D-010

連続旅行計画の為の safe-region 生成方法
Safe-region generation method for continuous trip planning queries

塩澤　光貴 *　　Htoo Htoo*　　大沢　裕 *
Koki Shiozawa　　Htoo Htoo　　Yutaka Ohsawa

1. はじめに

車などの移動体が，移動しながら常に最新の結果を得る検索を連続検索という．本稿では，旅行計画問題の一つである OSR(optimal sequenced route) を対象とした連続検索を扱う．OSR 探索とは，現在地 (q)，訪問データ点カテゴリ，目的地 (d) が指定されたとき，q から指定された順に各データ点カテゴリ中の点を一つずつ経由し d に至る最短経路を求める問題である．連続検索では，検索結果が変わらない領域を生成する safe-region という考え方が提案されており，本稿では，この safe-region を OSR 探索において高速に生成する方式を提案する，また，データ点密度によらず安定した処理時間で探索可能な OSR 探索も提案する．

2.OSR 探索

本稿における旅行計画を定義する．

定義 1 (旅行計画)．M 個のデータ点カテゴリ $C_i(i \leq i \leq M)$ と検索点 q，目的地 d が与えられたとき，現在点 q から出発し，それぞれのデータ点カテゴリ C_i から $p_i(\in C_i)$ を 1 つずつ選び経由し，d に至るまでの距離が最短となる *TPR(trip planning route)* を求める．このとき，*TPR* は $R_{1\ldots M}(q)$ と表し，与えられたデータ点カテゴリを M 個すべて訪れる場合は単純に $R_M(q)$ とも表す．

2.1. 従来法

本稿では OSR 探索に Htoo ら [1] のアルゴリズムを用いる．出発地を q，途中で訪れるデータ点カテゴリ数を M，C_i を i 番目に訪問するカテゴリ，目的地 d とする．出発地 q から，順序通りに $C_1, C_2, \ldots, C_M$ に属するデータ点を探索する．最初に q から C_1 に属するデータ点を探索し，その結果見つかった C_1 のデータ点ごとに次の C_2 に属するデータ点の探索を進め，C_M まで探索した後，d に至る経路を見つける．この探索を高速化するため，A*アルゴリズムをベースとした探索法を採用する．具体的には，現在 q から k 番目までのデータ点が見つかっているとき，q から $p_k(\in C_k)$ を結ぶ道路網距離と，p_k と d 間のユークリッド距離の和が最小のノードから順に探索を行う方式である．

2.2. 提案方式

従来法 [1] では，各カテゴリーのデータ点の存在密度が処理時間に影響を及ぼす欠点がある．最初に訪れるカテゴリーの密度が他のカテゴリーの密度に比して高い場合大幅な処理時間を要する．OSR 探索では，訪問順にデータ点の探索を行うのではなく，データ点の密度が最も低いカテゴリーから順に訪問データ点を決定していくことにより高速化できることが分かっている．そこで，以下の処理手順で OSR 探索を高速化する．但し，q を p_0，d を p_{M+1} とする．

Step 1 $m = 0$，$n = M + 1$ とする．

Step 2 p_m と p_n を結ぶ最短路を A*アルゴリズムで求めた後，その経路を中心に更に探索範囲を拡大し，$C_i(m < i < n)$ に属すデータ点を各カテゴリーから 1 つずつ探す．それらの内，最も最短路からの距離が離れたカテゴリ C_k に属し，かつそのカテゴリで最初に見つかるデータ点を p_k とする．

Step 3 $k - m > 1$ であれば，$n = k$ とし Step 2 以下を繰り返す．更に，$n - k > 1$ であれば，$m = k$ とし Step 2 以下を繰り返す．

Step 4 以上の処理が終了したあと，p_i と p_{i+1} 間の距離を A*アルゴリズムで求める．

上記のアルゴリズムが行うことは，q と d の最短経路に対して最も疎に存在するデータ点カテゴリを決定し，そのカテゴリのデータ点を経由する経路から順次求めていく方式である．この処理を m，n 間のカテゴリー数が 1 つ以下になるまで再帰的に実行する．

3.safe-region 生成

本稿における，旅行計画の為の safe-region を定義する．

定義 2 (旅行計画の為の safe-region)．*SR(safe-region)* とは，検索点 q が移動したとき，訪れるデータ点カテゴリが同じとなる領域を表したものである．つまり，*SR* 内の点 q，点 q' ならば，$R_M(q) = R_M(q')$ となる．

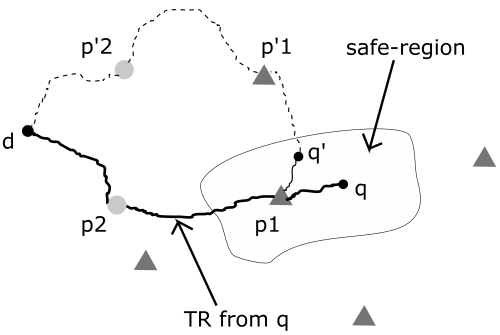

図 1: 旅行計画における safe-region

図 1 は旅行計画における SR の概念を示している．カテゴリ数 $M = 2$ で，p_1, p_2 はそれぞれ C_1, C_2 に属するデータ点を表しており，SR は検索点 q に対して，検索結果が $\{p_1, p_2\}$ となる領域を表している．SR は，q が移動し，領域外へ出た場合に再計算され，新たな SR を生成する．また最初に訪れるカテゴリに属するデータ点 p_1 に到達した場合も新たに生成され，その場合は p_1 の地点から C_2 を経て d へ至る TR の SR を生成する．

3.1.Basic 方式

SR を生成する方法として，最も基本的なアルゴリズムを述べる．最初に検索点 q から TR を求める．なお TR を求める際には 2.2 節で提案した方法を用いる．その後ダイクストラ法を用いて q から SR 領域を広げて

*埼玉大学

いく．探索は優先順位付きキュー (PQ) を用いて，PQ はレコード $< c, n, l >$ を要素として管理する．c は初期の検索点 q からの道路網距離，n は現在のノード，l は端点が n と既に訪れたノードの道路セグメントである．$R_M(q) = R_M(n)$ ならば n から更にノード展開し PQ へ追加し，すべての n で $R_M(q) \neq R_M(n)$ となるまで広げる．しかしこの方法では，SR 内のノードすべてで OSR 探索を行う必要があり，非常に処理時間がかかる．よって，3.2 節, 3.3 節で改善法を提案する．

3.2.Preceeding Rival Addition Algorithm

SR の形状は，p_1 付近にある C_1 のデータ点から影響を受ける．ここで，SR の形状に影響を及ぼすデータ点のことを rival object(RO) とする．Preceeding Rival Addition Algorithm(PRA) は，RO それぞれの TR をユークリッド距離で先に求め，TR の下限を設定することにより OSR 探索を行う回数を抑える．またユークリッド距離での探索には RTree[2] 構造を用いる．ユークリッド距離での TR 長を $L_M^E(q)$ とすると，$L_M(q) \geq L_M^E(q)$ となる．ある TR で p_1 が最初に訪れすデータ点だとする．あるノード n が SR 内に含まれているとすると，以下の式を満たせば $p_1'(\in C_1)$ は RO となり得る．

$$L_{2...M}(p_1) + 2d_N(p_1, n) \geq L_M^E(p_1') \qquad (1)$$

PRA は，初めの検索点が q，q の存在する道路セグメントを l，l の端点を a, b として q の位置で分割した道路セグメントを l_a, l_b，したとき，PQ にレコード $< d_N(q, d) + |l_a|, a, l_a >, < d_N(q, d) + |l_b|, b, l_b >$ を追加しコストが低い順に探索を進める．現在の検索点が n としたとき，式 (1) を満たす RO を C_1 が格納されている RTree を参照して探索し，条件を満たしたデータ点を RO に追加し，追加したデータ点は RTree から削除する．RO の中から，それぞれ経由した場合 n からの TR 長が最も短いものを計算し，その TR 長と n のレコードのコスト $n.c$ を比べ，$n.c$ が短かい場合のみ SR とし，さらに n の隣接ノードを e としてレコード $< n.c + |e.l|, e, e.l >$ を PQ に追加する．これを PQ が空になるまで繰り返し SR を決定する．

PRA アルゴリズムは，毎回ユークリッド距離で RO を探索し，さらに RO から d までの TR も探索しなければならないため，RO の数に比例して処理時間が増大していく．さらに RO を見つける際に，Rtree 構造から削除していく必要もある．これらの欠点を解決するため，3.3 節で Tardy Rival Addtion Algorithm(TRA) という近似アルゴリズムを提案する．

3.3.Tardy Rival Addtion Algorithm

TRA では，探索の広げ方は PRA と同様だが RO の探索法が異なり，現在ノード n からの最近傍探索で行い，p_1 を除いた C_1 の中から n の最近傍となるものを探索し，RO に追加する．ただし，TRA は RO の見落としが発生することがあり，実際の SR 領域よりも数 %広がってしまう可能性がある．しかし PRA に比べ，RO を n の近傍に制限することにより，RO の数を抑えることができ，Rtree 構造の変更もしなくて良いため，処理時間の短縮が可能となる．

4. 実験結果

提案方式の性能を評価するため，実際の道路地図を用いた評価実験を行った．ノード数 16, 284，道路セグメント数 24, 914 の地図を用いた．Java により実装し，計算機は Intel Core i7-4790CPU(3.60GHz) を用いた．

まず OSR 探索の従来法と提案方法の処理時間の比較実験を行った．図 2 はカテゴリ数 k=4 のときの処理時間を示している．従来法と提案法それぞれ 50 パターン行ったもので，横軸が経路長で縦軸が処理時間となっている．提案法では, 従来法で処理時間が大幅にかかっていたケースで処理時間を大幅に短縮できていることがわかる．

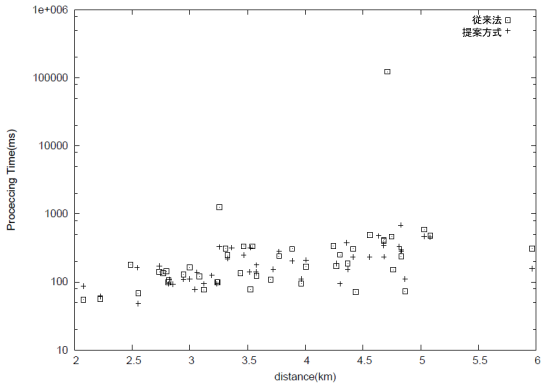

図 2: OSR 探索の処理時間

次に SR 生成の Basic，PRA，TRA，の処理時間の比較実験を行った．図 3 は k =3 のときの処理時間を示している．横軸はデータ点密度で，縦軸が生成時間となっている．Basic と比べ，PRA，TRA の方が大幅に短い時間で SR を生成できていることがわかる．

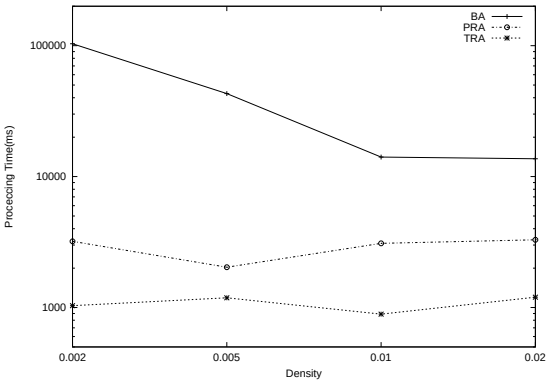

図 3: safe-region の生成時間

5. 結論

本稿では OSR 探索における処理時間の高速化方式と，safe-region 生成方式の提案をした．OSR 探索では従来法に比べ最大 100 倍処理時間を短縮することができた．safe-region 生成では，Basic に比べ PRA は大幅に処理時間を短縮することができた．さらに，TRA は safe-region の精度は数パーセント誤差が生じるが，さらに処理時間を短縮することができた．

参考文献

[1] H.Htoo, Y.Ohsawa, N.Sonehara, M.Sakauchi,“Optimal Sequenced Route Query Algorithm Using Visited POI Graph”,WAIM2012(LNCS 7418),pp.198-209,2012.

[2] A.Guttman,“R-Trees: A dynamic index structure for spatial searching”, Proc.ACM SIGMOD Conference on Management of Data,pp.47-57,1984