多階層のカテゴリ分類を用いたSkySR問合せの効率化について

佐々木勇和[†,††]　　石川　佳治[†]

† 名古屋大学大学院情報科学研究科
†† 名古屋大学未来社会創造機構
E-mail: †yuya@db.ss.is.nagoya-u.ac.jp, ††ishikawa@is.nagoya-u.ac.jp

あらまし　ビッグデータの特徴の一つは，多種多様なデータであり，レストランなどの地図上の PoI (Point of Interest) も様々な属性値を保持するのが一般的になっている．近年では，PoI に多階層のカテゴリが付与されており，単純に目的のカテゴリの PoI までの経路を探索する場合においても，ネットワーク距離と意味的な距離の 2 つのスコアを考慮する必要がある．筆者らはこれまでに，スカイライン検索の考えを取り入れた SkySR(Skyline Sequenced Route) 問合せを提案し，効率的なアルゴリズムである BSSR 法を提案した．この検索では，ユーザは始点と順序付けされた複数の PoI カテゴリを指定し，ネットワーク距離と意味的な距離の両方の観点において支配されない経路を検索する．本稿では，BSSR 法のコスト解析と最適化を目的とする．また，実データを用いたシミュレーション実験において，BSSR 法が計算時間を大きく削減できることを確認する．

キーワード　経路探索，スカイライン検索，道路ネットワーク

1. はじめに

道路ネットワークにおける経路探索は広く普及しており，スマートフォンやカーナビゲーションシステムによる問合せは一般的になっている．本研究では，目的地となる終点を指定する問合せではなく，レストランなどの地図上の PoI (Point of Interest) のカテゴリを指定する問合せに着目する．ユーザは始点から指定したカテゴリに属する PoI までの経路を検索する．今日では，多くの PoI 情報が Foursquare(注1) などで公開されている．PoI のカテゴリを指定する問合せのうち，最も基本的なものは**最近傍問合せ**である．最近傍問合せでは，指定したカテゴリに属する PoI のうち，最も始点から近いものを検索結果とする．最近傍問合せの拡張として，**シーケンスド経路問合せ**がある．シーケンスド経路問合せでは，順序付けしたカテゴリ集合を指定し，そのカテゴリの PoI を順序通りに通過する経路を対象とし，最もネットワーク距離が短いものを問合せ結果とする．

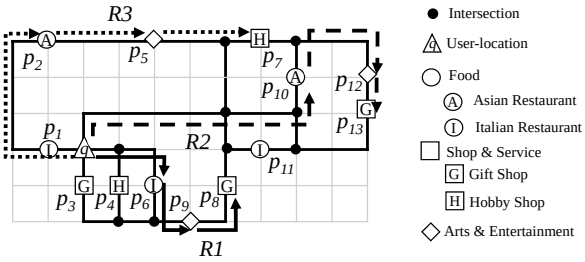


図 1: 道路ネットワークと PoI の例

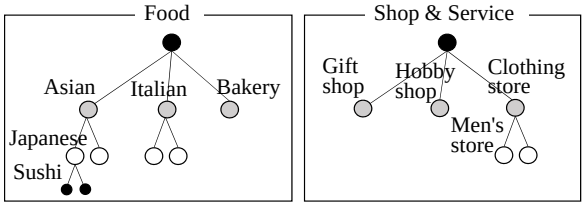


図 2: 木構造のカテゴリ分類

例 1　図 *1* は，道路と *PoI* からなるグラフであり，*q* がユーザの現在地，*p* が *PoI*（丸が *food*，四角が *shop&service (s&s)*，菱形が *arts&entertainment (a&e)*）を示している．正方形の *1* 辺は長さ *1* とする．現在地を始点として，⟨*food, a&e, s&s*⟩ を順に訪問したいという要求に対して，シーケンスド経路問合せでは $R1$ を出力する．

既存のシーケンスド経路問合せでは，カテゴリの分類は一階層と想定している．しかし，PoI のカテゴリは多様化しており，Foursquare のように木構造のような多階層であることが一般的である（図 2 参照）．例えば，Foursquare のカテゴリ木では，“food” はサブカテゴリとして，“Asian restaurant”, “Italian restaurant”, “bakery” をもっている．シーケンスド経路問合せにおいても，より詳細なカテゴリを指定することは可能である．先ほどの例 1 において，food を Asian restaurant などと指定することにより，それに適合した最短路を求めることができる．しかし，詳細なカテゴリを指定すると，ネットワーク距離が非常に長くなってしまう可能性があり，問合せ結果がユーザにとって好ましいとは限らない．また，ユーザは，どの PoI の影響により，ネットワーク距離が長くなったかを問合せ結果から判断できないため，ユーザ自身がカテゴリの指定条件を変更しても，ネットワーク距離を効率的に短くすることは難しい．

そこで，筆者らは，指定したカテゴリと経路上の PoI の意味的な距離と経路のネットワーク距離を指標とした **Skyline Sequenced Route (SkySR)** 問合せを提案した [7]．SkySR

(注1)：https://developer.foursquare.com/categorytree

問合せでは，意味的な距離とネットワーク距離を経路の指標として，他の問合せ結果の経路より意味的な距離およびネットワーク距離のどちらかが小さければ（スカイライン検索における支配されていなければ），ユーザにとって有益であると考え，その経路は SkySR 問合せの問合せ結果に含まれる．これにより，様々なカテゴリの組合せのシーケンスド経路問合せを実行せずに，一度にユーザにとって有益な全ての経路を出力可能である．

筆者らはこれまでに効率的に SkySR 問合せを実現する **Bulk Skyline Sequenced Route Algorithm (BSSR 法)** を提案した．BSSR 法では，まず，SkySR 問合せのネットワーク距離の上限値を最近傍問合せを基とした方法で求める．その後，その上限値よりネットワーク距離が短い経路のみを探索する．このとき，上限値を更新していくことで，計算範囲を削減する．本稿の大きな目的は，BSSR 法のコスト解析とそれに基づく最適化である．最適化として，まず優先度付きキューの最適化を行なう．ダイクストラ法のように始点から最も近い節点から探索範囲を拡張すると，上限値が更新されづらく，探索範囲が広がってしまう．そこで，ネットワーク距離だけではなく，意味的な距離と経路内の PoI 数を考慮して，経路の優先度を決める．さらに，BSSR 法では，既に探索済みの経路を重複して探索する可能性がある．そこで，一時的に結果をキャッシュしておくことで，無駄な探索を省くことを可能とする．

提案手法の評価として，実際の道路図と PoI を用いたシミュレーション実験を行なう．提案手法は，単純な手法に比べて，実行時間を大きく削減できることを確認する．

論文の構成は以下の通りである．まず，2. にて関連研究，3. にて問題定義についてそれぞれ述べる．その後，4. にて提案手法について説明し，5. にて提案手法の性能評価を行う．最後に，6. にて本稿をまとめる．

2. 関 連 研 究

本章では，まずシーケンスド経路問合せとグラフ上のスカイライン経路検索について説明する．

OSR (optimal sequenced route) 検索 [8] は，順序付けしたカテゴリ集合を指定し，そのカテゴリの PoI を順序通りに通過する経路を対象とし，最もネットワーク距離が短いものを問合せ結果とする．OSR 検索は，最初に提案されたシーケンスド経路問合せである．論文 [8] では，道路ネットワークを対象として OSR 検索法として，ダイクストラベース法と PNE 法を提案している．SkySR 問合せを実現する最も単純な手法は，OSR 検索を繰り返し実行する方法である．また，OSR 検索のインデックスとして様々な方法が提案されている [2], [6], [9]. 論文 [9] では，ネットワークボロノイ図を事前に構築することで，節点にそれぞれのカテゴリに対して最も近い PoI を計算なしで求めることができる．論文 [6] では，ユークリッド距離を用いて不必要な PoI の探索を防ぐ．このとき，それぞれの PoI 間のユークリッド距離を計算するのはコストが大きいため，R 木を構築し，計算の効率化を行なっている．論文 [2] では，PoI 間の最短路を事前に計算しておくことで，処理を効率化している．論文 [2], [6] では，終点を指定することを想定しているため，本稿のシーケンスド経路問合せとは若干異なる．本研究では，オンライン検索により実現するため，これらのインデックスは用いない．

グラフ上のスカイライン経路検索に関する研究も盛んに行われている [1], [4], [5], [10], [11], [13]．スカイライン経路検索のアルゴリズムは，Hansen [3] が二基準最短経路問題として初めて提案した．この種のスカイライン経路検索では，グラフの枝に様々なコスト（移動距離，移動時間，信号機の数など）が割り当てられており，経路は複数のコストをもつ．本研究においても，ネットワーク距離と意味的距離という 2 つのコストを用いるスカイライン経路検索である．しかし，これらの研究では，指定した始点と終点間の経路検索を想定している．複数の PoI を指定する問合せを対象とするスカイライン経路検索は，著者らの知る限り我々の先行研究以外存在せず，単純に解決することはできない．

3. 予 備 知 識

本稿では，連結グラフ $G = (V, E)$ を想定する．V および $E \subseteqq V \times V$ は，それぞれ節点集合および枝集合を表す．このグラフは，道路ネットワークや PoI 情報から作成されると想定する．節点 $v \in V$ のうち，いくつかの節点はカテゴリ c をもつものとし，カテゴリをもつ節点を PoI とする．$v.c$ は節点 v のカテゴリを表す．カテゴリはカテゴリ木に属しており，カテゴリ木は C 個あるとする．$ct(c_i) = \langle c_1, c_2, \ldots, c_i \rangle$ は，カテゴリ木において，カテゴリ c_i から根までの経路を表す．PoI がカテゴリ c_i に属している場合，$ct(c_i)$ の内の全てのカテゴリに属しているとする．例えば，ct(sushi restaurant) = ⟨food, Asian restaurant, Japanese restaurant, sushi restaurant⟩ となるため，カテゴリが寿司店である PoI はレストラン，アジアンレストラン，日本食レストランにも同時に属している．$c \in ct(c_i)$ と記述した場合は，c は c_i の経路上のカテゴリを表し，$d(c)$ はカテゴリ c の深さを表す．カテゴリ c の祖先節点をスーパーカテゴリとよぶ．また，v_i から v_j 間の枝を $e(v_i, v_j) \in E$ と表し，$e(v_i, v_j)$ は，重み $e(v_i, v_j).w$ ($\geqq 0$)（節点間の距離や移動時間など）をもつものとする．本稿では，無向グラフとして説明するが，有向グラフへの拡張も容易である．

3.1 問 題 定 義

本稿で使用する単語と SkySR 問合せについて定義する．表 1 は本稿で使用する記号を示す．

定義 1 (経路) 経路 $R = \langle v_1, \ldots, v_r \rangle$ は，v_1 から v_r までのグラフにおける経路を表し，v_i は *PoI* である．$R[i]$ は R における i 番目の *PoI*（つまり, v_i）を表す．さらに，$R.c$ は R のカテゴリを表す（つまり，$\langle v_1.c, \ldots, v_r.c \rangle$）．

$R = \langle v_1, \ldots, v_r \rangle$ と v が与えられたとき，$R \oplus v = \langle v_1, \ldots, v_r, v \rangle$ とする．

定義 2 (カテゴリシーケンス) カテゴリシーケンス $S = \langle c_1, c_2, \ldots, c_s \rangle$ は，順序付きのカテゴリ集合を表し，s はそ

表 1: 記号の要約

記号	意味
c	カテゴリ
C	カテゴリ木の個数
$ct(c)$	カテゴリ木における c の経路
$d(c)$	カテゴリ木における c の深さ
R	経路 $\langle v_1, \ldots, v_r \rangle$
r	R の PoI の数
$R[i]$	R の i 番目の PoI
$R.c$	R のカテゴリシーケンス
S	カテゴリシーケンス $\langle c_1, \ldots, c_s \rangle$
s	S のサイズ
$S[i]$	S の i 番目のカテゴリ
S_{sup}	S のスーパーカテゴリシーケンス
$\mathcal{SUP}(S)$	S のスーパーカテゴリシーケンスの集合
$ld(R)$	R の経路長
$sd(R)$	R の非類似度スコア
SR	シーケンスド経路
$\mathcal{SKY}$	スカイラインシーケンスド経路の集合
S_q	ユーザに指定されたカテゴリシーケンス
q	ユーザに指定された始点

のサイズである．$S[i]$ は S における i 番目のカテゴリ（つまり，c_i）を表す．$S_{sup} = \langle S_{sup}[1], \ldots, S_{sup}[s] \rangle$（$S_{sup}[i]$ は $S[i]$ もしくは $S[i]$ のスーパーカテゴリ） $(1 \leqq i \leqq s)$ を S におけるスーパーカテゴリシーケンスとよぶ．$\mathcal{SUP}(S)$ は S におけるスーパーカテゴリシーケンスの集合を示す．

例えば，S が ⟨Asian restaurant, a&e, gift shop⟩ の場合，$\mathcal{SUP}(S)$ は ⟨Asian restaurant, a&e, gift shop⟩，⟨food, a&e, gift shop⟩，⟨Asian restaurant, a&e, s&s⟩，および ⟨food, a&e, s&s⟩ を含む．

定義 3 (カテゴリ類似度) *2* つのカテゴリ c_a と c_b が与えられたとき，カテゴリ間の類似度 $sim(c_a, c_b) \in [0, 1]$ は任意の関数で計算される．以下に *2* つカテゴリの関係を定義する．

- c_a と c_b が異なる木に属するカテゴリである場合，c_a は c_b と無関係である．$sim(c_a, c_b) = 0$.
- c_a と c_b が同じ木に属するカテゴリでる場合，c_a は c_b に**適合**する．$0 < sim(c_a, c_b) \leqq 1$.
- c_a と c_b が同じカテゴリの場合，c_a と c_b は**完全適合**する．$sim(c_a, c_b) = 1$.

定義 4 (シーケンス類似度) *2* つのシーケンス $S_1 = \langle S_1[1], \ldots, S_1[s_1] \rangle$ と $S_2 = \langle S_2[1], \ldots, S_2[s_2] \rangle$ が与えられたとき，シーケンス類似度 $ssim(S_1, S_2)$ は集約関数 f で求められる．

$$ssim(S_1, S_2) = f(h_1, h_2, \ldots, h_{\min(s_1, s_2)}). \quad (1)$$

h_i $(1 \leqq i \leqq \min(s_1, s_2))$ は，$sim(S_1[i], S_2[i])$ を表す．集約関数 f は単調な関数で，全ての h_i が *1* の場合，$ssim(S_1, S_2) = 1$ となる．また，$f(h_1, h_2, \ldots, h_{i-1}) \geqq f(h_1, h_2, \ldots, h_{i-1}, h_i)$ と想定する．

シーケンス非類似度 $dssim(S_1, S_2) = 1 - ssim(S_1, S_2)$ と定義する．

次に，シーケンスド経路を上記の定義を用いて定義する．

定義 5 (シーケンスド経路) カテゴリシーケンス S が与えられたとき，*2* つの条件 *(1)* $r = s$ と *(2)* $S[i]$ は $v_i.c$ に適合 $(1 \leqq i \leqq s)$ を満たす場合，$R = \langle v_1, \ldots, v_r \rangle$ はシーケンスド経路 SR である．

定義 6 (経路のスコア) カテゴリシーケンス S と始点 q が与えられたとき，経路 R のスコアとしてネットワーク距離を表す経路長 $l(R)$ と意味的な距離を表す非類似度スコア $ds(R)$ を定義する．$l(R)$ は以下の式で計算される．

$$l(R) = D(q, R[1]) + \Sigma_{i=1}^{r-1} D(R[i], R[i+1]). \quad (2)$$

$D(v_i, v_j)$ は v_i から v_j への最短ネットワーク距離を表す．一方，$ds(R)$ は以下の式で計算される．

$$ds(R) = dssim(S, R.c). \quad (3)$$

どちらのスコアも小さい値の方がよりよい経路となる．

SkySR 問合せはユーザの要求に適合する全ての好ましい経路を求める．つまり，SkySR 問合せの結果は，他の経路に支配されていない経路集合となる．支配と SkySR 問合せについて定義する．

定義 7 (支配) 始点 q とカテゴリシーケンス S が与えられたとき，全てのシーケンスド経路を $\mathcal{R}$ とする．*2* つのシーケンスド経路 $SR_i, SR_j \in \mathcal{R}$ が与えられたとき，$ds(SR_i) < ds(SR_j)$ かつ $l(SR_i) \leqq l(SR_j)$，もしくは $l(SR_i) < l(SR_j)$ かつ $ds(SR_i) \leqq ds(SR_j)$ の場合，SR_i は SR_j に支配されている．

定義 8 (意味的な階層を用いた SkySR 問合せ) 始点 q とカテゴリシーケンス S_q が与えられたとき，スカイラインシーケンスド経路は他のシーケンスド経路に支配されていない経路である．*SkySR* 問合せは，全てのスカイラインシーケンスド経路を含む集合 $\mathcal{SKY}$ を問合せ結果とする．

ここで，スコアが全く同じ経路が出力される可能性があるが，その場合そのうちの一つの経路を問合せ結果とする．

補題 1 *SkySR* 問合せは *NP-hard* である．

証明*:* 二基準最短経路問題は NP-hard として知られている [3]. 二基準最短経路問題では，スカイライン経路の数がホップ数毎に指数的に増加する可能性がある．一方，SkySR 問合せでは，経路のサイズ r が増加する度にスカイラインシーケンスド経路数が指数的に増加する可能性がある．そのため，SkySR 問合せは NP-hard である．□

3.2 単純なアプローチ

SkySR 問合せを最も簡単に実行する方法は，カテゴリシーケンス S_q の全てのスーパーカテゴリシーケンスに対して，OSR 検索を実行する方法である．スーパーカテゴリシーケンス数は，$\prod_{i=1}^{s} d(S_q[i])$ となり，全ての OSR 検索を実行するまで大きな時間がかかる．

4. 提案アプローチ

本章では，まず提案したアプローチについて説明する [7]．その後，コスト解析と最適化について議論する．提案アプローチでは，まず枝刈りのためにシーケンスド経路をみつけ，その後 BSSR 法を実行する．

4.1 枝刈り条件と初期検索

SkySR 問合せの検索範囲は非常に広い．そのため，枝刈りを効率的に行なうことが重要である．以下の 2 つの性質が定義より明らかである．

性質 1 非類似度スコアが *0* のシーケンスド経路より経路長が大きいシーケンスド経路はスカイラインシーケンスド経路にならない．

性質 2 シーケンスド経路 $SR_i \in \mathcal{SKY}$ と経路 R_j において，もし $ds(SR_i) \geqq ds(R_j)$ かつ $l(SR_i) \geqq l(R_j)$ の場合，R_j は不必要である．

まず，性質 1 は，非類似度スコアが 0 の経路の経路長が上限値となることを表す．そのため，探索中の経路が上限値より長くなる場合，すぐに枝刈りすることができる．性質 2 はさらに枝刈りできることを示唆している．経路長と非類似度スコアが既に発見しているシーケンスド経路より，どちらも大きくなる場合，スカイラインシーケンスド経路にはならない．一方で，性質 2 は非類似度スコアが探索中の経路より小さいシーケンスド経路を発見できていない場合，枝刈りができないことも示している．そのため，初期検索では，少なくとも 1 つの非類似度スコアが 0 のシーケンスド経路と，加えて他のシーケンスド経路を探すことが重要である．

効率的な探索のために，**最近傍ベース法**を提案した．この手法では，ダイクストラ法により，最も近い完全適合する節点を繰り返し探索する．$S[s]$ と完全適合した節点を探索したとき，非類似度スコアが 0 のシーケンスド経路を発見できる．この際，完全適合する節点を探索中に適合する（完全ではない）節点を発見することがある．これにより，コストの追加なしに複数のシーケンスド経路を発見することができる．

アルゴリズム 1 に最近傍ベース法の擬似コードを示す．まず，変数を初期化し（2 行目），優先度付きキュー PQ に始点 q を挿入する（3 行目）．優先度付きキューはキュー内の初期節点（最初の初期節点は q）から最も近い節点を取り出す．その後，$S_q[i]$ に完全適合する節点を繰り返し探索する（4–20 行目）．$S_q[i]$ に完全適合する節点を見つけた場合，その節点を R に挿入し，優先度付きキューをその節点で初期化する（21–22 行目）．$S_q[s]$ に適合（完全でなくてもよい）する節点を発見した場合，そのシーケンスド経路を $\mathcal{SKY}$ に挿入する（12–14 行目）．この方法では，最大で $d(S_q[s])$ 個のシーケンスド経路を一度に発見することができる．

Algorithm 1: シーケンスド経路の初期探索（最近傍ベース探索手法）

```
1  procedure NNmethod(q, S_q)
2  SKY ← φ, R ← φ;
3  priority_queue PQ ← {q};
4  for i : 1 to s do
5      //execute s times Dijkstra algorithms
6      dist[u] = inf for all vertices u ∈ V;
7      dist[PQ.top] = 0;
8      while PQ is not empty do
9          v ← PQ.dequeue;
10         if v is already visited then
11             continue;
12         if i = s and v matches S_q[i] then
13             SR ← R ⊕ v;
14             SKY.update(SR);
15         if v perfectly matches S_q[i] then
16             break;
17         for each u for e(v,u) ∈ E do
18             if dist[v] + e(v,u).w < dist[u] then
19                 dist[u] = dist[v] + e(v,u).w;
20                 PQ.enqueue(u);
21     R ← R ⊕ v;
22     PQ ← {v};
23 return SKY;
24 end procedure
```

4.2 Bulk SkySR アルゴリズム

単純なアプローチは，複数回の OSR 検索が必要となるため効率的ではない．提案した **Bulk SkySR algorithm** (BSSR 法) では，一度の問合せで全てのスカイラインシーケンスド経路を発見する．

アルゴリズム 2 に BSSR 法の擬似コードを示す．$\mathcal{SKY}$ を最近傍ベース法で初期化し（2 行目），優先度付きキュー PQ_{bssr} を初期化する（3 行目）．そこから，口述する枝刈りダイクストラ法により，経路を徐々に広げていき，優先度付きキューが空になったら終了する（4–7 行目）．

効率的に SkySR 問合せを実行するためには，優先度付きキュー内の経路数を少なくすることが重要である．そこで，SkySR 問合せのためにダイクストラ法を拡張した**枝刈りダイクストラ法**を提案した．アルゴリズム 3 に節点 v_d からカテゴリ c_d を探索する枝刈りダイクストラ法の擬似コードを示す．$\mathtt{ub}(x)$ $(0 \leqq x \leqq 1)$ は，$\mathcal{SKY}$ にあるシーケンスド経路のうち非類似度スコアが x 以下のシーケンスド経路の経路長の最小値を返す．枝刈りダイクストラ法のための優先度付きキュー PQ_{dij}

Algorithm 2: BSSR アルゴリズム

```
1  procedure BSSR(q, S_q)
2  SKY ← NNmethod(q, S_q);
3  priority_queue PQ_bssr ← φ;
4  PrunedDijkstra(φ, S[1], q, PQ_bssr, SKY);
5  while PQ_bssr is not empty do
6      R ← PQ_bssr.dequeue();
7      PrunedDijkstra(R, S[r+1], R[r], PQ_bssr, SKY);
8  return SKY;
9  end procedure
```

は v_d で初期化される（4 行目）．PQ_{dij} は v_d に最も近い節点を取り出す（6 行目）．経路 R_t は，R_d に取り出した節点 v を追加した経路で（7 行目），もし R_t の経路長が $\mathtt{ub}(ds(R_d))$ より大きい場合，枝刈りダイクストラ法は終了する（8–9 行目）．v が c_d に適合する場合，R_t の経路長を確認する（12–13 行目）．R_t の経路長が $\mathtt{ub}(ds(R_t))$ より長い場合，性質 2 に基づいてその経路は無視される．R_t がシーケンスド経路の場合，$\mathcal{SKY}$ は更新される（14–15 行目）．シーケンスド経路ではない場合，PQ_{bssr} に挿入される（16–17 行目）．v の隣接節点は v が c_d と完全適合しない場合，PQ_{dij} に挿入される（18–22 行目）．

Algorithm 3: 節点 v_d からカテゴリ c_d に適合する経路探索（枝刈りダイクストラ法）

```
1  procedure PrunedDijkstra(R_d, c_d, v_d, PQ_bssr, SKY)
2  dist[u] = inf for all vertices u ∈ V;
3  dist[v_d] = 0;
4  priority_queue PQ_dij ← {v_d};
5  while PQ_dij is not empty do
6      v ← PQ_dij.dequeue;
7      R_t ← R_d ⊕ v;
8      if l(R_t) > ub(ds(R_d)) then
9          break;
10     if v is already visited then
11         continue;
12     if v matches c_d then
13         if l(R_t) ≦ ub(ds(R_t)) then
14             if R_t is a sequenced route then
15                 SKY.update(R_t);
16             else
17                 PQ_bssr.enqueue(R_t);
18     if v does not perfectly match then
19         for each u for e(v,u) ∈ E do
20             if dist[v] + e(v,u).w < dist[u] then
21                 dist[u] = dist[v] + e(v,u).w;
22                 PQ_dij.enqueue(u);
23 end procedure
```

4.3 コスト解析

本節では，SkySR 問合せの時間コストを分析する．簡単化のために，正方格子グラフ（全ての節点が重み 1 の 4 つの枝をもつ）と想定し，全ての節点が PoI でカテゴリは均一に位置するとする．節点数および枝数をそれぞれ $|V|$ および $|E|$ と表す．節点数 N に対するダイクストラ法のコストを $d_{cost}(N)$ と表す．また，それぞれのカテゴリに属する PoI 数は同数（つまり，$\frac{|V|}{C}$）とし，あるカテゴリが PoI に適合する確率は $\frac{1}{C}$ となる．θ は，あるカテゴリに PoI が完全適合する確率を表す．実際の確率の値は，PoI 数，節点数，カテゴリ木，および指定するカテゴリの深さに依存する．

正方格子グラフでは，半径 l 内にある節点数は $2l(l+1)$ となる．少なくとも一つの完全適合する節点までの距離の最大値は，$l_{max} = \lfloor \frac{\sqrt{2\cdot\theta^{-1}+1}-1}{2} \rfloor$ となる．距離 i で完全適合する節点が見つかる確率 θ_i は以下の式で表される．

$$\theta_i = \begin{cases} 0 & (i=0), \\ \overline{\theta_{i-1}} & (i=l_{max}), \\ \overline{\theta_{i-1}} \cdot ((1-(1-\theta)^{4i})) & (otherwise). \end{cases}$$

ここで，始点は必ず完全適合する節点ではない（つまり，$\theta_0 = 0$）とし，$\overline{\theta_i} = 1-\theta_i$ を示す．そのため，l の完全適合する節点までの期待値 l_e は，$l_e = \sum_{i=1}^{l_{max}} \theta_i \cdot i$ となる．初期上限値の最大値および期待値は，それぞれ $s \cdot l_{max}$ および $s \cdot l_e$ となる．初期上限値以内の節点数は，$O(s^2 \cdot \theta^{-1})$ でスケールする．最近傍ベース法の最大時間コストは，s 回の距離 l_{max} までのダイクストラ法のコストであるため，$s \cdot d_{cost}(l_{max}^2)$ となる．

BSSR 法のコストは，(1) 枝刈りダイクストラ法の回数と (2) 枝刈りダイクストラ法のコストに依存する．(1) と (2) ともに，上限値が重要な要素である．上限値が十分に小さい場合，枝刈りダイクストラ法の範囲が小さくなり，優先度付きキュー内の経路数も少なくなる．BSSR 法の探索範囲は，初期検索により，初期上限値より距離が短い節点のみとなる．まず，上限値を用いない場合，実行される枝刈りダイクストラ法の回数は，$(\frac{|V_{in}|}{C})^{s-1}$（$|V_{in}|$ は初期上限値より距離が短い節点の数）となる．上限値を用いる場合，探索中に経路が上限値より大きくなった経路は枝刈りされる．簡略化のため，全ての経路の上限値を L とすると，以下の式で枝刈りダイクストラ法の回数を近似できる．

$$1 + \frac{4}{C} \sum_{i=1}^{s-1} \sum_{\substack{l_1 \in (1,\cdots,L) \\ l_2 \in (1,\cdots,L-l_1) \\ \cdots \\ l_i \in (1,\cdots,L-l_{(1,i-1)})}} \prod_{k=1}^{i} l_k.$$

この式では，l_j は経路における v_{j-1}（v_0 は q）から v_j 間の距離を表しており，$l_{(1,i)}$ は $\sum_{j=1}^{i} l_j$ を表す．i は経路のサイズに対応しており，全ての経路のサイズにおいて，枝刈りダイクストラ法の始点の数を求めている．経路長が上限値 L を超える経路は枝刈りされるため，i が大きくなると，各 PoI 間の距離も大きなものを選択できなくなるので値は小さくなっていく．次に，枝刈りダイクストラ法のコストを求める．経路 R から次の PoI を探す探索範囲は，上限値から R の経路長を引いた値になるため，$L - l(R)$ となる．そのため，経路 R から次の PoI を探索する場合の枝刈りダイクストラ法のコストは $d_{cost}((L-l(R))^2)$ となる．最終的に BSSR 法のコストは，以下の式で表すことができる．

$$BSSR_{cost} = d_{cost}(L^2) + \frac{4}{C}\sum_{i=1}^{s-1} \sum_{\substack{l_1\in(1,\cdots,L)\\ l_2\in(1,\cdots,L-l_1)\\ \cdots\\ l_i\in(1,\cdots,L-l_{(1,i-1)})}} d_{cost}((L-l_{(1,i)})^2)\prod_{k=1}^{i} l_k.$$

実際の L は，経路 R の非類似度スコア $ds(R)$ に依存するため，$\mathtt{ub}(ds(R))$ となる．このコスト式より，上限値が小さいほどコストが下がることがわかる．

上限値を効率的に小さくしていくには，$\mathcal{SKY}$ を効率的に更新する必要がある．効率的な更新には，(1) シーケンスド経路を発見するために必要な枝刈りダイクストラ法の回数，および (2) シーケンスド経路のスコアが重要である．そのためには，どの経路から探索していくかが問題とある．既存手法のように，経路長のみを考慮して経路の探索順を決定する場合，シーケンスド経路を発見するためには多くの PoI を探索しなければならない．なぜなら，経路長が短い経路は，経路内の PoI 数が少ない可能性が高いためである．さらに発見したシーケンスド経路の非類似度スコアが大きい可能性が高く，非類似度スコアが大きい経路は枝刈りの際にあまり有効ではないことが多い．また，別の問題として，枝刈りダイクストラ法の始点となる PoI 数は $(s-1)\cdot\frac{|V_{in}|}{C}$ であるにも関わらず，枝刈りダイクストラ法の実行回数が $s\cdot\frac{|V_{in}|}{C}$ より多いことである．PoI 数と s に比例して増加するべきで，この点においてもコストを削減できる．

4.4 最 適 化

最適化方法として，まず優先度付きキュー PQ_{bssr} からどの経路を優先的に取り出すかを議論する．さらに，枝刈りダイクストラ法の結果を再利用するキャッシュを提案する．

4.4.1 優先度付きキュー内の経路の順序

従来のダイクストラ法では，始点から最も近い節点をキューから優先的に取り出す．同じ方法を BSSR 法に用いた場合，上限値は効率的に更新されない．経路の優先度として，以下の経路の 3 つの要素を考える．

- l: 経路長（小さいほうがよい）．
- ds: 非類似度スコア（小さいほうがよい）．
- r: PoI 数（大きいほうがよい）．

要素の優先度の順番として，(l,ds,r), (l,r,ds), (ds,l,r), (ds,r,l), (r,l,ds), と (r,ds,l) の 6 つが考えられる．これらの要素の組合せを分析する．

- l を最重要とした場合，BSSR 法は優先的に始点 q に近い範囲を探索する．l は基本的に全ての経路で異なる値をもつので，他の要素は無関係となる．
- r を最重要とした場合，BSSR 法は深さ優先探索となる．同じ値の r をもつ経路は多数存在するので，2 番目の優先度影響する．
- ds を最重要とした場合，BSSR 法は完全適合する節点を優先的に探索する．同じ値の ds をもつ経路は多数存在するので，2 番目の優先度影響する．

最適な優先度として，3 つの優先度 (ds,l), (ds,r,l), と (r,ds,l) が考えられる．(ds,l) もしくは (ds,r,l) を選択した場合，BSSR 法は非類似度スコアが小さい順にシーケンスド経路を発見する．つまり，探索範囲を徐々に確実に小さくすることができる．(ds,r,l) は，(ds,l) より早くシーケンスド経路を発見することができるが，(ds,l) の方が経路長が短いシーケンスド経路を早く見つける可能性がある．一方で，(r,ds,l) を選択した場合，BSSR 法は前者 2 つより早くシーケンスド経路を発見することができるが，経路のスコアがどのような順番で見つかるかは予測できない．前者 2 つの方が探索範囲をより小さくできるが，後者はキュー内の経路数をより小さくすることができる．これらの効率性は環境に依存する．

表 2: データセット．*fs* と *syn* はそれぞれ Foursqure からの抽出した実データと人工データを表す．

データセット	位置	節点数	枝数	PoI 数	C	カテゴリ木
Tokyo	東京	576,314	499,397	174,421	10	*fs*
Cal	カリフォルニア	108,413	109,328	87,365	62	*syn*
SynCal	カリフォルニア	105,240	105,885	84,192	20	*syn*
OL	オルデンブルグ	30,525	31,455	24,420	20	*syn*

4.4.2 一時的なキャッシュ

BSSR 法は同じ節点から枝刈りダイクストラ法を実行する可能性がある．そこで，枝刈りダイクストラ法の結果（v_d から適合した PoI とその PoI までの距離）を一時的にキャッシュし，再利用する．しかし，一時的なキャッシュにより，枝刈りダイクストラ法の実行回数が最大で $(s-1)\frac{|V_{in}|}{C}$ になるわけではない．枝刈りダイクストラ法の結果を再利用できない場合が存在する．

補題 2 同じ節点からの枝刈りダイクストラ法の探索範囲が $\mathtt{ub}(ds(R_d))-l(R_d)$ より小さい場合，枝刈りダイクストラを再度実行する必要がある．

証明: 探索範囲は経路のスコアに依存し，最大探索範囲は $\mathtt{ub}(ds(R_d))-l(R_d)$ となる．これから探索する経路の探索範囲が $\mathtt{ub}(ds(R_d))-l(R_d)$ より大きい場合，正しい結果を得ることができないのは自明である．□

5. 実　　験

全てのアルゴリズムは C++で実装され，CPU が Intel(R) Xeon(R) CPU E5620 @ 2.40GHz で，メモリが 32.0GB の計算機を用いた．

5.1 セッティング

アルゴリズム． 単純手法（NAIVE），上限値を用いた単純手法（UOSR），および BSSR 法を用いて評価する．NAIVE と UOSR では，ダイクストラベース法と PNE 法を OSR 検索アルゴリズムとして用いる．BSSR 法では，4 つの優先度 (ds,r,l), (r,ds,l), $(ds,\ l)$, および (l)，と一時的なキャッシュを用いる場合と用いない場合の 8 つのパターンを評価する．例えば，BSSR(ds,r,l)cache は，優先度が (ds,r,l) で一時的なキャッシュを用いた場合を示す．評価基準は，CPU 時間，最大メモリ使用量（RSS），スカイラインシーケンスド経路数とする．

データセット． 実験では，複数種の道路図を用いる．グラフは，距離を枝の重みとして用い，無向グラフとする．全てのグラフは隣接リストにより実装する．表 2 は，データセットの位置，

表 3: 実験結果の概要．カテゴリシーケンスのサイズは 4 とする．

	Tokyo		Cal		SynCal		OL	
	CPU time	RSS	CPU time	RSS	CPU time	RSS	CPU time	RSS
BSSR(ds,r,l)cache	3.36 sec	339.5 MB	2.50 sec	59.2 MB	**0.37** sec	50.6 MB	**0.27** sec	16.0 MB
BSSR(r,ds,l)cache	**3.27** sec	301.6 MB	3.29 sec	51.7 MB	0.46 sec	50.3 MB	0.38 sec	15.5 MB
BSSR(ds,l)cache	3.90 sec	339.5 MB	**2.31** sec	68.2 MB	0.40 sec	50.5 MB	0.30 sec	16.7 MB
BSSR(l)cache	4.12 sec	337.4 MB	3.03 sec	81.3 MB	0.62 sec	51.4 MB	0.64 sec	18.0 MB
BSSR(ds,r,l)	5.38 sec	316.7 MB	3.96 sec	51.7 MB	0.39 sec	49.9 MB	0.32 sec	15.8 MB
BSSR(r,ds,l)	5.69 sec	299.1 MB	5.20 sec	51.9 MB	0.49 sec	49.6 MB	0.44 sec	15.5 MB
BSSR(ds,l)	5.93 sec	317.2 MB	3.63 sec	58.8 MB	0.38 sec	49.5 MB	0.33 sec	16.4 MB
BSSR(l)	6.74 sec	318.3 MB	5.20 sec	68.3 MB	0.61 sec	49.5 MB	0.77 sec	17.6 MB
UOSR(PNE)	338.9 sec	306.3 MB	84.7 sec	52.8 MB	6.29 sec	49.2 MB	4.73 sec	15.5 MB
UOSR(Dij)	17.6 sec	333.5 MB	6.47 sec	60.2 MB	5.41 sec	50.2 MB	1.82 sec	15.6 MB
NAIVE(PNE)	835.3 sec	307.6 MB	329.9 sec	54.4 MB	9.45 sec	48.8 MB	15.4 sec	16.1 MB
NAIVE(Dij)	5847 sec	18.4 GB	178.9 sec	543.4 MB	49.1 sec	150.4 MB	72.3 sec	93.2 MB
		# of SKY		# of SKY		# of SKY		# of SKY
All algorithms		3.69		6.04		7.24		6.32

節点数，枝数，PoI 数，カテゴリ木の個数，カテゴリ木の種類を示す．Tokyo データセットでは，道路ネットワークは open street map(注2) から，PoI は Foursquare から取得した．それぞれの PoI を最も近い枝に埋め込み，カテゴリ木は実データを用いる．カリフォルニアとオルデンブルグの道路図はウェブから取得した (注3)．Cal データセットでは，PoI の位置とカテゴリは実データだが，カテゴリ木はない．SynCal データセットと OL データセットは，道路図上に PoI をもっていないため，枝の上に一様に PoI を生成する（つまり，全ての枝上で PoI 数はほぼ同数である）．Cal データセット，SynCal データセット，および OL データセットにおいて，PoI は人工的なカテゴリ木を保持する．人工的なカテゴリ木は，高さが 3 でそれぞれの中間節点が 3 つの子節点をもつ．それぞれの PoI はランダムでカテゴリ木の葉節点をカテゴリとして割り当てられる．

データセットに対して，サイズが s のカテゴリシーケンスをもつクエリを 100 個生成する．始点は，PoI となる節点以外の節点からランダムに選ばれる．また，カテゴリシーケンス内のカテゴリは，ランダムに葉節点から選ばれるが，重複したカテゴリ木からは選ばない．カテゴリ類似度として *Wu and Palmer similarity measure* [12]，シーケンス類似度として総乗を用いる．具体的には，以下の式で計算される．

$$sim(c_a, c_b) = \max_{c_x \in ct(c_b)} \frac{2 \cdot d(common)}{d(c_a) + d(c_x)}. \quad (4)$$

$$ssim(S_1, S_2) = \prod_{i=1}^{\min(s_1,s_2)} sim(S_1[i], S_2[i]). \quad (5)$$

$common$ は c_a と c_x の最も深い共通の祖先節点を表す．

5.2 結果の概要

結果の概要を表 3 に示す．この結果より，一時的なキャッシュを用いた BSSR 法が全てのデータセットで最も小さい CPU 時間を達成していることがわかる．キューの優先度を比較すると，データセットにより最もよいものは異なるが，(ds,r,l) が全てのデータセットで比較的小さい CPU 時間を達成している．一時的なキャッシュは CPU 時間を削減するが，RSS が増加する．しかし，この結果では，RSS はデータセットのサイズに大きく依存しているため，増加率は大きくない．最適化を行なっていない BSSR 法（つまり，BSSR(l)）は，BSSR(ds,r,l)cache に比べて CPU 時間が 2 倍程度大きくなっている．

ダイクストラベース法と PNE 法を用いた UOSR を比較すると，ダイクストラベース法を用いた方が CPU 時間が小さい．上限値を用いる場合は，ダイクストラベース法の OSR 検索の方が優れていることを示しているが，上限値を用いない場合は，データセットに依存している．ダイクストラベース法を用いた単純手法の RSS が非常に大きい．これは，優先度付きキュー内の経路数が非常に大きくなっていることに起因する．この結果より，キュー内の経路数が RSS に大きな影響を及ぼすことを確認できる．

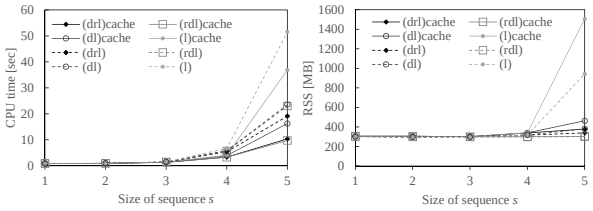
また，全ての手法で同じ数のスカイラインシーケンスド経路を問合せ結果としているため，全ての手法が厳密解を求められていることを確認できる．

5.3 カテゴリシーケンスのサイズの影響

まず Tokyo データセットにおける，BSSR 法のカテゴリシーケンスのサイズの影響を調べる．図 3 は，Tokyo データセットにおける CPU 時間と RSS を示している．この図より，s が増加すると，CPU 時間が大きく増加していることがわかる．一時的なキャッシュは実データでより効果を発揮している．これは，同じ節点から枝刈りダイクストラ法が実行されやすいためである．また，s が増加するにつれ，RSS も増加している．優先度が (l) の場合に顕著である．これは，キュー内の経路が増加するからであるが，s が大きくなると優先度 (l) ではなかなかシーケンスド経路を見つけることができないためである．

図 4 は，s を変化させたときの各データセットにおけるスカイラインシーケンスド経路の数を示している．この結果より，Tokyo データセットのスカイラインシーケンスド経路の数が最

(注2)：https://www.openstreetmap.org
(注3)：http://www.cs.utah.edu/~lifeifei/SpatialDataset.htm



(a) CPU time (b) RSS

図 3: s を変化させた場合の Tokyo データセットの結果

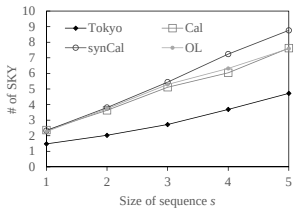


図 4: s を変化させた場合のスカイラインシーケンスド経路数

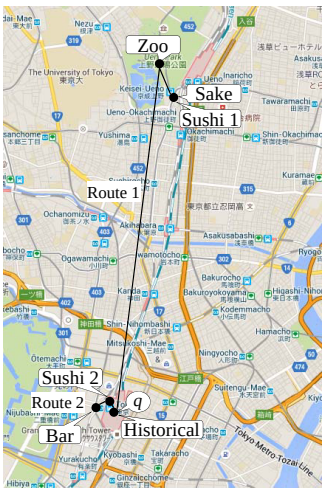


図 5: 東京における出力経路の例

も小さいのがわかる．スカイラインシーケンスド経路の数は，CPU 時間や RSS に無関係であることがわかる．さらに，PoI およびグラフどちらも実データを用いた場合，スカイラインシーケンスド経路はそれほど大きくならないことも示している．

5.4 出力経路の例

Tokyo データセットを用いて，スカイラインシーケンスド経路の検索結果の例を示す．この例では，カテゴリシーケンスは，⟨zoo, sushi restaurant, sake bar⟩ とし，始点は東京駅とした．SkySR 問合せを用いることで，6 つの経路を見つけることができる．図 5 は，2 つの代表的な経路である．Route 1 (⟨zoo, sushi 1, sake⟩) はユーザの要求するカテゴリに完全に一致した経路であるが，経路長は長い．一方，Route2 は，要求するカテゴリに完全に一致するわけではないが，経路長は非常に短い．他の 4 つの経路はどちらかの経路に類似している（つまり，zoo を含むか含まないか）．この例より，SkySR 問合せは旅行計画の決定に有効であることがわかる．

6. おわりに

本稿では，スカイラインシーケンスド経路（SkySR）問合せのためのアルゴリズム BSSR 法の最適化を行なった．最適化を実施する上で，まずコスト分析を行い，最適化の要素を検証した．最適化のために，キュー内の経路の優先度とダイクストラ法の結果を一時的にキャッシュする方法を提案した．実データを用いた実験より，これらの最適化を用いると CPU 時間が半分程度になり，メモリ使用量もそれほど大きくならないことを確認した．

謝　辞

本研究は科学研究費（15K21069）の支援によって行われた．ここに記して謝意を表す．

文　献

[1] S. Aljubayrin, Z. He, and R. Zhang. Skyline trips of multiple pois categories. In DASFAA, pages 189–206, 2015.

[2] J. Eisner and S. Funke. Sequenced route queries: Getting things done on the way back home. In ACM SIGSPATIAL, pages 502–505, 2012.

[3] P. Hansen. Bicriterion path problems. In Multiple criteria decision making theory and application, pages 109–127. 1980.

[4] H.-P. Kriegel, M. Renz, and M. Schubert. Route skyline queries: A multi-preference path planning approach. In ICDE, pages 261–272, 2010.

[5] E. Q. V. Martins. On a multicriteria shortest path problem.European Journal of Operational Research, 16(2):236–245,1984.

[6] Y. Ohsawa, H. Htoo, N. Sonehara, and M. Sakauchi. Sequenced route query in road network distance based on incremental euclidean restriction. In DEXA, pages 484–491,2012.

[7] 佐々木勇和, 石川佳治. 多階層のカテゴリ分類を用いたスカイライン経路検索について. 第 7 回データ工学と情報マネジメントに関するフォーラム (DEIM 2015), 2015.

[8] M. Sharifzadeh, M. Kolahdouzan, and C. Shahabi. The optimal sequenced route query. The VLDB Journal, 17(4):765–787, 2008.

[9] M. Sharifzadeh and C. Shahabi. Processing optimal sequenced route queries using voronoi diagrams. GeoInformatica, 12(4):411–433, 2008.

[10] M. Shekelyan, G. Jossé, and M. Schubert. Linear path skylines in multicriteria networks. In ICDE, pages 459–470, 2015.

[11] Y. Tian, K. C. Lee, and W.-C. Lee. Finding skyline paths in road networks. In ACM SIGSPATIAL GIS, pages 444–447, 2009.

[12] Z. Wu and M. Palmer. Verbs semantics and lexical selection. In ACL, pages 133–138, 1994.

[13] B. Yang, C. Guo, C. S. Jensen, M. Kaul, and S. Shang. Stochastic skyline route planning under time-varying uncertainty. In ICDE, pages 136–147, 2014.