令和元年度　卒業論文

待ち行列シミュレーターによるテーマパークの混雑問題の分析とその対策

千葉大学工学部　都市環境システム学科

学籍番号：16T0318W

氏　　名：三井 知樹

指導教員：塩田 茂雄　教授

令和２年２月５日提出

目次

第１章

はじめに

１.１　研究背景

現在多くのテーマパークが世の中にはあふれている．しかし，どのようなテーマパークであっても必ず客が来てそれらの客がそれぞれの乗りたい，体験したいアトラクションに対して長い列を作る．その長蛇の列を見た時にそれらの列には並びたくないと考える．これは，アトラクションに乗りたい反面列に並びたくないと考える負の側面が勝ってしまう．それでは，せっかくのアトラクションを十全に楽しむことができない．それらの軽減方法としてアトラクションの数を増やすという方法がある．アトラクション増やせば確かに各アトラクションの待ち時間は軽減できるであろうが，経営する側に対して多大な負担を強いることとなり金銭面でも敷地面でも現実的ではない．では何らかの戦略を用いて来園者の行動に干渉することはできないだろうか．そのような考えから来場者の待ち時間の軽減や混雑度の改善，満足度の向上を目的として様々な研究が行われている[1]．そしてそれらの戦略が実際に使用されている．

本研究ではテーマパーク混雑問題を実規模で分析するためのシミュレーターを構築し，テーマパーク混雑問題に対する現実的な対策を検討する．とっくに各アトラクションで客が待ち行列と，客がアトラクションを相互に行きかう状況を作り出すため各アトラクションを構築する単一の待ち行列を相互に連結した「待ち行列ネットワーク」によりテーマパークを模擬するシミュレーターを構築する．さらに，構築したシミュレーターにより、テーマパークの混雑の特徴を考察するとともに，混雑を緩和するためのいくつかの方法を検討する．

１.２　本論の構成

本論文の構成は以下のようになっている．第２章では本論文で使用するテーマパーク混雑度を模擬する待ち行列ネットワークシミュレーターについて記述する．第３章ではテーマパーク混雑問題に対して本論で扱った戦略の説明を，第４章ではそれらの戦略に対する評価をする．第５章では本論文の実験の結果を記した．また，付録として今回用いたプログラムのソースコードを記述した．

第２章

テーマパーク混雑度を模擬する待ち行列ネットワークシミュレーターの構成と設定

２.１　シミュレーターの基本構成

本研究でテーマパークの混雑を模擬する待ち行列ネットワークシミュレーターは，テーマパーク内の各アトラクション（入り口も一つのアトラクションとして扱う）を単一窓口待ち行列としてモデル化する．開始時刻から客が到着して，入場ゲートに並ぶその後テーマパークの入場時間になり次第先頭に並んでいる客から順に、園内に入場する．入場した客は時間をかけて移動してアトラクションに到着した訪問客は待ち行列を作り並び，順番に窓口でサービスを受ける．窓口におけるサービス時間は，例えば観覧車の場合ゴンドラが到着する時間間隔をゴンドラに乗車する客数で割ったものに相当し，窓口でサービスを受ける行為はゴンドラに乗り込む行為に相当する．窓口でサービスを受け終わった客は，道を移動し時間をかけて次のアトラクションに移動する．すべてのアトラクションを回り終えた客は退場し，待ち時間やサービス数，滞在時間などの各パラメータを格納する．これらのパラメータは規定時間ごとの計算や，シミュレーション終了時に総待ち時間，総滞在時間などの計算に用いられる．

２.２　客（エージェント）の行動

客に相当するエージェントは以下のとおりの行動規範に従う．

（１）テーマパークへ到着し入園する．（開園時間前に到着した客は入場口で開園時間まで待機する；営業開始の２時間前から客がゲートに並び始めるものとする.）

（2）その日に回るアトラクションと，どの順番でアトラクションを回るかを決める．この時，必ず始めに入場口を通り，入場口に行列がある場合，その最後尾に並ぶものとする．

（3）入場口を通過した客は（2）で決めた順番に従い各アトラクションに移動する．

（4）到着したアトラクションに行列があればその最後尾に並び，なければサービスを受ける．

（5）（2）で決めた回る予定のアトラクションを回り終えているのならば，退園する．まだ訪問予定のアトラクションが残っているのならば（3）へ戻り，次の予定のアトラクションを訪問する．

2.3 シミュレーションの詳細設定

2.3.1 シミュレーションの時間設定

実規模での想定をしているため，テーマパークの開園は8時，終了は22時として前述したとおり6時（開園の2時間前）から訪問客がゲートに並ぶものとする．

また，シミュレーションにおいての時間は1秒単位で考えるものとし，開始時刻（6時）を0として終了時刻（22時）57600までをシミュレーション時間とする．

2.3.2 テーマパーク園内の設定

2.3.2.1 アトラクションの設定

アトラクション数は入場口を含め20とする．前述したように，テーマパーク内の各アトラクションは単一窓口待ち行列としてモデル化する．実際のアトラクションでは，ゴンドラや列車などの乗り物に複数の人が一斉に乗り込むため，単一窓口待ち行列でのサービスとは異なるが，各アトラクションの混雑度はおおよそ模擬できると考える．なお，単一窓口待ち行列としてモデル化した場合，窓口でのサービス時間は，ゴンドラなどの乗り物の到着間隔を乗り物に

乗り込む人数で割った値になる．シミュレーションにおいては，入場口以外のサービス時間は一律平均 2 秒としたが，これは 30 秒おきに到着する乗り物に 15 人が乗り込む場合に相当する．なお，入場口でのサービス時間は平均 0.1 秒に設定した．これは入場口にチケットをチェックする場所が 10 カ所あり，1 人あたりのチェック時間が 1 秒であることに相当する．加えて，アトラクション間の平均移動時間は 10 分とし，移動時間は指数分布に従うものとする．また，アトラクションでの乗り物の乗車時間は 10 分とした．

２.３.２.２ エージェントの設定

外部から園への客の到着は非斉時ポアソン過程に従うものとし，図 2.1 のように到着率を 1 時間ごとに定める．特に，開園前に 1 日の客の約 15%が既に到着し，さらに開園後の 1 時間に 1 日の客の約 30%が到着するように到着率を定める．その後，到着率はゆるやかに減少するが，午前中に 1 日の客の約 8 割が園に到着する．その後，しばらく新規客がほとんど到着しない時間帯が続き，さらにテーマパーク内部のナイトパレードを見に来る客がいることを想定し，１６時から１８時までの 2 時間，入園者がやや増えるように到着率を設定する．この時間帯に到着する客は（パレードだけを見るのではなく）各アトラクションを訪問させる．それ以降はエージェントの到着はほとんどなくなり，閉園時間までの間２.２節の行動規範に従いアトラクションを訪問後退園する．もしアトラクション訪問時間が閉園の 5 分前であれば，エージェントはアトラクションを訪問せずに退園する．

1 日に園を訪れる客の総数は 4 万人とする．これは，2018 年の東京ディズニーリゾートの訪問者数が 32,558,000 人であり，1 日あたり 89200 人であること，東京ディズニーリゾートのメインの施設はディズニーランドとディズニーシーの二つのテーマパークであるがこのほかの施設（イクスピアリなど）もあること，従って一つのテーマパークに訪れる客はおよそ全体の 45%であると考えられることなどから概算したものである．

各アトラクションには園内での人気度として訪問確率を設定する．エージェ

ントはこの訪問確率に従って訪れるアトラクションを選び，アトラクションを選んだ後に，訪問順序を決める．ただし，入場口は必ず通過するものとする（図 2.2）．訪問確率は来園者数に対しおよそどのくらいの割合のエージェントが訪問するかを表したものであり，例えば訪問確率 0.4 のアトラクションはシミュレーション内で約 40000 人のエージェントが来園した場合そのうちのおよそ 16000 人が訪問するようなアトラクションであることを示す．

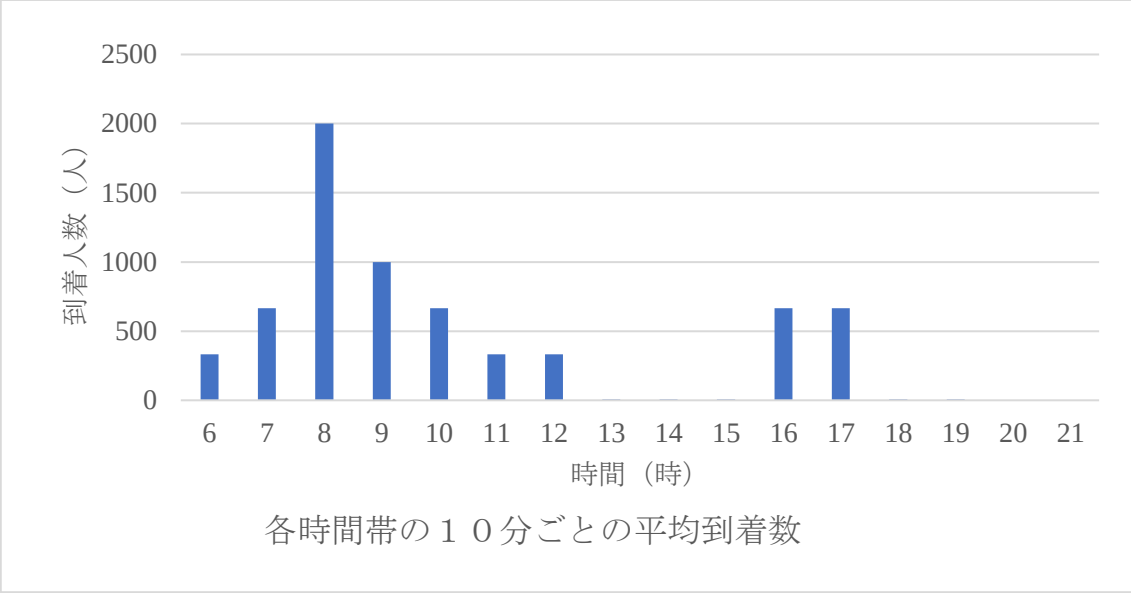


図 2.1 各時間の１０分ごとの平均到着数

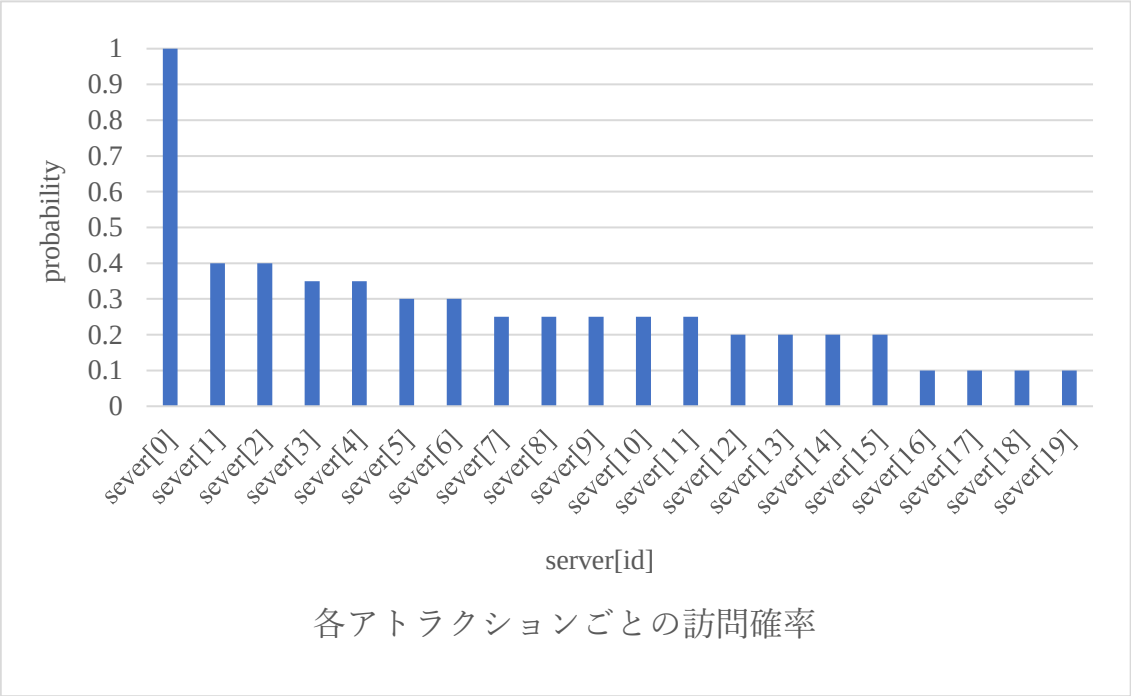


図 2.2 各アトラクションの訪問確率

シミュレーションにおいては，各エージェントはアトラクションを通過したときに待ち時間を格納するための配列を持ち，それらを退園時にすべて足し合わせて，そのエージェントの園内での総待ち時間を算出する．また，テーマパークに到着した時間を格納し，退出時にその時間と退出した時間の差をそのエージェントの総滞在時間とする．

客にはそれぞれに個性があり，例えば，性別，年齢，カップルか家族連れかなどにより，その振る舞いは異なると考えられる．このため，各アトラクションでのサービス時間（各乗り物へ乗り込む客数で変動する）や次のアトラクションへの移動時間には若干ばらつきがあることを想定し，シミュレーションにおいてはそれぞれの時間を指数分布に従って変動させている．

２.４　待ち時間，待ち行列長などの測定手順

エージェントがアトラクションを訪問するたびに，そのアトラクションの訪

問客数を更新する．同様に，エージェントがアトラクションでサービスを受け終わるたびに，そのアトラクションでの待ち時間を計算し，アトラクションでの客の待ち時間の総和とアトラクションでのサービス数（退去客数）を更新する．また，客が園を退出する度に，その客のテーマパーク内での総滞在時間，総待ち時間を計算し，それまで園を退出した客の総滞在時間，総待ち時間の和を更新する．また，1 秒おきに，各アトラクションの待ち行列長を調べる．

各アトラクションでの平均待ち時間，訪問客数，退去客数，園を退出した客の平均滞在時間，平均総待ち時間は 30 分毎に（過去 30 分の）値を出力するとともに，シミュレーションの終了時に，シミュレーション開始から終了までの客の平均総滞在時間，平均総待ち時間を算出し，出力する．

第３章

テーマパーク混雑緩和のための戦略

今回，比較し評価をする戦略は何もしない通常の状態（以下戦略 A）と，一般によく知られている，単位時間ごとに入場する客数の制限（以下戦略 B）と単位時間ごとに売るチケットの枚数を制限する戦略（以下戦略 C）との三つの条件を比較する．

戦略 B は通常の与えられた到着確率に従ってゲートに客が到着するが，1800 秒（30 分）ごとに決められた人数（3000 人）までしかゲートを通過させず，それを超過した場合，次の時間までゲートで待機させる戦略（入場口での入場制限）である．シミュレーションにおいては，全ての客に通し番号を振るともに，各客が入場口を通過した時間を記録し，n 番の客が入場口を通過できる時刻は n-3000 番の客が入場口を通過した時刻より 30 分後以降になるように入場制限を行った．なお，3000 番より若番の客については入場制限を行わない．

戦略 C は各チケットに入場できる時間帯を定めておく方法である．チケットは全て事前販売されていることを想定する（当日販売のチケットは無い）．客はチケットに指定された時刻に来園する．なお，シミュレーションではチケットに定められた時間は 1 時間単位であるとし，各時間 32000/6=5333 人分のチケットを販売することとする．これにより，8 時の開園から 2 時までの 6 時間に 32000 人の客がほぼ同じ割合で来園することとなり，開園時のピークが平滑化され，待ち時間が大幅に減少することが期待できる（残りの客は 16 時から 18 時までに来園する）．

第４章

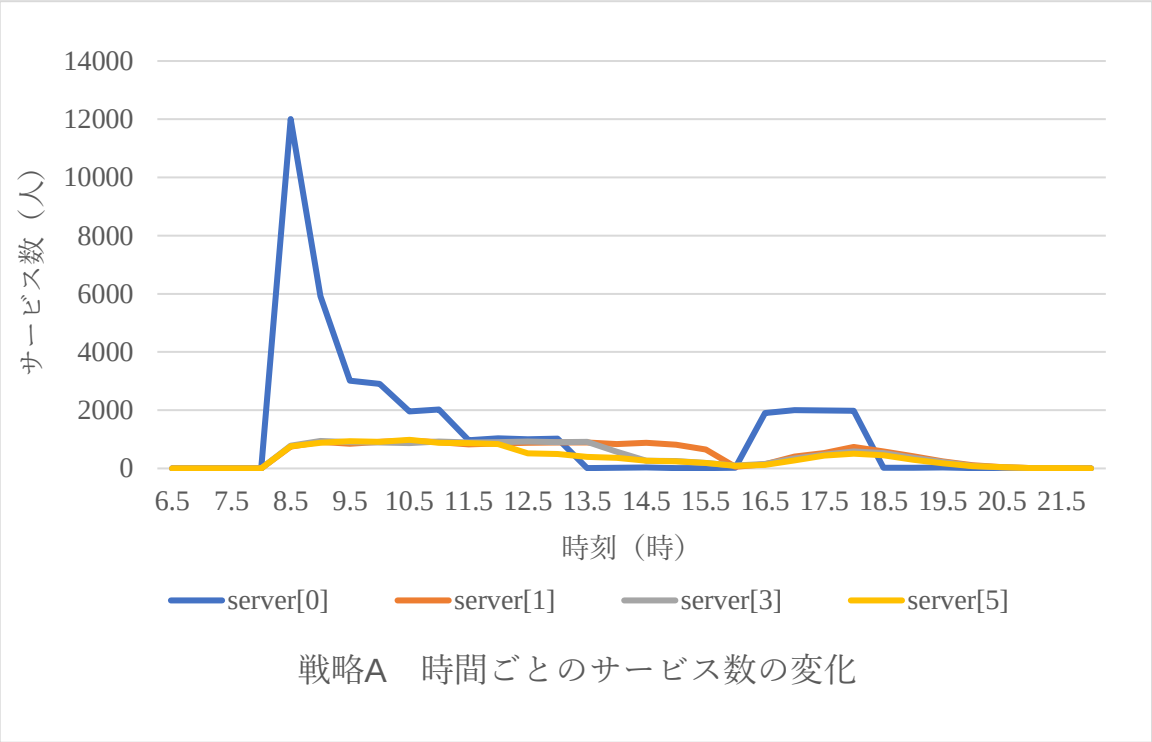
シミュレーション

４.１　条件設定

シミュレーションの初期条件の設定をする，まずは２章で示した通り客の到着率，アトラクションの人気度を設定する。その後，一日でのおよその客の到着数を決める．今回のシミュレーションでは約 40000 人とした．

４.２　結果

完成したシミュレーターを用いて，各戦略においての園内での総滞在時間，



総平均待ち時間，そしてアトラクションごとのサービス数と平均待ち時間，平均待ち行列長の比較を行いその結果をグラフとしてまとめ比較をした。まずは，戦略 A を示す．時間ごとのサービス数の変化（図 4.1）と平均待ち行列長の変化（図 4.2）と，平均待ち時間の変化（図 4.3）のグラフである．

図 4.1　戦略 A 時間ごとのサービス数の変化

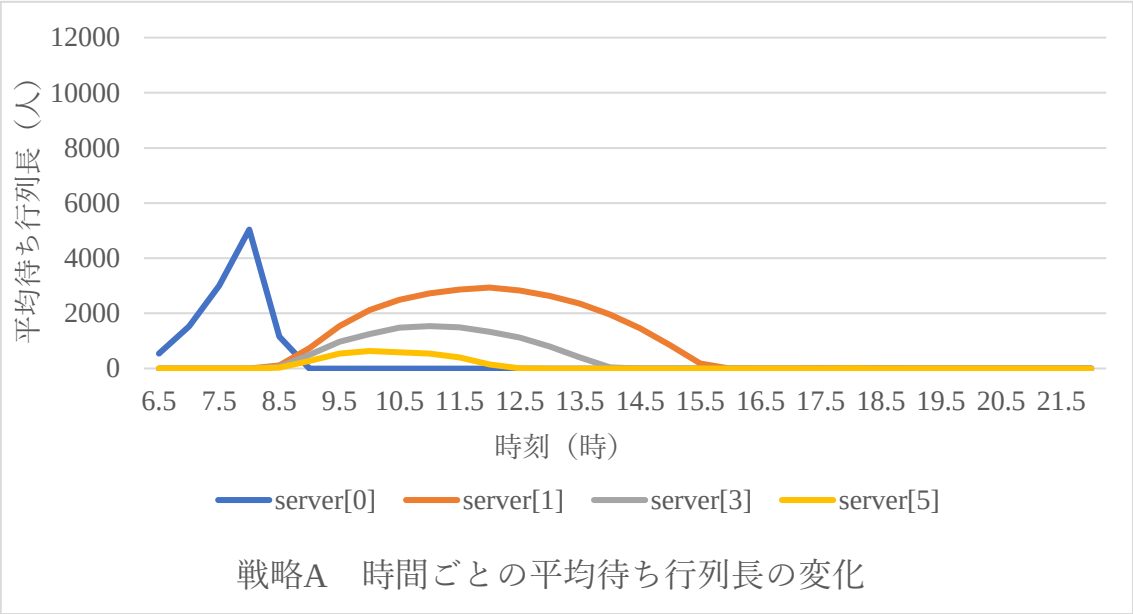


図 4.2　戦略 A 時間ごとの平均待ち行列長の変化

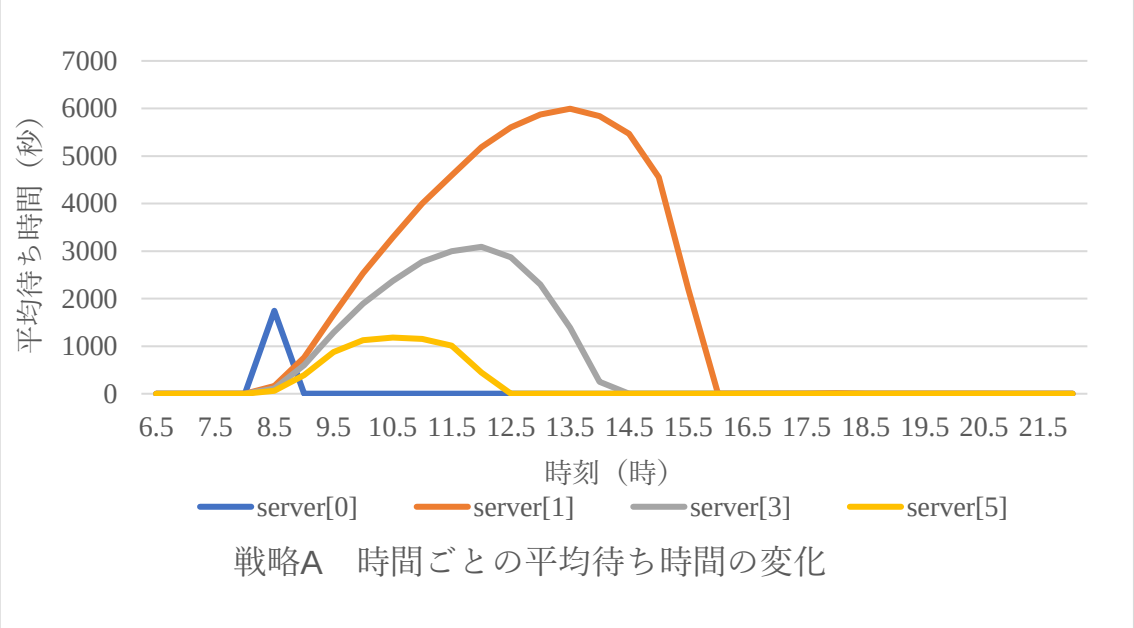


図 4.3　戦略 A 時間ごとの平均待ち時間の変化

シミュレーションの結果上記のような実験結果が得られた。これをもとに戦略 B，戦略 C，についても同様のグラフにして，結果を見ていく．戦略 B について次に示す．

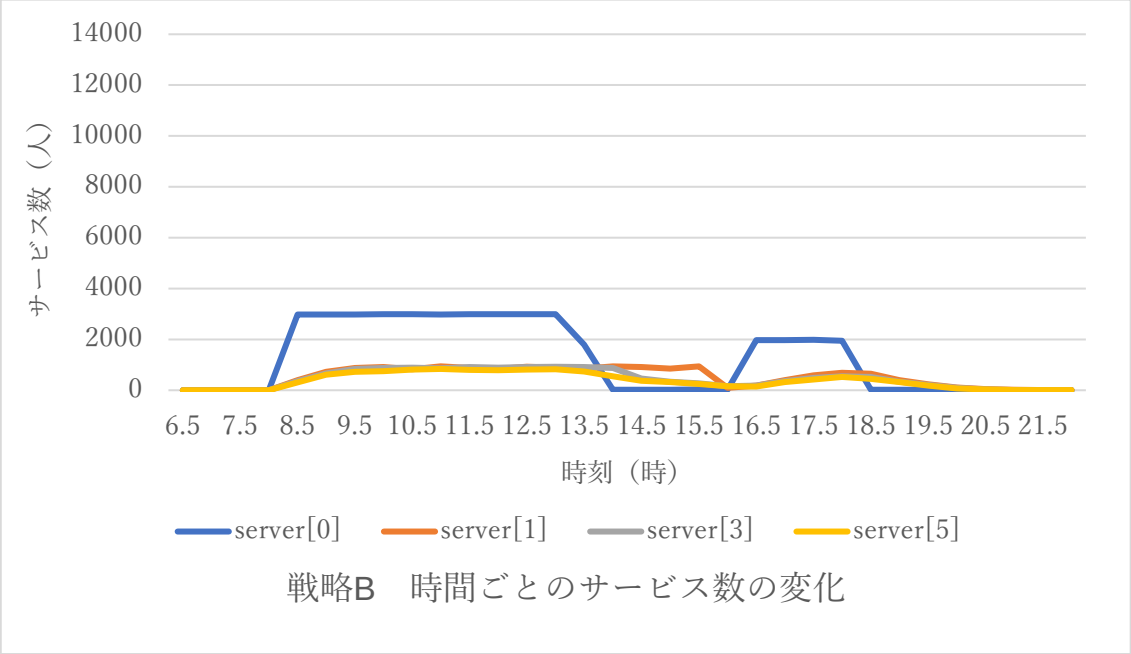


図 4.4　戦略 B 時間ごとのサービス数の変化

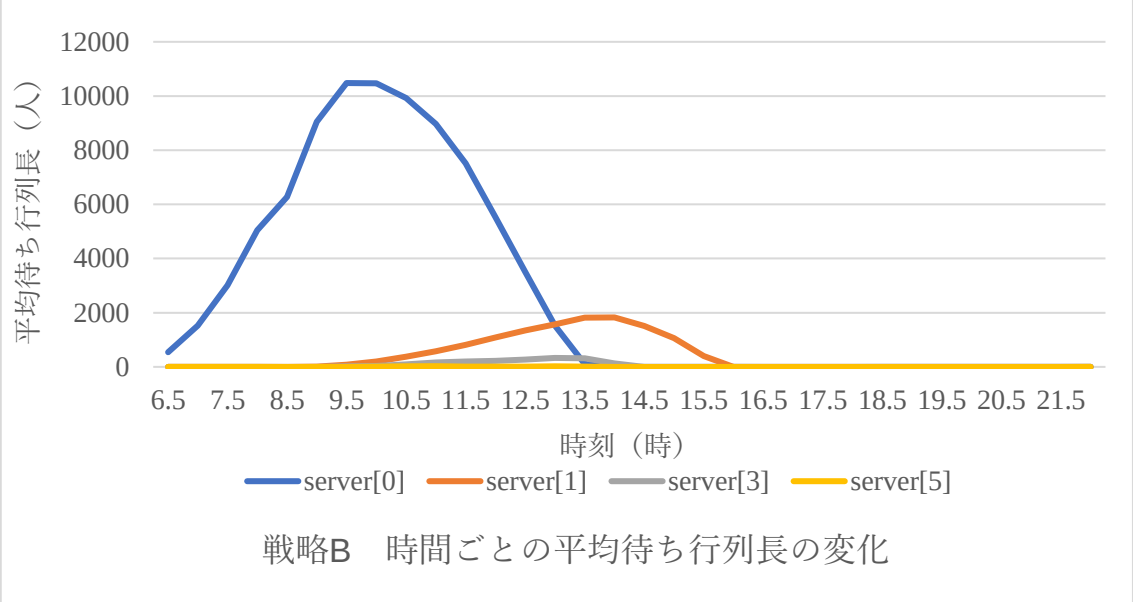


図 4.5　戦略 B 時間ごとの平均待ち行列長の変化

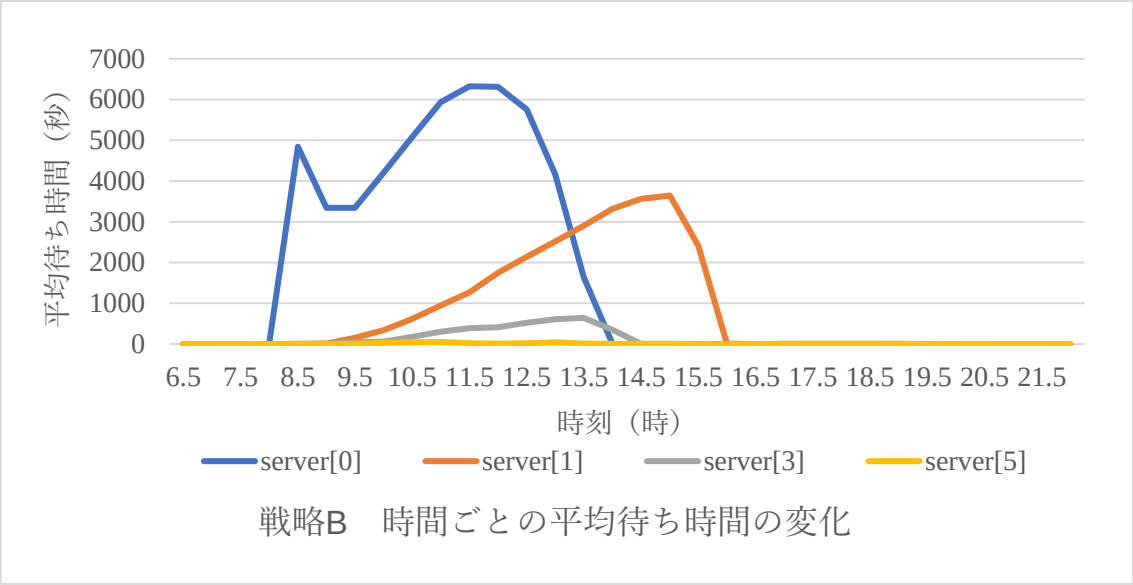


図 4.6　戦略 B 時間ごとの平均待ち時間の変化

戦略 B においては上記のような結果となり戦略 A に比べ大きく変わったように見える．また，戦略 C の結果を以下に示す．

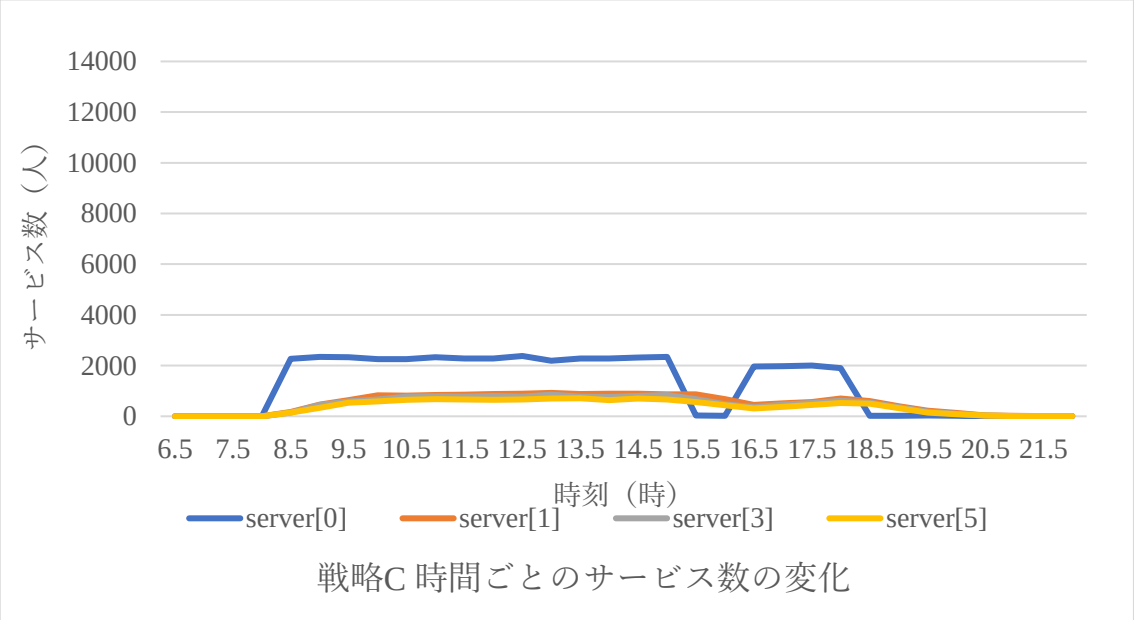


図 4.7　戦略 C 時間ごとのサービス数の変化

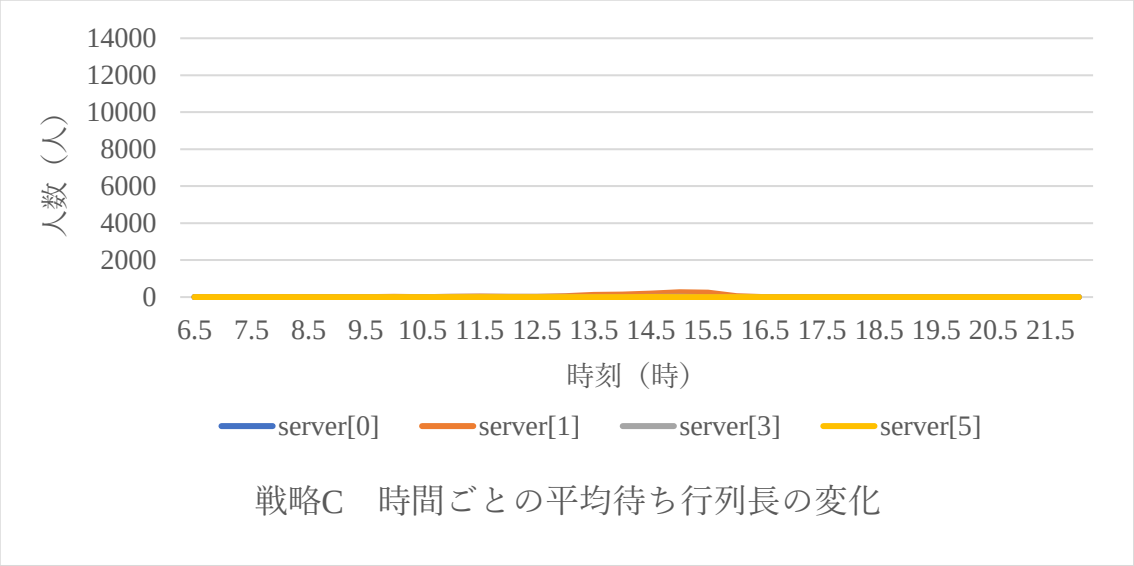


図 4.8　戦略 C 時間ごとの平均待ち行列長の変化

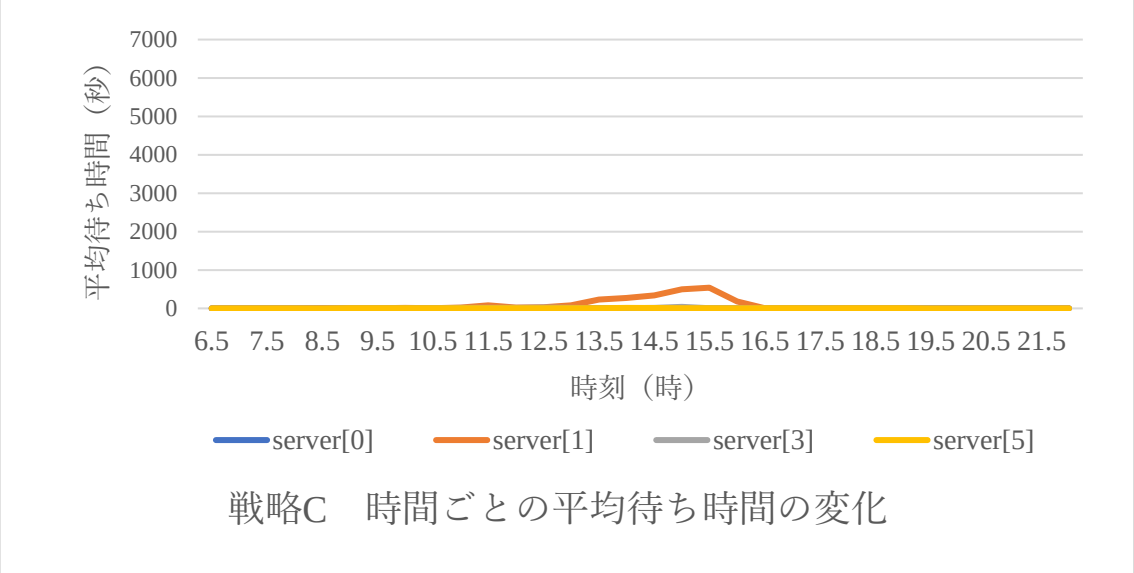


図 4.9　戦略 C 時間ごとの平均待ち時間の変化

各戦略の結果を示すと上記のようになった．また，すべての戦略のシミュレーション時間内での平均総滞在時間と平均総待ち時間について比較したグラフを図 4.10 に示す．

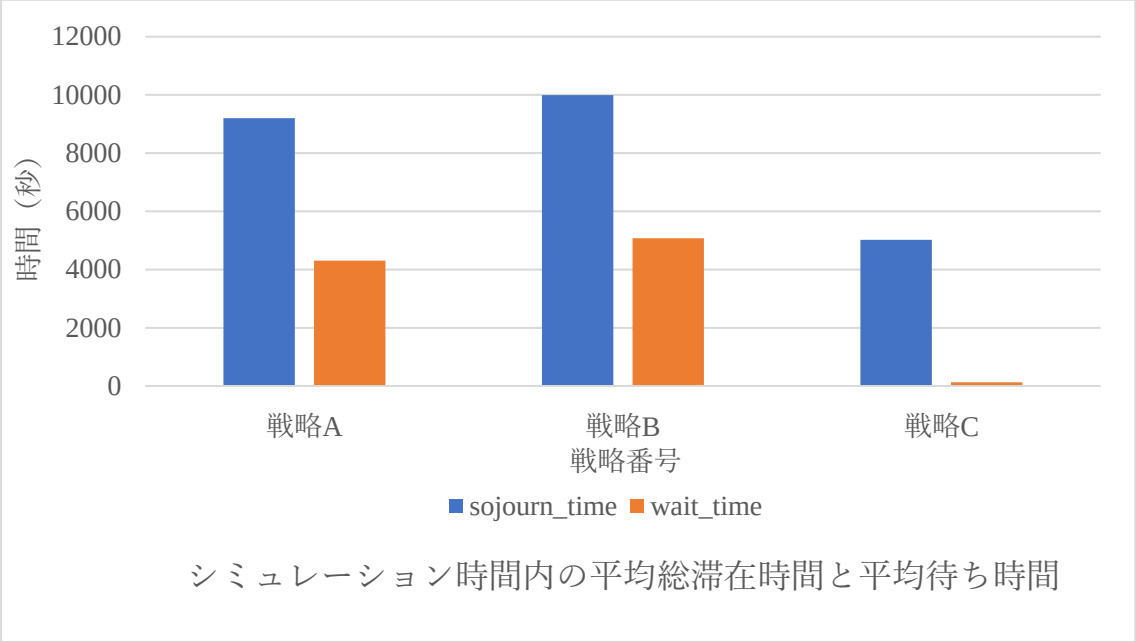


図 4.10　シミュレーション時間内での平均総滞在時間と平均待ち時間

これらの結果を用いて考察をする．

４.３　考察

まずは、戦略 A の結果である図 4.1，4.2，4.3 を用いて通常の遊園地についてわかることがある．それは，アトラクションへの訪問割合が少し変わるだけで平均待ち行列長と平均待ち時間には大きな差が出ているということである．アトラクション１とアトラクション２に注目してみると訪問割合が０．５しか変わらないのに対して，平均待ち行列長と平均待ち時間はともに最大値がおよそ２倍になることが分かる．つまり，ほかのアトラクションよりも少しだけ人気なだけで待ち行列長，時間ともに大幅に増えることが確認できる．

次に戦略 A と戦略 B を比較する．サービス数について戦略 B は入場口で 30 分に 3000 人しか入れないように制限をしたことにより，入場口での数を抑制することはできているが，ピークのサービス数である時間帯が長くなり常に一定以上の客が入場口にたまっている状態となっている．それに関連して入場口での平均待ち行列長と平均待ち時間はともに大幅に長くなっている．これより

入場口での待ち時間はかえって長くなっていることが分かる．園内の各アトラクションについてみてみるとサービス数は変わらないが，平均待ち行列長と平均待ち時間はともに減っている．これらのことより，園内に入るまでの時間は長くなっているが一度園内に入ってしまえば戦略 A よりも快適にサービスを受けることができる．しかし図 4.10 を用いて全体について考えると，総待ち時間と総待ち時間については双方ともに少しではあるが増加している．これらのことを総合して考えると，園内を快適に過ごせるようになった分よりも多く入場口での待機時間が長くなっていることが示されている．全体を評価する限り戦略 B は戦略 A よりも劣っているように見える．

では次に，戦略 C はどうか．この戦略は昼のピークの意時間帯までの到着数をチケットにより制限しているので，一定の時間帯でのサービス数のおおよその数は決まっている．そのことより入場口での開園直後のピークも戦略 A よりも少ない．また入場口と園内での各アトラクションの待ち行列長は大幅に減っている．これは戦略 B と同様にアトラクションに来る客を時間帯ごとに一定にしたことによるものである．平均待ち時間に関してもそれらのことより大きく抑えられ，他のどの戦略よりも短く抑えられている．それらのことは図 4.10 より総滞在時間，総待ち時間ともに顕著に出ており，戦略 A に比べ滞在時間はおよそ%に 60，待ち時間に至っては 5%以下に抑えられている．

今回のシミュレーションでは各アトラクションを最大で 1 度しか回らないが，実際では同じアトラクションに何度も乗る人もいるだろう．そのことを考えれば，今回のシミュレーションで総滞在時間が抑えられれば，一日により多くの回数アトラクションに乗ることができるであろう．また，待ち時間が短ければ短いほど，客の不満はたまりにくくテーマパークでの満足度の向上に寄与することが予想できる．

第５章

結論

本論文は、テーマパークの混雑度を模擬するためのシミュレーターを作成しそのシミュレーターを用いて、各戦略下でのシミュレーションを行いそのシミュレーションでの最終的結果や時間ごとのパラメータの変化を用いて考察をし、それらの戦略の評価を行った。

前章で行った考察で戦略 A と戦略 B の評価により、入場制限という戦略の評価が示されている。この戦略は、園内においては、待ち時間などの面で高い満足度を得られるであろうことが示されたが、入園までに大きく待たせることとなり総合的な満足度には繋がらない、それどころか満足度の低下を招くことが示された。

戦略 C はチケットによる到着数の制限という戦略であった。これは現実でいうところの予約制に近い。この戦略は戦略 B と違い、客側が入場できる時間帯を知っているので、入場時に長い時間待たせることはあまりない。さらに、時間帯ごとに来る客の数も制限されているため、園内に滞在する客の数も多くなりすぎず、アトラクションでの待ち時間も戦略 A に比べて、大きく削減されている。これらのことを総合して考えれば戦略 C は全体的な満足度を大幅に上げることができると評価できた。

上記の結果より入場制限という戦略は今回の場合においては逆効果であり有効な戦略であるとは言えず、安易にこの戦略をとった際はかえって混乱を招くという結論に至った。また、チケット制限という戦略は各面において客に対する満足度を上げることができた。もし欠点を上げるとするならば、チケットを予約しなければならないという点と、一日遊ぼうと思うならば競争率の高くなるであろう、早い時間帯のチケットを取らなくてはいけないという点であろ

う。

これらの戦略以外にもテーマパークにはファストパスのようないろいろな戦略がとられている。今後の課題としてこれらの戦略の評価であったり、乗客数などの各アトラクションの条件の変更、食事処の設定であったりとシミュレーターの改善が必要である。またそれらの戦略を組み合わせた場合に、テーマパーク全体にどのような効果があるのかの調査することにより、より効果的な手法を見つけて、テーマパークの満足度の向上に寄与することができると考える。

参考文献

［1］佃勇平，須貝康夫　“優先搭乗券の発見枚数調整によるテーマパークの混雑緩和”，情報処理学会大 76 回全国大会（2014）

［2］清水仁，松林達史，納谷太　“混雑不和状態の遊園地における待ち時間削減手法のシミュレーション評価”，特集論文「エージェント技術とその応用 2017」https://www.jstage.jst.go.jp/article/tjsai/32/5/32_AG16-F/_pdf

[3] 清水仁　“遊園地の混雑度シミュレーション”，数理システム　ユーザーコンファレンス 2018

[4] 小田島佳織，畑晶博，後藤雄介　“遊園地におけるアトラクションの選択方法による満足度の比較”，数理解析研究所講究録第 1887 巻　2014 年，http://www.kurims.kyoto-u.ac.jp/~kyodo/kokyuroku/contents/pdf/1887-05.pdf

[5] 増田靖　“混雑制御―ディズニーランドのジレンマ―”，日本オペレーションズ学会 http://www.orsj.or.jp/archive2/or63-8/or63_8_460.pdf

[6] 藤野直輝，小島一晃，田和辻可昌，村松慶一，松居辰則　“テーマパーク来場者に対する満足度向上に向けた混雑情報提供法の検討”，第２９回日本人工知能学会年次大会，2015 年

[7] 清水仁，松林達史，納谷太，澤田宏　“遊園地におけるアトラクション選択モデルとそのパラメータ推定手法”，特集論文「知能創発とネットワーク」

[8] 清水仁，松林達史，納谷太，澤田宏　“遊園地シミュレーションにおける許容限界モデル”，　第 33 回日本人工知能学会年次大会，2019 年

[9] 辺見 和晃，コンピュータの中の人工社会　“来場者に優しいテーマパーク混雑緩和問題と情報の共有”，pp.124-139,共立出版(2002)

[10] 柳田靖,鈴木恵二　“複雑ネットワークモデルにおけるテーマパーク問題に関する考察―ネットワークモデルの比較法”，情報処理学会論文誌, Vol. 50, No. 1, pp. 437–446 (2009)

謝辞

最後までなかなか結果の出ない研究を暖かな目で見守り，プログラムや本論文を書く上で適切な指導を賜った指導教官の塩田教授に感謝いたします.

また，始まった当初より右も左もわからなかった時より丁寧に教えてくれた白木さんにも，厚く御礼を申し上げ感謝の意を表します.

付録

テーマパーク内用待ち行列ネットワークシミュレーターのソースコード

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

#define N 80000
#define NC 0 //30分あたりの入場者数の上限．ただし，NC=0とすると入場制限なしとなる．
#define K 20

#define _USE_MATH_DEFINES
#define rnd() (1.0 / (RAND_MAX + 1.0)) * rand()


double average_inter_arrival[16]，sojourn_time，num_departure，num_block;
double start_time，next_arrival，end_time，measurement_time，buffer,all_time;
double num_customer,server_flag,num_server，way_service_time;
double sum_all_wait,count_ID0,count_ID,count_ID2,no_arrival;
int num_move;
double measurement_time2,measurement_interval;
double measurement_time_queue,num_measurement_queue,
measurement_interval_queue;
double sum_num_visit，sum_sojourn_time，num_departure，total_sojourn_time,
total_wait_time;
FILE *output1，*output2，*output3，*output4;
double time_enter[N] = {0};

//客を表現する構造体
typedef struct {
```

```
        double arrival_time[K];
        double sojourn_time;//総滞在時間
        double all_wait_time;//総待ち時間
        double wait_time[K];
        double service_time[K];
        double ID;

        int schedule[K];//訪問予定のサーバーを格納
        int pass[K];//訪問済のサーバーを格納 上記のコピー
        int num_visit;//いくつのサーバーを訪問したか
        int visit_attractions;//いくつのサーバーを訪問するか

} customer ;

//アトラクションを模擬する構造体
struct server{
        double time_in_attraction;
        double average_wait_time;
        double prob;
        double service_time;
        double num_service;//サービス数
        double num_arrival;//
        int num_customer;//サーバー内の客総数
        double sum_num_customer;//サーバー内の客総数
        double departure_time;//サーバーの出発時間
        int num_block;//何回あふれたか
        customer queue[N];//並んでいる客の構造体の格納用

        //double queue_length;//待機人数
        //int flag_busy;//仕事をしているか否か
};

struct server server[K];

//テーマパーク内を移動中の客を模擬する構造体
struct way{
```

```
        int exist;
        double arrival_time;
        double departure_time;
        //double service_time;
        customer customer;//一人分のみ

};
struct way way[N];

//イベントサーチ
double eventsearch(double *time){
        int i,  event;

        i=0;
        *time = end_time;
        event = -1;

        for(i = 0; i < K; i++){
                if(server[i].num_customer != 0 && *time > server[i].departure_time){
                        *time = server[i].departure_time;
                        event = i;
                }
        }

        if(*time > next_arrival){
                *time = next_arrival;
                event = K;
        }

        if(num_move > 0 && *time > way[0].departure_time){
                *time = way[0].departure_time;
                event = K + 1;
        }

        if(*time > measurement_time2){
                *time = measurement_time2;
```

```
                event = K + 2;
        }

        if(*time > measurement_time_queue){
                *time = measurement_time_queue;
                event = K + 3;
        }

        return event;
}

//テーマパーク内を移動中の客を次のアトラクションへの到着順にソーティングするサブルーティン
void MoveSort (){
        int i,k;//way.customer[num_move-1].arrival_timeにサーバーの退去時間をdepatureで格納する
        for(i = num_move - 1 ; i > 0 ; i-- ){//後ろの客と前の客の退去時間の比較
                if( way[i].departure_time < way[i-1].departure_time){//後ろの客<前の客の退去時間
                        k = way[i].departure_time;
                        way[i].departure_time = way[i-1].departure_time;
                        way[i-1].departure_time = k;

                }else{

                        break;
                }
        }
}

//テーマパーク内を移動中の客の1人がアトラクションに到着後に実行
void way_decrease (){
        int i;
        num_move = num_move - 1;
        for(i = 0 ; i < num_move ; i++){
                way[i].customer = way[i+1].customer;
```

```
                way[i].departure_time = way[i+1].departure_time;
                way[i].arrival_time = way[i+1].arrival_time;
        }
}

//テーマパークに到着した新規客の設定を行うためのサブルーティン
customer decide (double time){
        customer new_customer;
        int k,i,r;
        int flag[K] = {0};

        new_customer.visit_attractions = 0;
        new_customer.schedule[0] = 0;//始めにサーバー0を訪問する
        new_customer.num_visit = 0;
        new_customer.ID = count_ID;
        ++count_ID;

        if(rnd() <= server[new_customer.schedule[0]].prob){
                ++new_customer.visit_attractions;
                flag[0] = 1;
        }
        else flag[0] = 0;

        for(i=1; i < K; i++){
                new_customer.schedule[i] = rand()%(K - 1) + 1;
                r=1;
                while(r > 0){
                        r = 0;
                        for(k=1; k <= i ;    k++){
        if(new_customer.schedule[i]==new_customer.schedule[i-k]){
                                        r++;
                        }
                        }
                        if(r > 0){
                                new_customer.schedule[i] = rand()%(K - 1) + 1;
                        }
```

```
                }
                if(rnd() <= server[new_customer.schedule[i]].prob){
                        ++new_customer.visit_attractions;
                        flag[i] = 1;
                }
                else flag[i] = 0;
        }

        for(i=0; i < new_customer.visit_attractions; i++){
                while(flag[i] == 0){
                        for(k=i; k < K-1;    k++){
                                flag[k] = flag[k+1];
                                new_customer.schedule[k] =
new_customer.schedule[k+1];
                        }
                }
        }

        for(i=0; i < new_customer.visit_attractions ; i++){
                new_customer.pass[i] = new_customer.schedule[i];
                new_customer.wait_time[new_customer.schedule[i]] = 0;
        }

        next_arrival = - average_inter_arrival[(int) (time/3600)] * log(1-rnd()) +
time;//ここで次回到着予定を作る
        if(next_arrival > end_time - 1800) next_arrival = end_time + 1;

        return (new_customer);
}

//測定用サブルーティン
void measurement(double time){
        int i;
        printf("Result at %f oclock¥n",   time/3600 + 6);
        fprintf(output1,"%f,",time/3600 + 6);
        fprintf(output2,"%f,",time/3600 + 6);
```

```
        fprintf(output4,"%f,",time/3600 + 6);
        for(i=0;i < K ;i++){//0～K-1までの繰り返し　waittimeの初期化
                if(server[i].num_service > 0){
                   if(i < 3) printf("%d %f
(%f %f %f)¥n",i,server[i].average_wait_time/server[i].num_service,server[i].average_w
ait_time,server[i].num_service,server[i].prob);

        fprintf(output1,"%f,",server[i].average_wait_time/server[i].num_service);
                        fprintf(output2,"%f,",server[i].num_service);
                }
                else{
                   if(i < 3) printf("%d %d
(%f %f %f)¥n",i,0,server[i].average_wait_time,server[i].num_service,server[i].prob);
                        fprintf(output1,"%d,",0);
                        fprintf(output2,"%f,",server[i].num_service);
                }

        fprintf(output4,"%f,",server[i].sum_num_customer/num_measurement_queue)
;
                server[i].average_wait_time = 0;
                server[i].num_service = 0;
                server[i].sum_num_customer = 0;
        }
        fprintf(output1,"¥n");
        fprintf(output2,"¥n");
        fprintf(output4,"¥n");
        printf("num_measurement_queue = %f¥n",num_measurement_queue);
        if(num_departure > 0){
                printf("sojourn_time = %f,　wait_time = %f,　# of visits
= %f¥n",sum_sojourn_time/num_departure,sum_all_wait/num_departure,sum_num_vis
it/num_departure);

        fprintf(output3,"%f,%f,%f,%f¥n",time/3600,sum_sojourn_time/num_departur
e,sum_all_wait/num_departure,num_departure);
        }
        else fprintf(output3,"%f,　%d,　%d ,%f¥n",time/3600,0,0,num_departure);
```

```
		printf("num_move = %d,	# of arrivals = %f,	# of departures = %f,	# of illegal arrivals = %f (%f)¥n",num_move,count_ID,count_ID2,no_arrival,	count_ID0);
		total_sojourn_time += sum_sojourn_time;
		total_wait_time += sum_all_wait;
		sum_sojourn_time = 0;
		sum_all_wait = 0;
		num_departure = 0;
		sum_num_visit = 0;
		num_measurement_queue = 0;
		measurement_time2 = time + measurement_interval;
}

//各アトラクションの待ち行列長測定用サブルーティン
void measurement_queue(double time){;
		int i;
		for(i = 0 ; i< K ;i++){
			server[i].sum_num_customer += (double) server[i].num_customer;
		}
		++num_measurement_queue;
		measurement_time_queue = time + measurement_interval_queue;
		return;
}

//客のアトラクションへの到着
void arrival (int flag,	double time,	customer inside_customer){//次に行くサーバーの指定を必ずする

		++inside_customer.num_visit;
		inside_customer.arrival_time[flag] = time;
		inside_customer.service_time[flag] = - server[flag].service_time *	log(1-rnd());//サービス時間が指数の場合
		//side_customer.service_time[flag] = server[flag].service_time;//サービス時間を一定値にする場合はこの文を用いる

		if(server[flag].num_customer ==	N){
			++server[flag].num_block;
```

```
                printf("A new customer is blocked (%d %f %f)¥n",
flag,time,inside_customer.ID);
                return;
        }
        ++server[flag].num_customer;

        if(server[flag].num_customer == 1){//サーバー内の客総数が１のとき
                server[flag].queue[0] = inside_customer;
                server[flag].queue[0].wait_time[flag] = 0;
                if(NC > 0 && flag == 0 && inside_customer.ID >= NC &&
time_enter[(int)(inside_customer.ID - NC)] + 1800 > time){
                        server[flag].departure_time =
time_enter[(int)(inside_customer.ID - NC)] + 1800 +
inside_customer.service_time[flag];
                }
                else if(time < start_time){
                        server[flag].departure_time = start_time +
inside_customer.service_time[flag];
                }
                else server[flag].departure_time = time +
inside_customer.service_time[flag];
                if(flag == 0 && server[flag].departure_time < start_time)
printf("*** %f¥n",server[flag].departure_time);
        }
        else{//空いてなければ列の最後尾に並べる
                        server[flag].queue[server[flag].num_customer - 1 ] =
inside_customer;
                        ++inside_customer.num_visit;
        }
}
//客のアトラクションからの退去
void departure(int flag,   double time){
        int i;

        server[flag].average_wait_time += server[flag].queue[0].wait_time[flag];
        if(flag == 0){
```

```
                time_enter[(int) server[flag].queue[0].ID] = time;
                ++count_ID0;
        }
        ++server[flag].num_service;
        if (server[flag].queue[0].num_visit == server[flag].queue[0].visit_attractions ||
time > end_time - 600){//次に行くところがない場合
                server[flag].queue[0].sojourn_time = time -
server[flag].queue[0].arrival_time[server[flag].queue[0].pass[0]];//総滞在時間の算出
pass
                if(time > measurement_time){
                        sum_sojourn_time += time -
server[flag].queue[0].arrival_time[server[flag].queue[0].pass[0]];//総滞在時間の算出
pass
                        server[flag].queue[0].all_wait_time = 0;
                        for(i=0;i < server[flag].queue[0].visit_attractions ;i++){//0～
K-1までの繰り返し　waittimeの初期化
                                server[flag].queue[0].all_wait_time +=
server[flag].queue[0].wait_time[server[flag].queue[0].pass[i]];
                        }
                        sum_all_wait += server[flag].queue[0].all_wait_time;
                        sum_num_visit += server[flag].queue[0].num_visit;
                        ++count_ID2;
                        ++num_departure;
                }
        }

        else{//次に行くところがある場合
                for(i=0; i < server[flag].queue[0].visit_attractions -
server[flag].queue[0].num_visit ;i++){//スケジュールを一つずつ前にずらす
                        server[flag].queue[0].schedule[i] =
server[flag].queue[0].schedule[i+1];
                }
                ++num_move;
                way[num_move-1].customer = server[flag].queue[0];
                way[num_move-1].departure_time = time +
server[flag].time_in_attraction - way_service_time    *    log(1-rnd());
```

```
                if(way[num_move-1].departure_time > end_time) way[num_move-
1].departure_time = end_time - 300;
                MoveSort();
        }

        -- server[flag].num_customer;//サーバー内の客を減らす

        if(server[flag].num_customer > 0){
                for(i=0;i < server[flag].num_customer; i++){//キュー内に人がいる
とき1つずつ前にずらす
                        server[flag].queue[i] = server[flag].queue[i+1];//サーバー内
の客の移動
                        if(i==0){//先頭の客を（queue[0]←queue[1]）サーバー客
にした時の処理 外に出す
                                server[flag].departure_time = time +
server[flag].queue[0].service_time[flag] ;
                                if(NC > 0 && flag == 0 && server[flag].queue[i].ID
>= NC && time_enter[(int)(server[flag].queue[i].ID - NC)] + 1800 >
server[flag].departure_time){
                                        server[flag].departure_time =
time_enter[(int)(server[flag].queue[i].ID - NC)] + 1800;
                                }
                        }
                }
                server[flag].queue[0].wait_time[flag] = time -
server[flag].queue[0].arrival_time[flag];//その客の待ち時間に足す
        }else{
                server[flag].departure_time = end_time + 1;
        }
}

//初期化
void initilization(){
        int i;
        double proportion[16];
        double total;
```

```
        char f1[256], f2[256], f3[256], f4[256];
//これより下で各アトラクションでの訪問確率を設定
        server[0].prob = 1.0; //入場ゲートでの訪問確率
        server[1].prob = 0.4;//アトラクション1の訪問確率
        server[2].prob = 0.4; //アトラクション2の訪問確率
        server[3].prob = 0.35; //アトラクション3の訪問確率
        server[4].prob = 0.35; //アトラクション4の訪問確率
        server[5].prob = 0.3; //アトラクション5の訪問確率
        server[6].prob = 0.3; //アトラクション6の訪問確率
        server[7].prob = 0.25; //アトラクション7の訪問確率
        server[8].prob = 0.25; //アトラクション8の訪問確率
        server[9].prob = 0.25; //アトラクション9の訪問確率
        server[10].prob = 0.25; //アトラクション10の訪問確率
        server[11].prob = 0.25; //アトラクション11の訪問確率
        server[12].prob = 0.2; //アトラクション12の訪問確率
        server[13].prob = 0.2; //アトラクション13の訪問確率
        server[14].prob = 0.2; //アトラクション14の訪問確率
        server[15].prob = 0.2; //アトラクション15の訪問確率
        server[16].prob = 0.1; //アトラクション16の訪問確率
        server[17].prob = 0.1; //アトラクション17の訪問確率
        server[18].prob = 0.1; //アトラクション18の訪問確率
        server[19].prob = 0.1; //アトラクション19の訪問確率
        for(i=1;i < K ;i++){//0～K-1までの繰り返し　waittimeの初期化
                server[i].num_service = 0.;//サービス数を足す
                server[i].num_customer = 0.;
                server[i].sum_num_customer = 0.;
                server[i].average_wait_time = 0.;//アトラクションごとの待ち時間
を格納してアトラクションごとの総待ち時間を算出
                server[i].time_in_attraction = 600;
        }
        total = 40000;

        proportion[0] = 0.05; //6時
        proportion[1] = 0.10; //7時
        proportion[2] = 0.3; //8時
        proportion[3] = 0.15; //9時
```

```
        proportion[4] = 0.1; //10時
        proportion[5] = 0.05; //11時
        proportion[6] = 0.05; //12時
        proportion[7] = 0.001; //13時
        proportion[8] = 0.001; //14時
        proportion[9] = 0.001; //15時
        proportion[10] = 0.1; //16時
        proportion[11] = 0.1; //17時
        proportion[12] = 0.001; //18時
        proportion[13] = 0.001; //19時
        proportion[14] = 0.00001; //20時
        proportion[15] = 0.00001; //21時
        for(i = 0; i< 16; i++){
                if(total * proportion[i] > 0) average_inter_arrival[i] = 60.*60./(total *
proportion[i]);
                printf("¥n %d %f¥n",i,average_inter_arrival[i]);
        }
        count_ID = count_ID2 = count_ID0 = no_arrival = 0;
        sum_all_wait = 0;
        num_departure = 0;
        num_customer = 0;
        sum_sojourn_time = num_departure = sum_num_visit = 0;
        num_measurement_queue = 0;
        sprintf(f1,"result/wait_wc%d.csv",NC);
        sprintf(f2,"result/num_wc%d.csv",NC);
        sprintf(f3,"result/total_sojourn_wc%d.csv",NC);
        sprintf(f4,"result/queue_wc%d.csv",NC);
        output1 = fopen(f1,   "w");
        output2 = fopen(f2,   "w");
        output3 = fopen(f3,   "w");
        output4 = fopen(f4,   "w");

        total_sojourn_time = 0;
        total_wait_time = 0;
}
```

```
double erlang_c(double a,   int c){
        int i;
        double z,   p0;

        z = 1;
        p0 = z;
        for(i=1; i<c; i++){
                z = z * a/(double) i;
                p0 += z;
        }
        z = z * a / (c - a);
        return z/(z + p0);
}

int main(){
        int    event,flag,i;
        double sim_time;
        customer inside_customer;

        for(i=0; i < K; i++){
                server[i].service_time = 2; //サーバーの平均サービス時間
        }
        server[0].service_time = 0.1;
        way_service_time = 600; //道の平均サービス時間
        end_time = 60.*60.*16.+1;
        //end_time = 300;
        measurement_time = 0; //精度を上げるためにはじめの結果は捨てる.
        measurement_interval = 1800;
        measurement_interval_queue = 1;
        start_time = 3600*2.;
        measurement_time2 = measurement_interval;
        measurement_time_queue = measurement_interval_queue;

        initilization();
        sim_time = 0;
        srand(5);
```

```
        next_arrival = - average_inter_arrival[0] * log(1-rnd()) + sim_time;

        while(event >= 0){
                event = eventsearch(&sim_time);
                flag = event;
                if(event == K){
                        inside_customer = decide(sim_time);
                        if(inside_customer.visit_attractions > 1)
arrival(inside_customer.schedule[0],sim_time,inside_customer);
                        else ++no_arrival;
                        //printf("a ¥n");
                        }
                else if(event == K + 1){

        arrival(way[0].customer.schedule[0],sim_time,way[0].customer);
                        way_decrease();
                        //printf("b ¥n");
                }
                else if(event == K + 2){
                        measurement(sim_time);
                        //printf("c ¥n");
                }
                else if(event == K + 3){
                        measurement_queue(sim_time);
                        //printf("d %f¥n",sim_time);
                }
                else if(event != -1){
                        departure(flag,sim_time);
                        //printf("c ¥n");
                }
        }
        printf("sojourn = %f wait = %f¥n",total_sojourn_time/count_ID2,
total_wait_time/count_ID2);
        fprintf(output3,"sojourn = %f wait = %f¥n",total_sojourn_time/count_ID2,
total_wait_time/count_ID2);
```

```
        return 0;
}
```