M-010

AI 便乗サービスにおける車輌の乗り継ぎを導入したアルゴリズムの研究
A Study on Algorithms Introducing Vehicle Transfers in AI Ride-Sharing Services

山本 優汰[1)]　坂地 泰紀[1)]　野田 五十樹[1)]
Yuta Yamamoto　Hiroki Sakaji　Itsuki Noda

1　はじめに

最近，新たな公共交通機関として AI 便乗サービスが注目されている．AI 便乗サービスとは，AI 技術を用いて，ユーザの希望に応じて経路や到着時刻を変更しながら運行される交通手段である．AI 便乗サービスは，路線バスのように固定路線を持たないため，経路や到着時間の自由度が高く，また複数のお客さん同士が乗合いをするため，タクシーに比べ利用コストは小さくなる．すなわち，AI 便乗サービスは路線バスとタクシーのそれぞれの利点を両方兼ね備えた公共交通機関であるといえる．

AI 便乗サービスにおいて，お客さんが車輌を乗り継いで乗車地点から目的地へ向かう「車輌の乗り継ぎ」を導入することで，配車効率の改善が期待される．車輌の乗り継ぎの導入は，最適化の観点では探索空間の拡張と考えられ，1 台での配車よりもさらに良い解を得られる可能性がある．その一方で，配車のための計算量は大きくなるため，リアルタイムでの配車が可能な計算量に収まる範囲で探索空間を拡張することが重要である．本稿では，計算量を抑えた乗り継ぎアルゴリズムを提案し，マルチエージェントシミュレーションによって，車輌の乗り継ぎの有効性と有効な条件を分析する．また，その結果を踏まえ，実際に車輌の乗り継ぎを導入する場合には，どのような条件が効果的であるかも考察する．

2　SAVS

2.1　概要

AI 便乗サービスの一つに SAVS(Smart Access Vehicle Service)[1] があり，これまで多くの実証実験や市区町村への導入が行われてきた．SAVS は，リアルタイムに発生する乗車デマンドに対して即時に配車を行う交通サービスで，配車アルゴリズムには逐次最適挿入法 [2] というアルゴリズムが用いられている．

2.2　逐次最適挿入法

逐次最適挿入法の概要は以下の通りである．ユーザのデマンド (配車希望) が発生すると，随時以下の処理が行われる．

1. デマンドは乗車地点と降車地点の二つの経由地を持つ．各車輌は，割り当てられたデマンドを経由地点の配列として保持しており，その配列は経由する順番にソートされている．ここで，一度保持された経由地点の順序は変更されないものとする．
2. 各車輌は，各時点で保持する経由地の到着予定時刻を求める．
3. 新たにデマンドが発生したとき，図 1 に示すように，そのデマンドを各車輌の経由地配列の任意の位置に挿入し，全経由地点における挿入によって生じる遅延の総和を，2. の到着予定時刻を元にそれぞれ求める．求めた遅延の総和に新しく挿入したデマンドの達成予定時刻 (降車予定時刻) を加えたものをコストとする．すなわち，このコストは以下のように表される．

$$arrivalTime + delay \tag{1}$$

ここで $arrivalTime$ が発生した新たなデマンドの達成予定時刻，$delay$ が既に割り当てられたデマンドの遅延時間の総和とする．その後，コストが最小となる乗降地点の挿入位置のペアを，新たなデマンドの挿入位置の候補とする．ただし，デマンドの挿入により既存あるいは挿入したデマンドのいずれかの締切時間を過ぎてしまう場合は，候補から除外する．
4. すべての車両のコストを比較し，コストが最小となる車両にデマンドを割り当て，配車する．なお，いずれの車両においても受け入れ候補が見つからない場合，このデマンドは配車不可能とし，これをキャンセルと呼ぶ．

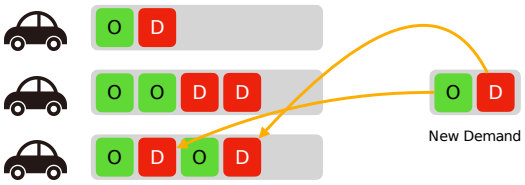

図 1　逐次最適挿入法の経由地の挿入

3　乗り継ぎを導入した配車アルゴリズム

3.1　乗り継ぎの概要

車輌の乗り継ぎとは，お客さんが目的地に行くまでに複数の車輌を乗り継いでいくことをいう．この乗り継ぎ車輌台数は，いくらでも増やすことは可能であるが，本研究では簡単のため 2 台までとする．また，車輌の乗り換えを行う地点の選択も非常に重要な要素であるが，本稿では簡単のため，乗り換え地点は一点の候補のみとする．

乗り継ぎを行うことによって，一台での配車を行う場合よりも効率が良くなると考えられる簡単な例を図 2 に示す．図 2 において，水色の経路が FirstSAV の未来の経路であり，緑色の経路が SecondSAV の未来の経路である．このとき，右上の深緑色の点で乗車して，左下の赤色の点で降車したいというデマンドが発生した場合，SAV の経路を大きく変更して一台で配車したり，3 台目の SAV で配車したりするよりも，FirstSAV で乗車地点から真ん中の黄色い点の乗り換え地点まで運び，

1)　北海道大学大学院 情報科学院 知能ソフトウェア研究室 Hokkaido University Graduate School of Information Science and Technology, Intelligent Software Laboratory

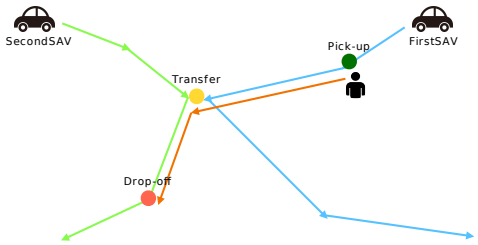

図 2　車輌の乗り継ぎの効率が良い例

SecondSAV で乗り換え地点から降車地点まで運んだほうが効率が良いことがわかる．以上のことから，2 台の車輌の未来の経路が，点を共有する (あるいは近い) 場合に，その点を乗り換え地点として，2 つの経路を横断するデマンドを処理する場合に乗り継ぎをした方が効率が良くなることがわかる．

以降は，乗り継ぎを行う仕組みや，生じる問題，アルゴリズムについて述べる．ここで，あるデマンドを乗り継ぎによる配車で処理する場合において，「お客さんを乗車地点から乗り換え地点まで送る SAV」と「お客さんを乗り換え地点から目的地まで送る SAV」が存在するが，前者を FirstSAV，後者を SecondSAV と表す．

3.2　デマンドの分割

乗り継ぎを行うために，乗車地点と降車地点の経由地のペアを持つデマンドを，乗車地点と乗り換え地点の経由地を持つデマンドと，乗り換え地点と降車地点の経由地を持つデマンドの 2 つに分割する．以降，分割した 2 つのデマンドをそれぞれ FirstDemand，SecondDemand と呼ぶことする．この FirstDemand と SecondDemand の 2 つを，それぞれ FirstSAV と SecondSAV に配車することで，車輌の乗り継ぎを実現する．

3.3　待ち合わせ問題

乗り継ぎを行うためには，FirstSAV と SecondSAV が乗り換え地点で待ち合わせをして，お客さんがそこで乗降車する必要がある．FirstSAV と SecondSAV のどちらも先に乗り換え地点に到着する可能性があり，先に到着した方は後に到着する方を待ち，到着次第乗り換えを行うものとする．この待ち時間を小さくすることは，乗り継ぎの配車によって効率よく配車する上で非常に重要である．

この待ち合わせにおいて，デッドロックに注意する必要がある．図 3 に待ち合わせのデッドロックの最も単純な例を示す．図 3 において，黄色の T と書いた経由地が待ち合わせを必要とする経由地を示している．また，矢印は待ち合わせの対象に対応する経由地 (デマンド) 同士を結んでいる．この例では，SAV1 が SAV2 の 2 つ目の待ち合わせの経由地の待ち合わせのために待機状態に入るが，一方 SAV2 も SAV1 の 2 つ目の待ち合わせの経由地の待ち合わせのために待機状態に入ってしまう．このような場合，デッドロックが生じて 2 台の SAVS は共に待機状態に入ってしまう．3 台以上の SAVS での場合も，この例よりもさらに複雑に待機状態がからみ，デッドロックが生じる．このようなデッドロックは，図 3 のように矢印が交差する場合，すなわち待ち合わせの経由地の順番が配車された順と変わってしまうことにより生じる．これに対処するために，本研究では，乗り継ぎを行うデマンドの処理の順番を配車した順とした．すなわち，乗り継ぎデマンドを割り当てする場合には，その乗降車地点の経由地点の挿入位置は，最後の乗り継ぎデマンドの挿入位置以降に制限した．

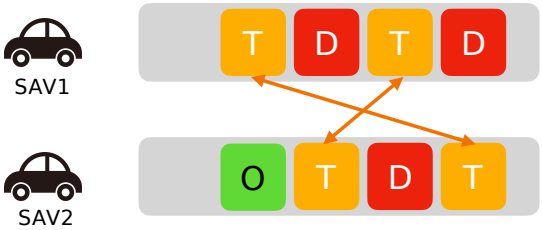

図 3　待ち合わせのデッドロックの例

3.4　探索する SAVS の候補の絞り込み

乗り継ぎを行う上で，FirstSAV と SecondSAV のペアを探索する必要があるが，全てのペアを探索しようとすると SAV の台数が大きい場合に計算量が大きくなってしまう．これを防ぐため，事前に探索する SAVS のペアの候補を絞り込む．具体的には，各 SAV の未来の経路 (乗降車地点の配列) において，デマンドの乗車地点と降車地点にそれぞれ近い経由地を持つ SAVS を近いものから順に m 台ずつ見つけ，FirstSAV と SecondSAV の候補とする．ただし，各 SAV の未来の経路は 3.3 の待ち合わせ問題のデッドロックの解消法の都合上，経路の最後尾の乗り継ぎを行う経由地以降の経路に限定する．この SAVS のペアの絞り込み方法は，図 2 の乗り継ぎが効率よく行える例に基づいている．実際の探索の際には，2 つの集合に共通の SAV がなければ，$m \times m$ のペアの探索を行うこととなる．

3.5　配車アルゴリズム

乗り継ぎの配車アルゴリズムの概要は以下の通りである．これは逐次最適挿入法を拡張した形となっている．通常の逐次最適挿入法と同様に，以下の操作はデマンドが発生すると随時行われる．

1. 通常の逐次最適挿入法を実行し，式 (1) のコストが最小となる配車車輌と経由地の挿入位置と，そのコストを求める．ただし，その挿入位置は最後の乗り換えの経由地の挿入位置以降とする．これは，待ち合わせをする前に経由地が増えてしまうと，乗り継ぎを行う相方の車輌の待ち合わせの待ち時間が増加し，効率が低下するおそれがあるためである．また，このデマンドがキャンセルになった場合も 2. 以降の処理に進む．
2. 探索する SAVS のペアの絞り込みを行い，要素数が m の FirstSAV の候補集合と，SecondSAV の候補集合を求める．
3. 新しいデマンドを，FirstDemand と SecondDemand に分割する．3. で絞り込んだ $m \times m$ 個のペアにおいて，FirstDemand と SecondDemand を FirstSAV と SecondSAV の経由地配列の最後の乗り換え経由地以降の任意の位置に挿入する．このとき，以下のコス

トを計算する．

$$arrivalTime + \sum_{FirstSAV,SecondSAV} delay(SAV) \quad (2)$$

このコストの計算は式 (1) の逐次最適法における配車コストの素直な拡張となっている．$arrivalTime$ には，乗り継ぎの待ち合わせによって生じる待ち時間も含まれており，このコストを小さくすることは，待ち合わせの待ち時間を小さくすることにも繋がる．また，$\sum_{FirstSAV,SecondSAV} delay(SAV)$ については，FirstSAV の FirstDemand の経由地の挿入によって生じる既に保持しているデマンドの遅延時間と，SecondSAV の SecondSAV の経由地の挿入によって生じる既に保持しているデマンドの遅延時間を足し合わせたものとしている．以上のような配車コストとすることで，逐次最適挿入法の 1 台による配車コストと大小を比較することができる．

このコストが最小となる FirstDemand と SecondDemand の 4 つ挿入位置を，新たなデマンドの挿入位置の候補とする．ただし，逐次最適挿入法と同様に．これらのデマンドの挿入によって既存あるいは挿入したデマンドのいずれかの締切時間を過ぎてしまう場合は，候補から除外する．また，この乗り継ぎによる配車がキャンセルとなった場合も．4 の処理に進む．

4. 1. の一台による配車がキャンセルとなっており，乗り継ぎによる配車もキャンセルとなっていた場合，このデマンドはキャンセルとする．また，一台による配車と乗り継ぎによる配車のいずれか一方がキャンセルとなっていた場合は，キャンセルとならない配車方法で配車する．いずれの配車方法についてもキャンセルとならなかった場合は，1. で求めた一台による配車のコストと，3. の乗り継ぎによる配車のコストを比較し，よりコストが小さかった方法で配車を行う．

4 実験

交通シミュレータである Simulation of Urban MObility (SUMO) [3] を用いて，図 2 のような乗り継ぎが効率的であるとされる例を意図的に誘発するような条件でシミュレーションし，その例が実際に効率が良くなるかどうかを検証する．また，車輌の乗り継ぎを効率的に導入するために必要な SAV の車輌台数等も調べる．

4.1 実験設定

4.1.1 マップとデマンド発生

図 4 にシミュレーションを行うマップを示す．このマップは，細長いグリッドマップになっており，各グリッドは長さ 200 メートルである．このうち，乗り換え地点はこのマップを左右に区切った時の，真ん中の点 (上下の 2 点のうち上の点) とする．また，その真ん中を境として左右のエリアに分かれている．デマンドは左右のそれぞれのエリア内を移動するデマンドと，左右のエリアを横断するデマンドが 2:1 の割合で発生する．このような特殊なマップは，図 2 に示されるような乗り継ぎが効果的な例を意図的に起こすことを念頭に作成した．左右のエリアに閉じたデマンドを処理していく中で，時折発生するエリアを横断するデマンドは乗り継ぎを行った方が効率的であると考えられる．また，このように細長いマップとしているのは，縦に長いグリッドマップの場合，乗り換えをせずにそのままエリアを横断した方が効率が良い場合が存在し，それを起こさないようにするためである．

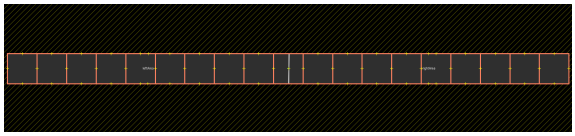

図 4　シミュレーションのマップ

4.1.2 シミュレーション設定

表 1 に，シミュレーションのパラメータの設定を示す．

パラメータ	値
シミュレーション時間	14400[sec]
SAVS の車輌数	2 から 14 (2 刻み)
一台の乗車人数	4
SAVS の絞り込み数 m	4
シミュレーション回数	一設定あたり 50 回

表 1　シミュレーション設定

5 結果

上記の設定でシミュレーション実験を行った結果を示す．

5.1 キャンセル率の分析

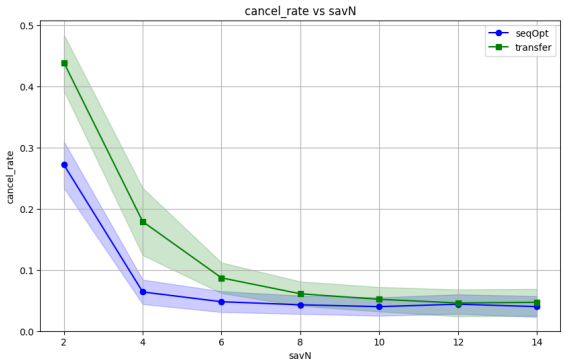

図 5　各配車方式の SAV の台数ごとのキャンセル率

図 5 は，逐次最適挿入法と提案手法 (乗り継ぎによる配車) における，SAVS の台数 (savN) ごとのキャンセル率をプロットしたものである．ただし，緑色のグラフが乗り継ぎ配車のもので，青色のグラフが逐次最適挿入法によるものである．また，グラフの色の薄い部分は標準偏差を表す．この図から，SAV の台数が少ない時には乗り継ぎ配車の効率はあまり良くないものの，SAV の台数が大きくなると逐次最適挿入法と同等程度のキャンセル率となっており，乗り継ぎを導入できる可能性があることを示している．ただし，乗り継ぎ配車の方が標準偏差が大きく，逐次最適挿入法よりもキャンセル率が良くなったり悪くなったりとブレの大きいことがわかる．

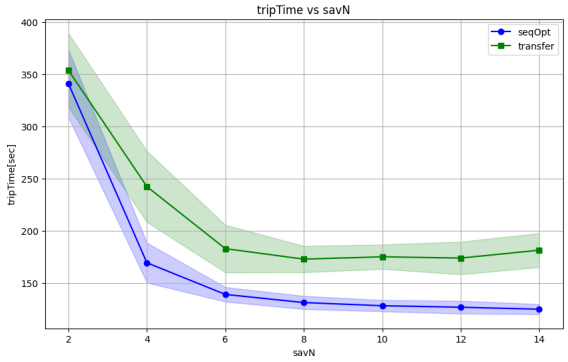

図 6　各配車方式の SAV の台数ごとの旅行時間

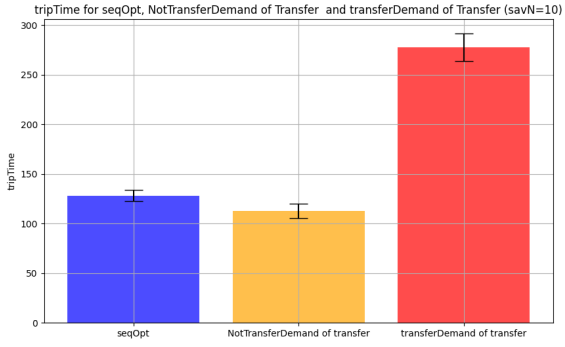

図 7　逐次最適挿入法における旅行時間と，乗り継ぎアルゴリズムにおける乗り継ぎデマンドと，乗り継ぎなし (一台) デマンドの旅行時間の比較 (savN=10)

5.2　旅行時間・乗車までの待機時間の分析

図 6 は，逐次最適挿入法と乗り継ぎによる配車における，SAVS の台数 (savN) ごとの旅行時間 (乗車してから降車するまでの時間) の平均値をプロットしたものである．グラフの形式等は図 5 と同様である．この図から，乗り継ぎによる配車の方が，基本的に逐次最適挿入法よりも旅行時間が長いことがわかる．これは，乗り継ぎの車輌の待ち合わせによって生じる待ち時間が加わることで旅行時間が長くなってしまうためであると考えられる．

図 7 は，逐次最適挿入法における旅行時間と，乗り継ぎアルゴリズムにおける乗り継ぎを行うデマンドと，乗り継ぎをせずに一台で目的地まで行くデマンドの旅行時間を比較したものである．ただし，SAVS の台数は 10 台とし，左から順に逐次最適挿入法における旅行時間，乗り継ぎアルゴリズムにおける乗り継ぎを行うデマンドの旅行時間，乗り継ぎをせずに一台で目的地まで行くデマンドの旅行時間となっている．このグラフから，やはり乗り継ぎの旅行時間は，待ち合わせの待ち時間も含まれるため，かなり大きくなってしまうことがわかる．一方で，逐次最適挿入法における 1 台の配車と，乗り継ぎアルゴリズムにおける 1 台の配車の旅行時間を比較すると，乗り継ぎアルゴリズムにおける 1 台の配車の方が旅行時間が若干小さくなっている．この理由について確かめるために，次に図 4 の左右それぞれの二つのエリア内に閉じたデマンドと，二つのエリアを横断するデマンドとで，結果がどのように異なるかを分析した．

図 8 と図 9 がそれぞれ，エリア横断デマンドとエリア内デマンドにおける各配車の旅行時間を表す．このグラフから，まず逐次最適挿入法ではエリア横断デマンドの旅行時間が 50 秒ほどエリア内デマンドに比べて大きいことがわかる．一方，乗り継ぎアルゴリズムの場合は，乗り継ぎによる配車の場合も 1 台による配車の場合も旅行時間はほとんど差がない．以上より，乗り継ぎアルゴリズムは図 2 の例のようなエリア横断デマンドを効率よく処理できる配車アルゴリズムであることがわかる．なお，乗り継ぎアルゴリズムの一台の配車の旅行時間が，デマンドの種類ごとにほとんど変化しない理由は，効率の悪くなってしまう一台の配車を乗り継ぎの配車が代わりに担うためであると考えられる．

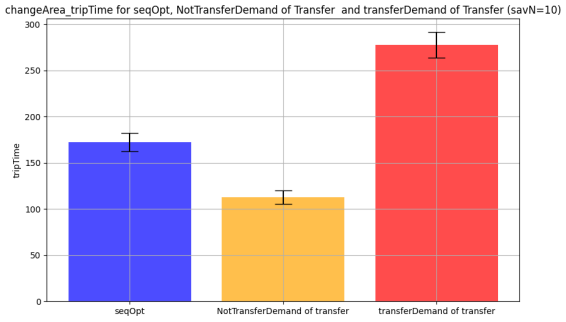

図 8　エリア横断デマンドの，逐次最適挿入法における旅行時間と，乗り継ぎアルゴリズムにおける乗り継ぎ配車と，乗り継ぎなし (一台) 配車の旅行時間の比較 (savN=10)

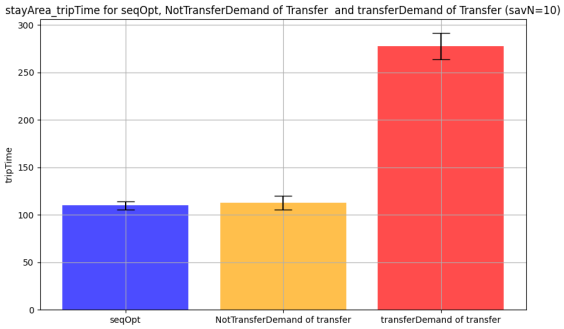

図 9　エリア内デマンドの，逐次最適挿入法における旅行時間と，乗り継ぎアルゴリズムにおける乗り継ぎ配車と，乗り継ぎなし (一台) 配車の旅行時間の比較 (savN=10)

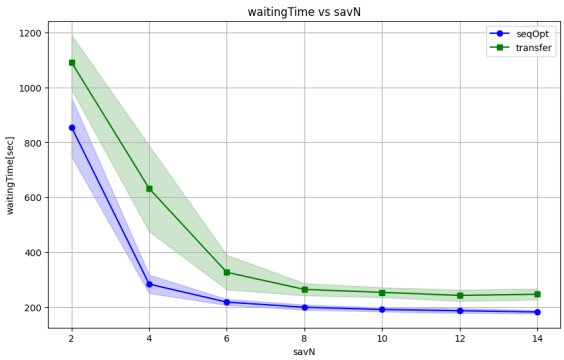

図 10　各配車方式の SAV の台数ごとの SAV に乗車するまでの待機時間

図 10 は，逐次最適挿入法と乗り継ぎによる配車における，SAVS の台数 (savN) ごとの配車を要求してから

SAV が到着して乗車するまでの待機時間の平均値をプロットしたものである．グラフの形式等は図 5 と同様である．この図から，2 つの配車方式とも SAVS の台数が大きくなると標準偏差が小さくなり，概ね 3 分程度の待ち時間に収束していることがわかる．また，図 6 と同様に乗り継ぎによる配車の方が待機時間は長く，これについても車輌の待ち合わせによって生じる待ち時間によって，車輌の到着時間が遅れやすいためであると考えられる．

5.3　乗り継ぎに関する分析

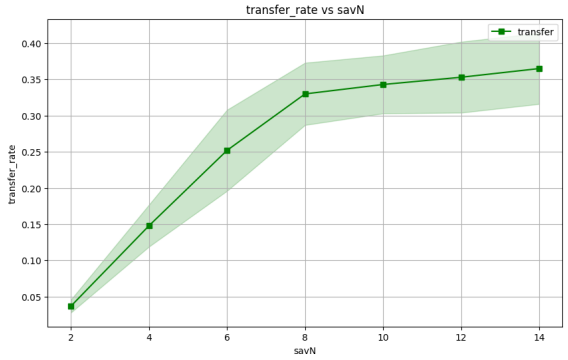

図 11　乗り継ぎの配車方式における，SAV の台数ごとの乗り継ぎ配車をする割合

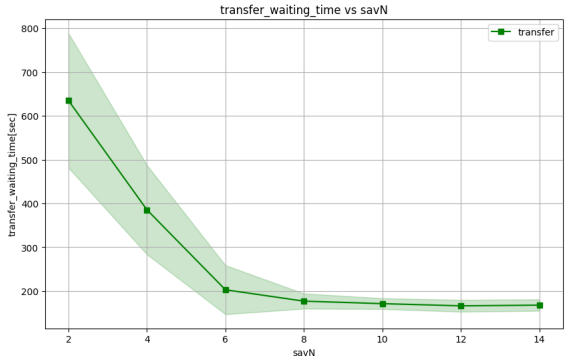

図 12　乗り継ぎの配車方式における，SAV の台数ごとの待ち合わせによって生じる待ち時間

図 11 および図 12 は，それぞれ乗り継ぎによる配車における，SAVS の台数 (savN) ごとの乗り継ぎの配車を行う割合と，乗り継ぎの待ち合わせによって生じる待ち時間の平均値をプロットしたものである．ただし，グラフの色の薄い部分は標準偏差を表す．これらの図から，SAV の台数を増加させると乗り継ぎ配車の割合が 35% 程度まで増加し，さらに待ち合わせによって生じる待ち時間も 3 分程度まで小さくなることがわかる．これは，SAVS の台数が増加すると，逐次最適挿入法よりも効率の良い乗り継ぎの配車を見つけられる可能性が大きくなるためであると考えられる．

図 13 では，図 4 の左右それぞれの二つのエリア内に閉じたデマンドと，二つのエリアを横断するデマンドとで，乗り継ぎ配車が行われる割合がどのように異なるかを示したものである．ただし，SAV の台数は 10 台に固定し，左の黄色い棒グラフがエリア内に閉じたデマンドにおける乗り継ぎ割合，右のオレンジのグラフがエリア

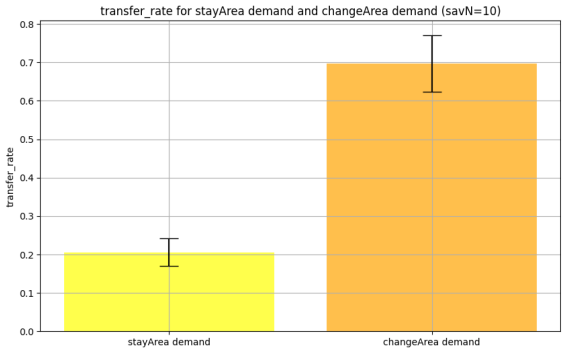

図 13　乗り継ぎ配車方式におけるエリア内デマンドとエリア横断デマンドの乗り継ぎ配車の割合

を横断するデマンドにおける乗り継ぎ割合を示している．この図から，エリアを横断するデマンドの方が乗り継ぎ割合が大きくなり，これはエリアを横断するデマンドは図 2 の例のような，乗り継ぎをした方が効率がよくなりやすい例の条件を満たしているためであると考えられる．

6　考察

実験の結果を踏まえ，考察を行う．本稿の実験では，意図的に図 2 の乗り継いだ方が効率が良くなると考えられる条件を満たしやすいような設定でシミュレーションを行った．その結果，図 13 のグラフからわかるように，エリア内デマンドよりもエリア横断デマンドの方が乗り継ぎを行う割合が 3 倍程度高く，また図 8 や図 9 からわかるように乗り継ぎアルゴリズムの方ではエリア内デマンドとエリア横断デマンドの旅行時間にほとんど差がないことから，確かに意図したように図 2 の条件を満たせば乗り継いだ方が効率が良くなることがわかる．一方で，図 5 のグラフからわかるように，乗り継ぎの配車におけるデマンド全体の効率は大きく改善しなかった．これは以下の主に 2 つ原因が考えられる．

1. 一つ目の原因として，提案した乗り継ぎ配車アルゴリズムがまだ真に効率の良い乗り継ぎ配車を求めれていないことが考えられる．例えば，提案アルゴリズムでは，通常の逐次最適挿入法がキャンセルとなり，かつ乗り継ぎによる配車がキャンセルとならなかった場合には自動的に乗り継ぎによる配車を行うようにしている．しかし，これには待ち時間が極端に長かったり，効率の悪い配車が含まれている可能性がある．よって，逐次最適挿入法がキャンセルとなった場合にも，真に効率の良い乗り継ぎの配車に絞る必要がある．また，提案アルゴリズムでは，通常の逐次最適挿入法の処理を行う部分で，探索する経由地の挿入位置は，最後の乗り換えを行う経由地以降の挿入位置に限定されている．しかし，本来は待ち合わせで待機している予定の時間にデマンドを処理できれば効率が良くなるはずであるが，現状のコスト関数ではそのような場合が考慮されていない．よってデマンドの経由地の挿入によって待ち合わせの待機時間が減少する場合は，そのようにコストが減少するような記述を処理に追加すれば，更なる効率の向上が見込めると考えられる．
2. 二つ目の原因として，本稿では乗り継いだ方が効率が良くなる例を図 2 のパターンしか考慮していない

が，本来は乗り継いだ方が効率が良くなるパターンがさらに存在する可能性がある．それは，上のパターンは 1 つのデマンドについて効率が良くなるが，全体として効率が良くなるようなパターンもあると考えられるためである．よって，今後さらに様々な条件で実験を行い，全体のデマンドで乗り継ぎの効率が良くなるパターンがあるかどうかを見つける必要がある．

6.1 実験結果を実際に応用する上での考察

実験結果を踏まえ，実際に乗り継ぎのアルゴリズムを応用する場合の考察を行う．図 5,6,11,12 からわかるように，乗り継ぎによる配車は多くの車輌台数が存在するときに，効率の良い配車ができる．これは，車輌の台数が多ければ，旅行時間や待ち合わせの待ち時間が短い効率の良い配車が見つかりやすいためであると考えられる．よって，乗り継ぎのアルゴリズムを実際に応用する場合は，小さい面積のエリアに多くの車輌の台数が必要となる都市部が適していると考えられる．一方で，車輌の台数を多くすると採算の取れなくなってしまうようなエリアの場合は，乗り継ぎの配車の割合自体も小さいものの，そのわずかな乗り継ぎ配車の待ち合わせの待ち時間も相当大きくなり，1 台の配車を行う逐次最適挿入法よりもかなり効率が落ちてしまうと考えられる．また，都市部のエリアのうちでも，今回の実験で行ったパターンのような二つ (あるいは複数) の主なデマンド発生分布が存在すると，図 2 の乗り継いだ方が効率が良くなるパターンが起きやすくなると考えられる．

7 おわりに

本研究では，AI 便乗サービスにおける計算量を抑えた乗り継ぎを導入した配車アルゴリズムを提案した．このアルゴリズムは，逐次最適挿入法を拡張した形になっており，コスト関数が逐次最適挿入法と同様のもので，コストが比較可能であった．また，乗り継ぎで生じる待ち合わせ問題におけるデッドロックの解消法の他，探索空間を減らすための車輌ペアの絞り込み方法を提案した．

車輌の乗り継ぎは図 2 の場合に効率が良くなると考えられるため，これを意図的に誘発するような設定でシミュレーションを行った．その結果，2 つのデマンド発生エリアとそれを横断するデマンドが発生する場合に，エリア横断デマンドでは乗り継ぎ配車の割合がかなり大きく，乗り継ぎアルゴリズムの方はエリア横断デマンドに対して旅行時間のロバスト性を示すため，確かに図 2 のような場合に乗り継いだ方が効率が良くなることが示された．一方で，全体のデマンドのキャンセル率で評価した場合には，十分に車輌の台数が大きい場合に逐次最適挿入法と同等のものになることがわかった．今後，全体のデマンドの評価で逐次最適挿入法を上回るような効率で乗り継ぎを行うためには，効率の良くなる乗り継ぎ配車をより的確に見つけ出すアルゴリズムが必要であると考えられる．

本研究の結果を踏まえ，実際に乗り継ぎの配車を応用する上では，車輌の台数とデマンドの発生分布が重要である．乗り継ぎによる配車は，実験結果からわかるように，十分な車輌の台数がエリア内に存在するときに良い効率を示す．よって，エリア内に十分な車輌台数が見込める都市部において乗り継ぎの導入が適していると考えられる．また，複数のデマンドの発生分布と，それらを行き来するデマンドが発生する場合は，図 2 のような例が起きやすく，乗り継ぎによる配車が適している．これについても，都市部であればエリア内に複数のデマンド発生分布 (駅やショッピングモールなど) を持つ例は多いと考えられ，やはり乗り継ぎによる配車は都市部が適していると考えられる．

参考文献

[1] 中島秀之, 小柴, 佐野渉二, 落合純一, 白石陽, 平田圭二, 野田五十樹, 松原仁ほか. Smart access vehicle system: フルデマンド型公共交通配車システムの実装と評価. 情報処理学会論文誌, Vol. 57, No. 4, pp. 1290–1302, 2016.

[2] 野田五十樹, 篠田孝祐, 太田正幸, 中島秀之ほか. シミュレーションによるデマンドバス利便性の評価. 情報処理学会論文誌, Vol. 49, No. 1, pp. 242–252, 2008.

[3] Pablo Alvarez Lopez, Michael Behrisch, Laura BiekerWalz, Jakob Erdmann, Yun-Pang Fl ¨ otter ¨ od, Robert Hilbrich, Leonhard L ¨ ucken, Johannes Rummel, Peter Wagner, and Evamarie Wießner. Microscopic traffic simulation using sumo. In The 21st IEEE International Conference on Intelligent Transportation Systems. IEEE, 2018.