DEIM Forum 2023 2a-4

# 距離結合問合せ高速化のための学習型空間パーティショニングの拡張

室崎　佑紀† 　堀内　美聡† 　佐々木勇和† 　天方　大地† 　鬼塚　真†

† 大阪大学大学院情報科学研究科　〒 565–0871 大阪府吹田市山田丘 1-5

E-mail: †{murosaki.yuki,horiuchi.misato,sasaki,amagata.daichi,onizuka}@ist.osaka-u.ac.jp

**あらまし**　距離結合問合せは，各空間データから任意の距離内に存在する空間データ点を検索する空間問合せである．空間問合せを並列分散処理する場合，空間パーティショニングによりデータセットを分割することで，より効率的な処理が可能である．既存の学習型空間パーティショニングを用いることにより，距離結合問合せの高速化が可能である．しかし，パーティションの探索空間となる境界線候補の集合 (グリッド) はデータ分布に依らず均等幅であるため，データ分布に適したパーティションを構築できない．本稿では，既存の学習型空間パーティショニング技術を拡張することで，空間データ分布に基づいたパーティションの構築を実現する探索空間最適化を提案する．提案手法では，KDB-tree を用いたグリッドの生成を行い，各線をその近傍のより空間データが少ない箇所に移動する．これにより，距離結合問合せをより高速化可能なパーティション構築を実現する．評価実験では，提案手法で構築したパーティションが既存手法で構築したものよりも，並列分散処理における距離結合問合せを最大で 17% 高速に実行可能であることを示す．

**キーワード**　深層強化学習，分散並列処理，空間データ処理，システム最適化，空間データパーティショニング

## 1 序　　論

距離結合問合せは，各空間データ点から任意の距離内に存在する空間データ点を検索する空間問合せである．近年では，ロケーションインテリジェンスという形で注目されることが増えており，出店・販売計画，土木・災害対策などで幅広く利用されている技術である．例えば，新規店舗出店時の商圏分析や，店舗の近くを通り過ぎたユーザへのクーポンの通知などに用いられる．収集される空間データは年々増加しており，より効率的な空間問合せが求められている．

並列分散処理は，大規模なデータセットに対して問合せを高速に処理する技術の一つである．空間問合せの並列分散処理の際は空間データパーティショニングによりデータセットを分割する．一般的に，距離結合問合せの際は互いに空間上の距離が近い空間データ点を同じパーティション領域に分割する．計算負荷を均等に分散させるために，空間データ点の数を各パーティション領域にできるだけ均等に割り当てる手法が多い [1–3]．しかし，境界線付近の空間データ点については問合せ処理時にマシン間で通信が発生するため，各マシンの計算負荷を分散するだけでなく，各マシン間で発生する通信量の削減も考慮する必要がある．そのため，理想的な空間データパーティショニングにより構築されるパーティションは，計算負荷分散と通信量削減の 2 点で最適化されていることが望ましい．

学習型空間データパーティショニング [4] は，計算負荷の分散と通信量の削減を，データ分布に加え，ワークロードとマシン環境に合わせて最適化した空間データパーティションを強化学習により実現した．学習型空間データパーティショニングでは，空間をパーティションの境界線候補であるグリッドにより離散化し，探索空間として与えることで，現実的な実行時間での探索を実現している．しかし，探索空間を画一的に決定することにより探索空間そのものを制限するという課題がある．これは，探索空間として用いるパーティションの候補線の集合であるグリッドを均等幅で与えることで，探索空間がデータ分布と関係なく空間を離散化したものになるためである．そのため，設定された探索空間が不適切なグリッド線のみで構成される可能性があり，距離結合問合せを高速化可能なパーティションが存在しない可能性がある．距離結合問合せのさらなる高速化を実現するには，データ分布を反映したグリッドを生成することで探索空間そのものを最適化する必要がある．

探索空間最適化は大きく分けて計算負荷分散と通信量削減からなる．まず，計算負荷分散では，KDB-tree を用いたグリッドの初期化を行う．KDB-tree は計算負荷分散を考慮したパーティショニング手法であるため，計算負荷分散を考慮したグリッドの生成が可能である．次に，通信量削減では，グリッドをその近傍のより空間データが少ない箇所に移動する．境界線付近のデータが多いほど問合せ処理時にマシン間での通信が増加するため，境界線付近のデータを減らすことで，通信量の削減を考慮したグリッドの調整が可能である．これら 2 段階からなるグリッドの生成により探索空間を最適化することで，距離結合問合せをより高速化可能なパーティション構築を実現する．

評価実験では，3 種類の実データを用いてパーティショニングを行い，構築されたパーティションによる距離結合問合せの実行時間の評価により提案手法の有用性を示す．提案手法で構築したパーティションが既存手法で構築したものよりも，並列分散処理における距離結合問合せを最大で 17% 高速に実行可能であることを示す．

本稿の構成は次の通りである．2 章にて関連研究について，3 章にて事前知識について，4 章にて探索空間最適化について説明し，5 章にて実験結果を示す．6 章にて本稿をまとめる．

## 2 関連研究

### 2.1 並列分散処理システム

空間問合せの並列分散処理を実現するシステムとして，主に Hadoop ベース [5] または Spark ベース [6] が存在する [7]．本稿では，Spark ベースである GeoSpark (Apache Sedona) [8] を用いて評価を行う．一般的に，並列分散処理システムは 1 台のマスタと複数台のワーカからなり，各ワーカはマスタの指示に従い処理を実行する．マスタはパーティショニング [9] によりデータを分割し，各ワーカは割り当てられたパーティションについて並列分散に処理を実行する．この際，各マシン間でできる限り通信が少なくなるよう，対象のデータにあったパーティショニングを行う必要がある．

### 2.2 空間データパーティショニング

空間データを対象にしたパーティショニングの場合，計算負荷分散の観点から各パーティションに割り当てられる空間データ数の差を小さくし，通信量削減の観点から近しい空間データ点を互いに同じパーティション領域に割り当てる方針を持つ手法が多い．

Uniform Grid は単純なアルゴリズムで，対象の空間を空間的に等しいサイズのセルを用いて分割するものである．しかし，一般的にデータ分布には偏りが存在するため，多くの場合でこの手法は最適でない．データ分布に応じて適切な空間データパーティショニングを行うアルゴリズムに R-Tree [1] Quad-tree [2]，および KDB-tree [3] が存在する．これらのアルゴリズムでは，より効果的な計算負荷の分散を目的とし，各パーティションに属する空間データ数をできる限り等しくする．R-Tree では位置の近いデータ群の最小外接矩形 (Minimum Bounding Rectangle: MBR) を階層的に入れ子になるようなパーティションに分割する．Quad-tree ではパーティションの再帰的な 4 分割を，指定するパーティション数になるまで繰り返す．KDB-tree ではデータ数が最も多いパーティション領域について，指定された任意の方向でデータ数が均等になるように分割することを繰り返す．しかし，これらの手法では，距離結合問合せの内容やパーティションの境界線付近の通信量削減が一切考慮されていない．AQWA [10] は，ワークロードやデータ分布の変化に合わせて動的にパーティションを更新する手法であり，クエリ実行に伴うコストを計算するコスト関数を実現することで高速なパーティショニングを実現している．

学習型空間データパーティショニング [4] は，深層強化学習を導入することで計算負荷分散に加えて距離結合問合せの内容やパーティションの境界線付近の通信量削減のマシン環境に合わせた考慮を可能にした距離結合問合せ高速化アルゴリズムである．強化学習の一種である Q-learning に深層学習を利用した Deep Q Network (DQN) を採用しており，探索空間である環境内で，エージェントが方策をもとに現在の状態における最適な行動を選択，それによって得た報酬をもとに最適な方策を探索することで，ワークロードやマシン環境に合わせたパーティショニングを可能とする．本稿では学習型空間データパーティショニング [4] を拡張する．

### 2.3 空間索引

空間索引は，空間問合せに適した索引の一種であり，各空間データ点の空間的な近さをもとに索引を構築する．空間データが分布する領域を分割し，互いに近い空間データ点に同じラベルを張ることで，空間的に近い空間データに関する情報を保持することが可能である．空間索引は，空間全体をセルに分割する空間起点の空間索引と，空間データそのものをクラスタに分割するデータ起点の空間索引に大別される．空間起点の空間索引には，空間データを追加してもインデックス構造を変更する必要がないという利点がある．先に述べた Uniform Grid や Quad-tree，および KDB-tree に加え，kd-tree [11] がよく用いられる．kd-tree は，$k$ 次元空間を座標軸に垂直な面により分割し，各ノードに一つの点が格納された木構造でデータを保持する．それに対し，データ起点の空間索引には，ストレージの利用効率が上昇し，かつ検索時間も短いという利点がある一方で，データの追加や削除の際にインデックスを修正する必要があるという欠点も存在する．一般的には，先に述べた R-Tree やその亜種が多く用いられる．近年では，機械学習を用いた索引の構築も研究されている．クエリ分布を元にグリッド索引構造の最適化を行う学習型多次元索引 [12] や，R-tree をストレージの利用効率と入出力コストの面から最適化する LISA [13] などが挙げられる．

空間索引は分割された領域の重複が許されているため，空間パーティショニングには直接応用できないものも多いが，Quad-tree など領域の重複がないものは空間パーティショニングに直接利用されている．

## 3 事前知識

本章では，本稿で用いる語句の定義を行った後，強化学習における空間データパーティショニングの問題定義を行う．その後，学習型空間データパーティショニング [4] について説明する．

### 3.1 定　　義

重要な語句の定義と問題定義を行う．

**定義 1 (空間データ点と空間データ)**　空間データ点 $d \in \mathcal{D}$ は二つの変数 $(x, y)$ により与えられる二次元座標における位置情報を含むデータとする．空間データ点の集合を空間データと定義する．本稿では，変数 $x$ は左右に変化する経度の値，$y$ は上下に変化する緯度の値に対応する．

**定義 2 (マップ)**　マップ $\mathcal{M}$ は，すべての空間データ点 $d \in \mathcal{D}$ を覆うように定義される領域である．$\mathcal{M}$ はすべての $(x, y) \in \mathcal{D}$ において，$(x, y) \in \mathcal{M}$ を満たす矩形の領域とする．$\mathcal{M}$ において，$x$ の最小値，$x$ の最大値，$y$ の最小値，$y$ の最大値をそれぞれ $x_{min}, x_{max}, y_{min}, y_{max}$ とする．

**定義 3 (境界線とパーティション，パーティション領域)** 境界線 $p$ は，並列分散処理において各ワーカに空間データを配置する際にマップを空間的に分割する線を意味する．境界線の集合をパーティション $\mathcal{P}$ と定義し，パーティションにより分割された各領域をパーティション領域と定義する．すべてのパーティション領域を結合した領域はマップが定義する領域と等しく，かつ各パーティション領域は互いに重複しない．

パーティションの探索は，以下で定義されるグリッド上で行われる．

**定義 4 (グリッドとグリッド線，グリッドセル)** グリッド線 $g$ は，マップを覆うように設定されるパーティション境界線の候補となる線である．グリッド線の集合をグリッド $\mathcal{G}$ と定義し，グリッド線により分割される領域をグリッドセルと定義する．また，$y$ 軸と平行なグリッド線は左から順に，$x$ 軸と平行なグリッド線は上から順に採番が行われ，それぞれ変数 $i$ と $j$ を用いて表す．また，あるグリッドセルを構成する左側のグリッド線 $i$ と上側のグリッド線 $j$ を用いて，そのグリッドセルの位置を $(i, j)$ により与える．

本稿では，以下で定義される $\epsilon$ 距離結合を対象にパーティショニングを行う．

**定義 5 ($\epsilon$ 距離結合)** $\epsilon$ 距離結合は，空間データに対する問合せである空間問合せの一種である．異なる属性を持つ空間データ点の集合 $\mathcal{T}$ と $\mathcal{U}$ に対して，距離の閾値 $\epsilon$ が与えられたとき，$t \in \mathcal{T}$，$u \in \mathcal{U}$ において，$dist(t, u) \leqq \epsilon$ を満たすオブジェクトペア $(t, u)$ を検索する．

**定義 6 (ワークロード)** ワークロード $\mathcal{W}$ は，クエリとそのクエリがワークロードに占める頻度 $w$ の集合である．$\epsilon$ 距離結合の場合，クエリは距離 $\epsilon$ の値を持つ．

### 3.2 学習型空間データパーティショニング

学習型空間データパーティショニング (LSDP) [4] では，深層強化学習により空間データパーティショニングを解いた．LSDP のフレームワークは，事前学習と本学習，2 つのフェーズからなる．初めに，強化学習における初期条件，状態，行動および報酬を定義する．

**初期条件** 環境をグリッドにより初期化する．グリッドは，マップ全体を覆う長方形のレイヤであり，エージェントはグリッド上で領域の分割を繰り返すことでパーティションを構築する．

**状態** 状態 $s$ は，各グリッドセル $(i, j)$ の状態 $s_{(i,j)}$ の集合として，以下のように定義する．

$$s_{(i,j)} = (h_{(i,j)}, v_{(i,j)}, c_{(i,j)}) \tag{1}$$

ここで，$i, j$ はグリッドセルの左上座標を指定する．$h \in \{0, 1\}$ と $v \in \{0, 1\}$ は，セルの上辺と左辺における $x$ 軸と平行および $y$ 軸と平行な境界線の有無である．$c_{(i,j)}$ はセルとパーティションのデータ数の割合を表し，$c_{(i,j)} = \frac{|\mathcal{D}_{(i,j)}| \cdot |P_{(i,j)}|}{|\mathcal{D}|^2}$ と定義する．このとき，$|\mathcal{D}|$，$|\mathcal{D}_{(i,j)}|$ と $|P_{(i,j)}|$ は，それぞれ空間データ全体，セル $(i, j)$，セル $(i, j)$ をカバーするパーティション領域におけるデータ数である．

**行動** 行動 $a \in \mathcal{A}$ は，グリッド上に新しい境界線を追加する操作として，以下に定義する．

$$a = (i, j, dir) \tag{2}$$

ここで，$i$ と $j$ は行動開始点の座標，$dir \in \{right, down\}$ は行動開始点 $(i, j)$ から追加する境界線の方向，右方向あるいは下方向を指定する．ただし，行動 $a$ は各状態において，パーティション領域を 2 分割するような行動から選択する．

**報酬** 報酬は，各エピソードで構築されたパーティションに対する，ワークロード実行時間を用いた評価値である．この報酬を最大化することは，空間データとワークロードの両方に対して，実行時間を最小化するようなパーティションであることを意味する．具体的には，報酬 $r$ は空間データに対するパーティションの集合 $\mathcal{P}$ を計算環境に適用し，ワークロード内の各クエリ $w$ の実行時間 $C(\mathcal{P}, w)$ に基づき，以下の計算式から導出される．

$$r = \left( \sum_{w_i \in W} f_i \cdot \frac{C(\mathcal{P}_b, w_i)}{C(\mathcal{P}_e, w_i)} \right)^2 \tag{3}$$

ここで，$w_i$，$f_i$，$\mathcal{P}_b$，および $\mathcal{P}_e$ はそれぞれワークロードのクエリ，クエリ $w_i$ の頻度，エピソード終了時のパーティション，および学習中に探索されたワークロードの実行時間が最も短いパーティション (ベストパーティションと呼ぶ) を表す．この計算式は，現在のエピソードでのパーティションとそれまでのベストパーティションの実行時間の間で相対的に評価を行い，実行時間の差を強調するために 2 乗する．$r > 1$ の報酬が得られた場合には $\mathcal{P}_e$ は $\mathcal{P}_b$ より優れていると判断し，次のエピソードでは $\mathcal{P}_b$ が $\mathcal{P}_e$ に更新される．報酬計算に使用する各クエリの実行時間は，誤差軽減のために複数回の試行の中央値を採用する．

次に，学習の流れについて説明する．まず，事前学習では既存アルゴリズムの行動を用いて模倣学習 [14] を行うことで，学習初期における学習過程を大幅に削減する．既存アルゴリズムによりパーティション構築を行う一連の分割行動の集合，遷移をデモデータと呼ぶ．この際，既存アルゴリズムによるパーティションは，各境界線についてそれぞれ最も近いグリッド線の位置で近似される．

**定義 7 (デモデータ)** デモデータは，事前学習で収集される既存手法によるパーティショニングにおける一連の分割行動 $a \in \mathcal{A}$ の遷移である．

次に，本学習では強化学習により，より良いパーティション

の探索を行う．学習は，以下の一連の流れを 1 エピソードとして繰り返し行われる．まず，パーティション数が目標のマシン台数に達するまで，エージェントが $\epsilon$-greedy 法を用いた確率的な行動選択を反復的に行いパーティションを生成する．次に，そのパーティションに従いデータを各マシンに配置，ワークロード実行，その結果から報酬を計算する．これを元に，デモデータを含む過去の経験にここで得られた新たな遷移を加え，モデルの学習を行う．これらの学習過程を規定回数まで反復的に繰り返すことで，最終的なモデルを生成する．

しかし，学習型空間データパーティショニングには，探索空間をデータ分布と関係なく定義してしまうという課題がある．これは，グリッドを均等幅で与えることで，探索空間がデータ分布と関係なく固定されるためである．距離結合問合せのさらなる高速化を実現するには，データ分布を反映したグリッド生成技術により，探索空間を最適化する必要がある．

よって，本稿で取り組む課題を以下のように定義する．

**問題定義 (探索空間最適化)** 空間データ $\mathcal{D}$，ワークロード $\mathcal{W}$，計算環境 *(システムとマシン台数)* が与えられたとき，計算環境上で空間データ $\mathcal{D}$ に対するワークロード $\mathcal{W}$ の実行時間を最小化するパーティションの探索に適したグリッドを生成する．

## 4 探索空間最適化

本章では，提案手法である探索空間の最適化について説明する．

### 4.1 概　　要

我々は，学習型空間データパーティショニングを拡張し，探索空間を最適化することで，距離結合問合せのさらなる高速化が可能なパーティションの探索を実現する．データ分布を基にグリッドを生成することで，探索空間の最適化を行う．

提案手法のフレームワークを図 1 に示す．探索空間最適化は，学習型空間データパーティショニングのグリッド，つまり強化学習の環境を最適化するものとして機能する．探索空間最適化では，空間データを入力とし，探索空間としてグリッドを生成し出力する．ここで，最適なグリッド線とは，高速な距離結合問合せを実行可能なパーティションの探索に適したグリッド線のことである．まず初めに，グリッド候補線を定義する．グリッドは，選択されたグリッド候補線の集合により設定される．

**定義 8 (グリッド候補線)** グリッド候補線 $l \in \mathcal{L}$ は，$y$ 軸と並行，または $x$ 軸と並行にひかれるグリッド線の候補となる線である．$\mathcal{L}$ は，$y$ 軸と並行な $m$ 本のグリッド候補線と，$x$ 軸と並行な $n$ 本のグリッド候補線からなる，マップ全体を均等幅で分割するグリッド候補線の集合である．

グリッドの生成は，(1) グリッド候補線の生成，(2) 計算負荷分散を考慮したグリッド選択，(3) 通信量削減を考慮したグリッド調整，(4) グリッド候補線の間引きの 4 つのフェーズから構成される．まず，グリッド候補線の生成では，指定された細かさで十分に細かく空間を均等幅で離散化し，計算を近似することでアルゴリズムを簡略化する．空間は離散化により，線として圧縮され，各線はグリッド候補線となる．計算負荷分散を考慮したグリッド選択では，KDB-tree を用いて構築されたパーティションを延長することで，グリッド候補線からグリッド線を選択する．生成されるグリッド線は，既存手法の拡張により生成されたグリッド線を，各境界線について最も近いグリッド候補線で近似したものである．通信量を考慮した調整では，各グリッド線をその近傍でより通信量を削減するようなグリッド候補線の位置に移動する．最後に，グリッド線の間引きを行う．互いに近すぎるグリッド線を除去することで，パーティションの探索に影響を与えない範囲で状態空間を削減する．

### 4.2 フレームワーク詳細

本項では，探索空間最適化の各フェーズの詳細について述べる．

**グリッド候補線の生成** グリッド候補線の生成では，グリッド生成を行うために空間を離散化する．最初にデータ全体を覆うマップを定義する．次に，マップを指定された細かさで均等に分割することで，$y$ 軸と平行に $m$ 本，$x$ と平行に $n$ 本のグリッド候補線を生成する．各グリッド候補線は，以下の定義で与えられる境界線スコアを持つ．

**定義 9 (境界線スコア)** グリッド候補線 $l_k \in \mathcal{L}$ の境界線スコア $b(\mathcal{D}, \mathcal{L}, l_k, \alpha)$ は，各グリッド候補線について以下の式で定義する．

$$b(\mathcal{D}, \mathcal{L}, l_k, \alpha) = \frac{\sum_{l \in d_k} |\mathcal{D}(l)|}{|d_k|} - |\mathcal{D}(l_k)| \qquad (4)$$

ここで， $d_k$ は $l_k$ から距離 $\alpha$ 内の $l_k$ に平行なグリッド候補線の集合である．また，$|\mathcal{D}(l)|$ は各グリッド候補線 $l$ について最も近い空間データ点の数である．近さは，各境界線と垂直方向のユークリッド距離により定義し，$x$ 軸と $y$ 軸に平行なグリッド候補線をそれぞれ独立して計算する．

境界線スコアは，あるグリッド候補線が，その近辺の他のグリッド候補線と比べて，どれほど空間データ数の少ない箇所を通っているかを示している．つまり，ただ単にそのグリッド候補線の近傍のデータ数が少ないだけでなく，データ分布の谷となるような箇所を通っているグリッド候補線ほど，高い境界線スコアが与えられる．

**計算負荷分散を考慮したグリッド選択** 計算負荷分散を考慮したグリッド選択では，グリッド候補線の中から計算負荷分散のみを考慮してグリッドを選択する．まず，KDB-tree によりパーティションの構築を行う．次に，そのパーティションの各境界線をマップの端から端まで延長することで，暫定のグリッドを生成する．暫定のグリッドを，各グリッド線についてそれぞれ最も近いグリッド候補線で近似することで，計算負荷分散を考慮したグリッド選択を行う．既存手法は計算負荷分散に重点を置いたパーティショニングを行うため，パーティションの

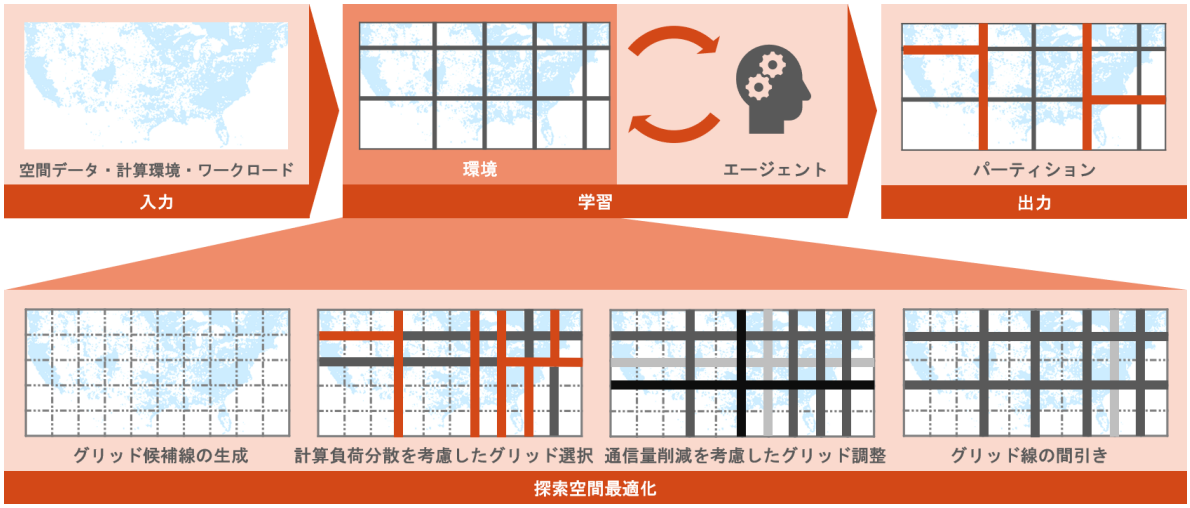



図 1: フレームワーク

各境界線の延長を行うことで，計算負荷分散を考慮したグリッドの生成が可能である．

**通信量削減を考慮したグリッド調整**　通信量削減を考慮したグリッド調整では，計算負荷分散を考慮したグリッド選択で生成したグリッドの各グリッド線について，その近傍のより通信量が少なくなるようなグリッド線の位置に移動する．これは，パーティションの境界線の付近に存在する空間データが多いほど，並列分散処理実行時にマシン間での通信が多くなるためである．対象のグリッド候補線を中心として，距離 $\beta$ 以内に存在するグリッド候補線の中から境界線スコアが最大のグリッド候補線の位置にグリッド線を移動する．複数の候補がある場合は，$x$ 軸に平行なグリッド候補線の場合は上側，$y$ 軸に平行な場合は左側を採用する．

**グリッド線の間引き**　グリッドの間引きでは，互いに近すぎる距離 $\gamma$ 以内に存在するグリッド線について，一方を残して他方を削除する．これには 2 つの理由がある．1 つ目は，グリッド線の本数が増えれば増えるほど探索空間が大きくなり，本質的に探索空間を広げないような線が探索効率を悪化させるためである．2 つ目は，互いに近すぎる選択肢は結果に大きく影響せず，報酬を分散させるだけになるためである．以上の理由から，互いに近すぎる線は削除することが望ましい．比較を行う隣同士のグリッド境界線について，$x$ 軸に平行なグリッド候補線の場合は下側，$y$ 軸に平行な場合は右側を削除する．

### 4.3　アルゴリズム

探索空間最適化の疑似コードをアルゴリズム 1 に示す．まず，グリッド候補線の生成を行う (1 行目)．次に，既存手法によりパーティショニングの構築を行い，その拡張によりグリッド候補線の中から計算負荷分散のみを考慮してグリッド生成する (3–11 行目)．さらに，各グリッド線について通信量の削減を考慮してより通信量が減るような近傍のグリッド候補線の位置にグリッド線を移動する (12–15 行目)．最後に，互いに近すぎるグリッド線の間引きを行う (16–22 行目)．

**Algorithm 1** 探索空間最適化

**Input:** Dataset $\mathcal{D}$, parameters $\alpha, \beta, \gamma$
**Output:** Coordinates of Grid $\mathcal{G}$

1: $\mathcal{L} \leftarrow$ horizontal $m$-lines and vertical $n$-lines that equally devides map $\mathcal{M}$
2: $\mathcal{G}, \mathcal{G}', \mathcal{G}'' \leftarrow \emptyset$
3: $\mathcal{P}_{\mathcal{KDB}} \leftarrow$ partition constructed by KDB-tree
4: **foreach** $p \in \mathcal{P}_{\mathcal{KDB}}$ **do**
5:　　**if** $p$ is $y$-axis parallel **then**
6:　　$g \leftarrow p$ extended from $y_{min}$ to $y_{max}$
7:　　**else**
8:　　$g \leftarrow p$ extended from $x_{min}$ to $x_{max}$
9:　　**end if**
10:　　$\mathcal{G}' \leftarrow \mathcal{G}' \cup argmin_{l \in L} dist(l, g)$
11: **end for**
12: **foreach** $g \in \mathcal{G}'$ **do**
13:　　$\mathcal{L}_t \leftarrow \{l \in \mathcal{L} | dist(l, g) < \beta\}$
14:　　$\mathcal{G}'' \leftarrow \mathcal{G}'' \cup argmax_{l \in \mathcal{L}_t} b(\mathcal{D}, \mathcal{L}, l, \alpha)$
15: **end for**
16: $g_t \leftarrow \emptyset$
17: **foreach** $g \in \mathcal{G}''$ **do**
18:　　**if** $dist(g, g_t) > \gamma$ **then**
19:　　$\mathcal{G} \leftarrow \mathcal{G} \cup g$
20:　　$g_t \leftarrow g$
21:　　**end if**
22: **end for**
23: Return $\mathcal{G}$

## 5　評価実験

本章では，提案手法の有用性を評価するため，空間データとして 3 種類のデータセットを使用して，距離結合問合せの実行時間と学習時間について，提案手法を既存手法と比較する．

### 5.1　実験設定

実験設定として，実験環境，データセット，ワークロード，及びパラメータについて述べる．

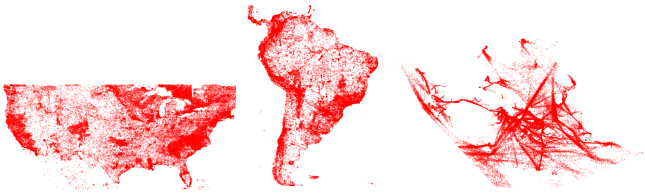
#### 5.1.1　実験環境

並列分散処理環境には Intel Celeron G4930T CPU

@3.00GHz，32GB の RAM を搭載したコンピュータ 9 台を利用する．並列分散処理には Spark ベースである GeoSpark (Apache Sedona) [8] を用いて，Spark クラスタとして 1 台のマスターノードと 8 台のワーカを配置する．

**5.1.2** データセット

実験では Open Street Maps (OSM) と Imis-3months(Imis) を利用する [15]．OSM は，湖，森林，建物，道路を緯度経度で示す地形データセットであり，米国 (OSM-US)，南米 (OSM-SA) を実験に用いる．Imis は，船舶情報を緯度と経度で表したものである．各データセットについてそれぞれ 10 万レコードの点データをサンプリングする．図 2 にデータセットのデータ分布を示す．また，表 1 にデータセットの分布する領域であるマップの範囲を示す．$x_{min}, x_{max}, y_{min}$, および $y_{max}$ は，それぞれ経度の最小値，経度の最大値，緯度の最小値，および緯度の最大値に対応する．



(a) OSMUS (b) OSM-SA (c) Imis

図 2: 空間データのデータ分布

表 1: データセットが分布する領域

| dataset | $x_{min}$ | $x_{max}$ | $y_{min}$ | $y_{max}$ |
|---|---|---|---|---|
| OSM-US | -124.8 | -66.88 | 24.43 | 49.38 |
| OSM-EU | -24.50 | 49.99 | 35.00 | 69.99 |
| Imis | 19.16 | 29.61 | 34.59 | 40.99 |

**5.1.3** ワークロード

ワークロードは 3 種類の異なる閾値 $\epsilon$ を持つ距離結合問合せを用いて構成し，各クエリは閾値 $\epsilon$ とワークロードに占めるそのクエリの頻度 $w$ からなる．ただし，データセット毎にデータ位置の範囲が異なるためそれぞれのデータセットに合わせて距離の閾値を調整する．また，ワークロードに含まれるクエリの頻度のバイアス (skew) が小さい場合 (small skew) と大きい場合 (large skew)，バランス (balance skew) の 3 通りを想定した検証を行う．ここで，バランスとは，実行時間とバイアスを掛けた値が各クエリで等しくなるようなバイアスを意味する．各データセットの各ワークロード $\mathcal{W}$ について，各クエリの距離結合問合せの距離 $\epsilon$ と頻度 $w$ を用いて $\mathcal{W} = \{(\epsilon_1, w_1), (\epsilon_2, w_2), (\epsilon_3, w_3)\}$ として示す．small skew について，OSM-US と OSM-SA では $\mathcal{W} = \{(500[m], 0.25), (1000[m], 0.50), (5000[m], 0.25)\}$, Imis では $\mathcal{W} = \{(50[m], 0.25), (100[m], 0.50), (500[m], 0.25)\}$ とする．large skew について，OSM-US と OSM-SA では $\mathcal{W} = \{(500[m], 0.02), (1000[m], 0.03), (5000[m], 0.95)\}$, Imis では $\mathcal{W} = \{(50[m], 0.02), (100[m], 0.03), (500[m], 0.95)\}$ とする．balance skew について，OSM-US では $\mathcal{W} = \{(500[m], 0.40), (1000[m], 0.40), (5000[m], 0.20)\}$, OSM-SA では $\mathcal{W} = \{(500[m], 0.45), (1000[m], 0.40), (5000[m], 0.15)\}$, Imis では $\mathcal{W} = \{(50[m], 0.43), (100[m], 0.43), (500[m], 0.14)\}$ とする．

**5.1.4** パラメータ設定

グリッド候補線の本数 $(m, n)$ は，デフォルト値を $(m, n) = (20000, 10000)$ とする．計算負荷分散を考慮したグリッド選択に用いる KDB-tree のパーティション領域の数を 128 とする．$\alpha$ を 0.03 [度] とする．$\beta$ を 0.5 [度] とする．$\gamma$ を 0.3 [度] とする．

**5.1.5** ベースライン

ベースラインには，学習型空間データパーティショニング (LSDP) [4] に加え，Uniform grid [1]，Quad tree [2]，KDB-tree [3] の 4 つを用いる．また，学習型空間データパーティショニング，提案手法ともに，KDB-tree によるパーティショニングを手本とする模倣学習を行ったモデルによるパーティショニング (Demo) についても評価を行う．

## 5.2 実験結果

本節で提案手法が構築するパーティションの性能の評価実験結果を示す．

**5.2.1** ワークロードの高速化

本項では，5.1.2 項に示した各データセットを用いて評価を行う．実験結果を表 2 に示す．idx は空間索引利用の有無を表し，w/o は利用しなかった場合，w/ は利用した場合を示す．また，skew は頻度のバイアスを示す．

まず，空間索引を用いない場合の結果について述べる．すべての場合で，既存手法と比較して提案手法がより高速な距離結合問合せを実現できていることがわかる．各距離別においても，ほぼすべての場合，すべての距離で提案手法が高速化を達成している．以上のことから，提案手法はワークロード内のクエリを満遍なく高速化可能であることがわかる．また，OSM-US の small skew では，学習型空間データパーティショニングが KDB-tree に対して 2.48% の高速化であるのに対し，提案手法は 14.5% の高速化を実現しており，学習型空間データパーティショニングと比べて高速化の度合いが非常に大きいことがわかる．次に，Demo を KDB-tree を比較する．提案手法の Demo は，KDB-tree について通信量の削減のみを行ったパーティションとほぼ同義であるため，これらを比較することで通信量削減を考慮したグリッド調整の有効性を評価することが可能である．学習型空間データパーティショニングの Demo は値が悪化しているのに対し，提案手法の Demo は最大で 34.8% の高速化を実現しており，通信量の削減が有効に機能していることがわかる．また，OSM-US では，すべての場合で学習型空間データパーティショニングにより作成されたパーティションよりも提案手法の Demo のほうが効果的なパーティションを構築していることがわかり，動的な学習を行わず，通信量の削減のみを行った場合でも一定の成果を望めることがわかる．

次に，空間索引を用いた場合の結果について述べる．すべての場合で，提案手法が学習型空間データパーティショニングを除く既存手法よりも優れていることがわかる．また，OSM-US では学習型空間データパーティショニングと比べて高速化を達

表 2: 距離結合問合せの実行時間

| data | idx | skew | Uniform | Quad | KDB | Demo | SDP | Demo | Ours |
|---|---|---|---|---|---|---|---|---|---|
| OSM -US | w/o | small | 11.22 | 8.93 | 4.67 | 4.90 | 4.55 | 3.96 | **3.94** |
| | | large | 15.83 | 13.29 | 8.65 | 8.94 | 6.70 | 6.37 | **5.93** |
| | | balance | 10.78 | 8.53 | 4.39 | 4.51 | 3.91 | **3.85** | 3.88 |
| | w/ | small | 3.35 | 3.82 | 2.99 | 2.99 | 1.78 | 2.22 | **1.75** |
| | | large | 7.93 | 7.67 | 7.32 | 7.08 | 3.57 | 4.61 | **2.97** |
| OSM -SA | w/o | small | 11.72 | 11.91 | 6.92 | 7.22 | 6.89 | 7.00 | **6.46** |
| | | large | 17.82 | 18.21 | 13.21 | 13.37 | 13.60 | 13.48 | **12.54** |
| | | balance | 10.52 | 10.55 | 6.17 | 6.46 | 5.71 | 6.27 | **5.46** |
| | w/ | small | 4.66 | 5.20 | 4.86 | 5.08 | **3.51** | 5.03 | 3.57 |
| | | large | 11.29 | 11.12 | 11.19 | 11.97 | **8.34** | 12.03 | 9.06 |
| Imis | w/o | small | 16.74 | 12.69 | 8.77 | 8.93 | 9.49 | **7.08** | 8.72 |
| | | large | 22.24 | 19.81 | 17.43 | 17.49 | **15.57** | 16.89 | 16.95 |
| | | balance | 16.01 | 11.45 | 7.36 | 7.76 | 7.53 | 7.47 | **6.10** |
| | w/ | small | 4.75 | 5.02 | 7.11 | 7.11 | 3.60 | 7.02 | **3.51** |
| | | large | 11.18 | 11.84 | 16.21 | 15.87 | 7.83 | 15.84 | **6.94** |

表 3: 学習時間 [h]

| data | idx | skew | LSDP | | Ours | |
|---|---|---|---|---|---|---|
| | | | total | run time | total | run time |
| OSM -US | w/o | small | 39.28 | 35.34 | 42.42 | 37.02 |
| | | large | 42.55 | 37.07 | 42.68 | 37.20 |
| | | balance | 37.52 | 33.51 | 42.77 | 37.23 |
| | w/ | small | 13.13 | 9.216 | 14.50 | 9.22 |
| | | large | 13.02 | 9.18 | 14.44 | 9.23 |
| OSM -SA | w/o | small | 46.54 | 43.03 | 48.89 | 39.69 |
| | | large | 44.84 | 41.30 | 49.14 | 39.88 |
| | | balance | 34.75 | 31.29 | 49.50 | 40.39 |
| | w/ | small | 13.40 | 9.94 | 19.60 | 10.69 |
| | | large | 15.95 | 12.45 | 21.11 | 12.06 |
| Imis | w/o | small | 58.10 | 54.50 | 53.65 | 44.83 |
| | | large | 64.59 | 60.99 | 60.20 | 51.67 |
| | | balance | 58.46 | 54.83 | 51.62 | 42.78 |
| | w/ | small | 15.73 | 12.19 | 19.30 | 10.68 |
| | | large | 16.78 | 13.29 | 20.47 | 19.93 |

成している．一方，OSM-SA では学習型空間データパーティショニングよりも高速化の度合いが小さい．これは，構築した各グリッド線が，学習型空間データパーティショニングで用いるグリッド線と比べてデータ数の多い箇所を通っていることが要因の一つである可能性がある． 空間索引を用いた場合，空間索引を用いなかった場合よりも相対的に計算性能が高く通信性能が悪くなるため，より通信量削減に特化したグリッドを生成するようなパラメータ設定を行うことで高速化を行うことが考えられる．

**5. 2. 2** 学習時間の比較

表 3 に，学習に要した時間を示す．total に学習に要した合計時間を示し，run time に total のうちパーティションの評価のために行ったクエリ実行の合計時間を示す．また，表 3 をもとに計算した，LSDP と比較したときの提案手法の学習時間の平均増加率は，total が 13.19%，run time が 1.45% となった．

まず，run time について LSDP と提案手法を比較する．局所的には Imis の 空間索引を用いる場合の balance skew や，空間索引を用いる場合の large skew のように大きな増減が見られる箇所があるものの，平均増加率が 1.45% であることから，誤差の範囲内といえる．一方，最終的に構築されたパーティションによる距離結合問合せの実行時間は，LSDP と比べて提案手法が平均して 8.18% の高速化を実現しており，run time が微量ながら増加していることと反している．このことから，実行時間測定の打ち切りが行われないようなパーティションの測定が増加していることが考えられる．

次に，total について LSPD と提案手法を比較する．全体的に学習時間が大幅に増加しており，平均して 13.19% の増加となっている．これはグリッド数の増加によるものと考えられる．探索空間が広がったことで，DNN の更新などにかかる計算量が大幅に増加した可能性がある．

**5. 2. 3** 探索空間最適化に要する時間

表 4 に，各データセットごとにグリッド生成に要する時間を示す．LSDP では均等幅のグリッドを生成しているのに対し，提案手法ではグリッドを適切に生成することにより探索空間最適化を行っているため，グリッド生成にかかる時間が長くなる．この時間の差を，探索空間最適化に要する時間とみなすことができる．均等幅のグリッドを生成した場合と比べて探索空間最適化には約 1 万倍の時間がかかっているが，これは学習時間に対して非常に短い時間であるため無視できる計算時間である．

表 4: グリッド生成に要する時間 [s]

| dataset | grid build time | |
|---|---|---|
| | LSDP | Ours |
| OSM-US | $1.741 \times 10^{-4}$ | 1.211 |
| OSM-SA | $1.778 \times 10^{-4}$ | 1.219 |
| Imis | $1.764 \times 10^{-4}$ | 1.293 |

**5. 2. 4** 境界線付近のデータ数および通信量の比較

表 5 に，KDB-tree と 提案手法の Demo についてパーティションの境界線付近のデータ数を示す．データセットには OSM-US を用いる．境界線から 500[m] 以内，1000[m] 以内，5000[m] 以内に存在するデータ数をパーティションごとにカウントし，その平均値と最大値を示す．すべての距離で，平均，最大値共に提案手法の Demo では境界線付近のデータ数が大幅に減少していることがわかる．このことから，提案手法の通信量の削減を目的とした境界線付近のデータを削減が正しく機能していることがわかる．

表 5: パーティションの境界線付近のデータ数

| タイプ | 平均 | | | 最大値 | | |
|---|---|---|---|---|---|---|
| | 500[m] | 1000[m] | 5000[m] | 500[m] | 1000[m] | 5000[m] |
| KDB | 35.63 | 74.38 | 366.38 | 59 | 129 | 682 |
| Ours (Demo) | 25.63 | 51.88 | 278.38 | 37 | 77 | 425 |

表 6 に，KDB-tree と 提案手法の Demo について通信量の比較を示す．データセットには OSM-US を用いる．通信量の測定には，各ワーカについて別のワーカに送出したパケットの数とデータ量 [byte] を用いる．パケット数は，すべてのパケットを測定したものと，0 [byte] のパケットのみ，および 0 [byte] のパケットを除いたものを示す．また，データ量にはすべての

パケットのデータ量を足し合わせたものを示す．

パケット数は，提案手法の方が平均して増加しているが，ボトルネックとなる最大値は KDB-tree の方が多い．標準偏差には，提案手法と KDB-tree の間で大きな差は見られない．0 [byte] のパケットは，その 99.7% が他ワーカへの通信の際に送信される ACK パケットである．そのため，0 [byte] のパケットが多いということは待ち時間が長いということとほぼ同義である．0 [byte] のパケット数は，平均的には提案手法の方が大きいものの，最大値は KDB-tree の方が大きい．データ量は，平均，最大値，標準偏差ともに提案手法の方が大きい．

以上より，KDB-tree では 0 [byte] の通信を多量に行っているマシンの通信が存在することがわかり，ビジー状態のワーカに対する ACK パケットが問合せ処理のボトルネックとなっていることが予想される．また，提案手法ではパケット数の最大値を抑えることで高速化が実現されている．一方で，平均的には通信量が増加しており，データ量でみると 36.2% 増えていることがわかる．表 5 で示したように，境界線付近付近のデータ数は減少しているため，通信量の増加の原因を解明するには，Apache Sedona の挙動の分析を進める必要がある．

表 6: 通信量

| タイプ | 手法 | 平均 | 最大値 | 標準偏差 |
|---|---|---|---|---|
| パケット数 | KDB | 17131 | 52940 | 14530.4 |
| | Ours (Demo) | 19092 | 47228 | 14193.7 |
| パケット数 (0[byte] のみ) | KDB | 10921 | 45632 | 14783.1 |
| | Ours (Demo) | 12352 | 38589 | 14167.7 |
| パケット数 (0[byte] 除く) | KDB | 6210 | 7308 | 664.8 |
| | Ours (Demo) | 6740 | 8639 | 1090.4 |
| データ量 [byte] | KDB | 47 | 85 | 23.8 |
| | Ours (Demo) | 63 | 115 | 30.9 |

## 6 結　　論

本稿では，学習型空間データパーティショニングを拡張し，パーティションの探索空間の最適化を可能とした．探索空間最適化では，探索空間であるグリッド上で構築されるパーティションの計算負荷分散と通信量削減を考慮したグリッド生成を実現することで，並列分散環境における距離結合問合せの高速化を図る．評価実験では，提案手法による空間データパーティショニングが，幅広い条件下において既存手法より短い時間で距離結合問合せを実行可能であることを示した．

今後の課題として，2 つの問題を取り上げる．1 つ目は，学習時間の短縮である．現在は，報酬計算を行うにあたり実際に与えられたパーティションで実行時間を計測しているが，これが学習時間の大半を占めている．これを解決する方法として，与えられたパーティションから実行時間を推定するコスト関数，またはモデルを構築することが考えられる．2 つ目は，本稿で扱うデータは点データに限られているため，矩形や多角形などのオブジェクトへの対応を行う．

## 謝　　辞

本研究は JSPS 科研費 JP22H03700 および JP20H00584 の支援によって行われた．ここに記して謝意を表す．

## 文　　献

[1] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the Special Interest Group on Management of Data*, pp. 47–57, 1984.

[2] R. A. Finkel and J. L. Bentley. Quad trees a data structure for retrieval on composite keys. *Acta Informatica*, Vol. 4, No. 1, pp. 1–9, 1974.

[3] John T. Robinson. The K-D-B-Tree: A search structure for large multidimensional dynamic indexes. In *Proceedings of the Special Interest Group on Management of Data*, pp. 10–18, 1981.

[4] 堀敬三, 佐々木勇和, 天方大地, 鬼塚真. 深層強化学習を用いた空間データパーティショニング手法の提案. データベース学会論文誌, Vol. 20-J, No. 11, pp. 1–8, 2022.

[5] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the Association for Computing Machinery*, Vol. 51, No. 1, pp. 107–113, 2008.

[6] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the Networked Systems Design and Implementation*, p. 2, 2012.

[7] Yuya Sasaki. A survey on iot big data analytic systems: Current and future. *IEEE Internet of Things Journal*, Vol. 9, No. 2, pp. 1024–1036, 2021.

[8] Jia Yu, Zongsi Zhang, and Mohamed Sarwat. Spatial data management in Apache Spark: The GeoSpark perspective and beyond. *GeoInformatica*, Vol. 23, No. 1, pp. 37–78, 2019.

[9] Mohammad Sultan Mahmud, Joshua Zhexue Huang, Salman Salloum, Tamer Z. Emara, and Kuanishbay Sadatdiynov. A survey of data partitioning and sampling methods to support big data analysis. *Big Data Mining and Analytics*, Vol. 3, No. 2, pp. 85–101, 2020.

[10] Ahmed M. Aly, Ahmed R. Mahmood, Mohamed S. Hassan, Walid G. Aref, Mourad Ouzzani, Hazem Elmeleegy, and Thamir Qadah. AQWA: Adaptive query workload aware partitioning of big spatial data. *Proceedings of the Very Large Data Base Endowment*, Vol. 8, No. 13, pp. 2062–2073, 2015.

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

[12] Vikram Nathan, Jialin Ding, Mohammad Alizadeh, and Tim Kraska. Learning multi-dimensional indexes. In *Proceedings of the Special Interest Group on Management of Data*, 2020.

[13] Pengfei Li, Hua Lu, Qian Zheng, Long Yang, and Gang Pan. Lisa: A learned index structure for spatial data. In *Proceedings of the Special Interest Group on Management of Data*, pp. 2119–2133, 2020.

[14] Stefan Schaal. Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences*, Vol. 3, pp. 233–242, 1999.

[15] Varun Pandey, Andreas Kipf, Thomas Neumann, and Alfons Kemper. How good are modern spatial analytics systems? *Proceedings of the Very Large Data Base Endowment*, Vol. 11, No. 11, pp. 1661–1673, 2018.