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

堀　敬三† 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　〒 565–00871 大阪府吹田市山田丘 1-5

E-mail: †{hori.keizo,sasaki,amagata.daichi,onizuka}@ist.osaka-u.ac.jp

あらまし　膨大な空間データに対する高速な分析処理のために，空間データ分析に特化した並列分散処理フレームワークが開発されている．データパーティショニング技術は，並列分散処理におけるクエリの実行時間に大きく影響するため様々な技術が提案されている．しかし，既存の空間データパーティショニング手法は汎用的な利用のために設計されたアルゴリズムであり，データ分布やクエリワークロードの特性に合わせたパーティションの最適化は実現できない．そのため，本研究では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を提案する．データ分布とクエリワークロードの両方に適応するように学習を行い，最適なパーティションを生成することで空間クエリの高速化を図る．評価実験より，提案手法では一部クエリにおける実行時間を削減できることを示す．

キーワード　空間データパーティショニング, 深層強化学習, 並列分散処理

1 はじめに

IoT 技術の進歩や SNS の普及に伴い，街中のセンサデータやジオタグ付き投稿など，日々膨大な空間データが生成されている．これら空間データの高速な分析は，ビジネスの成長や企業のロケーションインテリジェンスには不可欠であり，ビッグデータ社会を支えるための必要要件である．膨大な空間データに対する複雑な空間分析の必要性が高まり，Spark や Hadoop などの並列分散処理コンピューティング技術を拡張した Apache Sedona [1]，SpatialHadoop [2], Simba [3] などの空間データに特化させた並列分散フレームワークが開発されている．

これらの並列分散処理システムでは，計算処理をスケールアウトできるようにマシン間でデータを分割する必要があり，データパーティショニングが用いられる [4]．データパーティショニングは，データの特性や利用目的に合わせてデータ全体をパーティションと呼ばれる小さな単位に分割し，各マシンにパーティションを割り当てる．これにより，各マシンに割り当てられたパーティションに対するクエリの並列処理が可能となるためスループットが向上する．さらに，パーティショニングが効果的に行われていれば，参照が必要となるパーティションは限られるため I/O が大幅に削減される．パーティション単位での処理が可能となることで，大規模データセットの管理性や可用性の向上が期待できる．

しかしながら，一般的なデータベースシステムで用いられるハッシュなどのパーティショニング手法 [5] は空間データには適していない．例えば，カラムに緯度や経度の位置情報を含む空間データを持つテーブルにハッシュパーティショニングを適用する場合を考える．空間分析においては特定の地域や区域など，まとまった範囲を対象とする処理が多いが，ハッシュでは各レコードが持つ位置情報に関係なく分割されるため，検索時には参照が必要なパーティションが多くなりアクセスの効率が低下する．そのため，空間データパーティショニングにおいては以下の課題が挙げられている [6, 7]．第一に，位置的に近いデータ同士を同じパーティションに分割することで効率的なアクセスを可能にする．第二に，パーティション毎のデータ数を同程度に分割することでマシン間のワークロードバランスを保つ．第三に，境界オブジェクトを最小化することでオーバーヘッドを削減する．境界オブジェクトとは，ポリゴンやラインストリングなどの範囲を持つジオメトリタイプにおいて，複数のパーティションにまたがる位置に存在するデータのことを指し，境界オブジェクトに対する追加処理により，全体的なクエリ性能が低下する．

上記の課題を解決するために，データ分布を考慮した様々な空間データパーティショニング手法が提案されている．しかし，データセットやクエリワークロードに応じて，最適なパーティショニング手法は異なる．そこで，データセットの一部を用いてデータ分布などの特徴を分析し，既存手法の中から適切なものを自動的に選択する方法が提案されている [6]．ただ，既存の空間データパーティショニング手法は汎用的な利用を目的として設計されたものであり，入力されたデータやクエリに対して最適とはいえない．実際は，データ分布やクエリの特性に合わせて特化させた方がクエリ性能が高いことは容易に想像できる．一方で，データベース技術に機械学習を応用する研究が盛んに行われており，データベースやクエリに応じて最適化を図る学習型の手法が多く提案されている．クエリの高速化という観点では，データパーティショニングだけでなく，インデックス技術などにも広く機械学習技術が応用されている [8–15]．しかし，筆者らの知る限りでは空間データパーティショニングを学習する研究は存在しない．

そこで，本稿では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を提案する．まず，空間データパーティショニングを新たに強化学習問題として定式化する．データ分布やクエリワークロードの情報を状態に含め，矩形パーティションを構築するような行動を定義する．さらに，クエリ実行時間を報酬として与えることで，クエリ実行時間を最小化するような

学習設計を行う．学習には，Deep Q Network (DQN) アルゴリズムを用いた深層強化学習 (DRL) を採用することで，複雑な問題設定における膨大な状態数や行動パターンを処理する．DRL エージェントはデータセットとクエリワークロードの両方に適応するように行動ルールを学習し，入力に応じた最適なパーティションを構築することで空間クエリの高速化を図る．予備実験では，既存の空間パーティショニング手法を比較し，分析の結果から提案手法の必要性を示した．評価実験では，提案手法を用いた空間パーティショニングの学習を行うことで一部のクエリに対するクエリ実行時間を短縮した．

本稿の構成は以下の通りである．2 章にて関連研究について説明し，3 章にて事前知識について説明する．4 章にて深層強化学習における空間パーティショニング問題を定義し，5 章にて提案手法について説明する．6 章にて実験について述べ，7 章で本稿をまとめる．

2 関 連 研 究

関連研究として既存の空間パーティショニング手法とデータベース技術への機械学習の応用研究について説明する．

2.1 空間データパーティショニング

空間データパーティショニング手法には，データ分布を考慮した様々なアルゴリズムがある．最も基本的なアルゴリズムは Uniform Grids と呼ばれ，二次元空間に存在するデータに対して均一サイズのグリッドセルを用いて分割するため，一様な分布を持つデータに適している．しかし，多くの場合データ分布には偏りが存在するため, Uniform Grids では効果的な分割は期待できない．そこで，データ分布に合わせて不均一なサイズのグリッドセルを用いて分割する方法として，R-Tree [16], Quad-Tree [17]，KDB-Tree [18] などがある．例えば，R-Tree は位置の近いデータ群の最小外接矩形を階層的に入れ子になるように二次元空間に分割し，Quad-Tree は指定したデータ数に至るまで二次元空間を再帰的に四つの領域に分割する．これにより，データが多く集まる空間はより細かく分割され，パーティション間でのデータ数のばらつきが抑えられる．

2.2 データベース技術への機械学習の応用

データパーティショニングやインデックスといったクエリの高速化を図るデータベース技術に機械学習を応用する多くの研究がある．データパーティショニングへの応用として，[8] は深層強化学習を用いて OLAP スタイルのワークロードに向けたハッシュパーティショニングを学習する．異なるデータベーススキーマやワークロードの変化に対応できるように複数のエージェントを用意することで，最適なパーティショニングを自動的に構築する．[9] ではパーティショニングタスクを強化学習タスクとして定式化し，ワークロードに適切な垂直分割スキームを発見するように深層強化学習を行うことで，いくつかの最先端のアルゴリズムを上回る性能を達成している．

インデックスへの応用では，[10] において B-Tree などの従来のデータベースインデックスを一次元キーに対応するデータ格納位置を予測するニューラルネットワークモデルに置き換えることで，大幅な検索速度の向上と省メモリを達成している．それ以降，学習型インデックスを空間データのような二次元，さらには多次元データに拡張する研究が取り組まれている．LISA [12] は，ディスクベースの学習型空間インデックスであり，範囲クエリ，最近傍クエリ，挿入・削除をサポートする．ZM-index [13, 14] では，Z-order 曲線を用いた多次元データの 1 次元空間へのマッピングと [10] の Recursive-Model Indexes (RMI) を組み合わせている．多くの学習型の多次元インデックスでは，特定のデータセットやワークロードに対してインデックスを自動的に最適化するというアイデアが導入されている．Kraska ら [15] は自身の Flood [11] を拡張し，実際のアプリケーションでよく見られる相関のあるデータや歪んだクエリワークロードを新たに考慮することで，これまでの学習型の多次元インデックス以上の性能を達成している．

しかしながら，空間データパーティショニングを対象とする研究は少ない．深層強化学習を用いて適切な空間データパーティショニング手法を選択するなどの応用 [6] がされているが，これまでのところ空間パーティショニングそのものを学習するよう手法は存在しない．

3 事 前 知 識

提案手法では，深層強化学習の学習アルゴリズムである Deep Q-Network (DQN) を用いる．以下では，強化学習の理論と DQN の関連技術を説明する．

3.1 強 化 学 習

強化学習では，エージェントが環境との相互作用を繰り返すことで，教師データとなる情報を自律的に獲得しながら最適な行動ルールを導き出すものである [19]．相互作用とは，離散的な時間ステップ t 毎に，エージェントは状態 s_t を観測し，行動 $a \in A$ を選択することで，新しい状態 s_{t+1} に遷移し，報酬 r_t を得ることをいう．このとき，将来的により高い報酬を獲得するようにエージェントの行動ルールを最適化することが強化学習の課題である．

3.2 Q-learning

Q-Learning は強化学習手法の一つである [20]．状態 s において可能な行動 $a \in A$ の中から行動価値関数 (Q 関数) の値が最も高い行動を選択するように学習を行う．この値は Q 値と呼ばれ，ある状態とその状態下においてエージェントが可能な行動を対にした行動ルールに対する有効性を示す．エージェントは行動毎に Q 値の更新を行い，状態 s_t でエージェントが行動 a を選択し，状態が s_{t+1} に遷移するとき，Q 関数は以下のように更新される．

$$Q(s_t, a) \leftarrow Q(s_t, a) + \alpha[r_{t+1} + \gamma \max Q(s_{t+1}, a) - Q(s_t, a)] \tag{1}$$

ここで，α と γ はそれぞれ学習率と割引率である．$\alpha (0 \leqq \alpha \leqq 1)$ は現在の Q 値に反映させる更新量の大きさを決定する．

$\gamma(0 \leqq \gamma \leqq 1)$ は 1 に近づけると将来の報酬まで考慮し，0 に近づければ目先の即時的な報酬を優先させることになる．この更新式は現在の状態から次の状態に遷移したとき，その Q 値を次の状態で最も Q 値の高い状態の値に近づけることを意味する．すなわち，高い報酬を得た場合にはその状態に到達することが可能な過去の状態に対しても，その報酬が更新ごとに伝播することになる．これにより，最適な状態遷移が学習される．

3.3 Deep Q Network

Deep Q-Network (DQN) とは，Q 関数をパラメータ θ を持つニューラルネットワークで近似し強化学習を行う手法である [21]．$Q(s,a)$ はすべての状態とすべての行動の組み合わせで定義する必要がある．しかし，状態 s が非常に多くなるような問題設定では，すべての組み合わせを定義することが不可能である．そこで，深層学習の技術を導入することにより高次元な状態空間や行動空間を持つ問題に対しても強化学習を適用することを可能とした．

DQN において損失関数 L は以下の式で定義される．

$$L = \sum_{B}(r_t + \gamma \max Q(s_{t+1}, a; \theta^-) - Q(s_t, a; \theta))^2 \quad (2)$$

$r_t + \gamma \max Q(s_{t+1}, a; \theta^-)$ の部分は $Q(s_t, a; \theta)$ に対する教師データの役割を果たしているため target と呼ばれ，特に $Q(s_{t+1}, a; \theta^-)$ の部分は target Q-network と呼ばれる．B はエピソードデータであり，エージェントの試行錯誤の結果の組 (s_t, a_t, s_{t+1}, r_t) の集合により表現される．DQN を用いた学習では学習精度を向上させるための様々な工夫が取り入れられており，以下では代表的な 3 つを説明する．

3.3.1 ϵ-greedy

ϵ-greedy は強化学習において広く利用される行動選択手法である．エピソードデータはある Q 関数にしたがったエージェントの行動歴である．学習初期では Q 関数を適当に初期化しエピソードデータを蓄積するが，偶然に特定の行動を取りやすい初期 Q 関数であった場合には同じ行動を繰り返し，エピソードデータが偏るため局所解に陥いるという問題がある．そこで，ϵ-greedy では確率 ϵ で学習者が完全にランダムな行動を選択し，あらゆる可能性の探索を可能とすることで問題に対処する．

3.3.2 Experience Replay

学習に用いるエピソードデータはエージェントが蓄積した時系列データである．しかし，ステップ毎にそのエピソードデータを用いて Q 関数の更新を行う場合，時間的にデータ間の相関が高く上手く学習できないという問題がある．これは，類似するエピソードデータで更新を行うことで局所解に陥りやすいためである．そこで，Experience replay では観測した (s_t, a_t, s_{t+1}, r_t) をメモリに蓄積し，学習時にメモリからランダムにデータを取り出して学習することで問題に対処する．

3.3.3 Fixed Target Q-Network

式 (2) において教師データとなる $Q(s_{t+1}, a; \theta^-)$ の値は一つ前のパラメータ θ^- に依存する．しかし，Q 関数が更新される度に教師データが変動すると，学習が安定しないという問題が発生する．そこで，Fixed Target Q-Network では，main Q-network とは別に $Q(s_{t+1}, a; \theta^-)$ の値を計算するための target Q-network を用意する．target Q-network ではパラメータ θ^- を一定期間更新させずに教師データを固定させることで学習を安定させる．そして，一定期間ごとに target Q-network と main Q-network を同期させる．

しかし，target Q-network の更新間隔が大きいほど，ステップが進んだ際に教師データに含まれる誤差が大きくなることで学習精度に影響を与える．そのため，[22] で提案されているアルゴリズムでは τ により更新の度合いを調整し，以下のように緩やかな更新をより小さな間隔で繰り替えすことによりパラメータの安定性が向上する．

$$\theta'_{t+1} = (1 - \tau)\theta'_t + \tau\theta_t \quad (3)$$

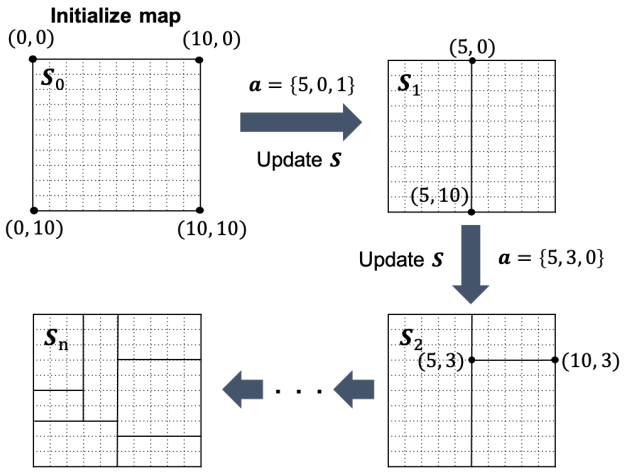


図 1: 空間データパーティショニング学習環境の例

4 問題定義

本章では空間データパーティショニングを新たに強化学習問題として設計する．以下では，本強化学習問題における環境，状態，行動，報酬をそれぞれ定義する．状態の定義には，入力として与えられるデータセットとクエリワークロードに対して，データ分布やワークロードに含まれている各種クエリの情報が明らかであるものとする．また，本稿では学習および実験において空間データ用の並列分散処理フレームである Apache Sedona[1] を利用する．Sedona における空間パーティショニングでは矩形パーティションをサポートしているため，本稿では矩形パーティションを生成するように行動を設計する．図 1 には学習環境と状態遷移の例を示す．

定義 1 (Environment) エージェントが配置される強化学習環境として，図 1 の左上のように二次元空間上にデータ全体を覆う一つのマップを初期状態として形成する．マップは均一な大きさのグリッドセルに分割されており，エージェントは各グリッド線上に新たな境界線を追加していくことでパーティションを構築していく．

1：https://sedona.apache.org/.

定義 2 (State)　状態 S として $s(P)$ と $s(Q)$ の二種類の状態を定義する．$s(P)$ ではグリッドセルの状態を以下のように定義し，全てのグリッドセルに対して適用する．

$$s_{ij}(P) = (h, v, p_{data}, p_{range}, p_{knn}) \tag{4}$$

i, j はグリッドセルの左上座標を指定する．h および v は各座標においてそれぞれ水平と垂直の境界線の有無を表す 0 か 1 である．p_{data} は全データに対してグリッドセル内に存在するレコード数の割合である．p_{range} および p_{knn} はクエリワークロード内の *Range Query* と *kNN Query* のそれぞれで指定される範囲および位置に各グリッドセルが含まれている確率である．学習中は，p_{data}，p_{range}，p_{knn} の値は固定である．

$s(Q)$ ではクエリワークロードの状態を以下のように定義する．

$$s(Q) = (f_1, f_2, f_3, ..., f_n) \tag{5}$$

$f_n (0 \leqq f_n \leqq 1)$ はワークロードに含まれる n 種類のクエリに対して，各クエリが含まれる頻度の割合である．

定義 3 (Action)　一つ一つの行動 $a \in A$ はある座標点から新たな境界線を追加することを指す．そこで，各座標から取り得る行動を以下のように定義する．

$$a = (i, j, d) \tag{6}$$

i と j は行動の開始点として座標を指定する．d は追加する境界線の向きを指定し，右向きであれば 0，下向きであれば 1 とする．

定義 4 (Reward)　現在ステップにおけるパーティション P_t に対して，入力したクエリワークロードからサンプルされたクエリ $q_i \in Q$ の実際の実行時間 $C(P_t, q)$ を測定する．ただし，エージェントは報酬の最大化を目標とするため，報酬の定義には負の符号を与える．

$$r = -\sum_{q \in Q} C(P_t, q) \tag{7}$$

問題定義　学習における目標は，入力されたデータセットやクエリワークロードに対して，クエリ実行時間を最小化するようなパーティショニングを見つけることである．

5　提 案 手 法

本章では，4 章で設計した強化学習問題をもとに DQN アルゴリズムを用いた空間データパーティショニングの学習を行う．図 2 には提案概要として学習全体の流れを示す．指定されたデータセット，クエリワークロード，マシン台数の情報を含めた状態を学習モデルに入力する．学習モデルはパーティションを構築するように行動を繰り返しながら行動価値を学習し，入力に対して最適なパーティションを推論することが目的である．以下では，5.1 節で提案手法の学習方針について説明する．続いて，5.2 節で学習手順について説明する．

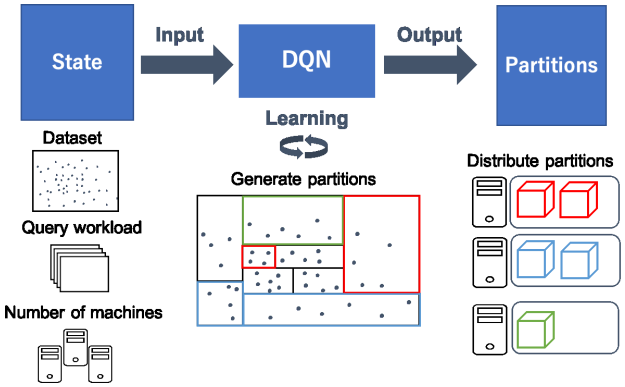


図 2: 提案概要

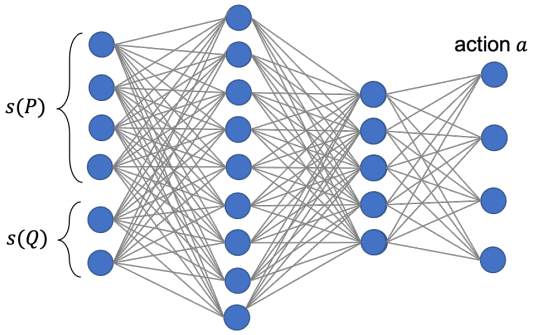


図 3: DQN ニューラルネットワーク

5.1　学 習 方 針

データセットとクエリワークロードの両方に最適な空間パーティショニングを学習するために，4 章ではそれらの情報を状態の一部として定義した．高精度な学習を行うためには初期マップにおいてグリッドセルをより細かく形成しておく必要がある．しかし，状態および行動の定義よりグリッドセル数が多くなるほど，状態空間および行動空間が膨大になる．そこで，DQN を用いた深層強化学習を採用し，学習に用いるニューラルネットワーク モデルのイメージを図 3 に示す．ニューラルネットワークの入力次元，出力次元はそれぞれ状態空間と行動空間の大きさとなる．ただし，二つの状態 $s(P)$ と $s(Q)$ を使用するため，DQN には 2 種類の入力を与えることを想定する．

しかしながら，本問題設定においては膨大な状態空間および行動空間による学習時間の増大が懸念される．特に学習初期では境界線をランダム的に追加するだけでパーティションを構築するような行動を選択することが困難である．そこで，学習の収束性を高めることを目的として，行動 a は境界線が存在する座標，すなわち状態 $s(P)$ において h もしくは v が 1 である座標からのみ行動可能とし，さらに既に境界線が存在する方向への行動は選択できないように制限を与える．これにより，各ステップで新たなパーティションが生成される．

また，過剰なパーティションの生成は空間クエリにおける処理効率を悪化させるため，本稿ではパーティション数はマシン台数の 2 倍という設定で学習を行う．このとき，main Q Network の更新は一定のステップ間隔で式 (2) の損失関数を用いた学習を行う．また，Target Q-Network の更新は main Q

Network の更新よりも細かなステップ間隔で，式 (3) の方法を用いた更新を行う．

報酬には現在のパーティションにおけるクエリ実行時間を測定する．ここで，本問題設定では最終的なパーティションの報酬値を最大化させることが目標である．しかし，行動毎に生成されたパーティションに対する報酬を測定した場合，最終的なパーティションではなく，ステップ毎の各状態における最適な行動の価値が高くなるように学習が進む．すなわち，ステップ毎に最適な行動を取ることが最終的なパーティションにとって良い行動ではない可能性がある．そこで，報酬は各エピソードにおいて行動を繰り返した後，指定のパーティション数に達したときに一度だけクエリの実行時間を測定することで求める．そこで得られた報酬値をこれまでにとった行動全てに対して与える．さらに，過去に同じ状態遷移をした経験がメモリに存在する場合，報酬値の高い方を保持するように更新を行う．

提案する DQN モデルでは，エージェントが行動を繰り返すことで，あらゆるパーティションの状態において取り得る行動の価値を学習し経験を蓄積する．これにより，各グリッドセルに含まれるレコード数や各種クエリの割合などに適応した行動ルールが形成されることで，入力したデータセットとクエリワークロードに最適なパーティションの生成を可能とする．

5.2 学習手順

Algorithm1 に DQN アルゴリズムに基づく提案手法の学習手順を示す．まず，与えられたデータセットおよびクエリワークロードから，マップを初期化し状態を取得する (5 行目)．次に，ϵ-greedy 法による行動選択を繰り返しながら状態を更新する (7–9 行目)．エージェントは指定数のパーティションが構築されるまで行動を繰り返し (10 行目)，$s(P)$ の境界線の有無から現在のステップ t のパーティション P_t を生成してマシンに割り当てる (11 行目)．その後，クエリワークロードからサンプルしたクエリに対して実行時間 $C(P_t, q)$ を測定し，報酬 r を求める (12–13 行目)．ここで，ステップ毎の経験 (s_c, a_c, r, s_{c+1}) をメモリに保持する (15–17 行目)．また，一定間隔ごとにメモリに保存した経験 (s_i, a_i, r_i, s_{i+1}) からサンプルしたデータで main Q-network の学習を行う (20–25 行目)．その後，ϵ を減衰させる (24 行目)．target Q-network のパラメータの更新は定期的に行う (26–27 行目)．ここまでを 1 エピソードとし，指定したエピソード数 e_{max} だけ繰り返す．最終的な DQN モデルの出力は，学習中に保持しておいた報酬が最大となるパーティションの矩形座標の集合 P_{best} である．

Algorithm 1 Learning using DQN

Input: Dataset D, Query Workload Q, The number of machines M

1: Randomly initialize Q-Network Q_θ
2: Randomly initialize target Q-network $Q_{\theta'}$
3: **for** $e = 1, \cdots, e_{max}$ **do** ▷ Episodes
4: **for** $t = 1, \cdots, t_{max}$ **do** ▷ Steps in Episode
5: Initialize map: Reset to state s_0
6: *partitions* $\leftarrow 1$
7: Choose $a_t = \arg\max_{a_t \in A} Q_\theta(s_t, a)$ with probability $1 - \epsilon$, otherwise random action
8: Execute action a_t
9: Update state s_t
10: **if** *partitions* $\geqq 2 * M$ **then**
11: Distribute partitions P_t to machines
12: Compute reward:
13: $r = -\sum_{q \in Q} C(P_t, q)$
14: Update best partitions P_{best}
15: **for** $c = 1, \cdots,$ *partitions* **do**
16: Store transition (s_c, a_c, r, s_{c+1}) in B
17: **end for**
18: **break**
19: **end if**
20: **if** Step interval of train **then**
21: Sample minibatch (s_i, a_i, r_i, s_{i+1}) from B
22: Train Q-network with loss:
23: $L = \sum_{i=1}^{b} (r_i + \gamma \arg\max_{a \in A} Q_{\theta'}(s_{i+1}, a) - Q_\theta(s_i, a_i))^2$
24: Decrease ϵ
25: **end if**
26: Update weights of target Q-network:
27: $\theta'_{t+1} = (1 - \tau)\theta'_t + \tau\,\theta_t$
28: **end for**
29: **end for**

Output: Coordinate of Partitions P_{best}

6 実　　験

本章では，6.1 節にて予備実験として既存の空間パーティショニング手法の性能比較を行う．さらに，6.2 では提案手法に対する評価実験を行う．

6.1 データセット

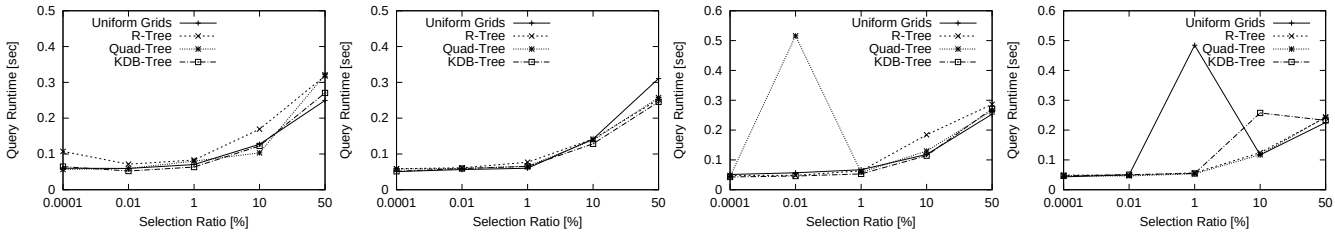
実験では [23] で公開されている Open Street Maps (OSM) データセットを利用する．データセットは，Nodes (Point), Roads (LineString)，Buildings (Polygon), Rectangles (Rectangle) から構成されている．Rectangles は Polygons データから矩形データを抽出して作成されている．予備実験には OSM データセットから一部をサンプルし，表 1 に使用したデータセットの詳細を示す．以降では，各データセットをジオメトリタイプ別に Point, LineString, Polygon, Rectangle と表記する．

表 1: データセット

データセット	ジオメトリタイプ	レコード数
OSM Nodes	Point	10 million
OSM Roads	LineString	10 million
OSM Polygons	Polygon	10 million
OSM Rectangles	Rectangle	10 million

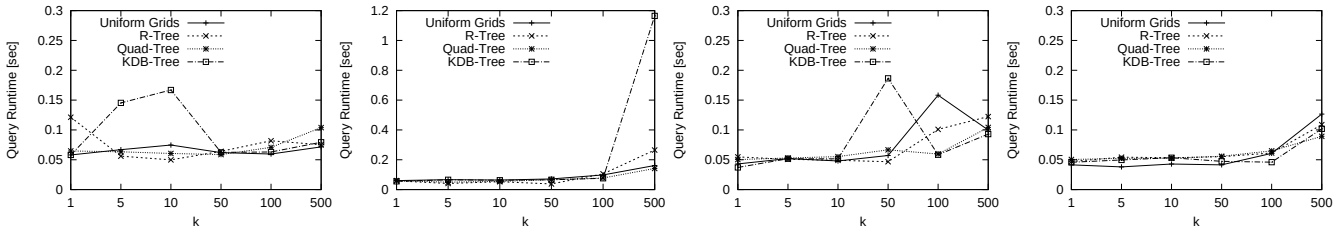
6.2 実験環境

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 台を利用し，Spark クラスタとして 1 台のマスターノードと 8 台の



(a) Point (b) LineString (c) Polygon (d) Rectangle

図 4: Range Query



(a) Point (b) LineString (c) Polygon (d) Rectangle

図 5: kNN Query

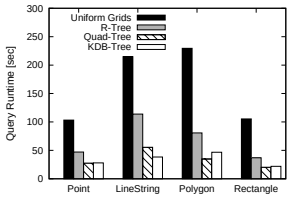


図 6: Distance Join

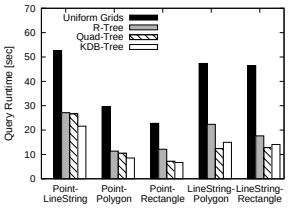


図 7: Spatial Join

スレーブノードを配置する．ただし，予備実験は 3 台のスレーブノードで実施している．

6.3 予備実験

6.3.1 実験設定

予備実験では既存の空間パーティショニング手法から Uniform Grids，R-Tree，Quad-Tree，KDB-Tree の 4 種類のパーティショニング手法を用いる．それぞれの手法に対して，Range Query，kNN Query，Distance Join，Spatial Join の 4 種類のクエリを実行する．ただし，Spatial Join は 2 種類のデータ間で包含されるものを結合する処理である．手法毎に各クエリの 10 回の実行時間を測定し，その平均時間を用いて評価する．

6.3.2 実験結果

本項では予備実験の結果について述べる．クエリ毎の比較結果を図 4–7 に示す．図 4 の Range Query では Point, LineString, Polygon に対して KDB-Tree のクエリ実行時間が比較的小さい．しかし，Rectangle では R-Tree や Quad-Tree の方が優れている．図 5 の kNN Query では Quad-Tree がどのクエリにおいても安定しているが，近傍数 k の値により優れている手法も変化している．これは，10 回のクエリ実行で指定される位置のランダム性も影響していると考えられる．しかし，ランダム性によるクエリの偏りのように，実際の分析においても目的に応じて偏りが生じることが予想できる．そのため，より高速なクエリ処理の実現にはパーティションの最適化は必要である．図 6 の Distance Join では全体的に Quad-Tree と KDB-Tree のクエリ実行時間が小さく，特に LineString では KDB-Tree, Polygon では Quad-Tree が優れている．図 7 の Spatial Join では Distance Join と同様に KDB-Tree と Quad-Tree のクエリ実行時間が小さい．また，使用したデータセットの組み合わせにより優れている手法は異なっている．

Range Query と kNN Query の二つは元々高速であり，特に Range Query ではパーティション間のシャッフルを必要としない処理である [1]．そのため，実行時間的には手法間で顕著な差は見られなかった．その一方で， Distance Join および Spatial Join においては，空間パーティショニングがクエリ実行時間にも大きく影響している．以上の実験を通してデータセットやクエリごとに適切なパーティショニング手法が異なることが証明された．

6.4 評価実験

6.4.1 実験設定

評価実験では，提案手法のクエリの高速化について実験を行う．評価には Range Query，kNN Query，Distance Join の 3 種類のクエリが含まれるワークロードに対する実行時間を測定する．比較対象には，既存の空間パーティショニング手法から Uniform Grids，R-Tree，Quad-Tree，KDB-Tree を用いて，学習設定と同じく 16 個のパーティションを生成した場合の結果を比較する．

ただし，学習および実験評価にはポイントデータセットから 100 万件のみをサンプルした同一のデータセットを利用する．そのため，ワークロードに含まれるクエリについてもポイントデータのみを対象とする．また，学習時の報酬測定で用いるク

表 2: クエリワークロード

Quantity	Range	kNN	Distance
10	0.4	0.4	0.2

表 3: 学習設定

Parameter	**Value**
Learning Rate	0.001
τ (Target network update)	0.001
Optimizer	Adam
Experience Replay Buffer Size	3000
Batch Size for Experience Replay	32
Epsilon Decay	0.9997
Episode	2000
Network Layout	(600,300)
γ (Reward Discount)	0.97
Machines	8
Grid Size	10×20

エリにはワークロード全体を用いる．クエリワークロードに含まれるクエリの総数と各クエリの割合を表 2 に示す．ここで，Range Query における範囲はランダムに指定し，kNN Query の近傍数は 1, 5, 10, 30, 50, 100, 300, 500, 1000，Distance Join の距離 (m) は 10, 50, 100 からランダムに選択する．また，学習に用いたハイパーパラメータを表 3 に示す．

6.4.2 実験結果

本項では評価実験の結果について述べる．各手法に対するクエリワークロード実行時間の比較結果を表 4 に示す．これは，各クエリの実行時間の総和およびクエリワークロード全体の合計時間を示す．Range Query および kNN Query に対する実行時間は提案手法が最短であった．しかし，クエリワークロード全体での実行時間は Distance Join の占める割合が大きく，それが最短である KDB-Tree が最も優れる結果となった．

図 8 には各手法で生成されたパーティションを示す．ここで，緑矩形は Range Query で指定される範囲，青点は kNN Query で指定される位置を示している．パーティションの比較より，それぞれの手法で分割が異なっていることがわかる．ただし，R-Tree に関してはどのグリッドにも当てはまらないオブジェクトを収容するためのオーバーフローパーティションが必要となる．それ以外の手法では，データセットの空間全体から分割を開始するため，オーバーフローパーティションは必要ない．

提案手法の結果について，Range Query ではパーティション間のシャッフルが発生しないため，指定範囲が多くのパーティションにまたがるように分割することで，並列的に処理を行い短縮されていると考えられる．一方で，kNN Query による指定位置は，他手法に比べて境界線から遠くに位置しており，一つのパーティション内で処理が完了することでシャッフルを回避していると考えられる． Distance Join では，パーティション間のデータ数の均等性が大きく影響しており，特に Quad-Tree や KDB-Tree では全てのパーティションでおおよそ均等に分割されていたのに対し，提案手法ではデータ数に偏りがあったため短縮されなかった．

7 おわりに

本稿では，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を提案した．汎用的な利用のために設計されている既存の空間パーティショニング手法に対して，提案手法ではデータ分布やクエリの特性に合わせてパーティショニングを学習し，最適なパーティションを生成することで空間クエリの高速化を図る．評価実験では，提案手法による空間パーティショニングが一部クエリに対して実行時間を短縮することを示した．

今後の展望として，より効果的な設計および学習を行うことでクエリの高速化の実現を目指す．そのために，行動パターンの追加などの新たなパーティション構築方法の考案に取り組む．さらに，現在の学習ではポイントデータのみを状態に与えており，境界オブジェクトに対する学習が不十分であるため，他のジオメトリタイプの追加が必要である．ただし，報酬計算時のクエリ実行にはジオメトリタイプやクエリの種類別に RDD の生成が必要となり，学習時間の増加が懸念される．そこで，実際にクエリ実行時間を測定せずにコストを近似する仕組みを追加することで収束性の向上を図る必要がある．また，パーティション数は処理効率にも影響を与えるが，現在はパーティション数を手動で設定しているため，学習により自動的に決定することを今後の課題とする．

謝辞

本研究は JSPS 科研費 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] Jia Yu, Zongsi Zhang, and Mohamed Sarwat. Spatial data management in Apache Spark: The GeoSpark perspective and beyond. *The Geoinformatica*, Vol. 23, No. 1, pp. 37–78, 2019.

[2] Ahmed Eldawy and Mohamed Mokbel. SpatialHadoop: A MapReduce framework for spatial data. Vol. 2015, pp. 1352–1363, 2015.

[3] Dong Xie, Feifei Li, Bin Yao, Gefei Li, Liang Zhou, and Minyi Guo. Simba: Efficient in-memory spatial analytics. In *Proceedings of the International Conference on Management of Data*, pp. 1071–1085, 2016.

[4] Mohammad Mahmud, Joshua Huang, Salman Salloum, Tamer Emara, and Kuanishbay Sadatdiynov. A survey of data partitioning and sampling methods to support big data analysis. *Big Data Mining and Analytics*, Vol. 3, pp. 85–101, 2020.

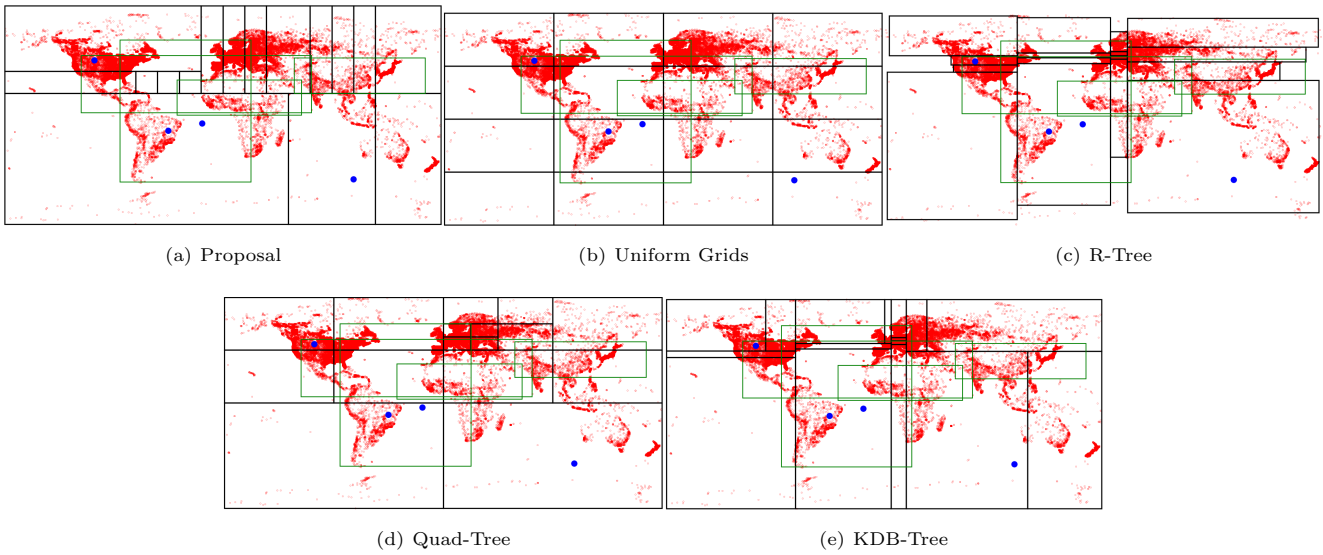
[5] Chengwen Liu and Hao Chen. A hash partition strategy for distributed query processing. In *Proceedings of International Conference on Extending Database Technology*, pp. 371–387. Springer, 1996.

[6] Tin Vu, Alberto Belussi, Sara Migliorini, and Ahmed Eldway. Using deep learning for big spatial data partitioning. *The ACM Transactions on Spatial Algorithms and Systems*, Vol. 7, No. 1, 2020.

[7] Hoang Vo, Ablimit Aji, and Fusheng Wang. SATO: A spatial data partitioning framework for scalable query processing. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Sys-*

表 4: クエリワークロードの実行時間 (sec)

	Uniform Grids	R-tree	Quad-Tree	KDB-Tree	Proposal
Range Query	1.52	1.04	2.34	1.25	0.78
kNN Query	1.12	1.40	2.24	1.19	0.74
Distance Join	23.19	23.75	9.55	9.21	14.63
Total	25.83	26.19	14.13	11.65	16.15



(a) Proposal (b) Uniform Grids (c) R-Tree

(d) Quad-Tree (e) KDB-Tree

図 8: パーティションの比較

tems, pp. 545–548, 2014.

[8] Benjamin Hilprecht, Carsten Binnig, and Uwe Röhm. Learning a partitioning advisor for cloud databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 143–157, 2020.

[9] Gabriel Campero Durand, Rufat Piriyev, Marcus Pinnecke, David Broneske, Balasubramanian Gurumurthy, and Gunter Saake. Automated vertical partitioning with deep reinforcement learning. *The New Trends in Databases and Information Systems*, pp. 126–134, 2019.

[10] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *Proceedings of the International Conference on Management of Data*, pp. 489–504, 2018.

[11] Vikram Nathan, Jialin Ding, Mohammad Alizadeh, and Tim Kraska. Learning multi-dimensional indexes. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 985–1000, 2020.

[12] Pengfei Li, Hua Lu, Qian Zheng, Long Yang, and Gang Pan. LISA: A learned index structure for spatial data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 2119–2133, 2020.

[13] Haixin Wang, Xiaoyi Fu, Jianliang Xu, and Hua Lu. Learned index for spatial queries. In *Proceeding of IEEE International Conference on Mobile Data Management*, pp. 569–574, 2019.

[14] Jianzhong Qi, Guanli Liu, Christian S. Jensen, and Lars Kulik. Effectively learning spatial indices. *The Proceedings of the VLDB Endowment*, Vol. 13, No. 12, pp. 2341–2354, 2020.

[15] Jialin Ding, Vikram Nathan, Mohammad Alizadeh, and Tim Kraska. Tsunami: A learned multi-dimensional index for correlated data and skewed workloads. *The Proceedings of the VLDB Endowment*, Vol. 14, No. 2, pp. 74–86, 2020.

[16] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 47–57, 1984.

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

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

[19] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, Vol. 4, pp. 237–285, 1996.

[20] Christopher JCH Watkins and Peter Dayan. Technical note: Q-learning. *The Machine Learning*, Vol. 8, pp. 279–292, 2004.

[21] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[22] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Tim Harley, Timothy P. Lillicrap, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, Vol. 48, pp. 1928–1937, 2016.

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