2022（令和 4）年度　修士論文

プローブ情報を活用した遅れ時間の AI 短期予測に基づく信号制御手法の構築

Development of Signal Control Method Based on

AI Short-Term Prediction of Delay Using Probe Information

東京都立大学大学院 都市環境科学研究科
都市基盤環境学域 社会基盤分野 交通研究室
21851520 髙橋翼

指導教員 小根山裕之 教授
柳原正実 助教

目次

第１章　序論.....4
1.1　研究背景.....5
1.1.1　信号制御の最適化.....5
1.1.2　現行の信号制御手法.....5
1.1.3　交通応答制御の問題点.....6
1.1.4　プローブ情報.....6
1.2　研究目的.....7
1.3　論文構成.....8
第２章　既往研究.....9
2.1　既往研究レビュー.....10
2.2　本研究の位置づけ.....11
第３章　提案信号制御手法の構築.....12
3.1　概要.....13
3.2　AI による予測.....14
3.2.1　AI とは.....14
3.2.2　本手法で AI を用いる理由.....15
3.2.3　本手法で用いる AI の構造.....15
3.2.4　AI の入力データ.....15
3.2.5　AI の出力データ.....17
3.2.6　AI の学習.....18
3.3　探索.....18
3.3.1　概要.....18
3.3.2　最適化する変数.....19
3.3.3　目的関数.....19
3.3.4　探索アルゴリズム.....19
3.3.5　探索の初期値.....20
3.3.6　探索時間.....20
第４章　シミュレーション評価.....23
4.1　概要.....24
4.2　諸条件.....24
4.2.1　ネットワーク構成.....24
4.2.2　交通条件.....25
4.2.3　シミュレーション実行条件.....27
4.2.4　提案信号制御手法の汎用性.....27
4.3　シミュレーションの流れ.....27
4.3.1　概要.....27

4.3.2　プローブエミュレータまでの流れ..... 28
4.3.3　提案信号制御手法以降の流れ..... 31
4.4　学習結果..... 35
4.5　制御効果の評価指標..... 37
4.6　従来信号制御手法による信号制御パラメータ算出..... 37
4.7　提案信号制御の制御効果..... 40
4.7.1　異なる探索アルゴリズムによる制御効果の違い..... 40
4.7.2　異なる初期値による制御効果の違い..... 44
4.7.3　異なる交通需要による制御効果の違い..... 50
4.7.4　異なるプローブ車混入率による制御効果の違い..... 58
4.7.5　従来信号制御手法と提案信号制御手法における制御効果の違い..... 68
4.7.6　信号制御パラメータと制御効果の推移..... 77
4.8　修士論文審査会後の追加分析..... 80
4.9　まとめ..... 82
第５章　結論..... 83
5.1　本研究の結論..... 84
5.2　本研究の課題..... 84
参考文献..... 85
謝辞 86
附録 87
提案信号制御手法プログラムコード..... 88
交通状況データ作成プログラムコード..... 113
総遅れ時間算出プログラムコード..... 120
AI 入力データセット作成プログラムコード 124
LSTM ニューラルネットワーク構築・学習プログラムコード 126
従来信号制御手法プログラムコード..... 133

第１章　序論

1.1　研究背景.....5
1.2　研究目的.....7
1.3　論文構成.....8

第１章　序論

1.1　研究背景

1.1.1　信号制御の最適化

信号制御の最適化とは，計測された交通状況を表す指標を目標の値に近づけるよう，適切な信号制御パラメータを設定することである．最適な信号制御による最も大きな効果は，渋滞緩和である．とりわけ都市部において，渋滞発生の主な要因は信号交差点での捌け残りにあることから，最適な信号制御による渋滞緩和への寄与は大きいと考えられる．さらに，渋滞緩和によって，交通事故の抑制と排ガス量の低減が期待できる．渋滞により交通流の円滑性が失われると，運転手のストレスが溜まり，見切り発進や無理な駆け込みを誘発させ，交通事故のリスクが高まる．また，発進と停止が繰り返されると，排ガス量が増加する．自動車による排ガスが地球温暖化の大きな要因の１つであることから，持続可能な世界を実現するために，最適な信号制御は重要であるといえる．以上より，最適な信号制御を行うことによって安全で円滑な，環境にやさしい道路交通を実現することができる．

1.1.2　現行の信号制御手法

現在，日本で用いられている信号制御手法は，大きく分けて２つある．１つはプログラム多段制御と呼ばれるものである．この制御手法は，曜日や時間帯に応じてあらかじめ定められた信号制御パラメータにより運用されるものであるため，車両感知器や管制センターとの接続が不要かつ，交通需要がある程度パターン化されるような交差点においては，制御効果が期待される．１交差点内で完結し，コストを抑えられることから，日本では多く用いられている手法である．短所として，都市や地域の再開発等により，交通需要が信号制御の設計時から経年変化した場合や，瞬間的な交通需要の変動が起こった場合に，制御効果が薄れてしまうという点が挙げられる．もう１つの信号制御手法は，交通応答制御と呼ばれるものである．これは，車両感知器により得られたデータに基づいて，一定の時間間隔で青時間スプリット・サイクル長・オフセット（以下，信号制御パラメータ）を決定し，算出された各階梯秒数をもとに信号制御を行うものである．交通応答制御の中でもプログラム選択制御とプログラム形成制御がある．プログラム選択制御は，あらかじめ定められた複数の信号制御パラメータの組合せから交通状況に適したものを選択する制御である．長所として，プログラム多段制御と比較し，より交通状況に適した信号制御が可能となる点がある．短所はプログラム多段制御と同様，交通状況が経年変化した場合に，再度設計し直す必要がある点や，交通状況の変動がパターン化される交差点でしか適用できないという点がある．プログラム形成制御は，あらかじめ有限個の信号制御パラメータが設定されているというわけでなく，交通状況に適したものを自動生成する制御であり，現行の信号制御手法のなかで最も高度な手法（以下，現行高度信号制御手法）といえる．長所として，プログラム選択制御と比較して，より交通状況に適した信号制御が可能である点や，信号制御パラメータを自動生成するため，信号制御の設計が不要であるという点がある．短所として，この制御手法は車両感知器の利用を前提にしているという点が挙げられる．

1.1.3　交通応答制御の問題点

前節で述べた通り，現行高度信号制御手法では車両感知器が用いられる．しかし車両感知器は，以下の 2 点の問題を抱える．1 点目は，地方自治体をはじめとした厳しい予算状況や人材不足により車両感知器の新たな設置や維持管理が容易でなくなっているという設置・維持コストの問題である．このようなコストの問題から，車両感知器が設置されている区間は限定されている．関ら[1)]によれば，車両感知器を用いた交通管制システム制御対象エリアの道路区間長（上下方向別）は約 3,100km で，これは都内にある自動車専用道路を除く道路幅員 5.5m 以上の道路区間長約 12.200km の約 25%に相当する．また，約 16,000 箇所の信号機の 50%に相当する約 8,000 箇所が交通管制センターに接続されている．つまり，車両感知器を用いた交通応答制御を行うことのできる交差点は半数ほどに留まっている．2 点目は，特定車両の走行を追跡するのが難しいことから，遅れ時間の精度を確保できないというデータ精度の問題である．最も多く設置されている車両感知器は，超音波送受信器から超音波パルスを周期的に発射し，その往復時間の長短によって車両の存在を感知するものである．しかしこのような車両感知器は定点観測であるため，車両の存在を離散的に検知することしかできない．そのため，特定車両の走行を追跡することは難しく，遅れ時間の精度は確保されない．遅れ時間とは，実際に信号交差点を通過するのにかかった時間と，信号が無いと仮定したときに規制速度で走行した場合にかかる時間との差である．信号待ちによる渋滞損失は遅れ時間で表現することができるため，信号制御の目標は，遅れ時間を小さくすることであると言える．従って，より適切な信号制御を実現するためには，遅れ時間を精度よく把握する必要がある．その他，精度に問題があるのは，待ち行列長や流入交通量である．現行高度信号制御手法では，信号制御パラメータを決定する際に，待ち行列長や流入交通量のデータを必要とする．しかし，車両感知器の設置位置よりも上流側に，待ち行列長が延伸した場合，待ち行列長と流入交通量を直接観測することは不可能となる．その場合，周辺にある複数の車両感知器を用いて推定することとなり，精度に問題が生じる．以上より，車両感知器を用いた現行高度信号制御手法により期待できる制御効果には，限界があるといえる．

1.1.4　プローブ情報

近年，プローブ車一台一台の時刻と位置（緯度，経度）等を含むデータであるプローブ情報を交通状況の把握や交通施策に活用する動きが見られる．これは，スマートフォンや ETC，カーナビゲーション等の普及によって，より多くのプローブ情報を収集できるようになったことが背景にある．プローブ情報の活用例として，一般財団法人道路交通情報通信システムセンター（VICS センター）では，車両感知器のみならずプローブ情報も併せて活用することによって，より多くの区間における渋滞情報を提供する取り組みが行われている[13)]．このようなプローブ車の走行を追跡することによって，前節で述べた遅れ時間を高精度に計測することが可能となる．しかし，プローブ情報を活用するにあたって，問題点が 2 点ある．1 点目はデータの伝送遅れの問題である．通常，プローブ車からデータが送信されてから，直接生のデータが利用者へ届くのではなく，一度サーバへ蓄積されたのちに遅れ時間などの交通状況を表現する指標に加工されたデータが利用者へと送られる．そのため，プローブ情報が送信されてから，利用者へ届くまでの間に伝送遅れが生じる．これにより，時々刻々変動する交通状況下におい

て，データが届いた頃には，既に異なる交通状況になっている可能性が考えられる．したがって，この伝送遅れを考慮した信号制御を行う必要がある．2 点目はプローブ車の混入率の問題である．吉岡ら[2)]によると，日中のプローブ車混入率は約 12%であった．この値は，交通需要の変動によっては，プローブ車が一台も通過しないという状況もあり得るため，高精度に時々刻々の遅れ時間を算出するためには必ずしも混入率が十分であるとは言えない．得られているプローブ情報に基づいて，将来の総遅れ時間が精度よく算出できれば，上述した伝送遅れや混入率の問題点は解決できると考えられる．

1.2　研究目的

本研究は，伝送遅れと混入率の低さの問題点を踏まえた上で，プローブ情報を活用した予測遅れ時間に基づく信号制御手法を構築し，シミュレーション評価を行うことを目的とする．シミュレーションを用いた評価を行うことで，様々な条件下における制御効果が定量的に示され，当該手法の課題や適用要件について整理を行うことが可能となる．本研究の成果により，プローブ情報を活用した信号制御の実用化が推進されることを期待している．将来，実用化されれば，より交通状況の実態に即した最適な信号制御を実現することができ，1.1.1 で挙げた，渋滞緩和・交通事故の抑制・排ガス量の低減といった効果が期待できる．また，1.1.3 で挙げた，車両感知器の設置・維持コスト・データ精度の問題を解決することができる．

1.3　論文構成

本論文は全５章で構成される．

第１章「序論」

第１章では，研究課題の背景を整理し本研究の目的を示す．

第２章「既往研究」

第２章では，既に行われた関連研究をレビューし，本研究の位置づけを明らかにするとともに新規性を示す．

第３章「提案信号制御手法の構築」

第３章では，提案信号制御手法の概要を説明したのちに，AI と探索の２つのブロックに分けて，それぞれ詳細に説明をする．

第４章「シミュレーション評価」

第４章では，構築した提案信号制御手法による制御効果を，交通シミュレーションを用いて確かめる．様々な条件における制御効果を検証することで，当該手法の課題や適用要件についての考察を行う．

第５章「結論」

第５章では，本論文における研究の結論を述べ，今後の課題を考察する．

附録

附録として，本研究で扱ったプログラムコードを添える．

第２章　既往研究

2.1　既往研究レビュー..... 10
2.2　本研究の位置づけ..... 11

第２章　既往研究

2.1　既往研究レビュー

信号制御にプローブ情報を活用することを試みる既往研究は，車両感知器とプローブ情報を併用するものと，プローブ情報単体のみを使うものの２つに大別される．

まず，車両感知器とプローブ情報を併用した既往研究を紹介する．長島ら[3)]は，プローブ情報を活用することにより，信号制御パラメータ算出のために必要な待ち行列長の推定精度の向上を図った．車両感知器による計測だけでなく，プローブ情報も組み合わせることにより制御効果の向上が見られたが，プローブ車混入率が低い場合は，高い場合と比べて制御効果が小さくなったとしている．また，西内ら[4)]は，車両感知器により得られる交通需要・飽和交通流率と，プローブ情報により得られる区間旅行時間を用いて，交通シミュレーションを実行することで，観測された交通状況に対し，平均遅れ時間を最小にする信号制御パラメータを決定する手法を構築した．その結果，現状の平均遅れ時間に対して，15 秒だけ短縮効果が期待できる信号制御パラメータを見つけることができた．しかし，最適な信号制御パラメータの探索に，交通シミュレーションを用いており，計算に時間がかかるため，効率的な制御手法であるとは言えない．

次にプローブ情報のみを活用した既往研究を紹介する．花房ら[5)]は，信号制御の評価指標のための遅れ時間を，プローブ情報のみを使って推定を行った．その結果，プローブ車混入率が高い（50～70%）場合は，相応の精度で推定されていたが，混入率が低い（5～10%）場合は，過小評価になっていることが分かった．また今後の課題点として，プローブ情報の伝送遅れを考慮する必要性が挙げられた．吉岡ら[2)]は，車両感知器の計測情報を使わずに，プローブ情報から得られた平均旅行時間と現状の信号制御パラメータを組み合わせて，負荷率を算出する方法を提案した．課題として，プローブ車混入率が 10%程度の時に，捌け残りが発生しない非飽和状態ではプローブ情報から負荷率を算出するのが困難であることが挙げられた．関ら[1)]は，定点設置する車両感知器では検知されない渋滞を，プローブ情報を用いることで把握可能であることを確認した．また，車両感知器が設置されない従道路の交通状況をプローブ情報により把握することで，適切な信号制御を行うことができ，幹線道路の渋滞対策に効果を発揮することを確認した．しかし，この研究では，ある期間・時間帯におけるプローブ情報を抽出し，分析したのちに後日，同じ時間帯の信号制御パラメータを調整するという手法が取られており，プローブ情報に基づいた即時的な信号制御は行われていない．塚田[6)]は，プローブ情報から読み取った遅れ時間を活用し，プログラム多段制御の信号制御パラメータの見直しを行う手法の提案を行った．この手法により，ある条件においては，平均旅行時間が短くなった．しかし，どのような条件において制御効果を最大限に発揮しうるかについて，検討が必要であるとしている．

以上より，プローブ情報を信号制御へ活用するにあたっての課題は以下の通りである．

- プローブ情報の伝送遅れの影響
- プローブ車混入率の低さの影響
- 最適な信号制御パラメータの探索時間が長いこと

2.2　本研究の位置づけ

本研究では，既往研究で言及されてきた，プローブ情報単体を信号制御に活用する際の課題点を踏まえて，新たな信号制御手法の構築・シミュレーション評価を行う．様々な条件下における制御効果を定量的に示すことによって，課題や適用要件について整理を行う．本研究の成果が，プローブ情報を活用した信号制御の実装に関する是非について，議論を深めるための基礎的な知見として用いられることを想定する．

本研究の新規性として，プローブ情報の伝送遅れを考慮している点や，プローブ車混入率の低さを考慮している点，最適な信号制御パラメータの探索に AI を取り入れることで効率的な探索を可能にしている点，現行の信号制御手法の枠組みにとらわれない新たな信号制御手法を構築している点が挙げられる．

既往研究

1)　関達也，島津利行，和智誠，榊原肇，大口敬：プローブ情報を活用した信号制御の見直しについて，交通工学論文集，第 8 巻，第 1 号，pp.31-38，2022.1

2)　吉岡利也，榊原肇，テンハーゲンロビン，ローコウスキステファン，大口敬：プローブカーデータを用いた信号制御パラメータ算出手法，生産研究，74 巻 1 号，pp.115-122，2022

3)　長島靖，服部理，小林雅文：プローブ情報の活用による信号制御高度化，SEI テクニカルレビュー，第 184 号，pp.40-pp.43，2014.1

4)　西内裕晶，吉井稔雄：簡易車両感知器とプローブカーを用いた信号制御システム　～システムの構築と道路工事実施時の片側交互通行区間への適用～，交通工学第 39 巻，4 号，p.46，2004

5)　花房比左友，飯島護久，堀口良太：リアルタイム信号制御アルゴリズムのためのプローブ情報を利用した遅れ時間評価，第 8 回 ITS シンポジウム，2009

6)　塚田悟之：プローブ情報を活用した信号制御定数パターン見直し支援システムの構築，情報システム学会誌，Vol.14，No.2，pp.65-78，2018.12

第３章　提案信号制御手法の構築

3.1　概要 13
3.2　AI による予測 14
3.3　探索 18

第３章　提案信号制御手法の構築

3.1　概要

提案信号制御手法では，現時点までに得られている交通状況と信号情報に基づきながら，将来のあらゆる信号制御の中で，最小の予測遅れ時間をもたらす信号制御を実施する．将来の遅れ時間の予測にはAIを用い，予測遅れ時間を最小にする信号制御は探索により見つける．**図3.1**に提案信号制御手法のフローを示した．次節以降，各段階の詳細を説明する．

2.1の既往研究レビューで挙げた，プローブ情報を信号制御へ活用するにあたっての課題に対しては，以下のように考慮した．

- プローブ情報の伝送遅れの影響
  将来の予測された遅れ時間に基づいた信号制御を行う

- プローブ車混入率の低さの影響
  プローブ車と非プローブ車両方を含む，全車両を計測対象とした遅れ時間に基づいた信号制御を行う

- 最適な信号制御パラメータの探索時間が長いこと
  AIを用いることで，予測遅れ時間を素早く算出する

次に現行高度信号制御手法との違いについて触れる． 現行高度信号制御手法では，車両感知器による計測データを利用する．一方，提案信号制御手法では，プローブ情報を活用する．また，現行高度信号制御手法では，流入交通量と捌け残り台数の和を飽和交通流率で除した値である負荷率に基づいて，信号制御パラメータを決める．一方，提案信号制御手法では，予測遅れ時間が最小となるように，信号制御パラメータを決める．また，オフセットを段階的に変

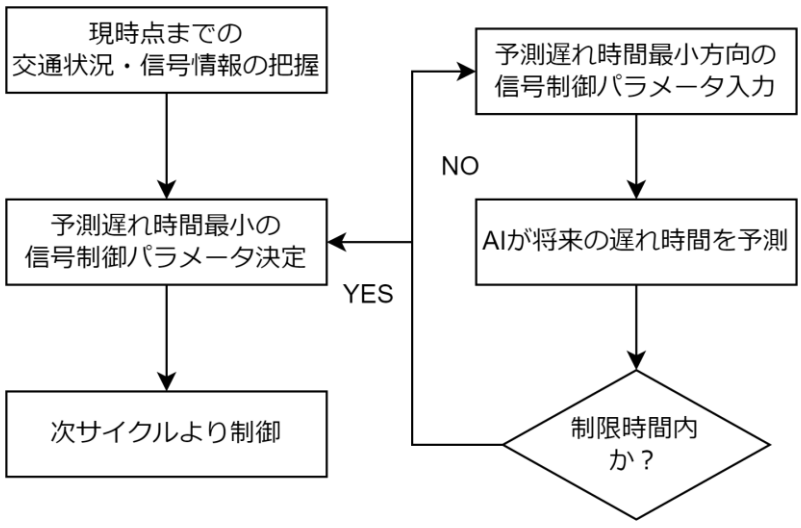

図3.1　提案信号制御手法のフロー

化させることで交通流への影響を緩和させるオフセット追従等の信号制御パラメータの制約が含まれている現行高度信号制御手法に比べて，提案信号制御手法ではそのような制約を取り除くことによって，信号制御パラメータの自由度が高くなっているため，遅れ時間をより小さくできる可能性がある．

提案信号制御手法の構築にあたり，使用する言語は Python とした．

3.2　AI による予測

3.2.1　AI とは

AI(Artificial Intelligence)とはコンピュータの性能が大きく向上したことを背景とし，人間にとって代わって,コンピュータが学習・理解・行動をする技術の総称である．AI の基盤となる技術である機械学習は，収集されたデータから法則性を見出す手法である．中でも，人間の神経細胞（ニューロン）のモデルをネットワーク化し，脳の中の情報処理を模したニューラルネットワーク（以下，NN）は，近年著しく発展している．土木工学分野においても，コンクリートの品質管理や損失評価，景観評価など広くその応用が試みられている[7]．しかし，ニューラルネットワークは入出力されるデータが互いに独立しているとして扱われる．そのため，時系列データは扱うことができない．

そこで，登場したのがリカレントニューラルネットワーク（以下，RNN）である．RNN は，NN にはなかった過去の状態を保持する機構を持っており，現時点での入力に対し，過去の状態を反映した出力が可能である．しかし RNN では重みを最適化する目的関数の微分値となる勾配が，ほぼ 0 となる勾配消失，或いは無限大となる勾配爆発により長期依存を学習できないといった問題がある．

そこで，RNN の拡張モデルとなる LSTM(Long Short Term Memory)層が登場した．LSTM 層では，内部状態を記憶するメモリセルと 3 つ（忘却・入力・出力）のゲートを導入することにより勾配消失，或いは勾配爆発を軽減させ，短期的なものだけでなく，長期的な時系列データも扱うことが可能となった．**図 3.2** は RNN と LSTM それぞれの内部構造を表したものである．

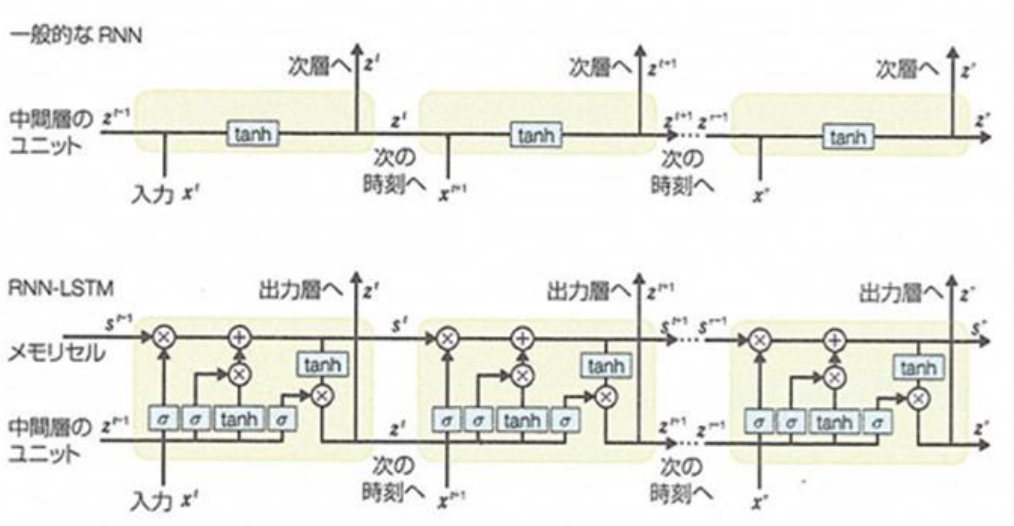

図 3.2　RNN と LSTM の内部構造

（参考：「山下隆義著，イラストで学ぶディープラーニング改訂第 2 版」より引用）

3.2.2　本手法で AI を用いる理由

AI を用いることによって，瞬時に予測総遅れ時間を算出することができ，最適な信号制御パラメータの探索を効率よく行うことが可能となる．その他，予測総遅れ時間を算出する方法として，交通シミュレーションの結果を用いることが考えられる．しかしこの場合，予測総遅れ時間の算出自体，AI 以上に時間がかかる．その結果，探索に膨大な時間がかかってしまい，現実的な時間内での探索は不可能となる．以上の理由より，本研究では，予測総遅れ時間の算出に AI を導入する．

3.2.3　本手法で用いる AI の構造

本手法では，交通状況データという長期的な時系列データを取り扱うため，LSTM 層を含むニューラルネットワークを用いることとする．

3.2.4　AI の入力データ

AI の入力データは大きく 2 つある．交通状況データと信号制御情報の時系列に並べたものである．1 タイムステップにおける各データについて説明する．

交通状況データとは，プローブ情報に基づいて，交通状況を定量的に表す指標へ加工されたものである．具体的には 1 分間平均遅れ時間と 1 分間平均流出台数である．平均遅れ時間とは，実際に信号交差点を通過するのにかかった時間と，信号が無いと仮定したときに規制速度で走行した場合にかかる時間との差を，車両 1 台あたりに平均した値である．直近 1 分間に交差点を流出した車両がリンク内を通過するのに発生した平均遅れ時間を算出したものを，直近 1 分間平均遅れ時間とする．さらにこれを，直近 30 分間分だけ収集したのちに，移動平均したものを 1 分間平均遅れ時間とし（式(a)），1 タイムステップごとに入力する．

$$\boldsymbol{T_k = \frac{1}{30}\sum_{l}\left(\frac{1}{n_l}\sum_{i}\left(t_{i,k,l} - \frac{l_k}{\widetilde{v_k}}\right)\right)} \qquad (\boldsymbol{a})$$

T_k：リンク k内の平均遅れ時間[秒]
$t_{i,k,l}$ ： 時間lのリンク k内の車両i の旅行時間[秒]
$\widetilde{v_k}$ ： リンク k内の規制速度[m/秒]
l_k ：リンクkの距離[m]
n_l： 時間lの全流出車両台数[台]

流出台数とは，停止線を通過した車両の台数である．流出台数も同様に，1 分間に流出した車両台数を直近 30 分間分だけ収集したのちに，移動平均したものを 1 分間平均流出台数とし（式(b)），1 タイムステップごとに入力する．

$$\boldsymbol{V_{k,d} = \frac{1}{30}\sum_{l} v_{k,d,l}} \qquad (\boldsymbol{b})$$

$V_{k,d}$：リンク k内の流出dの平均流出台数[台]
$v_{k,d,l}$ ： 時間lのリンク k内の流出d の流出台数[台]

以上の各データは，プローブ情報により算出したものであるため，全車両についての交通状況を表した値ではないことに留意されたい．

信号制御情報とは，各階梯秒数・サイクル経過時間・オフセットをまとめたものである．各階梯秒数とは，信号表示の切り替えの最小単位ごとの表示時間のことである．サイクル経過時間とは，サイクル開始時点からの経過秒数である．オフセットとは，隣接する交差点と対象交差点における青信号開始のずれのことである．系統制御等により複数交差点を制御対象とする場合は，交差点数分だけ信号制御情報を収集し，1 タイムステップごとに入力する．

交通状況データと信号制御情報は，1 分おきに送信され，直近 60 分間分（60 タイムステップ分）を収集したのちに変数ごとに標準化する．標準化を行う理由は，変数のスケーリングを揃えるためである．変数によってスケーリングが異なる場合，スケーリングの大きい変数の重みが大きくなり，偏りが生じる恐れがある．これにより正常に学習することができなくなる可能性がある．これを防ぐために標準化を行う．1 タイムステップには，交通状況データの発生時刻に対し，15 分後の信号制御情報を入れる．**図 3.3** にデータセットの形式の一例を示す．

例：
8:30時点のデータ

●：更新済
－：未更新

発生時刻	交通状況データ	信号制御情報(15分後)
7:28	●	●
⋮	⋮	⋮
8:15	●	●
8:16	●	－
⋮	⋮	⋮
8:26	●	－
8:27	●	－
8:28	－	－
8:29	－	－
8:30	－	－

AIに入力するデータセット（60ステップ分）

伝送遅れ（3分）

図 3.3　入力データセットの形式

図 3.3 の一例では，8:30 までに送信されてきた累積データを表している．これまでに述べてきたようにプローブ情報は伝送遅れが生じ，その長さはおよそ 3 分である．そのため，8:30 に送られてくる交通状況データは，3 分前の 8:27 に送信されたプローブ情報をもとに加工されてできたものであるため，実際には 8:27 に発生したとしてデータを並べる．その上で，発生時刻の 15 分後の信号制御情報を並べる．この一例では 8:30 までの信号制御情報が分かっているため，発生時刻 8:15 の行まで，信号制御情報が分かっている．発生時刻 8:15 から 8:27 の 12 分間の信号制御情報は，次節で説明する，探索の変数とする．以上のことを考慮すると，8:30 に AI へ入力するデータセットは．**図 3.3** の赤枠で囲んだ部分となる．

3.2.5　AI の出力データ

AI の出力データは，プローブ車・非プローブ車の両方を含んだ全車両の挙動を予測した結果における 10 分後から 15 分後の間の総遅れ時間（以下，予測総遅れ時間）である（式(c)）．

$$T = \sum_{k}\left(\sum_{i}\left(t_{i,k} - \frac{l_k}{\widetilde{v_k}}\frac{l_{i,k}}{l_k}\right)\right) \qquad (c)$$

T： 全リンク　内の総遅れ時間[秒]
$t_{i,k}$ ： リンク　k内の車両i　の旅行時間[秒]
$\widetilde{v_k}$ ： リンク　k内の規制速度[m/秒]
$l_{i,k}$ ： リンク　k内の車両i　の走行距離[m]
l_k ：リンクkの距離[m]

図 3.4 は，3.2.3～3.2.5 までを一連の流れとして図にまとめた，モデルの概要である．

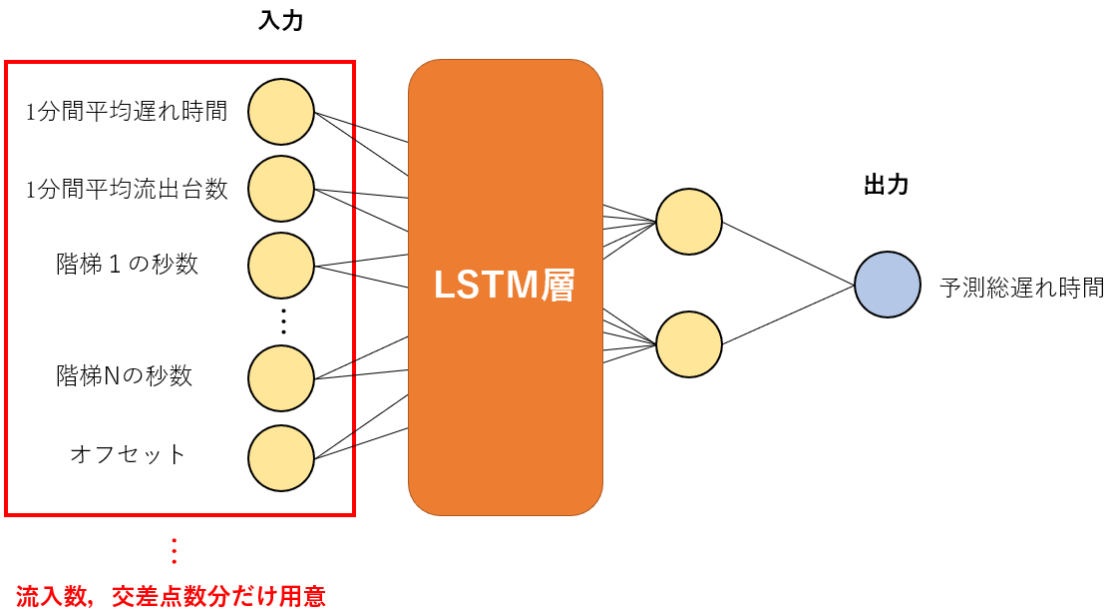

図 3.4　モデルの概要

3.2.6　AI の学習

AI に，各交差点の階梯秒数に乱数を与えた交通シミュレーションの結果を学習させる．これにより，交通シミュレーションの結果をそのまま用いる場合と同等の予測総遅れ時間を短時間で算出可能にした．AI の学習にあたり，交通シミュレーションの結果を用いて，学習データを収集した．学習データ収集のフローを**図 3.5** に示す．学習データ収集のための，交通シミュレーション設定条件や学習結果は第 4 章で示す．

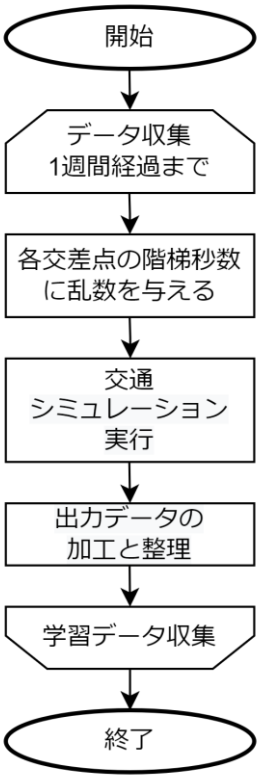

図 3.5　学習データ収集のフロー

3.3　探索

3.3.1　概要

探索の目標は，予測総遅れ時間の最小にする信号制御を行うことである．探索はヒューリスティックを適用する．ヒューリスティックとは，必ずしも厳密解を求められるとは限らないが，ある程度の精度で厳密解に近い値を得ることができる方法である．ヒューリスティックは，厳密解が保証されない代わりに，求解までにかかる時間が短いといった利点を持つ．最適化する変数は，将来の信号制御パラメータである．目的関数は，予測総遅れ時間を算出する AI を含む関数である．

3.3.2　最適化する変数

最終的に最適化したいものは，将来の信号制御パラメータであり，これは，主方向青時間スプリット・重要交差点のサイクル長・重要交差点サイクル長との差である．この前段階として探索する変数は，$x(-\infty,\infty)$の実数である．実数$x(-\infty,\infty)$と信号制御パラメータの関係は式$(\boldsymbol{d})$に示した通りである．

$$\boldsymbol{P}=\frac{min-MAX}{1+\exp(x)}+\boldsymbol{MAX} \qquad (\boldsymbol{d})$$

P： 信号制御パラメータ（主方向青時間スプリット・重要交差点サイクル長・重要交差点サイクル長との差）
x ： $(-\infty,\infty)$の実数
MAX ： 信号制御パラメータの最大値
min ： 信号制御パラメータの最小値

これを変数にした理由は，変数の制約条件を無くすこと，より自由度の高い探索を行うことができ，結果として，より適切な信号制御パラメータを発見することができる可能性があると考えたからである．最適化したい将来の信号制御パラメータの個数分だけ，xを用意する必要がある．必要な信号制御パラメータは，主方向青時間スプリット・重要交差点のサイクル長・重要交差点サイクル長との差である．例えば，将来 N サイクル分までの M 個の信号交差点の青時間スプリット・重要交差点サイクル長・重要交差点サイクル長との差を最適化したい場合，用意すべきxの個数は，主方向の青時間スプリット M 個と，重要交差点のサイクル長 1 個と，それ以外の交差点における重要交差点サイクル長との差（M－1）個を合わせて，2M 個となる．オフセットに関しては，重要交差点のサイクル長との差で考慮される．

3.3.3　目的関数

次に目的関数について説明する．まず変数である信号制御パラメータを，12 分間 1 分刻みの信号制御情報へ変換する．この 12 分間の信号制御情報を組み込んだ AI の入力データに対し，AI が予測総遅れ時間を算出する．**図 3.6**（この章の最後）に，目的関数のフローを示す．

3.3.4　探索アルゴリズム

Python のライブラリの 1 つである SciPy を用いて，探索を行う．SciPy とは，数学、科学、工学分野のための数値解析ソフトウェア・ライブラリである．局所解を求める手法として用いるものは Nelder-Mead，Powell，CG，BFGS，SLSQP である．大域解を求める手法として用いるものは，DE，SA を用いる．局所解を探索するメリットは，初期値に対して大きく値が変わることはないため，信号制御パラメータが急激に変動することはないことである．デメリットは，必ずしも予測総遅れ時間が最小であるとは限らないことである．一方で，大域解を探索するメリットは，取り得る変数の中で，予測総遅れ時間が最小となることである．デメリットは，信号制御パラメータが急激に変動する可能性があり，交通運用上，望ましいものではない．以

下，これらの探索アルゴリズムを簡単に説明する．

- Nelder-Mead
  n 次元ユークリッド空間の実数値関数 f(x)を与え，その下で f(x)の最小値を微分に頼らずに求める方法の一つである．
- Powell
  厳密には Powell の共役方向法と呼ばれる．Michael JD Powell によって提案された，関数の極小値を見つけるためのアルゴリズムである．関数は微分可能である必要はなく，導関数は使用されない．
- CG
  共役勾配法とも呼ばれる．連立一次方程式の数値解法で，1952 年に M.R.Hestenes と E.Stiefel によって提案された．

- BFGS
  非制限非線形最適化問題に対する反復的解法の一つである．BFGS 法は山登り法の一種である準ニュートン法に属しており，関数の停留点を探索する．
- SLSQP
  逐次最小二乗法とも呼ばれ，非線形最適化のための反復解法の一つである．目的関数と制約関数の両方が二階微分可能であるような問題に対して使われる．
- DE
  Storn と Price によって 1995 年 12 月に提案された，単目的最適化アルゴリズムである．基本的には遺伝的アルゴリズム（GA）とほぼ同じである．GA とは，データ（解の候補）を遺伝子で表現した「個体」を複数用意し，適応度の高い個体を優先的に選択して交叉・突然変異などの操作を繰り返しながら解を探索するものである．
- SA
  広大な探索空間内の与えられた関数の大域的最適解に対して，よい近似を与える． S. Kirkpatrick、C. D. Gelatt, M. P. Vecchi らが 1983 年に考案し，1985 年に V. Cerny が再発見した．焼きなまし法とも呼ばれるのだが，これは金属工学における焼きなましから来ている．焼きなましは，金属材料を熱した後で徐々に冷やし，結晶を成長させてその欠陥を減らす作業である．

3.3.5　探索の初期値

探索の初期値は，2 種類を考える．1 つ目は，需要率に基づいて算出された信号制御パラメータが将来 12 サイクル分継続した場合である．2 つ目は，前のサイクルで適用していた信号制御パラメータが将来 12 サイクル分継続した場合である．

3.3.6　探索時間

探索を行い，局所解或いは大域解に落ち着くまでに，通常の場合，数時間～数日程度かかる．しかし，交通状況は時々刻々変動するため，信号制御を行う上で，このような探索時間は

実用的であるとは言えない．したがって，本手法では探索時間に制限を設けることとする．その制限とは，「前のサイクルが終了するまで」というものである．このようにすることで，前のサイクルが終了したのちに次のサイクルから，限られた探索時間内で最適化された信号制御パラメータを実行することができる．

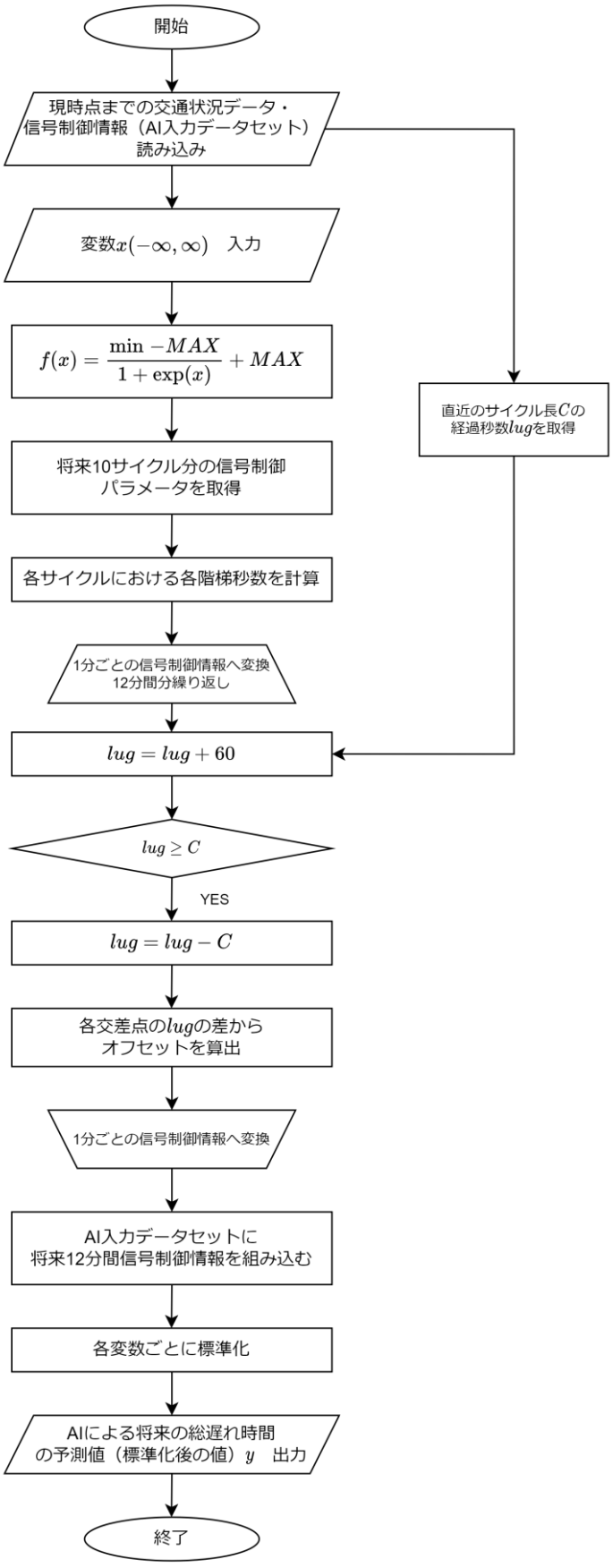

図 3.6　目的関数のフロー

第４章　シミュレーション評価

4.1　概要 24
4.2　諸条件 24
4.3　シミュレーションの流れ 27
4.4　学習結果 35
4.5　制御効果の評価指標 37
4.6　従来信号制御手法による信号制御パラメータ算出 37
4.7　提案信号制御の制御効果 40
4.8　修士論文審査会後の追加分析 80
4.9　まとめ 82

第４章　シミュレーション評価

4.1　概要

シミュレーション評価では，プローブ情報を信号制御に活用する際の課題や要件について，考察することを目的とする．そのために，交通シミュレーションを用いることで，様々な条件における提案信号制御手法による制御効果を定量的に示す．評価に交通シミュレーションを用いる理由は，低コストかつ，実社会へ影響を及ぼすことなく，様々な条件下における制御効果を把握することができるためである．一方，実証実験の場合，コストがかかり，不測の事態によって実社会へ負の影響を及ぼす可能性がある．こうしたことから，交通シミュレーションによる評価を行う．本研究では，株式会社アイ・トランスポート・ラボより提供されている AVENUE[11]を用いる．

4.2　諸条件

4.2.1　ネットワーク構成

ネットワークのモデルは，東京都世田谷区にある駒沢通り（都道 416 号）の東西約 700m，「東京医療センター前」，「東京医療センター正門前」，「東根小学校南」の３交差点である（**図 4.1**）．リンク数は７個で，ノード数は８個である．リンク長は「東京医療センター前」以西が

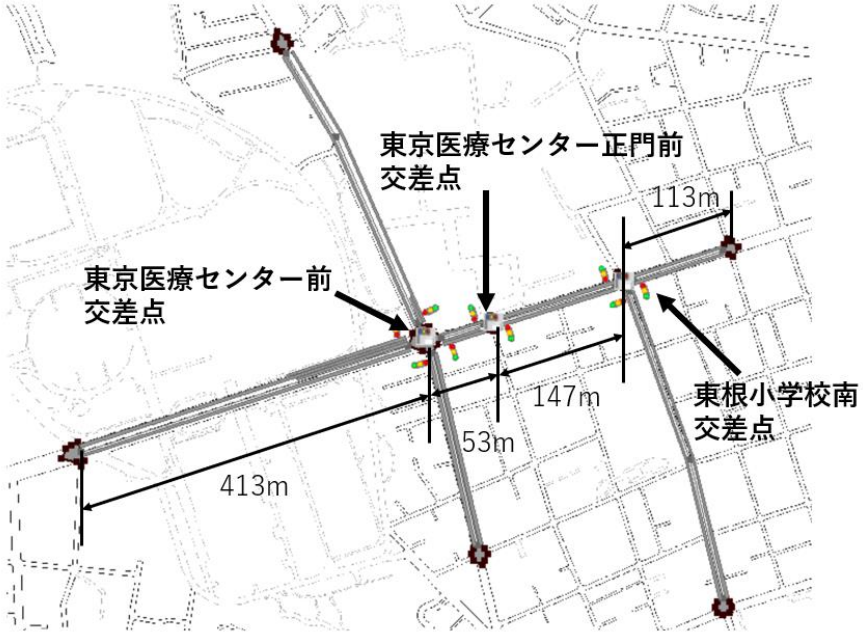

図 4.1　ネットワークモデル

413m，「東京医療センター前」と「東京医療センター正門前」の間が 53m，「東京医療センター正門前」と「東根小学校南」の間が 147m，「東根小学校南」以東が 113m である．次に各交差点の車線構成について説明する．本研究において，車両を発生させるのは東行きのみであるため，東行き流入部の車種構成のみ説明する．「東京医療センター前」は直進・右折・左折専用

車線の 3 車線である．「東京医療センター正門前」は直進・右折・左折混用の 1 車線である．「東根小学校南」は直進・右折混用の 1 車線である．

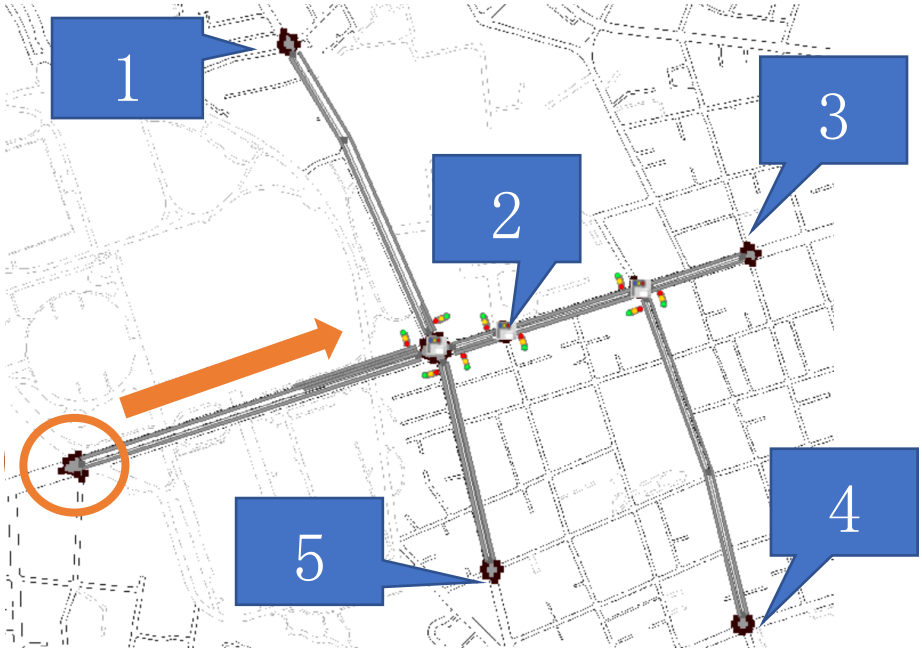

図 4.2　OD となるノードの位置

4.2.2　交通条件

交通条件で設定するのは，10 分おきの OD 交通量・車種構成・プローブ車混入率・希望速度である．OD 交通量を設定するにあたり，まずは発生交通量を決める必要がある．発生交通量は，全国道路・街路交通情勢調査 [12] の結果を参考にした．全国道路・街路交通情勢調査とは，道路の整備計画に向けて，現状の交通状況を把握するために，国土交通省が 5 年に 1 度行うものである．この調査結果にある，「昼間 12 時間交通量」と「ピーク比率」からピーク時間帯の交通量を算出し，半分の値にすることで，片方向の交通量を算出する．さらに 1/6 を乗じることで 10 分間の交通量を算出する．この値を平均発生交通量とした．この平均発生交通量を設定したのちに，ポアソン分布を仮定した確率分布で 10 分おきに変動させることで，10 分間発生交通量を設定する．さらに，この 10 分間発生交通量に対して，各 OD 交通量にランダムに振り分ける．このようにして，10 分おきの OD 交通量を設定する．OD は，橙色ノードにおいて車両を発生させ（Origin），**図 4.2** の青吹き出し 1～5 が指すノードにおいて車両を到着させる（Destination）．ここで，本来であれば，1～5 のノードからも車両を発生させることで，より現実に近いネットワークを設定するべきである．しかし，本研究の初期段階においては信号制御手法の構築にフォーカスを当てており，この段階ではネットワーク設定条件をできる限り軽くすることで，万が一，制御効果が悪かったり，制御手法がうまく回らなかった場合に，原因を特定するのが容易になることから，橙色ノードの 1 か所のみから車両を発生させることとした．車種は全車両，小型車で，希望速度は，規制速度である 40 km/h とする．またプローブ車混

入率は，30%を基本として，10%，50%，70%，90%のケースを用意する．また，今回のケースでは従方向の車両を発生させない代わりに，従方向スプリットは 0.4 以上確保するように制約をかける．以上をまとめたものが，**表 4.1** である．ここで，プローブ車の決め方について説明する．AVENUE からは全車両のプローブ情報が送信されてくるため，全車両の ID を取得する．ここで，0 から 1 までの 0.1 刻みの数字を用意し，ここからランダムに 1 つ選ぶ．これを，ある車両の ID と紐づけする．もしもこの番号が 0.3 以下の場合は，プローブ車としてその車両のプローブ情報を取得する．一方で 0.4 以上の場合は，非プローブ車として扱い，その車両のプローブ情報は破棄される．この説明をフローにしたものが**図 4.3** である．

表 4.1　交通条件

10 分間平均発生交通量（台） （平均に対し，ポアソン分布を仮定した 確率分布で 10 分おきに変動）	53（閑散時交通量） 107（混雑時交通量） 214（超混雑時交通量）
車種構成	小型車のみ
プローブ車混入率（%）	10，30，50，70，90
希望速度（km/h）	40（規制速度）
従方向の青時間スプリット	0.4 以上を確保

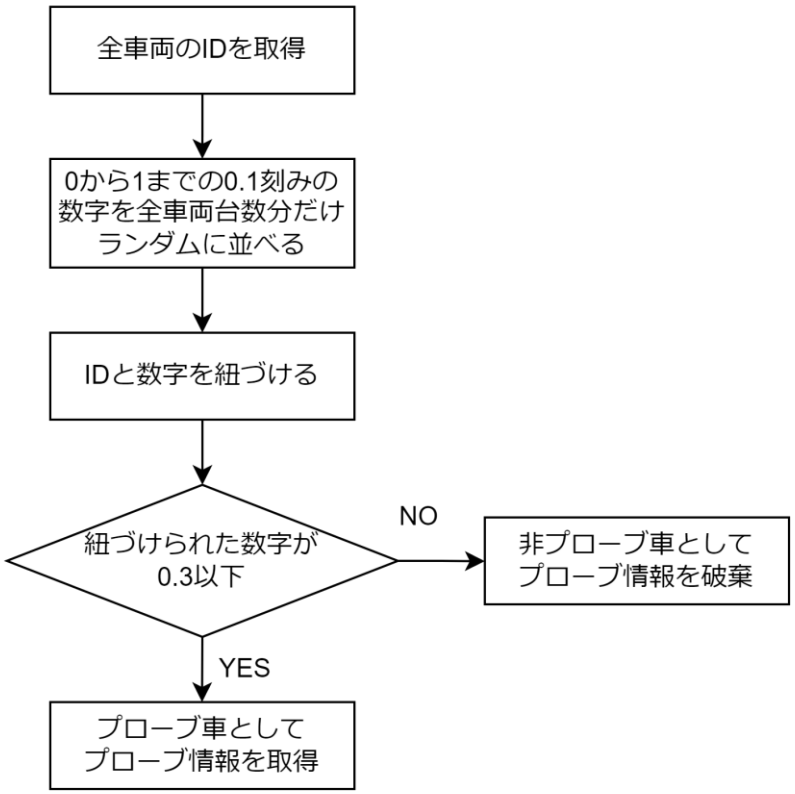

図 4.3　プローブ車の決め方

4.2.3　シミュレーション実行条件

シミュレーションの実行条件では，シミュレーション開始・終了時刻と，発生交通量・OD 交通量に対してランダムな変動を与えるシード値，さらには車両発生間隔のシード値を設定する．シミュレーション開始時刻は 6:50 で，終了時刻は 9:30 とする．シード値は 100 から開始し，サンプル数に応じて 1 刻みで大きくしていく．車両発生間隔のシード値は，1000 で固定する．以上をまとめたものが，**表 4.2** である．

表 4.2　シミュレーション実行条件

シミュレーション時間	6:50～9:30
発生交通量のシード値	100 から開始し，サンプル数に応じて 1 刻みで増やしていく
車両発生間隔のシード値	1000

4.2.4　提案信号制御手法の汎用性

今回，対象とするネットワークは，駒沢通りとしたが，他の交差点でも適用できるかどうかについて述べる．提案信号制御の枠組み自体は，他の交差点においても適用可能であるが，AI の学習データの収集と学習の工程は，各自，制御手法利用者が行う必要がある．また，学習データの収集方法については，交通シミュレーションによる収集と，実際の交差点でのプローブデータによる収集の 2 通りがある．現段階においては，交通シミュレーションによる収集を推奨する．理由は，現段階においては，プローブデータの精度検証が途上にあり，信号制御への活用に足るものであるかどうかが明らかとなっていないためである．

4.3　シミュレーションの流れ

4.3.1　概要

図 4.4 にシミュレーションの全体フローを示す．シミュレーションを動かす司令塔は信号制御フレームワーク（以下，信号制御 FW）が担う．信号制御 FW は 1 秒おきに AVENUE に対し，シミュレーションを動かすよう指示する．AVENUE から信号制御 FW を介して，プローブエミュレータに，プローブ情報と信号情報が 1 秒おきに引き渡される．プローブエミュレータはこれらの情報を加工・分析し，交通状況データを作成後，信号情報と合わせて AI の入力データセット形式に沿って時系列に整理する．これに基づいて提案信号制御手法が，将来の最適信号制御パラメータを決定する．決定後，適切な形式に合わせて，信号制御 FW を通じて，AVENUE へ送信し，次の 1 秒のステップを動かすよう指示をする．このサイクルを繰り返すことで，シミュレーションを動かしていく．

次節以降，「プローブエミュレータまでの流れ」，「提案信号制御手法以降の流れ」の 2 節に分けて詳細に説明する．

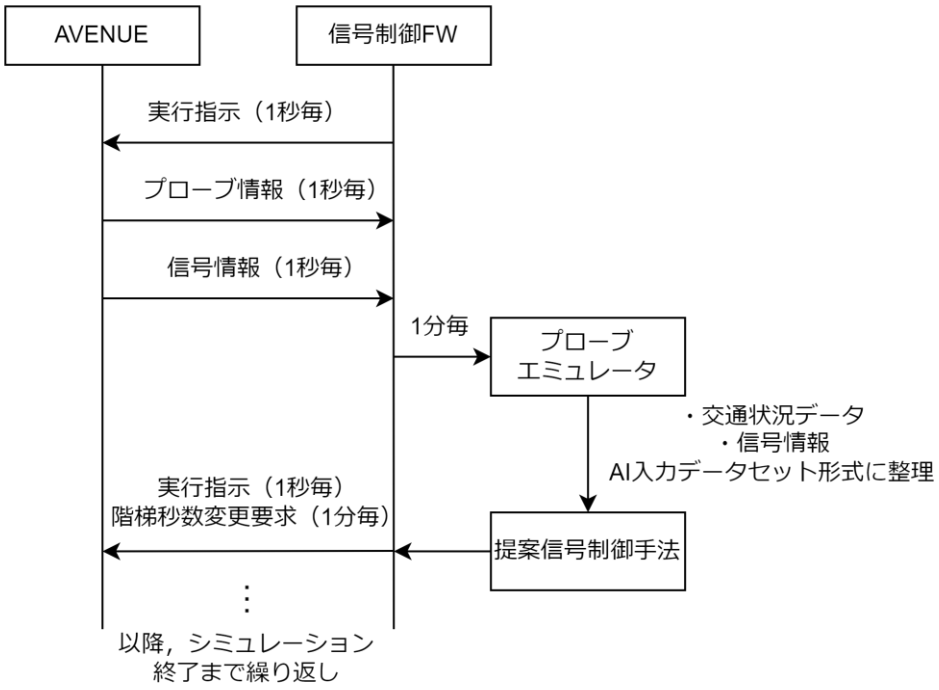

図 4.4　シミュレーションの全体フロー

4.3.2　プローブエミュレータまでの流れ

プローブエミュレータまでの流れを，順を追って詳細に説明する．まず，ソケット通信により AVENUE と信号制御 FW を接続する．AVENUE がクライアント型，信号制御 FW がサーバ型である．その次に，信号制御 FW から AVENUE へ初期設定値の送信を行う．初期設定値の項目ならびに本手法における設定値を**表 4.3** に示した．

表 4.3　ソケット通信の初期設定値

初期設定値	補足	設定値
情報の送信間隔	AVENUEが外部モデルへ情報を送信する時間間隔[秒]。	1
プローブ情報送信有無	0:無し。 1:有り。	1
信号情報送信有無	0:無し。 1:有り。	1
感知器集計情報有無	0:無し。 1:有り。	0
感知器パルス情報有無	0:無し。 1:有り。	0
プローブ情報出力対象車種番号	外部モデルへ車両情報を出力する車種番号マスク値(-1は全車種)。	-1
プローブ混入率	プローブ車両の混入率[%](1%刻み)。	100
プローブ情報送信種別	プローブ車両情報収集種別。 0:プローブ情報間隔に従って随時送信。 1:感知器を通過した際に蓄積情報を送信。	0
プローブ情報収集単位種別	プローブ情報収集単位の種別。 0:時間[秒] 1:距離[m]	0
プローブ情報収集間隔(距離or時間)	プローブ車両情報収集間隔[秒]。 プローブ情報収集単位種別が0(時間)の場合は秒。 プローブ情報収集単位種別が1(距離)の場合はm。	1
感知器集計交通量時間単位	感知器集計情報(交通量)の時間単位[秒]。	0
感知器パルス情報集約時間単位	感知器パルス情報を集約する時間単位[秒]。通過時刻単位は1秒。たとえば、60秒とすると、感知器パルス情報は60秒間の通過履歴(バイト列で通過時刻をOn:1/Off:0で表現する)として送信する。	0

次のフェーズからは，シミュレーション終了まで，1 秒ごとに繰り返されるものである．まず，シミュレーション時刻，全車両のプローブ情報，信号情報が AVENUE から信号制御 FW へ送信される（各項目の詳細は**表 4.4**）．プローブ車のプローブ情報は，4.2.2 で説明した方法により，抽出する．これらを csv ファイルへ累積する．ファイル名は，プローブ車のプローブ情報をまとめたものが，「probe_data.csv」，信号情報をまとめたものが「signal_data.csv」である．

表 4.4　プローブ情報と信号情報の詳細

プローブ情報	信号情報
車両ID	信号 ID
車種番号	現在の制御パターン ID
起点ゾーン ID	現在の階梯番号
終点ゾーン ID	現在のオフセット値
位置情報数	オフセットの基準信号 ID
時刻	現在の制御パターンで設定されている全階梯数:
現在位置:X	現在の制御パターンで設定されている各階梯秒数
現在位置:Y	

現在位置:Z	
現在位置:リンク ID	
現在位置:レーン番号、トラック番号	
現在位置:停止線からの距離	
速度	

次にシミュレーション経過秒数が 60 で割り切れる場合，つまり，1 分おきに，プローブエミュレータへシミュレーション時刻を引き渡す．プローブエミュレータでは，出力した csv ファイルを読み込み，シミュレーション時刻から直近 1 分間のプローブ情報に基づいて，直近 1 分間平均遅れ時間と直近 1 分間流出台数を算出し，csv ファイルへ累積する．このファイル名は「TrafficConditionData.csv」である．次に，このファイルを読み込み，1 分間平均遅れ時間と 1 分間流出台数を，直近 30 分間分だけ抽出し，その平均を取り，平均遅れ時間と平均流出台数を算出する．また，「signal_data.csv」を読み取り，シミュレーション時刻における，各階梯秒数，サイクル経過秒数，オフセットを算出する．平均遅れ時間・平均流出台数・各階梯秒数・サイクル経過秒数・オフセットを流入数分，交差点数分だけ，算出したのちに，AI の入力データセット形式へ整理して新たに csv ファイルへ累積する．このファイル名は「Input_data.csv」である．ここまでがプローブエミュレータで行われる作業である

表 4.5 は，今回のネットワークにおける AI 入力データセットの全 19 変数名をまとめたものである．また，p.32 の**図 4.5** にプローブエミュレータまでのフローをまとめた．

表 4.5　AI 入力データセットの変数名

交差点名	入力変数
東京医療センター前	➢ 平均遅れ時間（秒） ➢ 直進車平均流出台数（台） ➢ 右折車平均流出台数（台） ➢ 左折車平均流出台数（台） ➢ 主方向青時間（秒） ➢ 右折矢印時間（秒） ➢ 従方向青時間（秒） ➢ サイクル経過秒数（秒）
東京医療センター正門前	➢ 平均遅れ時間（秒） ➢ 直進車平均流出台数（台） ➢ 主方向青時間（秒） ➢ 従方向時間（秒） ➢ 「東京医療センター前」とのオフセット
東根小学校南	➢ 平均遅れ時間（秒）

	➢ 直進車平均流出台数（台） ➢ 右折車平均流出台数（台） ➢ 主方向青時間（秒） ➢ 従方向時間（秒） ➢ 「東京医療センター正門前」とのオフセット

4.3.3　提案信号制御手法以降の流れ

ここでは提案信号制御手法以降の流れを，順を追って詳細に説明していく．まず，プローブエミュレータで作成されたAI入力データセット「Input_data.csv」を読み込む．続いて，探索の初期値x_0を求めるための信号制御パラメータを決める．具体的には，10サイクル分の主方向青時間スプリット・重要交差点サイクル長・重要交差点サイクル長との差である．**表4.6**は今回のネットワークにおいて，探索変数となる信号制御パラメータをまとめたものである．

表4.6　探索変数となる信号制御パラメータ

交差点名	信号制御パラメータ （いずれも10サイクル分）
東京医療センター前	➢ サイクル長 ➢ 主方向青時間スプリット ➢ 右折矢印時間スプリット
東京医療センター正門前	➢ 「東京医療センター前」とのサイクル長の差 ➢ 主方向青時間スプリット
東根小学校南	➢ 「東京医療センター前」とのサイクル長の差 ➢ 主方向青時間スプリット

これを，式$(\boldsymbol{d})$の逆関数である式$(\boldsymbol{e})$により，変数$x(-\infty,\infty)$へ変換し，初期値x_0を求める．

$$\boldsymbol{x} = \log_e\left(\frac{min - MAX}{P - MAX} - 1\right) \qquad (\boldsymbol{e})$$

次に，現在実行しているサイクル長の残りの秒数を取得する．この秒数または30秒のいずれか小さい方を探索の制限時間として，収束したかどうかに関わらず，制限時間が来たら，その時点で探索を打ち切り，それまでの探索結果の中で，最良の結果を最適信号制御パラメータとする．これは3.3.6で説明したように，前サイクルが終了した後に，最適信号制御パラメータを実行しなければいけないためである．また，AVENUEとの通信が何もないまま30秒間経過するとタイムアウトにより接続が切れてしまうため，最長でも30秒という制限を設けた．

初期値x_0が用意出来たら，続いて探索のフェーズに入る．探索アルゴリズムは，3.3.4 で挙げたものの中から 1 つ選択する．そして，制限時間内で探索を行い，予測総遅れ時間が最小となるような最適な変数x_{opt}を決定する．

探索終了後は，式$(\boldsymbol{d})$にx_{opt}を代入することで，将来 10 サイクル分の信号制御パラメータを算出し，その後に将来 12 分間の最適な各階梯秒数を算出する．これを 12 分間階梯情報として，信号制御 FW を介して，AVENUE へ送信し，階梯秒数の変更要求をする．こうして次のサイクルから，最適化された階梯秒数を開始する．P.33 の**図 4.6** に提案信号制御手法以降のフローを示す．

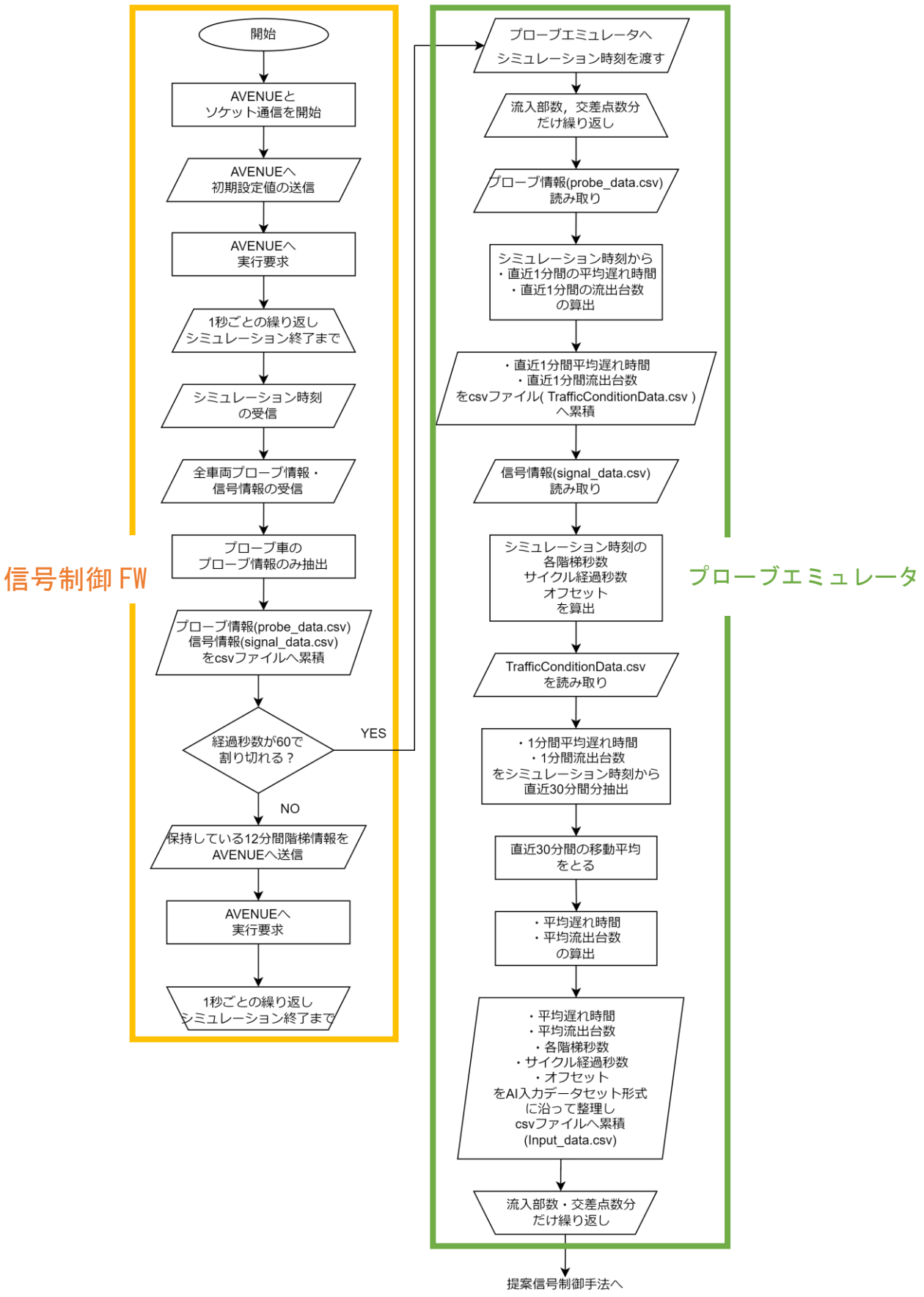

図 4.5　プローブエミュレータまでのフロー

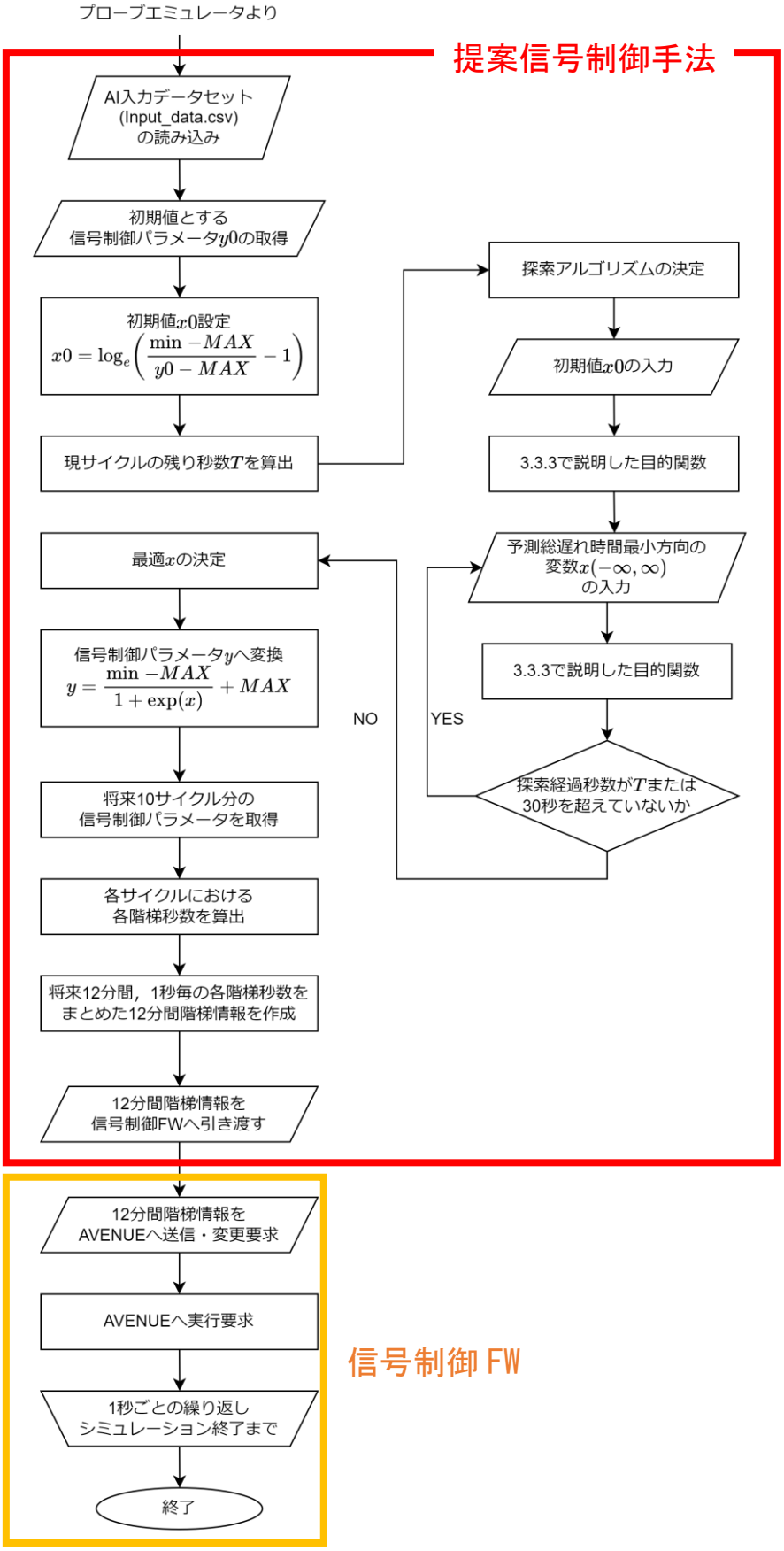

図 4.6　提案信号制御手法以降のフロー

4.4　学習結果

AI の予測精度について，損失関数である平均二乗誤差と，評価関数である絶対平均誤差を指標として見てみる．損失関数とは，「正解値」と，AI による出力された「予測値」とのズレの大きさ（これを損失値と呼ぶ）を出力する関数である．学習に当たっては，損失関数を目的関数として，損失値が最小となるように重みを探索する．評価関数とは，最適化された重みをもった AI の予測性能を測るために使われる関数である．今回の場合，予測する値はスカラーであるため，評価関数には絶対平均誤差を使う．学習時にはドロップアウトを用いる．これは，学習時にノードをランダムに不活性にする手法のことであり，これにより，過学習をある程度抑制することができる．学習データを収集するにあたり，AVENUE の設定条件を**表 4.7** にまとめた．

表 4.7　AVENUE の設定条件

ネットワーク構成	駒沢通り(4.2.1 参照)
10 分間平均発生交通量（台） （平均に対し，ポアソン分布を仮定した 確率分布で 10 分おきに変動）	107（混雑時交通量）
車種構成	小型車のみ
プローブ車混入率（%）	30
希望速度（km/h）	40（規制速度）
従方向の青時間スプリット	制約なし
シミュレーション時間	6:50～9:30（データ収集は 7:30～8:29）
発生交通量のシード値	乱数
車両発生間隔のシード値	1000

その結果，9000 個の学習データを収集することができた．9000 個あるデータのうち，7200 個を訓練データ，1800 個を検証データとし，エポック数(学習する回数)を 200 回，バッチサイズ（1 エポックあたりのデータ個数）を 100 個とした．7200 個のうち 100 個を選ぶ選び方はランダムとする．学習の結果を，次ページの**図 4.1**，**図 4.2** に示す．

図 4.1 は損失値の推移をエポック数ごとに示したものである．これを見ると，エポック数がおよそ 150 までは，エポック数が増えるにつれて訓練データ・検証データ共に損失値は単調に減少している．しかし，それ以降は，ほぼ横ばいになっており，これ以上エポック数を増やしても，損失値は小さくならないと考えられる．また，検証データにおいては，損失値の増加が見られないことから，過学習は起きてないと言える．

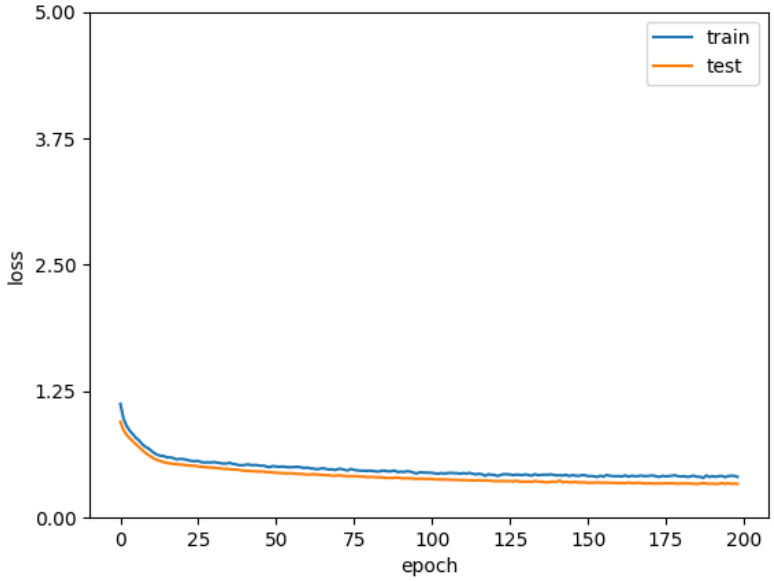

図 4.1　損失値の推移

図 4.2 は，評価関数である絶対平均誤差の推移を，エポック数ごとに示したものである．これを見ると，エポック数が増えるごとに，訓練・検証データ共に，絶対平均誤差が単調に減少するという損失と同様の傾向が見られた．さらに検証データは，絶対平均誤差の増加が見られないことから，過学習は起きていないと言える．以降，提案信号制御手法ではこれらの学習済み AI を用いる．

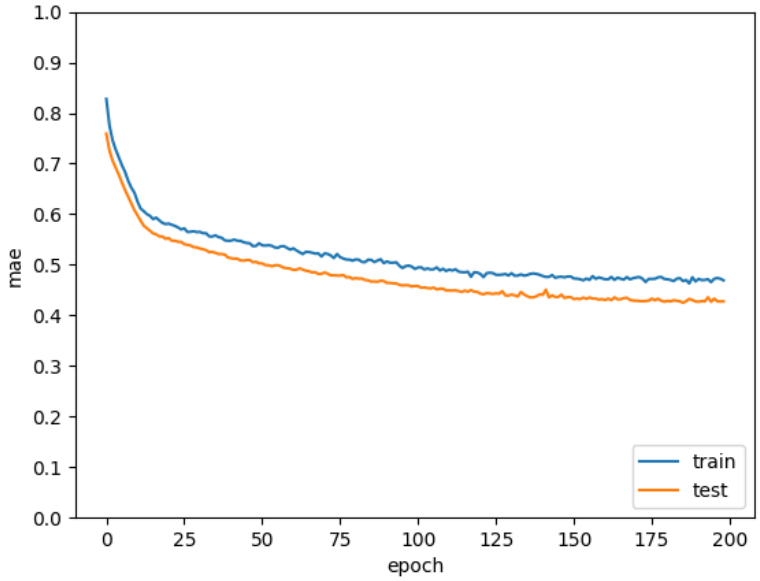

図 4.2　評価関数の出力推移

4.5　制御効果の評価指標

総遅れ時間，平均渋滞長，平均旅行速度を制御効果の評価指標とする．総遅れ時間の算出方法は 3.2.5 の式(c)と同様である．ただし，計測時間は制御開始からシミュレーション終了までの 56 分間である．次に平均渋滞長の算出方法を説明する．車線をブロックに分け，ブロック内の速度が 10 km/h 未満である場合に，渋滞と判断し，渋滞長の一部としてブロック長を累積することで渋滞長を求め（**図 4.3**），これを単位時間あたりで平均することで平均渋滞長を算出する．平均旅行時間は，リンク内における単位時間あたりの各車両の旅行時間を平均することにより算出する．

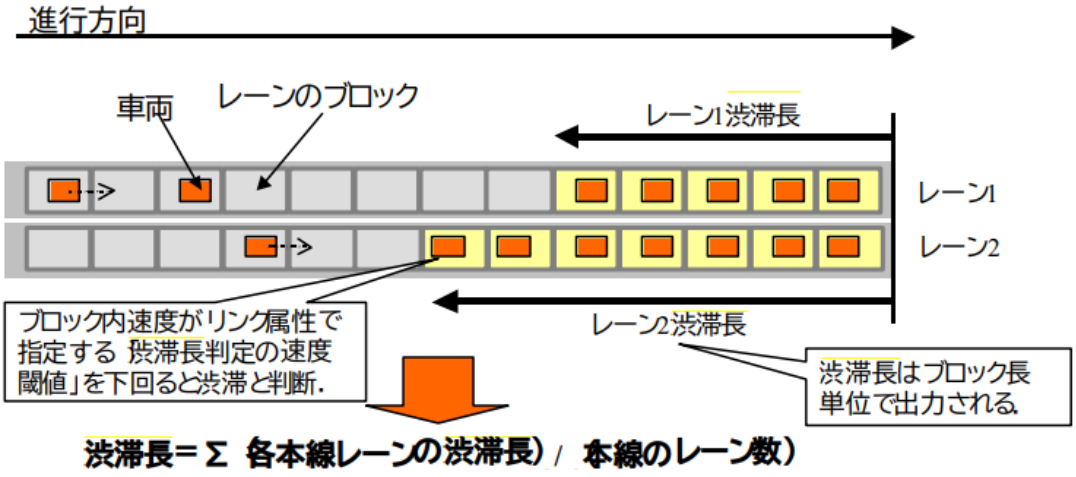

図 4.3　渋滞長の算出方法

4.6　従来信号制御手法による信号制御パラメータ算出

提案信号制御手法の有用性を確かめるための比較対象として，従来信号制御手法による制御効果を確かめる必要がある．従来信号制御手法として採用するのは，1.1.2 で説明した，プログラム多段制御とする．

従来信号制御手法による信号制御パラメータの算出方法を示す．ここでは，発生交通量が平均 107（台/10min）の場合を考える．まずは各流入方向の需要率を求める．需要率とは，設計交通量を飽和交通流率で除したものである（式(f)）．

$$\lambda = \frac{q}{s} \qquad (f)$$

λ： 需要率

q ： 設計交通量[台/時間]

s ： 飽和交通流率[台/青 1 時間]

発生交通量が 107（台/10min）である場合，1 時間あたりに換算すると，642（台/ｈ）となる．これを 5 つの OD に均等に分けると，1 つの OD 交通量は 128（台/ｈ）となる．ゆえに，各交差点の設計交通量は**図 4.4** のようになる．

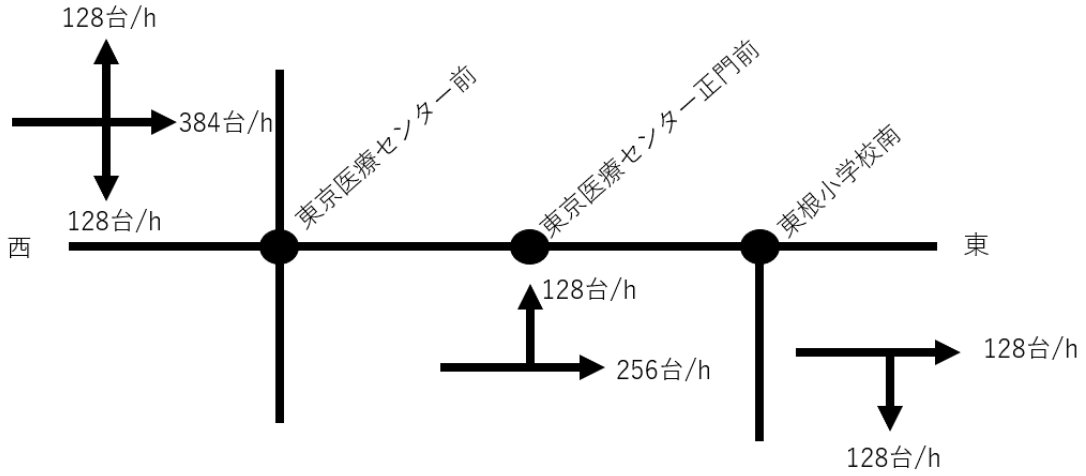

図 4.4　各交差点の設計交通量

「東京医療センター前」の直進方向の需要率は，q =384，s=2000 より 384/2000 = 0.192 となる．左折・右折方向需要率はそれぞれ，q =128，s=1800 より 128/1800 = 0.071 となる．よって，「東京医療センター前」主方向の需要率は 0.192 となる．今回のケースでは従方向の青時間スプリットを 0.45 確保する．よって，従方向の需要率は，0.215 となる．したがって「東京医療センター前」交差点の需要率は，0.192＋0.071＋0.215＝0.478 となる．この交差点は重要交差点であるため，ここのサイクル長を全交差点において共通のものとする．サイクル長は Webster の実験式（式(g)）にしたがって算出する．

$$C = \frac{1.5 * L + 5}{1 - \lambda} \qquad (g)$$

C: サイクル長[秒]
L ： 損失時間[秒]
λ ： 交差点の需要率

L = 13，λ = 0.478 より，(1.5*13+5)/(1-0.478) = 47（秒）が共通サイクル長となる．次に青時間を求める．主方向の青時間は，(47-13)* 0.192/0.478 = 14（秒）．右折矢印は(47-13)* 0.071/0.478 = 5（秒）．従方向は，(47-13)* 0.215/0.478 = 15（秒）

「東京医療センター正門前」の主方向の需要率は，q =256，s=2000 より 256/2000 = 0.128 となる．また，同様に従方向の青時間スプリットを 0.45 確保する．よって，従方向の需要率は，0.105 となる．以上より，「東京医療センター正門前」交差点の需要率は，0.128＋0.105＝0.233 となる．サイクル長は「東京医療センター前」と共通より，47（秒）．主方向の青時間は，(47-3)* 0.128/0.233 = 24（秒）．従方向は(47-3)* 0.105/0.233 = 20（秒）．

「東根小学校南」の主方向は右直混用車線である．そのため，飽和交通流率の基本値に対し

て補正率を乗じる必要がある．まず，左折車混入率は(128/256)*100=50%である．また，右折車の直進車換算係数は 3.95 とする．したがって右折車混入による飽和交通流率の補正率は，100/(50+50*3.95)=0.404 となる．よって飽和交通流率は 2000*0.404=808 となるため，主方向の需要率は，128/808=0.158 となる．また，同様に従方向の青時間スプリットを 0.45 確保するため，従方向の需要率は，0.129 となる．以上より，「東根小学校南」交差点の需要率は，0.158＋0.129＝0.287 となる．サイクル長は「東京医療センター前」と共通より，47（秒）．主方向の青時間は，(47-10)* 0.158/0.287 = 20（秒）．従方向は(47-10)* 0.129/0.287 = 17（秒）．

4.7　提案信号制御の制御効果

AVENUE を用いて，以下の様々なケースにおける，提案信号制御手法による制御効果を定量的に示す．その上で，プローブ情報を信号制御に活用する際の課題や要件について，考察を行う．

- 異なる探索アルゴリズムによる制御効果の違い
- 異なる初期値による制御効果の違い
- 異なる交通需要による制御効果の違い
- 異なるプローブ車混入率による制御効果の違い
- 従来信号制御手法と提案信号制御手法における制御効果の違い
- 制御効果と信号制御パラメータの推移

4.7.1　異なる探索アルゴリズムによる制御効果の違い

3.3.4 で説明した各探索アルゴリズムにおける，制御効果の違いを見る．諸設定条件は**表 4.8** の通りである．

表 4.8　諸設定条件

ネットワーク構成	駒沢通り(4.2.1 参照)
10 分間平均発生交通量（台） （平均に対し，ポアソン分布を仮定した 確率分布で 10 分おきに変動）	107（混雑時交通量）
車種構成	小型車のみ
プローブ車混入率（%）	30
希望速度（km/h）	40（規制速度）
従方向の青時間スプリット	最低でも 0.4 を確保
シミュレーション時間	6:50～9:30（制御開始は 8:30～）
発生交通量のシード値	100 から開始し，サンプル数に応じて 1 刻みで増やしていく
車両発生間隔のシード値	1000
探索アルゴリズム	Nelder-Mead, Powell, CG, BFGS, SLSQP, COBYLA, DE, SA
探索初期値	従来信号制御手法による 信号制御パラメータが 10 サイクル分継続

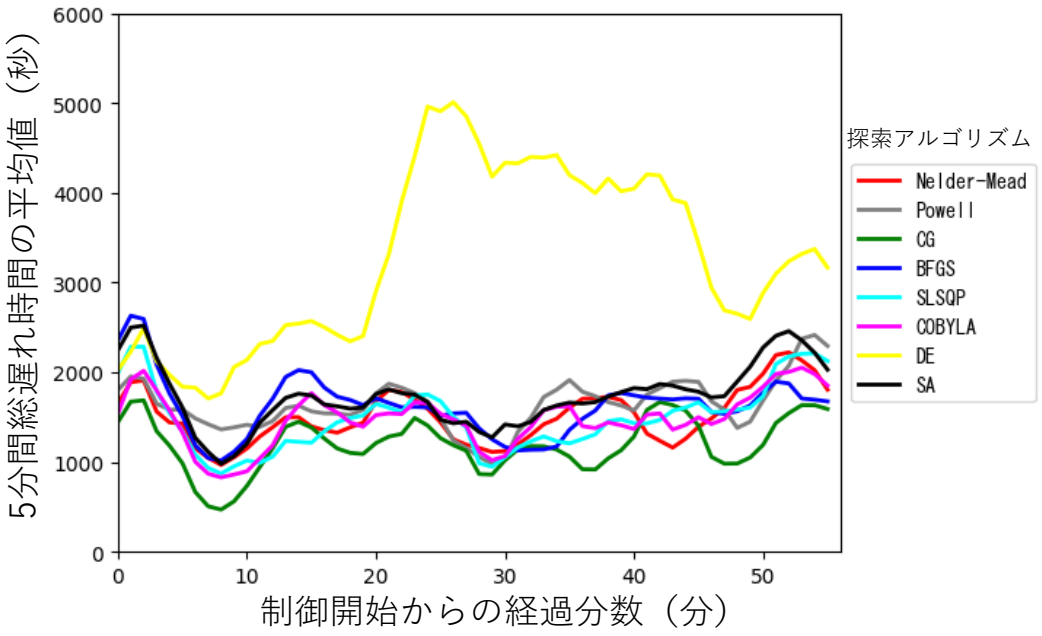

図 4.5　各探索アルゴリズムに対する 5 分間総遅れ時間の推移

図 4.5 は，各探索アルゴリズムに対する 5 分間総遅れ時間の平均値の推移を，制御開始後からシミュレーション終了まで表したものである．これを見ると，全時点を通して，DE 以外において，CG が 5 分間遅れ時間が小さい傾向が見られ，SA が大きい傾向が見られた．その差は，700 秒近くであり，探索アルゴリズムによって，時点毎の 5 分間総遅れ時間の大小の傾向があることが認められる．全体を通して，大域解探索である DE と SA が 5 分間総遅れ時間が大きくなっている．また，DE 以外のどの探索アルゴリズムにおいても，概ね同じような変動をたどって推移していることが分かる．よって，DE 以外は，探索アルゴリズムの違いによる，5 分間総遅れ時間の変動に影響はあまり無いと言える．一方，DE は，制御開始から 20 分後～45 分後の間において，他のアルゴリズムと比較して，時点毎の 5 分間総遅れ時間が大きくなっている．したがって，DE を用いることは得策でないと言える．また，DE を除いたどの探索アルゴリズムを用いても，5 分間総遅れ時間の推移はほぼ横ばいである．これは，最小の予測総遅れ時間を十分に探索ができていない可能性が考えられる．このことに関しては，4.8.5 に考察を行っているので参照されたい．以上より，探索アルゴリズムによって，平均 5 分間総遅れ時間の大小の傾向はあると言える．DE や SA といった大域解探索アルゴリズムを用いた場合は，平均 5 分間総遅れ時間が大きくなる傾向があると言える．一方で，CG は全探索アルゴリズムの中で，平均 5 分間総遅れ時間が最も小さいと言える．

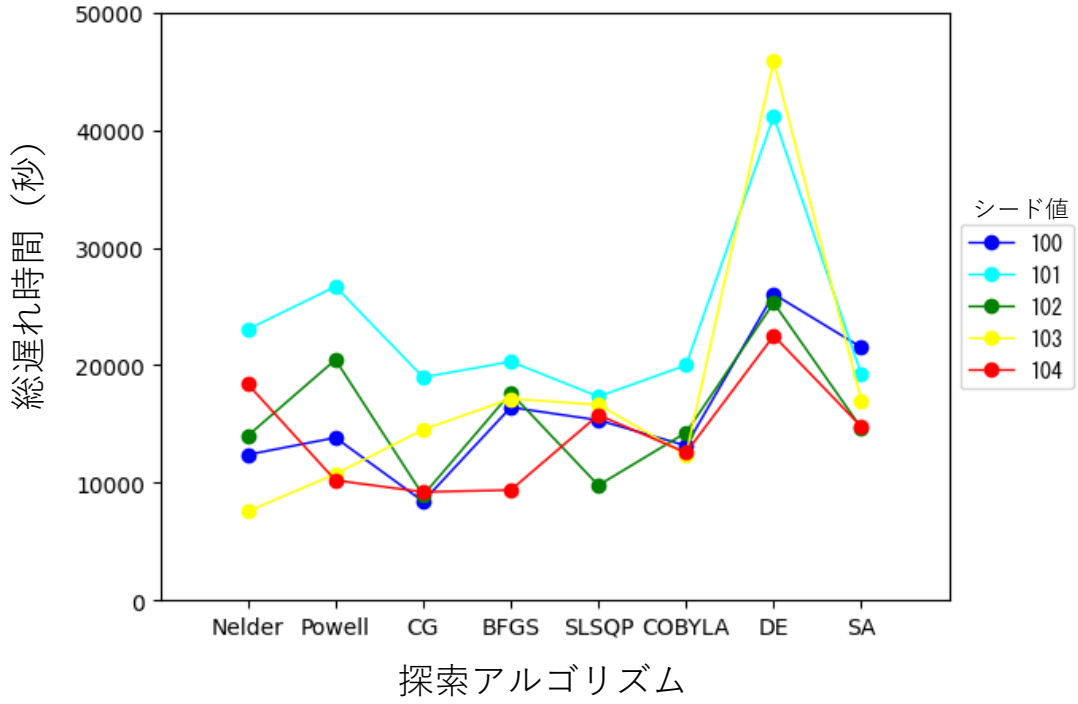

図 4.6　各探索アルゴリズムに対する総遅れ時間の分布

図 4.6 は，各探索アルゴリズムにおける，制御開始からシミュレーション終了までの総遅れ時間を，シード値毎に算出したものである．これを見ると，同じシード値であっても，探索アルゴリズムによって総遅れ時間に違いが生じている．中でも CG は，他のアルゴリズムと比較して，総遅れ時間が小さい傾向が見られる．一方で DE は，大きい傾向が見られる．したがって探索アルゴリズムを選定する際は，様々な探索アルゴリズムに対する制御効果を比較した上で選定することが重要であり，本研究で用意したものの中では，CG を選ぶことが適切であると言える．

次ページの**表 4.8** は，Tukey 法を使って，異なる探索アルゴリズムに対する総遅れ時間の平均の差の検定を行った結果である．DE 以外における，探索アルゴリズム同士には，有意な差が無いことが分かる．一方 DE は，他のどの探索アルゴリズムと比較しても，総遅れ時間の平均値が大きくなり，その差は有意であった．特に，CG・COBYLA・SLSQP とは 0.1%に有意であり，明確に差があると言える．したがって統計的には，DE 以外のどの探索アルゴリズムを用いたとしても，制御効果に影響がないことは否定できない．一方で，DE を用いた場合，他の探索アルゴリズムと比べて，制御効果は悪くなり，DE を用いることは避けるのが望ましいと言える．

異なる探索アルゴリズムに対する制御効果の違いについてまとめると，以下のようになる．

- 探索アルゴリズムによって，制御効果に違いが大きく出るものがある
- 大域解探索では，局所解探索と比較して制御効果が悪くなる傾向がある
- どの探索アルゴリズムを使っても，総遅れ時間の推移に影響はあまりなく，横ばいの傾向となる

表 4.8　各探索アルゴリズムにおける総遅れ時間の平均の差の検定結果

探索アルゴリズム	平均値の差	95%信頼区間の下限	95%信頼区間の上限	p値	
CG-BFGS	-4167.522028	-15893.45113	7558.407074	0.939690443	
COBYLA-BFGS	-1703.907442	-13429.83654	10022.02166	0.999719192	
DA-BFGS	1246.437123	-10479.49198	12972.36623	0.999965575	
DE-BFGS	16002.35318	4276.424075	27728.28228	0.002399755	**
Nelder-BFGS	-1099.951101	-12825.8802	10625.978	0.999985318	
Powell-BFGS	210.0937243	-11515.83538	11936.02283	1	
SLSQP-BFGS	-1211.52647	-12937.45557	10514.40263	0.999971621	
COBYLA-CG	2463.614585	-9262.314516	14189.54369	0.9969798	
DA-CG	5413.959151	-6311.969951	17139.88825	0.804044199	
DE-CG	20169.8752	8443.946102	31895.80431	9.35E-05	***
Nelder-CG	3067.570926	-8658.358175	14793.50003	0.988642556	
Powell-CG	4377.615752	-7348.31335	16103.54485	0.923203196	
SLSQP-CG	2955.995558	-8769.933544	14681.92466	0.990863396	
DA-COBYLA	2950.344565	-8775.584536	14676.27367	0.990966181	
DE-COBYLA	17706.26062	5980.331517	29432.18972	0.000646724	***
Nelder-COBYLA	603.956341	-11121.97276	12329.88544	0.999999766	
Powell-COBYLA	1914.001166	-9811.927935	13639.93027	0.999397157	
SLSQP-COBYLA	492.3809722	-11233.54813	12218.31007	0.999999944	
DE-DA	14755.91605	3029.986952	26481.84516	0.00609822	**
Nelder-DA	-2346.388224	-14072.31733	9379.540877	0.997774005	
Powell-DA	-1036.343399	-12762.2725	10689.5857	0.999990236	
SLSQP-DA	-2457.963593	-14183.89269	9267.965509	0.997022672	
Nelder-DE	-17102.30428	-28828.23338	-5376.375176	0.001033135	**
Powell-DE	-15792.25945	-27518.18855	-4066.33035	0.002813567	**
SLSQP-DE	-17213.87965	-28939.80875	-5487.950545	0.000947735	***
Powell-Nelder	1310.044825	-10415.88428	13035.97393	0.999951753	
SLSQP-Nelder	-111.5753688	-11837.50447	11614.35373	1	
SLSQP-Powell	-1421.620194	-13147.5493	10304.30891	0.999916221	
			*** p < 0.001, ** p < 0.01, * p < 0.05		

4.7.2　異なる初期値による制御効果の違い

3.3.5 で説明した 2 種類の探索初期値によって，制御効果に違いがあるかどうかを確かめる．ここで簡単のため，従来信号制御手法に基づいて算出された信号制御パラメータが将来 12 サイクル分継続した場合を「従来パラメータ」，前のサイクルで適用していた信号制御パラメータが将来 12 サイクル分継続した場合を「前サイクルパラメータ」と呼ぶことにする．諸設定条件は**表 4.9** の通りである．

表 4.9　諸設定条件

ネットワーク構成	駒沢通り(4.2.1 参照)
10 分間平均発生交通量（台） (平均に対し，ポアソン分布を仮定した 確率分布で 10 分おきに変動)	107（混雑時交通量)
車種構成	小型車のみ
プローブ車混入率（%)	30
希望速度（km/h)	40（規制速度)
従方向の青時間スプリット	最低でも 0.4 を確保
シミュレーション時間	6:50～9:30（制御開始は 8:30～)
発生交通量のシード値	100 から開始し，サンプル数に応じて 1 刻みで増やしていく
車両発生間隔のシード値	1000
探索アルゴリズム	Nelder-Mead
探索初期値	従来信号制御手法による 信号制御パラメータが 10 サイクル分継続 (従来パラメータと呼ぶ)
	前サイクルの 信号制御パラメータが 10 サイクル分継続 (前サイクルパラメータと呼ぶ)

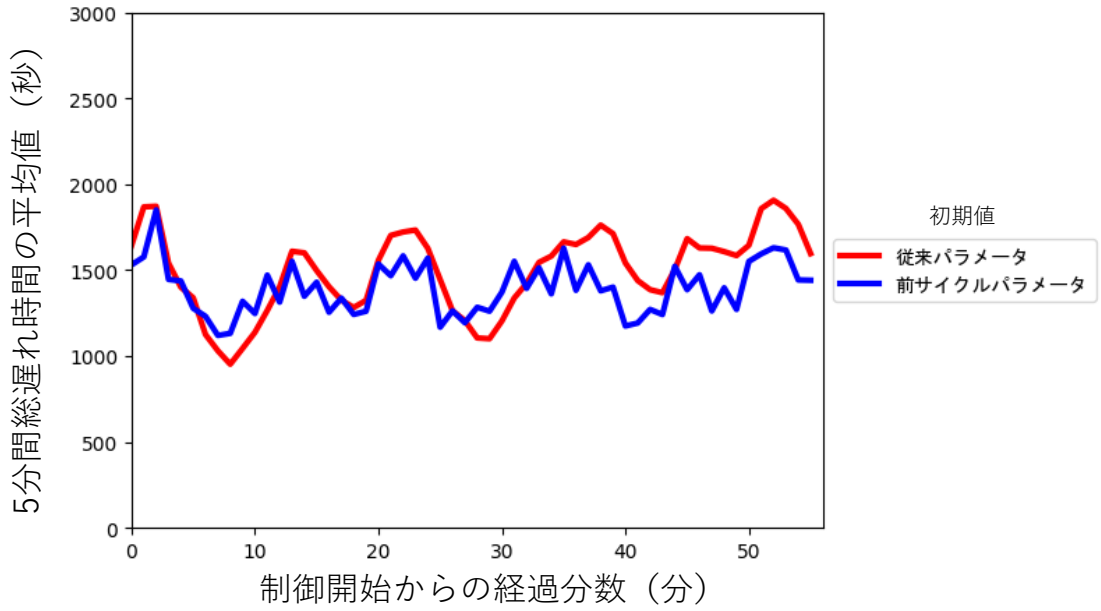

図 4.7　各初期値に対する 5 分間総遅れ時間の推移

図 4.7 は各初期値における，5 分間総遅れ時間の推移を，制御開始からシミュレーション終了まで表したものである．これを見ると，前サイクルパラメータに比べて，従来パラメータの方が，変動が大きい．前サイクルパラメータでは変動幅がおおよそ 500 秒以内であるが，従来パラメータでは，おおよそ 800 秒である．したがって，従来パラメータの方が，5 分間総遅れ時間が小さくなる場合もあるが，大きくなる場合もある．前サイクルパラメータの方は，微小な変動が多い．これは，従来パラメータの場合と比べて，前サイクルパラメータの方が，サイクル長が短く，1 分ごとに信号待ち→捌け完了→信号待ち→・・・と短周期で繰り返していることが原因であると考えられる．また大きな傾向として，両初期値とも，5 分間総遅れ時間は横ばいになっており，渋滞が緩和しているといった傾向は確認できない．これは，4.8.1 で述べたことと同様に，十分に最小の予測総遅れ時間を探索しきれていないことが原因であると考えられる．

以上より，初期値の違いは，総遅れ時間の推移に影響を与えると言える．従来パラメータを用いた場合は，前サイクルパラメータよりも，5 分間総遅れ時間が小さくなる場合もある一方，大きくなる場合もあり，変動が大きい．

次のページの**図 4.8** は，発生交通量のシード値を揃えた時の，各初期値に対する総遅れ時間の差の分布を，箱ひげ図で表したものである．差の求め方は，以下の通りである．

（シード値 X における総遅れ時間の差）＝
（シード値 X で初期値が従来パラメータの総遅れ時間）－
（シード値 X で初期値が前サイクルパラメータの総遅れ時間）

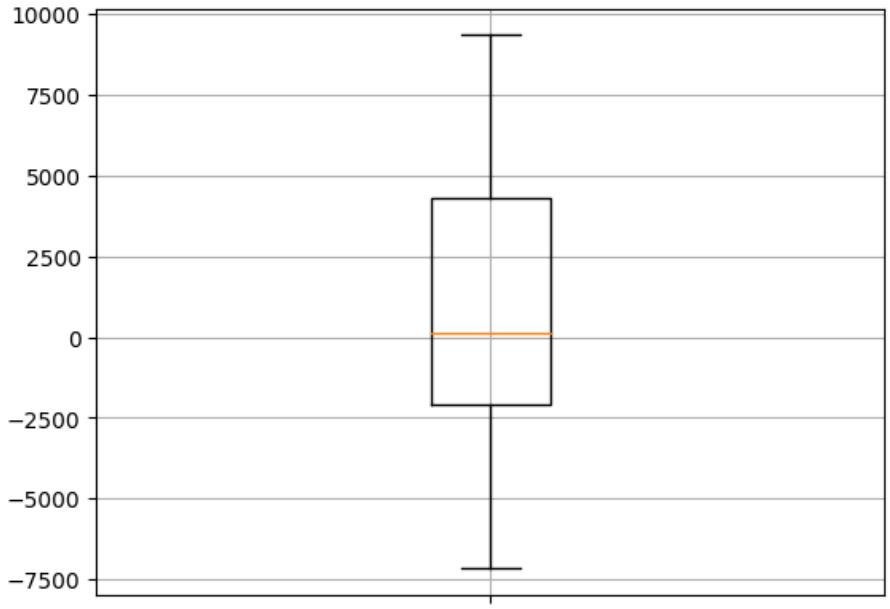

図 4.8　同シード値における総遅れ時間の差の分布

図 4.8 を見ると，同じ交通量でも，初期値によって総遅れ時間が 0〜4000 秒ほどの違いが生じることが分かる．また従来パラメータを初期値にした場合の方が，総遅れ時間が大きい傾向が見られる．したがって同じ発生交通量でも，初期値によって探索結果が変わることから総遅れ時間にも違いが表れると言うことができ，初期値に従来パラメータを用いた場合は，前サイクルパラメータを用いた場合よりも総遅れ時間が大きくなる可能性が高いと言える．次に，対応のある t 検定により，この差が有意なものなのかどうかを確かめた．検定の結果，有意な差は見られなかった．（t = 0.564，df = 18，p = 0.586）したがって，統計的には，初期値の違いによる，総遅れ時間への影響があるとは言い切れない

次のページの**図 4.8** は，制御開始からシミュレーション終了まで，各リンク渋滞長を全て足し合わせた総渋滞長の推移を表したものである．これを見ると，前サイクルパラメータの方が，総渋滞長が大きくなる場合も一部あるが，従来パラメータの方が，全体を通して総渋滞長が若干大きい傾向が見られ，特に後半 30 分は，差が比較的大きくなっている．また，前サイクルパラメータの方が，微小な変動をしているのは，総遅れ時間と同様に，サイクル長が短いことが原因であると考えられる．以上より，初期値の違いは，総渋滞長に多少の影響を及ぼすと言え，前サイクルパラメータを用いた方が，総渋滞長は小さくなる傾向がある．

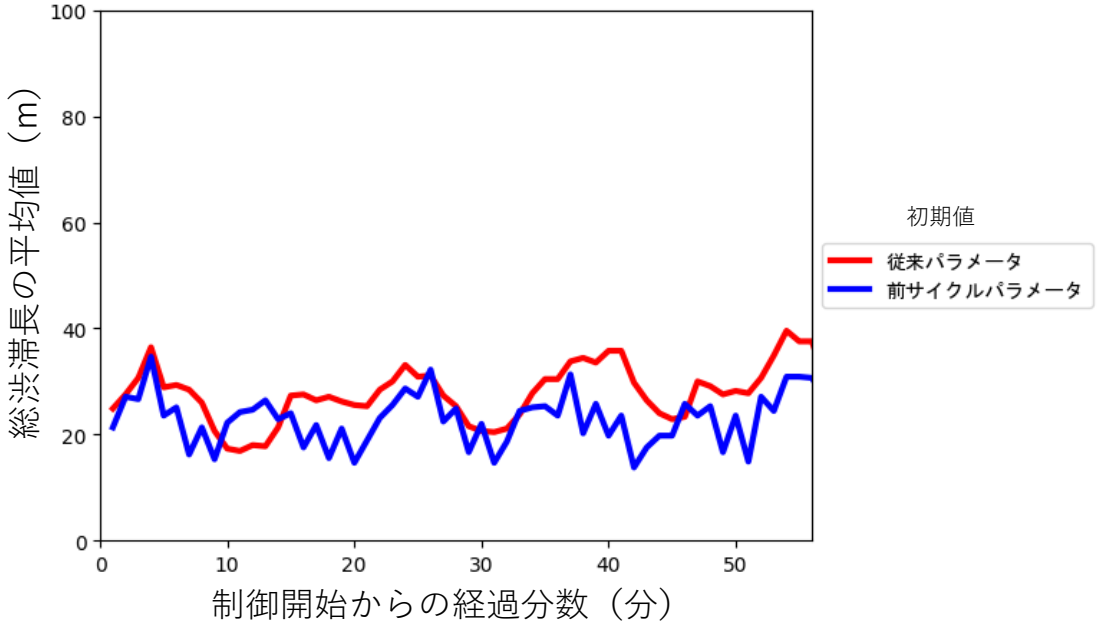

図 4.8　各初期値に対する総渋滞長の平均値の推移

図 4.9 は，発生交通量のシード値を揃えた時の，各初期値に対する最大渋滞長の差の分布について，交差点ごとに箱ひげ図で表したものである．主方向の需要率は，「東京医療センター前」，「東京医療センター正門前」，「東根小学校南」の順に大きくなっている．差の求め方は，以下の通りである．

（シード値 X における最大渋滞長の差）＝
（シード値 X で初期値が従来パラメータの最大渋滞長）－
（シード値 X で初期値が前サイクルパラメータの最大渋滞長）

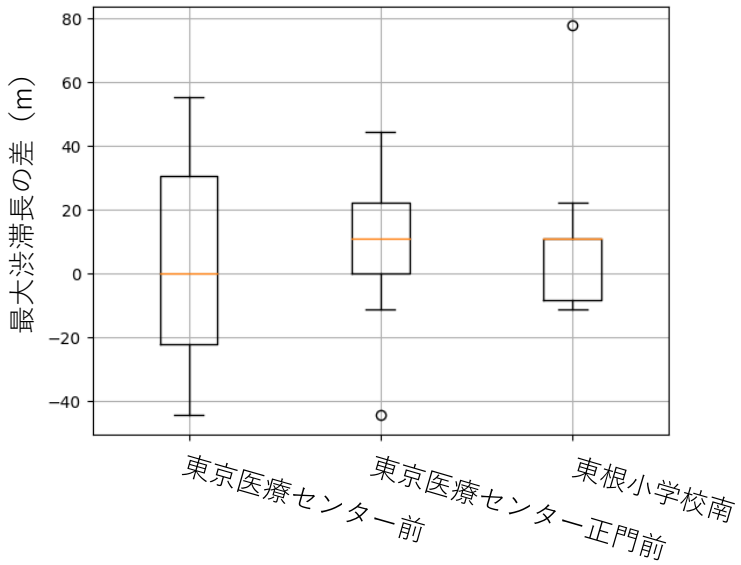

図 4.9　同シード値における最大渋滞長の差の分布

図 4.9 を見ると，「東京医療センター前」では，－20 から 30 までに集中していて，比較的大きな差が生じており，従来パラメータの方が，最大渋滞長が若干大きな傾向がある．「東京医療センター正門前」では，0 から 20 の間に差が集中しており，従来パラメータの方が，最大渋滞長が大きい傾向がある．「東根小学校南」では，差が－10 から 10 の間に集中しており，初期値による違いはあまりないと言える．以上より，需要率が高い交差点ほど，初期値による最大渋滞長のばらつきが大きく，また，従来パラメータを用いた場合，最大渋滞長は大きくなる傾向があると言える．

次に，対応のある t 検定により，この差が有意なものなのかどうかを確かめた．検定の結果，全交差点において有意な差は見られなかった．（「東京医療センター前」：t = 0.101，df = 18，p = 0.922，「東京医療センター正門前」：t = 1.124，df = 18，p = 0.290，「東根小学校南：t = 1.342，df = 18，p = 0.213」）したがって，統計的には，初期値の違いによる最大渋滞長への影響があるとは言い切れない．

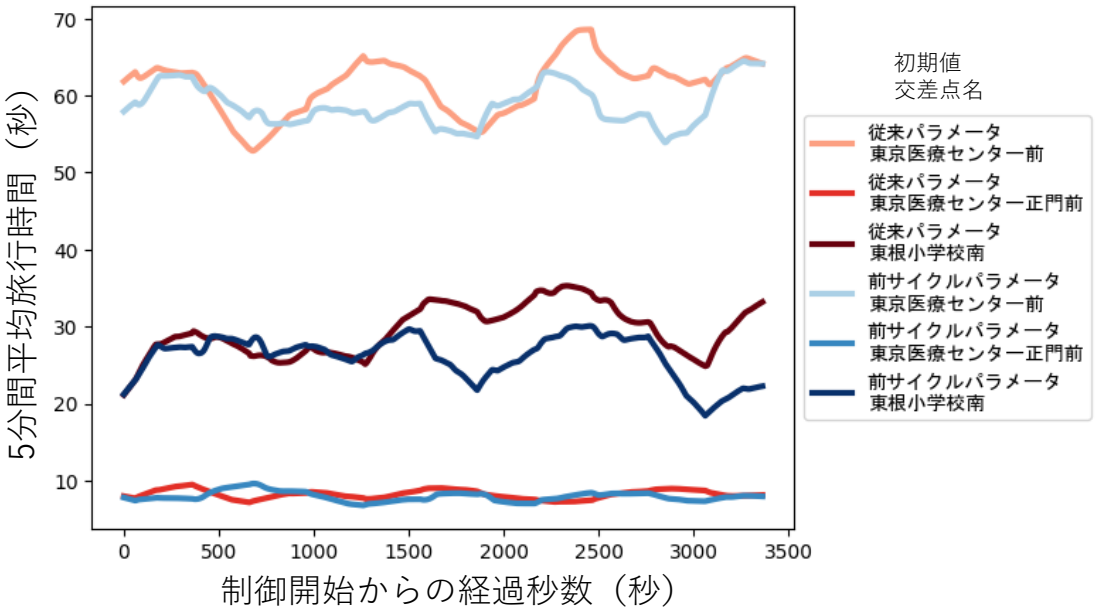

図 4.10　5 分間平均旅行時間の推移

図 4.10 は制御開始からシミュレーション終了までの，各初期値における，5 分間平均旅行時間の推移を表したものである．これを見ると，「東京医療センター前」では，従来パラメータの方が，変動が大きいことが分かる．そのため，前サイクルパラメータより，小さくなる場合もあるが，大きくなる場合もある．一方，前サイクルパラメータは，従来パラメータほど変動は大きくなく，安定している．どちらにおいても，全体として見ると横ばいとなっている．「東根小学校南」では，制御開始から 1500 秒まで，時点毎の平均 5 分間旅行時間や推移に違いはほとんど見られないが，それ以降は，従来パラメータの方が，時点毎の平均 5 分間旅行時間が大きくなるという傾向が見られた．また，前サイクルパラメータの方は，1500 秒以降，若干の減少傾向が見られた．「東京医療センター正門前」では，全体を通して，両初期値ともほぼ同じ値で推移しており，違いはほとんどない．以上のことから，リンク長が長いほど，初期

値の違いによる 5 分間平均旅行時間の差が大きく表れ，前サイクルパラメータの方が，その値は小さくなる傾向がある．したがって，交差点間距離が長い場合，前サイクルパラメータを用いた方が，より良い制御効果を得られる可能性がある．

図 4.11 は，発生交通量のシード値を揃えた時の，各初期値に対する平均旅行時間の差の分布について，箱ひげ図で表したものである．差の求め方は，以下の通りである．

（シード値 X における平均旅行速度の差）＝
（シード値 X で初期値が従来パラメータの平均旅行時間）－
（シード値 X で初期値が前サイクルパラメータの平均旅行時間）

図 4.11 を見ると，「東京医療センター前」では，差が－3 から 7 に集中しており，比較的大きなばらつきとなった．さらに，従来パラメータの方が，平均旅行時間は大きい傾向が見られた．「東京医療センター正門前」では，差のばらつきは小さかった．しかし若干ではあるが，従来パラメータの方が，平均遅れ時間が大きい傾向が見られた．「東根小学校南」では，差が－2 から 5 までに集中しており，「東京医療センター前」に次いでばらつきが大きくなった．以上のことから，どの交差点においても，従来パラメータの方が，平均旅行時間は大きくなる傾向があると言える．また，交差点間距離が長いほど，両初期値の違いによる，平均旅行時間の差が大きくなる傾向があると言える．

次に，対応のある t 検定により，この差が有意なものなのかどうかを確かめた．検定の結果，全交差点で有意な差は見られなかった．（「東京医療センター前」：t = 1.003，df = 18，p = 0.342，「東京医療センター正門前」：t = 0.515，df = 18，p = 0.619，「東根小学校南：t = 1.750，df = 18，p = 0.114」）したがって，統計的には，初期値の違いによる平均旅行速度への影響があるとは言い切れない．

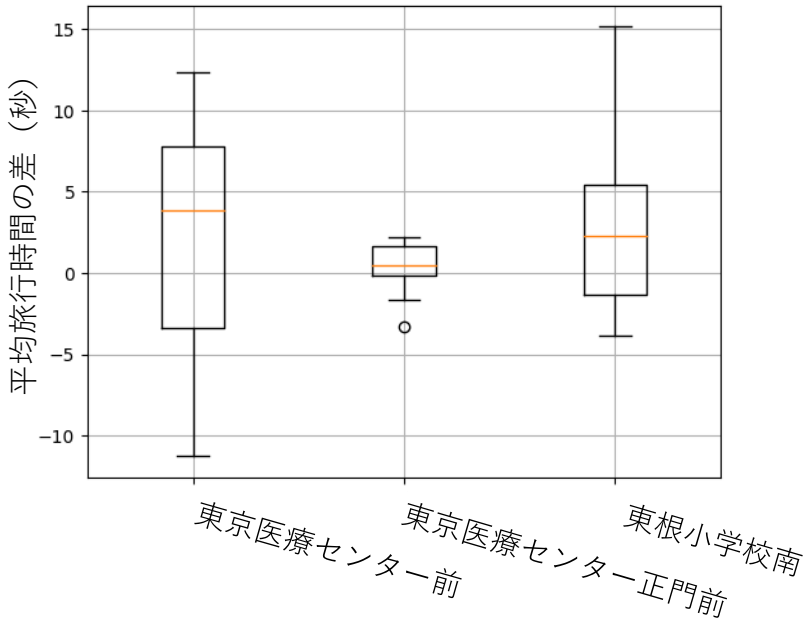

図 4.11　同シード値における平均旅行時間の差の分布

4.7.3　異なる交通需要による制御効果の違い

交通需要によって制御効果の違いがあるかどうかを確かめる．交通需要は，AVENUE における発生交通量を変えることで設定する．AVENUE の諸条件は**表 4.10** の通りである．

表 4.10　諸設定条件

ネットワーク構成	駒沢通り(4.2.1 参照)
10 分間平均発生交通量（台） （平均に対し，ポアソン分布を仮定した 確率分布で 10 分おきに変動）	53（閑散時交通量） 107（混雑時交通量） 214（超混雑時交通量）
車種構成	小型車のみ
プローブ車混入率（%）	30
希望速度（km/h）	40（規制速度）
従方向の青時間スプリット	最低でも 0.4 を確保
シミュレーション時間	6:50～9:30（制御開始は 8:30～）
発生交通量のシード値	100 から開始し，サンプル数に応じて 1 刻みで増やしていく
車両発生間隔のシード値	1000
探索アルゴリズム	Nelder-Mead
探索初期値	従来信号制御手法による 信号制御パラメータが 10 サイクル分継続

次のページの**図 4.12** はそれぞれの交通需要に対する，5 分間総遅れ時間の推移を表したものである．これを見ると，発生交通量が大きいほど，5 分間総遅れ時間の値も大きくなることが分かる．また，発生交通量が大きいほど，変動も大きくなることが分かる．発生交通量が 214 台の場合，ジグザグと微小な変動を繰り返している．これは，飽和状態に近いほど，1 回の青時間中の流出台数が多くなることから，1 サイクルという短い周期で，飽和→非飽和→飽和→・・・を繰り返すことが原因として考えられる．全発生交通量のケースにおいて，5 分間総遅れ時間の推移はほぼ横ばいとなっている．以上より，交通需要が多いほど，5 分間総遅れ時間は大きくなり，変動も大きくなる．裏を返せば，交通需要が多いほど，特定の条件下では，5 分間総遅れ時間を大幅に小さくできる可能性があると言える．

次のページの**図 4.13** は，発生交通量のシード値を揃えた時の，各交通需要に対する総遅れ時間の差について，箱ひげ図で示したものである．X 軸の数字は例えば，「53－214」の場合，発生交通量が 53 台の時と 214 台の時，それぞれの総遅れ時間の差の分布であることを示す．総遅れ時間の差の求め方は，以下の通りである．

（シード値 X における総遅れ時間の差）＝
（シード値 X で交通需要が Y の総遅れ時間）－
（シード値 X で交通需要が Z の総遅れ時間）

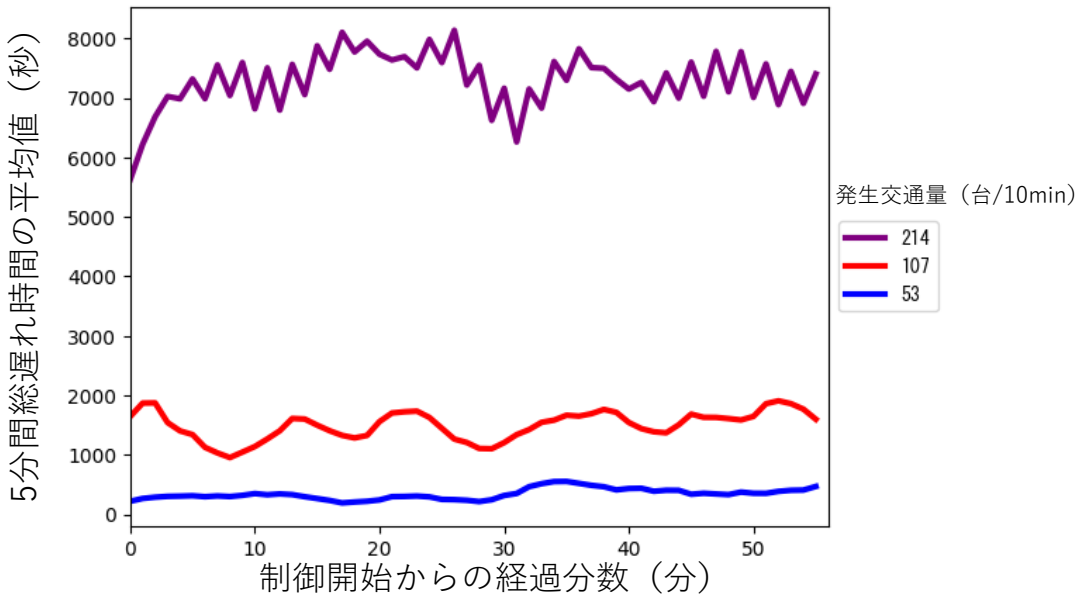

図 4.12　各発生交通量に対する 5 分間総遅れ時間の推移

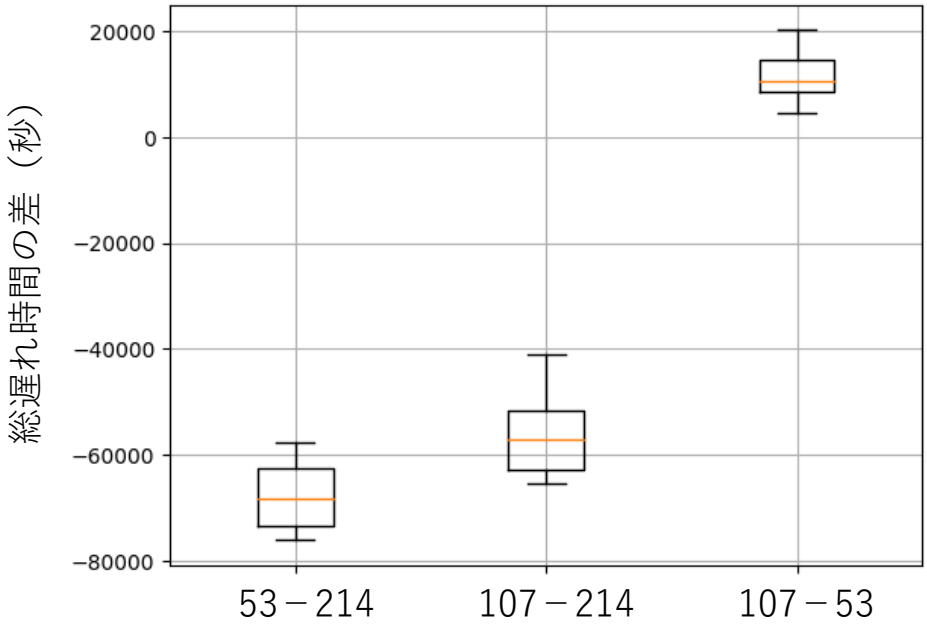

図 4.13　同シード値における総遅れ時間の差の分布

図 4.13 を見ると，どの発生交通量の組み合わせにおいても，総遅れ時間の差は明確であることが分かる．53 台と 214 台の時の差は 70000 秒近くあり，かなり大きな差となっている．差のばらつきに関しては，どの組み合わせにおいても，同じくらいとなっている．以上より，交通需要が変化した場合，総遅れ時間に大きな影響を与え，その差は明確である．107 台から 214 台に増加した場合，総遅れ時間はおよそ 70000 秒だけ増え，反対に 53 台に減少した場合は，およそ 10000 秒だけ減る．

表 4.11 は，Tukey 法を使って，異なる発生交通量に対する総遅れ時間の平均の差の検定を行った結果である．これを見ると，各発生交通量に対する総遅れ時間の差は，どれも 0.1%に有意であった．したがって統計的に，交通需要の違いによる制御効果への影響が無いとは言えない

表 4.11　各発生交通量における総遅れ時間の平均の差の検定結果

発生交通量	平均値の差	95%信頼区間の下限	95%信頼区間の上限	p値	
53-214	-67773.21884	-73311.07659	-62235.36108	4.22E-15	***
107-214	-56387.87313	-61925.73088	-50850.01537	4.22E-15	***
107-53	11385.34571	5847.487957	16923.20347	6.78E-05	***

図 4.14 はそれぞれの発生交通量における，総渋滞長の平均値の推移を表したものである．総遅れ時間の時と同様の傾向が見られた．発生交通量が大きいほど，総渋滞長の平均値やその変動が大きくなる．原因も，総遅れ時間の時と同様であると考えられる．全流入路の合計長さは約 610m であることから，発生交通量が 53 台の時は，全流入路の約 3%が渋滞しており，107 台の時には約 6%が渋滞，214 台の時にはおよそ 13%～25%が渋滞している．以上から，交通需要が変化すると，総渋滞長に影響を与えると言え，交通需要が多いほど，総遅れ時間は大きくなり，変動も大きくなる．裏を返せば，交通需要が多いほど，特定の条件下では，総渋滞長を大幅に小さくできる可能性があると言える．

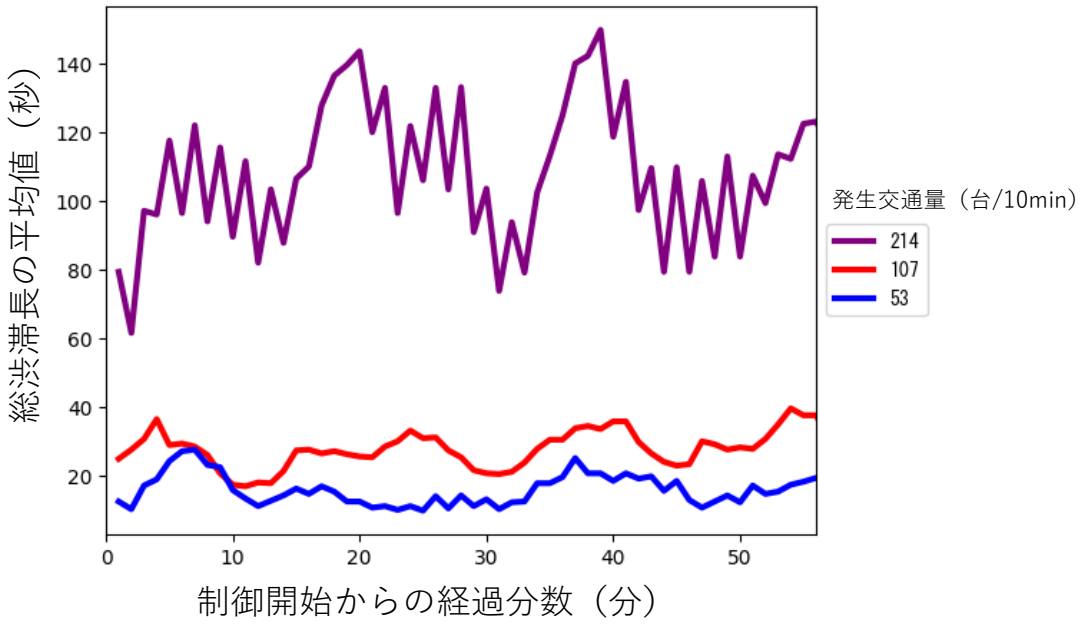

図 4.14　各初期値に対する総渋滞長の平均値の推移

図 4.15 は，発生交通量のシード値を揃えた時の，各発生交通量に対する最大渋滞長の差について，箱ひげ図で示したものである．差の求め方は，以下の通りである．

（シード値 X における最大渋滞長の差）＝
（シード値 X で交通需要が Y の最大渋滞長）－
（シード値 X で交通需要が Z の最大渋滞長）

図 4.15 を見ると，「東京医療センター前」では，同じシード値でも，発生交通量が変化すると最大渋滞長は大きく変わることが分かる．その差は，53 台と 214 台の組み合わせでは流入路の 48～70%分，107 台と 214 台の組み合わせでは流入路の 45～60%分，107 台と 53 台の組み合わせではおよそ 12%分だけ生じている．214 台の場合において，総渋滞長のばらつきが大きかったことと同様に，最大渋滞長のばらつきも大きいことが考えられ，その結果，214 台を含む組み合わせの時に差のばらつきも大きくなっている．「東京医療センター正門前」においても，「東京医療センター前」同様，同じシード値でも，発生交通量が変化すると最大渋滞長は大きく変わることが分かる．その差は，53 台と 214 台の組み合わせでは流入路の約 56%分，107 台と 214 台の組み合わせでは流入路の約 40%分，107 台と 53 台の組み合わせではおよそ 5%分だけ生じている．差のばらつきに関しては，どの発生交通量の組み合わせでも同じくらいの大きさである．よって，この交差点においては，他交差点では渋滞長変動が大きい発生交通量が 214 台の条件であっても，渋滞長に大きな変動は生じないことが分かる．「東根小学校南」においても他交差点と同様に，発生交通量の変化に伴って，最大渋滞長が変わることが分かる．その差は，発生交通量が 214 台と 53 台の組み合わせでは流入路の約 42%分，107 台と

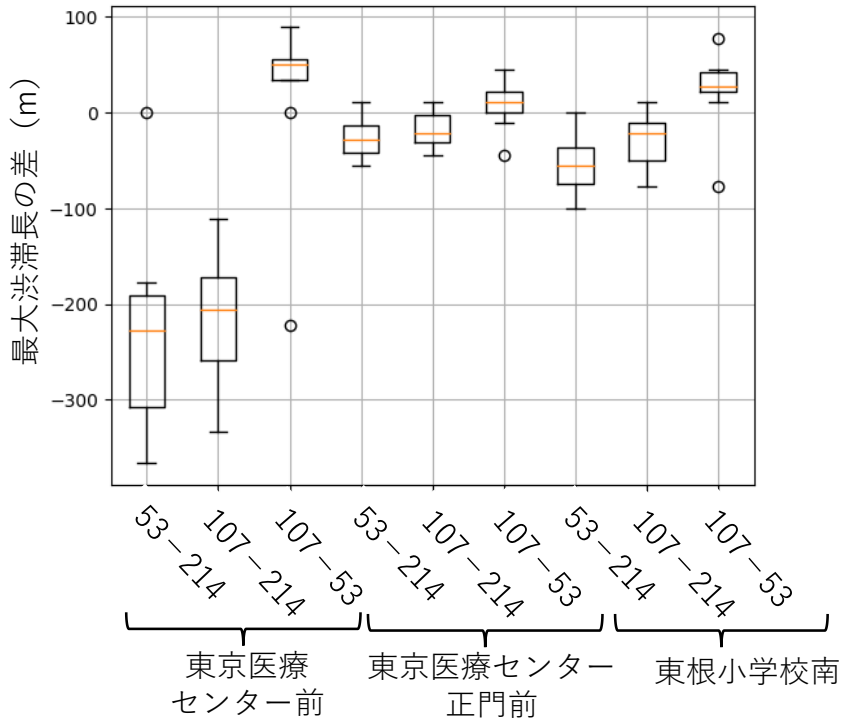

図 4.15　同シード値における最大渋滞長の差の分布

214 台の組み合わせでは流入路の約 20%分，107 台と 53 台の組み合わせではおよそ 14%分だけ生じている．ただし，他交差点と比較した場合，発生交通量の変化による最大渋滞長の差は

小さい．差のばらつきに関しては，「東京医療センター前」と同様の傾向で，214 台を含む組み合わせの場合において，差が大きくばらついている．つまり，214 台の場合，この交差点では，最大渋滞長の変動が大きいと言える．以上のことから，交差点ごとに差の大きさは微小に異なるが，総じて，交通需要が変化すると最大渋滞長も大きく変化する．交通需要が大きくなるほど，最大渋滞長も大きくなり，その逆もしかりである．また，流入路長さが大きいほど，最大渋滞長のばらつきが大きくなると言える．つまり，ある条件においては，最大渋滞長を大幅に小さくできる可能性があると言える．

次に，tukey 法により，各発生交通量に対する最大渋滞長の差が有意なものなのかどうかを確かめた．その結果が**表 4.12～表 4.14** である．全交差点において， 発生交通量が 214 台を含む組み合わせの場合，0.1～5%に有意であった．107 台と 53 台の組み合わせでは，どの交差点においても有意な差はなかった．以上より，交通需要が大きく増加した場合，最大渋滞長に対して影響が無いとは言えないこと，逆に交通需要が半減した場合は，最大渋滞長へ影響があるとは言い切れないことが分かる．

表 4.12　各発生交通量における最大渋滞長の平均の差の検定結果

（交差点名：「東京医療センター前」）

発生交通量	平均値の差	95%信頼区間の下限	95%信頼区間の上限	p値	
53-214	-232.22229	-297.7122396	-166.7323404	6.17E-09	***
107-214	-210.00012	-275.4900696	-144.5101704	4.48E-08	***
107-53	22.22217	-43.26777957	87.71211957	0.681049	
				*** p < 0.001, ** p < 0.01, * p < 0.05	

表 4.13　各発生交通量における最大渋滞長の平均の差の検定結果

（交差点名：「東京医療センター正門前」）

発生交通量	平均値の差	95%信頼区間の下限	95%信頼区間の上限	p値	
53-214	-25.55556	-41.74131929	-9.369800711	0.001557548	**
107-214	-16.66668	-32.85243929	-0.480920711	0.042617257	*
107-53	8.88888	-7.296879289	25.07463929	0.374663519	
				*** p < 0.001, ** p < 0.01, * p < 0.05	

表 4.14　各発生交通量における最大渋滞長の平均の差の検定結果

（交差点名：「東根小学校南」）

発生交通量	平均値の差	95%信頼区間の下限	95%信頼区間の上限	p値	
53-214	-52.22225	-78.68887891	-25.75562109	0.00011731	***
107-214	-28.88887	-55.35549891	-2.422241087	0.03031584	*
107-53	23.33338	-3.133248913	49.80000891	0.091848564	
				*** p < 0.001, ** p < 0.01, * p < 0.05	

図 4.16 は，異なる発生交通量に対して，5 分間平均旅行時間の推移を交差点ごとに表したものである．「東京医療センター前」では，発生交通量が 107 台と 53 台のときの 5 分間平均旅行時間の差が，全体を通して 10 秒以内であり，比較的差は小さく，両者とも，時間経過に伴う変動はあまり見られない．一方で，発生交通量が 214 台のとき，他の発生交通量との差が小さくても 20 秒ほどあり，差が大きい．また，変動が大きくなった．「東京医療センター正門前」では，全体を通して発生交通量が 214 台，107 台，53 台の順に 5 分間平均旅行時間が大きい傾向が見られるが，その差は「東京医療センター前」ほど大きくはなかった．また，変動も比較的小さい．「東根小学校南」では，制御開始から 2000 秒ほどまでは，発生交通量の大小に関わらず，8 秒ほどとなっている．それ以降も，214 台と他との差は最大でも 10 秒ほどとなってお

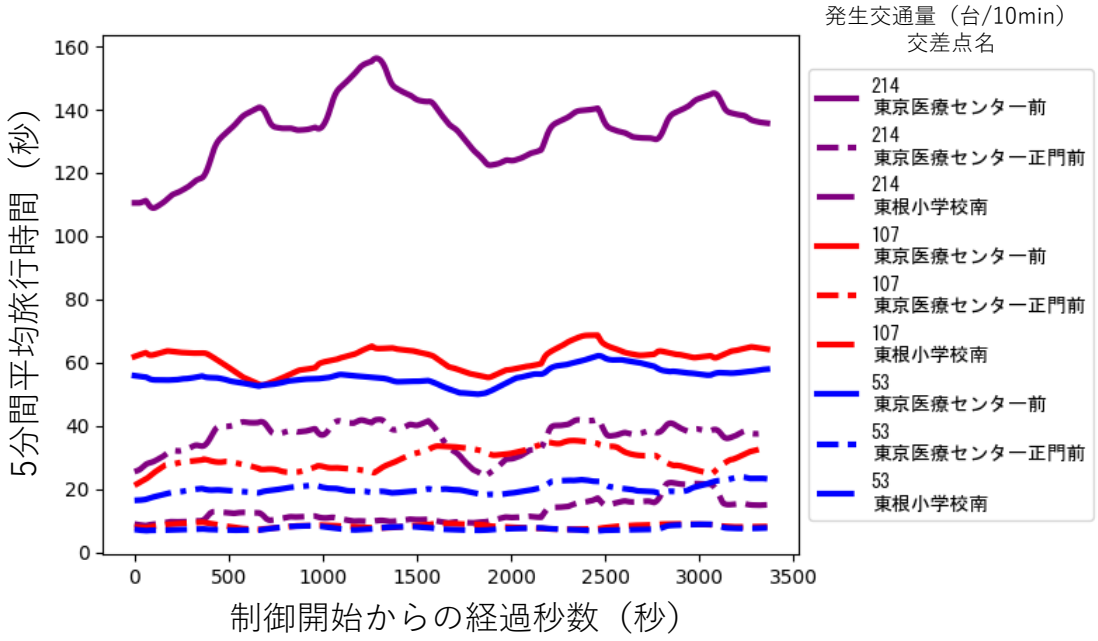

図 4.16　5 分間平均旅行時間の推移

り，差は比較的小さい．変動も全発生交通量において小さい．以上から，流入路長さが大きいほど，交通需要に変化により 5 分間平均旅行時間は大きく影響を受けると言える．具体的には，交通需要が多くなるほど，5 分間平均旅行時間も大きくなり，かつ変動も大きくなる．裏を返せば，ある条件下においては，この値を小さくできる可能性があると言える．

次のページの**図 4.17** は，それぞれの交差点において発生交通量のシード値を揃えた時の，異なる発生交通量に対する平均旅行時間の差の分布について，箱ひげ図で示したものである．差の求め方は，以下の通りである．

（シード値 X における平均旅行速度の差）＝
（シード値 X で交通需要が Y の平均旅行速度）－
（シード値 X で交通需要が Z の平均旅行速度）

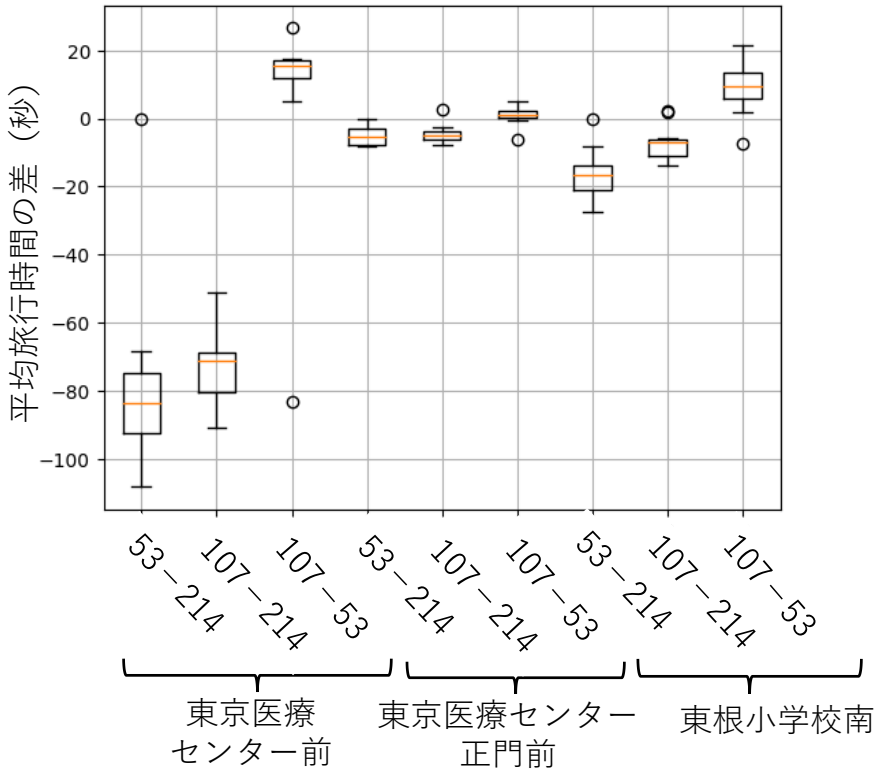

図 4.17　同シード値における平均旅行時間の差の分布

図 4.17 を見ると，「東京医療センター前」では，発生交通量による平均旅行時間の差が顕著に表れている．特に 214 台のときは，他の発生交通量との差が 70～90 秒ほどある．また，差のばらつきも大きくなっている．一方で，107 台と 53 台の差は 15～20 秒となっており比較的差は小さい．「東京医療センター正門前」では，発生交通量による平均旅行時間の差は，どの発生交通量の組み合わせにおいてもほとんどが 10 秒以内に収まっており，差は比較的小さくなっている．また，ばらつきもそれほど大きくない．「東根小学校南」では，「東京医療センター前」に次いで，発生交通量による平均旅行時間の差が大きい．ばらつきも同様である．以上より，どの交差点においても交通需要の違いが，平均旅行時間に影響を及ぼすと言える．特に，流入路の長さが大きい交差点ほど，この傾向は強くなると言える．

次に，tukey 法により，発生交通量による平均旅行時間の差が有意なものなのかどうかを交差点ごとに確かめた．その結果が**表 4.15～表 4.17** である．　どの交差点においても，発生交通量が 214 台のときの平均旅行時間との差は 0.1%に有意である．また，「東根小学校南」においては，53 台と 107 台の時の平均旅行時間の差が 0.1%に有意であった．以上より，交通需要に違いが平均旅行時間に影響が無いとは言い切れない．特に，交通需要が急激に増加した場合に，その影響は無視しきれないと言える．

表 4.15　各発生交通量における平均旅行時間の平均の差の検定結果

（交差点名：「東京医療センター前」）

発生交通量	平均値の差	95%信頼区間の下限	95%信頼区間の上限	p値	
53-214	-77.46902486	-97.33204647	-57.60600324	8.61E-10	***
107-214	-71.84913958	-91.71216119	-51.98611797	4.12E-09	***
107-53	5.619885277	-14.24313634	25.48290689	0.76465101	
				*** p < 0.001, ** p < 0.01, * p < 0.05	

表 4.16　各発生交通量における平均旅行時間の平均の差の検定結果

（交差点名：「東京医療センター正門前」）

発生交通量	平均値の差	95%信頼区間の下限	95%信頼区間の上限	p値	
53-214	-5.184048074	-7.316050697	-3.052045451	5.74E-06	***
107-214	-4.468123464	-6.600126087	-2.33612084	5.21E-05	***
107-53	0.715924611	-1.416078012	2.847927234	0.686390794	
				*** p < 0.001, ** p < 0.01, * p < 0.05	

表 4.17　各発生交通量における平均旅行時間の平均の差の検定結果

（交差点名：「東根小学校南」）

発生交通量	平均値の差	95%信頼区間の下限	95%信頼区間の上限	p値	
53-214	-16.18131658	-21.59818113	-10.76445203	1.69E-07	***
107-214	-7.04162797	-12.45849252	-1.624763422	0.008965391	**
107-53	9.13968861	3.722824061	14.55655316	0.000771137	***
				*** p < 0.001, ** p < 0.01, * p < 0.05	

4.7.4　異なるプローブ車混入率による制御効果の違い

プローブ車混入率の変化によって制御効果に違いがあるかどうかを確かめる．諸設定条件は**表 4.18** の通りである．また，AI の学習データは混入率を 30%としていたため，それ以外の混入率のケースでは，30%へ補正したものを AI の入力データとした．

表 4.18　諸設定条件

ネットワーク構成	駒沢通り(4.2.1 参照)
10 分間平均発生交通量（台） （平均に対し，ポアソン分布を仮定した 確率分布で 10 分おきに変動）	107（混雑時交通量）
車種構成	小型車のみ
プローブ車混入率（%）	10，30，50，70，90
希望速度（km/h）	40（規制速度）
従方向の青時間スプリット	最低でも 0.4 を確保
シミュレーション時間	6:50～9:30（制御開始は 8:30～）
発生交通量のシード値	100 から開始し，サンプル数に応じて 1 刻みで増やしていく
車両発生間隔のシード値	1000
探索アルゴリズム	Nelder-Mead
探索初期値	従来信号制御手法による 信号制御パラメータが 10 サイクル分継続

図 4.18 は異なる混入率における，5 分間総遅れ時間の平均値の推移を制御開始からシミュレーション終了まで表したものである．

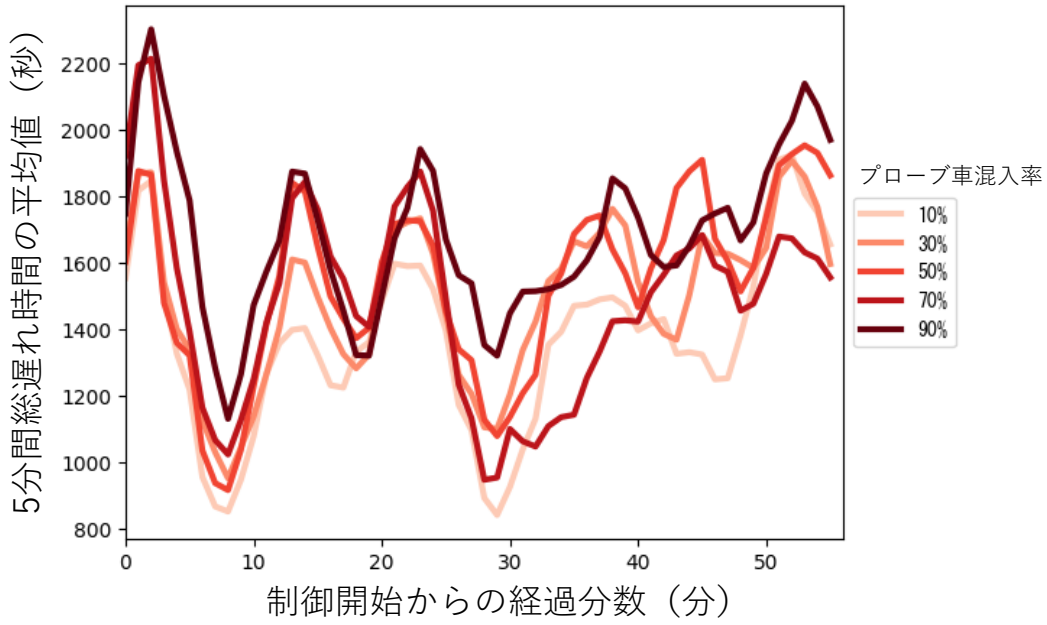

図 4.18　各混入率に対する 5 分間総遅れ時間の推移

制御開始から 20 分程までは，学習データで設定していた 30%から値が離れるほど，時点毎の 5 分間総遅れ時間の平均は大きくなるといった傾向が見られる．それ以降は，10%が全体を通して小さく，90%が大きいといった傾向にあるが，それ以外は，大小が入れ替わり，何らかの傾向は見られない．また，変動の仕方は，いずれの混入率においても概ね同じ傾向が見られた．以上より，学習データで想定していた混入率に対して，実際の混入率が大きく異なる場合，平均 5 分間総遅れ時間は大きくなる傾向があり，プローブ車混入率による平均 5 分間総遅れ時間への影響は少なからずあると言える．

次ページの**図 4.19** は，発生交通量のシード値を揃えた時の，異なる混入率に対する総遅れ時間の分布について，折れ線グラフで示したものである．これを見ると，シード値によっては，混入率の違いにより，総遅れ時間が何らかの傾向も持たずに大きく変動する場合もあるが，それ以外は，同じシード値であっても混入率が 30%から大きく離れるほど，総遅れ時間が大きくなる傾向が見られる．

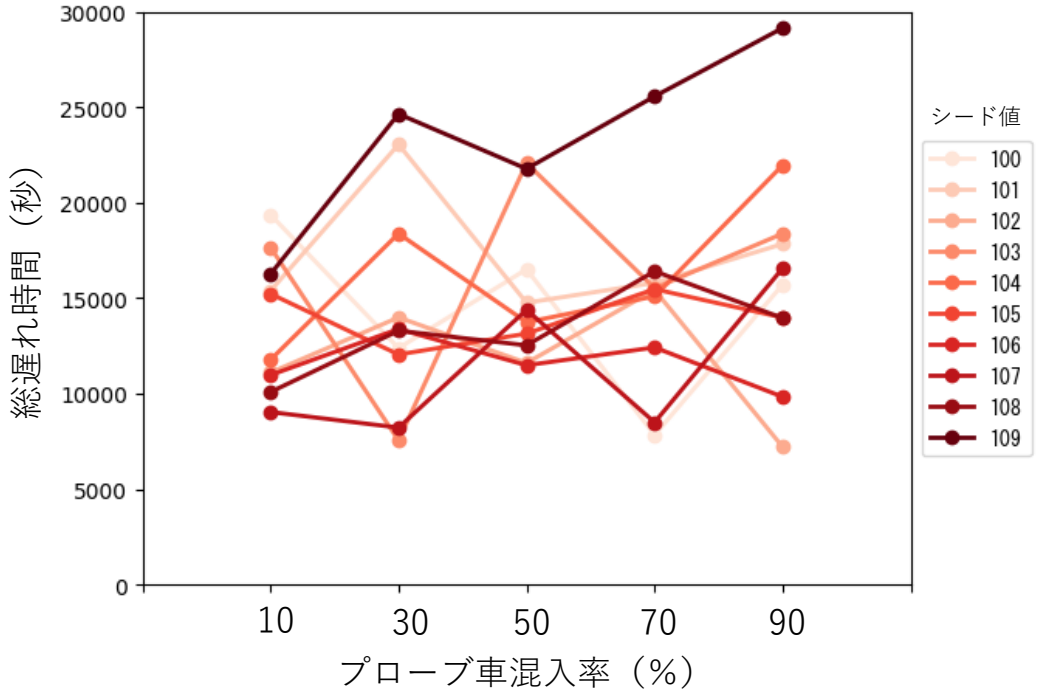

図 4.19　同シード値における総遅れ時間の分布

表 4.19　異なる混入率における総遅れ時間の平均の差の検定結果

混入率	平均値の差	95%信頼区間の下限	95%信頼区間の上限	p値
90-50	1257.029027	-5000.376894	7514.434947	0.978653203
70-50	-396.1905428	-6653.596463	5861.215378	0.999756141
10-50	-1514.332228	-7771.738149	4743.073693	0.958149125
30-50	-506.5144027	-6763.920323	5750.891518	0.99935527
70-90	-1653.21957	-7910.62549	4604.186351	0.943092285
10-90	-2771.361255	-9028.767176	3486.044666	0.717417669
30-90	-1763.54343	-8020.94935	4493.862491	0.928996425
10-70	-1118.141685	-7375.547606	5139.264235	0.986180181
30-70	-110.3238599	-6367.729781	6147.082061	0.999998511
30-10	1007.817825	-5249.588095	7265.223746	0.990657557
				*** p < 0.001, ** p < 0.01, * p < 0.05

表 4.19 は，tukey 法により，異なる混入率に対する，総遅れ時間の平均差の検定を行った結果である．10%と 90%の組み合わせでは，他と比べて p 値が小さくなっていることから，混入率の差が大きくなるほど，総遅れ時間の差も大きくなる傾向が読み取れる．しかし，全ての混入率の組み合わせにおいても，有意な差は見られず，統計的には，異なる混入率による総遅れ時間への影響はあるとは言い切れない．

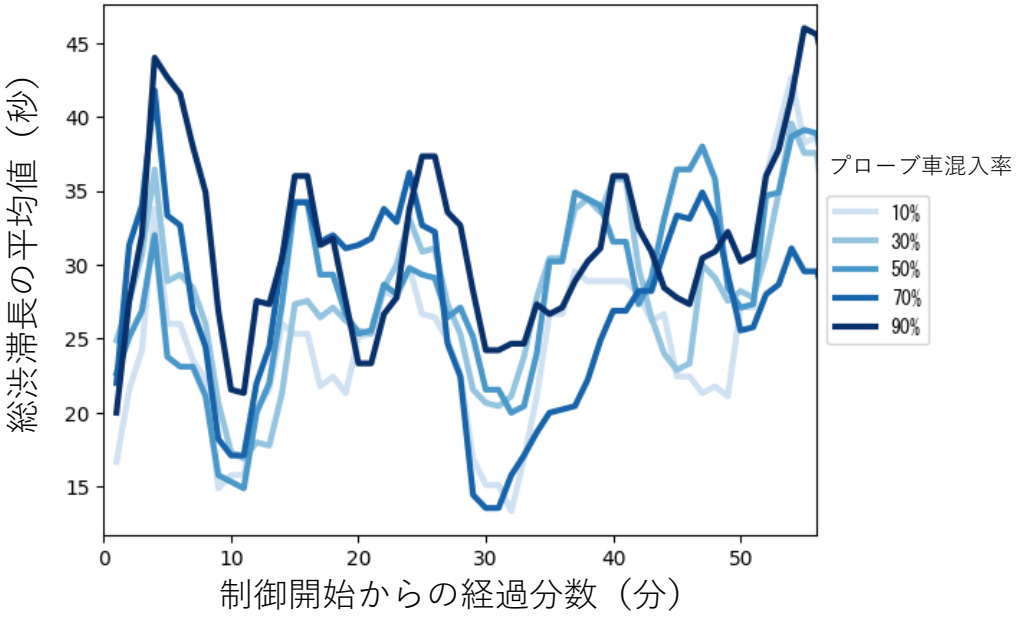

図 4.20　各混入率に対する総渋滞長の平均値の推移

図 4.20 は，異なる混入率に対する平均総渋滞長の推移を，制御開始からシミュレーション終了まで表したものである．平均 5 分間総遅れ時間の時と同様の傾向が見られ，混入率が 30%から離れるほど，平均総渋滞長が大きくなる傾向が見られる．また，変動の仕方も，混入率に寄らず同じ変動の傾向を見せている．

次のページの**図 4.21～図 4.23** は，各交差点において，発生交通量のシード値を揃えた時の，異なる混入率に対する最大渋滞長について，折れ線グラフで示したものである．「東京医療センター前」では，一部のシード値で，混入率の変化によって最大渋滞長が大きく変わる場合もあるが，全体としては，混入率による最大渋滞長の変化はそれほど大きくはない．「東京医療センター正門前」と「東根小学校南」においても同様の傾向が見られた．したがって，混入率によって，最大渋滞長は影響をそれほど受けないと言える．

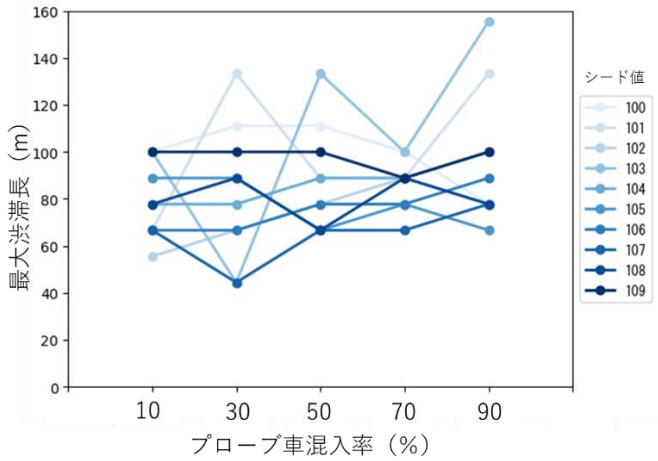

図 4.21　同シード値で異なる混入率における最大渋滞長
（「東京医療センター前」）

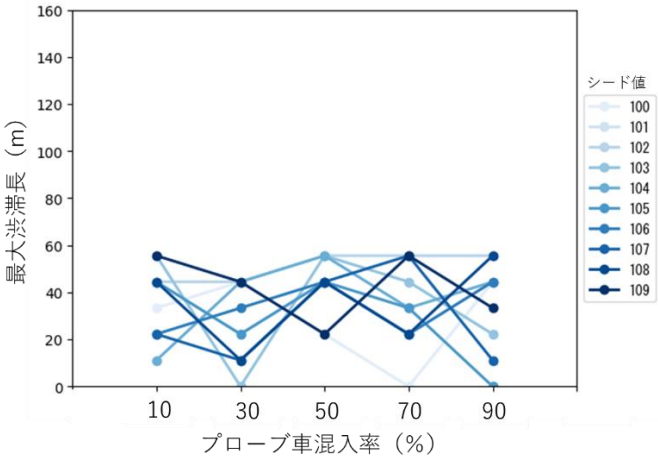

図 4.22　同シード値で異なる混入率における最大渋滞長
（「東京医療センター正門前」）

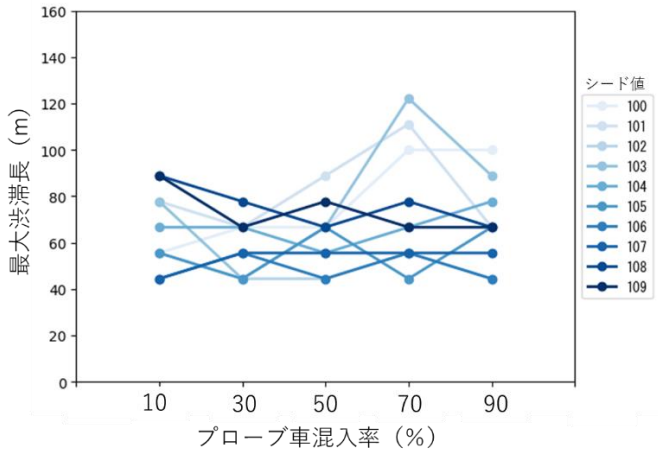

図 4.23　同シード値で異なる混入率における最大渋滞長
（「東根小学校南」）

表 4.20～表 4.22 は，tukey 法を用いて，異なる混入率に対する最大渋滞長の平均差を検定した結果である．どの交差点においても，異なる混入率による最大渋滞長に有意な差が見られなかった．したがって，統計的には，混入率による最大渋滞長への影響があるとは言い切れない．

表 4.20　異なる混入率における最大渋滞長の平均の差の検定結果
（「東京医療センター前」）

混入率	平均値の差	95%信頼区間の下限	95%信頼区間の上限	p値
90-50	7.77783	-20.37427182	35.92993182	0.933644402
70-50	-1.11107	-29.26317182	27.04103182	0.999962814
10-50	-7.77773	-35.92983182	20.37437182	0.933647313
30-50	-5.55557	-33.70767182	22.59653182	0.980009051
70-90	-8.8889	-37.04100182	19.26320182	0.896446605
10-90	-15.55556	-43.70766182	12.59654182	0.523822625
30-90	-13.3334	-41.48550182	14.81870182	0.664571541
10-70	-6.66666	-34.81876182	21.48544182	0.96125519
30-70	-4.4445	-32.59660182	23.70760182	0.991339759
30-10	2.22216	-25.92994182	30.37426182	0.999416265

*** p < 0.001, ** p < 0.01, * p < 0.05

表 4.21　異なる混入率における最大渋滞長の平均の差の検定結果
（「東京医療センター正門前」）

混入率	平均値の差	95%信頼区間の下限	95%信頼区間の上限	p値
90-50	-7.77778	-28.64589391	13.09033391	0.826190212
70-50	-8.88888	-29.75699391	11.97923391	0.745441157
10-50	-5.55556	-26.42367391	15.31255391	0.941574186
30-50	-15.55557	-36.42368391	5.31254391	0.230427224
70-90	-1.1111	-21.97921391	19.75701391	0.999877464
10-90	2.22222	-18.64589391	23.09033391	0.998106189
30-90	-7.77779	-28.64590391	13.09032391	0.826189545
10-70	3.33332	-17.53479391	24.20143391	0.990946274
30-70	-6.66669	-27.53480391	14.20142391	0.892439576
30-10	-10.00001	-30.86812391	10.86810391	0.654763832

*** p < 0.001, ** p < 0.01, * p < 0.05

表 4.22　異なる混入率における最大渋滞長の平均の差の検定結果
（「東根小学校南」）

混入率	平均値の差	95%信頼区間の下限	95%信頼区間の上限	p値
90-50	4.44444	-18.57377349	27.46265349	0.981557805
70-50	12.22219	-10.79602349	35.24040349	0.562333694
10-50	2.22222	-20.79599349	25.24043349	0.998710117
30-50	-4.44445	-27.46266349	18.57376349	0.981557651
70-90	7.77775	-15.24046349	30.79596349	0.871364811
10-90	-2.22222	-25.24043349	20.79599349	0.998710117
30-90	-8.88889	-31.90710349	14.12932349	0.806997882
10-70	-9.99997	-33.01818349	13.01824349	0.731523237
30-70	-16.66664	-39.68485349	6.351573492	0.25636437
30-10	-6.66667	-29.68488349	16.35154349	0.922151115
				*** p < 0.001, ** p < 0.01, * p < 0.05

次のページの**図 4.24～図 4.26** は，各交差点において，異なる混入率に対する 5 分間平均旅行時間の推移を制御開始からシミュレーション終了まで表したものである．「東京医療センター前」では，90%の場合，他の混入率と比べて全体を通して 5 分間平均旅行時間が大きい傾向がある．他の混入率に関しては，ほとんど同じ値で推移している．「東京医療センター正門前」では，混入率に関わらず，ほぼ同じ 5 分間平均旅行時間で推移していることが分かる．「東根小学校南」では，「東京医療センター前」と同様の傾向が見られ，90%の時の 5 分間平均旅行時間が大きい傾向がある．制御開始から 1500 秒ほどまでは，どの混入率に対する値も同じような変動で推移しているが，それ以降は，30%から離れている混入率において，大きな変動で推移している．以上より，混入率が 30%から離れているほど，5 分間平均旅行時間が大きくなる傾向があると言える．また，流入路長さが大きいほど，変動が大きくなる傾向があると言える．

p. 66 にある**図 4.27～図 4.29** は，各交差点において，発生交通量のシード値を揃えた時の，異なる混入率に対する平均旅行時間について，折れ線グラフで示したものである．「東京医療センター前」では，混入率が 30%から離れるほど，平均旅行時間が大きくなる傾向が若干ではあるが見られる．また，同じ混入率でも，シード値によって平均旅行時間の変動が比較的大きくなった．「東京医療センター正門前」では，混入率の変化による平均旅行時間の変動はほとんど見られなかった．「東根小学校南」では，「東京医療センター前」と同様の傾向が見られ，混入率が 30%から離れるほど，平均旅行時間が大きくなる傾向が若干ではあるが見られる．また，同じ混入率でも，シード値によって平均旅行時間の変動が「東京医療センター前」に次いで大きくなった．以上より，流入路長さが大きい交差点では，混入率が 30%から離れるほど，平均旅行時間が大きくなる傾向があり，混入率に関わらず平均旅行時間の変動が大きいと言える．

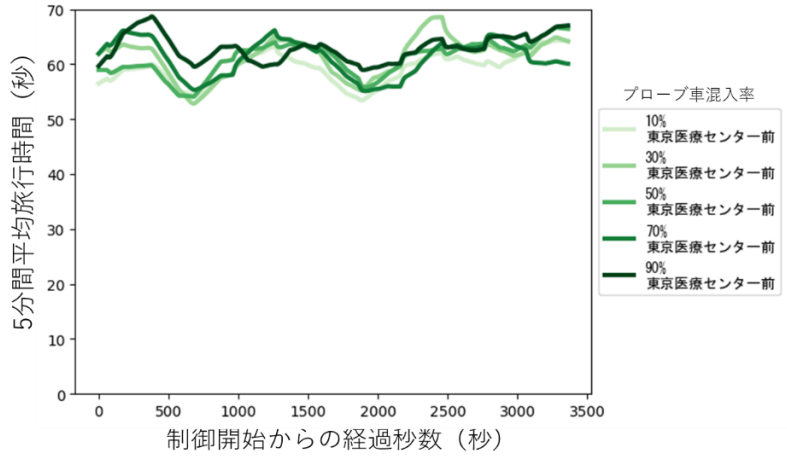

図 4.24　異なる混入率における 5 分間平均旅行時間の推移
（「東京医療センター前」）

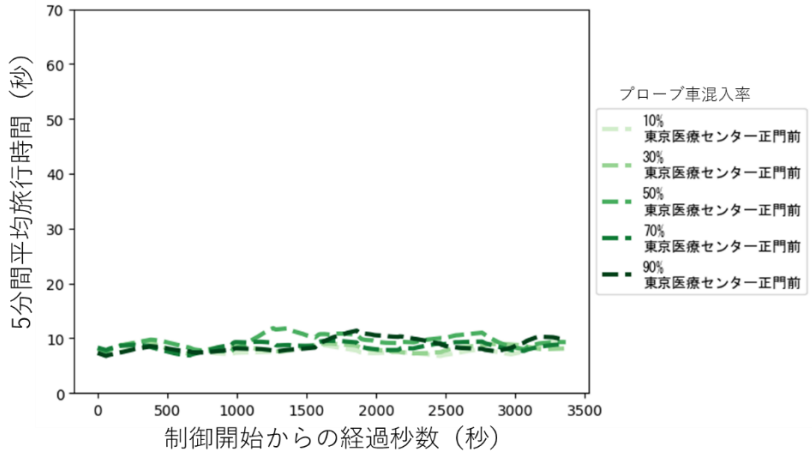

図 4.25　異なる混入率における 5 分間平均旅行時間の推移
（「東京医療センター正門前」）

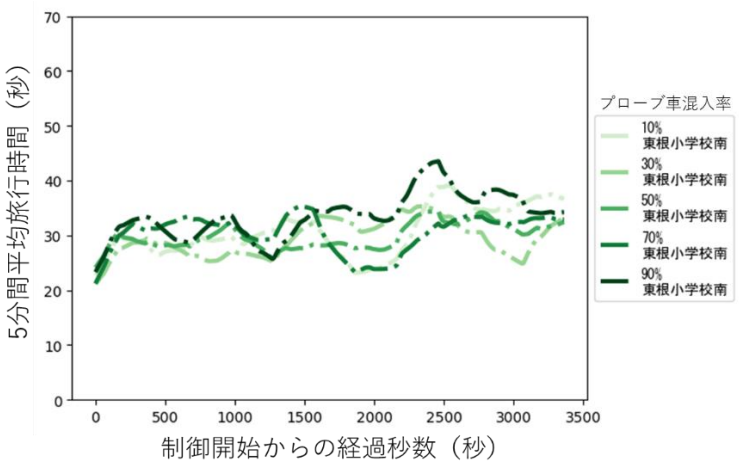

図 4.26　異なる混入率における 5 分間平均旅行時間の推移
（「東根小学校南」）

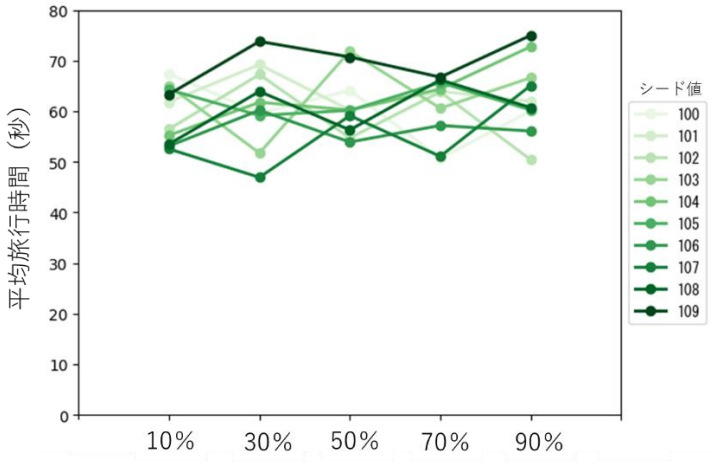

図 4.27　同シード値で異なる混入率における平均旅行時間
（「東京医療センター前」）

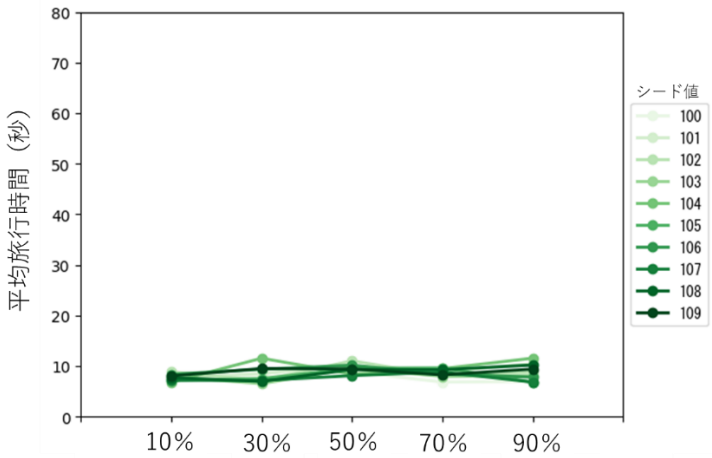

図 4.28　同シード値で異なる混入率における平均旅行時間
（「東京医療センター正門前」）

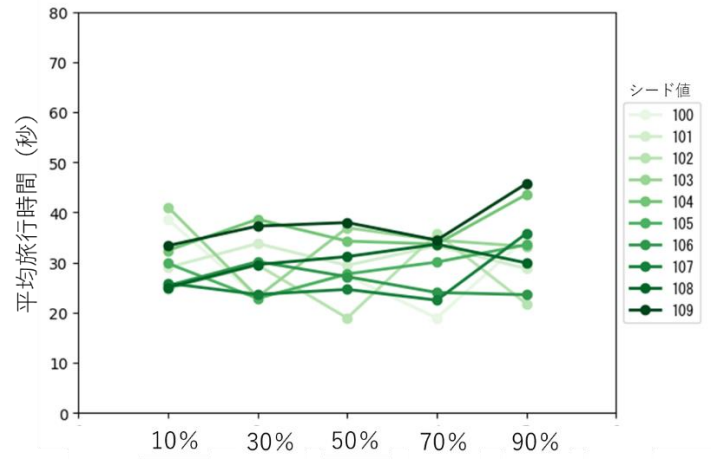

図 4.29　同シード値で異なる混入率における平均旅行時間
（「東根小学校南」）

表 4.23～表 4.24 は，tukey 法を用いて，異なる混入率に対する平均旅行時間に有意な差があるかどうか検定した結果である．「東京医療センター正門前」における，混入率 10%と 50%の組み合わせが，5%に有意な差が見られた．しかし，それ以外では有意な差が見られなかった．したがって，統計的には，混入率によって平均旅行時間に影響があるとは言い切れない．

表 4.23　異なる混入率における平均旅行時間の平均の差の検定結果
（「東京医療センター前」）

混入率	平均値の差	95%信頼区間の下限	95%信頼区間の上限	p値	
90-50	1.752936356	-6.736927688	10.2428004	0.976391217	
70-50	-0.106446326	-8.59631037	8.383417718	0.999999619	
10-50	-1.865446599	-10.35531064	6.624417444	0.970389019	
30-50	0.219338978	-8.270525065	8.709203022	0.999993147	
70-90	-1.859382682	-10.34924673	6.630481361	0.970736052	
10-90	-3.618382955	-12.108247	4.871481088	0.745042249	
30-90	-1.533597378	-10.02346142	6.956266666	0.985608003	
10-70	-1.759000273	-10.24886432	6.730863771	0.976090817	
30-70	0.325785305	-8.164078739	8.815649348	0.999966755	
30-10	2.084785578	-6.405078466	10.57464962	0.955934238	
			*** p < 0.001, ** p < 0.01, * p < 0.05		

表 4.24　異なる混入率における平均旅行時間の平均の差の検定結果
（「東京医療センター正門前」）

混入率	平均値の差	95%信頼区間の下限	95%信頼区間の上限	p値	
90-50	-0.740808522	-2.25354215	0.771925106	0.636158865	
70-50	-0.90016389	-2.412897518	0.612569738	0.4497263	
10-50	-1.672275335	-3.185008963	-0.159541707	0.023597025	*
30-50	-1.235700628	-2.748434256	0.277033	0.157143841	
70-90	-0.159355367	-1.672088995	1.353378261	0.998184623	
10-90	-0.931466812	-2.44420044	0.581266816	0.415035691	
30-90	-0.494892106	-2.007625734	1.017841522	0.883901319	
10-70	-0.772111445	-2.284845073	0.740622183	0.599216384	
30-70	-0.335536739	-1.848270367	1.177196889	0.969361843	
30-10	0.436574706	-1.076158922	1.949308334	0.923074027	
			*** p < 0.001, ** p < 0.01, * p < 0.05		

表 4.25　異なる混入率における平均旅行時間の平均の差の検定結果
（「東根小学校南」）

混入率	平均値の差	95%信頼区間の下限	95%信頼区間の上限	p値
90-50	3.521332969	-4.470034476	11.51270041	0.721193822
70-50	0.525949194	-7.465418251	8.517316639	0.999715673
10-50	0.963780388	-7.027587057	8.955147833	0.996924612
30-50	-0.401857416	-8.393224861	7.589510029	0.999902381
70-90	-2.995383775	-10.98675122	4.99598367	0.823232387
10-90	-2.557552581	-10.54892003	5.433814864	0.891816686
30-90	-3.923190385	-11.91455783	4.06817706	0.634002815
10-70	0.437831194	-7.553536252	8.429198639	0.999862712
30-70	-0.92780661	-8.919174055	7.063560835	0.997347265
30-10	-1.365637804	-9.357005249	6.625729641	0.988313442
				*** p < 0.001, ** p < 0.01, * p < 0.05

4.7.5　従来信号制御手法と提案信号制御手法における制御効果の違い

提案信号制御手法と従来信号制御手法における制御効果に違いがあるかどうかを確かめる．提案信号制御手法の諸設定条件は**表 4.26** の通りである．

表 4.26　提案信号制御手法の諸条件

ネットワーク構成	駒沢通り（4.2.1 参照）
10 分間平均発生交通量（台） （平均に対し，ポアソン分布を仮定した 確率分布で 10 分おきに変動）	107（混雑時交通量）
車種構成	小型車のみ
プローブ車混入率（%）	30
希望速度（km/h）	40（規制速度）
従方向の青時間スプリット	最低でも 0.4 を確保
シミュレーション時間	6:50～9:30（制御開始は 8:30～）
発生交通量のシード値	100 から開始し，サンプル数に応じて 1 刻みで増やしていく
車両発生間隔のシード値	1000
探索アルゴリズム	Nelder-Mead
探索初期値	従来信号制御手法による 信号制御パラメータが 10 サイクル分継続

従来手法信号制御手法の諸設定条件は**表 4.27** の通りである．

表 4.27　従来信号制御手法の諸条件

ネットワーク構成	駒沢通り(4.2.1 参照)	
10 分間平均発生交通量（台） （平均に対し，ポアソン分布を仮定した 確率分布で 10 分おきに変動）	107（混雑時交通量）	
車種構成	小型車のみ	
希望速度（km/h）	40（規制速度）	
従方向の青時間スプリット	0.45	
シミュレーション時間	6:50～9:30（制御開始は 8:30～）	
発生交通量のシード値	100 から開始し，サンプル数に応じて 1 刻みで増やしていく	
車両発生間隔のシード値	1000	
「東京医療センター前」 サイクル長・青時間	サイクル長（秒）	47
	主方向の青時間（秒）	14
	従方向の青時間（秒）	15
	右折矢印（秒）	5
「東京医療センター正門前」 サイクル長・青時間	サイクル長（秒）	47
	主方向の青時間（秒）	24
	従方向の青時間（秒）	20
「東根小学校南」 サイクル長・青時間	サイクル長（秒）	47
	主方向の青時間（秒）	20
	従方向の青時間（秒）	17

次のページの**図 4.30** は両信号制御手法における，5 分間総遅れ時間の平均値の推移と，探索の前後における予測総遅れ時間の推移を，制御開始からシミュレーション終了まで表したものである．従来信号制御手法では時間経過ごとに，平均 5 分間総遅れ時間が上昇していくことが分かる．一方で，提案信号制御手法では横ばいに推移していることがわかる．両手法に対する平均 5 分間総遅れ時間の差はかなり明確に出ていることが分かる．ただし，提案信号制御手法では，平均 5 分間総遅れ時間が減少している傾向は見られない．また，探索前後における予測総遅れ時間は，ほとんど差が無く，周期的な変動を繰り返しているように見られる．さらに，提案信号制御手法により得られた実際の 5 分間総遅れ時間に対して，予測総遅れ時間はおおよそ 5000 秒だけ大きくなっており，過大に予測していることが分かる．ここで，AI の予測精度を検証する．次ページの**図 4.31** は，5 分間総遅れ時間の予測値と実測値における分布をプロットしたものである．図中の黒線は y=x の線を示しており，赤線は，プロットに対する回帰直線である．すなわち，赤線が黒線に近いほど，AI 予測精度が高いと言える．**図 4.31** を見ると，黒線に対して赤線が寝ている．すなわち，5 分間総遅れ時間が大きくなるほど，過小に予測

し，5 分間総遅れ時間が小さくなるほど，過大に評価する傾向があると言える．このような AI 予測精度であるために，**図 4.30** において，比較的小さな 5 分間総遅れ時間（赤線）に対して，予測総遅れ時間が過大に予測されたと考えられる．**図 4.31** において，プロットが寝てしまう原因として，予測総遅れ時間を説明するための情報が入力変数に不足していることや，入力変数に情報はあるものの，不要な情報に埋もれてしまい入力変数から重要な情報を抽出できていないことなどが考えられる．対処方法として，より大きな，あるいはより小さな 5 分間総遅れ時間となったときのデータを拡充し，AI に学習させることが考えられる．また，今回構築

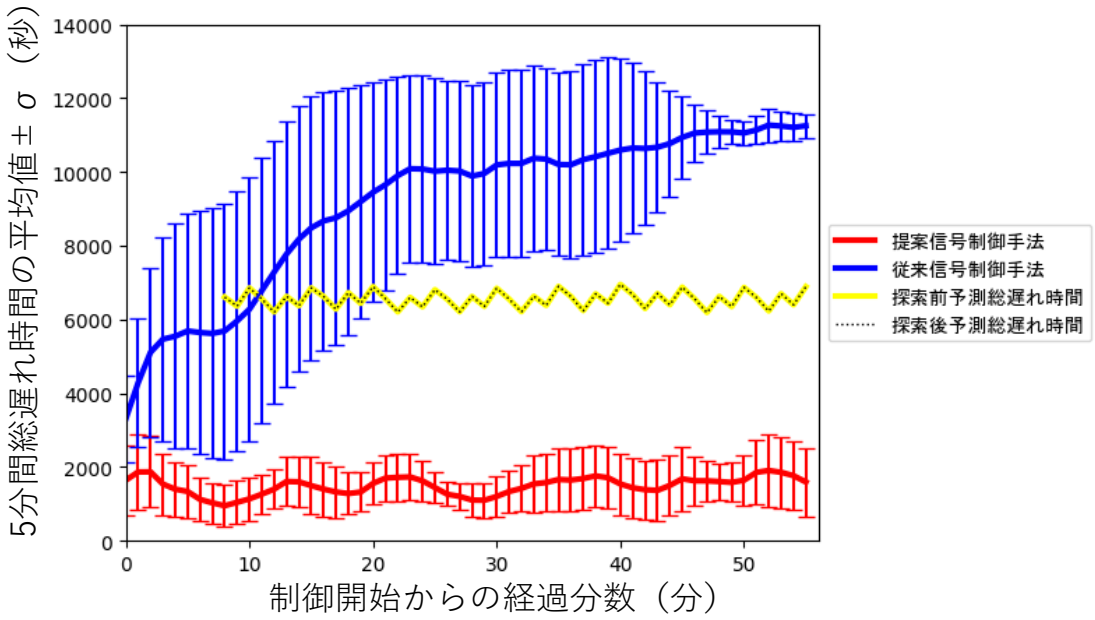

図 4.30　各制御手法に対する 5 分間総遅れ時間の推移

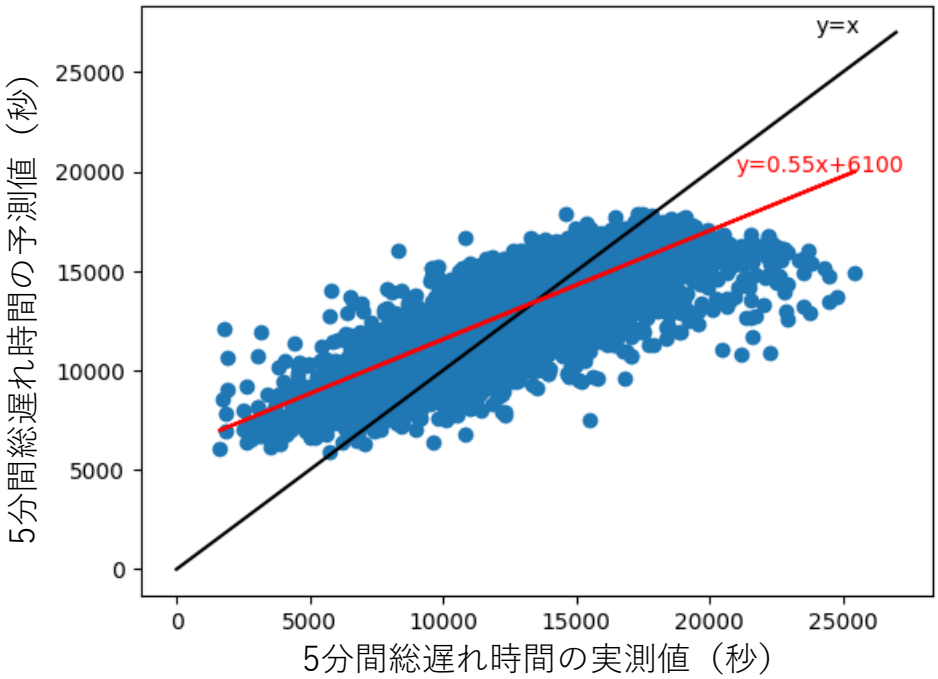

図 4.31　5 分間総遅れ時間の実測値と予測値の分布

したニューラルネットワークは**図 3.4** に挙げたように，LSTM 層 1 個と通常の層 1 個といった単純なものとなっているため，LSTM 層や通常の層，ノード数を増やすことによって，より複雑な構成にするということも考えられる．

図 4.30 で見た通り，提案信号制御手法では，平均 5 分間総遅れ時間が減少している傾向や，探索前後での予測総遅れ時間差はほとんど見られない．そこで，探索の性能を確認するために，探索前後における予測総遅れ時間の差の分布を**図 4.32** にパレート図にして示す．

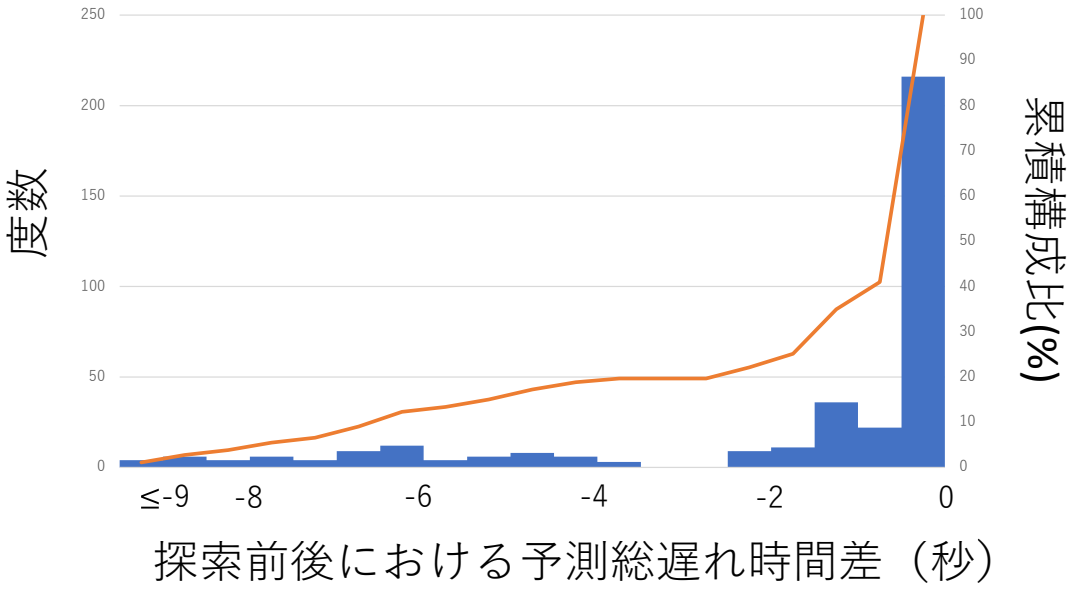

図 4.32　探索前後における予測総遅れ時間の差の分布

これを見ると，探索した結果，短縮することができた総遅れ時間の秒数は，9 秒以上短縮できた場合もあるものの，ほとんどが 0～0.5 秒となっており，最小の予測総遅れ時間を探索できているとは言い難い．その原因として，4 つ考えられる．1 つは，十分な探索時間が取れていないことが考えられる．探索時間は最大でも 30 秒としている．そのようにした理由は，今回用いた AVENUE では，30 秒以上何も通信が無い場合，タイムアウトとなり接続が切れてしまうためである．探索アルゴリズムによって，収束時間の長短はあるものの，収束するのには，数時間～数日ほどかかる．それにもかかわらず，最大 30 秒以内で探索をしていることから，ほとんど探索を行えていない可能性がある．2 つ目は，探索変数が多いことにより，計算処理に時間がかかってしまうことである．探索では繰り返し計算が行われるが，変数が多い場合，一回の計算に時間がかかり，十分な繰り返し回数を得られない可能性がある．3 つ目は，１点の局所解に陥ってしまい，脱出できなくなったために，より小さな総遅れ時間を見つけることができなかった可能性が考えられる．以上より，探索アルゴリズムの選定にあたっては，局所解を探索し，収束が速いものを選ぶべきであると言える．

続いて，**図 4.33** は発生交通量のシード値を揃えた時の，各制御手法に対する総遅れ時間の差について，箱ひげ図で示したものである．差の求め方は，以下の通りである．

（シード値 X における総遅れ時間の差）＝
（シード値 X で提案信号制御手法の総遅れ時間）－
（シード値 X で従来信号制御手法の総遅れ時間）

図 4.33 を見ると，両制御手法による総遅れ時間の差は 65000 秒～80000 秒ほどに集中しており，かなり明確な差が出ていると言える．提案信号制御手法による総遅れ時間は，従来信号制御手法のおよそ 84%だけ減少している．よって，提案信号制御手法の優位性を主張することができる．

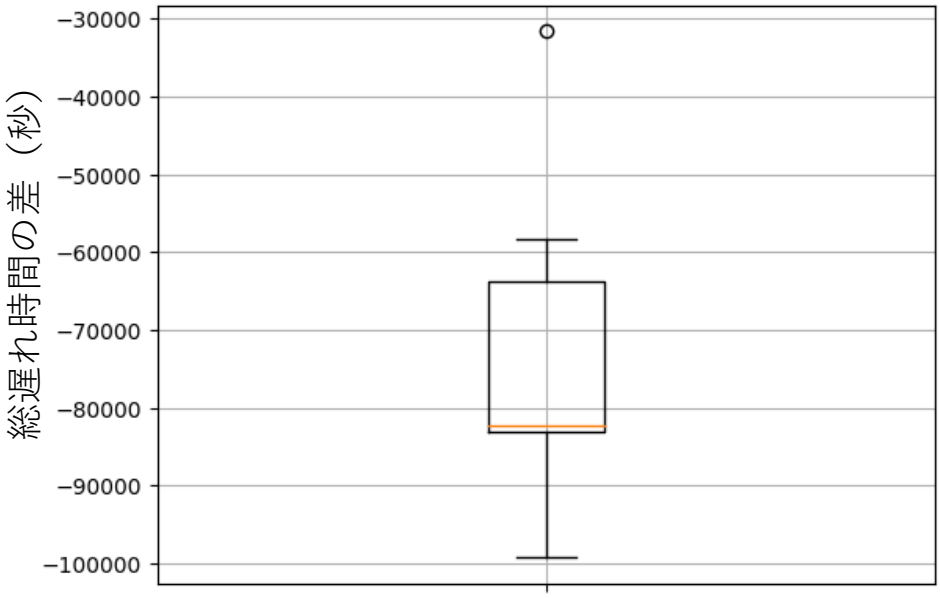

図 4.33　同シード値における総遅れ時間の差の分布

次に，対応のある t 検定により，両制御手法による総遅れ時間の差が有意なものなのかどうかを確かめた．検定の結果，0.1%に有意な差であった（t = -12.224，df = 18，p = 6.576e-07）．したがって，制御手法の違いによる総遅れ時間への影響はないとは言い切れず，提案信号制御手法を用いた方が，総遅れ時間が小さくなる可能性が高いと言える．

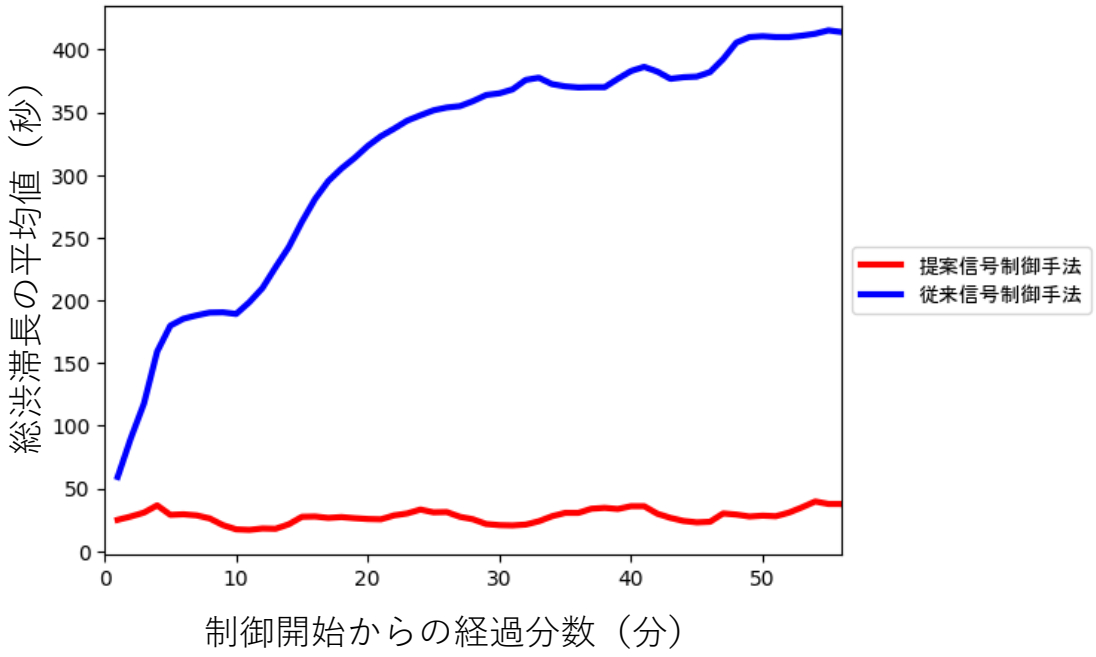

図 4.34　各制御手法に対する総渋滞長の平均値の推移

図 4.34 はそれぞれの制御手法における，総渋滞長の平均の推移を制御開始からシミュレーション終了まで表したものである．平均 5 分間総遅れ時間の時と同様の傾向が見られた．

次ページの**図 4.35** は，各交差点において，発生交通量のシード値を揃えた時の，両制御手法に対する最大渋滞長の差について，箱ひげ図で示したものである．差の求め方は，以下の通りである．

（シード値 X における最大渋滞長の差）＝
（シード値 X で提案信号制御手法の最大渋滞長）－
（シード値 X で従来信号制御手法の最大渋滞長）

図 4.35 を見ると，特に「東京医療センター前」において差が大きくなっており，その差は 350m ほどである．この交差点の流入路長さが約 413m であるため，流入路のおよそ 85%分だけの差がある．「東京医療センター正門前」では，両手法に対する最大渋滞長の差は小さいものの，提案信号制御手法の方が若干大きい傾向が見られる．これは，従来信号制御手法の場合，「東京医療センター前」において捌け残りが多く発生し，それより下流側交差点にほとんど車両が流入しなくなってしまい，自由流に近い状態となって渋滞が生じづらくなったことが原因であると考えられる．「東根小学校南」においても「東京医療センター前」と同様の傾向が見

られた．以上より，各制御手法により，総遅れ時間に影響を及ぼす．特に，需要率が最も高い交差点において，最大渋滞長の差は大きくなり，提案信号制御手法の方が明らかに小さい．それより下流の交差点においては，相対的に，提案信号制御手法の方が総遅れ時間が大きくなる．また，提案信号制御手法では，需要率に寄らず，総遅れ時間はほぼ横ばいに推移する傾向があると言える．

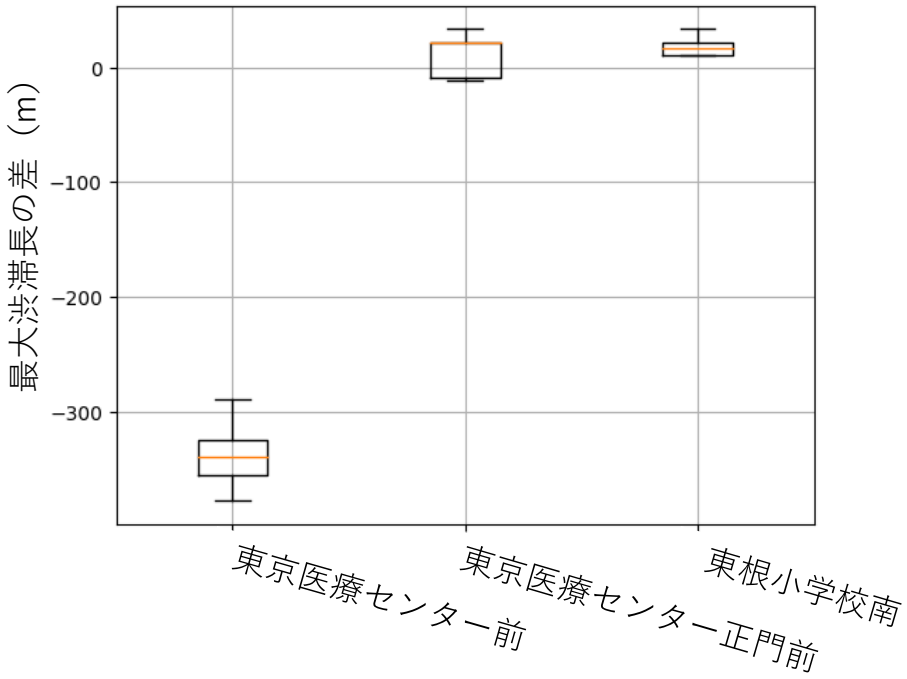

図 4.35　同シード値における最大渋滞長の差の分布

次に，対応のあるｔ検定により，両制御手法に対する最大渋滞長の差が有意なものなのかどうかを確かめた．検定の結果，「東京医療センター前」では，（t = -37.987，df = 18，p = 3.014-11），「東京医療センター正門前」では，（t = 2.0226，df = 18，p = 0.0738），「東根小学校南」では，（t = 7.236，df = 18，p = 4.888-05）となり，「東京医療センター前」以外は0.1%に有意であった．したがって，統計的には，信号制御手法の違いによる最大渋滞長への影響が無いとは言い切れず，特に需要率の高い交差点においては，提案信号制御手法を用いた方が，最大渋滞長が小さくなる可能性が高いと言える．

図 4.36 はそれぞれの制御手法における，5 分間平均旅行時間の推移を制御開始からシミュレーション終了まで表したものである．「東京医療センター前」においては，従来信号制御手法における 5 分間平均旅行時間が圧倒的に大きな値で推移している．「東京医療センター正門前」では，両制御手法に対する 5 分間平均旅行時間の推移はほとんど同じであった．「東根小学校南」では，制御開始から 250 秒ほどまでは，従来信号制御手法の方が，大きな値で推移していたが，それ以降は，値が下がり，相対的に提案信号制御手法の方が大きな値で推移している．これは，従来信号制御手法において，需要率の高い「東京医療センター前」にほとんどの車両が捌け残ってしまい，下流交差点にほとんど車両が行かなくなったことで自由流に近い状態となったことが原因であると考えられる．以上より，需要率の高い交差点において，両制御手法による平均旅行時間への影響は明確にあると言える．それより下流側交差点において，相対的に提案信号制御手法による平均旅行速度が大きくなる傾向があると言える．また，提案信号制御手法では，需要率に寄らず，平均旅行時間はほぼ横ばいの推移をたどると言える．

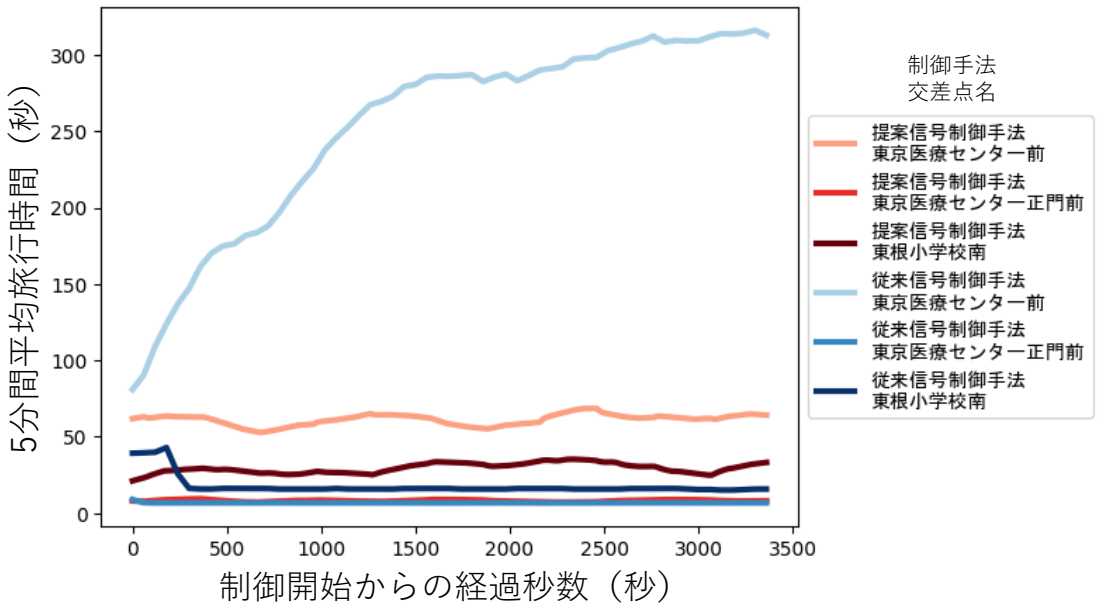

図 4.36　5 分間平均旅行時間の推移

次のページの**図 4.37** は，各交差点において，発生交通量のシード値を揃えた時の，各制御手法に対する平均旅行時間の差について，箱ひげ図で示したものである．差の求め方は，以下の通りである．

（シード値 X における平均旅行速度の差）＝
（シード値 X で提案信号制御手法の平均旅行速度）－
（シード値 X で従来信号制御手法の平均旅行速度）

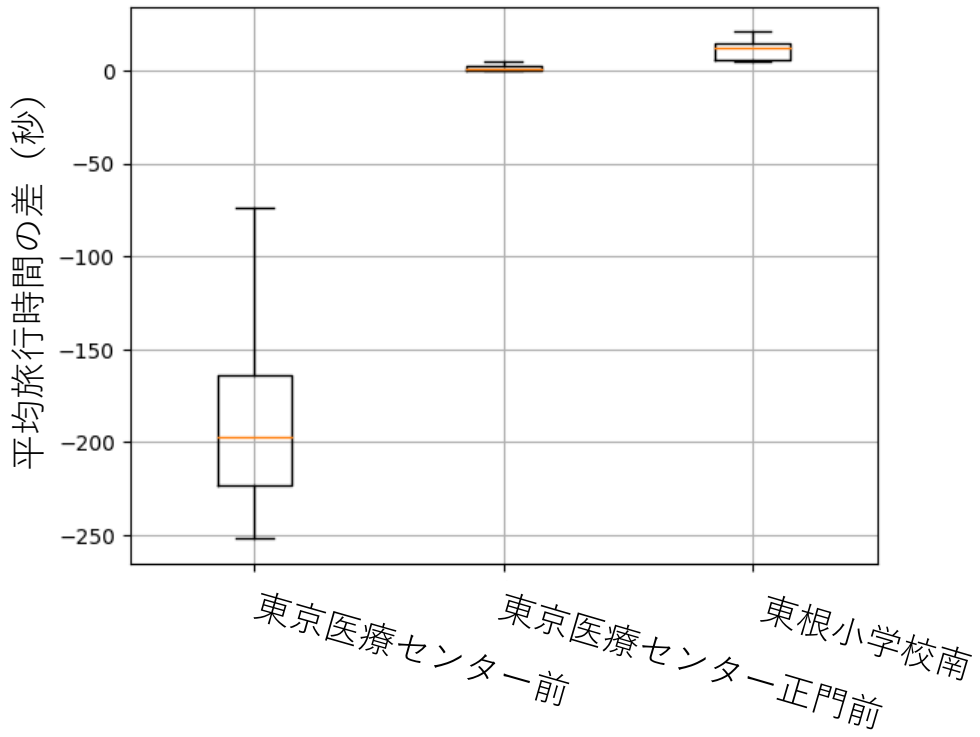

図 4.37　同シード値における平均旅行時間の差の分布

おおよそ，最大渋滞長の時と同様の傾向が見られる．ただし，「東京医療センター前」においては差のばらつきが大きい．これは，従来信号制御手法による平均旅行時間のばらつきが大きいことが原因であると考えられる．

次に，対応のあるｔ検定により，両制御手法による平均旅行時間の差が有意なものなのかどうかを確かめた．検定の結果，「東京医療センター前」では（t = −11.319，df = 18，p = 1.265e−06），「東京医療センター正門前」では（t = 2.750，df = 18，p = 0.022），「東根小学校南」では（t = 6.029，df = 18，p = 0.0002）となった．全交差点において，5%に有意であった．したがって，制御手法の違いによる平均旅行速度への影響はないとは言い切れず，特に需要率の高い交差点では，提案信号制御手法を用いた方が，平均旅行時間が小さくなる可能性が高いと言える．

4.7.6　信号制御パラメータと制御効果の推移

提案信号制御手法により実行された信号制御パラメータと制御効果の関係を確かめるために，これらの推移を見てみる．諸設定条件は**表 4.28** の通りである．

表 4.28　諸設定条件

ネットワーク構成	駒沢通り(4.2.1 参照)
10 分間平均発生交通量（台） （平均に対し，ポアソン分布を仮定した 確率分布で 10 分おきに変動）	107（混雑時交通量）
車種構成	小型車のみ
プローブ車混入率（%）	30
希望速度（km/h）	40（規制速度）
従方向の青時間スプリット	最低でも 0.4 を確保
シミュレーション時間	6:50～9:30（制御開始は 8:30～）
発生交通量のシード値	103
車両発生間隔のシード値	1000
探索アルゴリズム	Nelder-Mead
探索初期値	従来信号制御手法による 信号制御パラメータが 10 サイクル分継続

結果を次ページ以降の**図 4.38～図 4.43** に示した．**図 4.38～図 4.40** は発生交通量ごとに 5 分間総遅れ時間とサイクル長・青時間の推移を表したものである．これを見ると，発生交通量が変わると，サイクル長・青時間が変動していることが分かる．発生交通量が大きくなった場合は，サイクル長が長くなり，それに伴って各方向の青時間も長くなる．一方で発生交通量が少なくなった場合は，逆に，サイクル長が短くなり，それに伴って各方向の青時間も短くなる．総遅れ時間との関係については，サイクル長や青時間が総遅れ時間の大小に追従して，変化している傾向は見られない．これは，p.70 で述べたように，十分に最小の総遅れ時間を探索しきれていない可能性が考えられる．以上より，交通需要の大小に追従して，信号制御パラメータが変化する傾向はあると言えるが，総遅れ時間に対する追従の傾向は見られないと言える．

図 4.41～図 4.43 は，発生交通量ごとに，5 分間総遅れ時間と青時間スプリットの推移を表したものである．これを見ると，発生交通量や総遅れ時間の大小に関わらず，一定の青時間スプリットが継続されていることが分かる．したがって，探索の結果，変化している変数はサイクル長のみであると言える．原因はやはり，十分に最小の総遅れ時間を探索しきれていない可能性が考えられる．

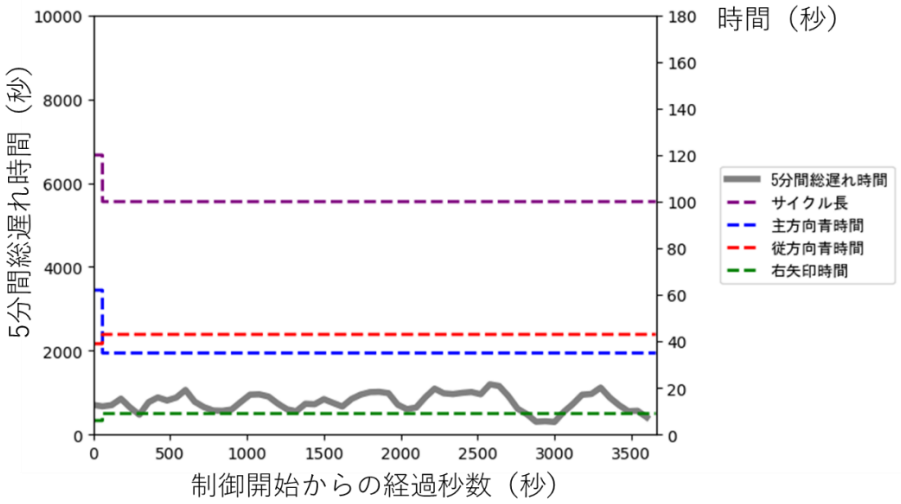

図 4.38　発生交通量(107 台/10mim)におけるサイクル長・青時間の推移

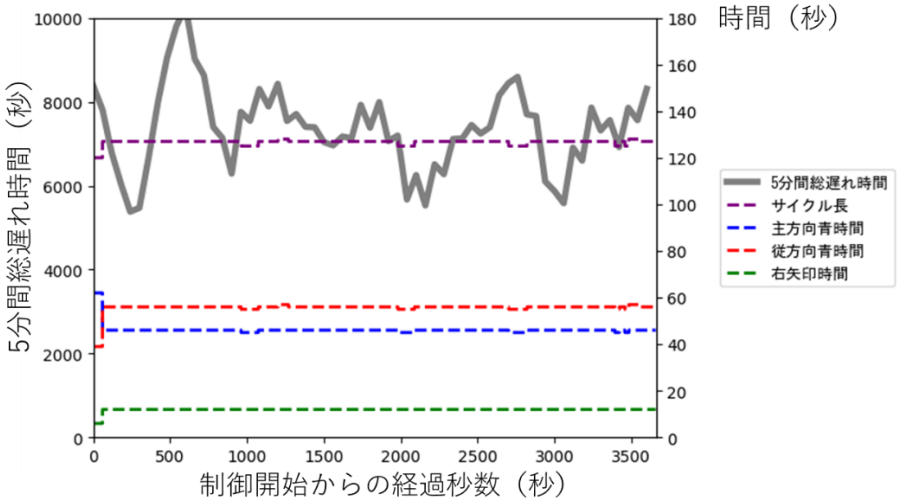

図 4.39　発生交通量(214 台/10mim)におけるサイクル長・青時間の推移

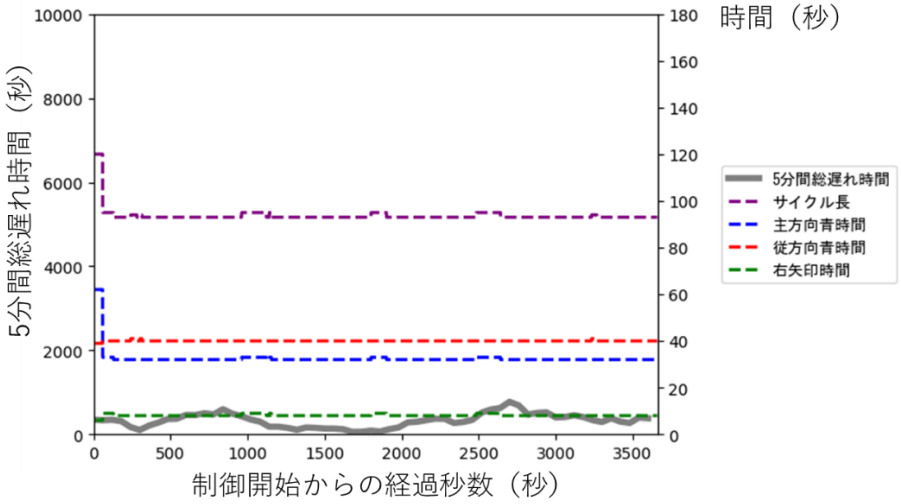

図 4.40　発生交通量(53 台/10mim)におけるサイクル長・青時間の推移

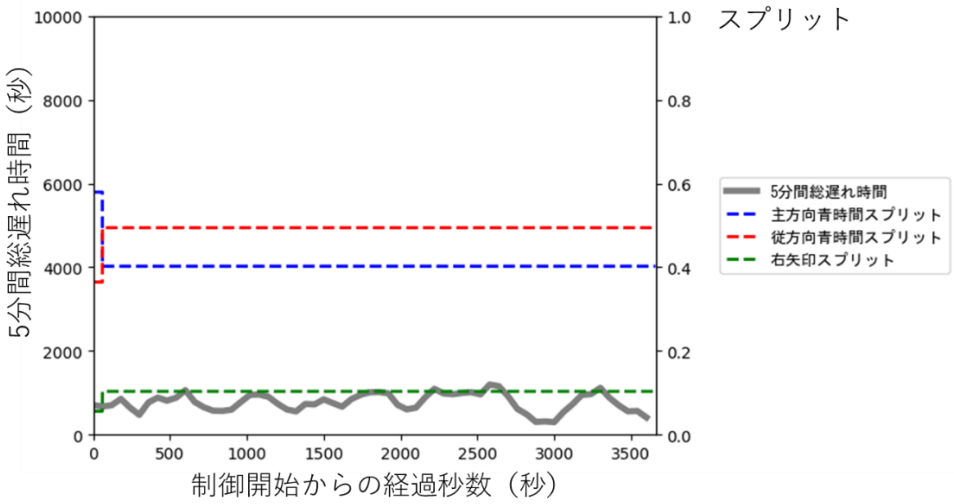

図 4.41　発生交通量(107 台/10mim)における青時間スプリットの推移

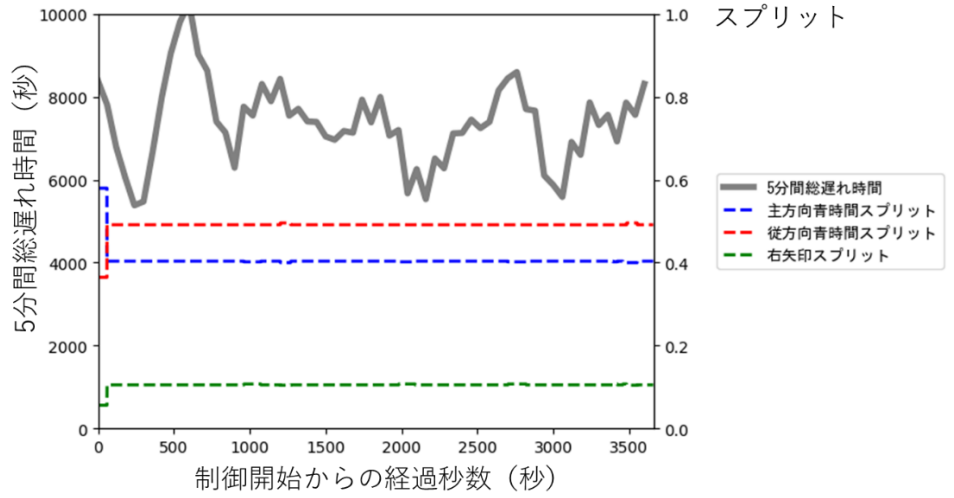

図 4.42　発生交通量(214 台/10mim)における青時間スプリットの推移

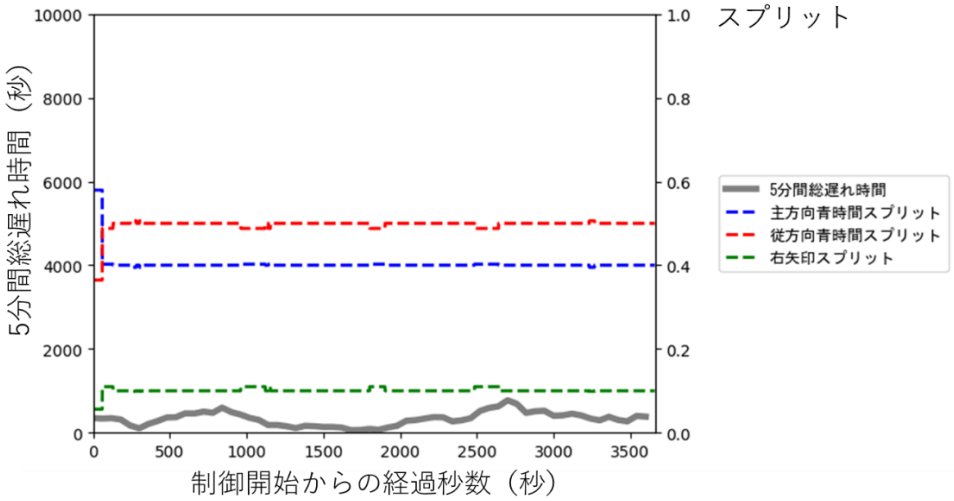

図 4.43　発生交通量(53 台/10mim)における青時間スプリットの推移

4.8　修士論文審査会後の追加分析

図 4.30，**図 4.32** において，探索前後の予想総遅れ時間にほとんど変化がない点や，**図 4.38**～**図 4.43** において，信号制御パラメータにほとんど変化がない点を踏まえ，探索部分のプログラムコードを再度見直した．その結果，初期値の与え方に変更を加えることによって，上記の問題を解決することができた．具体的には，70 変数を 1 セットのみを初期値として与えていたところを，70 変数を 71 セットだけ初期解候補群として与えるようにした．また，記述した式に間違いがあり，想定していた初期値の値とは異なっていたため，修正を加えた．**図 4.44** は，初期解候補群を与えた場合の，両信号制御手法における，5 分間総遅れ時間の平均値の推移と，探索の前後における予測総遅れ時間の推移を，制御開始からシミュレーション終了まで表したものである．なお，初期値に関する修正以外の諸設定条件は，**表 4.26** および**表 4.27** に等しい．

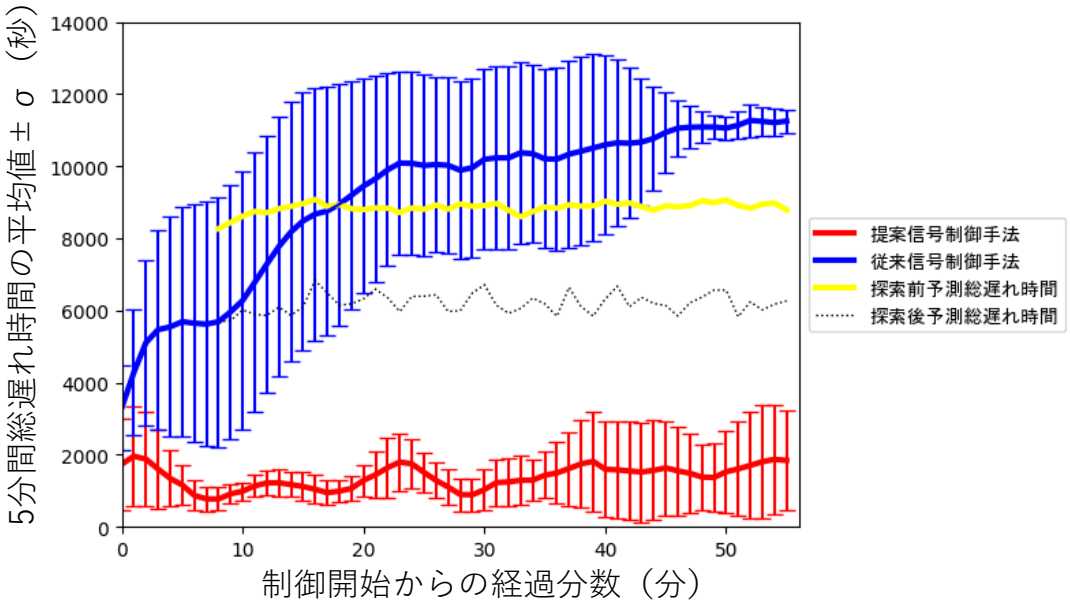

図 4.44　　各制御手法に対する 5 分間総遅れ時間の推移

これを見ると，探索後の予測総遅れ時間や，実際の 5 分間総遅れ時間自体は**図 4.30** のものとほとんど変わりがないが，探索前の予測総遅れ時間がおよそ 2200 秒だけ大きくなった．すなわち，探索前後の予測総遅れ時間の差は，修正前より明らかに大きくなった．次のページの**図 4.45** は，探索前後における予測総遅れ時間の差の分布をパレート図で表したものである．これを見ると，探索によって，1000 秒以下の短縮となる場合もあるものの，多くが 2000 秒以上短縮することができている．以上より，見直し後は，大幅に探索性能を改善することができたと言える．

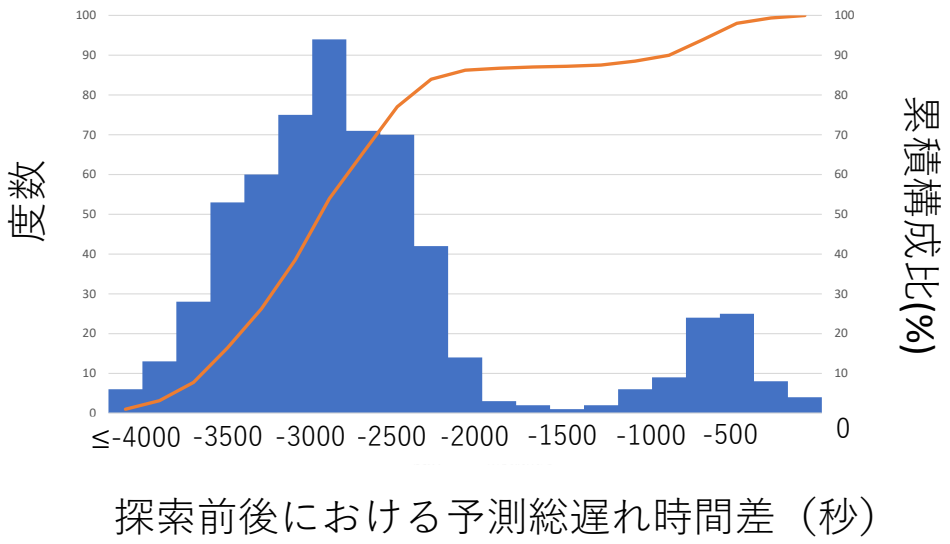

図 4.45　探索前後における予測総遅れ時間の差の分布

図 4.46 は，あるシード値における，5 分間総遅れ時間と 12 分後の予測総遅れ時間の推移，及び，実行された信号制御パラメータの推移を表したものである．これを見ると，信号制御パラメータが変動していることが分かる．しかし，5 分間総遅れ時間や予測総遅れ時間の推移と信号制御パラメータの推移に明確な関係があるとは言い難い．明確とは言えないが，12 分後予測総遅れ時間が比較的大きな値になった場合に，直近サイクル長が小さくなり，かつ，主方向青時間スプリットが従方向と比べて，小さくなるといった傾向がわずかながらに見られる．本来，予測総遅れ時間が大きくなる場合，主方向の青時間スプリットは大きくなることが望ましいと考えられるが，**図 4.46** では，これとは逆の傾向が見られる．今後は，この原因を追究する必要がある．

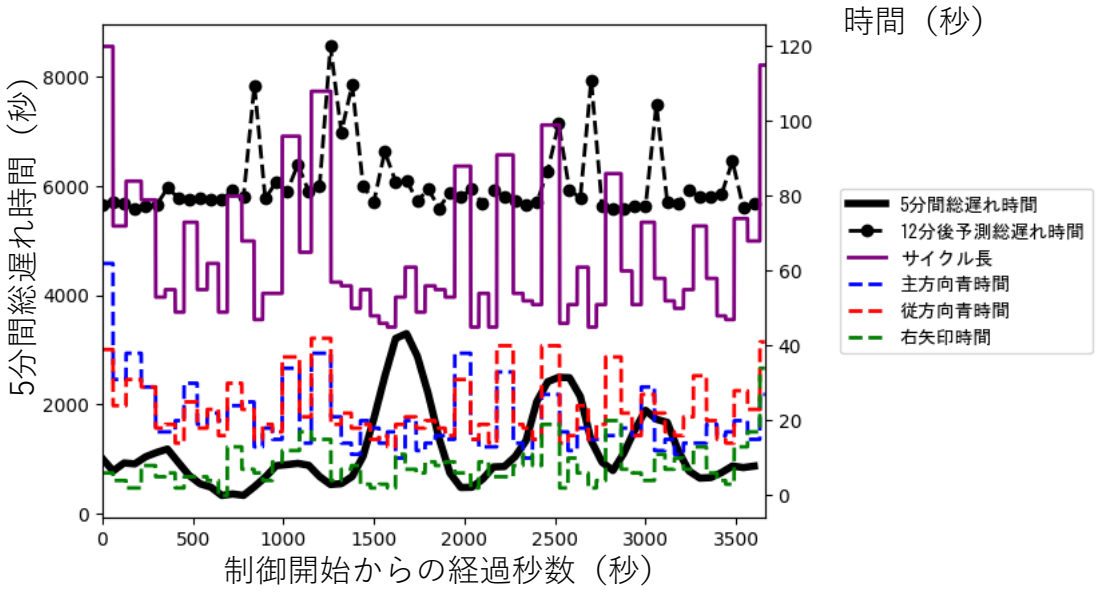

図 4.46　5 分間総遅れ時間・予測総遅れ時間及びサイクル長・青時間の推移

4.9　まとめ

これまでのシミュレーション評価による結果をまとめる．**表 4.29** は，それぞれの条件下において見られた傾向と有意差から判断した影響の有無，課題をまとめたものである．

表 4.29　シミュレーション評価のまとめ

条件	傾向	影響	課題
異なる探索アルゴリズム	局所解探索の方が制御効果○	○	どの探索アルゴリズムでも制御効果は横ばい
異なる初期値	前サイクルパラメータの方が制御効果○	×	両初期値とも制御効果は横ばい
異なる交通需要	少ない交通需要ほど制御効果○	○	交通需要増加により大きく制御効果落ちる
異なるプローブ車混入率	学習した混入率に近いほど制御効果○	×	混入率変動により制御効果が落ちる可能性あり
従来信号制御手法	提案信号制御手法の方が明確に制御効果○	○	提案信号制御の制御効果が横ばい
信号制御パラメータ	予測総遅れ時間が大きいとサイクル長→大，主方向青時間スプリット→小		総遅れ時間に適切に追従した信号制御パラメータを算出できているとは言えない

第５章　結論

5.1　本研究の結論 84
5.2　本研究の課題 84

第 5 章　結論

5.1　本研究の結論

本研究では，伝送遅れと混入率の低さの問題を踏まえた上で，プローブ情報を活用した予測遅れ時間に基づく信号制御手法を構築し，シミュレーション評価を行うことを目的として，様々な条件下における制御効果を定量的に示し，当該手法の課題や適用要件についての整理を行うことができた．プローブ情報に基づいた，将来の全車両総遅れ時間を高精度に予測することによって，既往研究で挙げられてきた，プローブ情報の伝送遅れと混入率の低さの問題を解決できると仮定して，信号制御手法の構築を行った．シミュレーション評価を行い，この制御手法による制御効果を定量的に示した結果，従来信号制御手法と比べて制御効果が小さく，さらには，プローブ車混入率の影響も小さいことが分かった．したがって，提案信号制御手法を用いることによって，上述のプローブ情報の問題を考慮することができると言える．提案信号制御手法を実用化することができれば，車両感知器を使った現行の交通応答制御における問題解決に繋がり，より交通状況に適した最適な信号制御を行うことができると考える．本研究の成果が，プローブ情報を活用した信号制御を推進する一助となることを願っている．

5.2　本研究の課題

1 つ目は，提案信号制御手法に対する総遅れ時間がほぼ横ばいになったことである．信号制御の目標は，総遅れ時間を小さくすることである．そのため，時間経過するごとに総遅れ時間がゼロへ近づくことが理想である．総遅れ時間が横ばいになっている原因として考えられるのが，最小の予測総遅れ時間を十分に探索しきれていないといった探索性能の問題と AI 予測精度の問題がある．探索性能の問題としては，不十分な探索時間・変数の多さ・初期値の与え方などが挙げられる．探索時間に関しては，本研究で用いた交通シミュレーションのシステム上，最大でも 30 秒としている．収束するまでに，通常，数時間～数日かかることを考えると，30 秒は極めて短い．対策として，信号制御と並行して常時探索を行うようにすることが考えられる．変数の多さに関しては，できる限り少なくなるように工夫する必要がある．そのためには，サイクル長や青時間スプリットを共通にするなどといった制約条件を設けることが考えられる．制約条件によって犠牲となる制御効果と，探索不十分により犠牲となる制御効果のバランスを見ながら，制約条件を設定することが重要であると言える．また，より高性能なコンピュータを用いて計算時間を高速にすることや，より適切な探索アルゴリズムを模索することも必要であると考えられる．また，初期値によって探索結果が変わる傾向があるため，初期値は慎重に選定を行う必要がある． AI 予測精度に関しては，総遅れ時間が大きくなるほど，あるいは，小さくなるほど，誤差が大きくなるといった問題がある．これに対しては，総遅れ時間が小さい場合と大きい場合のデータを拡充して AI に学習させることや，ニューラルネットワークの構造をさらに複雑にすることなどが考えられる．また，総遅れ時間に適切に追従した信号制御パラメータが実行されているとは言い難いことも，総遅れ時間が横ばいになっている原因の 1 つであると考えられる．これは，探索性能か AI 予測精度か，或いはその他のことが起因しているのかどうか慎重に分析を深めていく必要がある．

２つ目は，対向・交差方向の発生交通を考慮していない点や，3 交差点しか制御対象に入れていない点である．本研究では，信号制御手法の構築に主眼を置いたため，交通シミュレーションの設定条件はできるだけ単純にすることで，制御が上手くいかなかった場合に，その原因を特定しやすくするようにした．そのため，発生交通は 1 方向のみしか設定しておらず，また，3 交差点しか制御対象としていない．実際の交差点で信号制御を行うにあたっては，主方向・従方向の両方の交通状況を鑑みた上で，複数交差点において系統的に，最適な制御を行う必要がある．そのため，実際の状況に最適な信号制御手法の構築・シミュレーション評価を行うためには，交通シミュレーションにおいて，対向・交差方向の発生交通や，より多くの交差点を含んだネットワークの設定を行う必要がある．

参考文献

1) 関達也，島津利行，和智誠，榊原肇，大口敬：プローブ情報を活用した信号制御の見直しについて，交通工学論文集，第 8 巻，第 1 号，pp.31-38，2022.1
2) 吉岡利也，榊原肇，テンハーゲンロビン，ローコウスキステファン，大口敬：プローブカーデータを用いた信号制御パラメータ算出手法，生産研究，74 巻 1 号，pp.115-122，2022
3) 長島靖，服部理，小林雅文：プローブ情報の活用による信号制御高度化，SEI テクニカルレビュー，第 184 号，pp.40-pp.43，2014.1
4) 西内裕晶，吉井稔雄：簡易車両感知器とプローブカーを用いた信号制御システム　～システムの構築と道路工事実施時の片側交互通行区間への適用～，交通工学第 39 巻，4 号，p.46，2004
5) 花房比左友，飯島護久，堀口良太：リアルタイム信号制御アルゴリズムのためのプローブ情報を利用した遅れ時間評価，第 8 回 ITS シンポジウム，2009
6) 塚田悟之：プローブ情報を活用した信号制御定数パターン見直し支援システムの構築，情報システム学会誌，Vol.14，No.2，pp.65-78，2018.12
7) 斉藤進，堀井健一郎，依田照彦：多目標最適化手法へのニューラルネットワークの応用，土木学会論文集，No. 537/I-35，pp.277-289，1996. 4
8) 株式会社アイ・トランスポート・ラボ：AVENUE 紹介ホームページ，http://www.i-transportlab.jp/products/avenue
9) 一般社団法人　交通工学研究会：交通シミュレーション活用のススメ，2012.1
10) 一般社団法人　交通工学研究会：平面交差の計画と設計　基礎編　－計画・設計・交通信号制御の手引－，2018.11
11) Francois Chollet，巣籠悠輔：Python と Keras によるディープラーニング，マイナビ出版，2018.5
12) 舟橋尚平，小根山裕之，柳原正実：都市間高速道路サグ部の交通状態遷移に着目した事故・渋滞発生確率予測モデル，首都大学東京大学院修士論文，2019
13) 一般財団法人 道路交通情報通信システムセンター：公式ホームページ，https://www.vics.or.jp/

謝辞

本論文を執筆するにあたり，多くの方々よりご指導とご協力を賜りました．この場をお借りして御礼申し上げます．

小根山先生には，研究生活において，様々な場面でサポートいただきました．研究で行き詰ったときにメールをすれば，休日平日問わずにすぐさま打ち合わせをセッティングしてくださったり，研究ゼミにおいて，自分では気づかなかった視点からご指摘を頂いたりしました．おかげさまで，無事，修士論文の執筆まで至ることができました．今後，もしかすると一緒にお仕事をさせていただくことがあるかもしれませんが，その時は是非ともよろしくお願いいたします．3 年間本当にありがとうございました．柳原先生には，特に交通シミュレーションやプログラミングにおいて，多々サポートいただきました．初めは右も左も全く分からなかった私に，わかりやすく指南してくださりつつも，自分自身で考える余地も与えてくださったため，普通の人よりはプログラムを扱えるようになったと自負しております．この力は将来も生かせると思っています．3 年間本当にありがとうございました．石倉先生には，合同ゼミにおいて，貴重なご意見・ご質問を頂き，自身の研究を少し離れた視点から見つめ直すきっかけをくださいました．また，先生は昔バイクに乗られていたということで，バイク談義をしたことが思い出に残っています．「バイクかっこいいね」と言っていただけてとても嬉しかったです．3 年間本当にありがとうございました．

同期の長井さん・小松香貴君・奏太君は，授業や研究，就活などの学業から，旅行や食事，銭湯などといった娯楽まで，3 年間を共に過ごした戦友のような存在でした．あなた方が居なければ，もしかすると途中で心が折れていたかもしれません．来年度からはそれぞれ別の進路を歩むこととなりますが，またいつでも食事やら銭湯でも行きましょう．3 年間（小松香貴君に関しては 6 年間）本当にありがとうございました．

Dear Sumardi, I have supported you in daily life for 2 year as tutor. Through conversation with you, I am proud to say that my English skill has been clearly improved. You also gave me opportunity to do what I might have to do in the future, such as procedure for entering elementary school. I was glad to experience these. I hope my support will help Sumardi to have a fruitful life in Japan. Thank you very much for 2 years.

Rizky さんと Kumar とは，母国のことや日本での生活について，研究室でたくさん話しました．お二人は今後も日本で過ごすとお聞きしているので，食事にでも行って，またたくさん話しましょう．3 年間本当にありがとうございました．

Ｍ１の高松君とは，バイク談義で盛り上がって発表資料の提出が遅れたり，距離ガバツーリングに行ったりしました．これに懲りず，またツーリングに行きましょう．Ｍ１の高垣君，川田君，鈴木君，栗原君，Ｂ４の伊藤君，奈良岡君，橘川君，加登君とは，研究室で駄弁ったり，飲み会や鍋二郎会で楽しいひと時を過ごさせていただきました．中間発表等でまた楽しい話を聞かせてください．本当にありがとうございました．

最後にこれまで支えてくれた家族をはじめ，先輩方，友人など関わってくださったすべての方に感謝の意を表します．本当にありがとうございました．

附録

提案信号制御手法プログラムコード..... 88
交通状況データ作成プログラムコード..... 113
総遅れ時間算出プログラムコード..... 120
AI 入力データセット作成プログラムコード 124
LSTM ニューラルネットワーク構築・学習プログラムコード 126
従来信号制御手法プログラムコード..... 133

提案信号制御手法－1

```
import csv
from datetime import datetime, timedelta
import pandas as pd
import numpy as np

def signal_emu2(input,sim_time):
    from scipy.optimize import minimize
    import math
    import itertools
    from tensorflow.keras.models import Sequential, model_from_json
    import csv
    from csv import reader
    import sys
    import copy

    cross_split = 0.4
    minimizefunc = 2 #1 の場合は差分進化法，2 の場合はネルダーミード法,3 の場合は準ニュートン法
    initial = 2 #1 の場合は前信号情報，2 は従来手法による信号情報
    output_dir = 'C:/FeedbackSignalControl/FORAVENUE/AVENUE プロジェクト/KomazawaAvn.cases/C_1102145806590/AIoutput/'
    model = model_from_json(open(output_dir + "model.json", 'r').read()) # 保存したモデル構造の読み込み
    model.load_weights(output_dir + "weight.hdf5")  # 保存した学習済みの重みを読み込み

    with open(output_dir + "SeachResult.csv",'w',newline='') as a:
        writer = csv.writer(a)
        low = [100]*71
        low = [str(n) for n in low]
        writer.writerow(low)
```

提案信号制御手法－2

```
    def cycle(a):
        y = []
        for i in range(len(a)):
            y.append((40-180)/(1+math.exp(0.0000000001*a[i]))+180)

        return y

    def delta_cycle(b):
        y = []
        for i in range(len(b)):
            y.append(((-3)-3)/(1+math.exp(0.0000000001*b[i]))+3)

        return y

    def split(c,cross_split):
        y = []
        for i in range(len(c)):
            y.append((0.2-(1-cross_split))/(1+math.exp(0.0000000001*c[i]))+(1-
cross_split))

        return y

    def split_1(d,cross_split):
        y = []
        for i in range(len(d)):
            y.append((0.2-(1-cross_split))/(1+math.exp(0.0000000001*d[i]))+(1-
cross_split))

        return y
```

提案信号制御手法－3

```
    def split_arrow(e,split,cross_split):
        y = []
        for i in range(len(e)):
            y.append((0-((1-cross_split)-
           split[i]))/(1+math.exp(0.0000000001*e[i]))+((1-cross_split)-
           split[i]))

        return y

    def objective_fuc(x,args): #入力は 70 変数
        from datetime import datetime
        import pandas as pd
        import numpy as np

        TransmissionDelay = 3
        with open('C:/FeedbackSignalControl/FORAVENUE/AVENUE プロジェクト
        /KomazawaAvn.cases/C_1102145806590/AIoutput/SeachResult.csv','r',newlin
        e='') as f:
            s = reader(f)
            s = list(s)
            minOfDelay = float(s[-1][0])



        with open('C:/FeedbackSignalControl/FORAVENUE/AVENUE プロジェクト
        /KomazawaAvn.cases/C_1102145806590/AIoutput/SeachBestResult.csv','r',ne
        wline='') as m:
            s = reader(m)
            s = list(s)
            MostminOfDelay = float(s[-1][0])
        with open('C:/FeedbackSignalControl/FORAVENUE/AVENUE プロジェクト
        /KomazawaAvn.cases/C_1102145806590/AIoutput/SeachEndtime.csv','r',newli
        ne='') as g:
            t = reader(g)
            t = list(t)
            endtime = datetime.strptime(t[0][0], '%Y-%m-%d %H:%M:%S')
            endtime_30sec = datetime.strptime(t[0][1], '%Y-%m-%d %H:%M:%S')
```

提案信号制御手法－4

```
        data = copy.copy(args)
        cyclelist = cycle(x[0:10])
        deltacycle1 = delta_cycle(x[10:20])
        deltacycle2 = delta_cycle(x[20:30])

        split1 = split_1(x[30:40],cross_split)
        split1_arrow = split_arrow(x[40:50],split1,cross_split)
        split2 = split(x[50:60],cross_split)
        split3 = split(x[60:70],cross_split)

        cyclelist = np.array(cyclelist)
        deltacycle1 = np.array(deltacycle1)
        deltacycle2 = np.array(deltacycle2)

        cyclelist2 = cyclelist + deltacycle1
        cyclelist3 = cyclelist + deltacycle2

        effective1 = []
        effective2 = []
        effective3 = []

        for i in range(10):
            effective1.append(round(cyclelist[i]) - 13)
            effective2.append(round(cyclelist[i]) + round(deltacycle1[i]) - 3)
            effective3.append(round(cyclelist[i]) + round(deltacycle2[i]) -
           10)

        cyclelist = [i + 13 for i in effective1]
        cyclelist2 = [i + 3 for i in effective2]
        cyclelist3 = [i + 10 for i in effective3]
```

提案信号制御手法－5

```
        green1_1 = []
        green1_2 = []
        green1_3 = []
        green2_1 = []
        green2_2 = []
        green3_1 = []
        green3_2 = []
        for i in range(10):
            green1_1.append(round(effective1[i] * split1[i]))
            green1_2.append(round(effective1[i] * split1_arrow[i]))
            green1_3.append(effective1[i] - green1_1[i] - green1_2[i])

            green2_1.append(round(effective2[i] * split2[i]))
            green2_2.append(effective2[i] - green2_1[i])

            green3_1.append(round(effective3[i] * split3[i]))
            green3_2.append(effective3[i] - green3_1[i])
        cycle1 = sum(data[44+TransmissionDelay:45+TransmissionDelay,9:12][0])
       + 13 #最後に届いた信号情報を取得
        cycle2 = sum(data[44+TransmissionDelay:45+TransmissionDelay,13:15][0])
       + 3
        cycle3 = sum(data[44+TransmissionDelay:45+TransmissionDelay,16:18][0])
       + 10
        lug1   = data[44+TransmissionDelay:45+TransmissionDelay,12:13][0][0]
        offset2 = data[44+TransmissionDelay:45+TransmissionDelay,15:16][0][0]
        offset3 = data[44+TransmissionDelay:45+TransmissionDelay,18:19][0][0]
        lug2 = lug1 - offset2
        lug3 = lug2 - offset3
        GREEN1_1 = data[44+TransmissionDelay:45+TransmissionDelay,9:10][0][0]
        GREEN1_2 = data[44+TransmissionDelay:45+TransmissionDelay,10:11][0][0]
        GREEN1_3 = data[44+TransmissionDelay:45+TransmissionDelay,11:12][0][0]
        GREEN2_1 = data[44+TransmissionDelay:45+TransmissionDelay,13:14][0][0]
        GREEN2_2 = data[44+TransmissionDelay:45+TransmissionDelay,14:15][0][0]
        GREEN3_1 = data[44+TransmissionDelay:45+TransmissionDelay,16:17][0][0]
        GREEN3_2 = data[44+TransmissionDelay:45+TransmissionDelay,17:18][0][0]
```

提案信号制御手法－6

```
        lug1list = []
        offset2list = []
        offset3list = []
        green1_1list = []
        green1_2list = []
        green1_3list = []
        green2_1list = []
        green2_2list = []
        green3_1list = []
        green3_2list = []
        j = 0
        k = 0
        l = 0
        for i in range(15 - TransmissionDelay):
            #print(i)
            lug1 = lug1 + 60

            if lug1 >= cycle1:
                lug1 = lug1 - cycle1
                if j >= 10:
                    cycle1 = cyclelist[9]
                    GREEN1_1 = green1_1[9]
                    GREEN1_2 = green1_2[9]
                    GREEN1_3 = green1_3[9]

                else:
                    cycle1 = cyclelist[j]
                    GREEN1_1 = green1_1[j]
                    GREEN1_2 = green1_2[j]
                    GREEN1_3 = green1_3[j]

                j += 1
```

提案信号制御手法－7

```
            lug2 = lug2 + 60
            if lug2 >= cycle2:
                lug2 = lug2 - cycle2
                if k >= 10:
                    cycle2 = cyclelist2[9]
                    GREEN2_1 = green2_1[9]
                    GREEN2_2 = green2_2[9]
                else:
                    cycle2 = cyclelist2[k]
                    GREEN2_1 = green2_1[k]
                    GREEN2_2 = green2_2[k]
                k += 1
            lug3 = lug3 + 60
            if lug3 >= cycle3:
                lug3 = lug3 - cycle3
                if i >= 10:
                    cycle3 = cyclelist3[9]
                    GREEN3_1 = green3_1[9]
                    GREEN3_2 = green3_2[9]
                else:
                    cycle3 = cyclelist3[l]
                    GREEN3_1 = green3_1[l]
                    GREEN3_2 = green3_2[l]
                l += 1
            lug1list.append(lug1)
            offset2list.append(lug1 - lug2)
            offset3list.append(lug2 - lug3)
            green1_1list.append(GREEN1_1)
            green1_2list.append(GREEN1_2)
            green1_3list.append(GREEN1_3)
            green2_1list.append(GREEN2_1)
            green2_2list.append(GREEN2_2)
            green3_1list.append(GREEN3_1)
            green3_2list.append(GREEN3_2)
```

提案信号制御手法－8

```
        chagesignalinfo = []
        for i in range(15 - TransmissionDelay):
            chagesignalinfo.append(green1_1list[i])
            chagesignalinfo.append(green1_2list[i])
            chagesignalinfo.append(green1_3list[i])
            chagesignalinfo.append(lug1list[i])

            chagesignalinfo.append(green2_1list[i])
            chagesignalinfo.append(green2_2list[i])
            chagesignalinfo.append(offset2list[i])

            chagesignalinfo.append(green3_1list[i])
            chagesignalinfo.append(green3_2list[i])
            chagesignalinfo.append(offset3list[i])

        for i in range(15 - TransmissionDelay):
            data[45+TransmissionDelay+i:46+TransmissionDelay+i,9:][0] =
           chagesignalinfo[0+10*i:10+10*i]

        with open('C:/FeedbackSignalControl/FORlearndata/AVENUE プロジェクト
        /KomazawaAvn.cases/C_1102145806590/AIoutput/Xmean.csv','r',newline='')
        as f:
            Xmean = reader(f)
            Xmean = list(Xmean)
            Xmean = np.array(Xmean)
            Xmean = Xmean.astype(np.float)
        with open('C:/FeedbackSignalControl/FORlearndata/AVENUE プロジェクト
        /KomazawaAvn.cases/C_1102145806590/AIoutput/Xstd.csv','r',newline='')
        as g:
            Xstd = reader(g)
            Xstd = list(Xstd)
            Xstd = np.array(Xstd)
            Xstd = Xstd.astype(np.float)
```

提案信号制御手法－9

```
        for i in range(60):
            for j in range(19):
                data[i:i+1,j:j+1][0] -= Xmean[0][j]
                data[i:i+1,j:j+1][0] /= Xstd[0][j]
        data = data.reshape([1, 60,19])
        predict = model.predict(data,verbose=0)
        predict = predict[0][0]

        minlow = []
        if predict < minOfDelay:
            minlow.append(predict)
            minlow.extend(x)
            with open('C:/FeedbackSignalControl/FORAVENUE/AVENUE プロジェクト
            /KomazawaAvn.cases/C_1102145806590/AIoutput/SeachResult.csv','a',ne
            wline='') as f:
                writer = csv.writer(f)
                minlow = [str(n) for n in minlow]
                writer.writerow(minlow)
        Mostminlow = []
        if predict < MostminOfDelay:
            Mostminlow.append(predict)
            Mostminlow.extend(x)

            with open('C:/FeedbackSignalControl/FORAVENUE/AVENUE プロジェクト
            /KomazawaAvn.cases/C_1102145806590/AIoutput/SeachBestResult.csv','w
            ',newline='') as m:
                writer = csv.writer(m)
                Mostminlow = [str(n) for n in Mostminlow]
                writer.writerow(Mostminlow)

            with open('C:¥FeedbackSignalControl¥FORAVENUE¥AVENUE プロジェクト
            ¥KomazawaAvn.cases¥C_1102145806590¥AIoutput¥BestFlag.csv','w') as
            writer  :
                writer.write(str(1))
```

提案信号制御手法－10

```
            with open('C:¥FeedbackSignalControl¥FORAVENUE¥AVENUE プロジェクト
            ¥KomazawaAvn.cases¥C_1102145806590¥AIoutput¥FoundBesttime.csv','w')
            as writer  :
                writer.write(str(sim_time-1

        import datetime
        if endtime <= datetime.datetime.now() or endtime_30sec <=
        datetime.datetime.now():
            raise ValueError('まだ探索途中ですが終了時間です!')
        else:
            return predict

    import random
    import scipy.optimize as opt
    import datetime
    from datetime import timedelta
    TransmissionDelay = 3
    args = copy.copy(input) #送信データ入力箇所
    with
    open('C:/FeedbackSignalControl/FORAVENUE/InitialValue.csv','r',newline='')
    as m:
            s = reader(m)
            green = list(s)
            green = green[0]
            green = [int(s) for s in green]

    def cycle_rev(a):
        y = []
        for i in range(len(a)):
            y.append(1000000000*math.log(((40-180)/(a[i]-180))-1))
        return y

    def delta_cycle_rev(b):
        y = []
        for i in range(len(b)):
            y.append(1000000000*math.log((((-3)-3)/(b[i]-3))-1))
        return y
```

提案信号制御手法－11

```
def split_rev(c,cross_split):
    y = []
    for i in range(len(c)):
        y.append(1000000000*math.log(((0.2-(1-cross_split))/(c[i]-(1-
       cross_split)))-1))
    return y

def split_1_rev(d,cross_split):
    y = []
    for i in range(len(d)):
        y.append(1000000000*math.log(((0.2-(1-cross_split))/(d[i]-(1-
       cross_split)))-1))
    return y

def split_arrow_rev(e,split,cross_split):
    y = []
    for i in range(len(e)):
        y.append(1000000000*math.log(((0-((1-cross_split)-split))/(e[i]-
       ((1-cross_split)-split)))-1))
    return y

ini1 = cycle_rev([green[0]]*10)
ini2 = delta_cycle_rev([0]*10)
ini3 = delta_cycle_rev([0]*10)
ini4 = split_1_rev([green[1]/(green[0]-13)]*10, cross_split)
ini5 = split_arrow_rev([green[2]/(green[0]-13)]*10, green[1]/(green[0]-
       13), cross_split)
ini6 = split_rev([green[4]/(green[0]-3)]*10, cross_split)
ini7 = split_rev([green[6]/(green[0]-10)]*10, cross_split)
```

提案信号制御手法－12

```
    if initial == 2:
        xini = []
        xini.extend(ini1)
        xini.extend(ini2)
        xini.extend(ini3)
        xini.extend(ini4)
        xini.extend(ini5)
        xini.extend(ini6)
        xini.extend(ini7)
    else:
        #前の信号制御情報が続いた時
        lastcycle1 =
                    sum(args[44+TransmissionDelay:45+TransmissionDelay,9:12][0
                    ]) + 13
        lastcycle2 =
                    sum(args[44+TransmissionDelay:45+TransmissionDelay,13:15][
                    0]) + 3

        lastcycle3 =
                    sum(args[44+TransmissionDelay:45+TransmissionDelay,16:18][
                    0]) + 10

        if lastcycle1 >=180:
            lastcycle1 = 179
        elif lastcycle1 <= 40:
            lastcycle1 = 41
        else:
            lastcycle1 = lastcycle1

        lastdelta1 = lastcycle2 - lastcycle1

        if lastdelta1 >= 3:
            lastdelta1 = 2
        elif lastdelta1 <= -3:
            lastdelta1 = -2
        else:
            lastdelta1 = lastdelta1
```

提案信号制御手法－13

```
        lastdelta2 = lastcycle3 - lastcycle1
        if lastdelta2 >= 3:
            lastdelta2 = 2
        elif lastdelta2 <= -3:
            lastdelta2 = -2
        else:
            lastdelta2 = lastdelta2

        lastgreen1 = args[44+TransmissionDelay:45+TransmissionDelay,9:10][0][0]
        lastarrow  =
                    args[44+TransmissionDelay:45+TransmissionDelay,10:11][0][0
                    ]
        lastgreen2 =
                    args[44+TransmissionDelay:45+TransmissionDelay,13:14][0][0
                    ]
        lastgreen3 =
                    args[44+TransmissionDelay:45+TransmissionDelay,16:17][0][0
                    ]
        green1split = lastgreen1/(lastcycle1-13)

        if green1split >=0.6 :
            green1split = 0.59
        elif green1split <= 0.2:
            green1split = 0.21
        else :
            green1split = green1split

        arrowsplit = lastarrow/(lastcycle1-13)

        if arrowsplit >= (1-cross_split - green1split):
            arrowsplit = (1-cross_split - green1split) - 0.01
        elif arrowsplit == 0:
            arrowsplit - 0.01
        else:
            arrowsplit = arrowsplit
```

提案信号制御手法－14

```
        green2split = lastgreen2/(lastcycle2-3)

        if green2split >=0.6 :
            green2split = 0.59
        elif green2split <= 0.2:
            green2split = 0.21
        else :
            green2split = green2split

        green3split = lastgreen3/(lastcycle3-10)

        if green3split >=0.6 :
            green3split = 0.59
        elif green3split <= 0.2:
            green3split = 0.21
        else :
            green3split = green3split


        ini1 = cycle_rev([lastcycle1]*10)
        ini2 = delta_cycle_rev([lastdelta1]*10)
        ini3 = delta_cycle_rev([lastdelta2]*10)
        ini4 = split_1_rev([lastgreen1/(lastcycle1-13)]*10, cross_split)
        ini5 = split_arrow_rev([0.009]*10, lastgreen1/(lastcycle1-13),
               cross_split)
        ini6 = split_rev([0.59]*10, cross_split)
        ini7 = split_rev([0.59]*10, cross_split)

        xini = []
        xini.extend(ini1)
        xini.extend(ini2)
        xini.extend(ini3)
        xini.extend(ini4)
        xini.extend(ini5)
        xini.extend(ini6)
        xini.extend(ini7)
```

提案信号制御手法－15

```
    initial_simplexList=[]
    for simplex in range( len(xini)+1 ):
        x=[]

        for dim in range( len( xini ) ):
            x.append( xini[dim] + random.uniform(-10000000000,10000000000))
        initial_simplexList.append( x )

    cycle1_sub = sum(args[44+TransmissionDelay:45+TransmissionDelay,9:12][0])
                 + 13 #最後に届いた信号情報を取得
    cycle2_sub = sum(args[44+TransmissionDelay:45+TransmissionDelay,13:15][0])
                 + 3
    cycle3_sub = sum(args[44+TransmissionDelay:45+TransmissionDelay,16:18][0])
                 + 10
    lug1_sub   = args[44+TransmissionDelay:45+TransmissionDelay,12:13][0][0]
    offset2_sub = args[44+TransmissionDelay:45+TransmissionDelay,15:16][0][0]
    offset3_sub = args[44+TransmissionDelay:45+TransmissionDelay,18:19][0][0]
    lug2_sub = lug1_sub - offset2_sub
    lug3_sub = lug2_sub - offset3_sub

    seachtime = min([cycle1_sub-lug1_sub,cycle2_sub-lug2_sub,cycle3_sub-
                 lug3_sub])
    endtime = datetime.datetime.now().replace(microsecond=0) +
              timedelta( seconds = seachtime)
    endtime_30sec = datetime.datetime.now().replace(microsecond=0) +
              timedelta( seconds = 20)

    with open(output_dir + "SeachEndtime.csv",'w',newline='') as b:
        writer = csv.writer(b)
        writer.writerow([str(endtime),str(endtime_30sec)])
```

提案信号制御手法－16

```
    try :
        import scipy.optimize as opt
        if minimizefunc == 1:
            result= opt.dual_annealing(objective_fuc,
            args = (args,),
            bounds = [[-10000000000,10000000000]]*70,
            x0 = xini
            )
            raise ValueError('探索終了です!')
        elif minimizefunc == 2:
           result =  opt.minimize(objective_fuc,
                      args = (args,),
                      x0 = xini,
                      method='Nelder-Mead',
                      options = {'initial_simplex' : initial_simplexList},
                      )
            raise ValueError('探索終了です!')
        else:
            result =  opt.minimize(objective_fuc,
                      args = (args,),
                      x0 = xini,
                      bounds = [[-10000000000,10000000000]]*70,
                      method='BFGS'
                      )
            raise ValueError('探索終了です!')
```

提案信号制御手法－17

```
    except :
        import csv #ここから senddata 作成
        from csv import reader

        FoundminFlag = pd.read_csv('C:¥FeedbackSignalControl¥FORAVENUE¥AVENUE
                       プロジェクト
                       ¥KomazawaAvn.cases¥C_1102145806590¥AIoutput¥BestFlag.cs
                       v',header=None)
        FoundminFlag = FoundminFlag.iloc[0,0]
        FoundBestTime = pd.read_csv('C:¥FeedbackSignalControl¥FORAVENUE¥AVENUE
                       プロジェクト
                       ¥KomazawaAvn.cases¥C_1102145806590¥AIoutput¥FoundBestti
                       me.csv',header=None)
        FoundBestTime = FoundBestTime.iloc[0,0]
        if Method == 1 or ((sim_time-1) - FoundBestTime) == 720 & FoundminFlag
           == 0:
            output_dir = 'C:/FeedbackSignalControl/FORAVENUE/AVENUE プロジェクト
                          /KomazawaAvn.cases/C_1102145806590/AIoutput/'

            with open(output_dir + 'SeachResult.csv','r',newline='') as f:
                s = reader(f)
                s = list(s)
                x = s[-1][1:71]
                x = [float(s) for s in x]

            with open('C:¥FeedbackSignalControl¥FORAVENUE¥AVENUE プロジェクト
                 ¥KomazawaAvn.cases¥C_1102145806590¥AIoutput¥BestFlag.csv','w')
                 as writer  :
                writer.write(str(1))

            import math
            import numpy as np
```

提案信号制御手法－18

```
cyclelist = cycle(x[0:10])
deltacycle1 = delta_cycle(x[10:20])
deltacycle2 = delta_cycle(x[20:30])

split1 = split_1(x[30:40],cross_split)
split1_arrow = split_arrow(x[40:50],split1,cross_split)

split2 = split(x[50:60],cross_split)
split3 = split(x[60:70],cross_split)

cyclelist = np.array(cyclelist)
deltacycle1 = np.array(deltacycle1)
deltacycle2 = np.array(deltacycle2)

cyclelist2 = cyclelist + deltacycle1
cyclelist3 = cyclelist + deltacycle2

effective1 = []
effective2 = []
effective3 = []
for i in range(10):
    effective1.append(round(cyclelist[i]) - 13)
    effective2.append(round(cyclelist[i]) + round(deltacycle1[i])
   - 3)
    effective3.append(round(cyclelist[i]) + round(deltacycle2[i])
   - 10)

cyclelist = [i + 13 for i in effective1]
cyclelist2 = [i + 3 for i in effective2]
cyclelist3 = [i + 10 for i in effective3]
green1_1 = []
green1_2 = []
green1_3 = []
green2_1 = []
green2_2 = []
green3_1 = []
green3_2 = []
```

提案信号制御手法－19

```
            for i in range(10):
                green1_1.append(round(effective1[i] * split1[i]))
                green1_2.append(round(effective1[i] * split1_arrow[i]))
                green1_3.append(effective1[i] - green1_1[i] - green1_2[i])

                green2_1.append(round(effective2[i] * split2[i]))
                green2_2.append(effective2[i] - green2_1[i])

                green3_1.append(round(effective3[i] * split3[i]))
                green3_2.append(effective3[i] - green3_1[i])

            ForallSigInfo = []
            ForSecSigInfo1 = []
            ForSecSigInfo2 = []
            ForSecSigInfo3 = []
            print(cycle1_sub)
            print(lug1_sub)
            for i in range(int(cycle1_sub)-int(lug1_sub)-1):
                pregreen1 =
                            args[44+TransmissionDelay:45+TransmissionDelay,9:12
                            ][0].tolist()
                ForSecSigInfo1.append(pregreen1)
            for i in range(10):
                for j in range(int(cyclelist[i])):
                    ForSecSigInfo1.append([green1_1[i],green1_2[i],green1_3[i]]
                    )

            for i in range(int(cycle2_sub)-int(lug2_sub)-1):
                pregreen2 =
                            args[44+TransmissionDelay:45+TransmissionDelay,13:1
                            5][0].tolist()
                ForSecSigInfo2.append(pregreen2)
```

提案信号制御手法－20

```
            for i in range(10):
                for j in range(int(cyclelist2[i])):
                    ForSecSigInfo2.append([green2_1[i],green2_2[i]])

            for i in range(int(cycle3_sub)-int(lug3_sub)-1):
                pregreen3 =
                            args[44+TransmissionDelay:45+TransmissionDelay,16:1
                            8][0].tolist()
                ForSecSigInfo3.append(pregreen3)

            for i in range(10):
                for j in range(int(cyclelist3[i])):
                    ForSecSigInfo3.append([green3_1[i],green3_2[i]])

            for i in range((15-TransmissionDelay)*60):
                ForSecSigInfo1.append(ForSecSigInfo1[-1])
                ForSecSigInfo2.append(ForSecSigInfo2[-1])
                ForSecSigInfo3.append(ForSecSigInfo3[-1])

            for i in range((15-TransmissionDelay)*60):
                ForallSigInfo_sub = []
                ForallSigInfo_sub.extend(ForSecSigInfo1[i])
                ForallSigInfo_sub.extend(ForSecSigInfo2[i])
                ForallSigInfo_sub.extend(ForSecSigInfo3[i])
                ForallSigInfo.append(ForallSigInfo_sub)

            Ymean = pd.read_csv('C:/FeedbackSignalControl/FORlearndata/AVENUE プ
                    ロジェクト
                    /KomazawaAvn.cases/C_1102145806590/AIoutput/Ymean.csv',hea
                    der=None)
            Ymean = Ymean.iloc[0,0]

            Ystd = pd.read_csv('C:/FeedbackSignalControl/FORlearndata/AVENUE プ
                    ロジェクト
                    /KomazawaAvn.cases/C_1102145806590/AIoutput/Ystd.csv',head
                    er=None)
            Ystd = Ystd.iloc[0,0]
```

提案信号制御手法－21

```
        with open('C:/FeedbackSignalControl/FORAVENUE/AVENUE プロジェクト
                /KomazawaAvn.cases/C_1102145806590/AIoutput/SeachResult.cs
                v','r',newline='') as f:
            s = reader(f)
            s = list(s)
            ini_delay = float(s[1][0])
            ini_delay = ini_delay * Ystd + Ymean
            opt_delay = float(s[-1][0])
            opt_delay = opt_delay * Ystd + Ymean
            result = []
            result.append(ini_delay)
            result.append(opt_delay)

        seed =
               pd.read_csv('C:¥FeedbackSignalControl¥FORAVENUE¥seed.csv',he
               ader=None)
        seed = seed.iloc[0,0]

        with open('C:/FeedbackSignalControl/FORAVENUE/AVENUE プロジェクト
                  /KomazawaAvn.cases/C_1102145806590/AIoutput/Comparison_in
                  i_opt_{}.csv'.format(seed),'a',newline='') as g:
            writer = csv.writer(g)
            writer.writerow(result)

        return ForallSigInfo
```

提案信号制御手法－22

```
        else:
            output_dir = 'C:/FeedbackSignalControl/FORAVENUE/AVENUE プロジェクト
                         /KomazawaAvn.cases/C_1102145806590/AIoutput/'

            with open(output_dir + 'SeachBestResult.csv','r',newline='') as f:
                s = reader(f)
                s = list(s)
                x = s[-1][1:71]
                x = [float(s) for s in x]
            import math
            import numpy as np
            cyclelist = cycle(x[0:10])
            deltacycle1 = delta_cycle(x[10:20])
            deltacycle2 = delta_cycle(x[20:30])

            split1 = split_1(x[30:40],cross_split)
            split1_arrow = split_arrow(x[40:50],split1,cross_split)

            split2 = split(x[50:60],cross_split)
            split3 = split(x[60:70],cross_split)

            cyclelist = np.array(cyclelist)
            deltacycle1 = np.array(deltacycle1)
            deltacycle2 = np.array(deltacycle2)

            cyclelist2 = cyclelist + deltacycle1
            cyclelist3 = cyclelist + deltacycle2
           effective1 = []
            effective2 = []
            effective3 = []

            for i in range(10):
                effective1.append(round(cyclelist[i]) - 13)
                effective2.append(round(cyclelist[i]) + round(deltacycle1[i])
               - 3)
                effective3.append(round(cyclelist[i]) + round(deltacycle2[i])
               - 10)
```

提案信号制御手法－23

```
            cyclelist = [i + 13 for i in effective1]
            cyclelist2 = [i + 3 for i in effective2]
            cyclelist3 = [i + 10 for i in effective3]
            green1_1 = []
            green1_2 = []
            green1_3 = []
            green2_1 = []
            green2_2 = []
            green3_1 = []
            green3_2 = []
            for i in range(10):
                green1_1.append(round(effective1[i] * split1[i]))
                green1_2.append(round(effective1[i] * split1_arrow[i]))
                green1_3.append(effective1[i] - green1_1[i] - green1_2[i])

                green2_1.append(round(effective2[i] * split2[i]))
                green2_2.append(effective2[i] - green2_1[i])

                green3_1.append(round(effective3[i] * split3[i]))
                green3_2.append(effective3[i] - green3_1[i])
            ForallSigInfo = []
            ForSecSigInfo1 = []
            ForSecSigInfo2 = []
            ForSecSigInfo3 = []

            for i in range(int(cycle1_sub)-int(lug1_sub)-1):
                pregreen1 =
                            args[44+TransmissionDelay:45+TransmissionDelay,9:12
                            ][0].tolist()
                ForSecSigInfo1.append(pregreen1)

            for i in range(10):
                for j in range(int(cyclelist[i])):
                    ForSecSigInfo1.append([green1_1[i],green1_2[i],green1_3[i]]
                    )
```

提案信号制御手法－24

```
            for i in range(int(cycle2_sub)-int(lug2_sub)-1):
                pregreen2 =
                            args[44+TransmissionDelay:45+TransmissionDelay,13:1
                            5][0].tolist()
                ForSecSigInfo2.append(pregreen2)
            for i in range(10):
                for j in range(int(cyclelist2[i])):
                    ForSecSigInfo2.append([green2_1[i],green2_2[i]])
            for i in range(int(cycle3_sub)-int(lug3_sub)-1):
                pregreen3 =
                            args[44+TransmissionDelay:45+TransmissionDelay,16:1
                            8][0].tolist()
                ForSecSigInfo3.append(pregreen3)
            for i in range(10):
                for j in range(int(cyclelist3[i])):
                    ForSecSigInfo3.append([green3_1[i],green3_2[i]])
            for i in range((15-TransmissionDelay)*60):
                ForSecSigInfo1.append(ForSecSigInfo1[-1])
                ForSecSigInfo2.append(ForSecSigInfo2[-1])
                ForSecSigInfo3.append(ForSecSigInfo3[-1])
            for i in range((15-TransmissionDelay)*60):
                ForallSigInfo_sub = []
                ForallSigInfo_sub.extend(ForSecSigInfo1[i])
                ForallSigInfo_sub.extend(ForSecSigInfo2[i])
                ForallSigInfo_sub.extend(ForSecSigInfo3[i])
                ForallSigInfo.append(ForallSigInfo_sub)
            Ymean = pd.read_csv('C:/FeedbackSignalControl/FORlearndata/AVENUE プ
                    ロジェクト
                    /KomazawaAvn.cases/C_1102145806590/AIoutput/Ymean.csv',hea
                    der=None)
            Ymean = Ymean.iloc[0,0]

            Ystd = pd.read_csv('C:/FeedbackSignalControl/FORlearndata/AVENUE プ
                    ロジェクト
                    /KomazawaAvn.cases/C_1102145806590/AIoutput/Ystd.csv',head
                    er=None)
            Ystd = Ystd.iloc[0,0]
```

提案信号制御手法－25

```
        with open('C:/FeedbackSignalControl/FORAVENUE/AVENUE プロジェクト
                  /KomazawaAvn.cases/C_1102145806590/AIoutput/SeachResult.c
                  sv','r',newline='') as f:
            s = reader(f)
            s = list(s)
            ini_delay = float(s[1][0])
            ini_delay = ini_delay * Ystd + Ymean

            opt_delay = float(s[-1][0])
            opt_delay = opt_delay * Ystd + Ymean
            result = []
            result.append(ini_delay)
            result.append(opt_delay)

        seed =
               pd.read_csv('C:¥FeedbackSignalControl¥FORAVENUE¥seed.csv',he
               ader=None)
        seed = seed.iloc[0,0]

        with open('C:/FeedbackSignalControl/FORAVENUE/AVENUE プロジェクト
                 /KomazawaAvn.cases/C_1102145806590/AIoutput/Comparison_ini
                 _opt_{}.csv'.format(seed),'a',newline='') as g:
            writer = csv.writer(g)
            writer.writerow(result)

        return ForallSigInfo
```

交通状況データ作成－1

```
import pandas as pd
import numpy as np
import csv
from datetime import datetime, timedelta
import sys


def TrafficConditionDataOutPut(year,month,day,hour,minute,second):
        pd.set_option('display.max_columns',19) #表示する列数を 18 にする
        df = pd.read_csv('C:¥FeedbackSignalControl¥FORAVENUE¥probe_data.csv')
        seed =
               pd.read_csv('C:¥FeedbackSignalControl¥FORAVENUE¥seed.csv',header
               =None)
        seed = seed.iloc[0,0]
        signaldf =
                   pd.read_csv('C:¥FeedbackSignalControl¥FORAVENUE¥SignalData¥s
                   ignal_data_{}.csv'.format(seed)) #csv ファイル読み込み
        df['Time'] = pd.to_datetime(df['Time']) #日時を標準日時に変換
        df = df.set_index('Time') #日時データをインデックスに変換

        signaldf['Time'] = pd.to_datetime(signaldf['Time']) #日時を標準日時に変換
        signaldf = signaldf.set_index('Time') #日時データをインデックスに変換


        linkIDs =
                   ['I_11_1102151814992_L','I_11_1102150954131_L','I_11_1102151
                   113637_L']
        signalIDs =
                   ['I_16_1102172048249','I_16_1102172049183','I_16_11021720503
                   46']
```

交通状況データ作成－２

```
        starttime = str(year) + '/' + str(month) + '/' + str(day) + '/ ' +
                    str(hour) + ':' + str(minute).zfill(2) + ':' +
                    str(second).zfill(2) #開始時刻
        starttime = pd.to_datetime(starttime)



        linklength = {"I_11_1102151814992_L":413.98,
                      "I_11_1102150954131_L":53.38,
                      "I_11_1102151113637_L":147.19}
        reguratedspeed = 11.1111 #単位は m/s(40km/h)

        RightZones = ['I_27_1102171611026','no','I_27_0912160033485']
        LeftZones = ['I_27_1102171607100','no','no']

        numofphase = {'I_16_1102172048249' : 8, 'I_16_1102172049183' : 3,
                      'I_16_1102172050346' : 6}

        traveltime = []
        averagedelaytime = []
        outflowvolume = []
        signalinfo = []
        dataline = []
        luglist = []
        lugdashlist = []

        j = 0
        for linkID in linkIDs: #リンクごとに平均遅れ時間と方向別流出台数を算出
                lnklen = linklength[linkID]
                leftturnvolume = 0
                rightturnvolume = 0
                throughvolume = 0
                alldata = df.query('linkID == @linkID') #指定したリンク ID のデータ
                                                        を抽出
                alldata = alldata[str(starttime - timedelta(minutes=1)):
                            str(starttime)]
                alldata = alldata.query('stoplinedist <= @lnklen')
                carID = alldata['carID'].unique() #carID を取得
```

交通状況データ作成－３

```
            For I in carID:
                    lnklen = linklength[linkID]
                    eachcardata = df.query('carID == @i') #取得した carID のデ
                                                          ータを抽出
                    eachcardata = eachcardata.query('linkID == @linkID')

                    if eachcardata.index.max() >=  starttime -
                      timedelta(minutes=1) and eachcardata.index.max()
                      <=  starttime:
                           eachcardata2 = df.query('linkID == @linkID and
                                          carID == @i and stoplinedist <=
                                          @lnklen and stoplinedist >= 0')
                           TRAVELTIME = eachcardata2.index.max()-
                                        eachcardata2.index.min() #旅行時間を
                                        算出
                           TRAVELTIME = TRAVELTIME.total_seconds() #秒数に
                                                                   変換
                           traveltime.append(TRAVELTIME)

                           if LeftZones[j] in
                               eachcardata['destinationID'][0] : #流出方向別
                               に旅行時間と走行距離をリストにまとめる
                                   leftturnvolume = leftturnvolume + 1

                           elif RightZones[j] in
                               eachcardata['destinationID'][0]:
                                   rightturnvolume = rightturnvolume + 1
                           else:
                                   throughvolume = throughvolume + 1
                    else:
                           continue

            outflowvolume.append(throughvolume)
            outflowvolume.append(leftturnvolume)
            outflowvolume.append(rightturnvolume)
```

交通状況データ作成－4

```
                LinkLengthList = [lnklen]*len(traveltime) #リンク長を要素とするリスト
作成
                ReglatedSpeedList = [reguratedspeed]*len(traveltime) #規制速度を
要素とするリスト作成

                LinkLengthList = np.array(LinkLengthList) #リスト同士の除算ができる
ようにする
                ReglatedSpeedList = np.array(ReglatedSpeedList)
                EachDelayTime  = traveltime-(LinkLengthList/ReglatedSpeedList)
                AVERAGEDELAYTIME = np.average(EachDelayTime)
                averagedelaytime.append(AVERAGEDELAYTIME)
                j = j+1

        for signalID in signalIDs:

                forcycle =[]
                eachsignalinfo = signaldf.query('signalID == @signalID')

                eachsignalinfo = eachsignalinfo[str(starttime -
                                timedelta(seconds=600)) : str(starttime)] #現
                                在時刻から 600 秒間だけ遡ってみる
                forphase = eachsignalinfo[str(starttime ) : str(starttime)]
                for i in range(int(numofphase[signalID])):
                        phase = forphase['phase'+str(i)]
                        phase = phase.iloc[-1]
                        forcycle.append(phase)
                        signalinfo.append(phase)
                cyclelength = sum(forcycle)
                onlyzero = eachsignalinfo.query('CurrentPhase == 0') #階梯番号
                                                                0 のみを抽出
                zeroendtime = onlyzero.index.max() #階梯番号0が終わる最新時刻を記
録
                zeroendtimesiginfo = eachsignalinfo.query('Time ==
               @zeroendtime') #階梯番号0が終わる最新時刻の信号情報を抽出
                phasezeroduration = zeroendtimesiginfo['phase0'] #階梯番号0が終
わる最新時刻の階梯番号0の秒数を記録
                phasezeroduration = phasezeroduration.iloc[-1]
```

交通状況データ作成－５

```
                for k in range(int(phasezeroduration)): #階梯番号0の秒数だけ 1 秒
ずつ遡ってみる
                    judgeline = eachsignalinfo[str(zeroendtime -
timedelta(seconds=int(k+1))):str(zeroendtime - timedelta(seconds=int(k+1)))]
                    if len(judgeline) == 0: #シミュレーション初期時間まで遡ってしまった
場合
                        judgeline = eachsignalinfo[str(zeroendtime -
                                    timedelta(seconds=int(k))):str(zeroendtime -
                                    timedelta(seconds=int(k)))]
                        previouscycleendtime = judgeline.index.max() #初期時刻を
記録し，この時点がサイクル開始とする
                        zerosttime = previouscycleendtime
                        break
                    if judgeline['CurrentPhase'].iloc[-1]  == 0 : #現在の階梯番号
が0ならもう 1 秒遡る
                        if k == phasezeroduration - 1:
                              for i in range(600):
                                    judgeline = eachsignalinfo[str(zeroendtime
                                                -
                                                timedelta(seconds=k+1+i)):str(z
                                                eroendtime -
                                                timedelta(seconds=int(k+1+i)))]

                                    if judgeline['CurrentPhase'].iloc[-1]  ==
                                                                          0 :
                                          continue
                                    else:
                                       previouscycleendtime =
                                       judgeline.index.max() #階梯番号が0でなくな
                                       った時刻を記録し，その 1 秒後が現サイクルの開始
                                       時刻とする
                                       zerosttime = previouscycleendtime +
                                                    timedelta(seconds=1)
                                       break
                              break
                        else:
                              continue
```

交通状況データ作成－6

```
                else:
                    previouscycleendtime = judgeline.index.max() #階梯番号
がOでなくなった時刻を記録し，その 1 秒後が現サイクルの開始時刻とする
                    zerosttime = previouscycleendtime +
                                 timedelta(seconds=1)
                    break

            lug = starttime - zerosttime #階梯番号O開始時刻～現在時刻の時間を
経過秒数とする
            lug = lug.total_seconds() #秒数に変換
            if lug > cyclelength and forcycle[0] == 0 :
                eachsignalinfo = signaldf.query('signalID == @signalID')
                eachsignalinfo = eachsignalinfo[str(starttime -
                                 timedelta(seconds=600)) : str(starttime)]
                                 #現在時刻から 600 秒間だけ遡ってみる

                onlyone = eachsignalinfo.query('CurrentPhase == 1') #階梯番
号 1 のみを抽出

                oneendtime = onlyone.index.max() #階梯番号 1 が終わる最新時刻
を記録
                oneendtimesiginfo = eachsignalinfo.query('Time ==
                                    @oneendtime') #階梯番号 1 が終わる最新時
                                    刻の信号情報を抽出
                phaseoneduration = oneendtimesiginfo['phase1'] #階梯番号 1
が終わる最新時刻の階梯番号 1 の秒数を記録
                phaseoneduration = phaseoneduration.iloc[-1]
                for k in range(int(phaseoneduration)): #階梯番号 1 の秒数だけ
1 秒ずつ遡ってみる
                    judgeline = eachsignalinfo[str(oneendtime -
                                timedelta(seconds=int(k+1))):str(oneendtime
                                - timedelta(seconds=int(k+1)))]

                    if judgeline['CurrentPhase'].iloc[-1]  == 1: #現在の階梯
番号が 1 ならもう 1 秒遡る
                        continue
```

交通状況データ作成－7

```
                    else:
                        previouscycleendtime = judgeline.index.max() #階梯
番号が 1 でなくなった時刻を記録し，その 1 秒後が現サイクルの開始時刻とする
                        onesttime = previouscycleendtime +
                                    timedelta(seconds=1)
                        break
                lug = starttime - onesttime #階梯番号 1 の開始時刻～現在時刻の
時間を経過秒数とする
                lug = lug.total_seconds() #秒数に変換
            luglist.append(lug)
            lugdash = lug - cyclelength
            lugdashlist.append(lugdash)
            if signalID == 'I_16_1102172048249':
                signalinfo.append(lug)
            elif signalID == 'I_16_1102172049183':
                offset1 = luglist[0] - luglist[1]
                offset2 = luglist[0] - lugdashlist[1]
                offset3 = lugdashlist[0] - luglist[1]
                offset4 = lugdashlist[0] - lugdashlist[1]
                offset83 = min([offset1,offset2,offset3,offset4], key=abs)
                signalinfo.append(offset1)

            else:
                offset1 = luglist[1] - luglist[2]
                offset2 = luglist[1] - lugdashlist[2]
                offset3 = lugdashlist[1] - luglist[2]
                offset4 = lugdashlist[1] - lugdashlist[2]
                offset46 = min([offset1,offset2,offset3,offset4], key=abs)
                signalinfo.append(offset1)


        dataline.append(str(starttime))
        dataline.extend(averagedelaytime)
        dataline.extend(outflowvolume)
        dataline.extend(signalinfo)



        return dataline
```

総遅れ時間算出－1

```
import pandas as pd
import numpy as np
import csv
from datetime import datetime, timedelta
import sys

def DelayTimeOutPut(year,month,day,hour,minute,second):
  pd.set_option('display.max_columns',19) #表示する列数を 18 にする
  df = pd.read_csv('C:¥FeedbackSignalControl¥FORAVENUE¥Allprobe_data.csv')
#csv ファイル読み込み
  df['Time'] = pd.to_datetime(df['Time']) #日時を標準日時に変換
  df = df.set_index('Time') #日時データをインデックスに変換
  linkIDs =
['I_11_1102151814992_L','I_11_1102150954131_L','I_11_1102151113637_L']

  starttime = str(year) + '/' + str(month) + '/' + str(day) + '/ ' + str(hour)
+ ':' + str(minute).zfill(2) + ':' + str(second).zfill(2) #開始時刻
  starttime = pd.to_datetime(starttime)
  collectioniterval = 5 #データ収集時間間隔
  linklength = {"I_11_1102151814992_L":413.98, "I_11_1102150954131_L":53.38,
"I_11_1102151113637_L":147.19}
  reguratedspeed = 11.1111 #単位は m/s(40km/h)
  RightZones = ['I_27_1102171611026','no','I_27_0912160033485']
  LeftZones = ['I_27_1102171607100','no','no']

  TotalDelayTime =[]
  TotalThrDelayTime =[]
  TotalRightDelayTime = []
  TotalLeftDelayTime = []
  dataline =[]
  j = 0
```

総遅れ時間算出－2

```
  for linkID in linkIDs: #リンクごとに総遅れ時間を算出
    alldata = df.query('linkID == @linkID') #指定したリンク ID のデータを抽出
    alldata = alldata[str(starttime -
                      timedelta(minutes=int(collectioniterval))):
                      str(starttime)]
    carID = alldata['carID'].unique() #carID を取得
    traveltime  = []
    Ttraveltime = []
    Ltraveltime = []
    Rtraveltime = []

    rundist =  []
    Trundist = []
    Lrundist = []
    Rrundist = []

    for i in carID:
        lnklen = linklength[linkID]
        eachcardata = alldata.query('carID == @i and stoplinedist <= @lnklen
       and stoplinedist >= 0') #取得した carID のデータを抽出

        if len(eachcardata.index)==0:
          continue

        minTime = eachcardata.index.min() #最小、最大の時刻を抽出
        maxTime = eachcardata.index.max()
        TRAVELTIME = maxTime-minTime #旅行時間を算出
        TRAVELTIME = TRAVELTIME.total_seconds()
        minStoplinedist = eachcardata['stoplinedist'].min() #停止線からの距離の
最小値を算出
        maxStoplinedist = eachcardata['stoplinedist'].max() #停止線からの距離の
最大値を算出
        RUNDIST = maxStoplinedist-minStoplinedist #走行距離を算出
```

総遅れ時間算出－3

```
        traveltime.append(TRAVELTIME) #旅行時間と走行距離をリストにまとめる
        rundist.append(RUNDIST)


        if LeftZones[j] in eachcardata['destinationID'][0] : #流出方向別に旅行時
            間と走行距離をリストにまとめる
            Ltraveltime.append(TRAVELTIME)
            Lrundist.append(RUNDIST)
        elif RightZones[j] in eachcardata['destinationID'][0]:
            Rtraveltime.append(TRAVELTIME)
            Rrundist.append(RUNDIST)
        else:
            Ttraveltime.append(TRAVELTIME)
            Trundist.append(RUNDIST)
    LinkLengthList = [lnklen]*len(traveltime) #リンク長を要素とするリスト作成
    ReglatedSpeedList = [reguratedspeed]*len(traveltime) #規制速度を要素とするリ
スト作成
    ThrLinkLengthList = [lnklen]*len(Ttraveltime) #リンク長を要素とするリスト作成(以
下 3 つは方向別に算出)
    ThrReglatedSpeedList = [reguratedspeed]*len(Ttraveltime) #規制速度を要素とす
るリスト作成(以下 3 つは方向別に算出)
    RightLinkLengthList = [lnklen]*len(Rtraveltime)
    RightReglatedSpeedList = [reguratedspeed]*len(Rtraveltime)

    LeftLinkLengthList = [lnklen]*len(Ltraveltime)
    LeftReglatedSpeedList = [reguratedspeed]*len(Ltraveltime)
    LinkLengthList = np.array(LinkLengthList) #リスト同士の除算ができるようにする
    ReglatedSpeedList = np.array(ReglatedSpeedList)
    ThrLinkLengthList = np.array(ThrLinkLengthList) #リスト同士の除算ができるよう
にする(以下 3 つは方向別に算出)
    ThrReglatedSpeedList = np.array(ThrReglatedSpeedList)
    RightLinkLengthList = np.array(RightLinkLengthList)
    RightReglatedSpeedList = np.array(RightReglatedSpeedList)
    LeftLinkLengthList = np.array(LeftLinkLengthList)
    LeftReglatedSpeedList = np.array(LeftReglatedSpeedList)
```

総遅れ時間算出－4

```
    EachDelayTime  = traveltime-
                     (LinkLengthList/ReglatedSpeedList)*(rundist/LinkLengthList
                     )
    TEachDelayTime = Ttraveltime-
                     (ThrLinkLengthList/ThrReglatedSpeedList)*(Trundist/ThrLink
                     LengthList) #流出方向別に各車両の遅れ時間を算出（直進）
    REachDelayTime = Rtraveltime-
                     (RightLinkLengthList/RightReglatedSpeedList)*(Rrundist/Rig
                     htLinkLengthList) #遅れ時間算出(右折)
    LEachDelayTime = Ltraveltime-
                     (LeftLinkLengthList/LeftReglatedSpeedList)*(Lrundist/LeftL
                     inkLengthList) #遅れ時間算出（左折）

    TotalDelayTime.append(sum(EachDelayTime))
    TotalThrDelayTime.append(sum(TEachDelayTime))
    TotalRightDelayTime.append(sum(REachDelayTime))
    TotalLeftDelayTime.append(sum(LEachDelayTime))
    j = j+1

dataline.append(str(starttime - timedelta(minutes=int(collectioniterval))))
dataline.append(sum(TotalDelayTime))
dataline.extend(TotalDelayTime)
dataline.extend(TotalThrDelayTime)
dataline.extend(TotalRightDelayTime)
dataline.extend(TotalLeftDelayTime)

return dataline
```

AI 入力データセット作成－1

```
import pandas as pd
import numpy as np
import csv
from datetime import datetime, timedelta
import sys

def CombineData(year,month,day,hour,minute,second):
    pd.set_option('display.max_columns',19) #表示する列数を 18 にする
    trafficdf =
                pd.read_csv(r'C:¥FeedbackSignalControl¥FORAVENUE¥TrafficCondit
                ionData.csv')

    trafficdf['Time'] = pd.to_datetime(trafficdf['Time']) #日時を標準日時に変換
    trafficdf = trafficdf.set_index('Time') #日時データをインデックスに変換
    starttime = starttime = str(year) + '/' + str(month) + '/' + str(day) + '/
' + str(hour) + ':' + str(minute).zfill(2) + ':' + str(second).zfill(2) #開始時
刻
    starttime = pd.to_datetime(starttime)
    dataline = []
    #changestarttime = starttime + timedelta(minutes=int(changeminutes))
    thirtymintrafficdata =  trafficdf[str(starttime - timedelta(minutes=29)) :
                            str(starttime)] #直近 30 分間のデータを抽出
    signalinfo = trafficdf[str(starttime) : str(starttime)] #現在の信号情報を抽出

    thirtymintrafficdata = pd.DataFrame(thirtymintrafficdata,columns =
['AverageDelayTime in 92','AverageDelayTime in 31','AverageDelayTime in
37','ThroughVol from 92','LeftVol from 92','RightVol from 92','ThroughVol from
31','ThroughVol from 37','RightVol from 37']) #交通状況データのみ抽出
    thirtymintrafficdata = thirtymintrafficdata.mean() #列ごとに平均

    signalinfo = pd.DataFrame(signalinfo, columns = ['phase0 at 49','phase2 at
49','phase5 at 49','lug at 49','phase0 at 83','phase2 at 83','offset seconds
at 83','phase0 at 46','phase3 at 46','offset seconds at 46'])
    signalinfo = signalinfo.to_numpy().tolist()
    allsiginfo = []
```

AI 入力データセット作成－2

```
    for i in range(len(signalinfo)):
        allsiginfo.extend(signalinfo[i])

    dataline.append(str(starttime))
    dataline.extend(thirtymintrafficdata)
    dataline.extend(allsiginfo)

    print(len(dataline))

    return dataline
```

LSTM ニューラルネットワーク構築・学習－1

```
import os
os.add_dll_directory(os.path.join(os.environ['CUDA_PATH'], 'bin'))
import cvlib
import warnings
warnings.filterwarnings('ignore')

import pandas as pd
from pandas import Series, DataFrame

from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

import numpy as np
import matplotlib.pyplot as plt

import sys
print(sys.path)

import keras
from keras.datasets import fashion_mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, LSTM
from tensorflow.keras.optimizers import RMSprop
from keras.callbacks import ModelCheckpoint
from tensorflow.keras import utils as np_utils
from tensorflow.keras.optimizers import Adam
from keras.layers import Dense, LSTM, RepeatVector, TimeDistributed
from keras import regularizers

import common

import glob

import csv
from csv import reader
```

LSTM ニューラルネットワーク構築・学習－2

```
def save_plot(history, output_dir):
    fig, ax = plt.subplots()
    ax.plot(history.history['mae'])
    ax.plot(history.history['val_mae'])
    ax.set_yticks(np.linspace(0, 1, 11))
    plt.xlabel('epoch')
    plt.ylabel('mae')
    plt.legend(['train', 'test'], loc='lower right')
    plt.savefig(output_dir+"history-mae.png")

    fig, ax = plt.subplots()
    ax.plot(history.history['loss'])
    ax.plot(history.history['val_loss'])
    ax.set_yticks(np.linspace(0, 5, 5))
    plt.xlabel('epoch')
    plt.ylabel('loss')
    plt.legend(['train', 'test'], loc='upper right')
    plt.savefig(output_dir+"history-loss.png")

seqence = False
dname = "agg"
if seqence: dname = "seq"

layer_count = 10
train_count = 19
batch_count = 9000
time_length = 60
xdata_length = 19
ydata_length = 1
node_count = 2
cluster_length = 100000

time_batch = 1

input_file_v = 'ForMakingSampleData'
input_file_l = 'Label'
```

LSTM ニューラルネットワーク構築・学習－3

```
import copy

#学習データ読み込み

file_list = glob.glob("C:¥FeedbackSignalControl¥FORlearndata¥AVENUE プロジェクト
¥KomazawaAvn.cases¥C_1102145806590¥TrainingData¥*")

input_values = pd.DataFrame()
input_labels = pd.DataFrame()

Ydata = []
nancount = 0

for i in range(int(batch_count)):
    train = pd.read_csv(file_list[i], header = None)
    train = pd.DataFrame(train)

    if train.isnull().values.sum() != 0 :
        nancount += 1
        continue

    INPUT_values1 = train.iloc[:, 1 :10]
    INPUT_values2 = train.iloc[:, 11:]
    INPUT_values  = pd.concat([INPUT_values1, INPUT_values2], axis=1)
    INPUT_labels = train.iloc[:, [10]]
    input_values = pd.concat([input_values, INPUT_values], axis=0)
    input_labels = pd.concat([input_labels, INPUT_labels], axis=0)

batch_count = batch_count - nancount

input_labels =
input_labels.values.reshape(batch_count,time_length,ydata_length)
input_values =
input_values.values.reshape(batch_count,time_length,xdata_length)
```

LSTM ニューラルネットワーク構築・学習－4

```
#学習用と検証用に分ける
x_train, x_test, y_train, y_test = train_test_split(input_values,
input_labels, test_size=0.2)

x_train = np.array(x_train)
x_test  = np.array(x_test)
y_train = np.array(y_train)
y_test  = np.array(y_test)


x_train = x_train.astype(np.float)
x_test = x_test.astype(np.float)
y_train = y_train.astype(np.float)
y_test = y_test.astype(np.float)

optimization = copy.copy(x_test)
#標準化
Xmean = np.mean(x_train, axis=(0, 1))
x_train -= Xmean

Xstd  = np.std(x_train, axis=(0, 1))
print(Xstd)
x_train /= Xstd

x_test  -= Xmean
x_test /= Xstd

Ymean = np.mean(y_train, axis=(0, 1))
y_train -= Ymean
Ystd  = np.std(y_train, axis=(0, 1))
y_train /= Ystd

y_test  -= Ymean
y_test /= Ystd
```

LSTM ニューラルネットワーク構築・学習－5

```
Y_train = []
Y_test  = []
for i in range(round(batch_count*0.8)):
    Y_train.append(y_train[i,59])
for j in range(round(batch_count*0.2)):
    Y_test.append(y_test[j,59])
Y_train = np.array(Y_train)
Y_test  = np.array(Y_test)
Y_train = Y_train.astype(np.float)
Y_test = Y_test.astype(np.float)
output_dir = 'C:/FeedbackSignalControl/FORlearndata/AVENUE プロジェクト
/KomazawaAvn.cases/C_1102145806590/AIoutput/'
with open(output_dir + 'Xmean.csv','w',newline='') as j:
    writer = csv.writer(j)
    writer.writerow(Xmean)

with open(output_dir + 'Xstd.csv','w',newline='') as k:
    writer = csv.writer(k)
    writer.writerow(Xstd)

with open(output_dir + 'Ymean.csv','w',newline='') as l:
    writer = csv.writer(l)
    writer.writerow(Ymean)

with open(output_dir + 'Ystd.csv','w',newline='') as m:
    writer = csv.writer(m)
    writer.writerow(Ystd)
```

LSTM ニューラルネットワーク構築・学習－6

```
#LSTM 層の作成
def create_lstm_network(seqence,time_length, xdata_length, node_count,
layer_count, cluster_length):
    model = Sequential()
    model.add(LSTM(node_count, input_shape=(time_length,
xdata_length),return_sequences=False))
    model.add(Dropout(0.2))
    model.add(Dense(1))
    model.summary()
    model.compile(optimizer=RMSprop(), loss="mean_squared_error",
metrics=['mae'])
    return model

model = create_lstm_network(seqence, time_length, xdata_length, node_count,
layer_count, cluster_length)

#LSTM 層を使って学習
#EPOC 繰り返し中に val_loss が改善した時に結果（重みパラメータ）を上書き保存
#結果的に途中の val_loss が最もよいデータが残る
cp = ModelCheckpoint(output_dir + "model-weight.hdf5", monitor='val_loss',
verbose=1, save_best_only=True)
history =
model.fit(x_train,Y_train,batch_size=100,epochs=200,verbose=1,validation_data=
(x_test,Y_test),initial_epoch=1,callbacks=[cp])

score = model.evaluate(x_test,Y_test,verbose=1)
open(output_dir  + "model.json","w").write(model.to_json()) # モデル構造の保存
model.save_weights(output_dir + "weight.hdf5") # 学習済みの重みを保存
```

LSTM ニューラルネットワーク構築・学習－7

```
import itertools
from tensorflow.keras.models import Sequential, model_from_json

model = model_from_json(open(output_dir + "model.json", 'r').read()) # 保存したモデル構造の読み込み
model.load_weights(output_dir + "weight.hdf5")  # 保存した学習済みの重みを読み込み
predict = model.predict(x_train,verbose=0) #これを scipy の目的関数とする
predict = predict.reshape(-1) #1 次元配列に変換
Y_train = Y_train.reshape(-1)
a, b = np.polyfit(Y_train,predict ,1)
plt.scatter(Y_train,predict)
predict = predict.reshape(len(predict),1)
Y_train = Y_train.reshape(len(Y_train),1)
predict = common.deep_map(str, predict)
np.savetxt(output_dir + "train-predict.csv", predict, delimiter=",",fmt="%s")
predict = model.predict(x_test,verbose=0)

predict = predict.reshape(-1) #1 次元配列に変換
Y_test = Y_test.reshape(-1)
c, d = np.polyfit(Y_test,predict, 1)
plt.scatter(Y_test,predict)
x = [-3,-2,-1,0,1,2,3]
y = x
plt.plot(x,y,color="black")
y2 = a * Y_train + b
plt.plot(Y_train,y2,color="red")
plt.text(-4,0,'y='+str(a)+'x+'+str(b))

predict = predict.reshape(len(predict),1)
Y_test = Y_test.reshape(len(Y_test),1)

predict = common.deep_map(str, predict)
np.savetxt(output_dir + "test-predict.csv", predict, delimiter=",",fmt="%s")

save_plot(history, output_dir)
```

従来信号制御手法－1

```
DemandRate1 = []
DemandRate1_arrow = []
DemandRate2 = []
DemandRate3 = []
AllDemandRate_1 = []
AllDemandRate_2 = []
AllDemandRate_3 = []

totaleffectivegreentimelist1 = []
totaleffectivegreentimelist2 = []
totaleffectivegreentimelist3 = []

greentimelist1 = []
subgreentimelist1 = []
arrowtimelist = []
greentimelist2 = []
subgreentimelist2 = []
greentimelist3 = []
subgreentimelist3 = []
cycle1 = []
cycle2 = []
cycle3 = []

for i in range(18):
    greentimelist1.append(62)
    subgreentimelist1.append(39)
    arrowtimelist.append(6)

    greentimelist2.append(66)
    subgreentimelist2.append(31)

    greentimelist3.append(84)
    subgreentimelist3.append(26)

    cycle1.append(120)
    cycle2.append(100)
    cycle3.append(120)
```

従来信号制御手法－2

```
for i in range(6):
    eachODVol =  AveVol/5
    DemandRate1.append((eachODVol*3) * 6/2000)
    DemandRate1_arrow.append(eachODVol*6/1800)
    CrossDemandRate1.append((cross_split/(1-cross_split))*(DemandRate1[i] +
DemandRate1_arrow[i]))

    DemandRate2.append((eachODVol*2) * 6/2000)
    CrossDemandRate2.append((cross_split/(1-cross_split))*DemandRate2[i])

    RightRate = (eachODVol/(eachODVol*2))*100
    correction = 100/((100-RightRate)+3.95*RightRate)
    DemandRate3.append((eachODVol) * 6/2000*correction)
    CrossDemandRate3.append((cross_split/(1-cross_split))*DemandRate3[i])

    AllDemandRate_1.append(DemandRate1[i] + CrossDemandRate1[i] +
DemandRate1_arrow[i])
    AllDemandRate_2.append(DemandRate2[i] + CrossDemandRate2[i])
    AllDemandRate_3.append(DemandRate3[i] + CrossDemandRate3[i])
    if ((1.5 * 13 + 5)/(1 - AllDemandRate_1[i])) < 40:
        cycle1.append(41)
        cycle1.append(41)

        cycle2.append(41)
        cycle2.append(41)

        cycle3.append(41)
        cycle3.append(41)
    elif ((1.5 * 13 + 5)/(1 - AllDemandRate_1[i])) > 180:
        cycle1.append(179)
        cycle1.append(179)

        cycle2.append(179)
        cycle2.append(179)

        cycle3.append(179)
        cycle3.append(179)
```

従来信号制御手法－3

```
    else:
        cycle1.append(round((1.5 * 13 + 5)/(1 - AllDemandRate_1[i])))
        cycle1.append(round((1.5 * 13 + 5)/(1 - AllDemandRate_1[i])))

        cycle2.append(round((1.5 * 13 + 5)/(1 - AllDemandRate_1[i])))
        cycle2.append(round((1.5 * 13 + 5)/(1 - AllDemandRate_1[i])))

        cycle3.append(round((1.5 * 13 + 5)/(1 - AllDemandRate_1[i])))
        cycle3.append(round((1.5 * 13 + 5)/(1 - AllDemandRate_1[i])))


    totaleffectivegreentimelist1.append(round(cycle1[18 + 2*i]) - 13) #サイクル長から黄，赤時間を引く=有効青時間を算出
    totaleffectivegreentimelist2.append(round(cycle2[18 + 2*i]) - 3)
    totaleffectivegreentimelist3.append(round(cycle3[18 + 2*i]) - 10)

    greentimelist1.append(round(totaleffectivegreentimelist1[i] *
DemandRate1[i]/AllDemandRate_1[i]))

    arrowtimelist.append(round(totaleffectivegreentimelist1[i] *
DemandRate1_arrow[i]/AllDemandRate_1[i]))

    subgreentimelist1.append(totaleffectivegreentimelist1[i] -
greentimelist1[18 + 2*i] - arrowtimelist[18 + 2*i])

    greentimelist2.append(round(totaleffectivegreentimelist2[i] *
DemandRate2[i]/AllDemandRate_2[i]))

    subgreentimelist2.append(totaleffectivegreentimelist2[i] -
greentimelist2[18 + 2*i])

    greentimelist3.append(round(totaleffectivegreentimelist3[i] *
DemandRate3[i]/AllDemandRate_3[i]))

    subgreentimelist3.append(totaleffectivegreentimelist3[i] -
greentimelist3[18 + 2*i])
```