「マルチメディア，分散，協調とモバイル
(DICOMO2021)シンポジウム」 令和3年6月-7月

オープンデータを用いた バス路線推定と多路線描画手法の改良

水谷 颯吾[1]　金 鎔煥[1]　山本 大介[1]　高橋 直久[1]

概要：本研究は，地理的に正確なバス路線図の自動生成システムに着目する．従来の路線図は人の手によって作成されており，地図上に描画する場合は道路データとバスの路線データを対応づける必要があるなど，課題設定がより複雑になる．また，路線図は路線同士が重ならないようずらして描画されていることや，並走する路線が右左折時になるべく交差しないよう配置順が考慮されており，路線図作成には時間と労力がかかることが課題である．
本研究の先行研究では，停留所座標と道なり道路（ストローク）を用いたバス路線図の自動推定を実現した．この手法では，停留所座標から停留所ノードを生成し，停留所ノード間を右左折の回数が最も少なくなる道なり優先探索手法を使用してバス路線を推定している．また，複数の路線描画の際，重複区間の配置順を動的に求め右左折時の交差が少ない路線の地図上描画を実現した．
しかし，この手法では停留所座標から停留所ノードを生成しているため，高速道路の高架下などに停留所がある場合誤った場所に停留所ノードを生成してしまい，正確なバス路線推定ができないという課題がある．また，この手法では 10 路線の描画に留まり，より多くの路線に対応するためには様々な路線の形状を考慮する必要がある．
上記の課題を解決するために，本研究では停留所座標と道路リンクに加え，高速道路や国道などの道路の形状データである道路クラスを用いることで，より精度の高い路線の経路推定を実現した。提案システムにおけるバス路線の推定精度に対する評価実験を行った．40 路線に対して実際の路線との一致率を求め，従来手法の推定精度が約 91.6%，提案手法の推定精度が約 93.2%であり，1.6%の精度向上が見られた．
また，30 路線の描画に対する定性的な評価を行った．見つかった課題としては路線同士の色区別がわかりずらい，バスターミナル付近や 4 路線以上重なった時の視認性の低下が挙げられる

An Improvement of Bus Route Estimation and Multi-route Drawing Method using Open Data

SOGO MIZUTANI[1]　YONGHWAN KIM[1]　DAISUKE YAMAMOTO[1]　NAOHISA TAKAHASHI[1]

1. はじめに

我々は，電車やバスなどの公共交通機関を利用する際に路線図を見て行き先までの移動方法を考えることが多い．もともとの路線図は，路線同士の関係や路線内の順序を視覚的に伝える性格を持ち，方向や距離などの地理的正確性を求めるのではなく路線上の相対的な駅の位置や路線の接続関係を第一としたデフォルメ路線図である．また，地理的に正確な路線図も存在し、正確な位置情報をもとに地図上に描画することでデフォルメ路線図よりも停留所周辺の施設の情報や目的地までの経路を得ることができるため，目的地までの行き方を考える際に非常に便利である．しかし，地理的に正確な路線図は，路線数が多くなるほど視認性が低くなる可能性がある．また，路線図は並走する路線同士が重ならないようにずらして描画されており，路線同士の右左折時に生じる交差がなるべく少なくなるよう配置順が考慮されている．現在，バス路線図は人の手によって描画されており，路線図の作成には多くの時間と労力がかかっている．

[1] 名古屋工業大学 大学院工学研究科
Graduate School of Engineering, Nagoya Institute of Technology

近年は GoogleMaps[1] などの多くの Web マップが普及しており地理的に正確な地図に様々な情報を付随させることが有意になった。また，路線バスや地下鉄などの交通系データが公開されており，web マップに適用することでバスロケーションシステムなどのバス利用者の利便性を向上させるシステムの提供が可能となる．

本研究では，地理的に正確なバス路線図の自動生成に着目する．路線バスは大型駅のバスターミナルから多くの路線が各地域に運行しており，路線バスの利用は電車と比べても非常に複雑である．そのため，路線に対する正確な地理的情報を加えることで，目的地までの路線バスの利用の複雑さに対する有効性が期待できる．これまでの研究として，地図を用いた路線図の自動生成に関する研究は，デフォルメ路線図の自動生成の研究と比較するとあまり取り組まれてこなかった．理由として，路線図は紙媒体で作成されることが多く，視認性の高いデフォルメ路線図が主流となっていること，また, 地図を用いる場合は道路のデータとバスの路線データを対応づける必要があるなど，より課題設定が複雑になるためであると考えられる．

本研究の先行研究として服部 [2] は，停留所座標と道路ストロークを用いて地図上にバス路線を描画するシステムを提案した．このシステムでは，バス停留所座標と道路データから路線を推定し，得られた路線を web 上に描画している．

しかし，課題として以下の点が挙げられる．

課題 1 道路データとバス停留所の緯度・経度データを結び付ける際，バス停留所から最も近い道路上にバス停留所ノードを生成する．この時，高速道路などの高架下に停留所がある場合，誤って高速道路上にバス停留所ノードを生成してしまい正しい経路が推定できなくなる．

課題 2 先行研究では 10 路線の描画に留まり，より多くの路線に対応するためには様々な路線の形状を考慮する必要がある．

そこで，本研究では以下の特徴を持つシステムを提案する．

特徴 1 路線生成機能

停留所座標と道路リンクに加え，高速道路や国道などの道路の形状データである道路クラスを用いることで，より精度の高い路線の経路推定を実現する．

特徴 2 バスストローク (BS)/バスストロークフラグメント (BSF) 生成機能

道路ストロークとバス経路データを用いて BS を生成，バス路線間の重なりを BSF として生成する機能を実現する．

特徴 3 路線配置/路線描画機能

得られた，BSF を基に路線の配置順を決定し地図ライブラリである leaflet を用いて路線を web 上に描画する．

2. 提案システムの概要

この章では，提案システムに使用する道路データ及びバスデータの構造について述べ，提案システムが持つ機能とその構成について説明する．

2.1 道路データ

OpenStreetMap が提供するデータは，表 1 に示すリンクのデータテーブルがある．ノードとは道路ネットワーク上の交差点を表し，ノード間を緯度・経度座標系列であるポイントに沿ってつないだものが道路リンクである．リンクは始点ノードから終点ノードまでを結んだものであるため道路ネットワークとしては道路の方向が与えられた有向グラフとなる．道路の形状はポイント系列に沿ってジオメトリ型のアークで示される．

また，各道路にはそれぞれ道路クラスが割り当てられており，道路の種別を道路クラスにより識別することが可能である．道路クラスとそのクラスが表す道路の種別の例を表 2 に示す．

表 1 リンクテーブル

カラム名	データ型	説明
ID	Integer	リンク ID
clazz	Integer	道路のクラス
source	Integer	始点ノード ID
target	Integer	終点ノード ID
x1	Double	始点ノードの経度
y1	Double	始点ノードの緯度
x2	Double	終点ノードの経度
y2	Double	終点ノードの緯度
km	Double	ノード間の長さ
geom_way	geometry(LineString)	リンクの形状 (アーク)

表 2 道路クラス

clazz	kmh	説明
11	120	motoway(高速道路)
12	30	motorway link(高速道路への連絡路)
13	90	trunk(国道)
14	30	trunk link(国道への連絡路)
15	70	primary(主要地方道)

2.2 ストロークデータ

ストロークデータを表 3 に示す．ストロークとは認知心理学に基づきグループ化された道路ネットワークの集合であり，道なりに続く道路をリンク列で表す．図 1 の例だと，左側はノード同士を結んだリンク集合の道路ネットワークであり，右側は複数のリンクから構成される色分けされたストローク集合のストロークネットワークである．

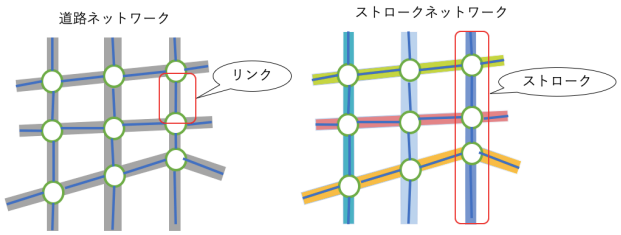

図 1　ストロークネットワーク

表 3　ストロークテーブル

カラム名	データ型	説明
id	Integer	ストローク ID
link_ids	Text	含まれるリンク ID 系列
stroke_length	Double	ストロークの長さ
arc_series	geometry(LineString)	ストロークの形状

2.3　バスデータ

本研究では，プロトタイプとして名古屋市営バスの路線図を生成する．そのため，名古屋市交通局 [3] の平成 29 年度のオープンデータに基づいたバスデータを使用してバスの停留所座標，系統，停留所の順序テーブルを作成する．

2.3.1　停留所データ

停留所データのテーブルを表 4 に示す．オープンデータに停留所 ID を付与させることで系統や順序データ作成や路線の自動推定の際の利便性を図る．緯度・経度座標は，停留所ノード作成の際に用いられる．

表 4　停留所テーブル

カラム名	型	説明
id	Integer	停留所 ID
busstop_name	varchar	名称
lat	Double	経度
lng	Double	緯度
noriba_info	varchar	補足
geom_way	geometry(POINT)	停留所座標

2.3.2　系統データ

系統データのテーブルを表 5 に示す．同路線でも行き先や方向、経由地などが違うため，系統は路線における具体的な運行ルートを示したものとなる．系統テーブルには系統内の運行区間に対する起点，終点の情報等が格納されており，系統コード，路線コード，方向コードの 3 つによって系統情報が一意に識別される．

2.3.3　停留所順序データ

停留所の停車順序のテーブルを表 2.3.3 に示す．停留所順序テーブルには系統の運行区間に対する起点から終点までに通過する停留所の順序データが格納されている．

2.4　バスストローク (BS)

複数路線の重複区間の配置順を動的に求めるために，バスストロークを定義する．バスストローク（BS）とは，1 つの路線において経路が通過するストロークの区間を示し，バス路線の道路リンクデータの集合とストロークデータから生成する．もともとバス路線の経路データは道路リンク集合で保存されているが，バス路線は直線的で大きな道を通過することが多いため，通過するストローク集合である BS で表現したほうが少ないデータ数かつ容易に路線の重複区間が割り出せるという利点がある．BS の例を図 2 に示す．この例では路線は 5 つのストロークを通過しているため，路線を BS1 から BS5 の 5 つの BS で表すことができる．

表 5　系統テーブル

カラム名	型	説明
route_code	Integer	系統コード
line_code	Integer	路線コード
dir_code	Integer	方向コード
route_name	varchar	系統記号
start_point	varchar	起点停留所名
end_point	varchar	終点停留所名
info	varchar	経由等情報

カラム名	型	説明
route_code	Integer	系統コード
line_code	Integer	路線コード
dir_code	Integer	方向コード
route_name	varchar	系統記号
start_point	varchar	起点停留所名
end_point	varchar	終点停留所名
info	varchar	経由等情報

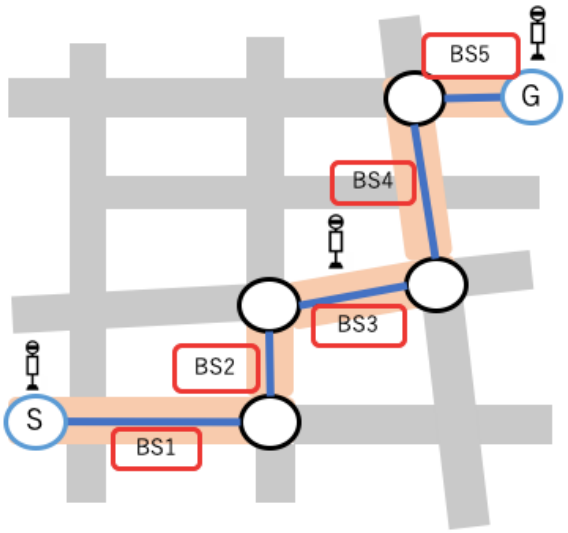

図 2　バスストローク

2.5　バスストロークフラグメント（BSF）

バスストロークフラグメント（BSF）とは，複数の路線

において重複を考慮したストロークの区間を示す．BSF の生成には各路線の BS が用いられ，各 BS の重複区間を分割することですべての路線を BSF で表現することができる．BSF を生成することで，複数路線の重複区間が容易に判定できるため配置順決定から路線描画までの計算量を短縮できるという利点がある．図 3 にバスストロークの例を示す．この例では，左側の図にあるように路線 A の BS（BS(a)1）と路線 B の BS（BS(b)1，BS(b)，2BS(b)3，BS(b)4）を入力とする．ここで，路線 A の BS(a)1 と路線 B の BS(b)3 は重複区間があるため重複区間とそうでない区間に分けることで 3 つの BSF を生成する．したがって，路線 A は（BSF1，BSF2，BSF3），路線 B は（BSF4，BSF5，BSF2，BSF6）で構成される．また，路線描画の際は BSF2 の配置順を決定することで視認性の高い路線図が生成される．

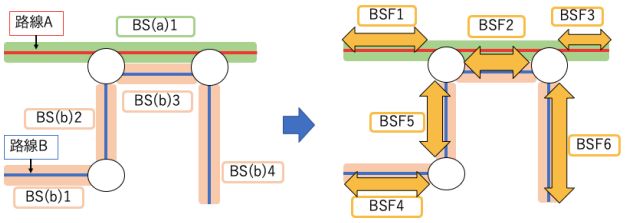

図 3　バスストロークフラグメント

2.6　提案システムの構成

提案システムの構成を図 4 に示す．提案システムは 4 つの機能から構成されており，1 つ目は経路生成機能，2 つ目は BS 及び BSF 生成機能，3 つ目は路線配置機能，4 つ目は描画機能である．

経路生成機能では，ノード生成において道路 DB から道路リンクと道路クラス，オープンデータからバス停留所座標を入力とし、停留所ノード及び分割リンクを生成しバスデータベース上に格納する．その後，道路 DB からストロークネットワーク，ユーザが指定した路線から系統の停留所ノードをバスデータベースから得ることにより，停留所ノード間の経路探索を行う．

BS および BSF 生成機能では，まず路線経路データと対象となるストロークから BS を生成し，バス DB に格納する．次に，複数路線の BS データを入力とし，BSF を生成しバス DB に格納する．同時に，各路線ごとの経路を BSF を用いて表す路線 BSF を生成しバス DB に格納する．

路線配置機能では、ユーザが指定した複数の路線 ID とバス DB から取得した BSF 及び路線 BSF を入力とし，路線が重複する BSF 内の路線の順序を隣接する BSF とのなす角度に基づき求める．

最後に描画機能において，路線配置機能で求めた順序結果を基に GeoJson 形式の描画データを生成し Leaflet を用いて web 上に描画される．

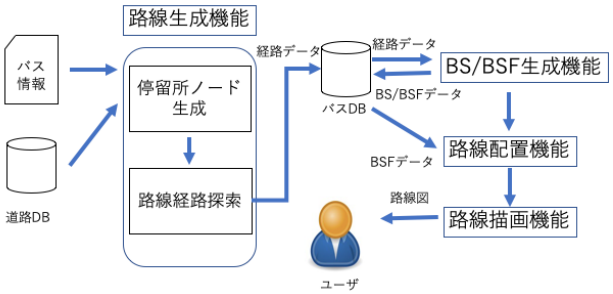

図 4　システム構成図

3.　提案システムの実現法

3.1　道路クラスを用いた路線経路推定手法

停留所の座標系列と道路クラスを用いて路線経路を推定する手法を述べる．主な流れとしては，停留所ノードの生成，分割リンクの作成，路線探索である．

3.1.1　停留所ノード生成

停留所の緯度経度座標と道路リンク，道路クラスを入力とし，限定した道路クラスにおける停留所に最も近い道路リンク上に停留所ノードを生成する．例として以下のように変数を定義する．

入力

- $L = (l_1, l_2, \cdots, l_i)$ ：リンク集合
- $Clazz$ ：道路クラス
- Bus_point ：バス停留所の緯度経度座標

出力

- Bus_node ：停留所ノード

停留所ノード生成の手順

手順 1　Bus_point と L の中で $Clazz$ が高速道路（$Clazz = 11$），もしくは高速道路への連絡路（$Clazz = 12$）以外の近傍リンク集合 L_near を取得する．

手順 2　$L\_near = (L\_near_1, L\_near_2, \cdots, L\_near_i)$ の中で、最近傍リンク L' を求める．

手順 3　Bus_point から L' の中での最近点を求め，Bus_node として，テーブルに格納する．

以上の手順において，PostGIS 関数 [4] を使用する．手順 1 では， 2 つのジオメトリを取得し，互いが指定した距離内にある場合に true を返す $ST\_DWithin$ を用いて L_near を取得する．この時，対象とするリンク集合を $Clazz$ が高速道路（$Clazz = 11$），もしくは高速道路への連絡路（$Clazz = 12$）以外に限定し誤ったリンク上に停留所ノードができるのを防ぐ．本研究では，距離を 0.002 と設定しリンク集合を取得した．

手順 2 では， 2 つのジオメトリから距離を算出する $ST\_Distance$ を用いて，L_near の中で最も距離が短いリンクを L' として取得する．

手順 3 では， Bus_point のポイントに対する L' のラインストリング上の最近点が始点・終点のどちらに近いかの

割合 r を $ST\_LineLocatePoint$ を用いて取得する．得られた割合 r と L' から $ST\_LineInterpolatePoint$ を使って最近点を，Bus_node として，テーブルに格納する．以下に停留所ノードのデータ形式を表 6 に示す．

表 6　停留所ノードテーブル

カラム名	データ型	説明
id	Integer	停留所 ID
link_id	Integer	最近傍道路リンク ID
node_lat	Double	停留所ノードの緯度
node_lng	Double	停留所ノードの経度
ratio	Double	リンク上の割合

3.1.2　分割リンクの生成

作成した停留所ノードをリンク上のノードとしてノード間の経路探索するためには，道路リンクを分割する必要がある．本研究では，道路リンクを停留所ノードで分割したリンク集合を分割リンクと定義する．図 5 の例では，右の道路リンクの Link2 を停留所ノードで分割してリンクの数を 3 から 4 つに増やしている．

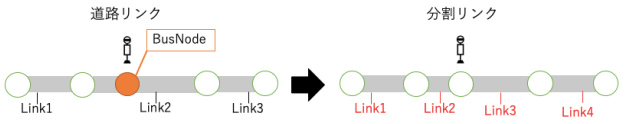

図 5　分割リンク

3.1.3　経路探索による路線の推定

以下に経路推定に用いる記号を定義し，路線の経路データを生成する手順を示す．

- $ID = (id_1, id_2, \cdots, id_n)$ ：求める路線の停留所 ID リスト
- $Node = (N_1, N_2, \cdots, N_n)$　：ノード ID リスト
- $V \ni (s, g)$ ：始点と終点の組み合わせ
- $R = (r_1, r_2, \cdots, r_n)$ ：経路データリスト

手順 1　停留所順序テーブルから指定した系統，路線，方向コードに対する ID を取得する．

手順 2　停留所ノードテーブルから ID に対応する停留所ノードを取得後，$Node$ に追加する．

手順 3　$v = (N_i, N_{i+1}) \in V$ において，経路テーブルに $(s, g) = (id_i, id_{i+1})$ または $(s, g) = (id_{i+1}, id_i)$ のレコードが存在するか調べる．

手順 4　存在する場合は，取得した経路データを r_i とする．

手順 5　存在しない場合は，v における経路探索を行い r_i を求める．

手順 6　すべての経路が見つかった場合，R を経路テーブルに格納する．

手順 3 では，すでに経路がテーブル上にある場合を調べることにより，経路探索が必要ない区間をみつけて全体の探索時間を削減することができる．手順 5 では，道なり優先探索を行う．道なり優先探索とは，通過するストロークの数が最も少ないかつその中で距離が最も短い経路を採用する手法である．生成した経路テーブルの形式を表 9 に示す．

表 7　経路テーブル

カラム名	データ型	説明
route_code	Integer	系統コード
line_code	Integer	路線コード
dir_code	Integer	方向コード
route_name	varchar	系統記号
s_order	Integer	始点側の順序番号
g_order	Integer	終点側の順序番号
s_gid	Integer	始点側の停留所 ID
s_name	varchar	始点側の停留所 ID
g_gid	Integer	終点側の停留所 ID
g_name	varchar	終点側の停留所 ID
geom_way	geometry(LineString)	経路の形状
link_ids	varchar	リンク ID 系列

3.2　BS 生成手法

BS 生成の手順を述べる．以下の記号を定義する．

- $L = (l_1, l_2, \cdots, l_n)$ ：路線経路に含まれているリンク ID リスト
- $S = (s_1, s_2, \cdots, s_m)$　：すべてのストローク ID リスト
- s_{l_i}　：ストロークに含まれているリンク ID リスト
- l_{S_i} ：l_i が含まれるストロークの ID リスト

手順 1　経路テーブルから指定路線のデータを取得し，通過するリンク ID のリスト L を作成する．

手順 2　ストロークの ID 集合 S と，それに含まれるリンク ID リスト s_{l_i} をストロークテーブルから取得する．

手順 3　路線の進行方向順に l_S を求め，同一の l_S となるリンク集合を BS として生成する．

この手法において，手順 3 では同じストロークを通るものを一つの BS として登録する．したがって、以下の図 6 の路線であれば各 BS のリンク集合は $BS1_{l_i} = (10, 11, 12)$，$BS2_{l_i} = (22, 23, 24)$，$BS3_{l_i} = (31, 32)$ となる．以下の表 8 に BS テーブルの形式を示す．

表 8　BS テーブル

カラム名	データ型	説明
route_code	Integer	系統コード
line_code	Integer	路線コード
dir_code	Integer	方向コード
num	Integer	順序番号
bs_id	Integer	BSID
stroke_id	Integer	ストローク ID
geom_way	geometry(LineString)	BS の形状
link_ids	varchar	BS に含まれるリンク ID 系列

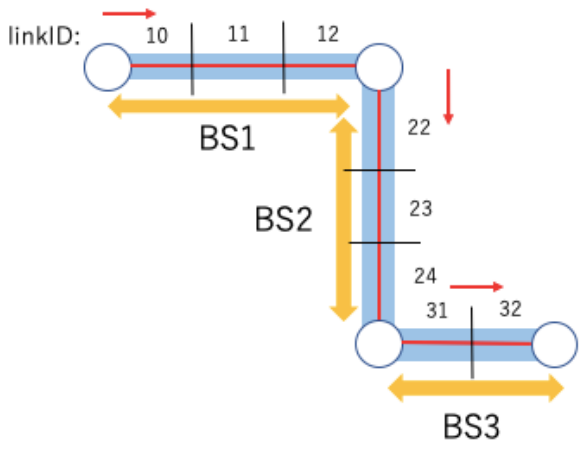

図 6　BS 生成

3.3　BSF 生成手法

以下に BSF 生成の手順を述べる．以下の記号を定義する．

- $R(l_i)$ ：i 番目のリンクに含まれる路線集合
- $R.data = (r_1, r_2, \cdots, r_n)$　：n 本の路線のデータリスト
- $r.BS$　：路線 r を構成する BS リスト
- $r.BSF$　：路線 r を構成する BSF リスト
- $r.BS_n.link\_ids$　：路線 r の n 番目の BS に含まれるリンク ID リスト

手順 1　BS テーブルから指定した複数路線の BS データを取得する．

手順 2　各リンクに含まれている路線の集合 $R(l_i)$ を求める．

手順 3　リンクに含まれる路線情報に基づき BS を分割し BSF を生成する．

手順 4　路線が通過する BSF 系列を求め，路線 BSF テーブルに格納する．

BSF 生成において，図 7 の例を用いて説明する．手順 1 において，a，b，c の 3 本の路線のデータリストから以下の図の入力のような路線 a，b，c の BS と BS に含まれているリンクリストを取得する．手順 2 では，リンク 10(a，b)，リンク 11(a，b)，リンク 12(a) というようにリンクごとにそのリンクを通る路線 $R(l_i)$ を求める．手順 3 では，得られた $R(l_i)$ を基に BS を分割する．分割の手順を以下に示す．

手順 1　路線の順序を基に $R(l_{(i)})$ と $R(l_{(i-1)})$ を比較し，含まれている路線が同じであれば false，違えば true を返す．

手順 2　false の場合分割はなし，true の場合 $l_{(i-1)}$ を分割位置とし BS を分割する．

手順 3　BS を分割したものを BSF として生成し BSF テーブルに格納する．

例の場合路線 a では，分割手順 1 に従うと，リンク 10，リンク 11 では false，リンク 12 では true となる．次に分割手順 2 で分割位置がリンク 11 と決まり，分割手順 3 において BSF に含まれるリンクと含まれる路線はそれぞれ BSF1=((10，11)，(a,b))，BSF2=((12)，(a)) というように BSF が生成される．同様にして他の路線に対しても BSF を生成すると，図 7 のような BSF が求まる．

手順 4 では求めた BSF を基に $BSF_a = (BSF1, BSF2)$，$BSF_b = (BSF1, BSF3, BSF4, BSF5)$，$BSF_c = (BSF6, BSF4, BSF7)$ というように各路線が通過する路線 BSF を $r.BSF$ として格納する．以下の表 9 に BSF テーブルの形式，表 10 に路線 BSF テーブルの形式を示す．

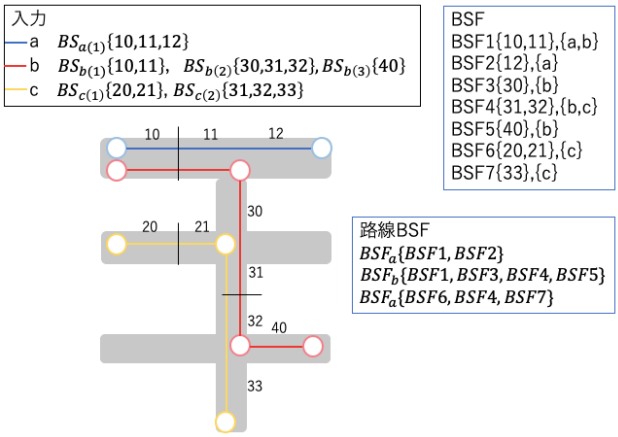

図 7　BSF 生成

表 9　BSF テーブル

カラム名	型	説明
bsf_id	Integer	BSFID
stroke_id	Integer	ストローク ID
geom_way	geometry(LineString)	BSF の形状
link_ids	varchar	リンク ID 系列
rode_codes	varchar	路線コード系列

表 10　路線 BSF テーブル

カラム名	データ型	説明
route_code	Integer	系統コード
line_code	Integer	路線コード
dir_code	Integer	方向コード
num	Integer	順序番号
bsf_id	Integer	BSFID
bsf_front	Integer	1 つ前の BSFID
bsf_behind	Integer	1 つ前の BSFID
geom_way	geometry(LineString)	BSF の形状

3.4　路線配置順決定手法

複数路線の配置順を決定するために，路線 BSF を入力とし，複数の路線に対する BSF 内の配置順を求める手法を述べる．以下に，手順を示す．

2 路線の場合

手順 1　入力された路線の BSF 系列データを取得する．

手順 2　配置順を求める必要のある対象 BSF リストを作成する．

手順 3　路線の始発側から順に対象 BSF の配置順を求める．

3 路線以上の場合

手順 1　入力路線のうち，2 路線の配置順を求める．

手順 2　現在の結果に 1 つの路線を追加し，2 路線の場合と同様に配置順を求める必要のある対象 BSF リストを求める．

手順 3　路線の起点側から順に対象 BSF の配置順を求め，必要であれば前回の配置順を利用する．

手順 4　入力路線分 2，3 を繰り返す．

対象 BSF とは，配置順を考慮しなければならない BSF のことで BSF 系列データから複数路線が通過する BSF を得ることにより決定される．決定された対象 BSF の配置順は以下のルールによって決められる．

手順 1　対象 BSF の配置順は，対象 BSF と 1 つ前の BSF のなす角度に基づき判定される．

手順 2　配置順が一意に決まらない路線は，一つ前の BSF の配置順に依存する．

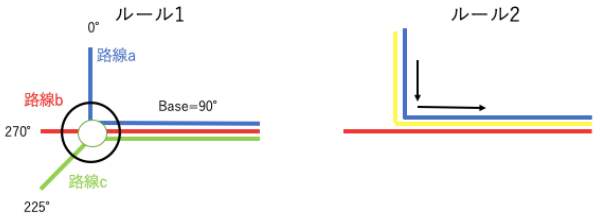

図 8　路線配置のルール

図 8 では路線配置の例を示している．ルール 1 では，始点側に接続される BSF と対象 BSF のなす角度に基づき路線をソートし路線の配置順が判定される．角度の計算に用いられるのが PostGIS 関数の ST_Azimuth 関数と degree 関数である．ST_Azimuth 関数は，ポイント 1 の鉛直線からポイント 2 への右回りの方位を北を基準にしたラジアン単位で返す関数であり，degree 関数はラジアンを度に変換する関数である．これらを用いて，始点側に接続される BSF と対象 BSF(Base) の角度を割り出す．図 8 の例の場合，(a，b，c，Base) =(0°，270°，225°，90°) となる．これらのソートの手順として，まず降順にならべると (270°，225°，90°，0°) となる．これらを，Base の値が先頭に来るまで先頭から最後尾まで回していった順序 (90°，0°，270°，225°) =(Base，a，b，c) が北から順に並べた配置順となる．ルール 2 では，青と黄色の始点側に接続される BSF と対象 BSF のなす角度が同じため，ひとつ前の BSF の配置順がそのまま対象 BSF の配置順に引き継がれる．

3.5　路線描画手法

求めた路線の配置順結果を基に描画データとなる GeoJson ファイルを生成し，ファイルを読み込むことで路線の配置順が動的に変化する路線図の描画手法について述べる．GeoJson とは地理データをエンコードするための形式で点や線，面を記述することができる．

生成された GeoJson ファイルを読み込み，Leaflet[5] を用いて Web マップ上に描画する．この際に同一区間における路線同士が重ならないように路線をずらして描画するために，ポリラインにオフセットを持たせることができる Leaflet Polyline Offset[6] を用いて BSF の描画を行う．

4.　プロトタイプシステム

4.1　プロトタイプシステムの概要

3 章で述べた各機能を有する提案システムのプロトタイプを作成した．開発には Windows10，Eclipse[7]，Java[8]，javascript，HTML，CSS，PostgreSQL[9]，PostGIS を使用した．地図データ及び道路データは OpenStreetMap[10] のデータを使用したサーバから取得し，生成したバスデータはローカルのバスデータベースから取得している．また，動作確認には Google Chrome を利用している．

4.2　プロトタイプシステムの動作

画面上部に入力パネル，下部に地図画像が配置されており，プロトタイプシステム起動時はデフォルトで中心座標が (35.1564224713,136.9229315060) でスケール値を 14（縮尺 1/35,000）とした地図画像を OpenStreetMap から取得して表示している．

4.3　プロトタイプシステムの操作手順

経路生成機能，路線配置機能の操作手順を以下に示す．生成した経路を図 9 に示す．また路線描画機能は，20 路線に対する路線描画を実行した．描画ファイルを生成後ブラウザ上で表示した結果を図 10 示す．

経路生成機能

手順 1　「道路データ」→「バスデータ」→「ストロークデータ」の順にボタンを押してデータを取得する．

手順 2　系統，路線，方向コードをテキストボックスに入力後，「系統取得ボタン」を押し停留所系列を取得する．

手順 3　「経路探索」ボタンを押し，路線の各停留所間の経路を道なり優先探索を用いて生成する．

手順 4　「従来研究」ボタンを押すことで従来手法で生成した経路，「提案研究」ボタンを押すことで提案手法で生成した経路を表示することができる．

手順 5　「格納」ボタンを押し路線の経路データをデータ

ベースに格納する．

路線配置機能

手順 1　「2 路線配置」ボタンを押し，コンソール上で 2 つの路線に対する系統，路線，方向コードを入力する．

手順 2　「描画ファイル出力」ボタンを押し，配置結果を考慮した描画ファイルを出力する．

手順 3　「路線配置」ボタンを押し，コンソール上で路線の系統，路線，方向コードを入力する．

手順 4　「描画ファイル出力」ボタンを押し，配置結果を考慮した描画ファイルを更新する．

手順 5　入力路線分手順 3，4 を繰り返す．

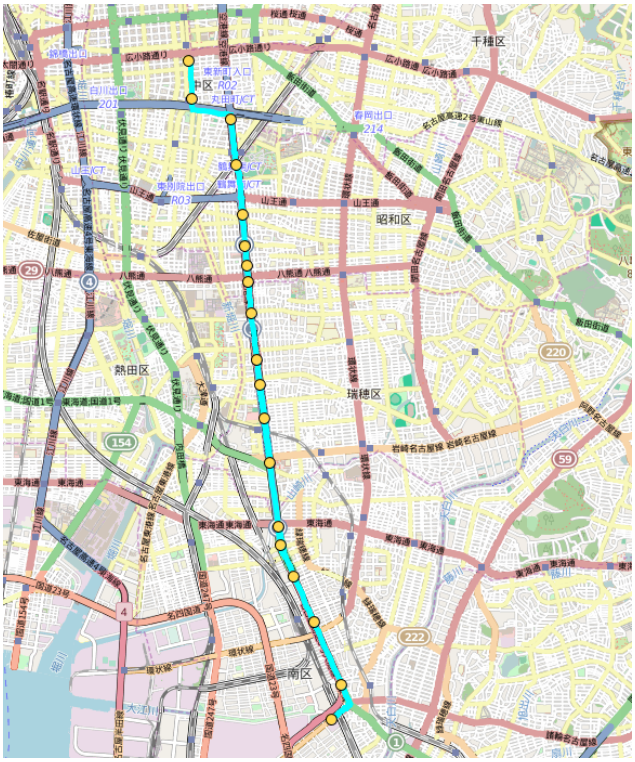

図 9　生成した経路

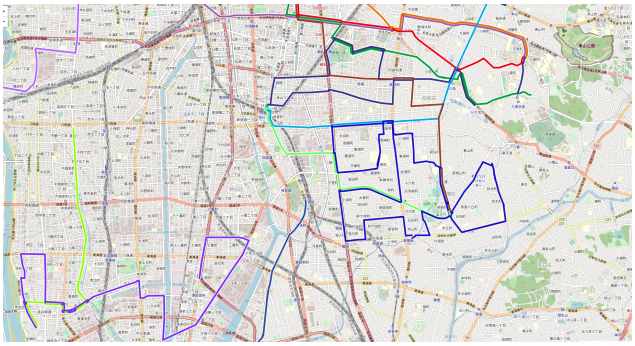

図 10　描画結果

5.　評価実験

本章では，プロトタイプシステムを用いた評価実験とその結果，考察について述べる．

5.1　評価実験 1

目的

評価実験 1 の目的は，提案手法の道路クラスを用いた停留所ノードの生成による路線経路の推定精度を従来手法と比較することにより，提案手法の停留所ノードの生成により実際のバス路線経路との一致率が向上したかを検証することである．

方法

提案手法と従来手法における路線探索の推定精度を名古屋市交通局が公開するバス路線の地図画像用いて評価する．評価方法としては，1 本を停留所と停留所の間の経路とし，実際の経路のうち，生成経路と一致している本数を求める．これらの本数から実経路に対する生成経路の推定精度を算出する．

評価対象

以下に示す評価対象により提案手法，従来手法，実経路を評価する．

- 名古屋市営バス計 40 路線

5.2　結果と考察

40 路線に対する推定精度の平均は，従来手法が 91.6%，提案手法が 93.2%であり，提案手法により推定精度が 1.6%向上した．したがって，提案手法の道路クラスを用いた停留所ノードの生成をすることで，より実路線に近い経路推定ができることが検証された． 推定精度が向上した路線は，いずれも高速道路や高速道路への連絡路沿いの路線である．これらの路線では，従来手法停留所ノードが高速道路上に生成されてしまうためインターチェンジやジャンクションに向かう経路を探索してしまい実経路から外れた経路推定をした．一方で，提案手法では停留所ノードが正しいリンクに生成されたため，高速道路などの高架下の道路リンクを通る実経路と同じ経路を生成した．また，提案手法で生成した経路と実経路が一致しなかったものの原因として，駅周辺のバスターミナルや分割リンク作成の際のアークの作成ミスが挙げられる．探索手法に関しては，道なり優先探索を使用することでほぼ実経路と同じ経路を探索できた．

5.3　評価実験 2

目的

評価実験 2 の目的は，路線描画を先行研究 [2] の 10 路線から 3 倍の 30 路線に増やして定性的な評価を行うことで，路線数を増やし描画したときの課題を発見することである．

方法

路線描画機能によりブラウザ上で描画された路線につい

て観察することで，視認性の観点から定性的に評価，考察を行う．評価によって得られたものを今後の課題として取り組む．

5.4 結果と考察

今回は 30 路線に対して路線描画を行った．以下の図 11 に 30 路線の描画結果を示す．図 11 の縮尺であると画面上に 14 路線が表示されている．この路線描画機能では任意の色分けにより路線を区別しているが，色の数が足りず似た色で違う路線を表示してしまっていることで視認性が低下していると考える．名古屋市路線バスは 600 以上の系統が存在するため，視認性を保ちながら路線を描画するためには重なりのある BSF 同士を遠い色で表すシステムの構築が今後の課題となる．また，図 12 は栄付近の描画結果をズームした画像である．この図より，3 路線以上重なりのある BSF の描画は実際の道路から離れた位置に描画してしまうことがわかる．また，バスターミナル付近は多くの路線が通過するため視認性が低下することがわかる．したがって，バスターミナル付近や路線の重なりが多くなった時に，視認性を保つ描画機能の構築が今後の課題である．

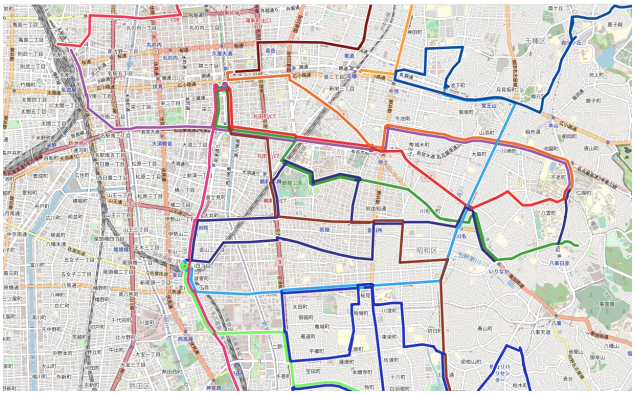

図 11 30 路線の描画結果

図 12 栄付近の描画

6. 終わりに

本論文では，地理的に正確なバス路線の自動生成のために，主に先行研究 [2] の 2 つの課題を解決する手法を提案した．1 つ目の課題は路線の自動推定の推定精度の向上であり，この課題を解決するために道路クラスを用いた停留所ノードの作成による経路推定を行った．2 つ目の課題としては，多くの路線を描画することであり，この課題を解決するために描画路線数を増やすことで課題を探した．

提案手法に基づくプロトタイプシステムを実装し，バス路線の自動推定精度に関して評価実験を行った．40 路線に対して実際の路線との一致率を求め，従来手法の推定精度が約 91.6%，提案手法の推定精度が約 93.2%であり，1.6%の精度向上が見られた．

今後の課題としては，BS 生成の改良手法が現在実装中であるため実装を完成させて描画できる路線を増やすこと，路線を増やしたときの視認性を向上することが挙げられる．

参考文献

[1] Google Maps
入手先 ⟨https://www.google.co.jp/maps/⟩ （参照 2021-05-02）.

[2] 服部真由, 山本大介, 高橋直久, オープンデータを利用したバス路線生成システム, DICOMO 2018, 8B-4, 2018.

[3] 市バス - 名古屋市交通局
入手先 ⟨https://www.kotsu.city.nagoya.jp/jp/pc/bus/⟩ （参照 2021-05-02）.

[4] PostGis
入手先 ⟨http://postgis.refractions.net/⟩(2021.1.29 参照)

[5] Leaflet - a JavaScript library for interactive maps
入手先 ⟨Leaflet - a JavaScript library for interactive maps⟩ （参照 2021-05-02）.

[6] Leaflet Polyline Offset
入手先 ⟨https://github.com/bbecquet/Leaflet.PolylineOffset⟩ （参照 2021-05-02）.

[7] Eclipse
入手先 ⟨https://eclipse.org/⟩ （参照 2021-05-02）.

[8] JAVA
入手先 ⟨https://www.java.com/ja/⟩ （参照 2021-05-02）.

[9] PostGresQL
入手先 ⟨https://www.postgresql.org/⟩ （参照 2021-05-02）.

[10] OpenStreetMap Japan - 自由な地図をみんなの手に/The Free Wiki World Map
入手先 ⟨https://openstreetmap.jp/⟩ （参照 2021-05-02）.