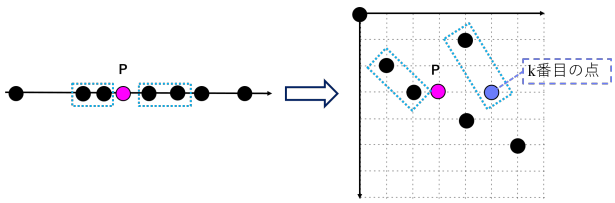

図 8

計算した距離についてソートする．k 番目の距離を R とする．

クエリ点 P を中心として，半径が R である超球を考える．また，この超球を含むような最小の矩形を考える．

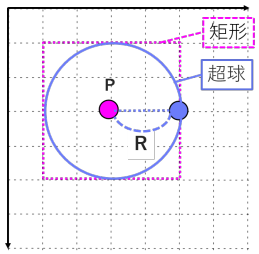

図 9 超球と超球を含む最小の矩形

この矩形の頂点の中で原点に最も近い点 (S 点) と原点から最も遠い点 (G 点) をとる．Z-curve 上で S 点から G 点まで探索をすれば，この矩形中の点を全て探索することができる．これによって範囲検索を効率的に行うことができ，以降，範囲検索実行時に探索するデータの集合を範囲検索領域と呼ぶ．

具体的にクエリデータ P の座標を (P_x, P_y) とすると，S 点の座標は $(R_x - R, P_y - R)$，G 点の座標は $(R_x + R, P_y + R)$ となる．S 点，G 点の Z 値を計算し，Z-curve 上でソートされたトレーニングデータの中で二分探索をすることで，探索するべき範囲が決まる．この探索する範囲に含まれる全てのトレーニングデータとクエリデータの元の多次元空間上での距離を計算する．超球が k 個以上のトレーニングデータを含むため，距離計算するトレーニングデータは k 個以上あり，計算した距離についてソートして k 番目までの距離に対応する点がクエリ点に対する k 近傍点である．

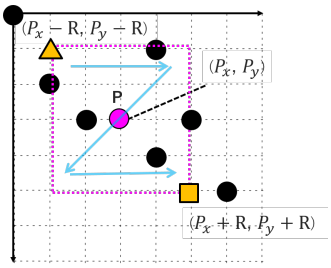

図 10 範囲検索領域

しかし，扱うデータが高次元になるほど効率の良い計算ができなくなり，この方法では高々 3 次元までのデータしか高速な計算を行えない．そこで [13] を参考に範囲検索領域を一度計算したあとに範囲検索領域の分割を行うことで計算コストを下げる手法を次節で説明する．この手法を用いることで 6 次元程度までのデータを効率よく計算できるようになる．但し，これより高い次元のデータについては次元の呪いにより，範囲検索が効率的に行えず計算効率が悪くなる．

3.2 検索範囲の分割による高速化

ここでは説明のため 2 次元のデータを用いる．空間充填曲線を用いて範囲検索をすることで計算の効率が良くなるが，範囲検索領域によって k 近傍の解とならない矩形の外の点が多く選ばれることがあり，計算効率が十分に良くならないことがある．具体的な例を紹介する．

下記の図は点線で囲まれた領域（マゼンタ）が範囲検索すべき多次元の領域で，斜線部分で表された領域（緑）で示された点が矩形外部の点である．

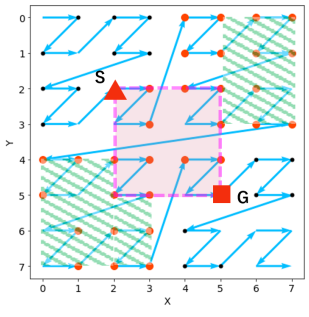

図 11 計算効率の悪い範囲検索領域

範囲検索領域が図 11 のように定まったときは k 近傍の解とならない矩形外部のトレーニングデータが多く計算の効率が悪くなる．この理由を説明するため n 階のクラスターを定義する．

図 12 の左の図の斜線部分 (青) で表された矩形の領域を 0 階のクラスターとする．0 階のクラスターは S 点の Z 値が 000000，G 点の Z 値が 111111 となる 1 つの矩形である．但し，説明のため Z 値の二進表記の長さは 6 としてい

る．図 12 の右の図の斜線と色で区分できる 4 つの矩形を 1 階のクラスターとする．1 階のクラスターは 4 つありS 点と G 点の組み合わせは 4 つある．S 点と G 点の Z 値の二進表記は上 2 桁が等しく (Z 値の二進表記の上二桁の組み合わせは 00, 10, 01, 11 の 4 つの組み合わせがある)，S 点の Z 値の二進表記の下 4 桁は 0000，G 点の Z 値の二進表記は下 4 桁が 1111 となるような矩形である．

具体的に矩形を構成する S 点と G 点の組は各々，S 点の Z 値の二進表記が 000000 であって G 点の Z 値の二進表記が 001111 となる組と，S 点の Z 値の二進表記が 100000 であって G 点の Z 値の二進表記が 101111 となる組と，S 点の Z 値の二進表記が 010000 であって G 点の Z 値の二進表記が 011111 となる組と，S 点の Z 値の二進表記が 110000 であって G 点の Z 値の二進表記が 111111 となる組である．同様に 2 階以降のクラスターも定義できる．

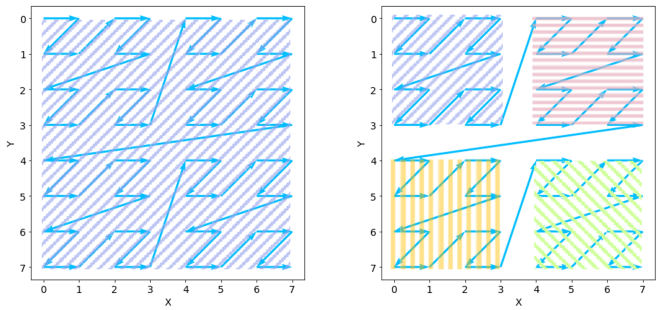

図 12 0 階のクラスター (左) と 1 階のクラスター (右)

範囲検索の矩形の S 点と G 点が同じクラスターにあるとき，距離計算は効率的になるが，異なるクラスターに入るとき計算効率が悪くなることがある．

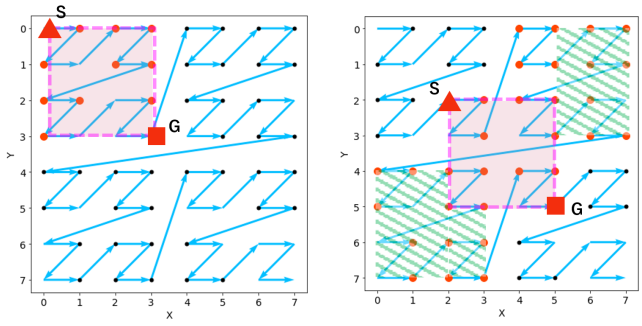

図 13 範囲検索の比較

図 13 の左の図は範囲検索の矩形の S 点と G 点が 1 階の同じクラスターに入るため計算効率が良い．一方で図 13 の右の図は範囲検索の矩形の S 点と G 点が 1 階の異なるクラスターに入るため，範囲検索実行時に別のクラスターを参照し，計算効率が悪くなる．範囲検索の矩形の S 点と G 点が異なるクラスターに入ることで計算効率が悪くなることを避けるため，範囲検索の矩形をいくつかの小さな別の矩形に分割して複数の S 点と G 点の組を再度計算し，同じクラスターに新しく計算した矩形の S 点，G 点を入れることで距離計算の計算コストを減らすことを考える．

すなわち，範囲検索領域が図 11 のようであったとき，S 点，G 点は 1 階のクラスターで別のクラスターに入っている (S 点は青の斜線のクラスター，G 点は黄緑の斜線のクラスター)．S 点から空間充填曲線上の格子点を走査して，別の赤いクラスターに入る直前の格子点 (3, 3) を新しい矩形の G 点とすることで，この S 点から (3, 3) までの新しい矩形は青の斜線のクラスターに入る．同様にして，(4,0) から (7,3) までの赤の横線のクラスターに入る矩形，(0,4) から (3,7) までの黄色の縦線のクラスターに入る矩形，(4,4) から (7,7) までの黄緑の斜線に入る矩形をとり，これら 4 つの矩形を新しい範囲検索領域とすると，各々の新しい矩形で S 点から G 点を走査するとき，矩形外部のトレーニングデータを参照しないため，計算効率が良い (図 14)．

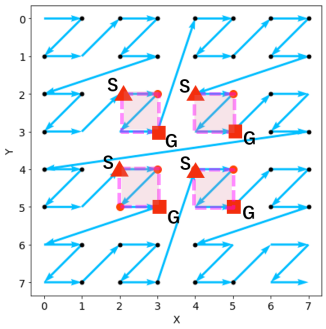

図 14 計算効率の良い範囲検索領域

また実装上では，範囲検索領域について各次元方向で，どこで分割するかは次のように計算する．

S 点と G 点の座標表記を (s_0, s_1)，(g_0, g_1) とする．s_i と g_i $(i = 0, 1)$ の二進表記の値を上の桁から順番に比較する．$s_i < g_i$ より，二進表記の値は必ずどこかで違う値となる．初めて異なる値をとった値に対応する元の多次元空間上での位置で，範囲検索領域は最も低階での別のクラスターに入るため，ここで範囲検索領域を分割する．この低階のクラスターで分割することで，矩形外部のトレーニングデータへの参照を減らしつつ，新しくできる矩形の数を最小限に抑えることができる．また範囲検索領域を分割する位置は s_i と $g_i (i = 0, 1)$ の排他的論理和を求めて，これの最上位ビットを計算することでも求められ，この計算はベクトル化が可能である．

4. Z-curve を用いた k-NN のベクトル化

本章では k 近傍法の計算のベクトル化のために行った実装について紹介する．

4.1 学習ステップ：トレーニングデータの Z 値の計算

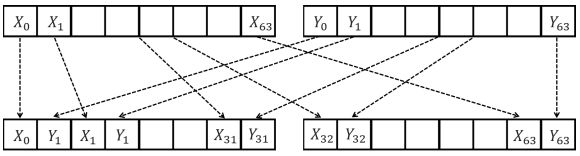

図 15 トレーニングデータの Z 値の計算

上の例は 2 次元 64bit のデータの Z 値の計算を示している．2 次元 64bit の多次元データを 2 進表記で 128bit の 1 次元の Z 値として表現するために 64bit の値を 2 つ用いている．トレーニングデータの Z 値としては，この場合 2 つの配列を用いることになる．また Z-curve 上での順番を知るためにソートを行う際は，基数ソートを用いている．基数ソートはベクトル化可能である [3].

3 章 1 節で紹介したように本実装における k-NN は前処理としてトレーニングデータを Z 値に変換して Z-curve 上での順番を計算するのであった．このとき，Z 値は bit 演算のみによって計算することができるため，十分な量のトレーニングデータがあればベクトル化をすることで計算の高速化ができる．また Z-curve 上での順番を決めるために Z 値によってトレーニングデータをソートするが，このときのソートは基数ソートによって実行している (図 15).

4.2 推論ステップ：範囲検索領域計算のベクトル化

クエリデータについて範囲検索領域を計算する際，個々のクエリデータについてベクトル化をしても一斉に処理できる量が少ない（ベクトル長が短い）ため，計算効率がほとんど良くならない．そのため，ここでは複数のクエリデータについてベクトル化をし計算の高速化をしている．3 章 1 節で紹介した範囲検索領域を決定するための計算はベクトル化可能で，トレーニングデータ同様に複数のクエリデータについて Z 値を求め Z-curve 上でソートされているトレーニングデータと比較して，複数のクエリデータについて同時に近傍球半径を決定し，この複数のクエリデータに紐づいた近傍球半径を用いて範囲検索領域を高速に計算することができる (図 16).

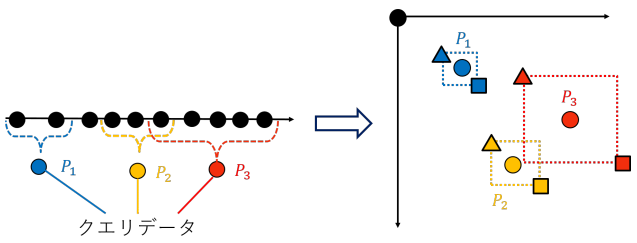

図 16 範囲検索領域計算のベクトル化

4.3 推論ステップ：距離計算のベクトル化

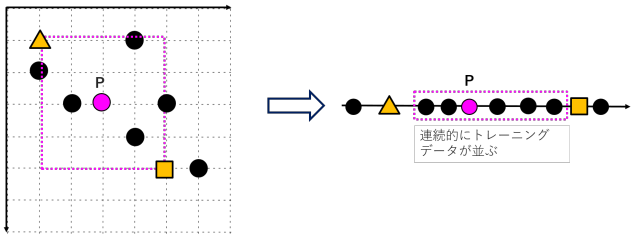

図 17 距離計算のベクトル化

最後に距離計算のベクトル化を説明する．4 章 2 節ではクエリデータ方向でベクトル化を行ったが，本節では個々のクエリデータについてトレーニングデータ方向にベクトル化を行う．範囲検索実行時にクエリデータ毎に決定する矩形を求めた後，矩形の頂点によってクエリデータの k 近傍の候補となるトレーニングデータが決定する．このとき k 近傍の候補となるトレーニングデータは，Z-curve 上で連続的に並んでいるためクエリデータとトレーニングデータの距離計算ステップにおいてベクトル化が可能となる (図 17).

トレーニングデータ方向にベクトル化をする理由は，データの次元の小さいときはベクトル長が短くなるものの，k 近傍の解の候補点であるトレーニングデータの数が少なく計算は十分に速くなり，データの次元が高くなるとクエリデータに対する k 近傍の解の候補点であるトレーニングデータの数が十分多くベクトル長が十分に長くなり，計算が高速化されるためである．

5. 評価

本章では，NEC で開発している並列分散処理ミドルウェア Frovedis [4] において提案手法によって実装した k-NN の性能評価について説明する．比較対象は，Python のオープンソース機械学習ライブラリ scikit-learn [5] に含まれる KNeighborsClassifier とする．Frovedis は SX-Aurora TSUBASA を用いた高速な処理を scikit-learn に類似したインターフェースで利用可能なミドルウェアであり，その内部は C++ および MPI を用いて実装されている．提案手法を用いた k-NN も Frovedis 内の機械学習ライブラリのひとつとして実装している．SX-Aurora TSUBASA は x86 CPU として Intel Xeon を，ベクトルプロセッサを内蔵したアクセラレータとしてベクトルエンジン（Vector Engine，以下 VE）を搭載しており，Frovedis も x86 CPU と VE の両アーキテクチャに対応している．計算時間の評価には人工データと実データの両方を用意した．人工データはデータ数が 100 万点のものと 200 万点のものを用意した．データの次元は 2 次元から 6 次元で一様乱数によって生成した非負整数のデータであり，データの値域は [0,10000] である．実データは kaggle のコンペティション

である「New York City Taxi Trip Duration」のものを用いている．但し，特徴量のうち用いるのはタクシーの乗車位置の緯度と経度，降車位置の緯度と経度の 4 つである．またデータ数は 145 万である．今回の評価ではトレーニングデータとクエリデータに同じものを使った k-NN グラフの計算を行い，近傍点の数は 3 とした．計算にあたって使用するコアの数は x86 CPU が 12 で VE は 8 である．

5.1 評価 1：100 万点の人工データを使った brute force と提案手法の比較

はじめに，素朴な実装による k-NN(VE, brute force) と提案手法 (VE) の計算時間の比較を行う．素朴な実装とは近傍を求めたいクエリデータに対して，すべてのトレーニングデータとの距離を計算し，その距離を比較することで近傍点を求める方法である．素朴な実装 (brute force) による k-NN は 2 次元のとき 631.01[sec]，3 次元のとき 648.63[sec]，4 次元のとき 630.25[sec]，5 次元のとき 618.06[sec]，6 次元のとき 630.75[sec] で計算が終わり，どの次元のデータについても約 600[sec] の計算時間がかかる．一方で，提案手法はデータが 2 次元のとき 1.24[sec]，3 次元のとき 1.43[sec]，4 次元のとき 1.84[sec]，5 次元のとき 3.05[sec]，6 次元のとき 6.72[sec] で計算が終了し，次元数が上がるごとに計算時間が増えるが素朴な実装の k-NN に比べて約 100 倍から約 600 倍高速に計算を高速化できていることが分かる (表 2)．

表 2 brute force(VE) と提案手法 (VE) の計算時間 [sec] の比較

	2	3	4	5	6
brute force (VE)	631.01	648.63	630.25	618.06	630.75
提案手法 (VE)	1.24	1.43	1.84	3.05	6.72

5.2 評価 2：100 万点の人工データを使った scikit-learn と提案手法の比較

次にデータ数が 100 万，次元数が 2 から 6 のときの提案手法 (x86, VE) と scikit-learn(kd-tree, x86) の計算時間の比較を行い，評価結果を表 3 と図 18 に示す．scikit-learn に実装されている k-NN にはいくつかの計算手法のオプションがあるが，今回行った評価では最も高速に計算ができた kd-tree を明示的に指定した．

100 万点の人工データを使った評価では scikit-learn(x86) は 2 次元のとき 3.37[sec]，3 次元のとき 5.91[sec]，4 次元のとき 13.22[sec]，5 次元のとき 21.89[sec]，6 次元のとき 44.77[sec] で計算が終了した．一方で，提案手法 (x86) は 2 次元のとき 1.33[sec]，3 次元のとき 2.45[sec]，4 次元のとき 4.16[sec]，5 次元のとき 9.50[sec]，6 次元のとき 23.8[sec] で計算が終了しており，どの次元についても scikit-learn の約 2 倍から 3 倍の速度で計算を実行できている．そして提案手法 (VE) は 2 次元のとき 1.24[sec]，3 次元のとき 1.43[sec]，4 次元のとき 1.84[sec]，5 次元のとき 3.05[sec]，6 次元のとき 6.72[sec] で計算が終了しており，scikit-learn(x86) の計算速度と比べると 2 次元のときは約 3 倍の速さであるが，次元が上がるほどに計算時間の差が大きくなり，6 次元のときは約 7 倍の速さで計算ができている．

また提案手法の x86 と VE による計算時間を比べると，2 次元，3 次元のときはほとんど計算時間の差がないことが分かる．これは範囲検索によってクエリデータに対する k 近傍点となるトレーニングデータが十分に絞られており距離計算ステップにおいてベクトル長が短いことが理由である．ベクトル長は短いが，k 近傍点の候補となるトレーニングデータが十分に絞られているため計算は速い．一方で次元数が上がると k 近傍点の候補となるトレーニングデータが増えるためにベクトル化によって計算の高速化が可能となり計算時間に差が出る．

表 3 人工データが 100 万点のときの計算時間 [sec] の比較

	2	3	4	5	6
scikit-learn(x86)	3.37	5.91	13.22	21.89	44.77
提案手法 (x86)	1.33	2.45	4.16	9.50	23.8
提案手法 (VE)	1.24	1.43	1.84	3.05	6.72

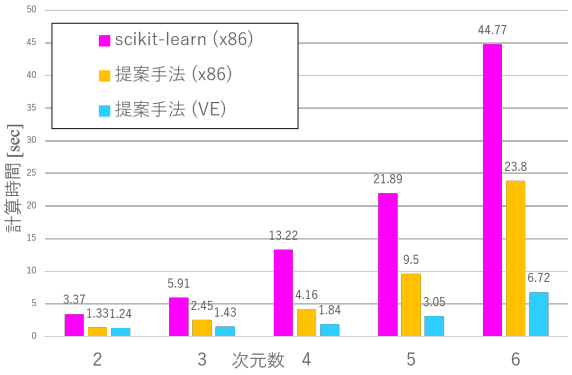

図 18 人工データが 100 万点のときの計算時間 [sec] の比較

5.3 評価 3：200 万点の人工データを使った scikit-learn と提案手法の比較

次にデータ数が 200 万，次元数が 2 から 6 のときの提案手法 (x86, VE) と scikit-learn(x86) の計算時間の比較を行い，評価結果を表 4 と図 19 に示す．

200 万点の人工データを使った評価では scikit-learn(x86) は 2 次元のとき 9.06[sec]，3 次元のとき 13.31[sec]，4 次元のとき 25.29[sec]，5 次元のとき 50.34[sec]，6 次元のとき 98.85[sec] で計算が終了した．一方で，提案手法 (x86) は 2 次元のとき 2.75[sec]，3 次元のとき 5.38[sec]，4 次元のとき 9.49[sec]，5 次元のとき 21.14[sec]，6 次元のとき 51.6[sec] で計算が終了しており，どの次元についても scikit-learn の約 2 倍から 3 倍の速度で計算を実行できている．そして提案手法 (VE) は 2 次元のとき 2.45[sec]，3 次元のとき 2.84[sec]，4 次元のとき 3.76[sec]，5 次元のとき 6.21[sec]，6 次元のと

き 13.38[sec] で計算が終了しており，scikit-learn(x86) の計算速度と比べると 100 万点のデータのときと同様に 2 次元のときは約 3 倍の速さであるが，次元が上がるほどに計算時間の差が大きくなり，6 次元のときは約 7 倍の速さで計算ができている．

表 4 人工データが 200 万点のときの計算時間 [sec] の比較

	2	3	4	5	6
scikit-learn(x86)	9.06	13.31	25.29	50.34	98.85
提案手法 (x86)	2.75	5.38	9.49	21.14	51.6
提案手法 (VE)	2.45	2.84	3.76	6.21	13.38

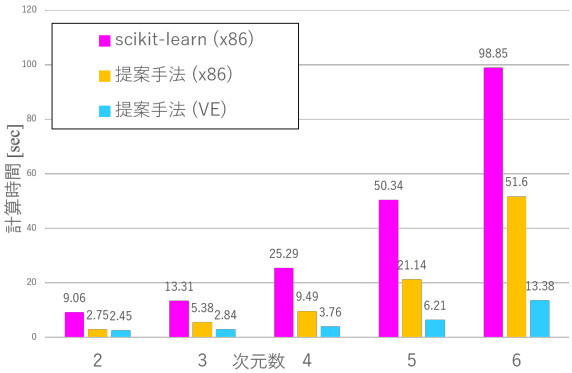

図 19 人工データが 200 万点のときの計算時間 [sec] の比較

5.4 評価 4：実データを使った scikit-learn と提案手法の比較

表 5 実データを用いたときの計算時間 [sec] の比較

	scikit-learn(x86)	提案手法 (x86)	提案手法 (VE)
計算時間	18.76	6.89	3.39

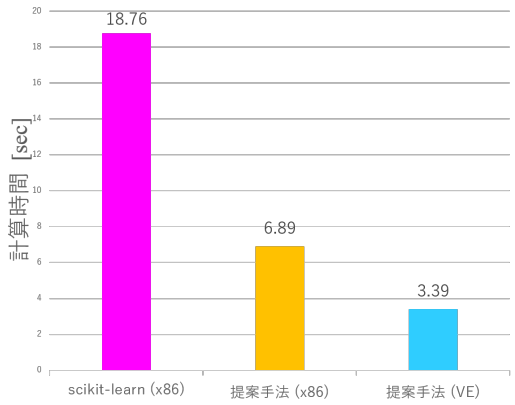

図 20 実データを用いたときの計算時間 [sec] の比較

最後に実データを用いて提案手法 (x86, VE) と scikit-learn(x86) の計算時間の比較を行う．データ数が 145 万で次元数が 4 のデータについて評価を行った結果を以下の表 5 と図 20 に示す．

実データを用いた評価では scikit-learn(x86) は 18.76[sec]，提案手法 (x86) は 6.89[sec]，提案手法 (VE) は 3.39[sec] で計算が終了した．人口データを使ったときと同様に提案手法 (x86) は scikit-learn に比べて計算速度が約 3 倍であって，提案手法 (VE) は scikit-learn(x86) に比べて約 6 倍の計算時間となった．

6. 関連研究

kNN の並列処理による高速化に関する研究には，Zhang ら [6] や Maillo ら [7] によるものや，Kuang ら [8] によるものなどがある．[6] ではデータを複数のサーバに分散し，並列して処理することで計算の高速化を実現する Hadoop を使う際に負荷分散が均一になるように Z 値を用いている．これによって厳密な k-NN および，近似的な k-NN を高速に計算する手法を実装している．また [7] では Hadoop での処理が難しいとされる複数のステップでデータを共有するアプリケーションを想定した k-NN においても高速な分散処理を行える Spark を用いた k-NN を実装している．そして [8] では GPU を利用した汎用並列コンピューティング プラットフォーム CUDA を用いて k-NN の距離計算ステップの高速化を行っている．CUDA を用いることでローエンドの GPU であっても CPU の約 90 倍の速度で brute-force の k-NN の計算ができたことを報告している．

また範囲検索を効率よく行うことで高速化を行う研究は多くある．木構造を用いる手法では，Hou ら [9] や Jakob ら [10] といったものがあり，空間充填曲線を用いる手法では Connor ら [11] や Jin [12] といった研究がある．

[9] では scikit-learn の KNeighborsClassifier でも使われている kd-tree による木構造でデータを持つことで，k-NN の高速な計算を実装している．[10] では Z-curve を用いてデータを木構造で持ち，k 近傍点の候補点を減らして計算の高速化をする方法を提案している．

[11] では，Z-curve を用いて範囲検索を行い，k 近傍点の候補点を減らした後，再帰的処理によって候補点を更に減らして計算の高速化をする方法を提案している．[12] では，Z-curve よりもデータの局所的な位置情報の保持に優れた Hilbert-curve を用いた方法が紹介されている．Hilbert-curve による範囲検索領域の計算を行ったのち，Hilbert-curve を回転およびシフトをして，再び範囲検索領域の計算を行う．その後，回転，シフト，及びその両方を行った範囲検索領域を比較して最も効率良く計算を行える範囲検索領域を選択し，その範囲検索を実行する．

そして Bates [13] は範囲検索領域の分割を提案しており，こちらは空間充填曲線上の点を走査し，トレーニングデータが矩形に外部に一定回数出ていれば矩形を分割する手法を提案している

提案手法では主に [11] と [13] を参考にアルゴリズムの実装を行っている．[11] では k 近傍法の計算の高速化のた

め，再帰的な処理を用いて距離計算の計算コストを減らしている．また [13] では範囲検索領域を小さくするために再帰的な処理を採用している．一方，提案手法はベクトルプロセッサによる計算の高速化を行うために再帰的な手法を用いずに距離計算の計算コストを減らし，ベクトル長を確保する実装を行った．

7. まとめ

本稿では，SX-Aurora TSUBASA が搭載するベクトルエンジンを用いて Z-curve を用いた k-NN を高速に計算する手法を提案した．アルゴリズムの比較としては，提案手法 (VE) は brute force(VE) に比べて約 100 倍から約 600 倍程度高速に計算できることを確認した．また機械学習ライブラリ scikit-learn(kd-tree, x86) と比較すると，提案手法 (x86) は約 2 倍から約 3 倍の速さで計算ができることを確認した．そして，提案手法 (VE) で計算を行った場合 scikit-learn(kd-tree, x86) に比べて約 3 倍から約 7 倍の速さで計算ができることを確認した．

謝辞

本研究の一部は，文部科学省「次世代領域研究開発」(高性能汎用計算機高度利用事業費補助金)「量子アニーリングアシスト型次世代スーパーコンピューティング基盤の開発」の補助を受けて実施している．

参考文献

[1] Foster Provost, Tom Fawcett (竹田 正和 (監訳) (翻訳), 古畠 敦 (翻訳), 瀬戸山 雅人 (翻訳), 大木 嘉人 (翻訳), 藤野 賢祐 (翻訳), 宗定 洋平 (翻訳), 西谷 雅史 (翻訳), 砂子 一徳 (翻訳), 市川 正和 (翻訳), 佐藤 正士 (翻訳)). 戦略的データサイエンス入門, オライリージャパン, 2014.

[2] Toby Segaran (當山 仁健 (翻訳)，鴨澤 眞夫 (翻訳)). 集合知プログラミング, オライリージャパン, 2008.

[3] M. Zhaga, G. E. Blelloch. Radix sort for vector multiprocessors. In Proceedings of the SuperComputing Conference, 1991.

[4] T. Araki. Accelerating Machine Learning on Sparse Datasets with a Distributed Memory Vector Architecture. In 2017 16th International Symposium on Parallel and Distributed Computing, 2017.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 2011.

[6] C. Zhang, F. Li, J. Jestes. Efficient parallel knn joins for large data in mapreduce, in: Proceedings of the 15th International Conference on Extending Database Technology, EDBT ’12, ACM, New York, NY, USA, 2012.

[7] J. Maillo, S. Ramírez, I. Triguero, F. Herrera. kNN-IS: An Iterative Sparkbased design of the k-Nearest Neighbors classifier for big data, Knowledge Based Systems, 2017.

[8] Q. Kuang, L. Zhao. A practical GPU based kNN algorithm. In: Proc. of the Second Symposium International Computer Science and Computational Technology Academy Publisher, 2009.

[9] W. Hou, D. Li, C. Xu, H. Zhang, T. Li. An Advanced k Nearest Neighbor Classification Algorithm Based on KD-tree. IEEE International Conference of Safety Produce Informatization, 2018.

[10] J. Jakob, M. Guthe. Optimizing LBVH-Construction and Hierarchy-Traversal to accelerate kNN Queries on Point Clouds using the GPU, COMPUTER GRAPHICS forum, 2021.

[11] M. Connor, P. Kumar. Fast construction of k-nearest neighbor graphs for point clouds, IEEE Transactions on Visualization and Computer Graphics, 2010.

[12] Y. Jin. Efficient Range Query Using Multiple Hilbert Curves, Current Trends and Challenges in RFID, 2011,

[13] D. Bates. COTS embedded database solving dynamic points-of-interest, A Raima Inc. Technical Whitepaper, September, 2008.