時系列データに対する
パターンマッチングと予測に関する研究

２０２０年　３月

中挾 晃介

時系列データに対する
パターンマッチングと予測に関する研究

中挾 晃介

システム情報工学研究科
筑波大学

２０２０年　3月

謝辞

本研究の遂行および論文の作成にあたり，筑波大学計算科学研究センターの北川博之教授には指導教官として終始，懇切なるご指導とご鞭撻を賜りました．ここに深く感謝の意を表します．また，筑波大学計算科学研究センターの天笠俊之教授，筑波大学システム情報系の櫻井鉄也教授，山本幹雄教授，筑波大学図書館情報メディア系の佐藤哲司教授には，本論文の審査におきまして副査を快く引き受けてくださり，本論文の内容についてご指導とご助言を賜りましたこと，深く感謝申し上げます．

また，本研究の遂行においては，公益財団法人鉄道総合技術研究所の皆様のご指導とご協力の上に成り立っております．特に，公益財団法人鉄道総合技術研究所に在籍したまま博士後期課程への進学を許可いただき，多くのご助言をいただいた，信号・情報技術研究部部長 川﨑邦弘氏，信号・情報技術研究部運転システム研究室上席研究員 GL 坂口隆氏，元信号・情報技術研究部運転システム研究室上席研究員 GL（現国際鉄道連合 (UIC)）平井力氏に深謝の意を表します．また，信号・情報技術研究部運転システム研究室主任研究員 國松武俊氏，副主任研究員 辰井大祐氏をはじめ，運転システム研究室の皆様には，数多くのご助言とご協力をいただきましたこと，深く感謝申し上げます．

また，筑波大学北川・天笠データ工学研究室の，渡辺知恵美准教授，早瀬康裕助教，塩川浩昭助教，堀江和正助教をはじめ，研究室のメンバー，OB/OG の方々にも多くのご指導，ご助言をいただきました．深く感謝申し上げます．さらに，投稿した論文に対して多くの査読者の方々から貴重なご意見をいただきました．また学会や研究会においても多くの皆様から貴重なご意見をいただきました．ここに感謝申し上げます．

最後に，いつも温かく見守り励ましていただいた両親，友人に深く感謝いたします．

概要

近年，情報通信技術，センサ技術等の発展に伴い，センサデータ，ログデータ，SNS への投稿データといった時系列データが日々大量に生成・配信されている．時系列データは，データ間の順序が重要な意味を持つデータである．本研究では，その属性として，各時系列に固有の属性 (sequence_id)，タイムスタンプのような各時系列の順序を表す属性 (order_key)，あるタイムスタンプにおける測定値のような，各イベント（行）に固有の属性の，3 種類の属性を持つ時系列データを対象とする．特に，各行に固有の属性を複数持つような，多属性値の時系列データを対象とする．例えば，鉄道を利用する旅客の移動を表す時系列データは，sequence_id として旅客の ID，order_key として処理時刻，各行に固有の属性として処理種別，処理された駅等を含む，多属性値の時系列データである．本研究では，このような時系列データを対象に，時系列データの解析と予測の 2 つの研究に取り組む．時系列データの解析では，複数存在する解析技術の中でも特に，RDB 等に格納された膨大な量の時系列データに対するパターンマッチング（行パターンマッチング）を対象とする．近年，行パターンマッチングのニーズは増加しており，SQL/RPR として SQL2016 において標準化されている．一方で，行パターンマッチングの処理は，Selection や Join といった処理と比較し，コストの大きい処理である．そのため，その処理の効率化が必要となる．そこで，多属性値の時系列データを対象に，行パターンマッチング処理の前に，Selection や Join といった比較的コストの小さい処理を組み合せ，行パターンマッチング対象となる行数を削減する前処理を適用することで，行パターンマッチングの処理コストを削減する 2 種類の最適化手法を提案する．評価実験により，提案する最適化手法の効果を検証する．時系列データに対する予測では，ニューラルネットワークを用いた列車遅延の予測手法を提案する．近年，大都市近郊の鉄道路線では，慢性的な遅延が発生しており，この遅延を抑制するためには，短時間先の遅延を予測する必要がある．そこで，列車の遅延の時系列データに含まれる，各行に固有の属性である発／着事象，駅，遅延などのデータ，また，列車の遅延に影響する他の列車の遅延の時系列データ，乗車率の時系列データといった複数の多属性値の時系列データを加味した，ニューラルネットワークによる列車の遅延予測モデルを提案する．ある実在路線を対象に，提案する予測モデルと他の予測モデルを用いて列車の遅延を予測し，その精度を比較，検証する．

目次

図目次

表目次

1 はじめに

1.1 本研究の背景

近年，情報通信技術，センサ技術の発展に伴い，センサデータ，人の移動データ，SNSへの投稿データといった時系列データが日々大量に生成・配信されている．時系列データは，データ間に順序が存在し，この順序が重要な意味を持つ．

時系列データには，その属性として，各時系列に固有の属性，タイムスタンプのような各時系列の順序を表す属性，あるタイムスタンプにおける測定値のような，各イベント（行）に固有の属性を持つものがある．例えば，鉄道を利用する旅客の移動を表す時系列データであれば，各時系列に固有の属性として個人旅客の ID，そして，各時系列の順序を表す属性として処理時刻，さらに，各行に固有の属性として処理種別，処理された駅などが含まれる．本研究では，このような時系列データを対象とする．対象とする時系列データの詳細は 1.2 節において説明する．

時系列データに対する分析には，時系列データに頻出するパターンの抽出や異常パターンの検出，時系列データに対する将来値の予測，また，類似時系列データの検索やクラスタリングなど，数多く存在する．本研究では，頻出するパターンの抽出や異常パターンの検出といった，ある 1 つの時系列データに着目した分析技術を時系列データに対する解析と分類し，この時系列データに対する解析と時系列データに対する予測を対象とする．

時系列データに対する解析の中で，ユーザが特定のパターンを指定し，そのパターンオカレンスを抽出する行パターンマッチングのニーズが増加している．例えば，ある地点からある地点まで移動する際にどの地点を経由した人がどれ程いるかを知りたいという場合に，人の移動の時系列データに対して行パターンマッチングを行うことで，その答えが得られる．一方で，この行パターンマッチングの処理は，後ほど 3.4 節において示すが，Selection や Join といった処理と比較し，処理コストが大きい．そのため，その処理コストの削減を検討する必要がある．

また，時系列データの中には，鉄道分野において列車運行実績データに含まれる列車の遅延の時系列データが存在する．近年では，大都市近郊の鉄道路線における慢性的な遅延が問題となっており，適切な列車の運行管理のために短時間先の将来の各列車の遅延を精度よく予測することが求められている．列車本数の多い大都市近郊の鉄道路線では，ある列車の遅延は，その列車が通過してきた駅の遅延だけでなく，乗車率や別の列車の遅延も影響する．また，停車する駅によって遅延の傾向は異なる．そのため，列車の遅延の時系

列データの予測には，他の複数の時系列データ，および，遅延以外の属性を考慮する必要がある．

1.2 本研究において対象とする時系列データ

本研究では，時系列データを 2 つの側面から分類する．1 つ目は，各時系列に固有の属性の有無である．各時系列に固有の属性とは，ある 1 つの時系列データに共通する値の属性であり，主に複数のオブジェクトが個々にデータを生成・配信するようなデータが，各時系列に固有の属性を持つ時系列データである．この属性を以降では *sequence_id* と呼ぶ．例えば，消費電力を監視するセンサデータであればセンサ ID，人の移動データであれば，個人 ID や性別などが sequence_id にあたる．sequence_id を持たない時系列データとしては，文字列データ，塩基配列，人口データ，ニュースデータなどが該当する．2 つ目は，各行に固有の属性の数である．本研究では，各行に固有の属性が 1 つのみである時系列データを単一属性値の時系列データと呼び，各行に固有の属性が複数存在する時系列データを多属性値の時系列データと呼び，この 2 種類に分類する．消費電力を監視するセンサデータは，各行に固有の属性は消費電力量であるため，単一属性値の時系列データである．一方で，人の移動データは，各行に固有の属性として，地点（緯度，経度）や立ち寄っている場所のジャンルなどが存在する．本研究では，sequence_id を持つ，多属性値の時系列データを対象とする．

sequence_id を持つ，多属性値の時系列データの例として，旅客の移動データを図 1.1 に示す．旅客の移動データには，旅客 ID，時刻，処理種別，駅の属性を含むとする．まず，旅客 ID が sequence_id に対応する．時系列データは，sequence_id の値を元にデータを分けることで，その値ごとの時系列（シーケンス）が形成される．旅客の移動データでは，旅客 ID 毎に時系列データを分けることで，各旅客の移動のシーケンスが形成される．時刻は，各シーケンスの順序を表す属性である．この属性によって各シーケンス内の順序が定まる．この属性を以降では *order_key* と呼ぶ．処理種別と駅は，各行に固有の属性に対応する．

1.3 本研究の目的

本研究では多属性値の時系列データを対象とした，時系列データの解析と予測に関連する 2 つの研究に取り組む．1 つは，RDB 等に格納された膨大な量の時系列データに対する行パターンマッチング処理の最適化である．行パターンマッチングの処理コストを削減

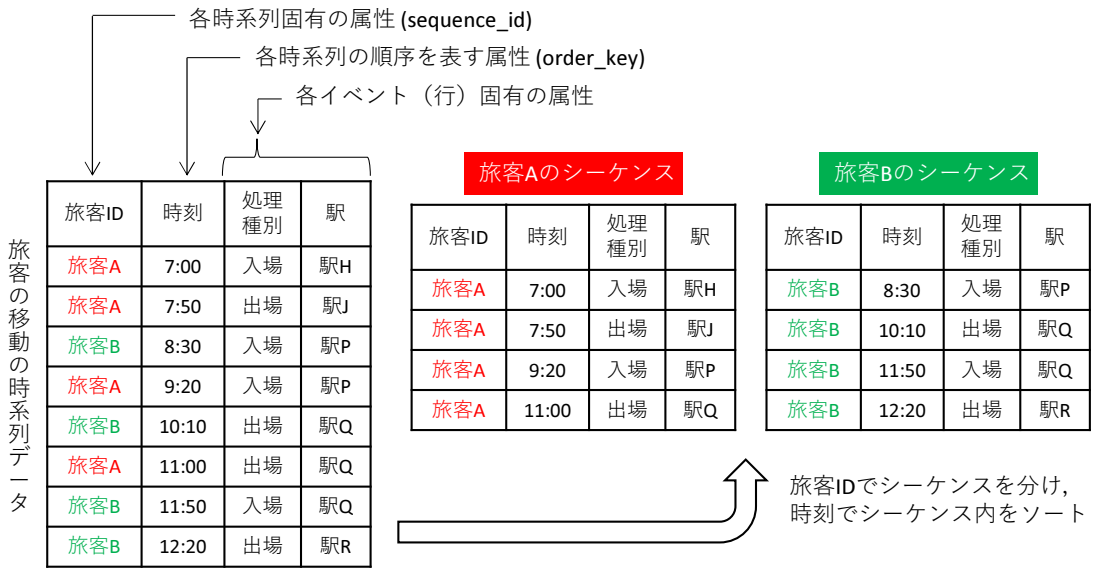

図 1.1 旅客の移動の時系列データ

するために，Selection や Join といった比較的コストの小さい処理を組み合せ，行パターンマッチング前に対象となる行数を削減する前処理を適用することで，行パターンマッチングの処理コストを削減する手法として Sequence Filtering と Row Filtering の 2 つの手法を提案する．本研究では，PostgreSQL と Spark を対象に行パターンマッチングを実装し，2 つの最適化手法の効果を検証する．また，これらの最適化手法のコストモデルを構築し，その妥当性を検証する．これにより，適切な最適化手法の選択が可能となる．

2 つ目は，ニューラルネットワークを用いた列車遅延の予測である．列車運行実績データに含まれる列車の遅延の時系列データを対象に，短時間先までの将来の各列車の各駅の遅延を予測する手法を提案する．時間的に近い他の駅や，他の列車の遅延，乗車率の時系列データを利用した，ニューラルネットワークによる予測モデルを提案する．実路線を対象に，複数の予測モデル間でその精度を検証する．

1.4 本論文の構成

本論文の構成は次の通りである．まず，2 章において，時系列データに対する解析と予測に関連する研究を示す．そして，それらの研究における本研究の位置づけを示す．次に，3 章において時系列データに対する解析に関して，時系列データに対する行パターンマッチングの最適化手法として Sequence Filtering と Row Filtering の 2 つの手法を提

案し，その効果を検証する．そして，4 章において時系列データに対する予測に関して，列車の遅延の時系列データを対象としたニューラルネットワークを用いた列車の遅延予測手法を提案し，その精度を検証する．最後に 5 章において，本研究をまとめる．

2　関連研究と本研究の位置付け

時系列データに対する分析技術として，Esling らは，次の 7 点を挙げている [30]：Query By Content（複数の時系列データの中から，特定の時系列データに最も類似した時系列データを検索），Clustering（時系列データ間で類似した時系列同士をクラスタリング），Classification（複数の時系列データを事前に定義したクラスに分類し，クラスが付与されていない時系列データをクラス分類），Segmentation/Summarization（ある時系列データを，その本質的な特徴を保ちながら次元削減し，その時系列データを正確に近似），Prediction（時系列データの最後の点から，その先の将来の値を予測），Anomaly Detection（時系列データの中から異常なサブシーケンスを検出），Motif Discovery（時系列データの中から繰り返し出現するサブシーケンス（モチーフ）を探索）．本研究では，ある 1 つの時系列データを対象とした解析にあたる技術 (Segmentation/Summarization, Anomaly Detection, Motif Discovery) と時系列データの予測 (Prediction) の技術に焦点を当て，関連する研究をまとめる．そして最後に，それらの関連研究に対する本研究の位置づけを示す．

2.1　時系列データに対する解析

前述の通り，ある 1 つの時系列データを対象とした解析としては，Segmentation/Summarization，Anomaly Detection，Motif Discovery（モチーフ発見）が挙げられる．一方で，Mooney らも，時系列データに対する解析技術として，頻出するサブシーケンスを検出する，Sequential Pattern Mining を挙げている [69]．また，これに関連する技術として，Streaming Data（ストリームデータに対する解析），String Matching and Searching（文字列に対するパターンマッチング），Rule Inference（ルール推論）を挙げている．また，[30] および [69] では挙げられていないが，本研究において対象とする行パターンマッチングも時系列データに対する解析に含まれると考える．本節では，時系列データに対する解析技術として，上述した研究に関連する研究を示す．なお，本節では，時系列データに対する解析技術を，静的な時系列データに対する解析とストリームデータに対する解析に分類し，それぞれ関連研究を示す．

2.1.1 静的な時系列データに対する解析

■Sequential Pattern Mining　RDB に格納された時系列データに対し，頻出するサブシーケンスを検出することを，Agrawal らは，Sequential Pattern Mining と呼び，これに対する手法を提案している [3]．[69] で言及されているように，Sequential Pattern Mining には，Apriori-based Algorithm，Pattern Growth Algorithm が存在する．Apriori-based Algorithm は，頻出アイテムセットマイニングにおけるアプリオリ・アルゴリズムをベースとする [4]．これを用いた手法は，[3] や，これを改良した GSP (Generalized Sequential Patterns) アルゴリズム [95] や，正規表現を用いたパターンの指定により，検出するサブシーケンスに制約を与える，SPIRIT (Sequential Pattern mIning with Regular expressIon consTraints) と呼ばれる手法の提案や [34]，サブシーケンスの検出をオートマトンではなく Hackle-tree と呼ばれる木構造を用いる，RE-Hackle (Regular Expression-Highly Adaptive Constrained Local Extractor) アルゴリズムが提案されている [8]．Apriori-based Algorithm の中でも，時系列データをその時系列順ではなく，各行に固有の属性に含まれる要素の種類別に，RDB に sequence_id を格納する，垂直フォーマッティングにより格納されているデータに対して Sequential Pattern Mining を行う手法も提案されている [119].

一方で，Pattern Growth Algorithm は，頻出アイテムセットマイニングにおける FP-growth アルゴリズムをベースとする [42]．これを用いた手法には，FreeSpan (Frequent pattern-projected Sequential Pattern Mining)[41] や，FreeSpan を改良した，発見済みのサブシーケンス (Prefix) にマッチした残りのサブシーケンス (Postfix) のみを保持する RDB を用いる，PrefixSpan (Prefix-projected Sequential Pattern Mining) が提案されている [84].

■文字列に対するパターンマッチング　Sequential Pattern Mining に関連して，文字列に対するパターンマッチングの高速化の研究が，パケット内の特定の文字列を見つけるための Data Packet Inspection (DPI) などの分野において広く行われてきている．Wang らは，パターンをシーケンシャルな正規表現を含む部分パターンと，繰り返しを表す正規表現を含む部分パターンに分割し，決定性有限オートマトン (Deterministic Finite Automaton, DFA) により高速に処理が可能な，シーケンシャルな正規表現を含む部分パターンからパターンマッチングを行うことにより高速化する手法を提案している [103]．また，Choi らは，文字列の最初の数文字がパターンにマッチするかを判定する前処理を行うことにより，パターンマッチングを高速化する手法を提案している [24]．Cha らは，

パターンマッチングを FFBF (feed-forward Bloom filter) による大まかなパターンマッチングと厳密なパターンマッチングアルゴリズムの 2 ステップに分けて処理する手法を提案している．最初の FFBF によるパターンマッチングにより，処理コストの大きい厳密なパターンマッチングを行う対象となる文字列を削減することで全体の処理の高速化を行っている [20]．Navarro らは，Suffix Automaton を用いて高速に文字列に対するパターンマッチングを行うアルゴリズムである BDM (Backward DAWG Match) を拡張した BNDM (Backward Nondeterministic Dawg Matching) アルゴリズムを提案している [74]．Wang らは，テキスト中から指定したパターンに類似したパターンマッチングを行うアルゴリズムを提案している [101].

文字列に対するパターンマッチングと類似したものとして，塩基配列に対するパターンマッチングの高速化の研究が存在する．Procházka ら [87] は，A，C，G，T の 4 つの塩基の配列に対する 4 つ（あるいは 8 つ）のパターンのパターンマッチングに特化した，BAPM (Byte-Aligned Pattern Matching) アルゴリズムを提案している．このアルゴリズムでは，A を 00，C を 01，G を 10，T を 11 にエンコードし，この 0 と 1 の配列に対するパターンマッチングに特化したアルゴリズムおよびその最適化手法を提案している．Grossi らは，類似した塩基配列のテキストの集合に対してインデックスを構築することでパターンマッチングを効率化する手法を提案している [39].

並列処理によりパターンマッチング処理を高速化する手法も提案されている．Rashmi らは，文字列を分割し，MPI を用いて複数スレッドでパターンマッチングを行うことで高速化を実現する手法を提案している [16]．これに対し，Forain らは OpenMP を用いて，パターンマッチング処理を並列化し，高速化する手法を提案している [32]．Deaton らは，ネットワーク監視システム (NMS) が生成するトラフィックデータに対する正規表現によるパターンマッチングを，Spark を用いて高速化する手法を提案している [26]．Sitaridi らは，SQL における Like 演算子のような，正規表現によるパターンマッチングを高速化するため，SIMD (Single Instruction Multiple Data) を用いて複数行に対して同時にパターンマッチングを行う手法を提案している [93]．Yu らは，文字列を等サイズに分割し，複数スレッドで AC (Aho-Corasick) アルゴリズムを処理し高速化するアルゴリズムを提案し，プログラマブルチップに実装している．[117].

■行パターンマッチング　時系列データに対し，ユーザが抽出したいパターンを定義し，そのパターンオカレンスを抽出する手法として行パターンマッチングがあり，SQL/RPR (Row Pattern Recognition)[1] として SQL2016 において標準化されている．SQL/RPR は，行パターンマッチングを実現するための SQL への拡張である．この SQL/RPR を

実装した RDBMS における，行パターンマッチングの最適化が提案されている [55]．この最適化では，ソートアルゴリズムの選択，クエリの実行計画，DFA と NFA の選択の 3 種類の最適化を提案している．この研究に対する本研究にの提案手法の優位性については，3.2 節において詳説する．行パターンマッチングを実装した RDBMS としては，この他に，Dindar らは，MySQL の内部に行パターンマッチングを行う NFA を実装し，SQL/RPR で記述した行パターンマッチングのクエリを受け取り，処理を実行する，DejaVu と呼ばれるシステムを構築している [28]．また，列指向データベースの一つとして，HP Vertica[57] が存在する．Vertica では，行パターンマッチングのための独自のクエリ記法を導入しており，行パターンマッチング処理を実装している．この他にも，Gehani らは，SQL/RPR の一部を実装した行パターンマッチング手法を提案している [35]．Cadonna らは，一部のパターンにおけるイベントの順序関係を無視したパターンマッチング（SES (sequenced event set) パターンマッチングと呼ぶ），つまり，複数のパターンに合致するような行パターンマッチングの指定が可能な NFA を提案している．行パターンマッチングの状態遷移時に，対象としている行が最終状態までの状態間の遷移条件の一部を満たさず，最終状態まで遷移ができないことが確定する場合，その時点で遷移を終了することで行パターンマッチングの処理を効率化する手法を提案している [18]．

上記に示した行パターンマッチング手法は，オートマトンを用いて行パターンマッチングを行う手法であるが，オートマトンを用いない行パターンマッチング手法も提案されている．Sadri らは，文字列に対するパターンマッチングアルゴリズムである KMP アルゴリズムを行パターンマッチングに対応させた，拡張した OPS (Optimized Pattern Search) アルゴリズムを提案している．OPS では，マッチング中の各イベント間について，比較演算子を含む条件が合致しているか否かの情報を保持し，マッチングに失敗した際，マッチングを再開するイベントを決定する．また，行パターンマッチングのクエリ記法として，SQL ライクなクエリ記法である SQL-TS を提案している [88][89]．Perng らは， SQL/LPP (Limited Patience Pattern) と呼ばれる，パターンマッチングのモデルとクエリ記法を提案している [85]．Attributed Queue と呼ばれる，シーケンス毎に用意された特定の長さのキューを用い，このキューにイベントを挿入・削除し，キュー内のイベントがパターンに合致するかを判定する．パターンマッチングを効率的に行うために，シーケンスデータを長さの異なるセグメントに分け，その各セグメントを節，節間の枝を探索順序とした有向グラフを構築する．この有向グラフがパターンの探索空間となり，マッチするパターンを探索する．また，パターンに対する条件を活用し，この探索空間を小さくすることで，処理の効率化を行っている．Chakravarthy らは，event (-condition) tree により，SQL/RPR の一部を実装した行パターンマッチング手法を提

案している [21].

■モチーフ発見　時系列データにおいて繰り返し出現するサブシーケンス（モチーフ）を発見する技術としてモチーフ発見が存在する．モチーフ発見の手法については，Patel らによって最初に提案されている [81]．Lin らは，モチーフの類似度としてユークリッド距離を利用し，モチーフの集合をバッチ処理により発見する手法を提案している [59]．Mueen らは，モチーフを正確に発見するアルゴリズムを初めて提案している [72]．Mohammad らは，モチーフ発見の処理の中で事前知識を組み込む，制約付きモチーフ発見の手法を提案している [68].

■ルール推論　ルール推論は，時系列データの中で，あるイベント A が出現した後には，イベント B，C が出現するというようなルールを推論する技術であり，Mannila[63] や Padmanabhan[75] によって，その手法が提案されている．Höppner は，時系列データ内の頻出パターンから時間的なルールを見つけるように，ルール推論のセマンティクスを変更している [45]．Papapetrou らは，時系列データ内のイベントは瞬間的ではなく一定期間にわたって続き，複数のイベントが同時に発生していることを考慮した，頻出パターンのルールを発見する手法を提案している [79].

■Segmentation/Summarization　ある時系列データを近似した時系列データを作成する研究としては，Shatkay らは，区分的線形近似 (PLA) を用いた手法を提案している [92]．また，Kehagias は，隠れマルコフモデルを用いた最尤法による手法を提案している [51]．Palpanas らは，時系列データの中で古いデータほどデータを削減する近似を行い，また，その方法をユーザが指定できる新しい時系列データの表現を提案している [77].

■Anomaly Detection　時系列データの中から，一部の異常なサブシーケンスを発見する手法としては，正常な時系列データの振る舞いを学習したモデルを利用して，そのモデルから離れたサブシーケンスを異常とみなす手法がある．この考え方に基づく手法として，Ypma らは，自己組織化マップ (SOM) モデルを用いた手法 [116] を提案している．また，あらゆるサブシーケンスの中で最も異なるサブシーケンスを異常とみなす，“Time Series Discords” と呼ばれる手法が存在する．Yankov らは，この考え方を利用し，時系列データに対して，2 回のスキャンのみで異常なサブシーケンスを発見することができる手法を提案している [113]．さらに，Chen らは，異常なサブシーケンスは，最も発生頻度が低いという考え方に基づき，PAV (Pattern Anomaly Value) と MPAV (Multi-scale Pattern Anomaly Value) の 2 つのアルゴリズムを提案している [23].

2.1.2 ストリームデータに対する解析

■複合イベント処理 ストリームデータに対しては，複合イベント処理 (Complex Event Processing, CEP) と呼ばれる，ストリームデータに対してパターンマッチングを行う手法が知られている．CEP の例としては，Cayuga[27]，SASE[108]，ZStream[67] などが挙げられる．Cayuga[27] では，Cayuga Event Language (CEL) と呼ばれる，SQL ライクなクエリ言語により，抽出したいイベントを記述する．Cayuga では，その内部で非決定性有限オートマトン (Non-deterministic Finite Automaton, NFA) を用いており，CEL で記述したパターンに合致するための条件が状態間の遷移条件となる．遷移条件に合致するイベントが到達すると，次の状態に遷移する．各状態においては，その状態まで遷移したイベントを保持しており，最後の状態まで遷移すると，パターンに合致したとみなし，各状態において保持されているイベントがパターンにマッチしたイベントとして出力される．SASE[108] は，RFID により生成されたイベントデータに対し，特定のパターンを抽出する手法を提案している．SASE では，NFA を構築する点においては Cayuga と同一であるが，合致するパターンを正規表現で記述し，特定のイベントが発生しないという否定の演算子を用いてパターンの記述が可能なクエリ言語を導入している．また，パターンにおける一部の条件をプッシュダウンすることにより，パターンマッチング処理の効率化を行っている．Kolchinsky らは，パターンマッチングのための NFA の構築に際し，パターンに対する条件の中で合致する頻度が低い条件から評価をすることで処理の効率を上げる，chain topology NFA および tree topology NFA を提案している [54]．Agrawal らは，NFA とマッチバッファを組み合せたパターンマッチング処理のモデルとして NFA^{b} を考案し，その最適化手法を提案している [5]．Ding らは，イベントストリームが事前に定義されたワークフローに基づき生成されるという前提の下，NFA によるマッチングの途中で，イベントの発生順序等の制約を満たさない場合にそれ以上のマッチングを取りやめる，Runtime Query Unsatisfiability (RunSAT) チェックアルゴリズムを提案している [29]．これにより，実行時に NFA における不要な遷移が排除され，パターンマッチング処理の効率化が可能となる．Barga らは，CEP や pub/sub 等の処理を統一的に扱う Complex Event Detection and Response (CEDR) と呼ばれる，イベントストリーム処理システムを提案している [12]． 1 つのイベントデータは，Validation time，Occurrence time，CEDR time の 3 つの時間属性を持つモデルであり，これらの属性により，データ間の順序の一貫性を維持する．また，CEP や pub/sub 等の処理を統一的に記述できる言語を提案している．

NFA ではなく，木構造をベースとしたモデルを用いてパターンマッチングを行う，

ZStream[67] が提案されている．ZStream では，合致するパターンを正規表現で記述し，パターンを構成する各サブパターンに合致する各イベントを葉，イベントに対する条件を枝で表した，木構造をベースとしたクエリプランを構築する．これにより，論理積や論理和といった条件を利用できる．また，一部の条件をプッシュダウンすることにより，パターンマッチング処理の効率化を行っている．ZStream のように，パターンマッチングにおける評価プランを構築する手法に対して，Kolchinsky らは，イベントが持つ特定の属性における値毎の到着率のような統計的な情報から，最適な評価プランをリアルタイムに構築する手法を提案している [53]．この手法では，評価プランの変更に伴って出力される結果に誤りがないことを保証している．Ju らは，Pattern Combination Graph (PCG) と呼ばれるデータ構造とそれに対応するアルゴリズムにより，ストリームデータに対して効率的に複合パターンマッチングを行う手法を提案している [48]．Zhou らは，時間的な不確実性に対応したシーケンスデータに対するパターンマッチング手法を提案している [124]．時間的に依存するイベント毎にグループ分けし，そのグループ単位で行パターンマッチングを行う．また，そのグループ内のイベントに対し，OI-tree (Order Instance Tree) と呼ばれる，時間的な順序を考慮した二分木を構築する．このインデックス構造を用いて，パターンマッチングを行う．

上記に挙げたものは，パターンマッチングを行う処理自体を効率化するアプローチであるが，そうではなく，前処理によりパターンマッチングを効率化する仕組みとして，Cadonna らは，前処理フェーズ（フィルタリング，パーティショニング）とパターンマッチングフェーズからなるパターンマッチング手法を提案している [17]．イベントが到達すると，パターンマッチングの前に特定のウィンドウ幅分のキューに格納される．このキューに格納されたイベントに対し，パターンマッチングの条件に合致しないイベントを除外する処理を行う．この前処理により，パターンマッチングにかかる処理コストを削減している．

■Sequential Pattern Mining　ストリームデータに対して，頻出するサブシーケンスを検出する手法も提案されている．Giannella らは，FP-growth アルゴリズムをベースとし，ストリームデータから頻出するサブシーケンスをマイニングする FP-tree ベースの FP-stream と呼ばれるモデルを提案している [36]．また，この他にも，回帰ベースのアルゴリズムである，FTS-DS (Frequent Temporal Patterns of Data Streams)[97] や，MAIDS (Mining Alarming Incidents from Data Streams)[19] といった手法が提案されている．

■モチーフ発見　モチーフ発見をオンラインで処理する手法が提案されている．Mueen らは，ウィンドウ幅のサブシーケンスから 1 組のモチーフ対を発見する手法を提案している [71]．Lam らは，これを改良し，k 組のモチーフ対を発見する手法を提案している [56]．また，加藤らは，ウィンドウ幅のサブシーケンスの中で最も多く出現するサブシーケンス（レンジモチーフ）を発見するアルゴリズムを提案している [49]．

■Anomaly Detection　オンラインで，正常な時系列データの振る舞いを学習し，その学習したモデルから異常なサブシーケンスを発見する手法として，Ma らは，サポートベクター回帰を用いた手法 [62] を提案している．また，Tan らは，木構造を用いた高速な 1 クラス異常検出器として，Streaming Half-Space-Trees (HS-Trees) と呼ばれるモデルを提案している [96]．

2.2 時系列データに対する予測

次に，時系列データに対する予測に関する研究を説明する．時系列データに対して予測を行う研究は数多く存在するが，ここでは，まず，単一の時系列データのみを対象とした予測に関する研究と，複数の時系列データを用いた予測に関する研究に分けて示す．そして，それらの中で，線形なモデルを用いて予測する研究と，非線形なモデルを用いて予測する研究に分けて示す．

2.2.1 単一の時系列データを対象とした予測に関する研究

■線形モデルを用いた時系列データに対する予測に関する研究　線形な予測モデルを用いて時系列データの予測を行うモデルとして，一般的には，ARIMA（自己回帰移動平均モデル）[14] や重回帰分析などが用いられる．

ARIMA を用いた時系列データに対する予測を行う研究としては，翌日の電力価格の予測 [25]，短期的な風速の予測 [38] などが存在する．また，Livera らは，時系列データの予測に広く用いられる指数平滑化モデルに，複数の季節変動要素を加味して予測ができるような拡張を行った，BATS (Box-Cox transformation, ARMA residual, Trend component, Seasonal component) モデルを提案している [61]．

■非線形モデルを用いた時系列データの予測に関する研究　時系列データの予測には，非線形な予測モデルも広く使われている．一般的な多層パーセプトロン (MLP) による Feed-Forward 型のニューラルネットワークを用いた時系列データの予測として，Gurmu らは，GPS データからバスの走行時間を予測し，停留所のバス到着時刻を予測する手法

を提案している [40]．Jindal らは，ニューヨークにおける GPS 対応のタクシーから取得できる GPS データを用いて，出発地から目的地までの距離と時間を，ディープニューラルネットワークを用いて同時に予測する，Spatio-Temporal Neural Network (ST-NN) と呼ばれる手法を提案している [47]．Zhang らは，為替レートの予測を対象に，複数のニューラルネットワークモデルが出力した予測結果から時系列データの予測を行うアンサンブル予測手法を提案している [121]．Innamaa は，高速道路上の特定の数地点に設置したビデオカメラから取得した，設置地点間毎の数分間隔の自動車の平均通過時間の時系列データを入力とし，将来の各地点間の平均通過時間を予測する手法を考案している [46]．また，Valipour らは，ダム貯水池の流入量の予測を，ARMA，ARIMA，ニューラルネットワークの 3 種類のモデルを用いて予測し，それらの精度を比較している [99]．

また，ニューラルネットワークのような非線形なモデルと線形なモデルを組み合せた予測手法も提案されている．Zhang らは，時系列データの予測に際し，線形成分の予測に ARIMA モデル，非線形成分の予測にニューラルネットワークを利用し，それらを合わせた予測結果を出力する予測手法を提案している [120]．Khandelwal らは，事前に，時系列データに対して Discrete Wavelet Transform (DWT) を適用し，その後，[120] の研究で提案している ARIMA とニューラルネットワークのハイブリッドな予測手法を用いて予測を行う手法を提案している [52]．

近年では，Recurrent Neural Network (RNN)[106]，Long Short Term Memory (LSTM)[44] といった，データの時系列性を考慮したニューラルネットワークが存在し，時系列データの予測に用いられている．Wu らは，時系列データをアンサンブル経験的モード分解によって分解し，分解された要素の中から適切な数の要素を選択し，それらをもとに LSTM によって価格を予測する手法を提案している [109]．Poornima らは，RNN における勾配消失問題に対応するために，LSTM の入力ゲートにおいてシグモイド関数および tanh 関数に入力値を乗算する，Intensified LSTM と呼ばれるモデルにより，降雨量を予測する手法を提案している [86]．Alahi らは，歩行者の将来の位置を LSTM によって予測し，自律車両が歩行者との衝突を回避する方法を提案している [7]．また，短期的な電力負荷データのような，非定常時系列データを RNN により精度良く予測可能な最適化アルゴリズムの提案や [122]，太陽光発電 (PV) モジュールの電力エネルギーの出力を予測する手法 [15]，バスの到着時刻を RNN により予測する手法 [78] なども提案されている．

また，サポートベクター回帰 [100] を用いて，財務データ [112]，高速道路上の自動車の走行時間 [107]，NASDAQ の株価 [50]，バスの到着時刻 [13] を予測する研究も行われている．Pai らは，サポートベクターマシンと ARIMA を用いたハイブリッドの予測手法を

提案している [76].

2.2.2 複数の時系列データを用いた予測に関する研究

■線形モデルを用いた時系列データに対する予測に関する研究 複数の時系列データを用いた線形モデルによる予測手法として，Yi らは，複数の共進化する時系列データに対して予測を行うことを目的とし，ある時系列データの値の予測に，自時系列データと他の時系列データから，多変量線形回帰を用いて予測する，MUSCLES と呼ばれる手法を提案している [115]．各タイムステップにおいて複数の時系列データを用いた線形式を作成し，最小二乗法により回帰係数の行列を導出する．MUSCLES では，時系列データ数や 1 つの時系列データのデータ量が増えることに伴う，回帰係数の導出コストと保持コストを削減するために，逆行列の補助定理を用いることで，スケーラビリティを持たせている．

■非線形モデルを用いた時系列データの予測に関する研究 非線形モデルを用いた，複数の時系列データを対象とした予測の研究として，Widipurta らは，時系列データ間の関係のプロファイルと，特定のプロファイルが出現した際の時系列データの周期的な値の傾向の検知とクラスタリングの 2 つのステップにより予測を行う，カーネル回帰と ECM (Evolving Clustering Method)[94] を組み合せた予測手法を提案している [105]．また，同氏は，カーネル回帰により複数の時系列データ間からそれらの関係を抽出し，k-近傍法を用いて過去から適切な関係を探索して予測する手法も提案している [104]．Abhishek らは，ニューラルネットワークを用いて気温と風速の時系列データから雨量を予測するモデルを提案している [2]．Pears らは，複数の時系列データの予測について，グローバルモデル，ローカルモデル，トランスダクティブモデルの 3 つのモデル層から構成される，Integrated Multi-Model Framework (IMMF) と呼ばれるフレームワークを用いて予測する手法を提案している [82]．松原らは，複数の時系列データから，同じ特徴を持つ時間幅のパターンを regime と呼び，この regime を捕捉して将来の時系列データを予測する，REGIMECAST と呼ばれる手法を提案している [65][66]．REGIMECAST は，regime の遷移 (regime shift) に着目し設計された手法で，時系列データの値が常微分方程式によって求まる Latent Non-linear Dynamical System と呼ばれるモデルを用いて予測を行う．この際，予測時点での時系列データがどの regime に属しているかを推定し，それに対応する常微分方程式を用いて予測を行う手法である．Zhou らは，複数のセンサデータに対する予測において，訓練フェーズが不要なガウス過程による予測手法として SMiLer を提案している [123]．SMiLer では，ガウス過程の訓練フェーズにおいて膨大な学習時間がかかる問題と，時系列データが時間経過によってその特徴が変化することで，時間的に

古い時系列データを含めて予測すると精度が落ちる問題 (concept drift) に対処するために，Support Vector Machine (SVM) と k-Nearest Neighbor (kNN) のハイブリッドの学習アプローチ（準遅延学習モデル）を提案している．直近のタイムスタンプにおける値を入力とし，kNN によって近い時系列データのセグメントを検索し，その最も近い時系列セグメントをガウス過程のモデルの構築に用いてセンサの値を予測する．Liu らは，離散値をとる複数の時系列データに対し，時系列データ予測用に拡張した単純ベイズ分類器 (T-NB) と決定木 (T-DT) を用いる予測手法を提案している．この手法を用いて，製油所に設置された 7 つのセンサデータから異常値の発生を予測している [60]．Pecev らは，バスケットボールのコート上の位置を表す 2 つの時系列データから，3 人の審判の位置を表す時系列データを予測する LTR-MDTS (Left To Right - Multiple Dependent Time Series Prediction) モデルを提案している [83]．多層のニューラルネットワークのモデルに対し，t1 時の入力層の出力が t2 時の 1 つ目の隠れ層の入力に，t1 時の 1 つ目の隠れ層の出力が t2 時の 2 つ目の隠れ層の入力というように，ある層の入力が，1 つ前の層の出力と，1 つ前のタイムステップ時の 1 つ前の層の出力も含む予測モデルを考案している．誤差関数値には，独自の評価値である SRC (Satisfactory Results Criteria) を利用している．このモデルにより，最適な審判の位置を学習・予測を行う．Gmati らは，複数の時系列データに対する予測にラフ集合を用いるアプローチを提案している [37].

2.3 本研究の位置付け

本研究は，各行に固有の属性を複数持つ，多属性値の時系列データを対象とし，時系列データの解析として行パターンマッチング，そして，時系列データの予測として複数の時系列データを用いた予測の研究を行う．

行パターンマッチングに関して，本研究では，行パターンマッチングの処理コストを削減する最適化手法の検討を行う．パターンマッチング処理の効率化という点では，文字列に対するパターンマッチングの処理の効率化が広く提案されている．一方で，3.2 節において詳説するが，文字列に対するパターンマッチングの処理の効率化手法は，パターンにマッチするための条件が，文字に対する等号条件であることが前提となっている．そのため，多値，多属性の時系列データに対する行パターンマッチングの処理の効率化には適用できない．本研究では，時系列データに対する行パターンマッチングに適用が可能な，処理コストを削減する最適化手法を提案する．また，既存の最適化手法では考慮されていない，各行に固有の属性に対する条件を加味した最適化手法を提案する．

複数の時系列データを用いた予測に関して，本研究では，列車の遅延の時系列データを

対象とした予測手法の検討を行う．大都市圏の通勤路線では，特に朝ラッシュ時においては列車の本数が多く，ある列車の前後列車が遅延すると，その列車にも遅延が波及する．そのため，ある列車の遅延の時系列データの予測においては，他の列車の遅延の時系列データと組み合わせて予測を行う必要がある．複数の時系列データに対する予測においては 2.2.2 節で示した通り多くの手法が提案されてきている．一方で，本研究で対象とする列車の遅延の時系列データの予測においては，遅延だけでなく，駅や，発／着といった，各行に固有の複数の属性の値を考慮する必要がある．これは，遅延の値が，その列車の時間的に近い駅における遅延の値ほど相関があることに加え，各駅の規模や機能（乗換駅や待避可能駅など）によって遅延の傾向が異なるためである．そのため，本研究では，上記を考慮した，複数列車の遅延の時系列データを用いた予測手法を検討する．

3 行パターンマッチングの処理コスト削減のための最適化手法

3.1 背景と目的

1.1 節で述べたように，日々，膨大な量の時系列データが生成・配信され，RDB 等に蓄積されている．この時系列データに対しては，ある特定のパターンが含まれるかどうかを検出する行パターンマッチングが重要となる．行パターンマッチングによって検出された部分シーケンスをパターンオカレンスと呼ぶ．行パターンマッチングの処理では，sequence_id によってデータを分割（パーティショニング）し，order_key によって行をソートすることで，行のシーケンスを形成する．そして，ユーザがある行のシーケンスのパターンを指定し，そのパターンに合致する全てのパターンオカレンスが抽出される．行パターンマッチングは，時系列データに対する分析において本質的に重要な処理である．

RDB に蓄積された静的な時系列データに対する行パターンマッチングは，SQL/RPR (Row Pattern Recognition)[1] として SQL2016 において標準化されている．SQL/RPR は，行パターンマッチングを実現するための SQL への拡張である．ユーザは，シーケンスから抽出するパターンを定義するために，FROM 句中に MATCH_RECOGNIZE 句を指定し，行パターンマッチングを行う．MATCH_RECOGNIZE 句については，3.3 節において詳述する．

一方で，この行パターンマッチングには，Selection や Join 等の他のクエリ処理に対し，その処理コストが比較的大きい．行パターンマッチングの処理コストについては，3.4 節において詳述する．そのため，この行パターンマッチングを効率的に処理する方法を考える必要がある．

そこで，本研究では，RDB 等に格納された膨大な時系列データに対する行パターンマッチングの処理コストを削減するために，行パターンマッチング処理の前に，Selection や Join といった比較的コストの小さい処理を組み合せ，行パターンマッチング対象となる行数を削減する *Filtering Query* を事前に適用することで，行パターンマッチングの処理コストを削減する．この手法は，多属性値の時系列データにも適用が可能である．本研究では，行パターンマッチング対象となる行数を削減する 2 種類の最適化手法を提案する．具体的には，MATCH_RECOGNIZE 句に含まれる行パターンマッチングのための条件を用いた 2 種類の Filtering Query を作成する．1 つ目は，MATCH_RECOGNIZE 句の条件を満たす行がシーケンス内に 1 行も存在しないシーケンスをフィルタリングする

Sequence Filtering である．2 つ目は，MATCH_RECOGNIZE 句の条件を満たす行とその前後数行を残し，それ以外の行をフィルタリングする *Row Filtering* である．これらの Filtering Query を行パターンマッチングを含むオリジナルのクエリの前に適用し，行パターンマッチング対象となる行数を削減する．

本研究で提案する Sequence Filtering と Row Filtering について，PostgreSQL と Spark[118] を対象に，その効果を検証する．Spark は，並列分散処理環境を用いて大量のデータを処理するフレームワークであり，Spark SQL[10] と呼ばれる SQL ライクなクエリ言語を用いて，ビッグデータ分析処理を記述できる．Spark SQL においても，行パターンマッチングは重要であるため，今後，SQL/RPR と類似の行パターンマッチングの処理の仕組みが Spark SQL にも実装されることが想定される．また，Spark はビッグデータ分析に標準的に用いられるフレームワークであるため，Spark SQL において効率的な行パターンマッチングを実現することは，重要な課題である．そこで，本研究では，SQL/RPR が Spark SQL に実装されることを想定し，Spark に SQL/RPR を実装する手法を提案し，この SQL/RPR を実装した拡張 Spark と，UDF により SQL/RPR を実装した PostgreSQL を用いて，Sequence Filtering と Row Filtering の効果を検証する．

本章では，以下の流れで説明を行う．まず，3.2 節において関連研究に対する提案手法の優位性を示す．3.3 節では，SQL/RPR で標準化されている MATCH_RECOGNIZE 句について説明する．その後，3.4 節において，行パターンマッチングの処理コストを示す．次に，3.5 節において，Sequence Filtering と Row Filtering の 2 つの最適化手法について説明する．そして，3.6 節において，提案する最適化手法のコストモデルを構築する．手法の検証の前に，3.7 節において，PostgreSQL における MATCH_RECOGNIZE 句の実装，3.8 節において，Spark における MATCH_RECOGNIZE 句の実装を説明する．3.9 節において，2 つの最適化手法の効果を検証するための評価実験と 3.6 節において構築したコストモデルの妥当性を検証する．最後に 3.10 節において本研究のまとめと今後の課題を示す．

3.2 関連研究

2.3 節においても述べたように，パターンマッチングの処理の効率化という点では，文字列に対するパターンマッチングの処理の効率化が広く提案されている．一方で，文字列に対するパターンマッチングと行パターンマッチングでは，以下の点において異なる．

- 文字列に対するパターンマッチングでは，オートマトンにおける状態の遷移の条件

が単一の属性の単一かつ特定の文字との等号条件であるのに対し，行パターンマッチングでは，オートマトンにおいて状態の遷移の条件が複数の属性に対する複数の等号条件以外の条件（不等号条件と呼ぶ）も含む．

- 行パターンマッチングでは，遷移の条件として今までマッチしてきた行の値を参照することができる一方で，文字列に対するパターンマッチングでは，他の行の値を参照しない．

文字列に対するパターンマッチングは，上記の性質が前提の効率化手法が提案されており，例えば，最初の数文字がパターンにマッチするかをビットマップインデックスを用いて判定し，パターンマッチング対象となる行数を減らしたり [24]，ブルームフィルタリングにより，事前にパターンマッチング対象となる行数を減らす手法が提案されている [20]．しかし，これらの手法は，パターンにマッチするための条件が有限の種類の文字に対する等号条件が前提となっている．そのため，多値，多属性である時系列データに対する行パターンマッチングの効率化のために，文字列に対するパターンマッチングの効率化手法は適用できない．本研究では，多値，多属性である時系列データに対する行パターンマッチングに対して適用可能な手法を検討する必要がある．

一方で，2.1.1 節で示したように，静的な時系列データを対象とした行パターンマッチングの最適化手法も提案されている．その中で，既に SQL/RPR を実装している RDBMS における行パターンマッチングの最適化 [55] では，主に 3 つの方法を導入している．1 つ目は，行パターンマッチング前にシーケンスデータをソートする際の，ソートアルゴリズムの選択である．2 つ目は，WHERE 句の条件の一部を行パターンマッチング前に実行するためのプッシュダウンである．これはクエリ中の WHERE 句において，sequence_id に対する条件が存在する場合，MATCH_RECOGNIZE 句の前にプッシュダウンし，そのシーケンスをフィルタリングするものである．3 つ目は，DFA と NFA の選択である．バックトラッキングが不要なパターンが定義された場合，線形時間で終了する DFA が選択される．

[55] の最適化手法と本研究において提案する最適化手法は，次の 2 点において異なる．1 点目は，シーケンスのフィルタリングに用いる属性である．[55] では，sequence_id の条件のみによってシーケンスがフィルタリングされるが，提案手法では，属性に対する制約なく，シーケンスのフィルタリングを行う．2 点目は，シーケンスによらない個々の行のフィルタリングである．[55] では，シーケンス単位でのみフィルタリングを行うが，本研究では，行単位でフィルタリングを行い，パターンマッチング対象となる行数をさらに

削減する．

本研究のように，前処理により処理の効率化を行う研究は，行パターンマッチング以外にも存在する．文字列に対するパターンマッチングの高速化では，2.1.1 節で紹介した，[24] や [20] が挙げられる．これらに加え，Aguilera らは，Publish/Subscribe System において, 送信されるデータのスキーマ情報から, 特定の属性に対する条件を基に Subscribe する，Content-based Subscription System における効率的な処理手法を提案している [6]．全体の処理を前処理フェーズとマッチングフェーズに分け，前処理フェーズでは Subscribe のための条件についての matching-tree を構築し，処理の効率化を行っている．Fabret らは，Subscribe のための条件に対する B-tree Index と Hash Index を事前に構築することで，イベントとサブスクリプションの条件の比較回数を減らすことで，処理の高速化を図っている [31]．Altinel らは，XPath クエリで表現されたユーザプロファイルに対し，XML ドキュメントのマッチングを効率的に行う XFilter を提案している [9]．処理の効率化のために，XPath クエリに含まれている要素をキーとした転置インデックスを構築し，入力となる XML ドキュメントを事前にスキャンすることで，XPath クエリに含まれていない要素を含む XML ドキュメントのフィルタリングを行う．ただし，これらの手法についても，先ほど挙げた行パターンマッチングと文字列に対するパターンマッチングの違いから，行パターンマッチングの処理の最適化には適用できない．

また，2.1.2 節において，ストリームデータに対するパターンマッチングの効率化において，前処理によりパターンマッチング対象となる行数を削減する手法 [17] を示したが，この手法はストリームデータを前提としているため，本研究において対象とする静的な時系列データに対する行パターンマッチングには適用できない．

3.3 SQL/RPR における MATCH_RECOGNIZE 句

3.3.1 MATCH_RECOGNIZE 句の構文

SQL/RPR において新しく導入された，MATCH_RECOGNIZE 句を説明する．これは, RDB に格納された時系列データに対する行パターンマッチングを行い, 出力としてパターンオカレンスのテーブルを得るものである．図 3.1 に, MATCH_RECOGNIZE 句の文法を示す．また, 図 3.2 に, MATCH_RECOGNIZE 句を用いたクエリの例（Example Query とする）を示す．

図 3.2 の Example Query は，person_id（sequence_id に対応），time（order_key に対応），location（各行に固有の属性に対応）という 3 つの属性を持つ，人の移動の時系列データ (moving_table) に対して，地点 A->任意の地点->地点 C の順に移動するよう

```
SELECT …
FROM <table_name> <pattern_recognition_clause> AS <result_table> , …
WHERE …
```

```
<pattern_recognition_clause> ::= “MATCH_RECOGNIZE  (”
                                    “PARTITION BY” <sequence_id_list>
                                    “ORDER BY” <order_key_list>
                                    “MEASURES” <measure_list>
                                    “ONE ROW PER MATCH” | “ALL ROW PER MATCH”
                                    “AFTER MATCH SKIP” { “PAST LAST ROW”
                                                       | “TO NEXT ROW”
                                                       | “TO FISRT” <correlation_name>
                                                       | “TO LAST” <correlation_name>
                                                       | “TO” <correlation_name> }
                                    “PATTERN (” <regexp> “)”
                                    “SUBSET” <subset_definition_list>
                                    “DEFINE” <corrname_definition_list> “)”
<sequence_id_list> ::= <sequence_id> { “,” <sequence_id_list> }
<order_key_list> ::= <order_key> { “,” <order_key_list> }
<corrname_definition_list> ::= <correlation_name_definition>
                                 { “,” <corrname_definition_list> }
<correlation_name_definition> ::= <correlation_name> “AS” <cond_list>
<cond_list> ::= <cond_list> { AND | OR | XOR } <cond_list>
              | NOT <cond_list>
              | “(” <cond_list> “)”
              | <condition>
              | <between_condition>
<condition> ::= <one_variable_cond> | <other_cond>
<one_variable_cond> ::= <correlation_name>“.”<column_name>
                          { “<” | “<=” | “>” | “>=” | “=” | “<>” } <non_attr_arith_expr>
<non_attr_arith_expr> ::= <const_value>
                        | <non_attr_arith_expr> { “+” | “-” | “*” | “/” } <non_attr_arith_expr>
                        | “(” <non_attr_arith_expr> “)”
<correlation_name> ::= <パターン変数>
<column_name> ::= <<table_name>の属性名>
```

図 3.1 MATCH_RECOGNIZE 句の構文

なパターンの行パターンマッチングを行い，最終的に出力されるパターンオカレンスとして地点 A と地点 C の時刻を含むテーブル (result_table) が得られるクエリである．以降で，MATCH_RECOGNIZE 句の文法の詳細を説明する．

MATCH_RECOGNIZE 句は，FROM 句内に指定する．図 3.1 における MATCH_RECOGNIZE 句の前の<table_name>に行パターンマッチング対象となる時系列データのテーブルを指定する．図 3.2 の Example Query では，moving_table を指定している．本研究では, FROM 句に指定する MATCH_RECOGNIZE 句に関連す

moving_table

person_id	time	location
1	5:00	D
1	6:00	A
1	7:00	A
1	8:00	B
1	9:00	C
1	10:00	A
1	11:00	B
1	12:00	B
1	13:00	G
1	14:00	C
2	8:00	A
...	...	...

```
SELECT *
FROM moving_table
MATCH_RECOGNIZE ( PARTITION BY person_id
                  ORDER BY time
                  MEASURES X.time AS X_time,
                           X.location AS X_loc,
                           Z.time AS Z_time,
                           Z.location AS Z_loc
                  ONE ROW PER MATCH
                  AFTER MATCH SKIP PAST LAST ROW
                  PATTERN ( X Y+ Z )
                  DEFINE X AS X.location = 'A',
                         Y AS COUNT(Y.*) <= 2,
                         Z AS Z.location = 'C'
                ) AS result_table
```

result_table

person_id	X_time	X_loc	Z_time	Z_loc
1	6:00	A	9:00	C
...	...	...	...	...

図 3.2 Example Query

る一連の部分，“<table_name> <pattern_recognition_clause> AS <result_table>”を，*MATCH_RECOGNIZE specification* と呼ぶ．MATCH_RECOGNIZE 句内の構文は<pattern_recognition_clause>に示している．

<table_name>で指定した時系列データを個々のシーケンスにパーティショニングするための sequence_id は PARTITION BY 句で指定する．Example Query では，person_id を指定することにより，各個人が経由した地点のシーケンスが形成される．

シーケンス内の行の順序を定める order_key は ORDER BY 句で指定する．Example Query では，time を指定することにより，各個人が経由した地点のシーケンスが時刻順に並べられ，移動履歴のシーケンスが形成される．

マッチングしたい行パターンは PATTERN 句で指定する．任意の文字列を用いた“パターン変数”でマッチングする行を表し，正規表現を用いてパターンを指定する．Example Query では，パターン変数 X，Y，Z を用いて，地点 A と地点 C の間に任意の 2 地点以下を経由するパターンを指定している．

マッチングしたい行に対する条件指定は，PATTERN 句で指定したパターン変数に対

する条件を DEFINE 句で指定することで行う．パターン内の行の属性は，“<パターン変数>.<属性名>” で表され，これを用いた条件式により指定する．Example Query では，パターン変数 X の地点を A，パターン変数 Z の地点を C と指定している．また，パターン変数 Y に対する条件のように，集約関数 COUNT を用いて 2 地点以下を経由するというように指定できる．本研究では，DEFINE 句における条件<condition>を，*one variable condition* とそれ以外の条件指定（*other condition* と呼ぶ）の 2 種類に分類する．one variable condition の定義を，図 3.1 の<one_variable_cond>に示す．one variable condition は，式中に<パターン変数>.<属性名>が 1 つのみ存在し，集約関数を含まない．other condition は，one variable condition 以外の全ての条件指定とする．Example Query では，パターン変数 X，Z に対する条件が one variable condition であり，パターン変数 Y に対する条件が other condition に対応する．

SUBSET 句では，一部のパターン変数を組み合わせ，1 つのパターンとして再定義できる．

出力するパターンオカレンスのテーブルの属性（メジャー）は，MEASURES 句で指定する．メジャーは，“<パターン変数>.<属性名>”，集約関数，およびそれらを組み合わせた式で定義できる．なお，sequence_id は，定義なしにメジャーに含まれる．Example Query では，パターン変数 X にマッチした行の地点および時刻とパターン変数 Z にマッチした行の地点および時刻をメジャーとして定義している．また，person_id もメジャーとして含まれる．出力するパターンオカレンステーブルの相関名は，図 3.1 の<result_table>に指定する．Example Query では，result_table と指定している．

SQL/RPR では，パターンオカレンスを出力するタイミングとして，パターンに一致した時点でそのパターンオカレンスを 1 行出力する ONE ROW PER MATCH と，パターンの一部にマッチする度にパターンオカレンスを 1 行出力する ALL ROWS PER MATCH のいずれかを選択できる．Example Query ではデフォルトの ONE ROW PER MATCH を指定している．

また，パターンに一致後，行パターンマッチングを再開する行を AFTER MATCH SKIP 句により指定できる．デフォルトは，パターンに一致した最後の行の次の行から行パターンマッチングを再開する，AFTER MATCH SKIP PAST LAST ROW である．この他に，最後に行パターンマッチングを開始した行の次の行から再開する AFTER MATCH SKIP TO NEXT ROW や，指定したパターン変数にマッチした最初／最後／当該の行からパターンマッチングを再開する AFTER MATCH SKIP TO {FIRST<パターン変数> , LAST <パターン変数>, <パターン変数>} が指定可能である．Example Query ではデフォルトの AFTER MATCH SKIP PAST LAST ROW を指定している．

MATCH_RECOGNIZE 句の処理の流れは，次のようになる．まず，FROM 句で指定したテーブルが sequence_id と，order_key を元にシーケンスとして形成される．次に，PATTERN 句，DEFINE 句，SUBSET 句で指定した条件から DFA あるいは NFA が構築され，行パターンマッチングが実行され，MEASURES 句で定義したメジャーを属性として持つパターンオカレンスが，ONE ROW PER MATCH あるいは ALL ROWS PER MATCH のいずれかのタイミングで出力される．そして，パターン一致後，AFTER MATCH SKIP 句により指定した行から行パターンマッチングが再開される．

3.3.2 本研究における SQL/RPR の前提条件

本研究では，SQL/RPR の前提条件として下記の 2 点を置く．1 点目として，パターンオカレンスを出力するタイミングを制御するオプションとして ONE ROW PER MATCH のみを対象とし，ALL ROWS PER MATCH は対象外とする．

2 点目として，PATTERN 句で指定するパターンによって，抽出されるサブシーケンス内の行数は一意に定まるものとする．例えば，$PATTERN$ ($X\ Y\ Z$) のような 3 地点間を順に経由するシーケンシャルのパターンの場合，X，Y，Z のそれぞれに 1 行ずつマッチするため，このパターンに対して抽出される各サブシーケンス内の行数は 3 行である．ただし，Example Query のように，パターンに*や + 等の繰り返しを表す正規表現を用いる場合，行数は一意に定まらない．一方で，実用上，無限に長いパターンオカレンスを取得したいという要求はほとんど無いと考えられるため，本研究では，抽出されるサブシーケンス内の最大行数がユーザによって指定されるものとする．Example Query の場合，パターン変数 Y にマッチする最大行数は 2 行であるため，パターン全体でマッチする最大行数は 4 行である．

3.4 行パターンマッチングの処理コスト

本章では，時系列データに対する行パターンマッチングの処理コストの大きさを予備実験により示す．ここでは，Selection を行うクエリ (Selection Query)，Join を含むクエリ (Join Query)，行パターンマッチングを行うクエリ (MR Query) の 3 種類のクエリの処理時間をテーブルの行数を変えて比較する．図 3.3 に 3 種類のクエリを示す．なお，行パターンマッチングを行うクエリの実行には，3.8 節において説明する，SQL/RPR を実装した Spark を用いる．実装には，後ほど 3.8.1 節において説明する図 3.13 の右側に示した Front-end Approach で実装した手法を用いる．実験環境としては，5 ノードからなるクラスタマシンを用いる．クラスタ構成として，1 ノードをマスターノード，残りの 4

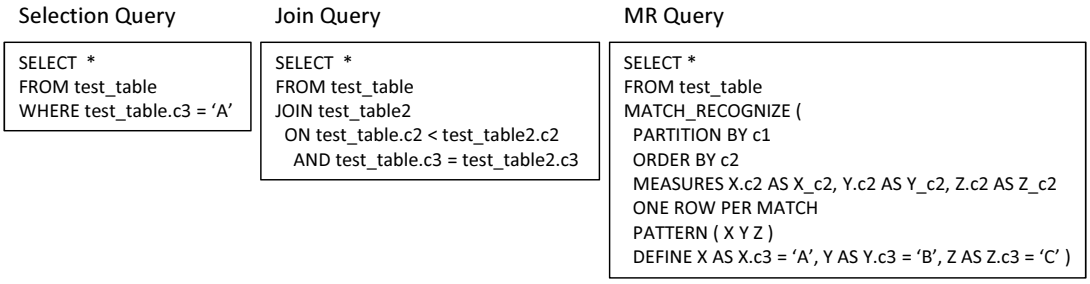

図 3.3 処理コストの比較に用いる 3 種類のクエリ

表 3.1 3 種類のクエリの処理時間

n (行)	Selection Query (s)	Join Query (s)	MR Query (s)
10,000	10.39	13.89	34.39
100,000	9.97	30.34	41.99
1,000,000	13.05	81.53	79.50
10,000,000	15.31	285.12	434.13

ノードをワーカーノードとする．また，分散ファイルシステムとして HDFS を用い，リソースマネージャとして，YARN を用いる．なお，各ノードには，Ubuntu 14.04 LTS, AMD Opteron™Processor 2435 @2.60GHz CPU，8GB RAM の PC を使用する．

予備実験に用いるデータは，c1，c2，c3 の 3 つの属性を持つテーブル (test_table) である．c1 は sequence_id に対応する属性であり，1 から始まり 10,000 行毎にインクリメントされる属性である．c2 は order_key に対応する属性であり，1 から 1 行毎にインクリメントされ，10,001 行目で 1 に戻り，再びインクリメントされる属性である．c3 はアルファベット 1 文字が入る文字列の属性であり，各行に固有の属性に対応する．

表 3.1 に，test_table の行数を変えたときの，各クエリの処理時間を示す．値は，5 回の試行の平均値である．結果として，Selection Query は処理時間がほぼ一定であるのに対し，MR Query は，行数が増えていくに従って処理時間が大きく増加し，10,000,000 行においては，Selection Query と Join Query に比べ，処理に時間がかかっていることが分かる．この結果から，行パターンマッチングの処理コストを削減することは重要な課題であることが言える．

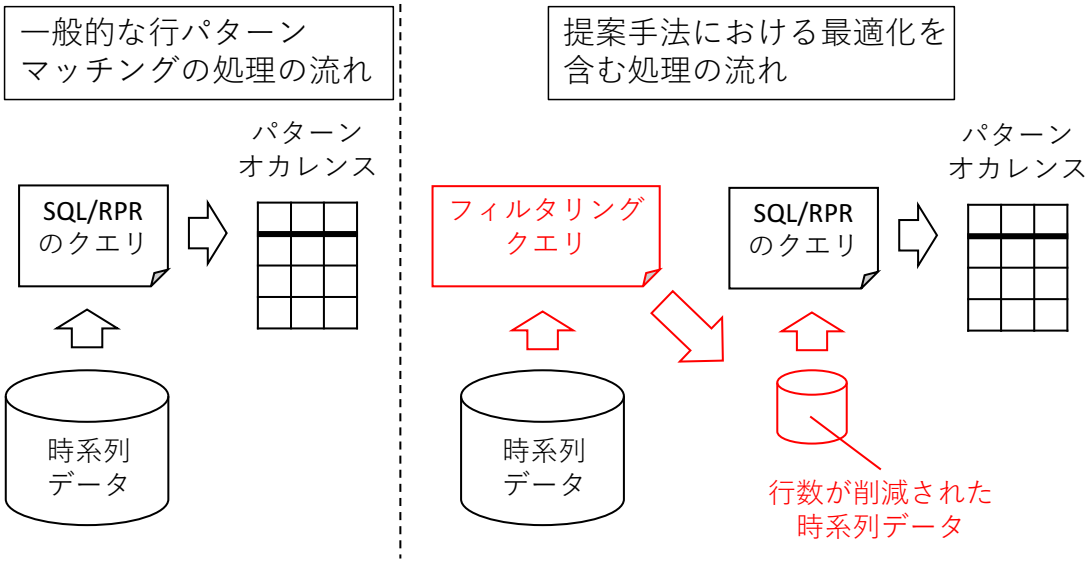

図 3.4 行パターンマッチングの処理コストの削減手法の概要

3.5 行パターンマッチングのための最適化手法

本研究では，行パターンマッチングの処理の前に，入力テーブルに対してフィルタリングを行い，入力テーブルの行数を削減することで，行パターンマッチングにかかる処理コストを削減する．本手法の概要を図 3.4 に示す．通常は，RDB に格納された時系列データに対し，MATCH_RECOGNIZE 句を用いた SQL/RPR のクエリにより行パターンマッチングを行い，その結果としてパターンオカレンスを得る．一方で，本研究において提案する手法は，Filtering Query を適用することで事前に時系列データを削減し，削減した時系列データに対して SQL/RPR のクエリにより行パターンマッチングを行う．これにより，行パターンマッチングの処理コストを削減する．入力データの行数を削減する最適化手法として，2 種類の手法を提案する．1 つ目は，対象シーケンス数を削減する Sequence Filtering，2 つ目は，対象行数を削減する Row Filtering である．以降において，それぞれの最適化手法について説明する．

3.5.1 Sequence Filtering

sequence_id によって形成された各シーケンスに着目し，あるシーケンス内に DEFINE 句において指定されている one variable condition を満たす行が 1 行もない場合，そのシーケンス全体を行パターンマッチングの対象外とできる．これらのシーケンスをフィル

タリングすることにより，処理コストが削減できる．Sequence Filtering が適用可能な条件を以下に示す．

- DEFINE 句において，少なくとも 1 つのパターン変数の条件に one variable condition が存在する．
- ONE ROW PER MATCH を指定する．

ALL ROWS PER MATCH は，パターンの一部にマッチする度にパターンオカレンスを 1 行出力するため，one variable condition を満たす行が 1 行もないシーケンスであっても，出力するパターンオカレンスが存在する．そのため，Sequence Filtering により，出力されるべき結果が欠落してしまう．本研究では，パターンに一致した時点でそのパターンオカレンスを 1 行出力する ONE ROW PER MATCH を指定することを前提とする．なお，Sequence Filtering は，SQL/RPR において PATTERN 句に指定できる正規表現を用いたパターンであれば，指定するパターンに制約はない．

Sequence Filtering は，*Sequence Filtering Query* によってなされる．図 3.5 に，Sequence Filtering Query の構文を示す．また，図 3.6 に，Example Query に対応する Sequence Filtering Query を示す．<filt_cond_list>は，MATCH_RECOGNIZE 句を含むクエリで DEFINE 句に指定した one variable condition を OR で繋げた条件である．Sequence Filtering では，まず，この<filt_cond_list>を満たす行を含むシーケンスの sequence_id のテーブルを作成する．その後，<table_name>と Join する．これにより，one variable condition を満たす行が存在しないシーケンスがフィルタリングされる．図 3.6 の Example Query の例では，まず，location の値として，“A” あるいは “C” を含むシーケンスの person_id のテーブルを作成する．その後，この person_id のテーブルと moving_table を person_id について Join することで，“A”，“C” を含まないシーケンスがフィルタリングされる．

3.5.2 Row Filtering

Sequence Filtering では，パターンオカレンスを構成する行が含まれる可能性のあるシーケンスを残し，その他のシーケンスを除去した．つまり，シーケンスがフィルタリングにおける単位であった．Row Filtering は，行パターンマッチングの結果としてパターンオカレンスを構成する行に含まれ得ない行を除去する．つまり，Row Filtering では，行がフィルタリングにおける単位となる．Row Filtering では，3.3.2 節において説明した，指定したパターンからマッチするサブシーケンス内の行数が定まることを利用する．

```
SELECT *
FROM ( SELECT DISTINCT <sequence_id_list>
         FROM <table_name>
         WHERE <filt_cond_list>
       ) AS <sequence_id_list_table>
  JOIN <table_name> ON <table_name>.<sequence_id_list>
                         = <sequence_id_list_table>.<sequence_id_list> ;
```

```
<filt_cond_list> ::= <one_variable_cond_list>
<one_variable_cond_list> ::= <one_variable_cond_list> “OR” <one_variable_cond>
                           | <one_variable_cond>
```

図 3.5 Sequence Filtering Query

```
SELECT *
FROM ( SELECT DISTINCT person_id
         FROM moving_table
         WHERE location = ‘A’ OR location = ‘C’
       ) AS moving_table2
JOIN moving_table ON moving_table2.person_id = moving_table.person_id ;
```

図 3.6 Example Query に対応する Sequence Filtering Query

パターン変数に対する条件に合致する行に加え，パターンにマッチするサブシーケンスの行数分, 前後の行を残し, それ以外の行をフィルタリングすることで, 行パターンマッチング対象となる行数を削減し，処理コストを削減する．なお，MATCH_RECOGNIZE 句を含むクエリに対して Row Filtering が適用できる条件は, 3.5.1 節の Sequence Filtering が適用できる条件と同一である．また，Sequence Filtering と同様，SQL/RPR において PATTERN 句に指定できる正規表現を用いたパターンであれば，指定するパターンに制約はない.

Row Filtering は，*Row Filtering Query* によってなされる．図 3.7 に，Row Filtering の構文を示す．また，図 3.8 に，Example Query に対応する Row Filtering Query を示す．まず，パターンにマッチしうるサブシーケンスの行数 n から 1 を引いた行数分，前後の行を対象とするウィンドウにより，<table_name>に対して Window 関数を適用する．Window 関数では，ウィンドウ内に<filt_cond_list>を満たす行が存在する場合，フラ

```
SELECT *
FROM ( SELECT <sequence_id_list>, <order_key_list>, <column_name_list>,
                    MAX ( CASE WHEN <filt_cond_list> THEN 1 ELSE 0 END )
                       OVER ( PARTITION BY <sequence_id_list>
                                  ORDER BY <order_key_list>
                                  ROWS BETWEEN (n – 1) PRECEDING AND (n – 1) FOLLOWING ) AS <flag>
          FROM <table_name>
        ) AS <flag_table>
WHERE <flag> > 0 ;
```

```
<column_name_list> ::= <column_name> { “,” <column_name_list> }
```

図 3.7 Row Filtering Query

```
SELECT *
FROM ( SELECT person_id, time, location,
                    MAX( CASE WHEN location = ‘A’ OR location = ‘C’
                            THEN 1 ELSE 0 END )
                    OVER ( PARTITION BY person_id
                              ORDER BY time
                              ROWS BETWEEN 3 PRECEDING AND 3 FOLLOWING ) AS flag
          FROM moving_table ) AS flag_table
WHERE flag > 0 ;
```

図 3.8 Example Query に対応する Row Filtering Query

グを立てる処理を行う．図 3.8 では，$n = 4$ であるため，対象行に加え前後 3 行分のウィンドウの中に，location の値として，“A” あるいは “C” を含む行が存在する場合，フラグを立てる．このようにフラグを立てたテーブルを作成し，最後に，このテーブルに対してフラグが立っていない行をフィルタリングする．これにより，パターンにマッチするサブシーケンスに含まれ得ない行がフィルタリングされる．

以上から，行パターンマッチング処理においては，(1) 最適化を行わない (No Optimization)，(2) Sequence Filtering を行う，(3) Row Filtering を行う，(4) Sequence Filtering の後，Row Filtering を行う (Sequence Filtering + Row Filtering)，の 4 種類の手法が考えられる．(4) を行う理由としては，全入力テーブルに対し Row Filtering を行うよりも，一度 Sequence Filtering により行を削減した後に Row Filtering を行う方が，Sequence Filtering のオーバーヘッド以上に処理時間が削減されることが期待されるためである．

表 3.2 コストモデルにおいて用いる記号の定義

記号	意味
N	テーブルの行数
S	テーブルに含まれるシーケンス数
c	1 行に対するシーケンシャルスキャンにかかるコスト
r	1 行に対する行パターンマッチングにかかるコスト
w	1 行に対する Window 演算にかかるコスト
α	Sequence Filtering によって残るシーケンスの割合
β	Sequence Filtering によって残ったシーケンスの中で，さらに Row Filtering によって残る行の割合

3.6 各最適化手法のコストモデル

前節の最後に示した, (1) No Optimization, (2) Sequence Filtering, (3) Row Filtering, (4) Sequence Filtering + Row Filtering の 4 種類の最適化手法について，RDBMS における処理を対象に，それぞれの処理コストを定義する．表 3.2 に記号の定義を行う．

まず，(1) No Optimization の処理コストについて，N 行のテーブルに対して行パターンマッチングを行うため，その処理コストは以下となる．

$$rN \tag{1}$$

次に, (2) Sequence Filtering の処理コストを考える．Sequence Filtering では, 3.5.1 節において説明したように, 主に, サブクエリ内の, WHERE 句における<filt_cond_list>を満たす行の探索のためのスキャンコストと，<filt_cond_list>を満たす行を含むシーケンスの sequence_id のテーブルと元テーブルの Join コストが存在する．元テーブルにはインデックスが張られていると仮定する．そのため，スキャンコストは無視できるほど小さいとし，0 とする．また，Join には，Hash Join を用いると仮定し，ハッシュ表の作成コストを作成対象のテーブルのスキャンコストと等しいとする．この行数は $\alpha\beta N$ であるため，そのコストは，$\alpha\beta cN$ となる．Hash Join のコストは，<filt_cond_list>を満たす行を含むシーケンスの sequence_id のテーブルの行数が αS となるため，$c(N + \alpha S)$

となる．行パターンマッチングの対象行数は，αN となるため，行パターンマッチングにかかるコストは，αrN となる．以上から，(2) Sequence Filtering の処理コストは以下となる．

$$\alpha\beta cN + c(N + \alpha S) + \alpha rN \tag{2}$$

次に，(3) Row Filtering の処理コストを考える．Row Filtering では，3.5.2 節において説明したように，主に，サブクエリ内における，Window 演算にかかるコストと，Window 演算の結果としてフラグが付与されたテーブルに対し，フラグが立っていない行をフィルタリングするスキャンコストが存在する．Window 演算にかかるコストは wN となり，また，フラグが付与されたテーブルは N 行であるため，そのスキャンコストは cN となる．よって，これらのコストの和は $(w + c)N$ となる．行パターンマッチングの対象行数は，$\alpha\beta N$ となるため，行パターンマッチングにかかるコストは，$\alpha\beta rN$ となる．以上から，(3) Row Filtering の処理コストは以下となる．

$$(w + c)N + \alpha\beta rN \tag{3}$$

最後に，(4) Sequence Filtering + Row Filtering の処理コストを考える．Sequence Filtering を行う処理コストは，上記に示した Sequence Filtering の処理コストと同一である．Row Filtering における，Window 演算の対象行数は，αN となるため，Row Filtering を行う処理コストは，$(w + c)\alpha N$ となる．以上から，(4) Sequence Filtering + Row Filtering の処理コストは以下となる．

$$\alpha\beta cN + c(N + \alpha S) + (w + c)\alpha N + \alpha\beta rN \tag{4}$$

上記に示した処理コストの妥当性を，3.9.3 節において検証する．

3.7 PostgreSQL における行パターンマッチング

本研究では，PostgreSQL における SQL/RPR を UDF により実装している．ここでは，UDF による実装方法と，最適化手法の適用方法を説明する．

```
SELECT *
FROM match_recognize (
  ‘ SELECT row_number() OVER (ORDER BY <sequence_id>, <order_key>) AS rid,
          <sequence_id>, <order_key>, <column_name_list>
   FROM <table_name> ’ ,
  ‘ <measure_list> ’ ,
  ‘ ( <regexp> ) ’ ,
  ‘ <corrname_definition_list> ’ )
  AS ( <sequence_id> <type>, <measure_and_type_list> ) ;
```

```
<measure_and_type_list> ::= <measure> <type> { “,” < measure_and_type_list > }
```

図 3.9 関数 match_recognize() の引数と戻り値

3.7.1 PostgreSQL への SQL/RPR の実装

PostgreSQL への SQL/RPR は，那須らの実装 [73] をベースとし，UDF により実装している．本研究では，行パターンマッチングを行う関数 match_recognize() を，C 言語を用いて作成している．関数 match_recognize() は，入力テーブルを受け取り，行パターンマッチングを行い，その結果のパターンオカレンステーブルを返す関数である．関数 match_recognize() の引数と戻り値を図 3.9 に示す．AS 句以降のカッコ内が戻り値である．関数 match_recognize() は，4 つの引数と複数の戻り値を有する．引数は，1 つ目が入力テーブルを取得するための SQL 文である．2 つ目がメジャーの定義である．MEASURES 句に定義するものに相当する．3 つ目が正規表現を用いたパターンである．PATTERN 句に定義するものに相当する．4 つ目がパターン変数に対する条件である．DEFINE 句に定義するものに相当する．戻り値は，パターンオカレンステーブルの各属性とそれぞれの型である．Example Query を関数 match_recognize() を用いたクエリに書き換えたものを図 3.10 に示す．メジャーの定義やパターン変数に対する条件の定義においては属性名ではなく各属性を識別する属性番号を用いている．関数内部では属性番号により各属性を識別するためである．

関数 match_recognize() は，その内部において，引数に指定した PATTERN 句，DEFINE 句，MEASURES 句にあたる情報から，NFA を構築し，行パターンマッチングを実行する．NFA においては，パターン変数が各状態に対応し，正規表現が状態間を遷移するエッジとなる．状態間を遷移する条件は，パターン変数に対する条件が対応する．行パターンマッチングにおける NFA では，各状態にマッチした行を保持しておくバッファが存在する．受容状態まで達した時，メジャーとして指定した属性の値を各バッファ

```
SELECT  *
FROM  match_recognize  (
   ‘ SELECT row_number() OVER (ORDER BY person_id, time) AS rid
              person_id, time, location
     FROM moving_table ’ ,
   ‘ X.2 AS X_time, X.3 AS X_location, Z.2 AS Z_time, Z.3 AS Z_location ’ ,
   ‘ ( X Y+ Z ) ’ ,
   ‘ X AS X.3 = “A”, Y AS COUNT(Y.*) <= 2, Z AS Z.3 = “C” ’ )
   AS ( person_id int, X_time timestamp, X_location char,
        Z_time timestamp, Z_location char) ;
```

図 3.10 Example Query に対応する関数 match_recognize() を用いたクエリ

から参照し，パターンオカレンステーブルの 1 行として生成する．また，このバッファにより，遷移の条件に今までマッチした行の属性の値の参照が可能となる．

図 3.11 に，Example Query に対応する NFA を示す．図 3.11 の右上は，入力データである moving_table の 5 行目のデータが入力され，受容状態に到達し，パターンにマッチした状態を表している．MEASURES 句に指定したメジャーの属性として，状態 X の time と location，および状態 Z の time と location を指定しているので，状態 X と Z のバッファから対応する行の値を参照し，パターンオカレンステーブルの 1 行を生成する．moving_table の 6 行目のデータから行パターンマッチングを再開する際，各状態のバッファはリセットされる．

3.7.2 PostgreSQL における最適化手法の適用

本研究における最適化手法を適用する場合は，図 3.9 に示した関数 match_recognize() を含むクエリの前に WITH 句や VIEW を用いて Filtering Query を入力テーブルに対して適用するように書き換える．本研究では，Filtering Query を入力テーブルに対して適用した結果の VIEW を作成し，それに対し関数 match_recognize() を含むクエリを適用する．図 3.12 に，図 3.10 に Sequence Filtering Query を適用する例を示す．なお，本研究では，最適化手法を適用するクエリの書き換えはハンドオプティマイズにより実現している．

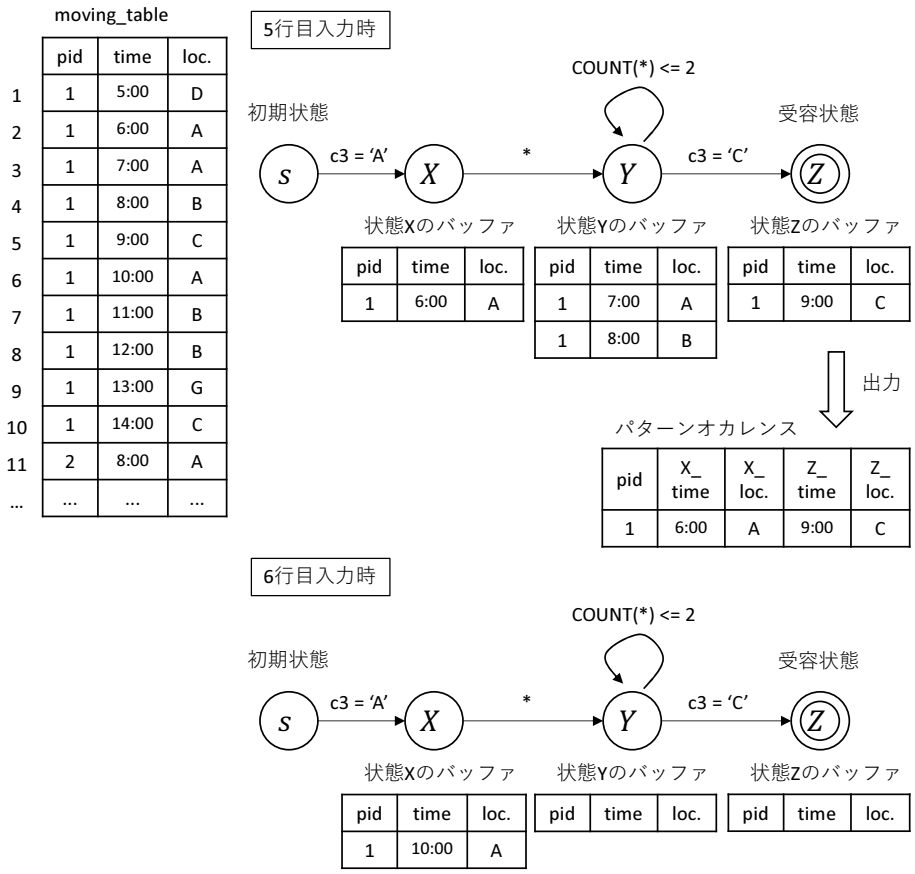

図 3.11 Example Query に対応する NFA

3.8 Spark における行パターンマッチング

本研究では，SQL/RPR が Spark に実装されることを想定している．そこで，Spark への SQL/RPR の実装手法と，SQL/RPR を実装した Spark における，3.5 節において説明した最適化手法の適用方法を説明する．

3.8.1 SQL/RPR を実装する Spark のアーキテクチャ

Spark SQL は Spark 上で SQL ライクなクエリを用いてデータ分析を行うコードを記述するモジュールである．本研究では，MATCH_RECOGNIZE 句を含んだ Spark SQL のクエリを実行できるシステムアーキテクチャを考える．このアプローチには図 3.13 のように，2 種類の方法が考えられる．

1 つ目として，図 3.13 (a) に示すような，現状の Spark SQL の内部のコードを

```
CREATE VIEW view AS
   SELECT *
   FROM ( SELECT DISTINCT person_id
            FROM moving_table
            WHERE location = 'A' OR location = 'C'
          ) AS moving_table2
   JOIN moving_table ON moving_table2.person_id = moving_table.person_id ;
SELECT  *
FROM  match_recognize  (
   ' SELECT row_number() OVER (ORDER BY person_id, time) AS rid
              person_id, time, location
     FROM view ' ,
   ' X.2 AS X_time, X.3 AS X_location, Z.2 AS Z_time, Z.3 AS Z_location ' ,
   ' ( X Y+ Z ) ' ,
   ' X AS X.3 = "A", Y AS COUNT(Y.*) <= 2, Z AS Z.3 = "C" ' )
   AS ( person_id int, X_time timestamp, X_location char,
        Z_time timestamp, Z_location char) ;
```

図 3.12 Example Query に Sequence Filtering Query を適用したクエリ

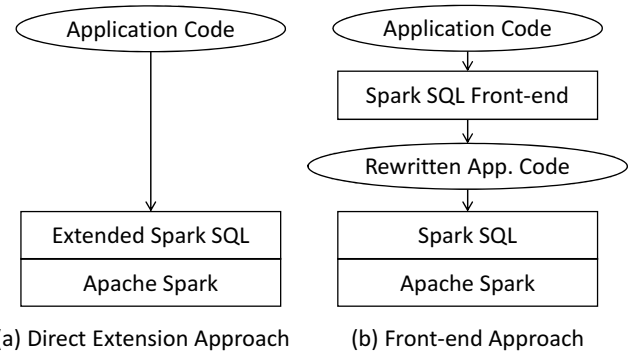

図 3.13 MATCH_RECOGNIZE 句を実装するアーキテクチャ

SQL/RPR が実行できるように直接書き換える方法が考えられる．これを，*Direct Extension Approach* と呼ぶ．2 つ目として，図 3.13 (b) に示すような，ユーザが記述した Spark のコードを解析し，MATCH_RECOGNIZE 句を含むクエリのコードを，通常の Spark でも処理が可能なように書き換える，フロントエンドのモジュールを用意する方法が考えられる．これを，*Front-end Approach* と呼ぶ．

本研究では，現状の Spark SQL の内部コードを変更することなく容易に実装が可能な，Front-end Approach で実装する．なお，3.5 節で提案した行パターンマッチングの最適化手法は，いずれのアプローチにも適用可能なものである．

3.8.2 MATCH_RECOGNIZE 句を含む Spark SQL のクエリ書き換え

フロントエンドモジュールによる，Spark のコードの書き換えの流れを説明する．Spark では，データを *dataframe* と呼ばれるコレクションの形で保持し，ユーザは，この dataframe に対して処理を記述する．本研究では前提として，この dataframe を spark の内部関数 createOrReplaceTempView() により SQL のクエリ内で用いるテーブル名を登録し，MATCH_RECOGNIZE 句を含むクエリ文字列を変数 *query* に格納し，この *query* を Spark の内部関数 sql() により実行する 3 つの処理が含まれたコードを対象に書き換えを行う．つまり，下記の処理が含まれるコードに対して書き換えを行う．

- <入力 dataframe 名>.createOrReplaceTempView(“<入力テーブル名>”)
- query = “<MATCH_RECOGNIZE 句を含むクエリ文字列>”
- <結果 dataframe 名> = spark.sql(query)

上記の 3 つの処理は，コード内の同一クラス，メソッドに含まれているものとする．また，<MATCH_RECOGNIZE 句を含むクエリ文字列>においては，FROM 句に関数 createOrReplaceTempView() により登録した入力テーブル名を指定するものとし，クエリ文字列の内部において，コード内の変数がバインドされておらず，固定の文字列が指定されているものとする．なお，ここでは説明のしやすさのため，クエリ文字列を変数 *query* に格納し，*query* を関数 sql() により実行しているが，関数 sql() の引数として直接クエリ文字列を指定する場合にも対応は可能である．

以降に，書き換えの流れを，Algorithm 1 に示す．また，コードの書き換えの例として，図 3.14 の上のコードの書き換えを例に説明する．なお，本研究においては，Java を用いて Spark における SQL/RPR を実装している．以降では，Java のコードをベースに説明するが，書き換えのアルゴリズムは，特定のプログラミング言語に依存するものではない．

- 1 行目から 5 行目にかけ，フロントエンドモジュールが，Spark の Application Code をスキャン．
- 2 行目において，MATCH_RECOGNIZE 句を含むクエリを見つけた場合，そのクエリを変数 q に格納．
- 3 行目において，関数 makeCode() により，q のクエリの処理に対応する Spark で実行可能なコードを生成．

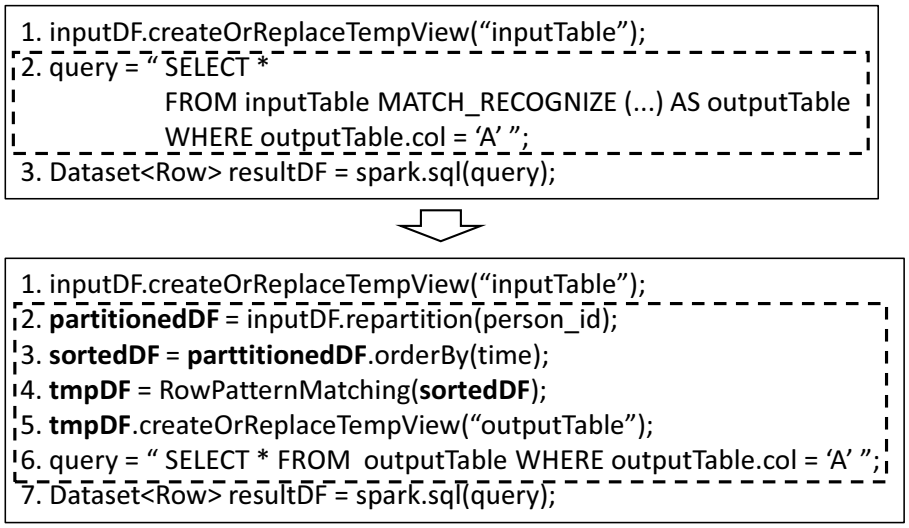

図 3.14 コードの書き換えの例

Algorithm 1 コードの書き換えの全体の流れ

Input: Application Code c

Output: Rewritten App. Code

1: **while** *scan* c **do**
2: $q \leftarrow$MATCH_RECOGNIZE 句を含むクエリ
3: $code \leftarrow makeCode(q)$
4: $code$ を c における抽出したクエリの部分に挿入
5: **end while**
6: **return** c

- 4 行目において，生成したコードを元コードの MATCH_RECOGNIZE 句を含むクエリの部分に挿入．

図 3.14 の上における 2 行目の点線で囲んだ部分のコードが，Algorithm 1 における変数 q に格納される．また，図 3.14 の下における 2 行目から 6 行目の点線で囲んだ部分のコードが，関数 makeCode() により生成されたコードである．なお，太字で示した変数名は，フロントエンドモジュールにより自動的に割り当てられる．

次に，関数 makeCode() の処理を説明する．Algorithm 2 に，関数 makeCode() の処理を示す．

Algorithm 2 関数 makeCode() の処理の流れ

```
1: function makeCode(q)
2:     code ← null
3:     mr ← q 内の MATCH_RECOGNIZE specification
4:     関数 execRPR(mr) により作成されたコードを code に追加
5:     q から MATCH_RECOGNIZE specification を除いたクエリを code に追加
6:     return code
7: end function
```

- 2 行目において，関数 makeCode() で生成するコードを保持する変数 *code* を用意.
- 3 行目において，q のクエリから MATCH_RECOGNIZE specification を抽出.
- 4 行目において，MATCH_RECOGNIZE specification に対応する行パターンマッチングを行うコードを関数 execRPR() において生成し，*code* に追加.
- 5 行目において，q のクエリから MATCH_RECOGNIZE specification を除いたクエリを *code* に追加.
- 6 行目において，*code* を返す.

図 3.14 の下における 2 行目から 4 行目のコードが，Algorithm 2 の 4 行目における関数 execRPR() によって生成された，行パターンマッチングを行うコードに対応する．また，図 3.14 の下における 5 行目と 6 行目のコードが，Algorithm 2 の 5 行目において追加された，q のクエリから MATCH_RECOGNIZE specification を除いたクエリを実行するコードに対応する.

次に，関数 execRPR() の処理を説明する．Algorithm 3 に，関数 execRPR() の処理を示す.

- 2 行目において，関数 execRPR() で生成するコードを保持する変数 *code* を用意.
- 3 行目において，MATCH_RECOGNIZE specification 内の各句を分割.
- 4 行目において，PARTITION BY 句に指定した sequence_id を用いてデータをパーティショニングするコードを *code* に追加.
- 5 行目において，ORDER BY 句に指定した order_key を用いてデータをソートするコードを *code* に追加.
- 6，7 行目において，PATTERN 句，DEFINE 句，SUBSET 句，MEASURES 句

Algorithm 3 関数 execRPR() の処理の流れ

1: **function** execRPR(q)
2: $code \leftarrow null$
3: $clause[] \leftarrow mr$ を句毎に分割
4: $clause[partitionBy]$ を元にパーティショニングするコードを $code$ に追加
5: $clause[orderBy]$ を元にソートするコードを $code$ に追加
6: $clause[pattern], clause[define], clause[subset], clause[measures]$ を元に NFA を構築
7: NFA を利用しパターンマッチングを行うコードを $code$ に追加
8: **return** $code$
9: **end function**

に指定した情報から，NFA を構築し，行パターンマッチングを実行するコードを $code$ に追加.

- 8 行目において，$code$ を返す.

図 3.14 の下の 2 行目のコードが Algorithm 3 の 4 行目，図 3.14 の下の 3 行目のコードが Algorithm 3 の 5 行目，図 3.14 の下の 4 行目のコードが Algorithm 3 の 6，7 行目において作成したコードに対応する．この，NFA の構築と行パターンマッチングの実行部分は，図 3.14 の下の 4 行目における，関数 RowPatternMatching() により実現する．関数 RowPatternMatching() は，本研究において独自実装するものである．コードの書き換えにおいては，Spark の内部関数と関数 RowPatternMatching() を用意することで，MATCH_RECOGNIZE 句を含むクエリを実行するコードが，本節で説明した変換器を作成することで，Spark 上で実行可能なコードに変換可能である．

関数 RowPatternMatching() の内部は，3.7.1 節において説明した，PostgreSQL における UDF（関数 match_recognize()）と同じ仕組みで実装している．PATTERN 句，DEFINE 句，SUBSET 句，MEASURES 句に指定した情報から，NFA を構築し，行パターンマッチングを実行する．

3.8.3 MATCH_RECOGNIZE 句の処理の Spark 実行時におけるデータフロー

図 3.15 に，MATCH_RECOGNIZE 句を含むクエリを Spark 上で実行する際のデータフローを示す．dataframe は，各ノード上にパーティション単位で分割され配置されて

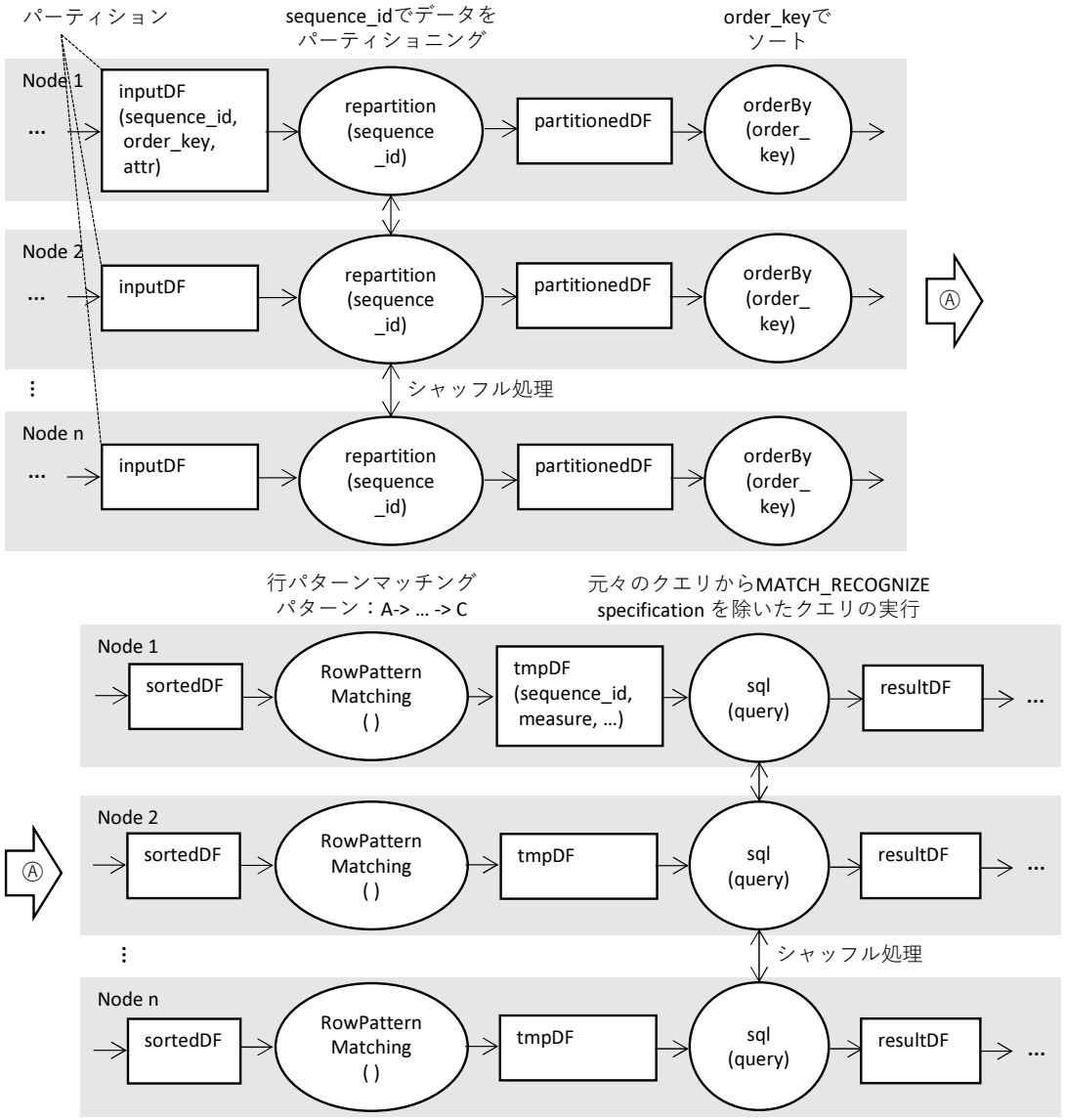

図 3.15 Spark 実行時のデータフロー

いる．ユーザが特に指定をしない限り，Spark がデータを等サイズのパーティションに分割する．図 3.15 では，inputDF という dataframe に対し，行パターンマッチングを行い，resultDF という dataframe を作成する流れを示している．MATCH_RECOGNIZE 句を含むクエリは，行パターンマッチングの処理と MATCH_RECOGNIZE 句を除いた部分のクエリの処理からなる．まず，inputDF に対し，行パターンマッチングの処理を行う．Algorithm 3 で示したように，まず，inputDF を sequence_id でパーティショニングする．この際，ノード間でシャッフル処理が行われる．次に，order_key によっ

Algorithm 4 最適化を含む関数 makeCode() の処理

```
1: function makeCode(q)
2:     code ← null
3:     mr ← q 内の MATCH_RECOGNIZE specification
4:     if mr に subexpression が存在 then
5:         subexpression を実行するコードを code に追加
6:     end if
7:     関数 optCode(mr) により作成されたコードを code に追加
8:     関数 execRPR(mr) により作成されたコードを code に追加
9:     q から MATCH_RECOGNIZE specification を除いたクエリを code に追加
10:    return code
11: end function
```

てソートする．そして，ソートされたデータに対し，行パターンマッチングを行う．最後に，MATCH_RECOGNIZE specification を除いたクエリを実行し，その結果として resultDF を得る．

3.8.4 最適化を含む Spark SQL の書き換え

最適化を含む Spark のコードの書き換えの流れを説明する．最適化を含まない場合の書き換えとは，Algorithm 2 の関数 makeCode() の処理が異なる．これを Algorithm 4 に示す．7 行目において，最適化を行うコードを生成する関数 optCode() が，新たに行パターンマッチングの前に *code* に追加されている．

関数 optCode() の処理を説明する．Algorithm 5 に，関数 optCode() の処理を示す．

- 2 行目において，関数 optCode() で生成するコードを保持する変数 *code* を用意.
- 3 行目において，MATCH_RECOGNIZE specification 内の各句を分割.
- 4 行目において，DEFINE 句に指定した one variable condition から Filtering Query を作成.
- 5 行目において，Filtering Query を実行するコードを *code* に追加.
- 6 行目において，*code* を返す.

図 3.16 に，図 3.14 の上のコードを，最適化を含んだコードに書き換えた例を示す．

Algorithm 5 関数 optCode() の処理

```
1: function optCode(mr)
2:     code ← null
3:     clause[] ← mr を句毎に分割
4:     fq ← clause[define] を元に Filtering Query を作成.
5:     fq を実行するコードを code に追加
6:     return code
7: end function
```

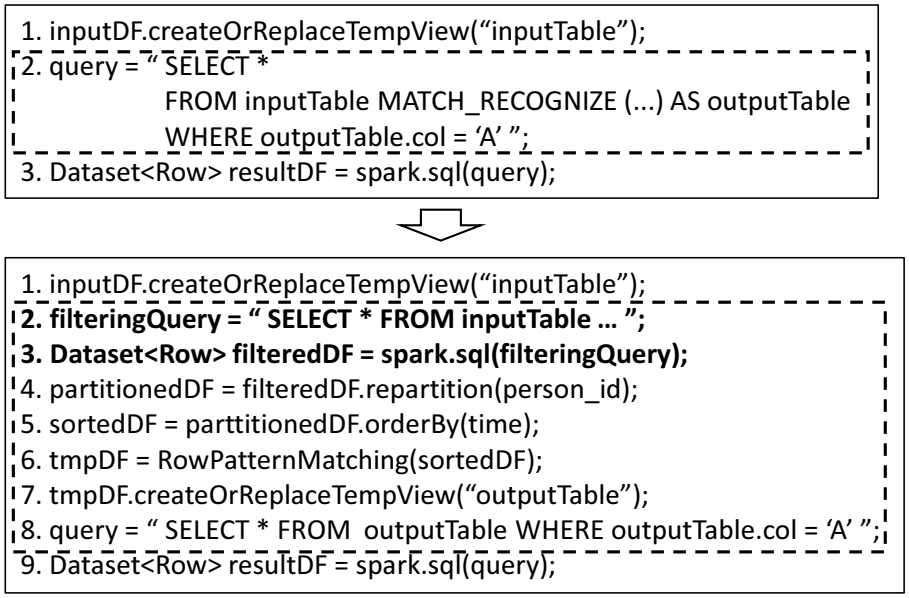

図 3.16 最適化を含むコードの書き換えの例

最適化を含まない図 3.14 の下のコードに対し，2，3 行目（太字で示している）が追加される．2 行目が最適化を行うクエリであり，3 行目において，そのクエリを実行する．Algorithm 4 における，7 行目の関数 optCode() によって生成されるコードに対応する．この結果のテーブルが，図 3.14 の下のコードの 2 行目におけるパーティショニングする dataframe となる．

図 3.17 に，最適化を含む場合のデータフローをソート処理まで示す．それ以降のフローは図 3.15 と同一である．データのパーティショニングの前に，Filtering Query の処理（太線で示している）が新たに追加される．

本研究では，コード書き換えの処理自体はハンドオプティマイズにより実現している．

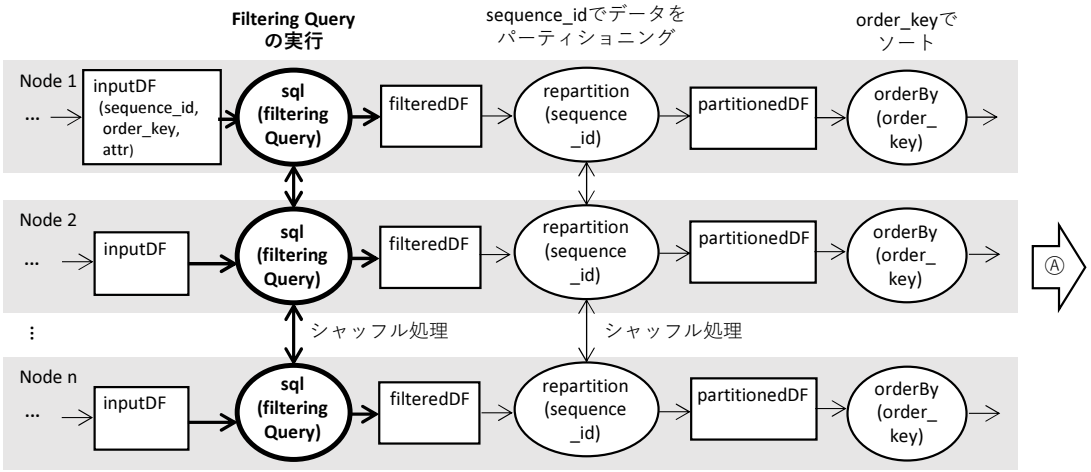

図 3.17 最適化を含む場合の処理のデータフロー

なお，最適化のコストには，Filtering Query の実行にかかるコストに加え，Filtering Query 作成のための処理コスト分のオーバーヘッドも考えられるが，Algorithm 5 に示したように，MATCH_RECOGNIZE 句を含むクエリをパースし，DEFINE 句に指定した one variable condition を抽出し，Filtering Query の文字列を作成する処理である．そのため，この処理自体の処理コストは Filtering Query の実行にかかるコストと比較し小さいと考え，ハンドオプティマイズによる実装であっても，本研究において提案する手法の効果はあると考える．Filtering Query 作成を含むコード書き換えの処理の実装に伴う処理のオーバーヘッドの分析は，今後の課題とする．

3.9 評価実験

3.9.1 人工データを用いた実験

本実験では，3.5.2 節の最後に示した 4 通りの最適化について，UDF により SQL/RPR を実装した PostgreSQL と，SQL/RPR を実装した拡張 Spark を対象に，人工データを用いて検証する．本実験では，パターンの異なる Q1 から Q6 の 6 種類のクエリを用いる．これらのクエリを，図 3.18 から図 3.23 に示す．Q1 はシーケンシャルなパターン，Q2 は繰り返しを表す正規表現を含むパターン，Q3 および Q4 は選択を表す正規表現を含むパターン，Q5 および Q6 は繰り返しや選択を表す正規表現が複合的に含まれているパターンである．

本実験において入力テーブルとして用いる人工データの属性，行数，シーケンス数は，

```
SELECT *
FROM test_table
MATCH_RECOGNIZE ( PARTITION BY c1
                  ORDER BY c2
                  MEASURES X.c2 AS X_c2, Y.c2 AS Y_c2, Z.c2 AS Z_c2
                  ONE ROW PER MATCH
                  PATTERN ( X Y Z )
                  DEFINE X AS X.c3 = 'A' )
```

図 3.18 問合せ Q1

```
SELECT *
FROM test_table
MATCH_RECOGNIZE ( PARTITION BY c1
                  ORDER BY c2
                  MEASURES X.c2 AS X_c2, Z.c2 AS Z_c2
                  ONE ROW PER MATCH
                  PATTERN ( X Y* Z )
                  DEFINE X AS X.c3 = 'A',
                         Y AS COUNT(Y.*) <= 2,
                         Z AS Z.c3 = 'C' )
```

図 3.19 問合せ Q2

```
SELECT *
FROM test_table
MATCH_RECOGNIZE ( PARTITION BY c1
                  ORDER BY c2
                  MEASURES Z.c2 AS Z_c2
                  ONE ROW PER MATCH
                  PATTERN ( ( X | Y ) Z )
                  DEFINE X AS X.c3 = 'A', Y AS Y.c3 = 'B' )
```

図 3.20 問合せ Q3

```
SELECT *
FROM test_table
MATCH_RECOGNIZE ( PARTITION BY c1
                  ORDER BY c2
                  MEASURES Z.c2 AS Z_c2
                  ONE ROW PER MATCH
                  PATTERN ( ( X | Y ) ( Z | W ) )
                  DEFINE X AS X.c3 = 'A', Y AS Y.c3 = 'B',
                         Y AS Y.c3 = 'C', W AS W.c3 = 'D' )
```

図 3.21 問合せ Q4

```
SELECT *
FROM test_table
MATCH_RECOGNIZE ( PARTITION BY c1
                  ORDER BY c2
                  MEASURES W.c2 AS W_c2
                  ONE ROW PER MATCH
                  PATTERN ( X+ ( Y | Z ) W )
                  DEFINE X AS COUNT(X.*) <= 3,
                         Y AS Y.c3 = 'B',
                         Z AS Z.c3 = 'C' )
```

図 3.22 問合せ Q5

```
SELECT *
FROM test_table
MATCH_RECOGNIZE ( PARTITION BY c1
                  ORDER BY c2
                  MEASURES X.c2 AS X_c2
                  ONE ROW PER MATCH
                  PATTERN ( X ( Y | Z W )* )
                  DEFINE X AS X.c3 = 'A',
                         Y AS Y.c3 = 'B' AND COUNT(Y.*) <= 2,
                         Z AS Z.c3 = 'C' AND COUNT(Z.*) <= 2,
                         W AS COUNT(W.*) <= 2 )
```

図 3.23 問合せ Q6

3.4 節で用いたものと同一である．また，表 3.2 に示した，パラメータ α, β については，その組合せを (α_i, β_i) とし，7 種類の組合せを用いる $(i = 1, ..., 7)$．これらを Configuration 1 から 7 とし，それぞれ，Configuration 1: $(\alpha_1, \beta_1) = (0.0, 0.0)$，Configuration 2: $(\alpha_2, \beta_2) = (0.2, 0.2)$, Configuration 3: $(\alpha_3, \beta_3) = (0.1, 0.9)$, Configuration 4: $(\alpha_4, \beta_4) = (0.2, 0.8)$, Configuration 5: $(\alpha_5, \beta_5) = (0.8, 0.2)$, Configuration 6: $(\alpha_6, \beta_6) = (0.8, 0.8)$, Configuration 7: $(\alpha_7, \beta_7) = (1.0, 1.0)$ とする．なお，実験環境は，PostgreSQL については，Ubuntu 16.04 LTS, Intel®Core™i9-7920X CPU @2.90GHz, 128GB RAM．Spark については，3.4 節で示した実験環境と同一である．

まず，PostgreSQL を用いた人工データに対する実験結果を図 3.24 から図 3.29 に示す．なお，具体的な値は付録 A.1 の表 A.1 から表 A.6 に示す．本節において示す実験結果はいずれも，処理時間は 5 回の試行の平均処理時間である．また，各棒グラフには，5 回の試行の処理時間の標準偏差を平均処理時間から正負の方向にエラーバーで示している．Sequence Filtering によって削減されるシーケンスが多い，Configuration 1 から 4 については，Sequence Filtering による処理時間の削減効果が得られた．Row Filtering については，Q2 と Q6 を除き，削減される行数が多い，Configuration 1 から 5 につ

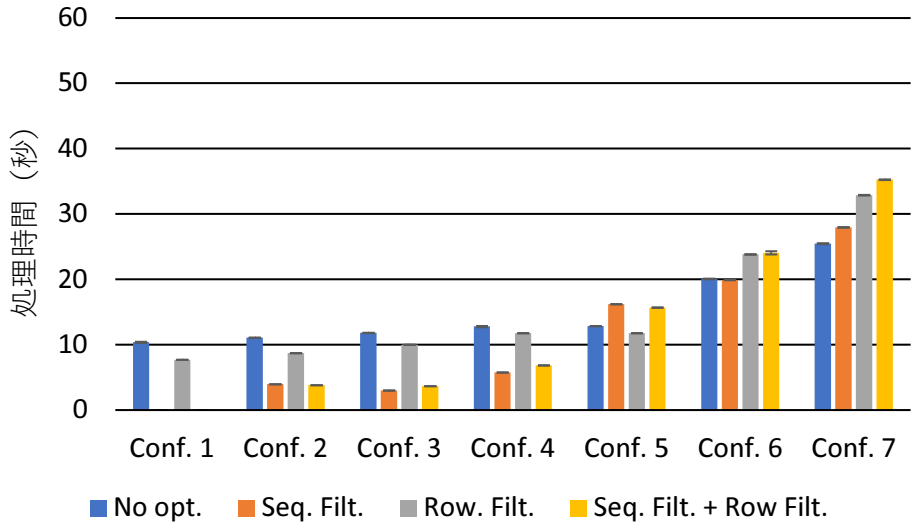

図 3.24 PostgreSQL を用いた人工データに対する実験結果 (Q1)

いて，Row Filtering による処理時間の削減効果が得られた．特に，Sequence Filtering によって削減されるシーケンスが少ない一方で Row Filtering によって削減される行数が多い Configuration 5 においては，Q6 を除き，Row Filtering の方が，Sequence Filtering よりも処理時間が削減された．なお，Q2 と Q6 については，Window 演算における Window 幅が広く設定されているため，Row Filtering の処理コストが大きくなり，No Optimization に対し，処理時間の削減効果が得られなかったと考えられる．Q5 についても Window 幅が広く設定されているが，パターンの先頭に条件指定がされていない繰り返しの正規表現を含むパターンが指定されているため，行パターンマッチングにおいて参照する行が増えることにより処理コストが大きくなったため，Row Filtering のオーバーヘッドが影響せず，処理時間の削減効果が得られたと考えられる．Sequence Filtering + Row Filtering については，Q2 と Q6 を除く Configuration 2 において，処理時間が最も短くなった．

次に，Spark を用いた人工データに対する実験結果を図 3.30 から図 3.35 に示す．なお，具体的な値は付録 A.1 の表 A.7 から表 A.12 に示す．また，Q4 の Configuration 7 については，いずれの最適化手法においてもメモリ不足により処理が完了できないため，結果を示していない．フィルタリングされる行が 1 行も存在しない Configuration 7 を除き，全ての最適化手法について，処理時間の削減効果が得られている．全ての行がフィルタリングされる Configuration 1 については，Sequence Filtering が最も処理時間が短くなった．一方で，Sequence Filtering によって削減されるシーケンスの割合が 0.2 よりも大き

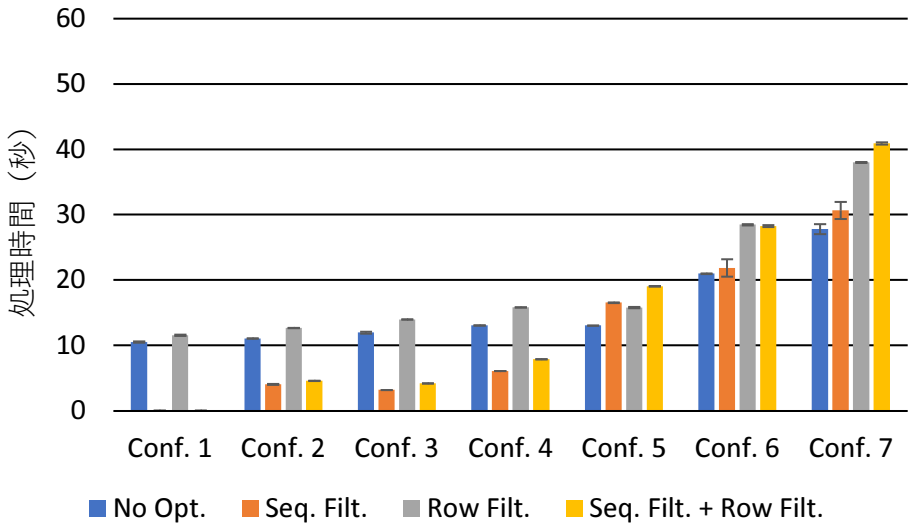

図 3.25 PostgreSQL を用いた人工データに対する実験結果 (Q2)

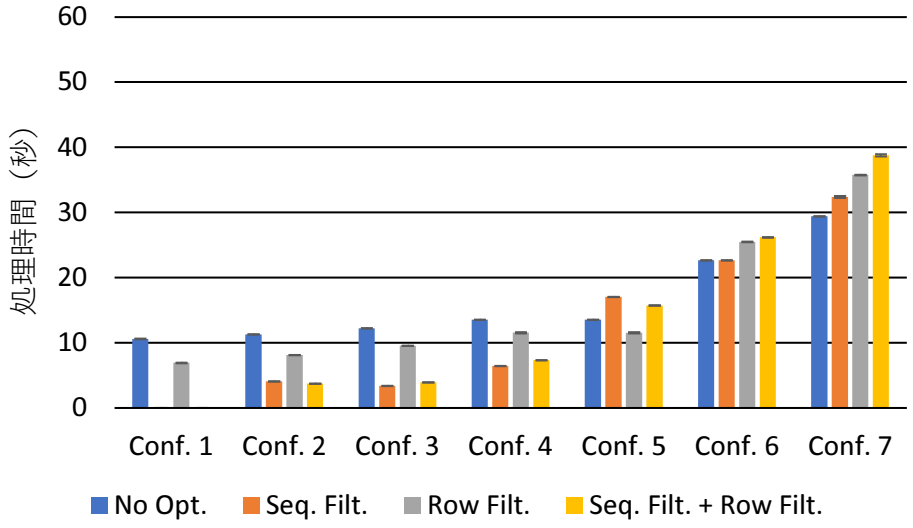

図 3.26 PostgreSQL を用いた人工データに対する実験結果 (Q3)

い，Configuration 2，4，5，6 については，Row Filtering あるいは，Sequence Filtering + Row Filtering が最も処理時間が短くなった．特に，フィルタリングされる行数が最も多い Configuration 2 については，Q3，Q5，Q6 において，Sequence Filtering + Row Filtering が最も処理時間が短くなった．なお，最も処理時間の削減効果が大きかったものは，Q4 の Configuration 1 における Sequence Filtering で，No Optimization に対し，86.57% 削減された．

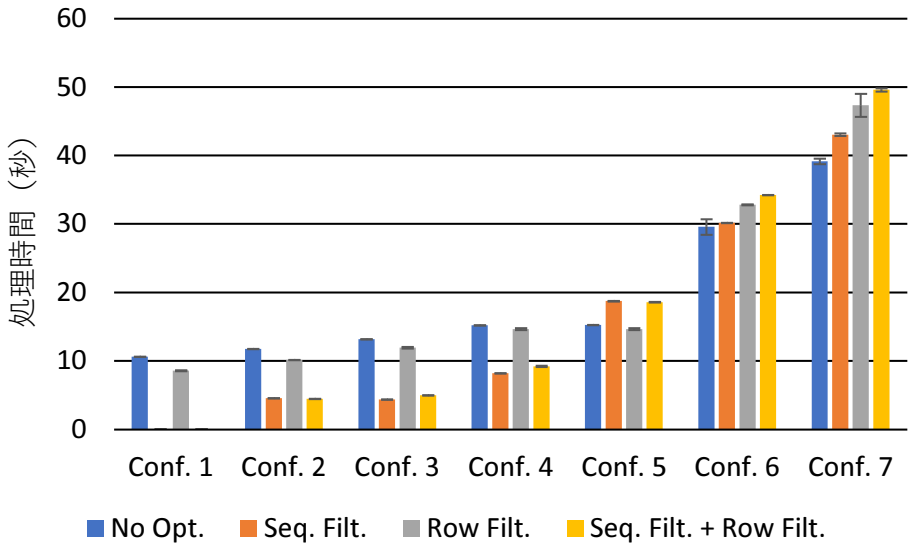

図 3.27 PostgreSQL を用いた人工データに対する実験結果 (Q4)

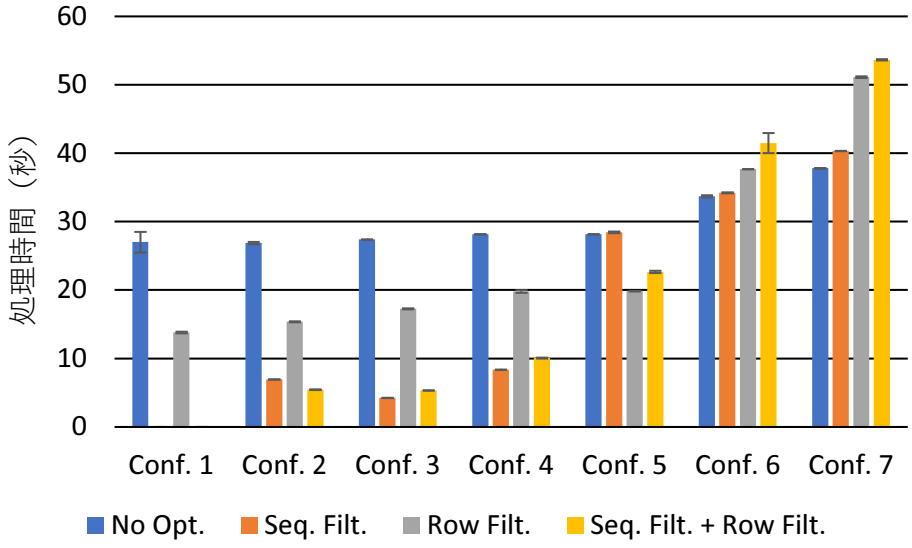

図 3.28 PostgreSQL を用いた人工データに対する実験結果 (Q5)

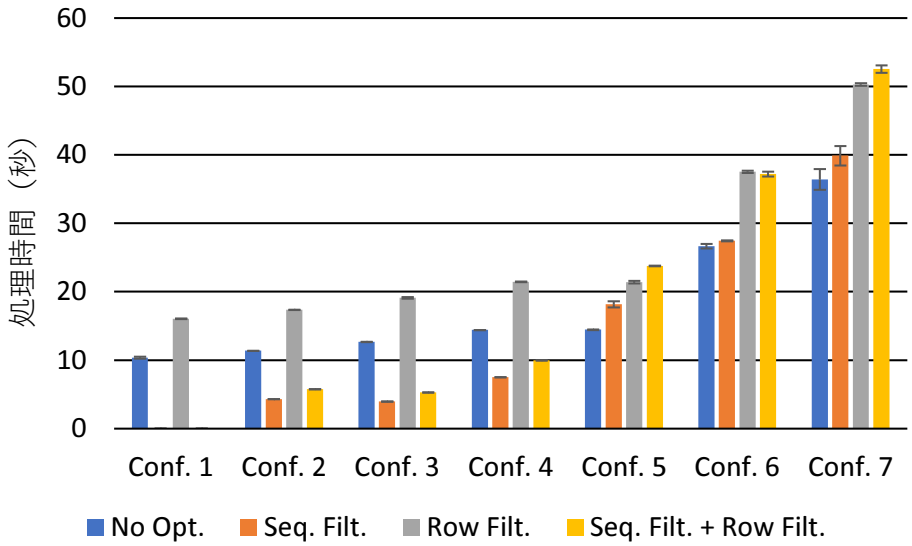

図 3.29 PostgreSQL を用いた人工データに対する実験結果 (Q6)

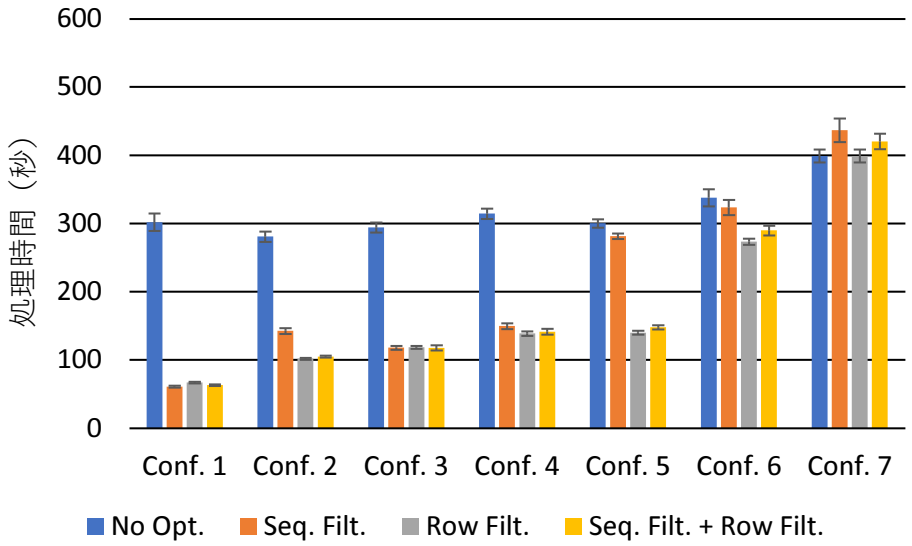

図 3.30 Spark を用いた人工データに対する実験結果 (Q1)

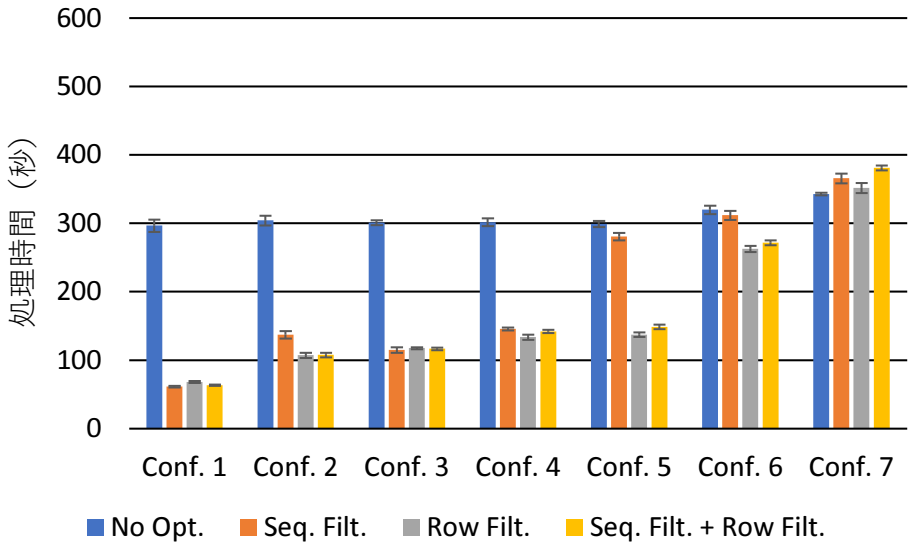

図 3.31 Spark を用いた人工データに対する実験結果 (Q2)

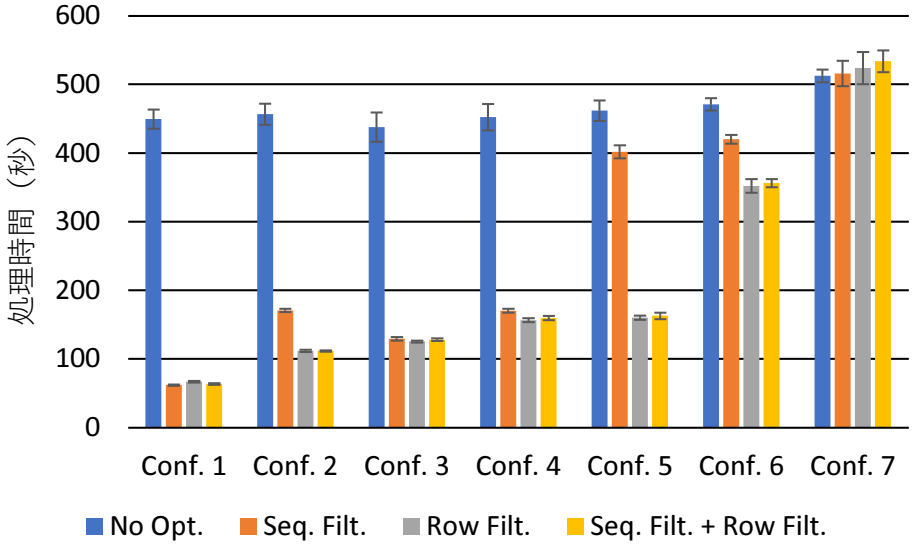

図 3.32 Spark を用いた人工データに対する実験結果 (Q3)

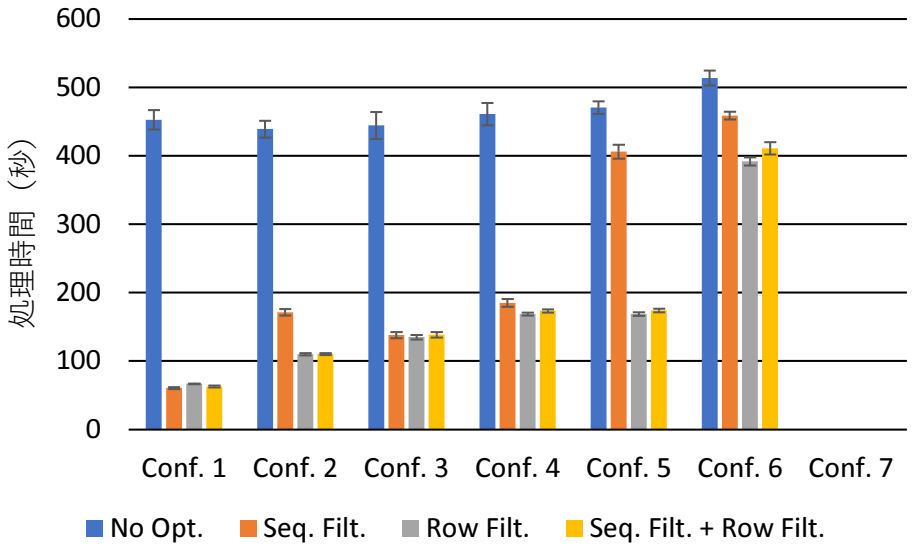

図 3.33 Spark を用いた人工データに対する実験結果 (Q4)

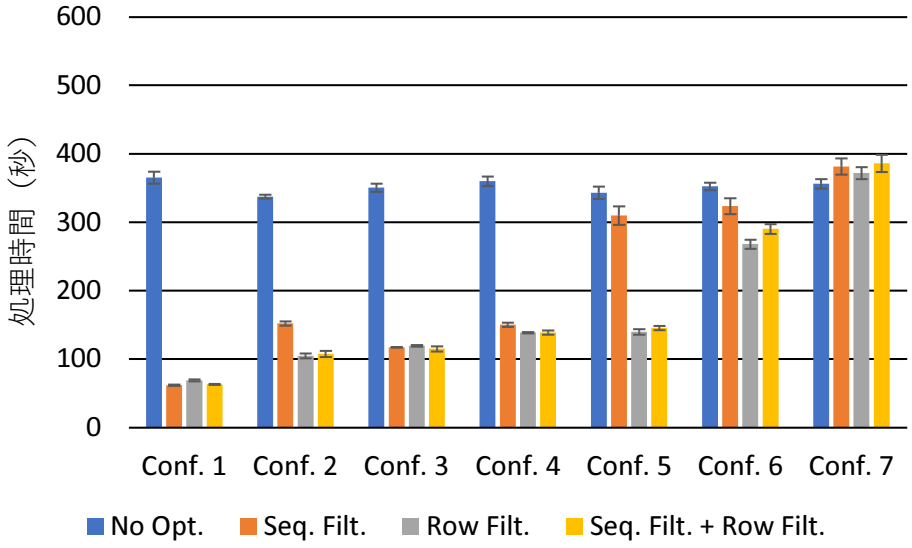

図 3.34 Spark を用いた人工データに対する実験結果 (Q5)

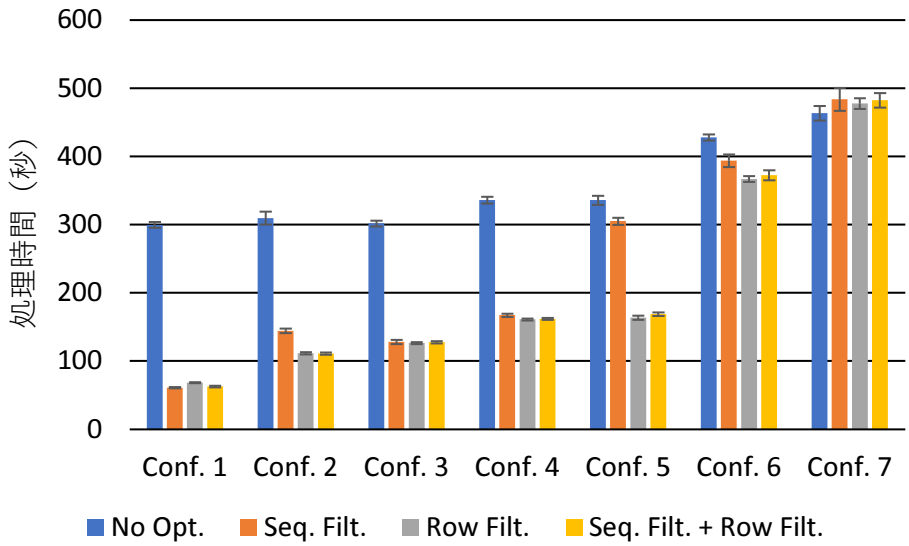

図 3.35 Spark を用いた人工データに対する実験結果 (Q6)

```
SELECT *
FROM dataset_TIST2015_Checkins JOIN dataset_TIST2015_POIs
        ON dataset_TIST2015_Checkins.Venue_ID = dataset_TIST2015_POIs.Venue_ID
MATCH_RECOGNIZE ( PARTITION BY User_ID
                          ORDER BY UTC_time
                          MEASURES X.UTC_time AS X_UTC_time,
                                   Y.UTC_time AS Y_UTC_time
                          ONE ROW PER MATCH
                          PATTERN ( X Y )
                          DEFINE X AS X.Venue_category_name = 'Home (private)',
                                 Y AS Y.Venue_category_name = 'Office' )
```

図 3.36 問合せ Q7

3.9.2 実データを用いた実験

次に，実データとして Foursquare Dataset を用い，最適化の効果を検証する．Foursquare[33] は，位置情報を活用したソーシャルネットワーキングサービスである．ユーザは携帯端末を用いて，Venue と呼ばれる特定の地点において Checkin をすることで，アプリケーション内において報酬を受け取ることができる．本実験は，Foursquare Dataset として公開されているものの中で，Global-scale Check-in Dataset[110][111] を用いる．このデータセットは，Foursquare を通じて集められた約 18 ヶ月分の全世界におけるチェックインデータである．使用するデータは，dataset_TIST2015_Checkins および dataset_TIST2015_POIs である．dataset_TIST2015_Checkins は，Checkin を集めたデータセットであり，属性として (User ID, Venue ID, UTC time, Timezone offset in minutes) の 4 つを持つ．本実験では，User ID を sequence_id，UTC time を order_key として扱う．なお，行数は，33,263,633 行，シーケンス数は，266,909 である．dataset_TIST2015_POIs は，Venue のマスタデータであり，属性として，(Venue ID, Latitude, Longitude, Venue category name, Country code) の 5 つを持つ．

本実験では，パターンの異なる Q7 から Q12 の 6 種類のクエリを用いる．これらのクエリを，図 3.36 から図 3.41 に示す．Q7 はシーケンシャルなパターン，Q8 および Q9 は繰り返しを表す正規表現を含むパターン，Q10 は選択を表す正規表現を含むパターン，Q11 および Q12 は繰り返しや選択を表す正規表現が複合的に含まれているパターンである．各クエリに指定した条件から，Sequence Filtering によって残るシーケンスの割合 α と，Sequence Filtering によって残るシーケンスの中で，さらに Row Filtering によって残る行の割合 β を表 3.3 に示す．Q7 から Q9 は，α の値が高いクエリとなっている．また，Q7，Q8，Q11 は，β の値が高いクエリとなっている．

```
SELECT *
FROM dataset_TIST2015_Checkins JOIN dataset_TIST2015_POIs
        ON dataset_TIST2015_Checkins.Venue_ID = dataset_TIST2015_POIs.Venue_ID
MATCH_RECOGNIZE ( PARTITION BY User_ID
                        ORDER BY UTC_time
                        MEASURES X.UTC_time AS X_UTC_time,
                                 Z.UTC_time AS Z_UTC_time
                        ONE ROW PER MATCH
                        PATTERN ( X Y+ Z )
                        DEFINE X AS X.Venue_category_name = 'Home (private)',
                                 Y AS COUNT(Y.*) <= 3,
                                 Z AS Z.Venue_category_name = 'Office' )
```

図 3.37 問合せ Q8

```
SELECT *
FROM dataset_TIST2015_Checkins JOIN dataset_TIST2015_POIs
        ON dataset_TIST2015_Checkins.Venue_ID = dataset_TIST2015_POIs.Venue_ID
MATCH_RECOGNIZE ( PARTITION BY User_ID
                        ORDER BY UTC_time
                        MEASURES X.UTC_time AS X_UTC_time,
                                 Z.UTC_time AS Z_UTC_time
                        ONE ROW PER MATCH
                        PATTERN ( X Y* Z )
                        DEFINE X AS X.Venue_category_name = 'Hospital'
                                        AND X.Country_code = 'FR',
                                 Y AS COUNT(Y.*) <= 3,
                                 Z AS Z.Venue_category_name = 'Restaurant' )
```

図 3.38 問合せ Q9

```
SELECT *
FROM dataset_TIST2015_Checkins JOIN dataset_TIST2015_POIs
        ON dataset_TIST2015_Checkins.Venue_ID = dataset_TIST2015_POIs.Venue_ID
MATCH_RECOGNIZE ( PARTITION BY User_ID
                        ORDER BY UTC_time
                        MEASURES X.UTC_time AS X_UTC_time
                        ONE ROW PER MATCH
                        PATTERN ( X ( Y | Z ) )
                        DEFINE Y AS Y.Venue_category_name = 'Shoe Store'
                                 Z AS Z.Venue_category_name = 'Liquor Store' )
```

図 3.39 問合せ Q10

```
SELECT *
FROM dataset_TIST2015_Checkins JOIN dataset_TIST2015_POIs
        ON dataset_TIST2015_Checkins.Venue_ID = dataset_TIST2015_POIs.Venue_ID
MATCH_RECOGNIZE ( PARTITION BY User_ID
                          ORDER BY UTC_time
                          MEASURES X.UTC_time AS X_UTC_time
                          ONE ROW PER MATCH
                          PATTERN ( ( X | Y )+ Z W )
                          DEFINE X AS X.Country_code = 'ES' AND COUNT(X.*) <= 2
                                   Y AS Y.Country_code = 'PT' AND COUNT(Y.*) <= 2)
```

図 3.40 問合せ Q11

```
SELECT *
FROM dataset_TIST2015_Checkins JOIN dataset_TIST2015_POIs
        ON dataset_TIST2015_Checkins.Venue_ID = dataset_TIST2015_POIs.Venue_ID
MATCH_RECOGNIZE ( PARTITION BY User_ID
                          ORDER BY UTC_time
                          MEASURES X.UTC_time AS X_UTC_time
                          ONE ROW PER MATCH
                          PATTERN ( X Y * ( Z | W ) )
                          DEFINE X AS X.Venue_category_name = 'Airport',
                                   Y AS Y.Country_code = 'DE' AND COUNT(Y.*) <= 3,
                                   Z AS Z.Country_code = 'CH',
                                   W AS W.Country_code = 'FR' )
```

図 3.41 問合せ Q12

表 3.3 Q7 から Q12 における α と β の値

	Q7	Q8	Q9	Q10	Q11	Q12
α	0.70	0.70	0.62	0.08	0.03	0.41
β	0.31	0.52	0.19	0.03	0.31	0.17

まず，PostgreSQL を用いた実験結果を図 3.42 に示す．なお，具体的な値は付録 A.2 の表 A.13 に示す．Sequence Filtering については，削減されるシーケンス数が最も少ない，Q7 と Q8 を除き，処理時間の削減効果が得られた．Row Filtering については，Q7，Q10，Q11 において処理時間の削減効果が得られた．削減される行数が少ない Q8 は処理時間の削減効果が得られていないが，削減される行数が比較的多い Q9 と Q12 においても処理時間の削減効果が得られなかった．これは，Window 演算における Window 幅が大きく，その分，Row Filtering の処理コストが大きくなってしまったためと考えられる．

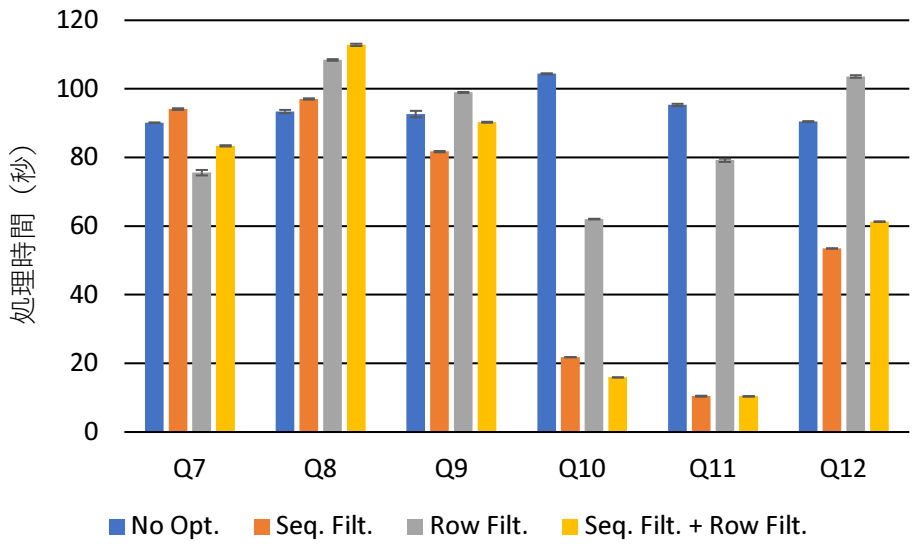

図 3.42 PostgreSQL を用いた Foursquare Dataset に対する実験結果

なお，最大の削減量となったのは，Q11 の Sequence Filtering + Row Filtering で，処理時間が No Optimization に対し 89.05% 削減された．

次に，Spark を用いた実験結果を図 3.43 に示す．なお，具体的な値は付録 A.2 の表 A.14 に示す．全てのクエリにおいて，Sequence Filtering および Row Filtering の処理時間が No Optimization よりも短くなっており，最適化手法による処理時間の削減効果が得られている．Spark においては，Window 演算の処理コストが小さいため，Row Filtering による処理コストの削減効果が大きく得られている．なお，最大の削減量となったのは，Q11 の Row Filtering で，処理時間が No Optimization に対し 85.59% 削減された．

3.9.3 各最適化手法のコストモデルの検証

3.6 節において構築したコストモデルを検証する．具体的には，表 3.2 中の，行数 N，シーケンス数 S，Sequence Filtering によって残るシーケンスの割合 α，Sequence Filtering によって残ったシーケンスの中で，さらに Row Filtering によって残る行の割合 β の値を変えた時の，コストモデルから算出した処理時間（見積値）と，実際の処理時間（実測値）の間で比較を行う．比較対象としては，2 通りの行数 N とシーケンス数 S の組合せ（N = 10,000,000，S = 10,000 と，N = 5,000,000，S = 1,000），2 種類のクエリ（3.9.1 節の人工データに対する評価実験において用いた Q1 および Q5）で，計 4 通りについて検証を行う．

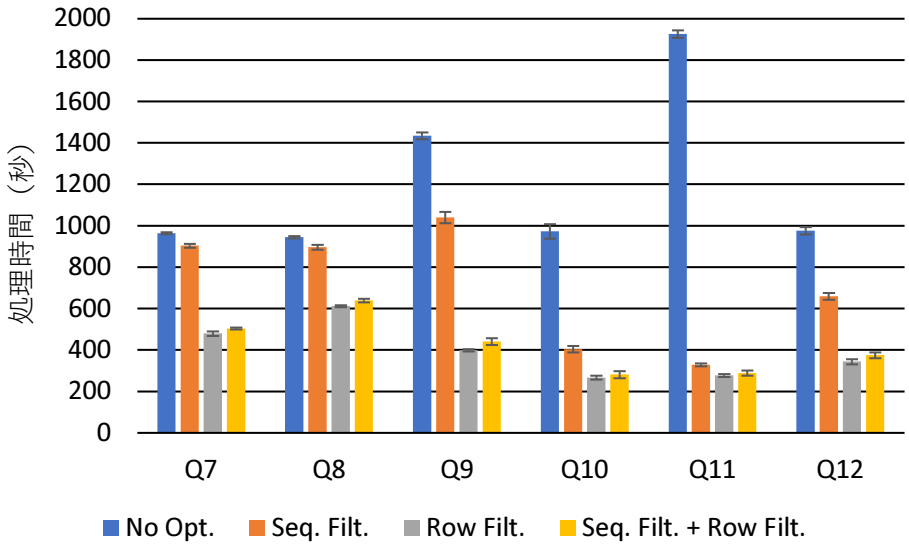

図 3.43 Spark を用いた Foursquare Dataset に対する実験結果

表 3.4 c，r，w の実測値

	PostgreSQL		Spark	
	Q1	Q5	Q1	Q5
c	2.21		14.54	
w	9.96	16.02	15.50	16.26
r (Conf. 1)	10.35	27.00	298.82	348.63
r (Conf. 2)	11.07	26.88	284.46	349.16
r (Conf. 3)	11.79	27.38	299.27	349.62
r (Conf. 4)	12.79	28.17	322.62	351.26
r (Conf. 5)	12.84	28.17	298.35	343.35
r (Conf. 6)	20.06	33.72	341.75	352.45
r (Conf. 7)	25.47	37.80	397.29	365.13

コストの導出に際しては，c，r，w の値が必要となる．そこで，N = 10,000,000，S = 1,000 の時の，シーケンシャルスキャンにかかる処理時間，行パターンマッチングにかかる処理時間，Window 演算にかかる処理時間を実測した．この値を，表 3.4 に示す．この c，r，w の値を利用してコストを見積もり，それぞれの実際の処理時間と比較する．まず，導出に用いた N = 10,000,000，S = 1,000 の時の見積値と実測値を図 3.44 から図 3.47 に示す．なお，これらの具体的な値は付録 A.3 の表 A.15 から表 A.22 に示す．PostgreSQL については，図 3.44 および図 3.45 から，Q1 と Q5 共に，Row Filtering の処理時間が若干長く見積もられているものの，見積値と実測値は概ね一致している．Spark についても，図 3.46 および図 3.47 から，PostgreSQL と同様に，見積値と実測値は概ね一致している．見積値と実測値の差の平均平方二乗誤差 (RMSE) と実測値との相対誤差 (RE) を比較したものを表 3.5 に示す．なお，RMSE および RE は，各最適化手法毎に算出する．算出式は以下の通りである．表 3.5 から，PostgreSQL と Spark のいずれにおいても，概ね 15% から 20% の誤差で処理時間を見積もることができている．

$$RMSE = \sqrt{\frac{\sum \left((\text{見積値}) - (\text{実測値})\right)^2}{(\alpha, \beta\text{の組合せ数})}}$$

$$RE = \frac{RMSE}{\frac{(\text{実測値})}{(\alpha, \beta\text{の組合せ数})}}$$

次に，N および S の値を変えた時の，クエリ Q1 および Q5 における見積値と実測値を比較する．この結果を図 3.48 から図 3.55 に示す．なお，これらの具体的な値は付録 A.3 の表 A.23 から表 A.38 に示す．PostgreSQL，Spark のいずれにおいても，見積値と実測値の各最適化手法間の値の大小の傾向は，N = 10,000,000，S = 1,000 の場合と概ね同じである．

表 3.6 に N = 10,000,000，S = 10,000 の場合，表 3.7 に N = 5,000,000，S = 1,000 の場合における，コストモデルから算出した見積値と実測値の RMSE および RE を示す．表 3.5 に示した，N = 10,000,000，S = 1,000 の場合の RE とほぼ同程度の誤差で処理時間を見積もることができている．N = 5,000,000，S = 1,000 の場合の一部においては，RE が 30% 前後になるものも存在するが，各最適化手法の処理時間の大小の傾向はおおむね一致している．このことから，提案するコストモデルにより処理時間を見積もることができるため，最も処理時間が短くなる最適化手法もまた，適切に選択することが可能となる．

以上から，c，r，w の値を適切に設定することで，本研究で提案するコストモデルによ

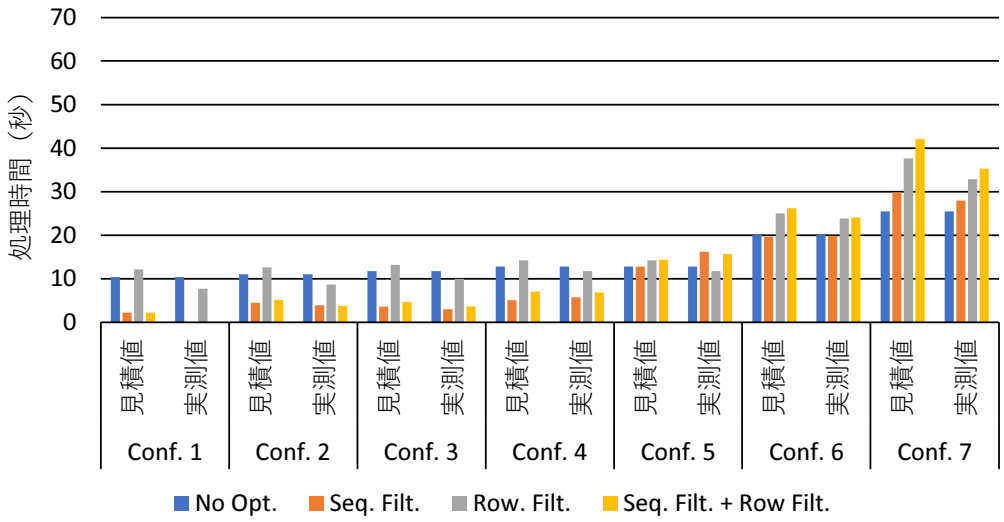

図 3.44 PostgreSQL における見積値と実測値の比較（Q1，N = 10,000,000，S = 1,000）

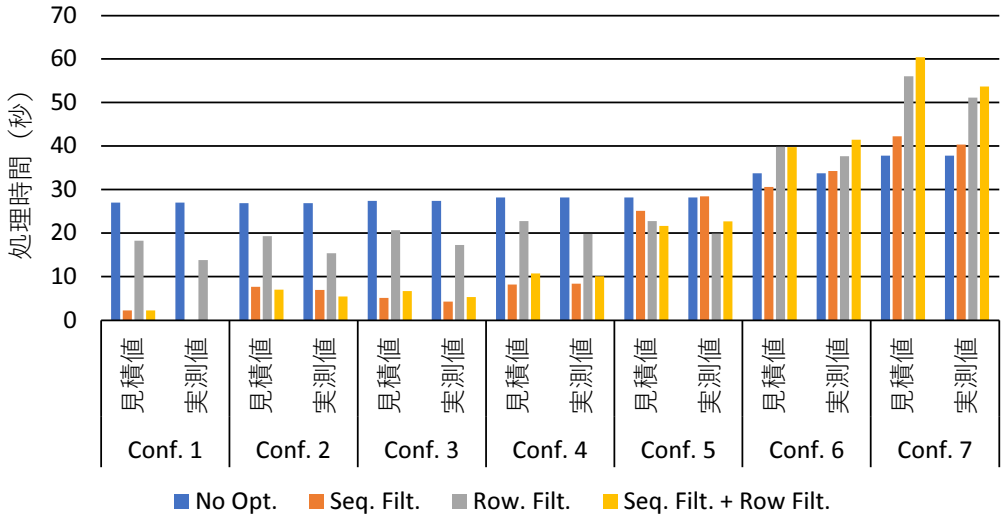

図 3.45 PostgreSQL における見積値と実測値の比較（Q5，N = 10,000,000，S = 1,000）

り処理コストが最小となる最適化手法を選択することが可能となる．

3.10 まとめと今後の課題

本研究では，RDB 等に格納された膨大な時系列データに対する行パターンマッチングの処理コストを削減するために，行パターンマッチング処理の前に，Selection や Join といった比較的コストの小さい処理を組み合せ，行パターンマッチング対象となる行数を

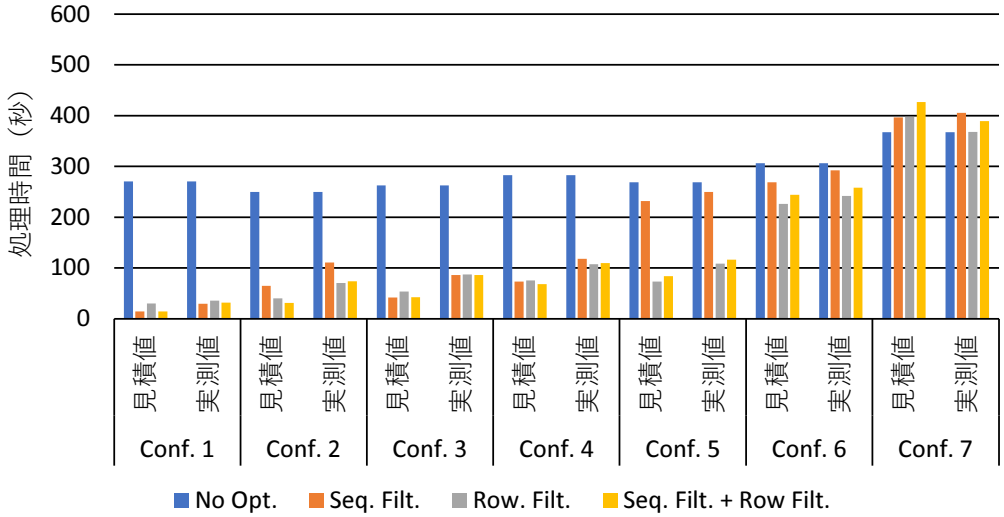

図 3.46 Spark における見積値と実測値の比較（Q1，N = 10,000,000，S = 1,000）

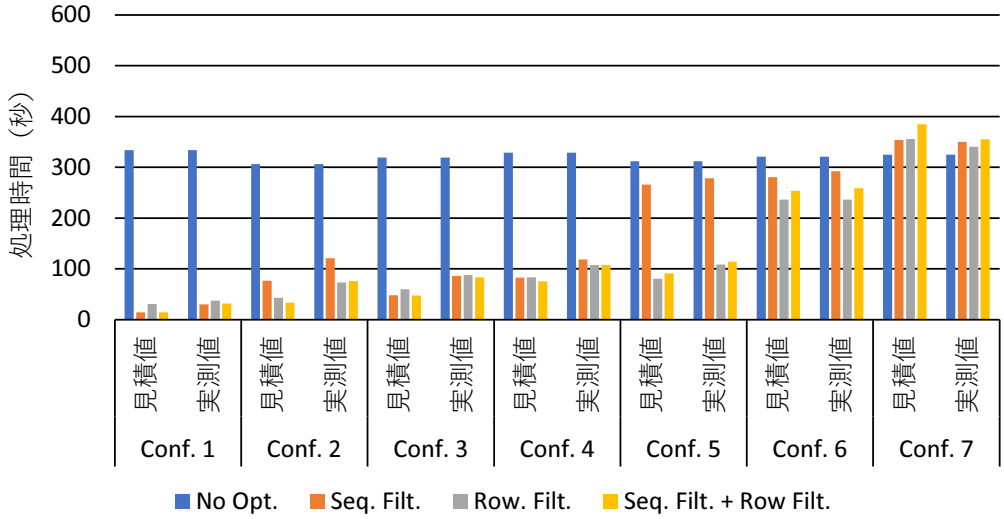

図 3.47 Spark における見積値と実測値の比較（Q5，N = 10,000,000，S = 1,000）

削減する Filtering Query を事前に適用することで，行パターンマッチングの処理コストを削減する最適化手法を提案した．行パターンマッチングの処理コストの削減のために，Sequence Filtering と，Row Filtering の 2 種類の最適化手法を提案した．この 2 種類の最適化手法について，コストモデルを構築し，その妥当性を確認した．また，UDF により SQL/RPR を実装した PostgreSQL，SQL/RPR を実装した拡張 Spark を用いて，人工データと Foursquare データセットの 2 種類のデータセットに対し，提案する最適化手

表 3.5 見積値と実測値の RMSE および RE（N = 10,000,000, S = 1,000）

	クエリ	最適化手法	RMSE（秒）	RE (%)
PostgreSQL	Q1	Seq. Filt.	1.74	15.85
		Row Filt.	3.42	22.48
		Seq. Filt. + Row Filt.	2.94	23.02
	Q5	Seq. Filt.	2.21	12.61
		Row Filt.	3.63	14.52
		Seq. Filt. + Row Filt.	2.92	14.74
Spark	Q1	Seq. Filt.	32.17	17.43
		Row Filt.	28.04	19.26
		Seq. Filt. + Row Filt.	34.73	22.83
	Q5	Seq. Filt.	27.44	15.05
		Row Filt.	21.83	15.42
		Seq. Filt. + Row Filt.	28.90	19.70

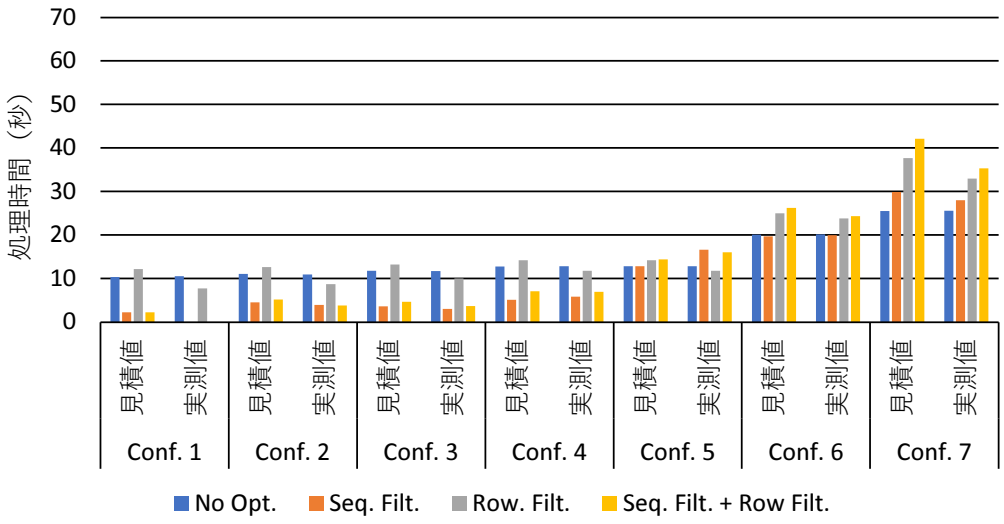

図 3.48 PostgreSQL における見積値と実測値の比較（Q1, N = 10,000,000, S = 10,000）

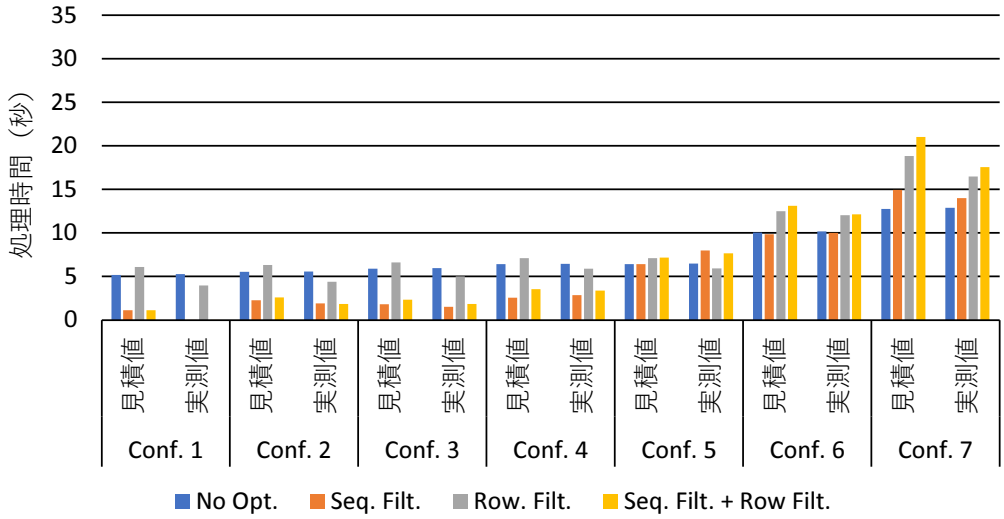

図 3.49 PostgreSQL における見積値と実測値の比較（Q1，N = 5,000,000，S = 1,000）

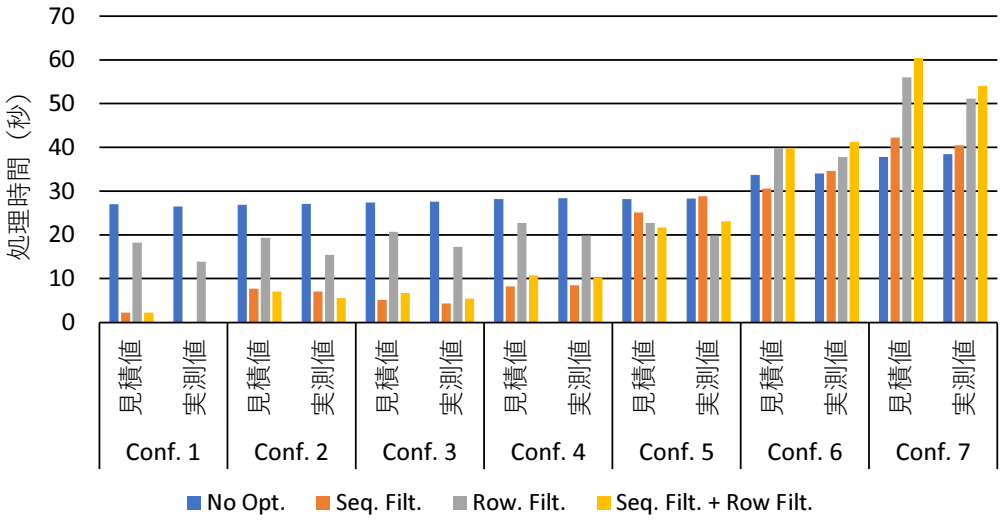

図 3.50 PostgreSQL における見積値と実測値の比較（Q5，N = 10,000,000，S = 10,000）

法の効果を検証した結果，最適化により行パターンマッチング対象の行数が多く削減される場合，処理時間削減の効果が得られていることを確認した．

今後の課題として，行パターンマッチング以外のクエリ演算を含めたコストモデルを構築し，クエリ全体の処理の最適化を行うことが挙げられる．また，本研究では，フィルタリングに用いる MATCH_RECOGNIZE 句の DEFINE 句における条件指定として，one variable condition のみを対象としたが，other condition も活用することで，行パ

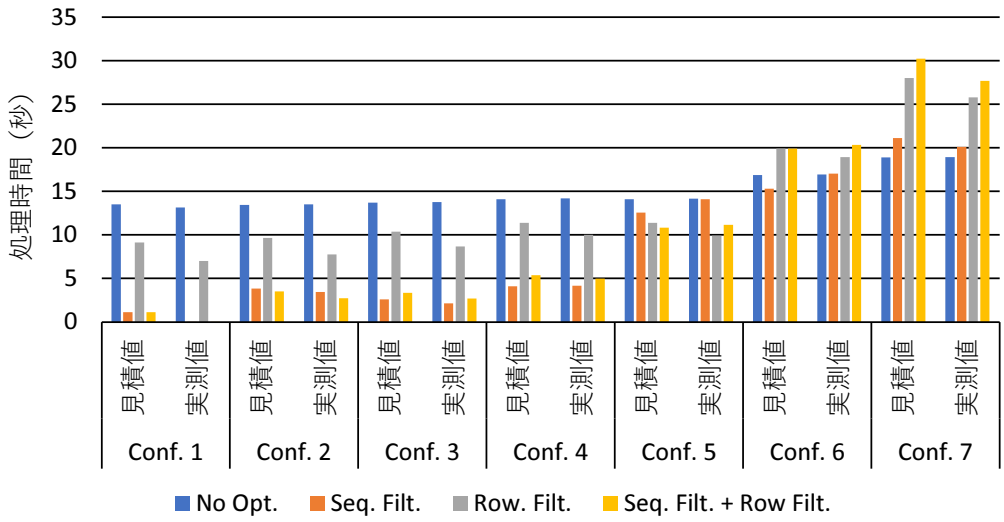

図 3.51 PostgreSQL における見積値と実測値の比較（Q5，N = 5,000,000，S = 1,000）

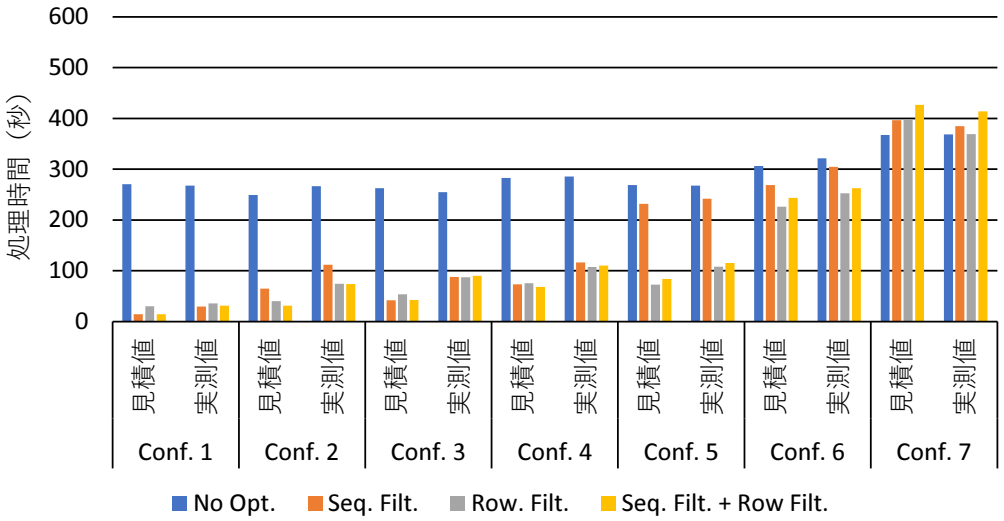

図 3.52 Spark における見積値と実測値の比較（Q1，N = 10,000,000，S = 10,000）

ターンマッチング対象となる行数をさらに削減する方法を検討することが考えられる．

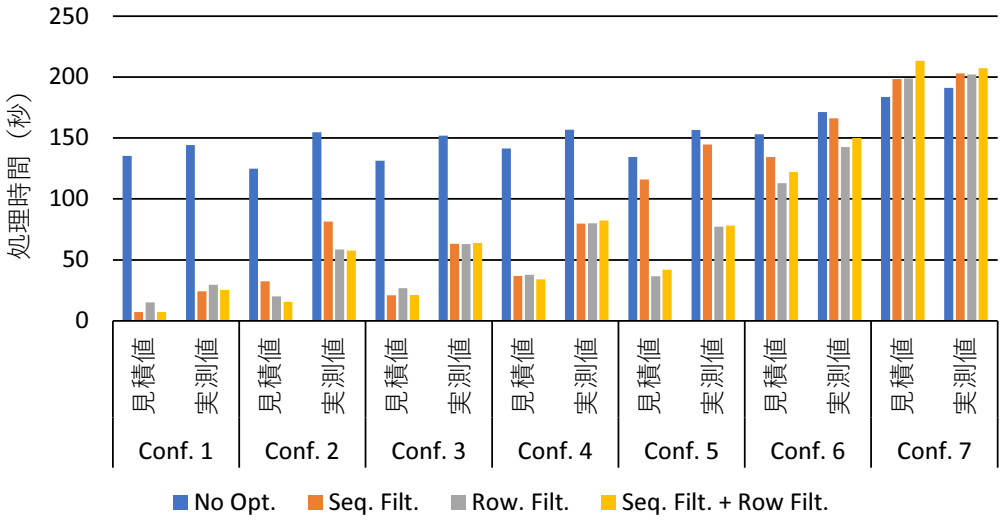

図 3.53 Spark における見積値と実測値の比較（Q1，N = 5,000,000，S = 1,000）

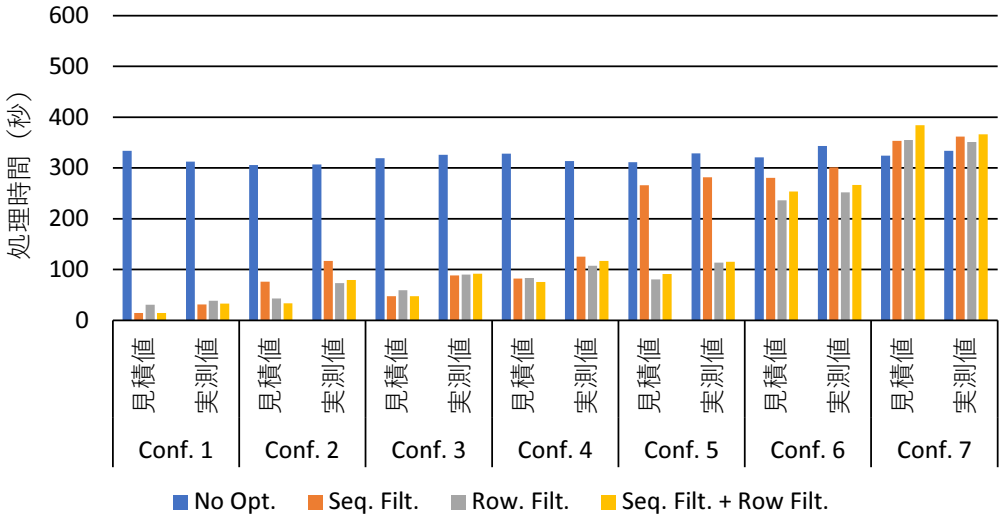

図 3.54 Spark における見積値と実測値の比較（Q5，N = 10,000,000，S = 10,000）

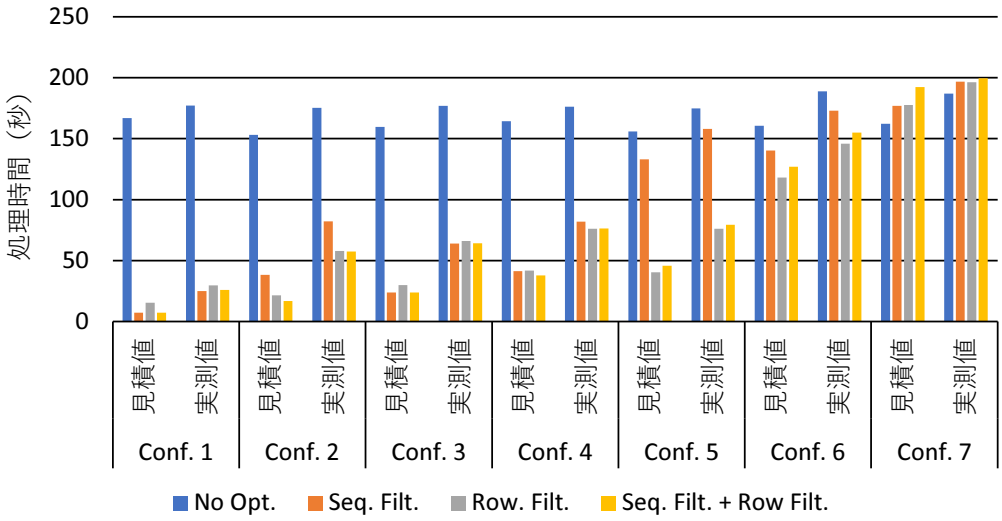

図 3.55 Spark における見積値と実測値の比較（Q5，N = 5,000,000，S = 1,000）

表 3.6 見積値と実測値の RMSE および RE（N = 10,000,000, S = 10,000）

	クエリ	最適化手法	RMSE (秒)	RE (%)
PostgreSQL	Q1	No Opt.	0.10	0.66
		Seq. Filt.	1.85	16.76
		Row Filt.	3.39	22.26
		Seq. Filt. + Row Filt.	2.92	22.72
	Q5	No Opt.	0.35	1.16
		Seq. Filt.	2.37	13.42
		Row Filt.	3.60	14.39
		Seq. Filt. + Row Filt.	2.79	14.00
Spark	Q1	No Opt.	9.30	3.20
		Seq. Filt.	33.57	18.40
		Row Filt.	29.45	19.94
		Seq. Filt. + Row Filt.	33.09	21.10
	Q5	No Opt.	15.18	4.69
		Seq. Filt.	29.84	15.97
		Row Filt.	23.55	16.05
		Seq. Filt. + Row Filt.	32.15	21.02

表 3.7 見積値と実測値の RMSE および RE（N = 5,000,000, S = 1,000）

	クエリ	最適化手法	RMSE (秒)	RE (%)
PostgreSQL	Q1	No Opt.	0.09	1.25
		Seq. Filt.	0.83	15.24
		Row Filt.	1.66	21.70
		Seq. Filt. + Row Filt.	1.47	23.23
	Q5	No Opt.	0.16	1.05
		Seq. Filt.	1.06	12.22
		Row Filt.	1.72	13.71
		Seq. Filt. + Row Filt.	1.15	11.57
Spark	Q1	No Opt.	18.94	11.77
		Seq. Filt.	34.18	31.38
		Row Filt.	32.37	34.70
		Seq. Filt. + Row Filt.	34.56	36.42
	Q5	No Opt.	20.05	11.17
		Seq. Filt.	32.94	29.53
		Row Filt.	30.32	32.74
		Seq. Filt. + Row Filt.	31.81	33.86

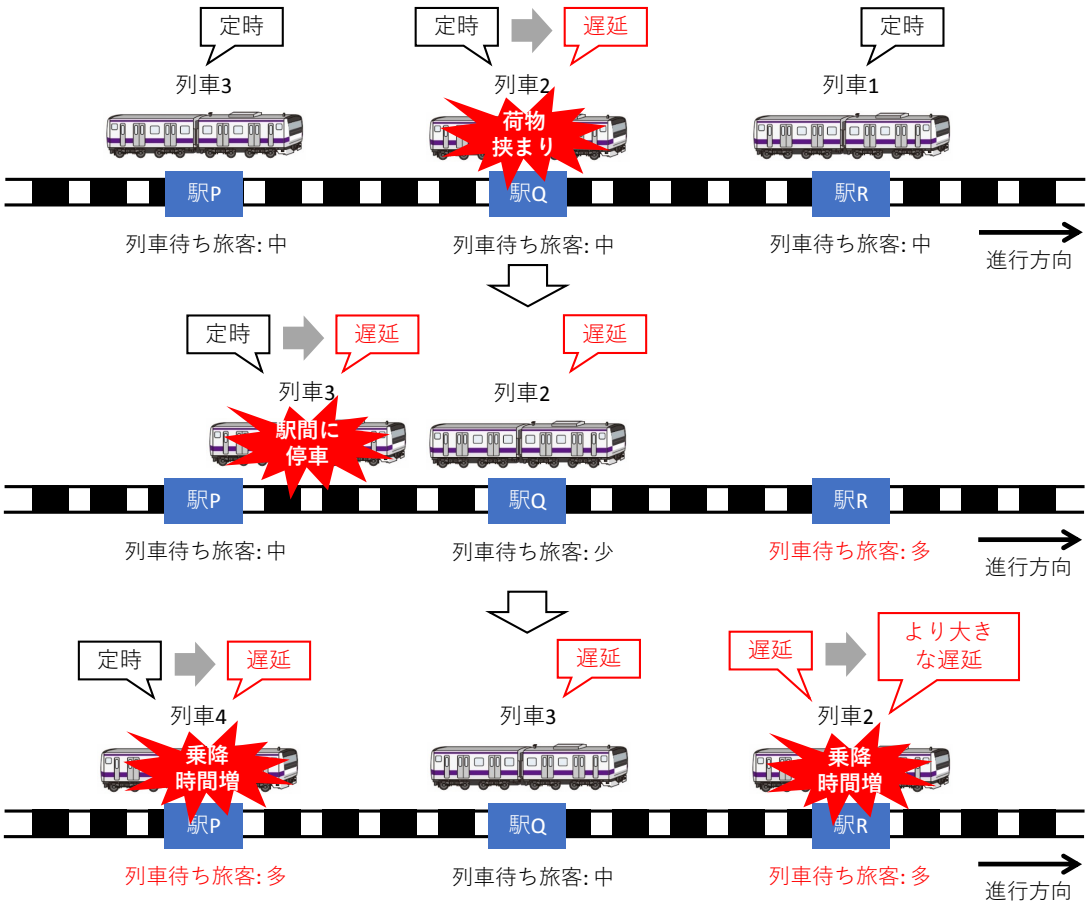

図 4.1 朝ラッシュ時の慢性的な遅延の原因

4　ニューラルネットワークを用いた列車遅延予測手法

4.1　背景と目的

近年，朝ラッシュ時間帯における大都市圏の通勤路線において，慢性的な遅延が発生している．このメカニズムを図 4.1 に示す．まず，列車 2 において，荷物挟まり等によって駅の停車時間が拡大することで，列車遅延が発生する．この遅延により，前列車である列車 1 との間隔が開いた結果，駅 R 以降での乗車旅客数が増加し，停車時間が延び，さらに遅延が拡大する．また，その後列車 3 に対しても，駅間での減速もしくは停車を引き起こし，連鎖的に他列車にも遅延が伝搬する．これらの遅延の拡大，伝搬が，日々の慢性的な遅延の原因となっている [125].

列車の運行を管理する指令室では，指令員が列車運行状況を監視し，ダイヤ乱れが発生して列車に遅れが生じた際，ダイヤ変更などにより旅客への影響を最小限に抑え，遅延を回復させるための手配を行う．この一連の手配は運転整理と呼ばれる．適切な運転整理の実施により，旅客の利便性を確保するほか，駅の混雑に伴う入場規制等のトラブルを防止する必要があり，社会的な責任も大きい業務である．朝ラッシュ時間帯における遅延に対しても，指令員は，発生した遅延の更なる拡大，伝搬を防止するため，列車の間隔調整や，列車の順序変更といった運転整理を行う．

列車の間隔調整，順序変更等を含む運転整理を的確に実施するためには，その時点の各列車の遅延状況や，各駅における混雑状況から，短時間先までの各列車の遅延，混雑の推移を正確に予測する必要がある．遅延が拡大するか，縮小するかにより，適切な運転整理は異なってくるが，現状，遅延の拡大・縮小は，主に指令員が経験を頼りに予測することが多い．しかし，利用者の乗降に要する時間の増加や混雑の影響による遅延時間の増減は，日々の状況によって変わるため，その予測は難しい．運行状況の変化を見誤ると，実施した運転整理について期待した効果が得られず，遅延の更なる拡大や伝搬を招き，旅客の利便性が更に低下する恐れがある．また，鉄道事業者では，列車の遅延時に，各列車の遅延と混雑状況を一定の精度で案内するように努めている．しかし，多くの場合が現時点での列車の遅延や混雑状況である．対象列車がその後，当駅に到着するまでの間に，遅延や混雑状況が変化する可能性があることを考えると，対象列車が当駅に到着し，旅客が乗車する時点での遅延や混雑状況をなるべく正確に予測し，それを適切に案内することが必要となる．

近年ではコンピュータによる支援も検討され，運行管理システムへの導入も始まっている．しかし，現状の予測手法は，現時点での各列車の遅延がそのまま持続する前提とした単純なもので，その予測の精度に課題がある．

そこで，本研究では，日々記録されている列車の遅延の時系列データを対象に，短時間先の列車遅延を高精度に予測する手法を提案する．具体的には，現在時刻から数駅前の予測対象列車の遅延と乗車率の時系列データ，および，予測対象列車の前後列車の遅延の時系列データから，短時間先までの間に予測対象列車が停車／通過する駅の遅延を予測する，ニューラルネットワークを用いた予測モデルを提案する．この予測モデルを用いて，短時間先までの列車遅延を高精度に予測することが可能となることで，指令員や駅係員による，適切な運転整理，旅客案内を実現できる．

4.1.1 本研究において対象とする遅延の規模

鉄道路線において事故や車両故障等が発生すると，列車の正常な運行が妨げられ，ダイヤ乱れが発生する．ダイヤ乱れは，その原因によって遅れの規模や対処法が異なり，大まかに，大乱れ，中乱れ，小乱れに分類できる [125]．大乱れは，人身事故，車両故障，設備故障などに起因するもので，概ね 30 分程度以上の遅延を伴うダイヤ乱れをいう．中乱れは，朝ラッシュ時の遅延拡大，線路内人立ち入りなどに起因するもので，概ね 10 分〜30 分程度の遅延のダイヤ乱れをいう．小乱れは，朝ラッシュ時の旅客混雑に伴う停車時間の拡大などに起因するもので，数分程度の遅延が発生している状況のことをいう．

大乱れや中乱れ時には，ダイヤの変更に加え，車両や乗務員の運用計画の変更，駅や車両基地での構内作業計画の変更も必要となるなど影響範囲が広いため，運転整理の計画，実施が難しく，鉄道事業者にとって大きな課題となっている．一方で，小乱れ時は線区の特性を踏まえた対処方法がある程度は確立されているものの，特に高密度運行を行うラッシュ時間帯においては，利用者集中に伴う小規模の遅延が慢性的に発生している．鉄道事業者においても朝ラッシュ時の遅延は，解決すべき重要課題の一つと位置付けられており，高頻度に発生する小乱れへの対策の改良も求められている．

本研究では，朝ラッシュ時の旅客混雑に伴う停車時間の拡大などに起因する，概ね 10 分未満の遅延が発生している小乱れを対象とする．小乱れ時の列車の遅延を高精度に予測することを目的とする．

4.1.2 列車の遅延の時系列データ

本研究では，列車の遅延の時系列データを対象とする．このデータは，線路上の軌道回路の落下扛上時刻を補正したデータであり，日々，各列車の各駅における遅延のデータとして指令室の運行管理システムに記録される．列車の遅延の時系列データは，各列車が停車／通過してきた駅の遅延が時刻順に並んでいるデータであり，ある列車のある駅の遅延は，その列車の時間的に近い（前駅や前々駅等の）遅延ほど相関がある時系列データである．このデータのフォーマットを表 4.1 に示す．属性としては，列車番号，時刻，発／着事象，駅，遅延といったものが含まれる．この中で，列車番号が sequence_id，時刻が order_key に対応する．また，各行に固有の属性として，発／着事象，駅，遅延を含む多属性値の時系列データである．なお，遅延は秒単位で記録，管理されている．そのため，遅延の予測においては秒単位の精度が求められる．本研究では，列車の発事象における遅延を予測する．なお，以降では，発事象における遅延を発遅延，着事象における遅延を着遅延と呼ぶ．

表 4.1 列車運行実績データ（列車の遅延データ）

列車番号	時刻	駅	発／着	遅延（秒）
1M	7:00:10	駅 A	発	10
1M	7:05:10	駅 B	着	10
1M	7:06:15	駅 B	発	15
...	...	...	...	...
2M	8:30:45	駅 D	発	30
...	...	...	...	...

表 4.2 列車の乗車率データ

列車番号	時刻	発駅	乗車率 (%)
1M	7:00:10	駅 A	120
1M	7:06:15	駅 B	125
...	...	...	...
2M	8:30:45	駅 D	140
...	...	...	...

また，本研究では，列車の遅延データとは別に，各駅間の列車の乗車率データを用いる．このデータのフォーマットを表 4.2 に示す．属性としては，列車番号，時刻，発駅，乗車率といった属性を含む時系列データである．

列車の遅延の時系列データに対する予測においては，以下に挙げるような特性が存在する．

- 列車の遅延のデータは，一定の時間間隔で取得されるデータではなく，データの取得は，列車の発／着事象においてなされる．
- 列車の遅延のデータには，周期性は存在しない．
- ある列車の遅延は，その列車の遅延だけでなく，他の列車の遅延や乗車率が影響する．

一般的に，時系列データに対する予測では，ARIMA，RNN，LSTM といった，時系列データの自己回帰性に着目した予測モデルが用いられる．一方で，列車の遅延の時系列データは，3 点目に示したように，乗車率の時系列データや他の列車の遅延の時系列データと相関が存在する．そのため，複数の時系列データを用いた予測モデルを検討する必要がある．また，1 点目に示した性質から，複数の列車の遅延データ間は，一定の時間間隔で同期していないため，単純に自己回帰モデルの入力を複数データに拡張したモデルは適用できない．以上から，各列車の発遅延の値を予測するためには，他の列車の遅延，乗車率といった複数の時系列データと，駅といった遅延以外の属性を考慮した予測モデルを検討する必要がある．

4.2 関連研究

4.2.1 列車の運行予測を対象とした研究

列車運行の予測に関しては，安部，荒屋らにより，プロジェクトマネジメント手法の 1 つである PERT を用いた手法が提案されている [127]．これは，列車の走行を模擬する有向グラフ上で，最長径路探索をすることで，列車運行をシミュレーションしている．また，國松らは，列車運行・旅客行動シミュレータによる乗車率，遅延予測手法を提案している [130]．これは，列車のダイヤデータと自動改札機から取得できる時間帯別駅間別の旅客の移動情報データを用いて，列車の運行と旅客の行動をシミュレーションするものである．同様のデータを用いて，長崎らは，列車の乗車率を高速に予測する手法を提案しており [134]，辰井らは，これを発展させ，ダイヤデータを変更すると，即時に列車の乗車率を再計算する，対話型ダイヤ作成システムによる乗車率予測手法を提案している [133]．岩倉らは，列車の行動と旅客の行動にルールを設定し，マルチエージェントシミュレーションを行うことで，列車の遅延の連鎖を再現し，遅延を予測する手法を提案している [128][131]．一方で，以上に挙げた研究は，いずれも，遅延の発生，伝搬や乗車率を，その基本的なメカニズムに沿って順次計算し予測したものであり，過去の日々の遅延，乗車率データを直接反映してはいない．つまり，例えば特定の駅（乗換駅等）での利用者集中による遅延の頻発など，様々な路線，駅に対し，精度良く予測できないという課題がある．

Şahin は，マルコフ連鎖モデルを利用して列車の遅延を予測する手法を提案している [90][91]．これは，過去の列車の遅延データから，列車の遅延が大きくなるか，減少するかを学習し，現在の駅の遅延から数駅先の遅延の状況を予測するモデルである．ただし，4.1.2 節に示したように，高頻度で列車が運行されている日本の鉄道路線においては，他列車の遅延や乗車率が遅延に大きく影響するため，遅延の予測においては，これらの影響

を反映した予測モデルを構築する必要がある．

岩本らは，短時間先までの遅延を予測しダイヤ図を描画する機能を開発し，一部路線の運行管理システムで実用化している [129]．しかし，路線，駅，列車種別，時間帯等に応じて個別にパラメータを設定し予測する手法であるため，そのパラメータの適切な設定方法や予測手法の汎用性の面で課題がある．高垣らは，過去の各駅間におけるグリーン車の乗車率の推移を表すデータから，類似度計算によって，短時間先のグリーン車の乗車率予測手法を提案している [132]．この手法は，過去の実績データを直接利用するものであるが，遅延の予測は行っていない．

4.2.2 複数の時系列データを用いた予測に関する研究

4.1.2 節において説明したように，列車の遅延の予測においては，複数の時系列データを用いて予測する手法を検討する必要がある．2.2.2 節において挙げた複数の時系列データを用いた予測手法として，Yi らは，複数の共進化する時系列データに対して予測を行うことを目的とし，ある時系列データの値の予測に，自時系列データと他の時系列データから，多変量線形回帰を用いて予測する，MUSCLES と呼ばれる手法を提案している [115]．一方で，日々の列車の遅延は，その日の乗客の数，乗降時間，他列車・他路線・他交通機関の運行状況により複雑に変動するため，その予測には，線形モデルではなく非線形モデルが適していると考える．そのため，本研究では非線形モデルを用いて列車の遅延を予測する．4.4 節において，本研究の提案手法と線形モデルによる予測との予測精度を比較し，検証する．また，2.2.2 節において挙げた他の非線形モデルを用いた各手法は，複数の時系列データが等間隔で同期していること，周期的な変動があること，非数値データあることを前提とした手法であったり，複数の時系列データから別の時系列データを予測する手法であるため，本研究で対象とする列車の遅延の時系列データの予測モデルとしては適さない．

4.2.3 非線形モデルを用いた列車の運行予測を対象とした研究

非線形な予測モデルとして，ニューラルネットワークを用いて，列車遅延データの予測を行う研究として，Chapuis は，ある列車のある駅における遅延を予測するために，その列車が最後に停車した駅の遅延とその駅間に存在する駅数，およびその駅に最後に停車した列車の遅延を入力とし，学習・予測するモデルを提案している [22]．フランス国鉄の路線を対象に手法の精度の検証を行っている．この研究の予測手法は，本研究において提案する手法と類似したものであるが，遅延の粒度が分単位であること，今まで停車してきた複数駅の遅延が考慮されていないこと，列車の緩急接続に伴う遅延の波及が未考慮である

こと，旅客の流動が未考慮であることといった問題がある．本研究において提案する手法は，上記の点を考慮した予測モデルである点に優位性がある．[22] の研究で提案されている予測モデルとの精度を，4.4 節において比較し，検証する．

Mou らは，オランダ鉄道を対象に，LSTM を用いて列車の遅延を予測する手法を提案している [70]．この予測モデルは，列車種別，現在の駅の発遅延，現在の駅の着遅延，前駅の発遅延，前駅と現在の駅の間の計画上の走行時間，現在の駅と次駅との間の計画上の走行時間の 6 種類のデータから，次駅の着遅延を予測するモデルである．また，Marković らは，セルビア鉄道を対象に，サポートベクター回帰により列車の着遅延を予測するモデルを提案している [64]．ただし，これらの手法は，ある列車の遅延の予測にその列車に関する情報しか利用していない．日本の鉄道路線は，列車の運転本数が多く過密なダイヤであり，前列車や後列車の遅延が自列車の遅延に影響を与えるため，これらの列車の遅延を加味した予測モデルとする必要がある．また，[22] の手法と同様に，列車の緩急接続に伴う遅延の波及や旅客の流動が考慮されていない．

上記の他にも，Yaghini らは，イラン鉄道を対象に，ある駅間，経路の日々の遅延から，将来の遅延を数分単位の粒度で予測する手法を提案している [114]．本研究では列車単位で各駅の遅延を予測することが目的であるため，[114] の手法は適用できない．朝倉らは，CNN[58] を用いて列車の遅延を予測する手法を提案している [11][126]．各列車の各駅の遅延を 2 次元配列として用意し，ある日の列車の遅延を予測するために，別の複数日の遅延を CNN を用いて学習し，予測対象日の列車の遅延を予測する手法である．ただし，この手法は，ある日のある列車の遅延を予測する際に，他の列車の遅延のデータがすでに揃っている必要があり，実際の列車運行における遅延の予測には利用できない．

本研究の予測対象外ではあるが，列車遅延の予測と類似したものとして，列車の乗車率を予測する研究も存在する．Pasini らは，パリの大都市圏の近郊路線を対象に，列車の乗車率の時系列データを，勾配ブースティング決定木，ランダムフォレスト，LSTM を用いて予測し，これらを，観測時点の乗車率が持続するという仮定に基づく手法 (Last Value, LV) と，平均乗車率が同一の日・時間帯の乗車率のデータを流用して予測する手法の，5 つの手法間で精度を比較，検証している [80]．他にも，北京の地下鉄の乗車率を，活性化関数に放射基底関数 (RBF) を用いたニューラルネットワークと SVM を用いて予測する手法 [102] や，オランダのトラムおよびバス路線を対象に乗車率の予測を，ランダムフォレスト，勾配ブースティング，ニューラルネットワーク，k-NN のそれぞれを用いて予測する手法 [43] が存在する．また，列車の乗車率を直接予測するわけではないが，乗客の流れを予測する研究として，Toqué らは，スマートカードデータから乗客の出発地／到着地毎の人数を LSTM を用いて予測する手法を提案している [98]．

4.3 ニューラルネットワークを用いた列車遅延予測モデルの検討

以上までに述べたことから，列車の遅延を予測するためのモデルは，下記の点を満たす必要がある．

- ある列車の遅延の予測に，その列車が今まで通ってきた駅の遅延を考慮する．
- ある列車の遅延の予測に，その列車が今まで通ってきた駅間の乗車率を考慮する．
- ある列車の遅延の予測に，他列車の遅延を考慮する．
- ある列車ある駅の遅延の予測に用いる列車・駅の時間的・物理的な近さを考慮する．

本研究では，上記を満たす，ニューラルネットワークを用いた短時間先までの列車の発遅延を予測するモデルを検討する．表 4.3 に，本節における説明に用いる記号の定義を行う．表中の記号を用いて，予測対象列車の発遅延を，d_T のように表す．また，予測対象列車の i 番目の駅の発遅延を，d_{T_i} のように表す．よって，予測対象列車の遅延の時系列データは，以下のように表される．

$$d_{T_1},\ a_{T_2},\ d_{T_2},\ \cdots,\ a_{T_i},\ d_{T_i},\ \cdots,\ d_{T_{N-1}},\ a_{T_N}$$

同様に，予測対象列車の乗車率の時系列データは以下のように表される．

$$c_{T_1},\ c_{T_2},\ \cdots,\ c_{T_i},\ \cdots,\ c_{T_{N-1}}$$

また，本研究では前列車および後列車を次のように定義する．

- 予測対象列車と同一方向，同一線路上の，物理的に前方を走行する列車を前列車とする．
- 予測対象列車と同一方向，同一線路上の，物理的に後方を走行する列車を後列車とする．

予測対象列車，前列車，後列車の関係を，図 4.2 に列車ダイヤ図上に示す．なお，列車ダイヤ図は，縦軸を駅，横軸を時刻とし，列車を，各駅の発時刻，着時刻を線で結んだスジと呼ばれる斜めの線で表す．図 4.2 において列車 2 を予測対象列車 T とする．今，列車 2 が i 番目の駅に到着したタイミング（時刻 $t1$ とし，以降ではこの時刻を現在時刻と

表 4.3 列車遅延の予測モデルにおいて用いる記号の定義

記号	意味
T	予測対象列車 (Target train)
F	予測対象列車の前列車 (Former train)
L	予測対象列車の後列車 (Latter train)
d	発遅延 (departure delay)
a	着遅延 (arrival delay)
c	発車時の乗車率 (congestion rate)
M	対象路線の列車数
N	対象路線の駅数
i	対象路線の起点からの駅数 ($1 \leq i \leq N$)
j	予測対象列車が i 番目の駅着時点で，後列車が最後に停車／通過した駅の i 番目の駅からの駅数
W	予測対象列車が i 番目の駅着時点以降の駅の発遅延を予測する時間幅

呼ぶ.）である場合，列車 2 の i 番目の駅の発遅延が d_{T_i}，着遅延が a_{T_i} にあたる．前列車 F は列車 2 の前を走る列車 1 となるため，i 番目の駅の列車 1 の発遅延が d_{F_i} にあたる．後列車 L は列車 2 の後ろを走る列車 3 となる．i 番目の駅の列車 3 の発遅延が d_{L_i} にあたるが，時刻 $t1$ においては列車 3 はまだ i 番目の駅に到達していないため，列車 2 の i 番目の駅の発遅延の予測に d_{L_i} の値は使えない．そのため，本研究では，時刻 $t1$ に後列車 L が最後に停車／通過した駅の，i 番目の駅からの駅数 j を定め，その駅の遅延 $d_{L_{i-j}}$ を用いる．図 4.2 では，列車 3 が時刻 $t1$ に最後に停車した駅は $i-3$ 番目の駅であるため，$d_{L_{i-j}} = d_{L_{i-3}}$ となる．図 4.3 に示す通り，乗車率についても同様に，列車 2 の i 番目の駅の発時点の乗車率が c_{T_i} にあたる．また，本研究では，現在時刻 $t1$ から短時間先の予測対象列車の発遅延を予測する．この時間幅を W とし，図 4.2 中に網掛けで示した部分になる．

列車 2 は，$i+1$ 番目の駅において，後列車である列車 3 に追い抜かれる．列車 2 が $i+2$ 番目の駅に到着したタイミングを $t2$ とし，この時点の列車ダイヤ図を図 4.4 に示す．$t2$ の時点では，前列車は列車 3，後列車は列車 4 に変わる．これに伴い，d_{F_i} は列車 3 の i 番目の駅の発遅延，$d_{L_{i-j}}$ は列車 4 の $i-j$ 番目の駅の発遅延となる．

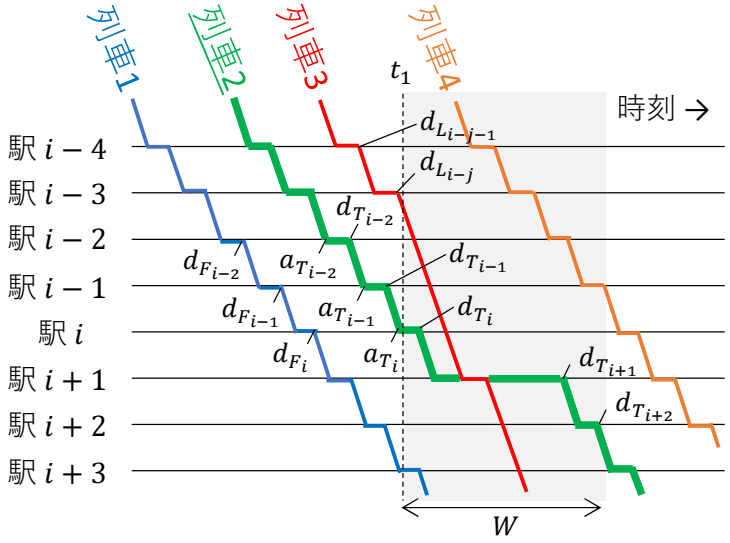

図 4.2 $t1$ における前後列車とその遅延の対応関係

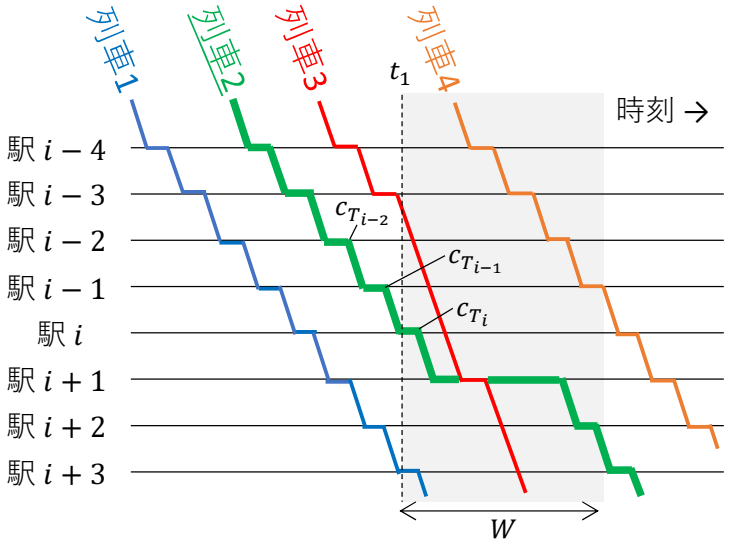

図 4.3 $t1$ における乗車率の対応関係

4.3.1 列車遅延を予測するモデルの単位

列車の遅延を予測するモデルについて，構築するモデルの単位として 3 種類が考えられる．1 つ目は，列車種別毎にモデルを構築する方法，2 つ目は，列車毎にモデルを構築する方法，3 つ目は，各列車の各駅毎にモデルを構築する方法である．

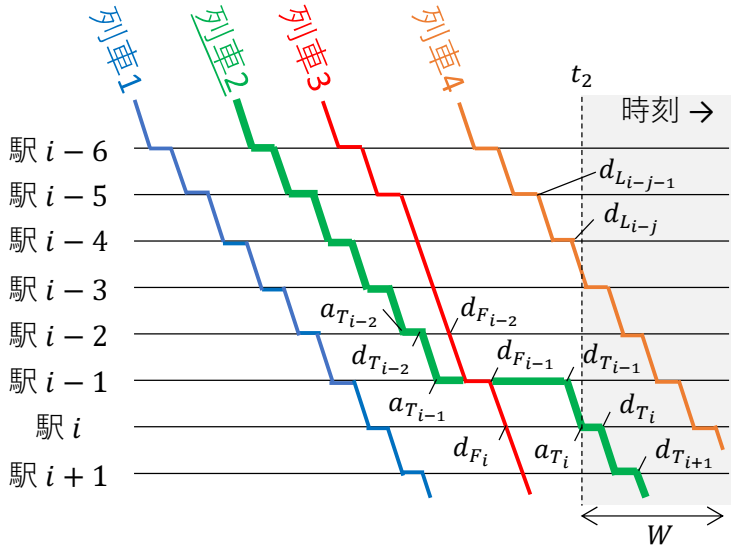

図 4.4 $t2$ における前後列車とその遅延の対応関係

1 つ目の，列車種別毎のモデルは，予測対象路線における，各駅停車や快速といった列車種別ごとに予測モデルを構築する方法である．図 4.5 の列車ダイヤを例に説明する．図 4.5 では，始発駅 A から終着駅 H までの 8 つの駅が存在し，各駅停車 2 本，快速 2 本の，計 4 本の列車が設定されているとする．列車種別毎のモデルでは，各駅停車の遅延を予測するモデルと快速の遅延を予測するモデルの 2 つの予測モデルを構築する．この方法は，モデルの数が少量で済み，多量の学習データを用いて学習を行うことができる．一方で，特に朝ラッシュ時の通勤時間帯においては，同じ列車種別であっても列車毎に遅延の傾向が異なる．そのため，本研究では列車種別毎のモデルは，予測モデルに適さないと考える．

2 つ目の，列車毎のモデルは，予測対象路線における列車 1 本 1 本毎に予測モデルを構築する方法である．図 4.5 では，4 本の列車それぞれに，4 つの予測モデルを構築する．この方法は，列車種別毎のモデルと比較するとモデルの数は多く，1 つのモデルに対して学習に使えるデータが少なくなるものの，列車毎の遅延の傾向を考慮した予測ができる．一方で，列車の遅延は駅によってその傾向が異なる．例えば，郊外の駅と都心部の駅では乗降客数が異なるため乗降時間が異なる．また，列車を待避する駅では，待避する列車と待避される列車によって，駅間の各列車の走行時間や停車時間が異なるため，列車を待避しない駅とは，遅延の傾向が異なる．さらに，他路線や他交通機関との乗換駅では，それらの路線の列車やバスの遅延状況が旅客の流れに影響するため，非乗換駅とは遅延の傾向が異なる．そのため，本研究では，列車毎だけでなく駅毎の遅延の傾向を考慮した予測モ

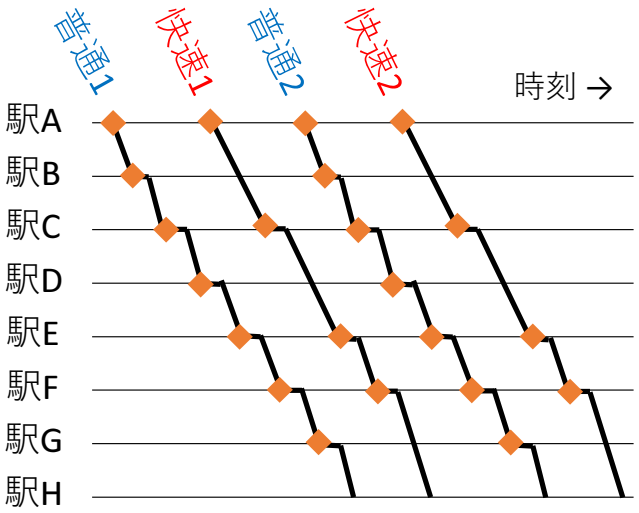

図 4.5 列車ダイヤ図の例

デルの方がより精度良く予測できると考える．

3 つ目の，各列車の各駅毎のモデルは，予測対象路線における各列車が停車する全駅について予測モデルを構築する方法である．図 4.5 では，4 本の列車が停車する各駅（橙の菱形で示している）に対して 1 つの予測モデルを構築する．そのため，合計で 22 個の予測モデルを構築する．この方法は，列車毎のモデルと比較するとモデルの数はさらに多く，1 つのモデルに対して学習に使えるデータはより少なくなるものの，各列車の各駅における遅延の傾向を考慮した予測ができると考える．

以上から，本研究では，各列車の各駅毎にモデルを構築する方法で，列車の遅延を予測する手法を検討する．なお，予測するモデルの単位毎の予測精度の検証を 4.4.4 節において行う．

4.3.2 ニューラルネットワークを用いた列車遅延予測モデル

本節では，各列車の各駅毎に構築する，短時間先までの列車の遅延を予測するモデルを説明する．予測モデルには，図 4.6 に示すような，入力層，隠れ層，出力層の 3 層からなる Feed-Forward Neural Network を用いる．活性化関数としてシグモイド関数，学習法として誤差逆伝搬法を用い，隠れ層のユニット数は 30 とする．各列車，各駅に予測モデルを用意し，対応する列車が対応する駅に到着した時点で，その時点から，計画ダイヤ上で時間幅 W だけ先までの間に停車する駅の発遅延を予測する．以降で，入力データと出力データについて説明する．

入力データは，対応する列車が対応する駅に到着した時点から，時間幅 W だけ先までの間に後列車を待避する場合としない場合で 2 パターンに分ける．後列車を待避する場合，4 種類の入力データを用いる．1 つ目は，予測対象列車の手前 5 駅分の発遅延である．ある駅の発遅延は，時間的に近い駅の遅延ほど相関があるため，手前 5 駅分の遅延を用いる．2 つ目は，予測対象列車の手前 5 駅分の乗車率である．列車が混雑しているほど乗降に時間がかかり，それが遅延の原因となる．そのため，乗車率を入力データに含める．3 つ目は，前列車の手前 5 駅分の発遅延である．前列車が遅れている場合，予測対象列車がその列車に近づくことで減速や駅間停車が発生し，それが遅延の原因となる．そのため，前列車の遅延を入力データに含める．4 つ目は，現在時刻 t において後列車が最後に発車した駅から手前 5 駅分の発遅延である．予測対象列車が短時間先までの間に後列車に追い越されるために待避する場合，後列車が遅れていると，その列車を待つために待避時間が延び，それが遅延の原因となる．そのため，後列車の遅延を入力データに含める．この後列車を待避する場合の入力データを入力データパターン 1 とする．後列車を待避しない場合は，後列車を待避する入力データパターン 1 から，後列車の発遅延を除いた 3 種類の入力データを用いる．これを入力データパターン 2 とする．以下に，予測対象列車 T が，i 番目の駅に到着した時点での，入力データパターン 1 の場合の入力データを $Input1$，入力データパターン 2 の場合の入力データを $Input2$ としてそれぞれ示す．なお，入力データとして含めるデータを変えた時の予測の精度の検証を 4.4.5 節において行う．

$$
\begin{aligned}
Input1 =&(d_{T_{i-5}},\, d_{T_{i-4}},\, d_{T_{i-3}},\, d_{T_{i-2}},\, d_{T_{i-1}},\, c_{T_{i-5}},\, c_{T_{i-4}},\, c_{T_{i-3}},\, c_{T_{i-2}},\, c_{T_{i-1}},\\
&d_{F_{i-4}},\, d_{F_{i-3}},\, d_{F_{i-2}},\, d_{F_{i-1}},\, d_{F_i},\, d_{L_{i-j-4}},\, d_{L_{i-j-3}},\, d_{L_{i-j-2}},\, d_{L_{i-j-1}},\, d_{L_{i-j}})\\
Input2 =&(d_{T_{i-5}},\, d_{T_{i-4}},\, d_{T_{i-3}},\, d_{T_{i-2}},\, d_{T_{i-1}},\, c_{T_{i-5}},\, c_{T_{i-4}},\, c_{T_{i-3}},\, c_{T_{i-2}},\, c_{T_{i-1}},\\
&d_{F_{i-4}},\, d_{F_{i-3}},\, d_{F_{i-2}},\, d_{F_{i-1}},\, d_{F_i})
\end{aligned}
$$

出力データは，時間幅 W だけ先までの間に停車する駅の発遅延となる．今，時間幅 W の間に k 駅進むとした場合の出力データ $Output$ を以下に示す．

$$Output = (d_{T_i},\, d_{T_{i+1}},\, \ldots,\, d_{T_{i+k}})$$

短時間先までの列車の遅延を予測するモデルの入出力データを表 4.4 に整理する．

上記に示した入力データと出力データは, それぞれ特定の入力層のユニットの入力, 出力層のユニットの出力に対応する．今, 入力データパターン 1 の場合の予測モデルを対象に, 入力層のユニットを, $I = (I_1, I_2, \ldots, I_{20})$, 出力層のユニットを, $O = (O_1, O_2, \ldots, O_k)$ とする．各入出力データと入出力層のユニットの対応を図 4.6 に示す．まず，$I_1,\, I_2,\, \ldots,\, I_5$

表 4.4 予測モデルにおける入出力データ

入力データ パターン 1	入力データ パターン 2	出力データ
・予測対象列車の 手前 5 駅分の発遅延 ・予測対象列車の 手前 5 駅分の乗車率 ・前列車の手前 5 駅分 の発遅延 ・後列車の現在時刻 t に おいて最後に発車した駅 から手前 5 駅分の発遅延	・予測対象列車の 手前 5 駅分の発遅延 ・予測対象列車の 手前 5 駅分の乗車率 ・前列車の手前 5 駅分 の発遅延	・時間幅 W だけ先の間に 停車する駅の発遅延

の順に，予測対象列車の 5 駅手前からの発遅延 $d_{T_{i-5}}$, $d_{T_{i-4}}$, $d_{T_{i-3}}$, $d_{T_{i-2}}$, $d_{T_{i-1}}$ を入力とする．次に，I_6, I_7, ... , I_{10} の順に，予測対象列車の 5 駅手前からの乗車率 $c_{T_{i-5}}$, $c_{T_{i-4}}$, $c_{T_{i-3}}$, $c_{T_{i-2}}$, $c_{T_{i-1}}$ を入力とする．次に，I_{11}, I_{12}, ... , I_{15} の順に，前列車の 5 駅手前からの発遅延 $d_{F_{i-4}}$, $d_{F_{i-3}}$, $d_{F_{i-2}}$, $d_{F_{i-1}}$, d_{F_i} を入力とする．最後に，I_{16}, I_{17}, ... , I_{20} の順に，後列車の現在時刻 t において最後に発車した駅から 5 駅手前からの発遅延 $d_{L_{i-j-4}}$, $d_{L_{i-j-3}}$, $d_{L_{i-j-2}}$, $d_{L_{i-j-1}}$, $d_{L_{i-j}}$ を入力とする．出力層のユニット O_1, O_2, ... , O_k からの出力は，それぞれ予測対象列車が直後に発車する駅の発遅延から k 駅先までの発遅延 d_{T_i}, $d_{T_{i+1}}$, ... , $d_{T_{i+k}}$ に対応する．同様に，入力データパターン 2 の場合の各入出力データと入出力層のユニットの対応を図 4.7 に示す．入力データパターン 2 は，入力データパターン 1 における，I_1 から I_{15} のみが入力となる．この予測モデルを用いて，学習し予測を行う．この予測モデルでは，学習データ数は学習に用いる日数に等しい．

以上に説明した予測モデルは，特定の路線の形態やダイヤ構成に依存しない予測モデルである．Algorithm 6 に，ダイヤ上の任意の列車，任意の駅における予測モデルの学習・予測に用いるデータを抽出する流れを示す．予測対象列車 T が，その路線の始発駅から i 番目の駅に到着した時点での予測モデルの入出力データを抽出する．まず，予測モデルの入力配列 $Input$ と，$Output$ を用意する．その後，入力データについては関

Algorithm 6 予測モデルに対する入出力データの抽出

Input: 全列車の遅延の時系列データ $SeqD$, 全列車の乗車率の時系列データ $SeqC$, 予測対象列車 T, 始発駅からの駅の数 i

Output: 入力データ配列 $Input$, 出力データ配列 $Output$

1: $Input = \{\}$, $Output = \{\}$
2: $t \leftarrow T$ が i 番目の駅に到着した時刻
3: $Input \leftarrow getInputData(SeqD,\ SeqC,\ T,\ i,\ t)$
4: $Output \leftarrow getOutputData(SeqD,\ T,\ i,\ t)$
5: **return** $Input$, $Output$

Algorithm 7 入力データを抽出する関数 getInputData()

1: **function** getInputData($SeqD$, $SeqC$, T, i, t)
2: $F \leftarrow i$ 番目の駅における列車 T の前列車
3: **if** t から時間 W の間に後列車を待避 **then**
4: $L \leftarrow i$ 番目の駅における列車 T の後列車
5: $j \leftarrow t$ における後列車 L が最後に停車した駅の i 番目の駅からの数
6: **end if**
7: **for** $h \leftarrow 1\ to\ 5$ **do**
8: **if** $1 \leq i - h$ **then**
9: $d_{T_{i-h}} \leftarrow SeqD$ から列車 T の $i - h$ 番目の駅の遅延を抽出
10: $c_{T_{i-h}} \leftarrow SeqC$ から列車 T の $i - h$ 番目の駅の乗車率を抽出
11: $d_{F_{i-h+1}} \leftarrow SeqD$ から列車 F の $i - h + 1$ 番目の駅の遅延を抽出
12: **end if**
13: **if** t から時間 W の間に後列車を待避 $AND\ 1 \leq i - j - h$ **then**
14: $d_{L_{i-j-h}} \leftarrow SeqD$ から列車 L の $i - j - h$ 番目の駅の遅延を抽出
15: **end if**
16: $Input \leftarrow d_{T_{i-h}},\ c_{T_{i-h}},\ d_{F_{i-h}}, d_{L_{i-j-h}}$ を追加
17: **end for**
18: $Input$ を入力ノード順にソート
19: **return** $Input$
20: **end function**

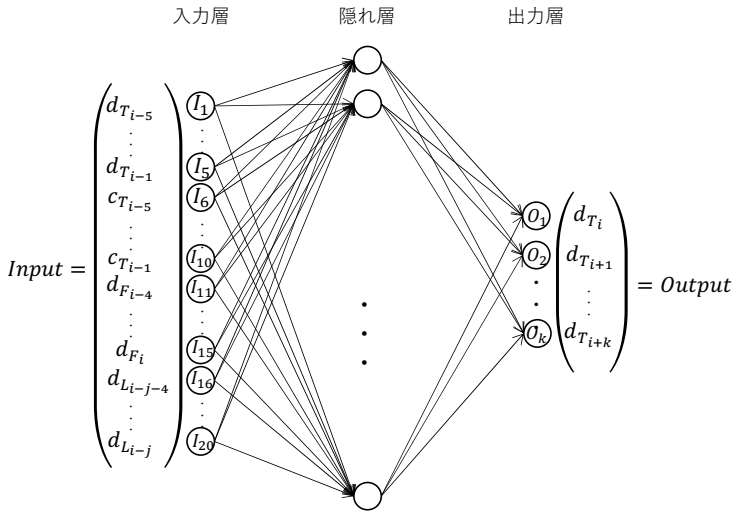

図 4.6 入力データパターン 1 の入出力データと入出力層のユニットの対応

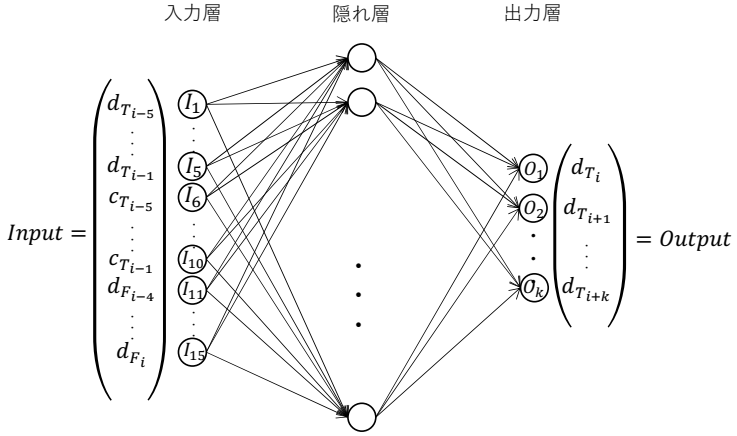

図 4.7 入力データパターン 2 の入出力データと入出力層のユニットの対応

数 $getInputData()$，出力データについては関数 $getOutputData()$ で抽出を行う．関数 $getInputData()$ の処理を Algorithm 7 に示す．関数 $getInputData()$ では，2 行目において，i 番目の駅における前列車 F を定める．3 行目から 6 行目において，予測対象列車 T が現在時刻 t から時間幅 W の間に列車を待避する場合，i 番目の駅における後列車 L を定め，後列車 L が現在時刻 t において最後に停車した駅の i 番目の駅からの駅数 j を定める．そして，8 行目から 12 行目において，その路線の始発駅を超えない限り，予測対象列車 T の遅延と乗車率，前列車 F の遅延を抽出する．予測対象列車 T が待避をする場合，13 行目から 15 行目において後列車 L の遅延を抽出する．そして，16 行目において

Algorithm 8 出力データを抽出する関数 getOutputData()

1: **function** getInputData($SeqD$, i, t)
2: $k \leftarrow t$ から時間 W の間に停車する駅数
3: **for** $h \leftarrow 0$ *to* k **do**
4: **if** $i + h \leq N$ **then**
5: $d_{T_{i+h}} \leftarrow seqD$ から列車 T の $i + h$ 番目の駅の列車の遅延を抽出
6: $Output \leftarrow d_{T_{i+h}}$ を追加
7: **end if**
8: **end for**
9: $Output$ を出力ノード順にソート
10: **return** $Output$
11: **end function**

抽出したデータを $Input$ に追加する．これを，手前 5 駅分に対して行う．最後に 18 行目において，$Input$ 内のデータをニューラルネットワークの各入力ノードに対応するようにソートする．

出力データを抽出する関数 $getOutputData()$ の処理を Algorithm 8 に示す．関数 $getOutputData()$ では，2 行目において現在時刻 t から予測対象列車 T が，時間幅 W の間に停車／通過する駅数 k を定める．そして，3 行目から 8 行目において，i 番目の駅も含めて $k + 1$ 駅分，その路線の終着駅を超えないところまで，予測対象列車 T の i 番目の駅から遅延を抽出する．最後に 9 行目において，$Output$ 内のデータをニューラルネットワークの各出力ノードに対応するようにソートする．

図 4.8 に例を示す．今，列車 2 を予測対象列車とし，駅 G に到着した時点の時刻 t での予測モデルの入出力データを説明する．駅 G が i 番目の駅であり，図中では星形のマークで示している．まず，予測対象列車 2 について，駅 G の 5 駅前である駅 B から駅 F の発遅延と乗車率が入力となる．図 4.8 では，発遅延は緑の丸，乗車率は緑の三角で示している（ダイヤ図上では，乗車率の緑の三角は発遅延の緑の丸と被るため示していない）．次に，列車 2 の前列車である列車 1 について，駅 G を含めて 5 駅前である駅 C から駅 G の発遅延が入力となる．図 4.8 では，青い丸で示している．さらに，列車 2 は時刻 t から時間 W の間に後列車 3 に追い抜かれるため，後列車 3 の発遅延も入力とする．後列車 3 は，時刻 t において駅 E と駅 F の間を走行中である．そのため，後列車 3 については，現在時刻 t の時点で最後に停車した駅は駅 E となり $(j = 2)$，駅 E を含めて 5 駅前である

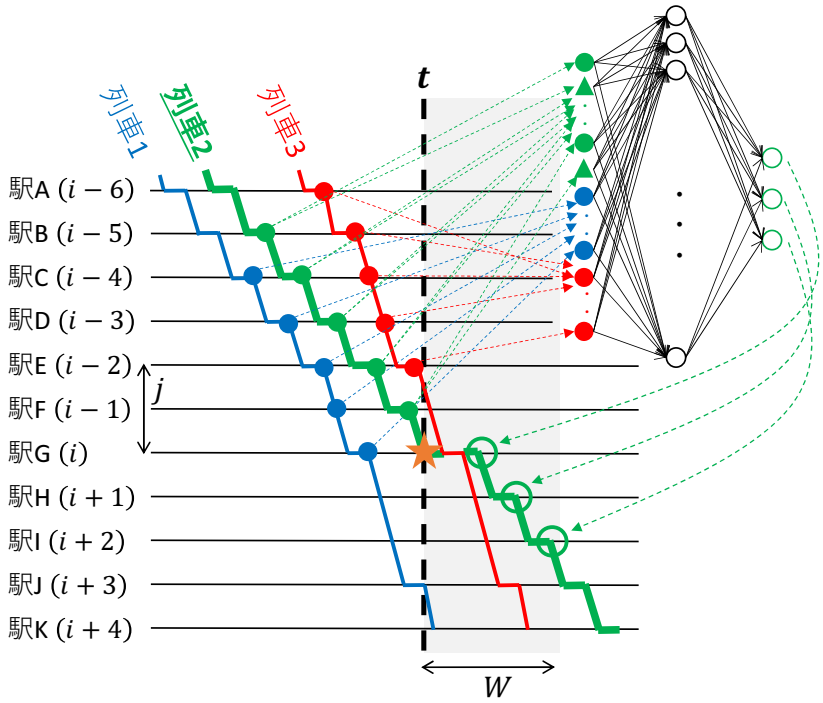

図 4.8 入出力データの例

駅 A から駅 E の発遅延が入力となる．図 4.8 では，赤い丸で示している．出力については，列車 2 は時刻 t から時間 W の間に駅 I（$i+2$ 番目の駅，$k=2$）まで走行するため，駅 G から駅 I までの発遅延が出力となる．図 4.8 では，白抜きの緑の丸で示している．

複数の路線の合流，分岐が存在する路線では，予測対象列車の前列車や後列車が他路線から合流したり，他路線に分岐していく場合が存在する．本研究ではこのような場合，他路線との合流・分岐駅をそれらの列車の始発・終着駅として扱うことで，本研究の手法を適用可能とする．例を図 4.9 に示す．図の例では，駅 B と駅 F を結ぶ路線 p に対し，駅 C において駅 A からの路線 q が合流し，駅 D において駅 G へ路線 r が分岐する．今，路線 p の駅 B から駅 F に向かう列車 2 を予測対象列車とし，駅 E 到着時点のその駅以降の遅延を予測する場合を考える．列車 2 の前列車が，路線 q の駅 A から路線 p を通り，路線 r の駅 G に向かう列車 1 であるとき，列車 1 は，路線 p の駅 C を始発駅とし，駅 D を終着駅とする列車として扱う．これにより，路線の合流，分岐の有無にかかわらず，任意の路線における任意の駅に対して予測が可能となる．

図 4.9 路線の合流と分岐の例

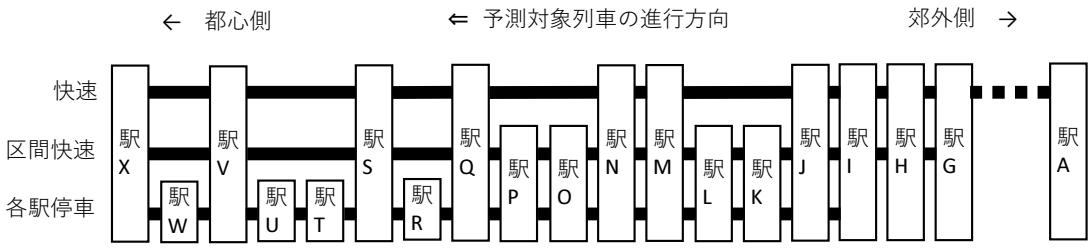

図 4.10 対象路線の概要

4.4 評価実験

4.4.1 対象路線の概要と使用データ

本評価実験では，大都市近郊の実在路線を対象とする．路線の概要を図 4.10 に示す．対象の路線は，最も郊外の駅 A から最も都心側の駅 X を結ぶ全 24 駅の通勤路線で，終日の上下方向合わせた列車数は 413 本（平日）である．列車種別は，快速，区間快速，各駅停車の 3 種類が存在し，快速は駅 A から駅 X まで走行し，駅 J から駅 X の間で通過駅が存在する．区間快速は駅 G または駅 I から駅 X まで走行し，駅 Q から駅 X の間で通過駅が存在する．各駅停車は駅 I または駅 Q から駅 X まで走行し，全駅に停車する．以降では，駅 I から駅 X までの長距離を走行する各駅停車を各停 1，駅 Q から駅 X までの短距離を走行する各停 2 と呼ぶ．また，駅 S において，各駅停車が快速あるいは区間快速を待避する．

本評価実験では，この路線の最も都心側の駅 X に朝ラッシュ時間帯のピークである 7 時 30 分から 8 時 30 分の間に到着する列車 15 本（快速: 3 本，区間快速: 4 本，各停 1: 3 本，各停 2: 5 本）を対象とする．列車の遅延のデータは，ある平日の 79 日分のデータを用いる．この 79 日間の 15 列車の平均遅延は 21.7 秒である．本研究では，朝ラッシュ時の小乱れ時の列車の遅延を予測することを対象としている．79 日間の 15 列車の全駅の遅延量の 97.4% が，150 秒以下であるため，本研究では 150 秒を閾値とし，それ以上の遅延があった列車の遅延のデータは除外し，予測の対象外とする．

乗車率データについては，車両に搭載された応荷重装置 [135] による乗車人数データの使用を想定している．しかし今回，対象路線の車両から応荷重装置のデータは取得不可能であったため，各駅の自動改札機から取得できる，ある 2 駅間の時間帯ごとの移動人数データ（自動改札機 OD データと呼ぶ）と列車の遅延データも含めた列車運行実績データを使用し，[133] の対話型乗車率推定システムにより，各列車，各駅間の乗車率を推定した値で代用する．

4.4.2 予測モデルによる学習と予測

本評価実験では，平日の 79 日分のデータに対し，前半の 64 日分を学習および検証用データとして用い，後半 15 日分をテストデータとして用いる．学習および検証では，64 日分のデータに対し，63 日分のデータで学習を行い，残りの 1 日で検証を行う処理を 64 回分，交差検証を行う（Leave One Out 法）．64 回分の交差検証における二乗誤差の平均を計算し，この誤差がパラメータとして設定した誤差閾値を下回るか，前回学習時の誤差と比較し増大した場合，学習を終了する．なお，本評価実験ではパラメータとして，誤差閾値を 0.0001，学習率を 0.01 としている．

なお，予測モデルのニューラルネットワークの活性化関数としてはシグモイド関数を用いている．シグモイド関数は，値域が (0, 1) である一方で，列車の遅延データは [0, 150]，乗車率のデータは $[0, \infty)$ の値を取る．そのため本研究では，列車の遅延データを 150 秒で正規化，乗車率のデータは 200% で正規化し，[0, 1] の範囲を取るようにする．また，出力層のユニットから出力された値は，150 秒の値で正規化の逆を行うことで，列車の遅延の予測値とする．

4.4.3 予測精度評価に用いる評価尺度

評価値として用いる遅延の予測精度は，15 日分のテストデータを用いて予測を行った場合の予測値（予測遅延）と実際の遅延の値（実績遅延）の平均平方二乗誤差 (RMSE) と相関係数 (R) を用いる．予測対象列車のある駅の実績遅延を d_{T_i}，予測遅延を $\widehat{d_{T_i}}$ とし，

算出式を以下に示す.

$$RMSE = \sqrt{\frac{\sum_{h=0}^{k}\left(d_{T_{i+h}} - \widehat{d_{T_{i+h}}}\right)^2}{k}}$$

$$R = \frac{\frac{1}{k}\sum_{h=0}^{k}\left(d_{T_{i+h}} - \overline{d_{T_{i+h}}}\right)\left(\widehat{d_{T_{i+h}}} - \overline{\widehat{d_{T_{i+h}}}}\right)}{\sqrt{\frac{1}{k}\sum_{h=0}^{k}\left(d_{T_{i+h}} - \overline{d_{T_{i+h}}}\right)^2}\sqrt{\frac{1}{k}\sum_{h=0}^{k}\left(\widehat{d_{T_{i+h}}} - \overline{\widehat{d_{T_{i+h}}}}\right)^2}}$$

なお，相関係数を用いる理由としては，予測遅延と実績遅延の増減の傾向がどの程度合致しているかを確認するためである．平均平方二乗誤差では，各駅において予測遅延と実績遅延がどれほど差があったかのみを確認できる．一方で，列車の遅延の予測においては，列車の遅延が今後増加するのか減少するのかを予測することも重要である．そこで，予測遅延と実績遅延の差だけでなく，相関係数を用いた遅延の増減の傾向の類似度合いも評価尺度として含める．

また，各モデルの予測結果について，平均平方二乗誤差 (RMSE) については，モデル間で比較した際に有意な差があるかについて t 検定を行っており，本研究では，p 値が 0.05 未満を統計的に有意とみなしている．相関係数 (R) については，あるモデルで予測した遅延と実績の遅延との間の無相関の検定を t 検定により行っており，p 値が 0.05 未満を統計的に有意とみなし，そのモデルが遅延の増減の傾向を予測できているとみなす.

4.4.4 列車遅延を予測するモデルの単位に関する精度評価

4.3.1 節において説明したように，本研究では，予測モデルの単位として，各列車の各駅毎にモデルを構築し，列車の遅延を予測する手法を提案している．本節では，4.3.1 節において示した，(1) 列車種別毎にモデルを構築する方法，(2) 列車毎にモデルを構築する方法，(3) 各列車の各駅毎にモデルを構築する方法の 3 種類の予測モデル間で予測精度の比較を行う．なお，以降では，(3) 各列車の各駅毎にモデルを構築する予測モデルを列車駅毎のモデルと呼ぶ.

本節の評価実験では，各予測モデル共通で入力データパターン 2 を採用する．これは，列車種別毎のモデルと列車毎のモデルは，全駅で共通の予測モデルを用いるため，到着する駅によって後列車の遅延を入力として含める場合と含めない場合がある入力データパターン 1 が利用できないためである．そのため，後列車の遅延を入力として含めない入力データパターン 2 の予測モデルで統一する．出力データについては，時間 W の間に全列車が 5 駅進むと仮定し $k = 5$ とする．よって，本節では $k = 5$ の場合の図 4.7 に示した

表 4.5 各モデルにおける学習データ数

予測モデル	学習に用いる日数	列車数	駅数	学習データ数
快速のモデル	64	3	15	2880
区間快速のモデル	64	4	9 (7)	2176
各停 1 のモデル	64	3	7	1344
列車毎（快速）のモデル	64	1	15	960
列車毎（区間快速）のモデル	64	1	9 (7)	576 (448)
列車毎（各停 1）のモデル	64	1	7	448
列車駅毎のモデル	64	1	1	64

予測モデルを共通で用いる．なお，本節における評価実験では，5 駅手前からの予測対象列車の発遅延，乗車率，前列車の発遅延から自駅を含めて 5 駅分の予測対象列車の発遅延を予測するモデルを用いるため，始発駅から終着駅までの駅数が 8 駅のみの各停 2 については，対象外とする．そのため，実験で対象とする列車数は，快速が 3 本，区間快速が 4 本，各停 1 が 3 本の 10 本とする．精度評価については，各列車が駅 Q および駅 S 到着時の 4 駅先までの各駅の遅延を予測した結果を用いて評価する．

各モデルにおける学習データ数を表 4.5 にまとめる．列車種別毎のモデルは，快速の予測モデル，区間快速の予測モデル，各停 1 の予測モデルの 3 種類を用意し，それぞれ学習・予測を行う．3 種類の予測モデルの中では，学習に用いるデータ数が最も多い．快速は，学習に用いる日数が 64 日分，列車数が 3 本，駅数は，5 駅手前の駅が始発駅となる駅 F から 5 駅先が終着駅となる駅 T までの 15 駅分のデータがあるため，学習データ数は，$64 \times 3 \times 15 = 2880$ となる．列車毎の予測モデルは列車種別ごとの予測モデルの次に学習データ数が多く，快速列車の学習データ数は，$64 \times 15 = 960$ となる．列車駅毎のモデルは，4.3.2 節で説明したように，学習データ数は学習に用いる日数に等しい．なお，区間快速は，4 本の内 3 本は駅 G，1 本は駅 I が始発駅となるため，表 4.5 中において駅 G 始発の列車は 9 駅分，駅 I 始発の列車は 7 駅分であることを括弧付きで示している．また，4.4.1 で説明したように，ある日に 150 秒以上の遅延があった列車は，その日のその列車のデータは学習データから除外している．そのため，実際の学習データ数は，表 4.5 中の学習に用いる日数が少なくなる分だけ少ない．

各予測モデルにおける予測精度を比較した結果を図 4.11 に示す．図 4.11 の結果から，

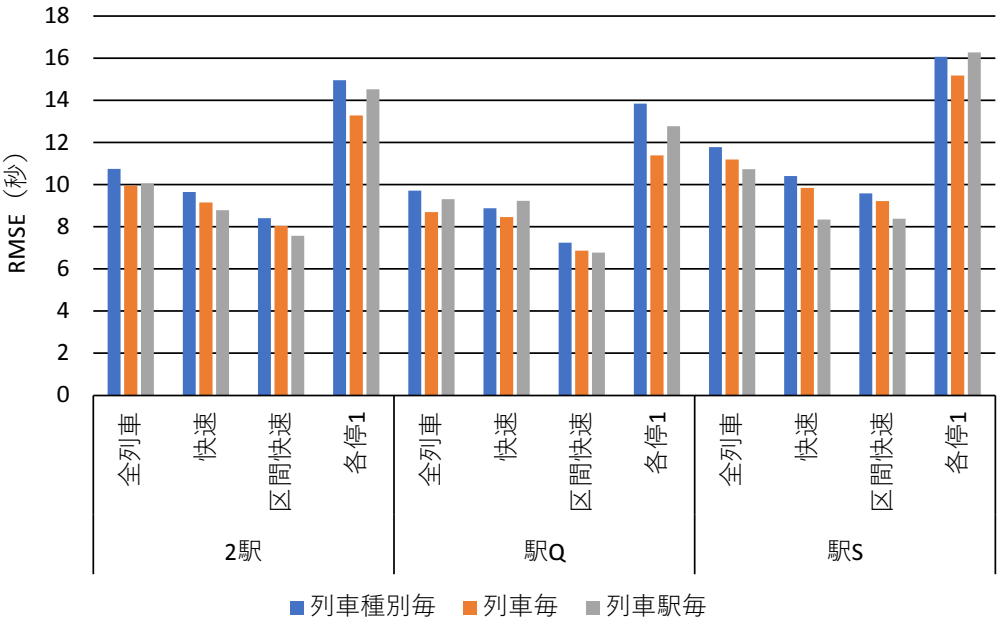

図 4.11 予測モデルの構築単位間の予測精度

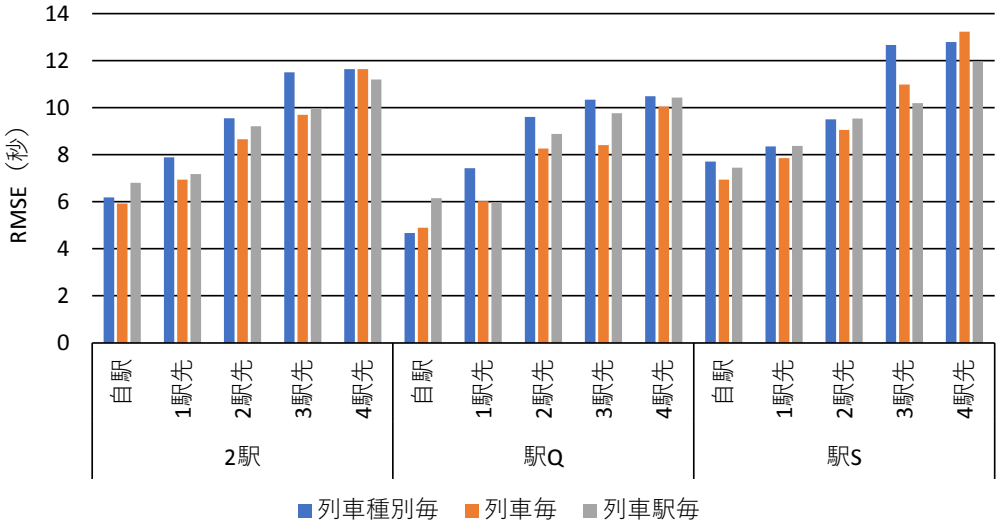

図 4.12 予測モデルの構築単位間の 4 駅先までの各駅の予測精度

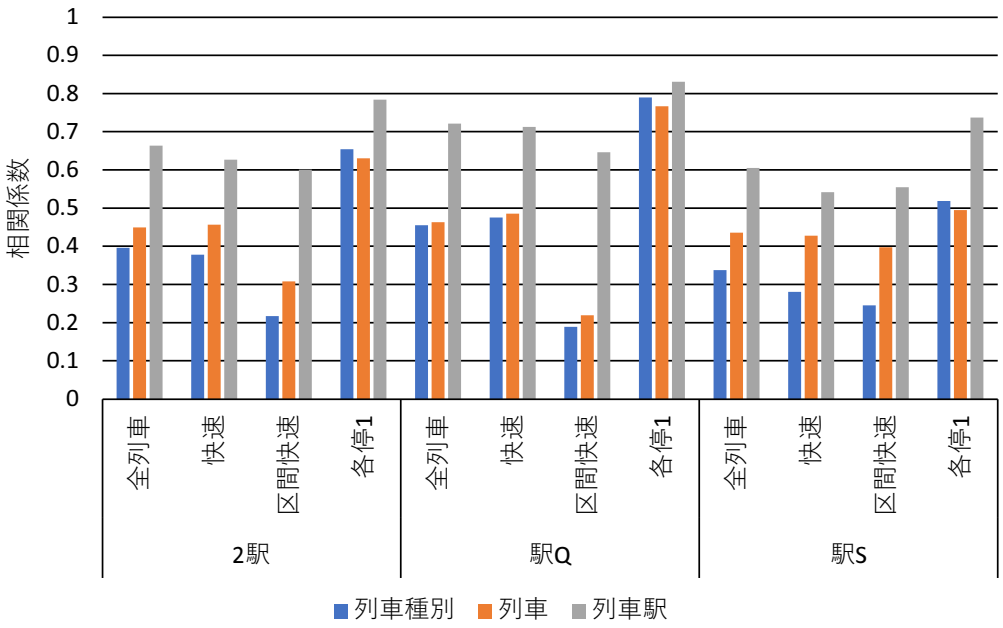

図 4.13 予測モデルの構築単位間の相関係数

全体的にいずれの予測モデルも予測精度に有意な差は見られなかった．駅 S 着時点における快速の予測については，列車駅毎の予測モデルが他のモデルに対し，有意に精度が良い事を確認している．一方で，各停 1 の予測精度がいずれのモデルにおいても良くない．これは，快速や区間快速に比べて遅延量が大きいこと，これに加えて，学習・検証に用いた日の遅延量に比べて，テストに用いた日の遅延量が小さかったために，予測精度が悪くなったと思われる．次に，図 4.12 に，4 駅先までの各駅での予測精度の結果を示す．全体的に，先の駅ほど予測の精度が悪くなっていくが，列車毎，列車駅毎のモデルが列車種別毎のモデルよりも精度が良く，特に，3 駅先における予測精度については有意な傾向があった．最後に，自駅から 4 駅先までの各駅の予測遅延と実績遅延の相関係数を図 4.13 に示す．図 4.13 から，列車駅毎の予測モデルが他の予測モデルに比べ，遅延の増減を精度よく予測できている．また，列車駅毎の予測モデルが，予測した遅延と実績の遅延との間に有意に相関があるものが最も多くなった．以上から，列車駅毎の予測モデルが，遅延の予測に最も適していると考えられる．なお，図 4.11，図 4.12，図 4.13 の具体的な値は，付録 B.1 の表 B.1，表 B.2，表 B.3 に示す．

表 4.6 入力データの組合せ

Case	予測対象列車の発遅延	予測対象列車の着遅延	予測対象列車の乗車率	前後列車の発遅延
Case i	始発からの全駅	-	始発からの全駅	-
Case ii	始発からの全駅	始発からの全駅	始発からの全駅	-
Case iii	始発からの全駅	-	始発からの全駅	始発からの全駅
Case iv	手前 5 駅分のみ	-	手前 5 駅分のみ	-
Case v	手前 5 駅分のみ	手前 5 駅分のみ	手前 5 駅分のみ	-
Case vi	手前 5 駅分のみ	-	手前 5 駅分のみ	手前 5 駅分のみ

4.4.5 列車遅延を予測するモデルの入力データに関する精度評価

4.3.2 節では，列車駅毎の予測モデルの入力データを定義した．本節では，この予測モデルに用いる入力データの種類およびその数が予測精度に与える影響を確認する．具体的には，表 4.6 に示す 6 種類の組合せ間で比較を行う．表中の “-” は，該当するデータを入力データに含めないことを表している．Case i は，入力データに始発駅から予測対象の 1 駅前までの全駅の発遅延と乗車率を含めるケースである．Case ii は，Case i に加えて始発駅から予測対象の 1 駅前までの全駅の着遅延も含めるケースである．Case iii は，Case i に加えて始発駅から予測対象の 1 駅前までの全駅の前後列車の発遅延も含めるケースである．Case iv，v，vi は，それぞれ Case i，ii，iii の入力データ数を手前 5 駅分のみに絞ったケースである．この 6 種類のケースの精度を比較する目的としては，まず，Case i，iv と Case ii，v の精度を比較することで，入力データに着遅延を含めた場合にどれほど精度に影響があるかを確認する．また，Case i，iv と Case iii，vi の精度を比較することで，入力データに前後列車の発遅延を含めた場合にどれほど精度に影響があるかを確認する．さらに，ある駅の列車の遅延の値は，時間的に近い駅の遅延の値ほど相関が高いと考えられる．そのため，入力データに用いるデータを手前数駅分に絞る方が，時間的に遠く相関が低いと思われる駅の遅延の値を含める場合よりも精度が良くなると考えられるため，Case i から iii と Case iv から vi を比較することで，入力データに用いるデータを手前数駅分に絞ることによる精度に与える影響を確認する．

各ケースにおける予測モデルの具体的な入出力データは以下のようになる．Case i の

入力を $Input_i$，Case ii の入力を $Input_{ii}$，Case iii の入力を $Input_{iii}$，Case iv の入力を $Input_{iv}$，Case v の入力を $Input_v$，Case vi の入力を $Input_{vi}$ とする．

$$
\begin{aligned}
Input_i =&(d_{T_1},\, d_{T_2},\, \ldots,\, d_{T_{i-2}},\, d_{T_{i-1}},\, c_{T_1},\, c_{T_2},\, \ldots,\, c_{T_{i-2}},\, c_{T_{i-1}})\\
Input_{ii} =&(d_{T_1},\, d_{T_2},\, \ldots,\, d_{T_{i-2}},\, d_{T_{i-1}},\, c_{T_1},\, c_{T_2},\, \ldots,\, c_{T_{i-2}},\, c_{T_{i-1}},\\
&a_{T_1},\, a_{T_2},\, \ldots,\, a_{T_{i-1}},\, a_{T_i})\\
Input_{iii} =&(d_{T_1},\, d_{T_2},\, \ldots,\, d_{T_{i-2}},\, d_{T_{i-1}},\, c_{T_1},\, c_{T_2},\, \ldots,\, c_{T_{i-2}},\, c_{T_{i-1}},\\
&d_{F_1},\, d_{F_2},\, \ldots,\, d_{F_{i-1}},\, d_{F_i},\, d_{L_1},\, d_{L_2},\, \ldots,\, d_{L_{i-j-1}},\, d_{L_{i-j}})\\
Input_{iv} =&(d_{T_{i-5}},\, d_{T_{i-4}},\, d_{T_{i-3}},\, d_{T_{i-2}},\, d_{T_{i-1}},\, c_{T_{i-5}},\, c_{T_{i-4}},\, c_{T_{i-3}},\, c_{T_{i-2}},\, c_{T_{i-1}})\\
Input_v =&(d_{T_{i-5}},\, d_{T_{i-4}},\, d_{T_{i-3}},\, d_{T_{i-2}},\, d_{T_{i-1}},\, c_{T_{i-5}},\, c_{T_{i-4}},\, c_{T_{i-3}},\, c_{T_{i-2}},\, c_{T_{i-1}},\\
&a_{T_{i-4}},\, a_{T_{i-3}},\, a_{T_{i-2}},\, a_{T_{i-1}},\, a_{T_i})\\
Input_{vi} =&(d_{T_{i-5}},\, d_{T_{i-4}},\, d_{T_{i-3}},\, d_{T_{i-2}},\, d_{T_{i-1}},\, c_{T_{i-5}},\, c_{T_{i-4}},\, c_{T_{i-3}},\, c_{T_{i-2}},\, c_{T_{i-1}},\\
&d_{F_{i-4}},\, d_{F_{i-3}},\, d_{F_{i-2}},\, d_{F_{i-1}},\, d_{F_i},\, d_{L_{i-j-4}},\, d_{L_{i-j-3}},\, d_{L_{i-j-2}},\, d_{L_{i-j-1}},\, d_{L_{i-j}})
\end{aligned}
$$

なお，出力 $Output$ は全てのモデルで共通である．

$$Output = (d_{T_i},\, d_{T_{i+1}},\, \ldots,\, d_{T_{i+k}})$$

本評価実験では，駅 Q，S，V の着時点における，その駅以降の各駅の発遅延の予測精度を用いて評価する．全ての列車がいずれの駅においても時間 W の間にその路線の駅 X に到着するとし，駅 Q，S，V から駅 X までの各駅の発遅延を予測する．各ケースの予測精度の結果を図 4.14 に示す．図 4.14 から，全体では，Case vi が最も良い精度となった．また，有意差が見られたのは，Case i と Case iii，Case i と Case vi，Case iii と Case iv，Case iv と Case vi の間であった．このことから，着遅延を入力データとして含めても，その予測精度は含まない場合と同程度であるが，前後列車の遅延を入力データとして含めると，含めない場合よりも精度良く予測できることが分かった．次に，各駅別で精度を比較した結果を示す．駅 Q での予測精度を図 4.15，駅 S での予測精度を図 4.16，駅 V での予測精度を図 4.17 に示す．なお，駅 Q の各停 2 における Case i，ii，iv，v の結果が存在しないのは，各停 2 が駅 Q 始発であり，入力データが存在しないことから予測ができないためである．全体では，駅 Q，駅 S は Case vi，駅 V は Case iii が最も良い精度となった．次に，駅 X までの各駅における予測精度を，全体については図 4.18 に，予測時点での駅別については図 4.19 から図 4.21 にそれぞれ示す．現在時刻から時間的に遠い，先の駅ほど，Case vi の予測精度が良いことが分かる．特に，4 駅先については，Case i，Case ii，Case iv，Case v の 4 ケースと Case vi との間に予測精度について有意な差が

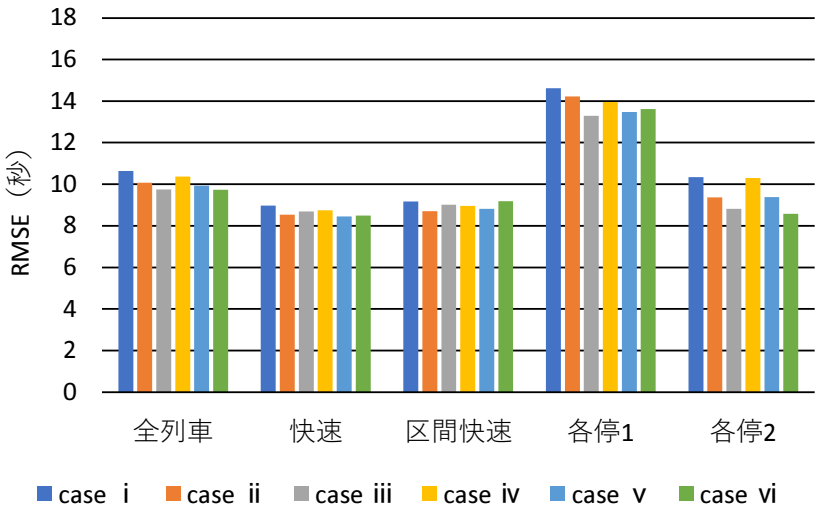

図 4.14 予測モデルの入力データの種類間の予測精度

あった．6 駅先についても，Case ii，Case v の 2 ケースと Case vi との間に予測精度の差に有意な傾向があった．Case iii については，他の 5 種類の Case 都の間に有意な差は確認できなかった．そのため，Case vi のように現在時刻から時間的に近い駅の遅延のみを入力とすることで精度良く予測できることが分かる．最後に，自駅から駅 X までの各駅の予測遅延と実績遅延の相関係数を図 4.22 から図 4.25 に示す．いずれの Case についてもほぼ同程度の値となった．図 4.23 では，入力データに前後列車を含める Case iii と Case vi の相関係数が，他の Case よりも悪くなっているが，これは，Case iii と Case vi のみ各停 2 を含んでいるため値が小さくなっているからであり，Case iii と Case vi の各停 2 を除いた相関係数の値は，他の Case とほぼ同じとなる．図 4.25 では，相関係数の値が全体的に小さいが，これは，サンプルサイズが 3 点のみと小さいためと思われる．なお，図 4.14 から図 4.25 の具体的な値は，付録 B.2 の表 B.4 から表 B.15 に示す．

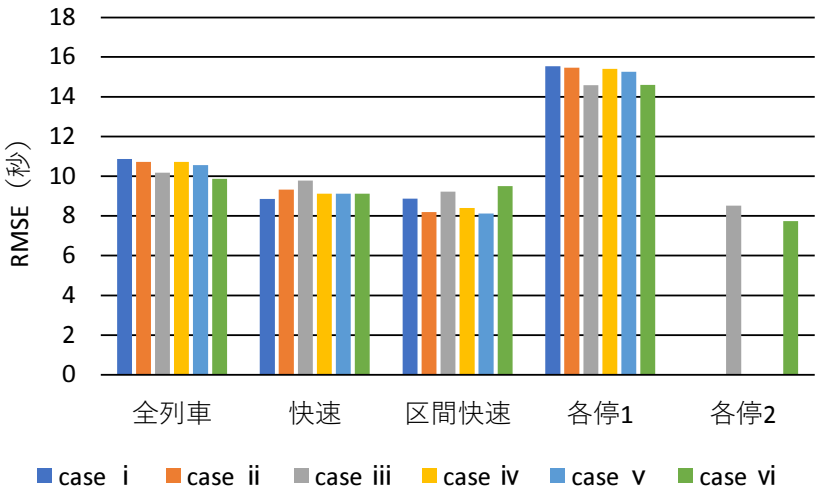

図 4.15 予測モデルの入力データの種類間の予測精度（駅 Q 着時点）

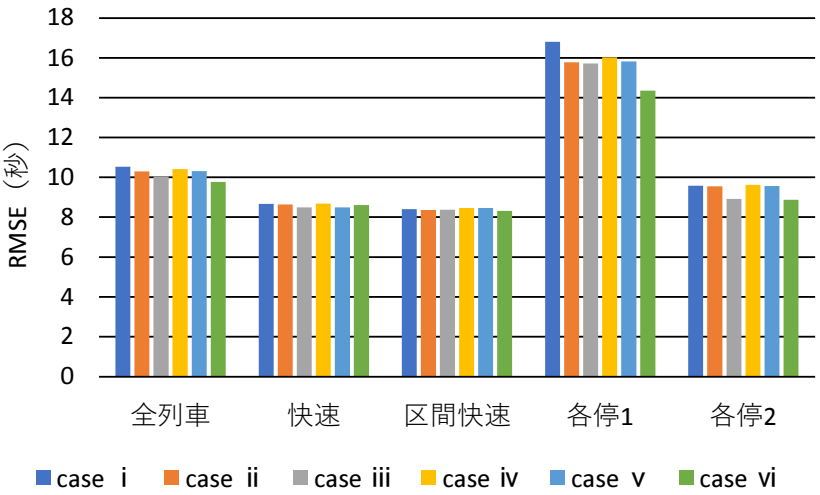

図 4.16 予測モデルの入力データの種類間の予測精度（駅 S 着時点）

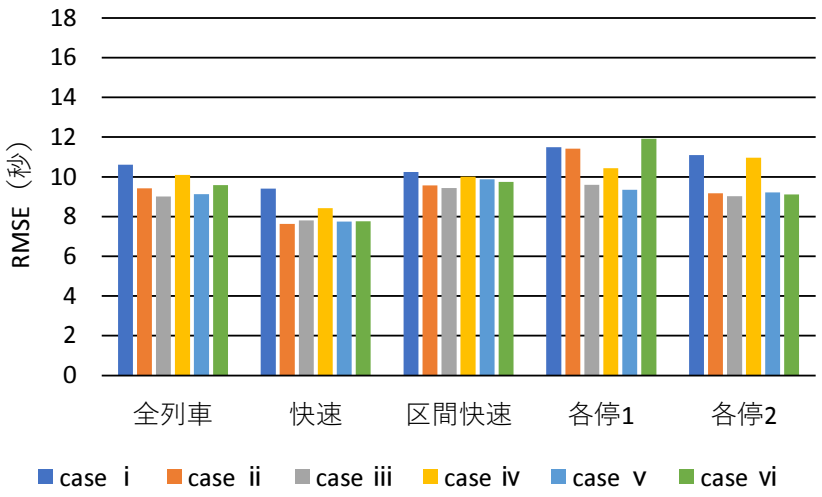

図 4.17 予測モデルの入力データの種類間の予測精度（駅 V 着時点）

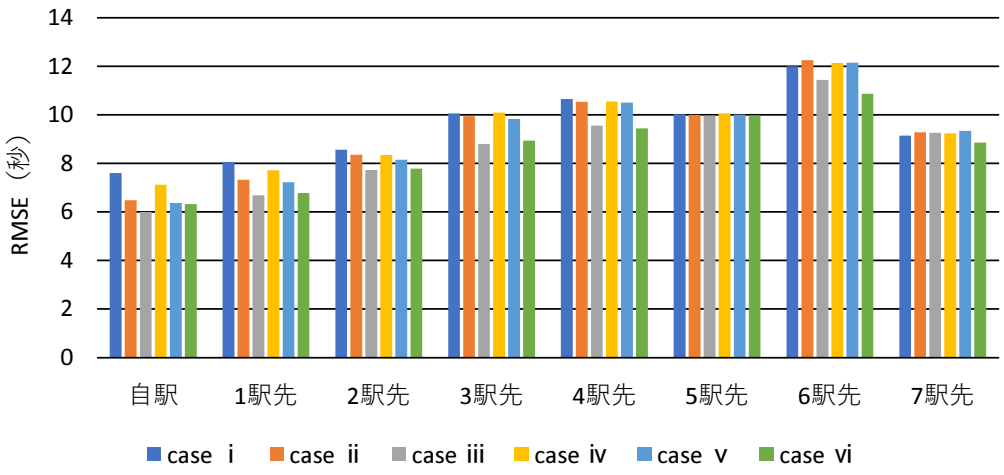

図 4.18 予測モデルの入力データの種類間に関する駅 X までの各駅の予測精度

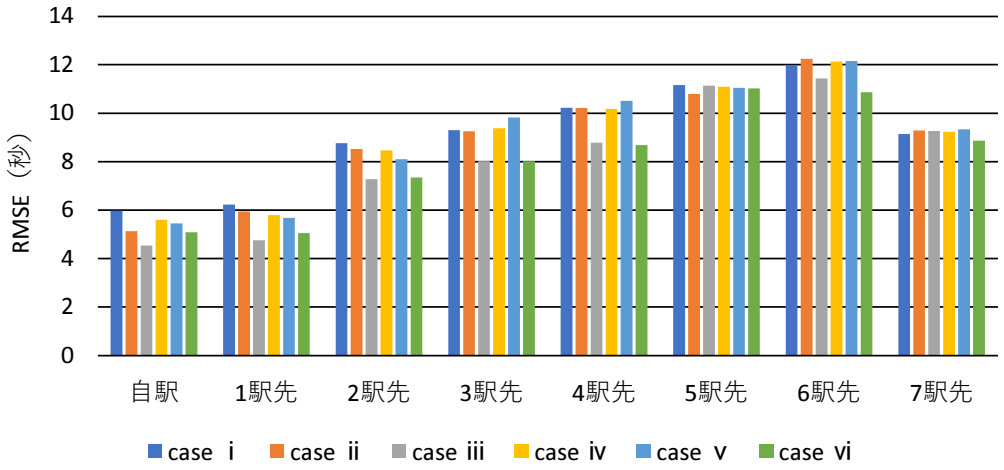

図 4.19 予測モデルの入力データの種類間に関する駅 X までの各駅の予測精度（駅 Q 着時点）

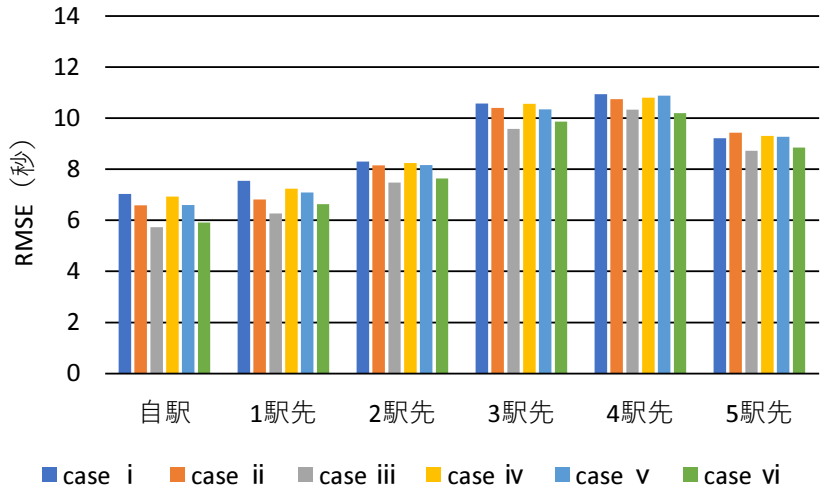

図 4.20 予測モデルの入力データの種類間に関する駅 X までの各駅の予測精度（駅 S 着時点）

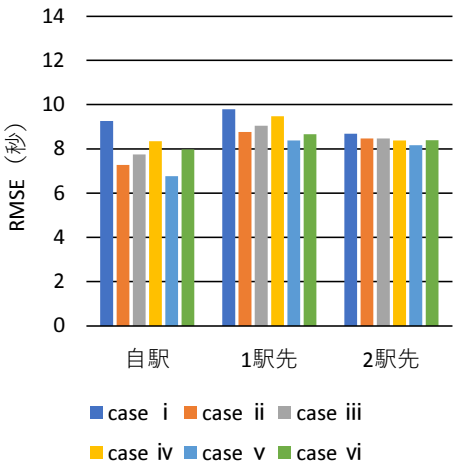

図 4.21 予測モデルの入力データの種類間に関する駅 X までの各駅の予測精度（駅 V 着時点）

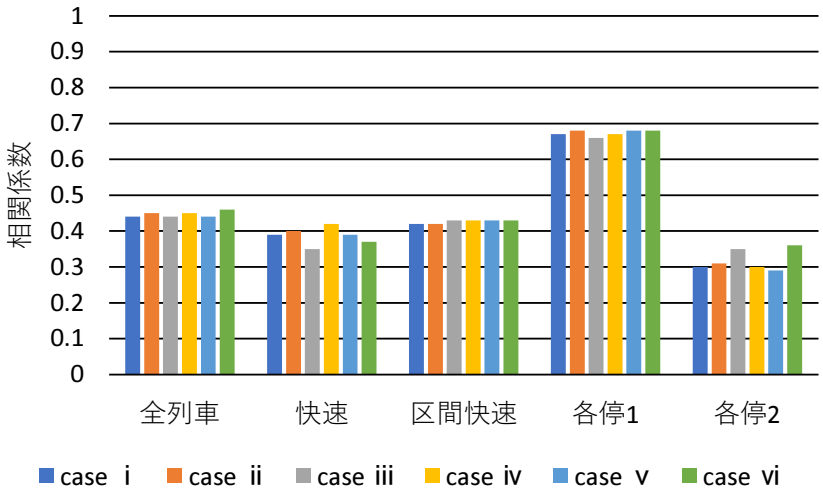

図 4.22 予測モデルの入力データの種類間の相関係数

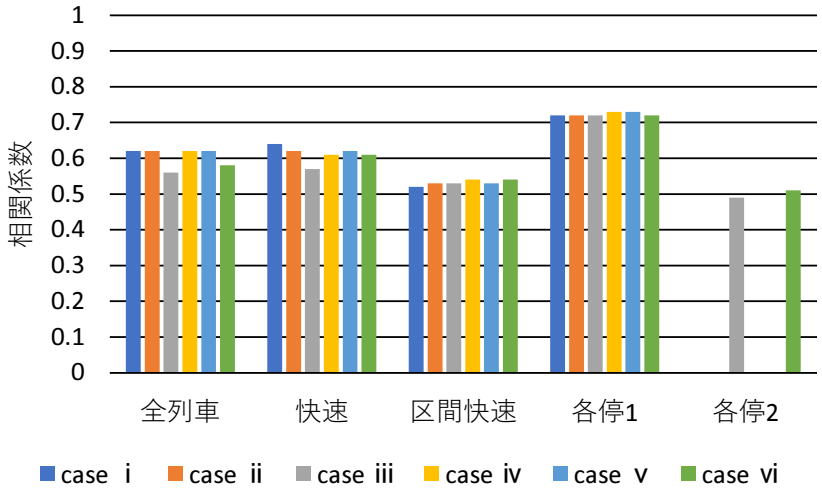

図 4.23 予測モデルの入力データの種類間の相関係数（駅 Q 着時点）

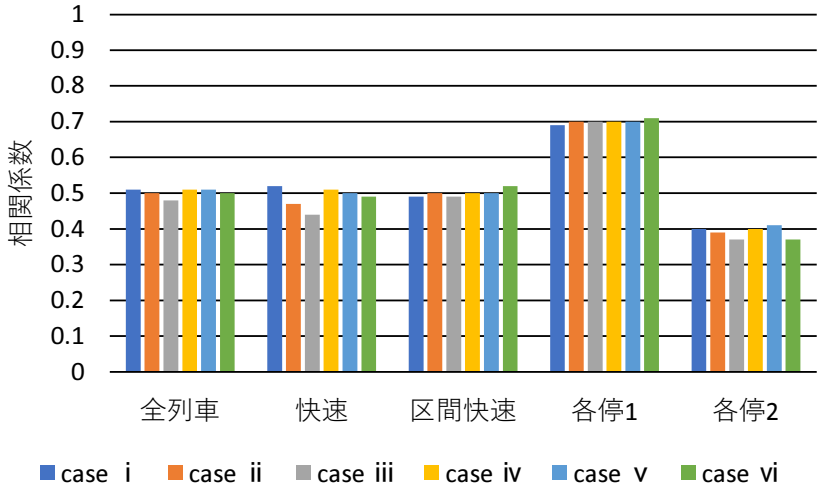

図 4.24 予測モデルの入力データの種類間の相関係数（駅 S 着時点）

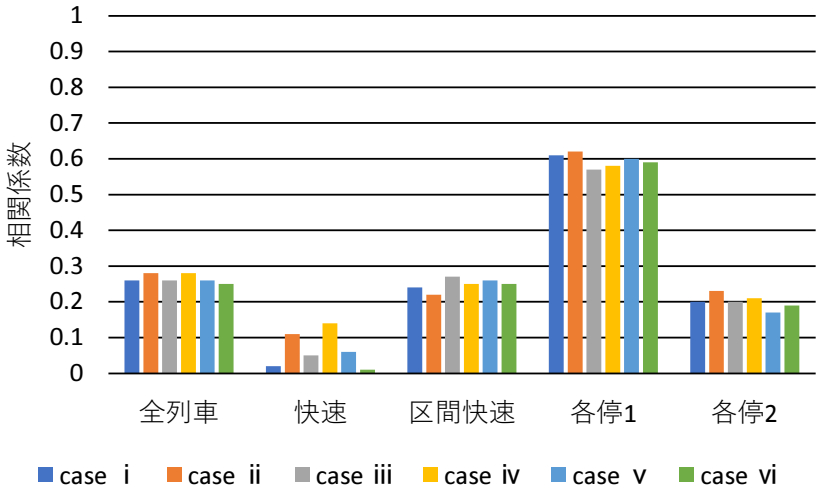

図 4.25 予測モデルの入力データの種類間の相関係数（駅 V 着時点）

4.4.6 他の予測手法との精度比較

本研究において提案する列車の遅延の予測モデル（以降，提案手法と呼ぶ.）の精度を，他の 2 つの手法の予測精度と比較する．1 つ目は，線形な予測モデルである重回帰分析を用いた予測手法である．ここでは，回帰式の説明変数および目的変数を，表 4.4 に示した入出力と同じとする．つまり，予測対象列車 T の i 番目の駅の遅延を予測する回帰式は以下のようになる．

$$\begin{aligned} d_{T_i} =& \beta_0 + \beta_1\, d_{T_{i-5}} + \beta_2\, d_{T_{i-4}} + \beta_3\, d_{T_{i-3}} + \beta_4\, d_{T_{i-2}} + \beta_5\, d_{T_{i-1}} \\ &+ \beta_6\, c_{T_{i-5}} + \beta_7\, c_{T_{i-4}} + \beta_8\, c_{T_{i-3}} + \beta_9\, c_{T_{i-2}} + \beta_{10}\, c_{T_{i-1}} \\ &+ \beta_{11}\, d_{F_{i-4}} + \beta_{12}\, d_{F_{i-3}} + \beta_{13}\, d_{F_{i-2}} + \beta_{14}\, d_{F_{i-1}} + \beta_{15}\, d_{F_i} \\ &+ \beta_{16}\, d_{L_{i-j-4}} + \beta_{17}\, d_{L_{i-j-3}} + \beta_{18}\, d_{L_{i-j-2}} + \beta_{19}\, d_{L_{i-j-1}} + \beta_{20}\, d_{L_{i-j}} \end{aligned} \tag{5}$$

式 5 の偏回帰係数 β_0 から β_{20} を，提案手法における学習・検証データとして用いている 64 日分の列車の遅延データおよび乗車率データから導出し，その求めた偏回帰係数を利用し，提案手法においてテストデータとして用いている 15 日分の列車の遅延データを予測する．なお，重回帰分析による予測では，$i+1$ 番目の駅から $i+4$ 番目の駅の遅延についても，式 5 に示した i 番目の駅の遅延を予測する回帰式と同一のものを用いる．つまり，式 5 を予測対象列車 T の i 番目の駅の遅延を予測する式 $d_{T_i} = f_i(d,\, c)$ とすると，$i+1$ 番目の駅の遅延を予測する式は，$d_{T_{i+1}} = f_i(d,\, c)$ である．各駅の遅延を予測する回帰式に対応する偏回帰係数 β_0 から β_{20} をそれぞれ導出し，遅延を予測する．$i+2$ 番目の駅から $i+4$ 番目の駅の遅延の予測式についても同様である．

2 つ目の予測手法は，関連研究の 4.2.3 節で説明した，Chapuis の提案するニューラルネットワークを用いた予測手法 [22] である．この手法は，予測対象列車 T の i 番目の駅の遅延 d_{T_i} を予測するために，列車 T が 1 つ前に停車した駅（その路線の始発駅から x 番目の駅とする）の発遅延 $d_{T_{i-x}}$，列車 T が 1 つ前に停車した x 番目の駅と i 番目の駅の間の駅数 $(i-x-1)$，i 番目の駅において列車 T の直前に発車した列車 F の i 番目における発遅延 d_{F_i} を入力データとする．入出力データ $Input_{Cha}$ および $Output_{Cha}$ は以下のようになる．また，この予測モデルを図 4.26 に示す．この予測モデルを列車種別，列車，駅，時間帯等に関係なく，全列車全駅のデータを用いて学習する．

$$\begin{aligned} Input_{Cha} =& (d_{T_{i-x}},\, i-x-1,\, d_{F_i}) \\ Output_{Cha} =& (d_{T_i}) \end{aligned} \tag{6}$$

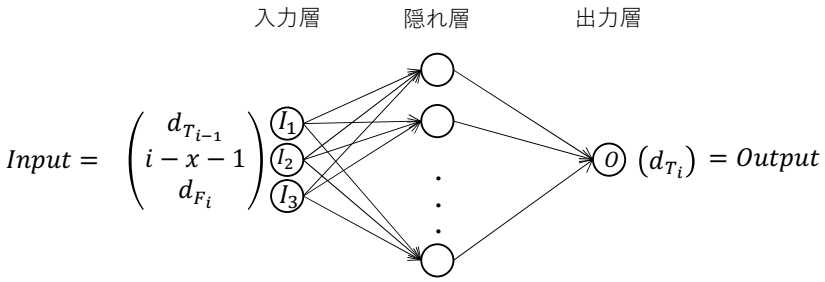

図 4.26 Chapuis が提案する予測モデル

本評価実験においても，4.4.5 での設定と同様に，駅 Q，S，V の着時点における，その駅からその路線の駅 X までの間の各駅の発遅延の予測精度を用いて評価する．なお，Chapuis の手法は，ある列車のある 1 つの駅の発遅延を予測するモデルである．そのため，まずは，提案手法と重回帰分析による予測手法について，駅別，列車種別別の予測精度を図 4.27 に示す．3 駅の全列車について，重回帰分析による予測手法よりも提案手法の方が有意に精度が良かった．列車種別別で見ると，郊外から運転されている快速の予測精度については，提案手法の方が精度よく予測ができている．次に，提案手法，重回帰分析，Chapuis の手法の 3 つの手法の予測精度の比較として，自駅から駅 X の遅延を予測した予測精度について，全体の結果を図 4.28，着駅別の結果を図 4.29 に示す．なお，Chapuis らの手法は，自駅の遅延の予測結果としてのみ示す．3 つの手法の予測精度に有意な差は見られなかったが，提案手法は，短時間先までの複数の駅の遅延を予測可能なモデルであるという点で，Chapuis の手法に対し優位性があると考える．図 4.28 の結果から，1 駅先については重回帰分析の方が有意に精度が良いが，5 駅先および 7 駅先の精度については，提案手法の方が有意に精度が良い．このことから，現在時刻から時間的に遠い，先の駅ほど，提案手法は精度良く予測ができることが分かる．また，図 4.29 のように着駅別で見ると，全体的に提案手法の方が精度よく予測ができている．最後に，図 4.30 に，駅別，列車種別別の相関係数を示す．提案手法の方が重回帰分析よりも有意に相関がある予測結果となったものが多く，提案手法の方が遅延の増減を予測できている．なお，図 4.27 から図 4.30 の具体的な値は，付録 B.3 の表 B.16 から表 B.19 に示す．

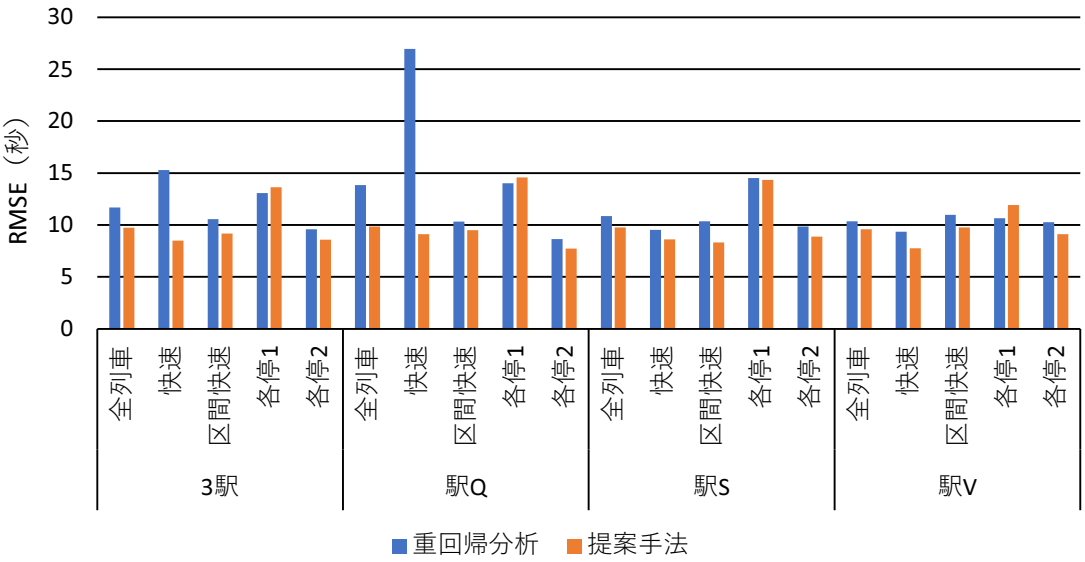

図 4.27 重回帰分析と提案手法の予測精度

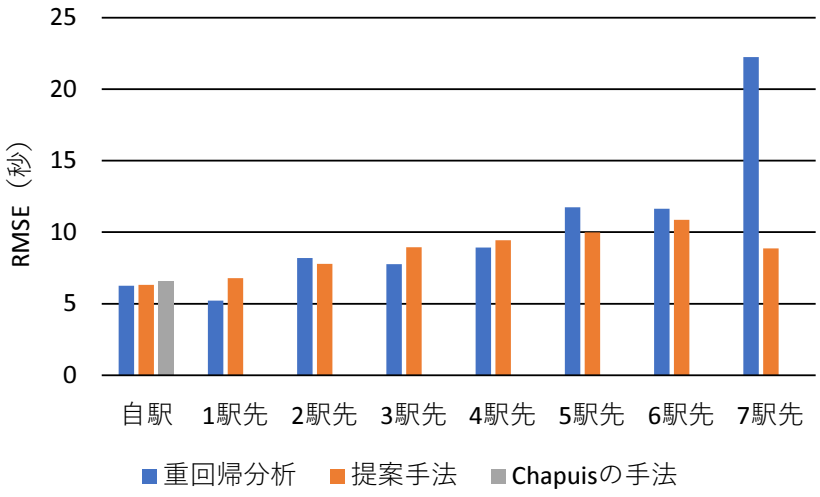

図 4.28 3 手法間における 4 駅先までの各駅の予測精度

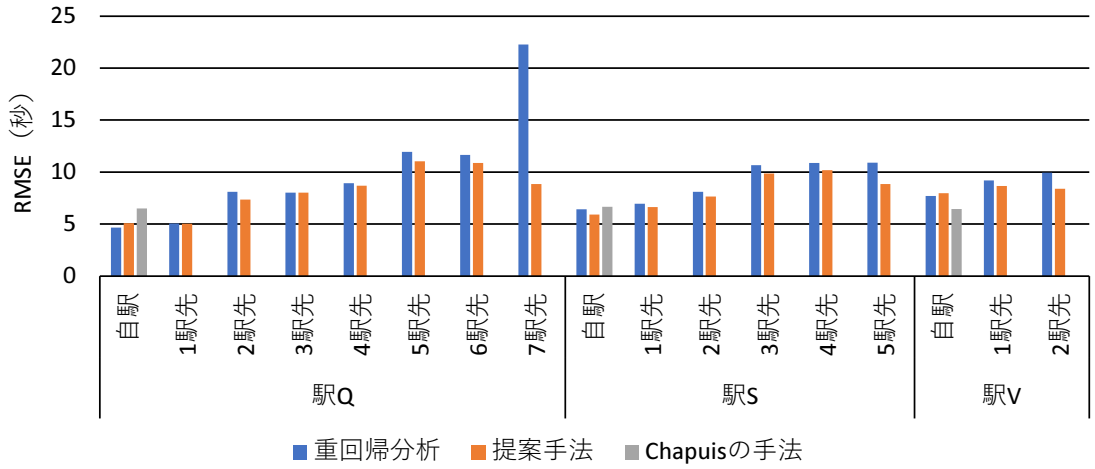

図 4.29 3 手法間における 4 駅先までの各駅の予測精度（着駅別）

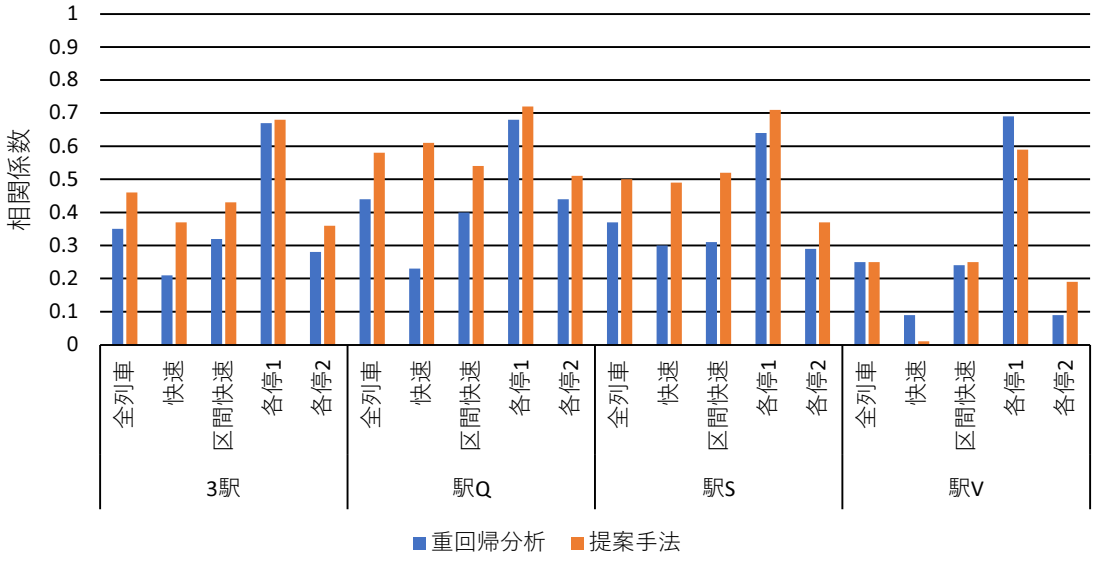

図 4.30 重回帰分析と提案手法の相関係数

4.5 まとめと今後の課題

本研究では，大都市近郊の通勤路線における朝ラッシュ時の小乱れ時を対象に，短時間先の列車遅延を高精度に予測する手法を提案した．具体的には，列車の遅延の時系列データを対象に，ニューラルネットワークを用いて，現時点から数駅前の予測対象列車の遅延と乗車率の時系列データ，および，予測対象列車の前後列車の遅延の時系列データから，短時間先までの間に予測対象列車が停車／通過する駅の遅延を予測する予測モデルを提案した．列車遅延を予測するモデルの単位，入力データの種類，および，他の予測モデル（重回帰分析による手法，[22] の研究において提案されている手法）との精度評価を行い，提案する予測モデルが他のモデルよりも列車の遅延の予測に適していることを確認した．

今後の課題として，下記に 5 点挙げる．

- より列車密度が高く，遅延の頻度や遅延量が大きい他の路線への，本研究での提案手法の適用．
- 乗車率のデータを [133] の対話型乗車率推定システムにより推定した値ではなく，車両に搭載された応荷重装置から取得できる乗車人数データを使用して予測精度を検証．
- 予測モデルの単位の検討の深度化．（今回は，各列車の各駅毎に遅延を予測するモデルを構築する方法を採用したが，4.3.1 節で述べたように，予測モデルの数が多くなり，1 つのモデルに対して学習に使えるデータが少なくなる．そのため，駅の規模や時間帯毎に予測モデルを用意するというように，予測モデルの単位について，さらに検討する．）
- 天候や曜日を反映した予測モデルの検討．
- 学習データが少ない状況下で高精度に予測可能なモデルの検討．（本研究で提案する予測モデルを用いて予測するためには，少なくとも数日分の学習データが必要となる．予測モデルの入出力は列車ダイヤの構造に依存しているため，列車ダイヤを変更するダイヤ改正直後では使用できる学習データが少ない．そこで，学習データが少ない状況でも精度よく予測できるモデルを検討する．）

今後は，上記に挙げた課題に取り組み，より高精度に列車の遅延を予測するモデルの構築を検討する．

5　本研究のまとめ

本研究では，属性として sequence_id，order_key，およびその他の属性を複数持つ多属性値の時系列データを対象に，行パターンマッチングの処理コストを削減する最適化手法の提案と，短時間先の列車遅延を予測する手法の提案を行った．行パターンマッチングの処理コストの削減においては，RDB 等に格納された大規模な時系列データに対する，SQL/RPR を用いた行パターンマッチングのクエリの処理コストを削減するために，MATCH_RECOGNIZE 句の DEFINE 句に指定された one variable condition に合致する行が 1 行もないシーケンスをフィルタリングする Sequence Filtering と，one variable condition に合致する行とその前後の行以外をフィルタリングする Row Filtering の 2 つの最適化手法を提案した．SQL/RPR を実装した PostgreSQL と Spark を用いて人工データと実データに対し評価実験を行い，これら 2 種類の最適化手法によって行パターンマッチングの処理コストが削減できることを確認した．また，各最適化手法のコストモデルを構築し，処理時間を見積もることができていることを確認した．

また，列車遅延の予測においては，慢性的な遅延が発生している大都市近郊の通勤路線における朝ラッシュ時の小乱れ時において，各列車，各駅における遅延を予測することを目的に，列車の遅延の時系列データを対象に，ニューラルネットワークを用いて，現時点から数駅前の予測対象列車の遅延と乗車率の時系列データ，および，予測対象列車の前後列車の遅延の時系列データから，短時間先までの間に予測対象列車が停車／通過する駅の遅延を予測する予測モデルを構築した．実在路線における列車の遅延を対象に評価実験を行い，提案するモデルを，モデルの構築単位，モデルの入出力，他の予測モデルと比較し，精度良く予測ができていることを確認した．

今後の課題として，行パターンマッチングの処理コスト削減のための最適化手法においては，行パターンマッチング以外のクエリ演算を含めたコストモデルの構築によるクエリ全体の処理の最適化や，より行パターンマッチング対象とする行を削減できる，処理コストの小さい最適化手法を検討することが挙げられる．また，ニューラルネットワークを用いた列車遅延予測手法においては，本手法を他路線へ適用した場合の精度検証や，天候や曜日を反映した予測モデルを検討することが挙げられる．これによって，より高精度に列車の遅延を予測するモデルの構築を目指す．

参考文献

[1] ISO/IEC 2016. Information technology database languages sql technical reports part 5: row pattern recognition in sql. Technical report, ISO copyright office, 2016.

[2] K. Abhishek, A. Kumar, R. Ranjan, and S. Kumar, "A Railfall Prediction Model using Artificial Neural Network," Proc. of the 2012 IEEE Control and System Graduate Research Colloquium (ICSGRC2012), pp.82–87, 2012.

[3] R. Agrawal and R. Srikant, "Mining Sequential Patterns," In Proc. of the 11th Internatoinal Conference on Data Engineering (ICDE1995), pp.3–14, 1995.

[4] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," In Proc. of the 20th International Conference on Very Large Data Bases (VLDB1994), pp.487-499, 1994.

[5] J. Agrawal, Y. Diao, D. Gyllstrom, and N. Immerman, "Efficient Pattern Matching over Event Streams," Proc. of the 2008 ACM SIGMOD International Conference on Management of Data, pp.147–160, 2008.

[6] M. A. Aguilera, R. E. Strom, D. C. Sturman, Mark Astley, and T. D. Chandra. "Matching Events in a Context-based Subscription System," In Proc. of the 18th Annual ACM Symposium on Principles of Distributed Computing (PODC1999), pp.53–61, 1999.

[7] A. Alahi, V. Ramanathan, K. Goel, A. Robicquet, A. Sadeghian, L. Fei-Fei, and S. Savarese, "Learning to Predict Human Behavior in Crowded Scenes," Group and Crowd Behavior for Computer Vision, pp.183–207, 2017.

[8] H. Albert-Lorincz and J. F. Boulicaut, "Mining Frequent Sequence Patterns under Regular Expressions: A Highly Adaptive Strategy for Pushing Constraints," In Proc of the 3rd SIAM International Conference on Data Mining, pp.316–320, 2003.

[9] M. Altinel and M. J. Franklin, "Efficient Filtering of XML Documents for Selective Dissemination of Information," In Proc. of the 26th International Conference on Very Large Data Bases (PVLDB2000), pp.53–64, 2000.

[10] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, and M. Zaharia, "Spark SQL: Relational Data

Processing in Spark," Proc. of the 2015 ACM SIGMOD International Conference on Management of Data, pp.1383–1394, 2015.

[11] F. Asakura, T. Matsumura, and T. Ono, "Basic Research of Predicting Train Operation Using Convolutional Neural Network (II)," IEE Japan, 4–C–p1–7 p.209, 2019.

[12] R. S. Barga, J. GoldStein, M. Ali, and M. Hong, "Consistem Streaming Through Time: A Vision for Event Stream Processing ," Proc. of the 3rd Biennial Conference on Innovative Data Systems Research (CIDR2007), pp.363–373, 2007.

[13] Y. Bin, Y. Zhongzhen, and Y. Baozhen, "Bus Arrival Time Prediction Using Support Vector Machine," Journal of Intelligent Transportation System, vol.10, issue 4, pp.151–158, 2006.

[14] G. E. P. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, "Time Series Analysis: Forecasting and Control," Third ed. Englewood Cliffs, NJ: Prentice Hall, 1994.

[15] V. L. Brano, G. Ciulla, and M. D. Falco, "Artificial Neural Networks to Predict the Power Output of a PV Panel," International Journal of Photoenergy, vo.2014, pp.1–12, 2014.

[16] Rashmi C. and Hemantha Kumar G., "Parallel Processing Approach for Pattern Matching using MPI," International Journal of Computer Applications, vol.180, no.11, pp.31–34, 2018.

[17] B. Cadonna, J. Gamper, and M. H. Böhlen, "Efficient Event Pattern Matching with Match Windows," Proc. of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD2012), pp.471–479, 2012.

[18] B. Cadonna, J. Gamper, and M. H. Böhlen, "Sequenced Event Set Pattern Matching," Proc. of the 14th International Conference on Extending Database Technology (EDBT2011), pp.33-44, 2011.

[19] Y. D. Cai, D. Clutter, G. Pape, J. Han, M. Welge, and L. Auvil, "MAIDS: Mining Alarming Incidents from Data Streams," Proc. of the ACM SIGMOD International Conference on Management of Data, pp.919–920, 2004.

[20] S. K. Cha, I. Moraru, J. Jang, J. Truelove, D. Brumley, and D. G. Anderson, "SplitScreen: Enabling Efficient, Distributed Malware Detection," Journal of Communications and Networks, Vol.13, No.2, pp.187–200, 2011.

[21] S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S. K. Kim, "Composite

Events for Active Databases: Semantics, Contexts and Detection," Proc. of the 20th International Conference on Very Large Data Bases (PVLDB1994), pp.606–617, 1994.

[22] X. Chapuis, "Arrival Time Prediction using Neural Networks ," Proc. of the 7th International Conference on Railway Operations Modeling and Analysis (RailLille2017), pp.1500–1510, 2017.

[23] X. Chen and Y. Zhan, "Multi-scale Anomaly Detection Algorithm based on Infrequent Pattern of Time Series," Journal of Computational and Applied Mathematics, vol.214, no.1, pp.227-237, 2008.

[24] B. Choi, J. Chae, M. Jamshed, K. Park, and D. Han, "DFC: Accelerating String Pattern Matching for Network Applications," Proc. of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI2016), pp.551–565, 2016.

[25] J. Contreras, R. Espinola, F. J. Nogales, and A. J. Conejo, "ARIMA Models to Predict Next-day Electricity Prices," IEEE Transactions on Power Systems, vol.18, issue 3, pp.1014–1020, 2002.

[26] S. Deaton, D. Brownfield, L. Kosta, Z. Zhu, and S. J. Matthews, "Real-time Regex Matching with Apache Spark," Proc. of the 2017 IEEE High Performance Extreme Computing Conference (HPEC2017), pp.1–6, 2017.

[27] A. Demers, J. Gehrke, B. Panda, M. Riedewald, V. Sharma, and W. White, "Cayuga: A General Purpose Event Monitoring System," Proc. of the 3rd Biennial Conference on Innovative Data Systems Research (CIDR2007), pp.412–422, 2007.

[28] N. Dindar, B. Güç, P. Lau, A. Özal, M. Soner, and N. Tatbul, "DejaVu: Declarative Pattern Matching over Live and Archived Streams of Events," Proc. of the 2009 ACM SIGMOD International Conference on Management of Data, pp.1023–1026, 2009.

[29] L. Ding, S. Chen, E. A. Rundensteiner, J. Tatemura, W. Hsiung, and K. S. Candan, "Runtime Semantic Query Optimization for Event Stream Processing," IEEE 24th International Conference on Data Engineering (ICDE2008), pp.676–685, 2008.

[30] P. Esling and C. Agon, "Time-Series Data Mining," ACM Computing Survey, vol.45, no.1, article 12, 2012.

[31] F. Fabret, H. A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha, "Filtering Algorithms and Implementation for Very Fast Publish/Subscribe Systems," In Proc of the 2001 ACM SIGMOD International Conference on Management of Data (SIGMOD2001), pp.115–126, 2001.

[32] I. Forain, R. O. Albuquerque, A. S. Orozco, L. G. Villalba, and T. Kim, "Endpoint Security in Networks: An OpenMP Approach for Increasing Malware Detection Speed," Symmetry, vol.9, no.9, pp.172, 2017.

[33] "Foursquare," 2018, https://foursquare.com.

[34] M. N. Garofalakis, R. Rastogi, and K. Shim, "SPIRIT: Sequential Pattern Mining with Regular Expression Constraints," Proc. of the 25th International Conference on Very Large Data Bases (VLDB1999), pp.223-234, 1999.

[35] N. H. Gehani, H. V. Jagadish, and O. Shmueli, "Composite Event Specification in Active Databases: Model & Implementation," Proc. of the 18th International Conference on Very Large Data Bases (VLDB1992), pp.327–338, 1992.

[36] C. Giannella, J. Han, J. Pei, X. Yan, and P. S. Yu, "Mining Frequent Patterns in Data Streams at Multiple Time Granularities," Next Generation Data Mining, pp.191–212, 2003.

[37] F. E. Gmati, S. Chakhar, W. L. Chaari, and H. Chen, "A Rough Set Approach to Events Prediction in Multiple Time Series," Proc. of the 31st International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems (IEA/AIE2018), pp.796–807, 2018.

[38] E. Grigonytė and E. Butkevičiūtė, "Short-term Wind Speed Forecasting using ARIMA Model," Energetika, vol.62, issue 1-2, pp.45–55, 2016.

[39] R. Grossi, C. S. Iliopoulos, C. Liu, N. Pisanti, S. P. Pissis, A. Retha, G. Rosone, F. Vayani, and L. Versari, "On-Line Pattern Matching on Similar Texts," Proc. of the 28th Annual Symposium on Combinatorial Pattern Matching (CPM2017), pp.7–8, 2017.

[40] Z. K. Gurmu and W. Fan, "Artificial Neural Network Travel Time Prediction Model for Buses Using Only GPS Data," Journal of Public Transportation vol.17, no.2, pp.45–65, 2014.

[41] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M. C. Hsu, "FreeSpan: Frequent Pattern-Projected Sequential Pattern Mining," In Proc. of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,

pp.355–359, 2000.

[42] J. Han, J. Pei, and Y. Yin, "Mining Frequent Patterns by Pattern-Growth: Methodology and Implications," ACM SIGKDD Explorations Newsletter - Special Issue on "Scalable Data Mining Algorithm," vol.2, issue 2, pp.14–20, 2000.

[43] L. Heydenrijk-Ottens, V. Degeler, D. Luo, N. V. Oort, and H. V. Lint, "Supervised Learning: Predicting Passenger Load in Public Transport," Conference on Advanced Systems in Public Transport (CAPST2018), 2018.

[44] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," NEURAL COMPUTATION, vol. 9, issue 8, pp.1735–1780, 1997.

[45] F. Höppner, "Discovery of Temporal Patterns. Learning Rules about the Qualitative Behaviour of Time Series," In Proc. of the 5th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD2001), pp.192–203, 2001.

[46] S. Innamaa, "Short-Term Prediction of Highway Travel Time Using MLP_Neural Networks," Proc. of the 8th World Congress on Intelligent Transportation Systems, vol. 32, no. 6, pp.649–669, 2001.

[47] I. Jindal, T. Z. Qin, X. Chen, M. Nokleby, and J. Ye, "A Unified Neural Network Approach for Estimating Travel Time and Distance for Taxi Trip," arXiv preprint arXiv:1710.04350 [stat. ML], 2017.

[48] C. Ju, H. Li, and F. Li, "PCG: An Efficient Method for Composite Pattern Matching over Data Streams," Proc. of the International Conference on Advanced Data Mining and Applications (ADMA2012), pp.488–501, 2012.

[49] S. Kato, D. Amagata, S. Nishio, and T. Hara, "Monitoring Range Motif on Streaming Time-Series," Proc. of the 29th International Conference on Database and Expert Systems Applications (DEXA2018), pp.251–266, 2018.

[50] A. Kazem, E. Sharifi, F. K. Hussain, M. Saberi, and O. K. Hussain, "Support Vector Regression with Chaos-based firefly Algorithm for Stock Market Price Forecasting," Applied Soft Computing, vol.13, issue 2, pp.947–958, 2012.

[51] A. Kehagias, "A Hidden Markov Model Segmentation Procedure for Hydrological and Environment Time Series," Stochastic Environment Research and Risk Assessment, vol.18, no.2, pp.117–130, 2004.

[52] I. Khandelwal, R. Adhikari, and G. Verma, "Time Series Forecasting using Hybrid ARIMA and ANN Models based on DWT Decomposition," Interna-

tional Conference on Intelligent Computing, Communication & Convergence (ICCC2015), Procedia Computer Science, vol.48, pp.173–179, 2015.

[53] I. Kolchinsky and A. Schuster, "Efficient Adaptive Detection of Complex Event Patterns," Proc. of the VLDB Endowment, vol.11, no.11, 2018.

[54] I. Kolchinsky, I. Sharfman, and A. Schuster, "Lazy Evaluation Methods for Detecting Complex Events," Proc. of the 9th ACM International Conference on Distributed Event-Based Systems (DEBS2015), pp.34–45, 2015.

[55] K. Laker, "A Technical Deep Dive into Pattern Matching using MATCH_RECOGNIZE," HrOUG 2016, Business Intelligence & Analytics, Hall C, 19 October 2016, http://www.oracle.com/technetwork/database/bi-datawarehousing/mr-deep-dive-3769287.pdf.

[56] H. T. Lam, T. Calders, and N. Pham, "Online Discovery of Top-k Similar Motifs in Time Series Data," Proc. of the 11th SIAM International Conference on Data Mining, pp.1004-1015, 2011.

[57] A. Lamb, M. Fuller, R. Varadarajan, N. Tran, B. Vandiver, L. Doshi, and C. Bear, "The Vertica Analytic Database: C-Store 7 Years Later," Proc. of the 38th International Conference on Very Large Databases (PVLDB2012), vol.5, no.12, pp.1790–1801, 2012.

[58] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, and W. Hubbard, "Backpropagation Applied to Handwritten Zip Code Recognition," Neural Computation, vol. 1, no. 4, pp.541-551, 1989.

[59] J. Lin, E. Keogh, S. Lonardi, and P. Patel, "Finding Motifs in Time Series," Proc. of the 2nd International Workshop on Temporal Data Mining, pp.53–68, 2002.

[60] B. Liu and J. Liu, "Multivariate Time Series Prediction via Temporal Classification," Proc. of the 18th International Conference on Data Engineering (ICDE2002), pp.268–275, 2002.

[61] A. M. D. Livera, R. J. Hyndman, and, R. D. Snyder, "Forecasting Time Series with Complex Seasonal Patterns using Exponential Smoothing," Journal of the American Statistical Association, 106(496), pp.1513–1527, 2011.

[62] J. Ma, and S. Perkins, "Online Novelty Detection on Temporal Sequences," In Proc of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD2003), pp.613–618, 2003.

[63] H. Mannila, H. Toivonen, and A. Verkamo, “Discovering Frequent Episodes in Sequences,” In Proc. of the 1st International Conference on Knowledge Discovery and Data Mining (KDD1995), pp.210–215, 1995.

[64] N. Marković, S. Milinković, K. S. Tikhonov, and P. Schonfeld, “Analyzing Passenger Train Arrival Delays with Support Vector Regression,” Transportation Research Part C Emerging Technologies, vol.56, pp.251–262, 2015.

[65] Y. Matsubara, Y. Sakurai, and C. Faloutsos, “AutoPlait: Automatic Mining of Co-evolving Time Sequences,” Proc. of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD2014), pp.193–204, 2014.

[66] Y. Matsubara and Y. Sakurai, “Regime Shifts in Streams: Real-time Forecasting of Co-evolving Time Sequences,” Proc. of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD2016), pp.1045–1054, 2016.

[67] Y. Mei and S. Madden, “ZStream: A Cost-based Query Processor for Adaptively Detecting Composite Events,” Proc. of the 2009 ACM SIGMOD International Conference on Management of data, pp.193–206, 2009.

[68] Y. Mohammad and T. Nishida, “Constrained Motif Discovery in Time Series,” New Generation Computing, vol.27, no.4, pp.319–346, 2009.

[69] C. H. Mooney and J. F. Roddick, “Sequential Pattern Mining – Approaches and Algorithms,” ACM Computing Survey, vol.45, no. 2, article 19, 2013.

[70] W. Mou, Z. Cheng, and C. Wen, “Predictive Model of Train Delays in a Railway System,” Proc. of the 8th International Conference on Railway Operations Modelling Analysis (RailNorrköping), pp.913–929, 2019.

[71] A. Mueen and E. Keogh, “Online Discovery and Maintenance of Time Series Motifs,” Proc. of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD2010), pp.1089–1098, 2010.

[72] A. Mueen, E. Keogh, Q. Zhu, S. Cash, and B. Westover, “Exact Discovery of Time Series Motifs,” In Proc. of the SIAM International Conference on Data Mining (SDM2009), pp.473–484, 2009.

[73] Y. Nasu, H. Kitagawa, and K. Nakabasami, “Efficient Pattern Matching using Pattern Hierarchies for Sequence OLAP,” Proc. of the 21st International Conference on Big Data Analytics and Knowledge Discovery (DaWaK2019), pp.89–104, 2019.

[74] G. Navarro and M. Raffinot, "A Bit-parallel Approach to Suffix Automata: Fast Extended String Matching," Annual Symposium on Combinatorial Pattern Matching (CPM1998), pp.14–33, 1998.

[75] B. Padmanabhan and A. Tuzhilin, "Pattern Discovery in Temporal Databases: A Temporal Logic Approach," In Proc. of the 2nd International Conference on Knowledge Discovery and Data Mining, pp.351–354, 1996.

[76] P. F. Pai and C. S. Lin, "A Hybrid ARIMA and Support Vector Machines Model in Stock Price Forecasting," Omega-International Journal of Management Science, vol.13, issue 6, pp.497–505, 2005.

[77] T. Palpanas, M. Vlachos, E. Keogh, and D. Gunopulos, "Streaming Time Series Summarization using User-defined Amnesic Functions," IEEE Transactions on Knowledge and Data Engineering, vol.20, issue 7, pp.992-1006, 2008.

[78] J. Pang, J. Huang, Y. Du, H. Yu, Q. Huang, and B. Yin "Learning to Predict Bus Arrival Time from Heterogeneous Measurements via Recurrent Neural Network," IEEE Transactions on Intelligent Transportation Systems, vol.20, issue 9, pp.3283–3293, 2019.

[79] P. Papapetrou, G. Kollios, S. Sclaroff, and D. Gunopulos, "Mining Frequent Arrangements of Temporal Intervals," Knowledge and Information Systems, vol.21, issue 2, pp.133-171, 2009.

[80] K. Pasini, M. Khouadjia, F. Ganansia, and L. Oukhellou, "Forecasting Passenger Load in a Transit Network using Data Driven Models," Proc. of the 25h World Congress on Railway Research (WCRR2019), OP_04, HAL Id: hal-02278238, 2019.

[81] P. Patel, E. Keogh, J. Lin, and S. Lonardi, "Mining Motifs in Massive Time Series Databases," Proc. of IEEE International Conference on Data Mining (ICDM2002), pp.370–377, 2002.

[82] R. Pears, H. Widipurta, and N. K. Kasabov, "Evolving Integrated Multi-model Framework for On Line Multiple Time Series Prediction," Evolving Systems, vol.4, issue 2, pp.99–117, 2013.

[83] P. Pecev and M. Racković, "LTR-MDTS Structure - A structure for Multiple Dependent Time Series Prediction," Computer Science & Information Systems, vol.14, issue 2, pp.467–490, 2017.

[84] J. Pei, J. Han, B. Mortazavi-Asl, H. Pint, Q. Chen, U. Dayal, and M. C. Hsu,

"PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth," In Proc. of the International Conference on Data Engineering (ICDE2001), pp.215–226, 2001.

[85] C. S. Perng and D. S. Parker, "SQL/LPP: A Time Series Extension of SQL Based on Limited Patience Patterns," Proc. of the 10th International Conference on Database and Expert Systems Applications (DEXA1999), pp.218-227, 1999.

[86] S. Poornima and P. Marudappa, "Prediction of Railfall using Intensified LSTM based Recurrent Neural Network with Weighted Linear Units," Atmosphere, vol.10, issue 11, pp.668, 2019.

[87] P. Procházka and J. Holub, "Byte-Aligned Pattern Matching in Encoded Genomic Sequences," Proc. of the 17th International Workshop on Algorithms in Bioinformatics (WABI2017), pp.20:1–20:13, 2017.

[88] R. Sadri, C. Zaniolo, A. Zarkesh, and J. Adibi, "Optimization of Sequence Queries in Database Systems," Proc of the 20th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS2001), pp.71–81, 2001.

[89] R. Sadri, C. Zaniolo, A. Zarkesh, and J. Adimi, "Expressing and Optimizing Sequence Queries in Database Systems," ACM Transactions on Database Systems, vol.29, no.2, pp.282–318, 2004.

[90] İ. Şahin, "Markov Chain Model for Delay Prediction of Trains," Proc. of the 8th International Conference on Railway Operations Modelling Analysis (RailNorrköping), pp.1460–1479, 2019.

[91] İ. Şahin, "Markov Chain Model for Delay Distribution in Train Schedules: Assessing the Effectiveness of Time Allowances," Journal of Rail Transport Planning & Management, vol. 7, pp.101–113, 2017.

[92] H. Shatkay and S. Zdonik, "Approximate Queries and Representations for Large Data Sequences," Proc. of the 12th International Conference on Data Engineering (ICDE1996), pp.536–545, 1996.

[93] E. Sitaridi, O. Polychroniou, and K. A. Ross, "SIMD-Accelerated Regular Expression Matching," Proc. of the 12th International Workshop on Data Management on New Hardware (DaMoN2016), Article No. 8, 2016.

[94] Q. Song and N. K. Kasabov, "ECM - A Novel On-line, Evolving Clustering Method and Its Applications," Ponser, M. I. (ed.) Foundations of Cognitive Science, pp.631–682, MIT Press, 2001.

[95] R. Srikant and R. Agrawal, "Mining Sequential Patterns: Generalizations and Performance Improvements," In Proc. of the 5th International Conference no Extending Database Technology (EDBT1996), vol.1057, pp.3–17, 1996.

[96] S. C. Tan, K. M. Ting, and T. F. Liu, "Fast Anomaly Detection for Streaming Data," Proc. of the 22nd International Joint Conference on Artificial Intelligence (IJCAI2011), vol.2, pp.1511–1516, 2011.

[97] W. G. Teng, M. S. Chen, and P. S. Yu, "A Regression-based Temporal Pattern Mining Scheme for Data Streams," Proc. of the 29th International Conference on Very Large Data Bases (VLDB2003), pp.93–104, 2003.

[98] F. Toqué, E. Côme, M. K. E. Mahrsi, and L. Oukhellou, "Forecasting Dynamic Public Transport Origin-Destination Matrices with Long-Short Term Memory Recurrent Neural Networks," Proc. of the IEEE 19th International Conference on Intelligent Transportation System (ITSC2016), pp.1071–1076, 2016.

[99] M. Valipour, M. E. Banihabib, and S. M. R. Behbahani, "Comparison of the ARMA, ARIMA, and the Autoregressive Artificial Neural Network Models in Forecasting the Monthly Inflow of Dez Dam Reservoir," Journal of Hydrology, vol.476, pp.433–441, 2013.

[100] V. N. Vapnik, "The Nature of Statistical Learning Theory," Springer-Verlag, New York, 1995.

[101] H. L. Wang and K. Y. Chen, "One-dimensional Approximate Point Set Pattern Matching with Lp-norm," Journal of Theoretical Computer Science, vol.521, pp.42–50, 2014.

[102] P. Wang, C. Wu, and X. Gao, "Research on Subway Passenger Flow Combination Prediction Model based on RBF Neural Networks and LSSVM," Chinese Control and Decision Conference (CCDC), 2016 Chinese.IEEE, pp.6064–1661, 2016.

[103] X. Wang, Y. Hong, H. Chang, K. Park, G. Langdale, J. Hu, and H. Zhu, "Hyperscan: A Fast Multi-pattern Regex Matcher for Modern CPUs," Proc. of the 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI2019), pp.631–648, 2019.

[104] H. Widiputra, "Nearest Neighbor Approach with Non-parametric Regression Analysis for Multiple Time-series Modelling and Predictions," International Journal of Bussiness Intelligence and Data Mining, vol.10, no.3, pp.253–279, 2015.

[105] H. Widipurta, R. Pears, and N. K. Kasabov, "Multiple Time-series Prediction through Multiple Time-series Relationships Profiling and Clustered Recurring Trends," Proc. of the 15th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining (PAKDD2011), pp.161–172, 2011.

[106] R. J. Williams and D. Zipser, "Gradient-Based Learning Algorithms for Recurrent Networks and Their Computational Complexity," Back-propagation: Theory, Architectures and Applications, Hillsdale, NJ: Erlbaum, 1995.

[107] C. H. Wu, J. M. Ho, and D. T. Lee, "Travel-Time Prediction with Support Vector Regression," IEEE Transaction on Intelligent Transportation Systems, vol.5, no.4, pp.276–281, 2005.

[108] E. Wu, Y. Diao, and S. Rizvi, "High-Performance Complex Event Processing over Streams", SIGMOD 2006, pp.407–418, 2006.

[109] Y. X. Wu, Q. B. Wu, and J. Q. Zhu, "Improved EEMD-based Crude Oil Price Forecasting using LSTM Networks," Physica A: Statistical Mechanics and its Applications, vol.516, no.15, pp.114–124, 2019.

[110] D. Yang, D. Zhang, L. Chen, and B. Qu, "NationTelescope: Monitoring and Visualizing Large-Scale Collective Behavior in LBSNs," Journal of Network and Computer Applications (JNCA), vol.55, pp.170–180, 2015.

[111] D. Yang, D. Zhang, and B. Qu, "Participatory Cultural Mapping Based on Collective Behavior Data in Location Based Social Networks," ACM Trans. on Intelligent Systems and Technology (TIST), 2015.

[112] H. Yang, L. Chan, and I. King, "Support Vector Machine Regression for Volatile Stock Market Prediction," Proc. of the 3rd International Conference on Intelligent Data Engineering and Automated Learning (IDEAL2002), pp.391–396, 2002.

[113] D. Yankov, E. Keogh, and U. Rebbapragada, "Disk Aware Discard Discovery: Fiding Unusual Time Series in Terabyte Sized Datasets," Knowledge and Information Systems, vol.17, no.2, pp.241–262, 2008.

[114] M. Yaghini, M. M. Khoshraftar, and M. Seyedabadi, "Railway Passenger Train Delay Prediction via Neural Network Model," Journal of Advanced Transportation, vol.47, no.3, pp.355–368, 2013.

[115] B. Yi, N. D. Sidiropoulos, T. Johnson, H. V. Jagadish, C. Faloutsos, and A. Biliris, "Online Data Mining for Co-Evolving Time Sequences," Proc. of the 16th

Internatinal Conference on Data Engineering (ICDE2000), pp.13–22, 2000.

[116] A. Ypma and R. Duin, “Novelty Detection using Self-organizing Maps,” Proc. of the International Conference on Neural Information Processing, pp.1322-1325, 1997.

[117] J. Yu and J. Li, “A Parallel NIDS Pattern Matching Engine and Its Implementation on Network Processor,” Proc. of the 2005 International Conference on Security and Management (SAM2005), pp.375–384, 2005.

[118] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stonica, “Spark: Cluster Computing with Working Sets,” Proc. of the 2nd USENIX Conference on Hot Topics in Cloud Computing (HotCloud2010), vol.55, p.10, 2010.

[119] M. J. Zaki, “SPADE: An Efficient Algorithm for Mining Frequent Sequences,” Machine Learning, vol.42, issue1–2, pp.31–60, 2001.

[120] G. P. Zhang, “Time Series Forecasting using a Hybrid ARIMA and Neural Network Model,” Neurocomputing, vol.50, issue 50, pp.159–175, 2003.

[121] G. P. Zhang and V. L. Berardi, “Time Series Forecasting with Neural Network Ensembles: An Application for Exchange Rate Prediction,” Journal of the Operational Research Society, vol.52, issue 6, pp.652–664, 2001.

[122] Y. Zhang, Y. Wang, and G. Luo, “A New Optimization Algorithm for Non-stationary Time Series Prediction based on Recurrent Neural Networks,” Future Generation Computer Systems, vol.102, pp.738–745, 2020.

[123] J. Zhou and A. K. H. Tung, “SMiLer: A Semi-Lazy Time Series Prediction System for Sensors,” Proc. of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD2015), pp.1871–1886, 2015.

[124] Y. Zhou, C. Ma, Q. Guo, L. Shou, and G. Chen, “Sequence Pattern Matching over Event Data with Temporal Uncertainty,” Proc. of the 17th International Conference on Extending Database Technology (EDBT2014), pp.205–216, 2014.

[125] 電気学会・鉄道における運行計画・運行管理業務高度化に関する調査専門委員会, 鉄道ダイヤ回復の技術，オーム社，2010.

[126] 朝倉扶三郎，松村太陽，小野隆，“ CNN を用いた列車の運行予測における精度の比較検討，” 第 26 回鉄道技術連合シンポジウム (J-RAIL2019)，S8-1，pp.288–289，2019.

[127] 安部恵介，荒屋真二，“ 最長径路法を用いた列車運行シミュレーション，” 情報処理学会論文誌，” vol.27，no.1，pp.103–111，1986.

[128] 岩倉成志，高橋郁人，森地茂，“ 都市鉄道の遅延連鎖予測のためのエージェントシミュレーション:田園都市線および半蔵門線を対象に，” 運輸政策研究，vol.15，no.4，pp.31–40，2013.

[129] 岩本章寛，佐藤剛士，弓田康弘，溝口和人，安河内崇，福井清純，“ 予測ダイヤからの運転整理入力機能の開発，” 鉄道サイバネ・シンポジウム論文集，vol.51，p.5，2014.

[130] 國松武俊，平井力，富井規雄，“ マイクロシミュレーションを用いた利用者の視点による列車ダイヤ評価手法，” 電気学会論文誌 D（産業応用部門），vol.130，no.4，pp.459–467，2010.

[131] 小林渉，岩倉成志，“ 駅構造を組み込んだ列車遅延シミュレーションの開発，” 土木学会論文集 D3（土木計画学），vol.72，no.5（土木計画学研究・論文集第 33 巻），pp.I_1067–I_1074，2016.

[132] 高垣良宏，荻原崇，倉林修一，清水康，“ 鉄道における乗車率予測手法とグリーン車の乗車予測データを用いた実証実験，” 第 8 回データ工学と情報マネジメントに関するフォーラム (DEIM2016)，D1-4，2016.

[133] 辰井大祐，國松武俊，石原裕介，坂口隆，“ 乗車率推定機能を有する対話型ダイヤ作成システムの構築，” 電気学会研究会資料 (TER)，交通電気鉄道研究会，vol.48，pp.23–28，2012.

[134] 長崎祐作，明日香昌，駒谷喜代俊，“ 鉄道における乗客流推定方式の高速化，” 電気学会論文誌 C，vol.126，no.11，pp.1406–1413，2006.

[135] 中挾晃介，武内陽子，小川知行，影山真佐富，東海勝人，平松英，“ 車両情報記録装置の大規模データの可視化手法の検討，” 電気学会交通・電気鉄道研究会 (TER)，2016 年 1 月 29 日.

付録 A　行パターンマッチングの処理コスト削減のための最適化手法における実験結果の詳細

A.1　人工データを用いた実験結果

本文において記載ができなかった，実験結果の具体的な値を示す．まず，3.9.1 節に示した行パターンマッチングの処理コスト削減のための最適化手法における人工データを用いた実験結果，図 3.24 から図 3.35 の具体的な値を，それぞれ表 A.1 から表 A.12 に示す．

表 A.1 PostgreSQL を用いた人工データに対する実験結果 (Q1)（図 3.24）

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6	Conf. 7
No. Opt.	10.35	11.07	11.79	12.79	12.84	20.06	25.47
Seq. Filt.	0.00	3.95	3.00	5.76	16.20	19.89	27.94
Row Filt.	7.69	8.71	9.98	11.76	11.75	23.82	32.86
Seq. Filt. + Row Filt.	0.01	3.80	3.65	6.84	15.67	24.04	35.25

表 A.2 PostgreSQL を用いた人工データに対する実験結果 (Q2)（図 3.25）

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6	Conf. 7
No Opt.	10.51	11.07	11.96	13.08	13.05	20.98	27.80
Seq. Filt.	0.00	4.03	3.19	6.09	16.56	21.86	30.66
Row Filt.	11.54	12.66	13.98	15.84	15.79	28.48	37.99
Seq. Filt. + Row Filt.	0.01	4.59	4.19	7.91	19.05	28.26	40.91

表 A.3 PostgreSQL を用いた人工データに対する実験結果 (Q3)（図 3.26）

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6	Conf. 7
No Opt.	10.61	11.29	12.23	13.55	13.55	22.66	29.42
Seq. Filt.	0.00	4.08	3.40	6.46	17.05	22.65	32.38
Row Filt.	6.92	8.13	9.56	11.55	11.55	25.48	35.76
Seq. Filt. + Row Filt.	0.01	3.75	3.92	7.34	15.73	26.18	38.78

表 A.4 PostgreSQL を用いた人工データに対する実験結果 (Q4)（図 3.27）

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6	Conf. 7
No Opt.	10.60	11.73	13.18	15.19	15.24	29.57	39.14
Seq. Filt.	0.00	4.53	4.35	8.18	18.73	30.17	43.05
Row Filt.	8.58	10.15	11.94	14.65	14.65	32.83	47.35
Seq. Filt. + Row Filt.	0.01	4.47	4.97	9.21	18.59	34.24	49.60

表 A.5 PostgreSQL を用いた人工データに対する実験結果 (Q5)（図 3.28）

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6	Conf. 7
No Opt.	27.00	26.88	27.38	28.17	28.17	33.72	37.80
Seq. Filt.	0.00	6.97	4.27	8.36	28.43	34.23	40.31
Row Filt.	13.82	15.39	17.27	19.80	19.86	37.67	51.11
Seq. Filt. + Row Filt.	0.01	5.44	5.34	10.07	22.68	41.48	53.64

表 A.6 PostgreSQL を用いた人工データに対する実験結果 (Q6)（図 3.29）

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6	Conf. 7
No Opt.	10.38	11.38	12.66	14.42	14.46	26.65	36.41
Seq. Filt.	0.00	4.32	3.97	7.51	18.19	27.45	39.89
Row Filt.	16.06	17.38	19.11	21.45	21.40	37.54	50.29
Seq. Filt. + Row Filt.	0.01	5.77	5.30	9.98	23.78	37.20	52.55

表 A.7 Spark を用いた人工データに対する実験結果 (Q1)（図 3.30）

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6	Conf. 7
No Opt.	301.94	280.83	294.11	314.50	300.25	337.75	398.98
Seq. Filt.	61.02	142.66	117.94	149.64	281.34	323.63	436.75
Row Filt.	67.04	101.81	118.65	138.82	139.95	273.39	399.11
Seq. Filt. + Row Filt.	63.19	105.06	117.81	141.40	148.07	289.76	420.41

表 A.8 Spark を用いた人工データに対する実験結果 (Q2)（図 3.31）

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6	Conf. 7
No Opt.	296.59	304.14	301.06	301.53	299.09	319.80	342.88
Seq. Filt.	61.27	137.38	115.04	145.65	280.70	311.73	365.81
Row Filt.	68.33	107.24	117.63	133.82	137.29	262.76	351.59
Seq. Filt. + Row Filt.	63.62	107.67	116.68	142.16	148.83	271.63	380.90

表 A.9 Spark を用いた人工データに対する実験結果 (Q3)（図 3.32）

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6	Conf. 7
No Opt.	449.50	456.53	437.75	452.28	461.87	471.01	512.39
Seq. Filt.	62.07	170.73	129.37	170.37	401.70	420.26	515.79
Row Filt.	66.88	111.89	125.45	156.57	160.22	352.04	523.84
Seq. Filt. + Row Filt.	63.65	111.71	128.14	159.64	163.01	356.26	533.91

表 A.10 Spark を用いた人工データに対する実験結果 (Q4)（図 3.33）

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6	Conf. 7
No Opt.	452.63	439.03	444.38	460.91	470.46	513.56	-
Seq. Filt.	60.77	171.24	138.11	185.15	405.91	458.65	-
Row Filt.	66.90	109.99	135.11	168.78	168.71	392.00	-
Seq. Filt. + Row Filt.	62.91	110.50	138.51	173.40	174.01	410.84	-

表 A.11 Spark を用いた人工データに対する実験結果 (Q5)（図 3.34）

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6	Conf. 7
No Opt.	365.27	337.46	350.54	360.10	343.15	352.46	356.04
Seq. Filt.	61.93	152.24	117.34	150.24	309.70	323.54	381.53
Row Filt.	68.89	104.71	119.44	138.68	139.95	268.01	371.78
Seq. Filt. + Row Filt.	63.14	107.71	115.04	138.86	145.64	290.21	386.06

表 A.12 Spark を用いた人工データに対する実験結果 (Q6)（図 3.35）

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6	Conf. 7
No Opt.	299.65	309.67	301.65	335.96	335.75	427.74	463.28
Seq. Filt.	61.13	144.26	127.99	167.30	305.00	393.72	483.67
Row Filt.	68.42	111.51	126.47	161.11	163.51	366.96	477.49
Seq. Filt. + Row Filt.	62.74	110.98	127.74	161.91	169.01	372.26	482.18

A.2 実データを用いた実験結果

次に，3.9.2 節において示した，Foursquare Dataset に対する Q7 から Q12 の実験結果，図 3.42 と図 3.43 の詳細を，それぞれ表 A.13 と表 A.14 に示す.

表 A.13 PostgreSQL を用いた Foursquare Dataset に対する実験結果

	Q7	Q8	Q9	Q10	Q11	Q12
No Opt.	90.13	93.40	92.65	104.42	95.32	90.47
Seq. Filt.	94.14	97.05	81.73	21.82	10.46	53.50
Row Filt.	75.57	108.45	98.98	62.04	79.21	103.61
Seq. Filt. + Row Filt.	83.37	112.83	90.29	15.95	10.44	61.32

表 A.14 Spark を用いた Foursquare Dataset に対する実験結果

	Q7	Q8	Q9	Q10	Q11	Q12
No Opt.	964.11	944.23	1434.58	973.15	1925.78	975.56
Seq. Filt.	903.87	896.45	1039.93	404.17	328.81	659.63
Row Filt.	479.57	612.07	399.11	267.31	277.46	343.85
Seq. Filt. + Row Filt.	504.19	638.88	441.15	281.70	288.98	374.95

A.3 コストモデルの検証結果

3.9.3 節において示した，N=10,000,000，S=1,000 および N=10,000,000，S=10,000 および N=5,000,000，S=1,000 の時の Q1 および Q5 の見積値と実測値の結果である，図 3.44 から図 3.55 の結果の具体的な値を，表 A.15 から表 A.38 にそれぞれ示す．なお，図 3.44 から図 3.55 の結果は，見積値と実測値を合わせて示しているが，ここではスペースの都合上，見積値と実測値を分けて示す．

表 A.15 PostgreSQL における見積値（Q1，N = 10,000,000，S = 1,000，図 3.44）

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6	Conf. 7
No. Opt.	10.35	11.07	11.79	12.79	12.84	20.06	25.47
Seq. Filt.	2.21	4.51	3.58	5.11	12.83	19.66	29.88
Row Filt.	12.16	12.61	13.22	14.21	14.22	25.00	37.64
Seq. Filt. + Row Filt.	2.21	5.17	4.68	7.04	14.34	26.18	42.05

表 A.16 PostgreSQL における実測値（Q1，N = 10,000,000，S = 1,000，図 3.44）

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6	Conf. 7
No. Opt.	10.35	11.07	11.79	12.79	12.84	20.06	25.47
Seq. Filt.	0.00	3.95	3.00	5.76	16.20	19.89	27.94
Row Filt.	7.69	8.71	9.98	11.76	11.75	23.82	32.86
Seq. Filt. + Row Filt.	0.01	3.80	3.65	6.84	15.67	24.04	35.25

表 A.17 PostgreSQL における見積値（Q5，N = 10,000,000，S = 1,000，図 3.45）

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6	Conf. 7
No. Opt.	27.00	26.88	27.38	28.17	28.17	33.72	37.80
Seq. Filt.	2.21	7.67	5.14	8.19	25.09	30.59	42.21
Row Filt.	18.23	19.30	20.69	22.74	22.74	39.81	56.02
Seq. Filt. + Row Filt.	2.21	7.01	6.69	10.71	21.65	39.78	60.43

表 A.18 PostgreSQL における実測値（Q5，N = 10,000,000，S = 1,000，図 3.45）

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6	Conf. 7
No. Opt.	27.00	26.88	27.38	28.17	28.17	33.72	37.80
Seq. Filt.	0.00	6.97	4.27	8.36	28.43	34.23	40.31
Row Filt.	13.82	15.39	17.27	19.80	19.86	37.67	51.11
Seq. Filt. + Row Filt.	0.01	5.44	5.34	10.07	22.68	41.48	53.64

表 A.19 Spark における見積値（Q1，N = 10,000,000，S = 1,000，図 3.46）

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6	Conf. 7
No. Opt.	270.50	249.43	262.66	282.88	268.72	306.18	367.49
Seq. Filt.	14.54	65.01	42.11	73.44	231.84	268.79	396.57
Row Filt.	30.04	40.02	53.68	75.30	73.04	226.00	397.53
Seq. Filt. + Row Filt.	14.54	31.11	42.49	68.13	83.89	243.83	426.61

表 A.20 Spark における実測値（Q1，N = 10,000,000，S = 1,000，図 3.46）

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6	Conf. 7
No. Opt.	270.50	249.43	262.66	282.88	268.72	306.18	367.49
Seq. Filt.	29.43	110.97	86.35	118.30	249.84	292.10	405.21
Row Filt.	35.64	70.60	87.20	107.33	108.47	241.89	367.89
Seq. Filt. + Row Filt.	31.71	73.61	86.32	109.82	116.47	258.22	388.98

表 A.21 Spark における見積値（Q5，N = 10,000,000，S = 1,000，図 3.47）

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6	Conf. 7
No. Opt.	333.86	306.01	319.15	328.47	311.64	320.93	324.61
Seq. Filt.	14.54	76.32	47.76	82.56	266.18	280.59	353.69
Row Filt.	30.80	43.04	59.53	83.36	80.66	236.20	355.41
Seq. Filt. + Row Filt.	14.54	33.52	47.65	75.58	91.37	253.88	384.49

表 A.22 Spark における実測値（Q5，N = 10,000,000，S = 1,000，図 3.47）

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6	Conf. 7
No. Opt.	333.86	306.01	319.15	328.47	311.64	320.93	324.61
Seq. Filt.	30.30	120.79	85.93	118.60	278.29	292.07	350.22
Row Filt.	37.60	73.22	88.05	107.19	108.38	236.35	340.29
Seq. Filt. + Row Filt.	31.70	76.16	83.58	107.49	114.32	258.85	354.92

表 A.23 PostgreSQL における見積値（Q1，N = 10,000,000，S = 10,000，図 3.48）

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6	Conf. 7
No. Opt.	10.35	11.07	11.79	12.79	12.84	20.06	25.47
Seq. Filt.	2.21	4.51	3.58	5.12	12.84	19.68	29.90
Row Filt.	12.16	12.61	13.22	14.21	14.22	25.00	37.64
Seq. Filt. + Row Filt.	2.21	5.17	4.68	7.04	14.36	26.20	42.07

表 A.24 PostgreSQL における実測値（Q1，N = 10,000,000，S = 10,000，図 3.48）

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6	Conf. 7
No Opt.	10.52	10.93	11.71	12.80	12.83	20.12	25.57
Seq. Filt.	0.00	3.97	3.01	5.86	16.60	19.96	27.95
Row Filt.	7.72	8.72	10.03	11.77	11.78	23.80	32.93
Seq. Filt. + Row Filt.	0.01	3.80	3.67	6.96	16.03	24.31	35.31

表 A.25 PostgreSQL における見積値（Q1，N = 5,000,000，S = 1,000，図 3.49）

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6	Conf. 7
No Opt.	5.18	5.53	5.89	6.39	6.42	10.03	12.74
Seq. Filt.	1.10	2.25	1.79	2.56	6.41	9.83	14.94
Row Filt.	6.08	6.30	6.61	7.10	7.11	12.50	18.82
Seq. Filt. + Row Filt.	1.10	2.58	2.34	3.52	7.17	13.09	21.02

表 A.26 PostgreSQL における実測値（Q1，N = 5,000,000，S = 1,000，図 3.49）

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6	Conf. 7
No Opt.	5.27	5.57	5.96	6.46	6.48	10.17	12.88
Seq. Filt.	0.00	1.90	1.50	2.84	7.97	10.01	13.99
Row Filt.	3.95	4.39	5.02	5.89	5.91	12.03	16.49
Seq. Filt. + Row Filt.	0.01	1.84	1.83	3.38	7.64	12.14	17.56

表 A.27 PostgreSQL における見積値（Q5，N = 10,000,000，S = 10,000，図 3.50）

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6	Conf. 7
No Opt.	27.00	26.88	27.38	28.17	28.17	33.72	37.80
Seq. Filt.	2.21	7.67	5.14	8.20	25.11	30.61	42.23
Row Filt.	18.23	19.30	20.69	22.74	22.74	39.81	56.02
Seq. Filt. + Row Filt.	2.21	7.02	6.69	10.72	21.67	39.79	60.46

表 A.28 PostgreSQL における実測値（Q5，N = 10,000,000，S = 10,000，図 3.50）

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6	Conf. 7
No Opt.	26.48	27.04	27.56	28.34	28.31	34.02	38.42
Seq. Filt.	0.00	7.00	4.28	8.47	28.85	34.63	40.46
Row Filt.	13.89	15.40	17.23	19.86	19.85	37.79	51.12
Seq. Filt. + Row Filt.	0.01	5.53	5.38	10.19	23.12	41.23	54.03

表 A.29 PostgreSQL における見積値（Q5，N = 5,000,000，S = 1,000，図 3.51）

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6	Conf. 7
No Opt.	13.5	13.44	13.69	14.09	14.09	16.86	18.90
Seq. Filt.	1.10	3.83	2.57	4.10	12.55	15.29	21.10
Row Filt.	9.11	9.65	10.35	11.37	11.37	19.9	28.01
Seq. Filt. + Row Filt.	1.10	3.51	3.35	5.36	10.82	19.89	30.22

表 A.30 PostgreSQL における実測値（Q5，N = 5,000,000，S = 1,000，図 3.51）

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6	Conf. 7
No Opt.	13.13	13.51	13.77	14.18	14.16	16.93	18.91
Seq. Filt.	0.00	3.43	2.14	4.16	14.09	17.02	20.10
Row Filt.	7.00	7.74	8.68	10.01	9.91	18.93	25.77
Seq. Filt. + Row Filt.	0.01	2.73	2.68	4.98	11.15	20.31	27.65

表 A.31 Spark における見積値（Q1，N = 10,000,000，S = 10,000，図 3.52）

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6	Conf. 7
No Opt.	270.50	249.43	262.66	282.88	268.72	306.18	367.49
Seq. Filt.	14.54	65.01	42.11	73.44	231.85	268.80	396.58
Row Filt.	30.04	40.02	53.68	75.30	73.04	226.00	397.53
Seq. Filt. + Row Filt.	14.54	31.11	42.49	68.14	83.90	243.85	426.63

表 A.32 Spark における実測値（Q1，N = 10,000,000，S = 10,000，図 3.52）

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6	Conf. 7
No Opt.	267.87	266.85	254.89	285.41	267.76	321.22	368.71
Seq. Filt.	29.71	111.79	88.16	116.22	241.85	304.72	385.05
Row Filt.	35.76	74.33	87.08	107.36	107.96	252.40	369.21
Seq. Filt. + Row Filt.	31.32	73.86	90.02	110.41	115.41	262.78	414.05

表 A.33 Spark における見積値（Q1，N = 5,000,000，S = 1,000，図 3.53）

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6	Conf. 7
No Opt.	135.25	124.72	131.33	141.44	134.36	153.09	183.75
Seq. Filt.	7.27	32.50	21.06	36.72	115.92	134.40	198.29
Row Filt.	15.02	20.01	26.84	37.65	36.52	113.00	198.77
Seq. Filt. + Row Filt.	7.27	15.55	21.25	34.07	41.95	121.92	213.31

表 A.34 Spark における実測値（Q1，N = 5,000,000，S = 1,000，図 3.53）

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6	Conf. 7
No Opt.	144.19	154.67	151.85	156.80	156.58	171.24	191.08
Seq. Filt.	24.33	81.39	63.28	79.79	144.71	166.17	202.93
Row Filt.	29.65	58.59	63.07	79.98	77.13	142.46	202.13
Seq. Filt. + Row Filt.	25.31	57.52	63.85	82.27	78.16	149.96	207.21

表 A.35 Spark における見積値（Q5，N = 10,000,000，S = 10,000，図 3.54）

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6	Conf. 7
No Opt.	333.86	306.01	319.15	328.47	311.64	320.93	324.61
Seq. Filt.	14.54	76.32	47.76	82.56	266.19	280.60	353.70
Row Filt.	30.80	43.04	59.53	83.36	80.66	236.20	355.41
Seq. Filt. + Row Filt.	14.54	33.52	47.65	75.58	91.38	253.89	384.50

表 A.36 Spark における実測値（Q5，N = 10,000,000，S = 10,000，図 3.54）

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6	Conf. 7
No Opt.	312.46	307.07	326.07	313.43	328.64	343.36	333.99
Seq. Filt.	31.51	117.20	88.59	125.66	281.71	301.61	362.03
Row Filt.	38.53	73.61	90.34	107.53	113.57	252.11	351.25
Seq. Filt. + Row Filt.	32.83	79.83	91.89	117.20	115.37	266.81	366.50

表 A.37 Spark における見積値（Q5，N = 5,000,000，S = 1,000，図 3.55）

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6	Conf. 7
No Opt.	166.93	153.01	159.57	164.24	155.82	160.47	162.30
Seq. Filt.	7.27	38.16	23.88	41.28	133.09	140.30	176.84
Row Filt.	15.40	21.52	29.76	41.68	40.33	118.10	177.71
Seq. Filt. + Row Filt.	7.27	16.76	23.83	37.79	45.69	126.94	192.25

表 A.38 Spark における実測値（Q5，N = 5,000,000，S = 1,000，図 3.55）

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6	Conf. 7
No Opt.	177.26	175.36	176.90	176.27	174.74	188.71	186.91
Seq. Filt.	24.92	82.09	64.02	82	157.97	173.03	196.66
Row Filt.	29.73	57.94	66.08	76.09	76.11	145.91	196.28
Seq. Filt. + Row Filt.	25.85	57.52	64.09	76.29	79.37	154.99	199.57

付録 B　ニューラルネットワークを用いた列車遅延予測手法における実験結果の詳細

B.1　予測モデルの単位に関する精度評価結果

4.4.4 節において示した図 4.11，図 4.11，図 4.11 の具体的な値を，それぞれ表 B.1，表 B.2，表 B.3 に示す．

表 B.1 予測モデルの構築単位間の予測精度（図 4.11）

		列車種別毎	列車毎	列車駅毎
2 駅	全列車	10.74	9.94	10.02
	快速	9.64	9.15	8.78
	区間快速	8.41	8.04	7.57
	各停 1	14.95	13.28	14.52
駅 Q	全列車	9.71	8.70	9.30
	快速	8.88	8.46	9.22
	区間快速	7.24	6.87	6.77
	各停 1	13.85	11.38	12.77
駅 S	全列車	11.77	11.19	10.74
	快速	10.41	9.85	8.35
	区間快速	9.58	9.22	8.38
	各停 1	16.06	15.17	16.27

表 B.2 予測モデルの構築単位間の 4 駅先までの各駅の予測精度（図 4.12）

		列車種別毎	列車毎	列車駅毎
2 駅	自駅	6.19	5.92	6.80
	1 駅先	7.89	6.94	7.18
	2 駅先	9.55	8.66	9.21
	3 駅先	11.50	9.70	9.98
	4 駅先	11.64	11.64	11.20
駅 Q	自駅	4.67	4.90	6.15
	1 駅先	7.42	6.03	5.98
	2 駅先	9.61	8.26	8.88
	3 駅先	10.34	8.41	9.76
	4 駅先	10.48	10.04	10.43
駅 S	自駅	7.70	6.94	7.45
	1 駅先	8.35	7.85	8.37
	2 駅先	9.50	9.05	9.53
	3 駅先	12.67	10.98	10.19
	4 駅先	12.80	13.23	11.97

表 B.3 予測モデルの構築単位間の相関係数（図 4.13）

		列車種別	列車	列車駅
2 駅	全列車	0.40	0.45	0.66
	快速	0.38	0.46	0.63
	区間快速	0.22	0.31	0.60
	各停 1	0.65	0.63	0.78
駅 Q	全列車	0.45	0.46	0.72
	快速	0.48	0.49	0.71
	区間快速	0.19	0.22	0.65
	各停 1	0.79	0.77	0.83
駅 S	全列車	0.34	0.44	0.61
	快速	0.28	0.43	0.54
	区間快速	0.25	0.40	0.55
	各停 1	0.52	0.49	0.74

B.2 予測モデルの入力データに関する精度評価結果

4.4.5 節において示した図 4.14 から図 4.25 の具体的な値を，それぞれ表 B.4 から表 B.15 に示す.

表 B.4 予測モデルの入力データの種類間の予測精度（図 4.14）

	case i	case ii	case iii	case iv	case v	case vi
全列車	10.64	10.07	9.84	10.37	9.93	9.74
快速	8.98	8.53	8.69	8.75	8.45	8.50
区間快速	9.17	8.71	9.22	8.96	8.82	9.19
各停 1	14.61	14.22	13.91	13.95	13.48	13.62
各停 2	10.34	9.37	8.59	10.3	9.39	8.58

表 B.5 予測モデルの入力データの種類間の予測精度（駅 Q 着時点）（図 4.15）

	case i	case ii	case iii	case iv	case v	case vi
全列車	10.86	10.71	10.08	10.72	10.56	9.86
快速	8.86	9.32	9.78	9.12	9.11	9.12
区間快速	8.87	8.2	9.45	8.4	8.11	9.49
各停 1	15.53	15.47	14.574	15.4	15.26	14.59
各停 2			7.97			7.74

表 B.6 予測モデルの入力データの種類間の予測精度（駅 S 着時点）（図 4.16）

	case i	case ii	case iii	case iv	case v	case vi
全列車	10.53	10.3	9.73	10.41	10.31	9.77
快速	8.67	8.63	8.49	8.69	8.5	8.61
区間快速	8.4	8.35	8.52	8.46	8.47	8.32
各停 1	16.81	15.78	14.38	16.02	15.82	14.34
各停 2	9.58	9.55	8.65	9.62	9.56	8.87

表 B.7 予測モデルの入力データの種類間の予測精度（駅 V 着時点）（図 4.17）

	case i	case ii	case iii	case iv	case v	case vi
全列車	10.61	9.42	9.71	10.10	9.13	9.58
快速	9.41	7.64	7.80	8.43	7.75	7.76
区間快速	10.24	9.57	9.68	10.02	9.88	9.75
各停 1	11.49	11.42	12.62	10.44	9.35	11.92
各停 2	11.09	9.18	9.16	10.97	9.21	9.12

表 B.8 予測モデルの入力データの種類間に関する駅 X までの各駅の予測精度（図 4.18）

	case i	case ii	case iii	case iv	case v	case vi
自駅	7.60	6.48	6.00	7.12	6.37	6.32
1 駅先	8.06	7.33	6.69	7.71	7.22	6.78
2 駅先	8.56	8.36	7.72	8.35	8.15	7.78
3 駅先	10.06	9.94	8.80	10.09	9.82	8.94
4 駅先	10.65	10.53	9.56	10.55	10.51	9.44
5 駅先	10.02	10.00	9.97	10.05	10.01	9.98
6 駅先	12.00	12.25	11.44	12.13	12.15	10.86
7 駅先	9.14	9.28	9.26	9.23	9.33	8.86

表 B.9 予測モデルの入力データの種類間に関する駅 X までの各駅の予測精度（駅 Q 着時点）（図 4.19）

	case i	case ii	case iii	case iv	case v	case vi
自駅	5.97	5.13	4.53	5.59	5.44	5.09
1 駅先	6.23	5.94	4.75	5.8	5.68	5.05
2 駅先	8.76	8.52	7.28	8.47	8.1	7.35
3 駅先	9.29	9.25	8.01	9.38	9.82	8.01
4 駅先	10.22	10.21	8.79	10.17	10.51	8.68
5 駅先	11.16	10.79	11.14	11.1	11.04	11.03
6 駅先	12	12.25	11.44	12.13	12.15	10.86
7 駅先	9.14	9.28	9.26	9.23	9.33	8.86

表 B.10 予測モデルの入力データの種類間に関する駅 X までの各駅の予測精度（駅 S 着時点）（図 4.20）

	case i	case ii	case iii	case iv	case v	case vi
自駅	7.03	6.58	5.73	6.93	6.59	5.92
1 駅先	7.55	6.82	6.26	7.24	7.09	6.64
2 駅先	8.30	8.15	7.47	8.24	8.16	7.64
3 駅先	10.57	10.40	9.58	10.57	10.35	9.87
4 駅先	10.93	10.75	10.33	10.80	10.88	10.19
5 駅先	9.21	9.44	8.72	9.31	9.27	8.85

表 B.11 予測モデルの入力データの種類間に関する駅 X までの各駅の予測精度（駅 V 着時点）（図 4.21）

	case i	case ii	case iii	case iv	case v	case vi
自駅	9.26	7.27	7.75	8.34	6.76	7.97
1 駅先	9.79	8.77	9.05	9.47	8.38	8.66
2 駅先	8.69	8.47	8.46	8.38	8.17	8.39

表 B.12 予測モデルの入力データの種類間の相関係数（図 4.22）

	case i	case ii	case iii	case iv	case v	case vi
全列車	0.44	0.45	0.44	0.45	0.44	0.46
快速	0.39	0.40	0.35	0.42	0.39	0.37
区間快速	0.42	0.42	0.43	0.43	0.43	0.43
各停 1	0.67	0.68	0.66	0.67	0.68	0.68
各停 2	0.30	0.31	0.35	0.30	0.29	0.36

表 B.13 予測モデルの入力データの種類間の相関係数（駅 Q 着時点）（図 4.23）

	case i	case ii	case iii	case iv	case v	case vi
全列車	0.62	0.62	0.56	0.62	0.62	0.58
快速	0.64	0.62	0.57	0.61	0.62	0.61
区間快速	0.52	0.53	0.52	0.54	0.53	0.54
各停 1	0.72	0.72	0.72	0.73	0.73	0.72
各停 2	-	-	0.49	-	-	0.51

表 B.14 予測モデルの入力データの種類間の相関係数（駅 S 着時点）（図 4.24）

	case i	case ii	case iii	case iv	case v	case vi
全列車	0.51	0.50	0.48	0.51	0.51	0.50
快速	0.52	0.47	0.44	0.51	0.50	0.49
区間快速	0.49	0.50	0.49	0.50	0.50	0.52
各停 1	0.69	0.70	0.70	0.70	0.70	0.71
各停 2	0.40	0.39	0.37	0.40	0.41	0.37

表 B.15 予測モデルの入力データの種類間の相関係数（駅 V 着時点）（図 4.25）

	case i	case ii	case iii	case iv	case v	case vi
全列車	0.26	0.28	0.26	0.28	0.26	0.25
快速	0.02	0.11	0.05	0.14	0.06	0.01
区間快速	0.24	0.22	0.27	0.25	0.26	0.25
各停 1	0.61	0.62	0.57	0.58	0.60	0.59
各停 2	0.20	0.23	0.20	0.21	0.17	0.19

B.3 他の予測手法との精度評価結果

4.4.6 節において示した図 4.27 から図 4.30 の具体的な値を，それぞれ表 B.16 から表 B.19 に示す.

表 B.16 重回帰分析と提案手法の予測精度（図 4.27）

		重回帰分析	提案手法
3 駅	全列車	11.69	9.74
	快速	15.27	8.50
	区間快速	10.55	9.19
	各停 1	13.07	13.62
	各停 2	9.60	8.58
駅 Q	全列車	13.85	9.86
	快速	26.94	9.12
	区間快速	10.33	9.49
	各停 1	14.00	14.59
	各停 2	8.66	7.74
駅 S	全列車	10.86	9.77
	快速	9.53	8.61
	区間快速	10.35	8.32
	各停 1	14.53	14.34
	各停 2	9.85	8.87
駅 V	全列車	10.36	9.58
	快速	9.35	7.76
	区間快速	10.98	9.75
	各停 1	10.66	11.92
	各停 2	10.28	9.12

表 B.17 3 手法間における 4 駅先までの各駅の予測精度（図 4.28）

	重回帰分析	提案手法	Chapuis の手法
自駅	6.26	6.32	6.54
1 駅先	5.21	6.78	-
2 駅先	8.19	7.78	-
3 駅先	7.76	8.94	-
4 駅先	8.92	9.44	-
5 駅先	11.75	9.98	-
6 駅先	11.64	10.86	-
7 駅先	22.25	8.86	-

表 B.18 3 手法間における 4 駅先までの各駅の予測精度（着駅別）（図 4.29）

		重回帰分析	提案手法	Chapuis の手法
駅 Q	自駅	4.66	5.09	6.51
	1 駅先	5.05	5.05	-
	2 駅先	8.1	7.35	-
	3 駅先	8.02	8.01	-
	4 駅先	8.92	8.68	-
	5 駅先	11.94	11.03	-
	6 駅先	11.64	10.86	-
	7 駅先	22.25	8.86	-
駅 S	自駅	6.42	5.92	6.65
	1 駅先	6.94	6.64	-
	2 駅先	8.1	7.64	-
	3 駅先	10.67	9.87	-
	4 駅先	10.87	10.19	-
	5 駅先	10.9	8.85	-
駅 V	自駅	7.7	7.97	6.45
	1 駅先	9.21	8.66	-
	2 駅先	9.93	8.39	-

表 B.19 重回帰分析と提案手法の相関係数（図 4.30）

		重回帰分析	提案手法
3 駅	全列車	0.35	0.46
	快速	0.21	0.37
	区間快速	0.32	0.43
	各停 1	0.67	0.68
	各停 2	0.28	0.36
駅 Q	全列車	0.44	0.58
	快速	0.23	0.61
	区間快速	0.4	0.54
	各停 1	0.68	0.72
	各停 2	0.44	0.51
駅 S	全列車	0.37	0.5
	快速	0.3	0.49
	区間快速	0.31	0.52
	各停 1	0.64	0.71
	各停 2	0.29	0.37
駅 V	全列車	0.25	0.25
	快速	0.09	0.01
	区間快速	0.24	0.25
	各停 1	0.69	0.59
	各停 2	0.09	0.19

研究業績

博士論文に関する論文

査読付き論文誌

- 中挾晃介，辰井大祐，國松武俊，「ニューラルネットワークを用いた列車遅延・乗車率予測手法」，『情報処理学会論文誌』，vol.60，no.4，pp.1129–1140，2019 年 4 月．

査読付き国際会議

- Kosuke Nakabasami, Hiroyuki Kitagawa, and Yuya Nasu, “Optimization of Row Pattern Matching over Sequence Data in Spark SQL,” Proc. of the 30th International Conference on Database and Expert Systems Applications (DEXA2019), pp.3–17, 2019.

研究会発表

- 中挾晃介，辰井大祐，國松武俊，「ニューラルネットワークを用いた列車運行予測手法」，『第 23 回鉄道技術連合シンポジウム (J-RAIL2016)』，S5-2-3，2016 年 12 月 14 日–16 日．
- 中挾晃介，那須勇弥，北川博之，「Spark SQL を用いたシーケンスデータに対する行パターンマッチングの最適化」，『第 11 回データ工学と情報マネジメントに関するフォーラム (DEIM2019)』，H4-1，2019 年 3 月 4 日–6 日．

その他論文

査読付き論文誌

- 中挾晃介，北川博之，Salman Ahmed Shaikh，天笠俊之，「StreamOLAP における問合せの最適化手法」，『日本データベース学会和文論文誌』，vol.14，no.4，article no.3，2016 年 3 月．

査読付き国際会議

- Kosuke Nakabasami, Toshiyuki Amagasa, and Hiroyuki Kitagawa, “Quering MongoDB with LINQ in a Server-side Javascript Environment,” Proc. of the 2nd International Workshop on Advances in Data Engineering and Mobile Computing (DEMoC2013), pp.344–349, 2013.
- Kosuke Nakabasami, Toshiyuki Amagasa, Salman Ahmed Shaikh, Franck Gass, and Hiroyuki Kitagawa, “Quering MongoDB with LINQ in a Server-side Javascript Environment,” Proc. of the 2nd International Workshop on Advances in Data Engineering and Mobile Computing (DEMoC2013), pp.344–349, 2013.
- Yuya Nasu, Hiroyuki Kitagawa, and Kosuke Nakabasami, “Efficient Pattern Matching using Pattern Hierarchies for Sequence OLAP,” Proc. of the 21st International Conference on Big Data Analytics and Knowledge Discovery (DaWaK2019), pp.89–104, 2019.

研究会発表

- 中挟晃介，天笠俊之，北川博之，「JavaScript 処理系における LINQ を用いた NoSQL ストレージに対する問合せ処理」，『情報処理学会第 75 回全国大会（IPSJ 全国大会 2013）』，2N-5，2013 年 3 月 6 日–8 日.
- 中挟晃介，北川博之，天笠俊之，「ストリーム処理エンジンを用いたストリームデータの OLAP 処理」，『第 6 回データ工学と情報マネジメントに関するフォーラム (DEIM2014)』，D3-1，2014 年 3 月 3 日–5 日.
- 中挟晃介，北川博之，天笠俊之，「ストリームデータに対するストリーム処理エンジンを用いた OLAP 処理」，『第 13 回情報科学技術フォーラム (FIT 2014)』，D-028，2014 年 9 月 3 日–5 日.
- 中挟晃介，北川博之，Salman Ahmed Shaikh，天笠俊之，「ストリーム OLAP における問合せの最適化手法」，『第 7 回データ工学と情報マネジメントに関するフォーラム (DEIM2015)』，D6-5，2015 年 3 月 2 日–4 日.
- 坂本沙季，中挟晃介，北川博之，天笠俊之，「ストリーム OLAP のための可視化ユーザインターフェース」，『情報処理学会第 77 回全国大会（IPSJ 全国大会 2015）』，

4N-07，2015 年 3 月 17 日–19 日.

- 中挾晃介，武内陽子，小川知行，影山真佐富，東海勝人，平松英，「車両情報記録装置の大規模データの可視化手法の検討」,『電気学会交通・電気鉄道研究会 (TER)』, 2016 年 1 月 29 日.
- 辰井大祐，中挾晃介，國松武俊,「ニューラルネットワークによる列車運行予測の直通路線への適用」,『平成 29 年電気学会電子・情報・システム部門大会』，2017 年 9 月 6 日–9 日.
- 中挾晃介，國松武俊，辰井大祐,「自動改札機データを用いた旅客の列車乗継経路推定手法」,『第 24 回鉄道技術・政策連合シンポジウム (J-RAIL2017)』，S5-2-1, 2017 年 12 月 12 日–14 日.
- 那須勇弥，中挾晃介，北川博之,「シーケンス OLAP におけるパターン拡張問合せ手法の提案」,『第 10 回データ工学と情報マネジメントに関するフォーラム (DEIM2018)』，E1-5，2018 年 3 月 4 日–6 日.
- 那須勇弥，中挾晃介，北川博之,「シーケンス OLAP のための複数行パターンマッチング問合せ処理」,『第 11 回データ工学と情報マネジメントに関するフォーラム (DEIM2019)』，H4-3，2019 年 3 月 4 日–6 日.
- 中挾晃介，國松武俊，辰井大祐，瀧本友晴,「自動改札機データを用いた列車の乗車人数推定手法の比較」,『平成 31 年電気学会全国大会』，G307-A3，5-272，2019 年 3 月 12 日–14 日.
- 瀧本友晴，中挾晃介，國松武俊，辰井大祐,「旅客入出場時刻データを用いた鉄道往復利用の分析」,『平成 31 年電気学会全国大会』，G307-A3，5-271，2019 年 3 月 12 日–14 日.